

MPC5510 Reference Manual

This is the MPC5510 Reference Manual set consisting of the following files:

- MPC5510 Reference Manual Addendum, Rev 1
- MPC5510 Reference Manual, Rev 1

MPC5510 Reference Manual Addendum

This addendum document describes corrections to the *MPC5510 Microcontroller Reference Manual*, order number MPC5510RM. For convenience, the addenda items are grouped by revision. Please check our website at <http://www.freescale.com/powerarchitecture> for the latest updates.

The current version available of the *MPC5510 Microcontroller Reference Manual* is Revision 1.0.

Table of Contents

1	Addendum for Revision 1.0.	2
2	Revision History	5

1 Addendum for Revision 1.0

Table 1. MPC5510RM Rev 1.0 Addendum

Location	Description
Section 2.2 "Signal Properties Summary"	<p>Added a note just above "MPC5510 Signal Properties" table: "Please note that analog input pins (ANx) are directly connected to the eQADC and that they are not routed through the PCRx register; thus, changes in the PCR register does not affect these inputs."</p>
Section 3.3 "System Clock Architecture Block Diagram"	<p>Updated the paragraph: To optimize system power consumption, the MPC5510 supports system-level clock dividers, static clock gating using peripheral-level module disable (MDIS) bits and a system-level halt mechanism. Figure 3-2 shows the device-level clock gating mechanism for the MPC5510. Figure 3-3 shows a more detailed implementation of the MDIS and halt mechanism connections for a given peripheral. These features are detailed in subsequent sections.</p> <p>Added a new NOTE: While combining DMA with peripheral modules (for example eSCI), the user has to use LPCLKDIVx = 0 (no divide) to ensure that the DMA.DONE (same as DMA ACK) is correctly acknowledged by the peripheral. Otherwise, DMA.DONE may not be sampled correctly, leading to data loss. In the case of eSCI peripheral, BERR flag will be set and the peripheral behavior will be unexpected if LPCLKDIV4 > 0.</p>

Table 1. MPC5510RM Rev 1.0 Addendum

Location	Description
<p>Section 3.3 "System Clock Architecture Block Diagram"</p>	<p>Updated Figure 3-2 "System Clock Architecture".</p> <p>The diagram illustrates the system clock architecture. It starts with an external oscillator (XOSC) that provides input to a PLL and an IRC. These two paths converge into a 'Switcher and divider' block. From this block, three main clock paths emerge: <ul style="list-style-type: none"> Bypass clock: This path goes directly to the 'System clock' bus. System clock: This path is the primary system clock, which is distributed to various components: <ul style="list-style-type: none"> A box containing 'Cores INTC, DMA, SIU, RAM, Flash, BAM, AIPS, AXBS, MCM, eQADC'. The 'CLKOUT divider' block, which produces the 'CLKOUT' signal. The 'MCKO divider' block, which produces the 'MCKO' signal. Various module clocks for peripherals: FlexRAY, FlexCAN_A, FlexCAN_B-F, DSPI_A, DSPI_B-D, ESCI_A, ESCI_B-H, eMIOS, and MLB. Each of these module clocks is derived from the system clock through an AND gate. Oscillator clock: This path is used for low-power clocks and is distributed to LPCLKDIV0 through LPCLKDIV6. Each LPCLKDIV block then provides a 'Module clock' (via an AND gate) and a 'Protocol clock' (via a divider block labeled 'DIV/2') to its respective peripheral block. Each peripheral block also includes an 'MDIS' (Master Disable) signal and a 'CLK_SRC' (Clock Source) input. </p>

Table 1. MPC5510RM Rev 1.0 Addendum

Location	Description																		
Section 3.5.2 "Halt Clock Gating"	Added a note: SIU_HLT is not delayed by any pending interrupt for specific modules to be serviced. If any interrupt is raised after the SIU_HLT request, it may cause the interrupt to hang. To avoid this, it is advised that all interrupts are disabled before entering the SIU_HLT state and then, re-enabled once SIU_HLT is exited. The result is that any interrupt will be flagged, but not triggered within its specific interrupt service routine. Once the SIU_HLT has been enabled, any pending interrupt will be taken as normal. In the case of STOP mode exit, an external interrupt may be required. In this case, the specific exit interrupt may be enabled, but software must ensure the interrupt does not occur simultaneously with SIU_HLT being enabled.																		
Section 5.3.2 "Low-power Mode Entry"	Updated "The system clock source should be set to the 16 MHz IRC prior to ..." to "The system clock source needs be set to the 16 MHz IRC (with the default divide by 1 system clock configuration) prior to ...".																		
Section 6.3.2.2 "Reset Status Register (SIU_RSR)"	Updated point 1. The updated text is as follows: "If any reset request has negated and the device is still in the resulting reset, and then an external reset is requested, both the original reset type and external reset status bits will be set. In this case, the device started the reset sequence due to a non-external reset request but ended the reset sequence after an external reset request." Updated figure note 3 of Figure 6-3, "Reset Status Register (SIU_RSR)". The updated text is as follows: "The ERS bit is also set if the RESET pin is held low to extend the reset sequence."																		
Section 6.3.2.25 "System Clock Register (SIU_SYSCLOCK)"	Aligned table 6-27 "LPCLKDIV Module Groups", as mentioned below. <div style="text-align: center;"> <p>Table 6-27: LPCLKDIV Module Groups</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th data-bbox="548 974 862 1031">LPCLKDIVn</th> <th data-bbox="862 974 1289 1031">Modules</th> </tr> </thead> <tbody> <tr> <td data-bbox="548 1037 862 1073">LPCLKDIV0</td> <td data-bbox="862 1037 1289 1073">FlexCAN_A, DSPI_A</td> </tr> <tr> <td data-bbox="548 1079 862 1115">LPCLKDIV1</td> <td data-bbox="862 1079 1289 1115">ESCI_A, I²C_A, PIT</td> </tr> <tr> <td data-bbox="548 1121 862 1157">LPCLKDIV2</td> <td data-bbox="862 1121 1289 1157">FlexCAN_B-F</td> </tr> <tr> <td data-bbox="548 1163 862 1199">LPCLKDIV3</td> <td data-bbox="862 1163 1289 1199">DSPI_B-D</td> </tr> <tr> <td data-bbox="548 1205 862 1241">LPCLKDIV4</td> <td data-bbox="862 1205 1289 1241">ESCI_B-H</td> </tr> <tr> <td data-bbox="548 1247 862 1283">LPCLKDIV5</td> <td data-bbox="862 1247 1289 1283">eMIOS</td> </tr> <tr> <td data-bbox="548 1289 862 1325">LPCLKDIV6</td> <td data-bbox="862 1289 1289 1325">MLB</td> </tr> <tr> <td data-bbox="548 1331 862 1367">LPCLKDIV7</td> <td data-bbox="862 1331 1289 1367">Reserved</td> </tr> </tbody> </table> </div>	LPCLKDIV n	Modules	LPCLKDIV0	FlexCAN_A, DSPI_A	LPCLKDIV1	ESCI_A, I ² C_A, PIT	LPCLKDIV2	FlexCAN_B-F	LPCLKDIV3	DSPI_B-D	LPCLKDIV4	ESCI_B-H	LPCLKDIV5	eMIOS	LPCLKDIV6	MLB	LPCLKDIV7	Reserved
LPCLKDIV n	Modules																		
LPCLKDIV0	FlexCAN_A, DSPI_A																		
LPCLKDIV1	ESCI_A, I ² C_A, PIT																		
LPCLKDIV2	FlexCAN_B-F																		
LPCLKDIV3	DSPI_B-D																		
LPCLKDIV4	ESCI_B-H																		
LPCLKDIV5	eMIOS																		
LPCLKDIV6	MLB																		
LPCLKDIV7	Reserved																		
Section 24.1.1 "Block Diagram"	Added a figure note for peripheral clock in the "eSCI Block Diagram" figure: Refer to Section 3.3, " System Clock Architecture Block Diagram".																		
Section 24.3.2.3 "eSCI Data Register (ESCIx_DR)"	Added a note: eSCI transmission delay will depend on the actual Tx load into the Data Register referenced with the internal clock, if the load occurs before 45% of a bit time has passed, the Tx load will be transmitted in less than a bit time. Otherwise, the Tx will take up to 1.5 of a bit time.																		

Table 1. MPC5510RM Rev 1.0 Addendum

Location	Description
Section 31.4.3.3 "External Trigger Input Multiplexing"	<ul style="list-style-type: none"> Updated the title from "External Trigger from eTPU to eMIOS Channels" to "External Trigger Input Multiplexing". Updated the text of this section to: The four eQADC external trigger inputs can be connected to two different external pins or one of two PIT channels. The input source for each eQADC external trigger is individually specified in the IMUX Select Register 0 (SIU_ISEL0). Figure 6-50 gives an example of the multiplexing of an eQADC external trigger input. As shown in the figure, the ETRIG[0] input of the eQADC can be connected to the PC4 pin, the PG4 pin, the PIT7 channel, or the PIT8 channel. Remaining ETRIG inputs are multiplexed in the same manner. <p>The eQADC trigger numbers specified by SIU_ETISR[TSEL(0-3)] correspond to CFIFO numbers 0-3. To calculate the CFIFO number that each trigger is connected to, divide the eDMA channel number by 2.</p>

2 Revision History

Table 2 provides a revision history for this document.

Table 2. Revision History Table

Rev. Number	Substantive Changes	Date of Release
1.0	First release.	04/2012

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

<http://www.freescale.com/support>

USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.



Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © Freescale Semiconductor, Inc. 2012. All rights reserved.

MPC5510RMAD
Rev. 1
04/2012

MPC5510 Microcontroller Family Reference Manual

Devices Supported:
MPC5517G/E/S
MPC5516G/E/S
MPC5515S
MPC5514G/E

Document Number: MPC5510RM
Rev. 1
06/2008

PRELIMINARY

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

<http://www.freescale.com/support>

USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.



Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2007, 2008. All rights reserved.

MPC5510RM

Rev. 1

06/2008

Chapter 1 Overview

1.1	Introduction	1-1
1.2	Block Diagram	1-2
1.3	MPC5510 Family Comparison	1-3
1.3.1	Family Feature Set Scaling	1-5
1.4	Chip-Level Features	1-6
1.5	Low-Power Operation	1-7
1.6	Memory Map	1-7

Chapter 2 Signal Descriptions

2.1	Introduction	2-1
2.2	Signal Properties Summary	2-1
2.3	Power and Ground Supply Summary	2-12
2.4	Pinout – 144 LQFP	2-14
2.5	Pinout – 176 LQFP	2-15
2.6	Pinout – 208 BGA	2-16
2.7	Detailed External Signal Descriptions	2-16
2.7.1	Port A Pins	2-16
2.7.2	Port B Pins	2-17
2.7.3	Port C Pins	2-19
2.7.4	Port D Pins	2-21
2.7.5	Port E Pins	2-24
2.7.6	Port F Pins	2-25
2.7.7	Port G Pins	2-28
2.7.8	Port H Pins	2-30
2.7.9	Port J Pins	2-32
2.7.10	Port K Pins	2-33
2.7.11	Miscellaneous Pins	2-33
2.7.12	Power and Ground Pins	2-34

Chapter 3 System Clock Description

3.1	Introduction	3-1
3.2	Clock Sources	3-1
3.2.1	External High-Frequency Crystal (XOSC)	3-2
3.2.2	External Low-Frequency Crystal (32kXOSC)	3-3
3.2.3	Internal High-Frequency RC Oscillator (IRC)	3-3
3.2.4	Internal Low-Frequency RC Oscillator (32kRC)	3-3
3.3	System Clock Architecture Block Diagram	3-4
3.4	Clock Dividers	3-5
3.4.1	System Clock Select	3-5
3.4.2	System Clock Dividers	3-5

3.4.3	External Bus Clock (CLKOUT) Divider	3-5
3.4.4	Nexus Message Clock (MCKO) Divider	3-5
3.4.5	Peripheral Clock Dividers	3-5
3.5	Software-Controlled Power Management	3-6
3.5.1	Module Disable (MDIS) Clock Gating	3-6
3.5.2	Halt Clock Gating	3-7
3.5.3	Core WAIT Clock Gating	3-7
3.6	Alternate Module Clock Domains	3-8
3.6.1	FlexCAN Clock Domains	3-8
3.6.2	FlexRay Clock Domains	3-8
3.6.3	RTC Clock Domain	3-9
3.6.4	SWT Clock Domain	3-9

Chapter 4

Frequency Modulated Phase Locked Loop (FMPLL)

4.1	Introduction	4-1
4.1.1	Block Diagram	4-1
4.1.2	Features	4-2
4.1.3	Modes of Operation	4-2
4.2	External Signal Description	4-2
4.3	Memory Map and Registers	4-2
4.3.1	Module Memory Map	4-3
4.3.2	Register Descriptions	4-3
4.4	Functional Description	4-11
4.4.1	General	4-11
4.4.2	PLL Off Mode	4-11
4.4.3	Normal Mode	4-11
4.5	Resets	4-18
4.5.1	Clock Mode Selection	4-18
4.5.2	PLL Loss-of-Lock Reset	4-19
4.5.3	PLL Loss-of-Clock Reset	4-19
4.6	Interrupts	4-19
4.6.1	Loss-of-Lock Interrupt Request	4-19
4.6.2	Loss-of-Clock Interrupt Request	4-19

Chapter 5

Clock, Reset, and Power Control (CRP)

5.1	Introduction	5-1
5.1.1	Block Diagram	5-1
5.1.2	Features	5-3
5.1.3	Modes of Operation	5-3
5.2	Memory Map and Registers	5-4
5.2.1	Module Memory Map	5-4
5.2.2	Register Descriptions	5-4

5.3	Functional Description	5-16
5.3.1	Low-Power Modes	5-16
5.3.2	Low-Power Mode Entry	5-17
5.3.3	Low-Power Operation	5-18
5.3.4	Low-Power Wakeup	5-24
5.4	Real-Time Counter (RTC)	5-28
5.4.1	RTC Features	5-28
5.4.2	RTC Functional Description	5-29
5.4.3	Register Description	5-31
5.5	Power Supply Monitors	5-32
5.5.1	Power-On Reset (POR)	5-32
5.5.2	Low-Voltage Monitors (LVI)	5-32
5.6	Low-Voltage Operation	5-32

Chapter 6 System Integration Unit (SIU)

6.1	Introduction	6-1
6.1.1	Block Diagram	6-1
6.1.2	Features	6-2
6.1.3	Modes of Operation	6-3
6.2	External Signal Description	6-4
6.2.1	Detailed Signal Descriptions	6-4
6.3	Memory Map and Registers	6-5
6.3.1	Module Memory Map	6-5
6.3.2	Register Descriptions	6-11
6.4	Functional Description	6-49
6.4.1	System Configuration	6-49
6.4.2	Reset Control	6-50
6.4.3	External Interrupt	6-50
6.4.4	GPIO Operation	6-51
6.4.5	Internal Multiplexing	6-51

Chapter 7 Reset

7.1	Introduction	7-1
7.2	External Signal Description.	7-1
7.2.1	Reset ($\overline{\text{RESET}}$)	7-2
7.2.2	Boot Configuration (BOOTCFG)	7-2
7.3	Functional Description	7-2
7.3.1	Z1, Z0 Cores Reset Vectors	7-2
7.3.2	Reset Sources	7-3
7.4	Reset Configuration	7-4
7.4.1	Reset Configuration Timing	7-4

Chapter 8 Interrupts

8.1	Introduction	8-1
8.2	Interrupt Vectors	8-2
8.2.1	Core Interrupts	8-2
8.2.2	External Input: Software Vector Mode	8-3
8.2.3	External Input: Hardware Vector Mode	8-3
8.2.4	Critical Input	8-4
8.3	Interrupt Sources	8-5
8.3.1	Interrupt Source Summary Table	8-5
8.4	Interrupt Operation	8-19
8.4.1	Software Vector Mode	8-19
8.4.2	Hardware Vector Mode	8-19
8.4.3	Non Maskable Interrupt (NMI)	8-19
8.4.4	Dynamic Priority Elevation	8-20

Chapter 9 Interrupt Controller (INTC)

9.1	Introduction	9-1
9.1.1	Features	9-1
9.1.2	Block Diagram	9-2
9.1.3	Modes of Operation	9-4
9.2	Signal Description	9-5
9.3	Memory Map and Registers	9-5
9.3.1	Module Memory Map	9-5
9.3.2	Register Descriptions	9-6
9.4	Functional Description	9-15
9.4.1	Interrupt Request Sources	9-15
9.4.2	Priority Management	9-16
9.4.3	Handshaking with Processor	9-17
9.5	Initialization/Application Information	9-20
9.5.1	Initialization Flow	9-20
9.5.2	Interrupt Exception Handler	9-20
9.5.3	ISR, RTOS, and Task Hierarchy	9-22
9.5.4	Order of Execution	9-22
9.5.5	Priority Ceiling Protocol	9-23
9.5.6	Selecting Priorities According to Request Rates and Deadlines	9-24
9.5.7	Software Settable Interrupt Requests	9-25
9.5.8	Lowering Priority Within an ISR	9-26
9.5.9	Negating an Interrupt Request Outside of its ISR	9-26
9.5.10	Examining LIFO contents	9-27

Chapter 10 e200z1 Core (Z1)

10.1	Introduction	10-1
10.1.1	Features	10-1
10.2	Microarchitecture Summary	10-2
10.2.1	Instruction Unit Features	10-3
10.2.2	Integer Unit Features	10-4
10.2.3	Load/Store Unit Features	10-4
10.2.4	e200z1 System Bus Features	10-4
10.2.5	MMU Features	10-4
10.3	Core Registers and Programmer's Model	10-5
10.3.1	Power Architecture Book E Registers	10-8
10.3.2	e200-Specific Special Purpose Registers	10-11
10.3.3	e200z1 Core Complex Features Not Supported on the MPC5510	10-13
10.4	e200z1 Memory Management Unit	10-13
10.4.1	Effective to Real Address Translation	10-13
10.4.2	Translation Lookaside Buffer	10-17
10.4.3	MMU Assist Registers (MAS)	10-18
10.5	Interrupt Types	10-23
10.6	Bus Interface Unit (BIU)	10-25

Chapter 11 e200z0 Core (Z0)

11.1	Introduction	11-1
11.1.1	Features	11-1
11.2	Microarchitecture Summary	11-2
11.2.1	Instruction Unit Features	11-3
11.2.2	Integer Unit Features	11-3
11.2.3	Load/Store Unit Features	11-4
11.2.4	e200z0 System Bus Features	11-4
11.3	Core Registers and Programmer's Model	11-4
11.3.1	Power Architecture Book E Registers	11-7
11.3.2	e200-Specific Special Purpose Registers	11-9
11.3.3	e200z0 Core Complex Features Not Supported on the MPC5510	11-11
11.4	Interrupt Types	11-11
11.5	Bus Interface Unit (BIU)	11-12

Chapter 12 Enhanced Direct Memory Access (eDMA)

12.1	Introduction	12-1
12.1.1	Block Diagram	12-1
12.1.2	Features	12-2
12.1.3	Modes of Operation	12-3
12.2	External Signal Description	12-3

12.3	Memory Map and Registers	12-3
12.3.1	Module Memory Map	12-3
12.3.2	Register Descriptions	12-7
12.4	Functional Description	12-24
12.4.1	eDMA Basic Data Flow	12-26
12.5	Initialization / Application Information	12-29
12.5.1	eDMA Initialization	12-29
12.5.2	DMA Programming Errors	12-31
12.5.3	DMA Request Assignments	12-32
12.5.4	DMA Arbitration Mode Considerations	12-32
12.5.5	DMA Transfer	12-33
12.5.6	TCD Status	12-36
12.5.7	Channel Linking	12-37
12.5.8	Dynamic Programming	12-38

Chapter 13 DMA Channel Mux (DMA_MUX)

13.1	Introduction	13-1
13.1.1	Block Diagram	13-1
13.1.2	Features	13-2
13.1.3	Modes of Operation	13-2
13.2	External Signal Description	13-2
13.3	Memory Map and Registers	13-2
13.3.1	Module Memory Map	13-2
13.3.2	Register Descriptions	13-3
13.4	Functional Description	13-7
13.4.1	DMA Channels 0–7	13-7
13.4.2	DMA Channels 8–15	13-9
13.4.3	Always Enabled DMA Sources	13-10
13.5	Initialization/Application Information	13-11
13.5.1	Reset	13-11
13.5.2	Enabling and Configuring Sources	13-11
13.6	Interrupts	13-14

Chapter 14 Peripheral Bridge (AIPS-lite)

14.1	Introduction	14-1
14.1.1	Terminology	14-1
14.1.2	Block Diagram	14-1
14.1.3	Features	14-2
14.1.4	Modes of Operation	14-2
14.2	External Signal Description	14-2
14.3	Memory Map and Registers	14-2
14.4	Functional Description	14-2

14.4.1 Read Cycles	14-3
14.4.2 Write Cycles	14-3

Chapter 15 Crossbar Switch (XBAR)

15.1 Introduction	15-1
15.1.1 Block Diagram	15-1
15.1.2 Features	15-2
15.1.3 Modes of Operation	15-3
15.2 Signal Description	15-3
15.3 Memory Map and Registers	15-3
15.4 Functional Description	15-3
15.4.1 Master Ports	15-4
15.4.2 Slave Ports	15-4
15.4.3 Arbitration	15-4
15.4.4 Slave Port State Machine	15-6
15.5 DMA Requests	15-8
15.6 Interrupt Requests	15-8

Chapter 16 Miscellaneous Control Module (MCM)

16.1 Introduction	16-1
16.1.1 Features	16-1
16.2 Memory Map and Registers	16-2
16.2.1 Module Memory Map	16-2
16.2.2 Register Descriptions	16-4
16.3 Functional Description	16-17
16.3.1 High-Priority Enables	16-17

Chapter 17 Memory Protection Unit (MPU)

17.1 Introduction	17-1
17.1.1 Block Diagram	17-1
17.1.2 Features	17-2
17.1.3 Modes of Operation	17-3
17.2 Signal Description	17-3
17.3 Memory Map and Registers	17-3
17.3.1 Module Memory Map	17-3
17.3.2 Register Descriptions	17-5
17.4 Functional Description	17-14
17.4.1 Access Evaluation Macro	17-14
17.4.2 Putting It All Together and AHB Error Terminations	17-16
17.5 Initialization Information	17-16
17.6 Application Information	17-17

Chapter 18 Semaphores

18.1	Introduction	18-1
18.1.1	Block Diagram	18-1
18.1.2	Features	18-2
18.1.3	Modes of Operation	18-3
18.2	Signal Description	18-3
18.3	Memory Map and Registers	18-3
18.3.1	Module Memory Map	18-3
18.3.2	Register Descriptions	18-4
18.4	Functional Description	18-10
18.4.1	Semaphore Usage	18-11
18.5	Initialization Information	18-12
18.6	Application Information	18-12
18.7	DMA Requests	18-13
18.8	Interrupt Requests	18-13

Chapter 19 IEEE 1149.1 Test Access Port Controller (JTAGC)

19.1	Introduction	19-1
19.1.1	Block Diagram	19-1
19.1.2	Features	19-2
19.1.3	Modes of Operation	19-2
19.2	External Signal Description	19-4
19.3	Memory Map and Registers	19-4
19.3.1	Instruction Register	19-4
19.3.2	Bypass Register	19-4
19.3.3	Device Identification Register	19-5
19.3.4	Boundary Scan Register	19-5
19.4	Functional Description	19-5
19.4.1	JTAGC Reset Configuration	19-5
19.4.2	IEEE 1149.1-2001 (JTAG) Test Access Port	19-6
19.4.3	TAP Controller State Machine	19-6
19.4.4	JTAGC Instructions	19-8
19.4.5	Boundary Scan	19-10
19.5	e200z0 and e200z1 OnCE Controllers	19-11
19.5.1	e200z0 OnCE Controller Block Diagram	19-11
19.5.2	e200z0 OnCE Controller Functional Description	19-11
19.5.3	e200z0 OnCE Controller Register Descriptions	19-12
19.6	Initialization/Application Information	19-14

Chapter 20 Nexus Development Interface (NDI)

20.1	Introduction	20-1
------	--------------	------

20.2	Block Diagram	20-2
20.2.1	Features	20-3
20.2.2	Modes of Operation	20-4
20.3	External Signal Description	20-6
20.3.1	Nexus Signal Reset States	20-6
20.4	Memory Map and Registers	20-6
20.4.1	Nexus Debug Interface Registers	20-6
20.4.2	Register Descriptions	20-7
20.5	Functional Description	20-16
20.5.1	Enabling Nexus Clients for TAP Access	20-16
20.5.2	Configuring the NDI for Nexus Messaging	20-17
20.5.3	Switching Ownership of Nexus2+	20-18
20.5.4	Programmable MCKO Frequency	20-18
20.5.5	Nexus Messaging	20-19
20.5.6	EVTO Sharing	20-19
20.5.7	Nexus2+ DMA Control	20-19
20.5.8	Debug Mode Control	20-19
20.5.9	Nexus Reset Control	20-22

Chapter 21 Internal Static RAM (SRAM)

21.1	Introduction	21-1
21.1.1	Block Diagram	21-1
21.1.2	Features	21-2
21.1.3	Modes of Operation	21-3
21.2	External Signal Description	21-3
21.3	Memory Map and Registers	21-3
21.3.1	Array Memory Map	21-3
21.3.2	Register Descriptions	21-4
21.4	Functional Description	21-4
21.4.1	Access Timing	21-4
21.4.2	Reset Operation	21-5
21.5	DMA Requests	21-5
21.6	Interrupt Requests	21-5
21.7	Initialization/Application Information	21-5
21.7.1	Example Code	21-6

Chapter 22 Flash Array and Control

22.1	Introduction	22-1
22.2	Block Diagram	22-2
22.2.1	Features	22-3
22.2.2	Modes of Operation	22-3
22.3	External Signal Description	22-3

22.4	Memory Map and Registers	22-4
22.4.1	Module Memory Map	22-4
22.4.2	Register Descriptions	22-5
22.5	Functional Description	22-18
22.5.1	Flash User Mode	22-18
22.5.2	Flash Read and Write	22-18
22.5.3	Read While Write (RWW)	22-19
22.5.4	Flash Programming	22-19
22.5.5	Flash Erase	22-22
22.5.6	Flash Shadow Block	22-25
22.5.7	Flash Stop Mode	22-26
22.5.8	Flash Reset	22-26
22.6	DMA Requests	22-26
22.7	Interrupt Requests	22-27

Chapter 23

Deserial Serial Peripheral Interface (DSPI)

23.1	Introduction	23-1
23.1.1	Block Diagram	23-1
23.1.2	Features	23-2
23.1.3	Modes of Operation	23-4
23.2	External Signal Description	23-4
23.3	Memory Map and Registers	23-4
23.3.1	Module Memory Map	23-4
23.3.2	Register Descriptions	23-5
23.4	Functional Description	23-29
23.4.1	Modes of Operation	23-30
23.4.2	Start and Stop of DSPI Transfers	23-31
23.4.3	Serial Peripheral Interface (SPI) Configuration	23-32
23.4.4	Deserial Serial Interface (DSI) Configuration	23-35
23.4.5	Combined Serial Interface (CSI) Configuration	23-41
23.4.6	Buffered SPI Operation	23-44
23.4.7	DSPI Baud Rate and Clock Delay Generation	23-44
23.4.8	Transfer Formats	23-47
23.4.9	Continuous Serial Communications Clock	23-53
23.4.10	Peripheral Chip Select Expansion and Deglitching	23-54
23.4.11	DMA and Interrupt Conditions	23-55
23.4.12	Power Saving Features	23-56
23.5	Initialization/Application Information	23-57
23.5.1	How to Change Queues	23-57
23.5.2	Baud Rate Settings	23-58
23.5.3	Delay Settings	23-59
23.5.4	Calculation of FIFO Pointer Addresses	23-60

Chapter 24

Enhanced Serial Communication Interface (eSCI)

24.1	Introduction	24-1
24.1.1	Block Diagram	24-1
24.1.2	Features	24-2
24.1.3	Modes of Operation	24-2
24.2	External Signal Description	24-2
24.3	Memory Map and Registers	24-2
24.3.1	Module Memory Map	24-3
24.3.2	Register Descriptions	24-3
24.4	Functional Description	24-16
24.4.1	Data Format	24-17
24.4.2	Baud Rate Generation	24-18
24.4.3	Transmitter	24-19
24.4.4	Receiver	24-23
24.4.5	Single-Wire Operation	24-29
24.4.6	Loop Operation	24-30
24.4.7	Disabling the eSCI	24-30
24.4.8	Interrupt Operation	24-31
24.4.9	Using the LIN Hardware	24-34

Chapter 25

Controller Area Network (FlexCAN)

25.1	Introduction	25-1
25.1.1	Block Diagram	25-1
25.1.2	Features	25-2
25.1.3	Modes of Operation	25-3
25.2	External Signal Description	25-4
25.3	Memory Map and Registers	25-4
25.3.1	Module Memory Map	25-4
25.3.2	Message Buffer Structure	25-6
25.3.3	Rx FIFO Structure	25-9
25.3.4	Register Descriptions	25-11
25.4	Functional Description	25-28
25.4.1	Transmit Process	25-29
25.4.2	Arbitration Process	25-29
25.4.3	Receive Process	25-30
25.4.4	Matching Process	25-31
25.4.5	Data Coherence	25-33
25.4.6	Rx FIFO	25-35
25.4.7	CAN Protocol Related Features	25-36
25.4.8	Modes of Operation Details	25-39
25.4.9	Interrupts	25-41
25.4.10	Bus Interface	25-41

25.5	Initialization and Application Information	25-41
25.5.1	FlexCAN Initialization Sequence	25-42

Chapter 26

Enhanced Modular I/O Subsystem (eMIOS200)

26.1	Introduction	26-1
26.1.1	Block Diagram	26-1
26.1.2	Features	26-2
26.1.3	Modes of Operation	26-3
26.1.4	Channel Types	26-3
26.2	External Signal Description	26-4
26.2.1	eMIOS[n]	26-4
26.2.2	Output Disable Input — eMIOS200 Output Disable Input Signal	26-5
26.3	Memory Map and Registers	26-5
26.3.1	Module Memory Map	26-5
26.4	Register Descriptions	26-6
26.4.1	eMIOS200 Module Configuration Register (EMIOS_MCR)	26-6
26.4.2	eMIOS200 Global FLAG Register (EMIOS_GFR)	26-8
26.4.3	eMIOS200 Output Update Disable (EMIOS_OUDR)	26-8
26.4.4	eMIOS200 Disable Channel (EMIOSUCDIS)	26-9
26.4.5	eMIOS200 A Register (EMIOS_CADR[n])	26-9
26.4.6	eMIOS200 B Register (EMIOS_CBDR[n])	26-10
26.4.7	eMIOS200 Counter Register (EMIOS_CCNTR[n])	26-11
26.4.8	eMIOS200 Control Register (EMIOS_CCR[n])	26-11
26.4.9	eMIOS200 Status Register (EMIOS_CSR[n])	26-16
26.5	Functional Description	26-16
26.5.1	Unified Channel (UC)	26-16
26.5.2	IP Bus Interface Unit (BIU)	26-43
26.5.3	Global Clock Prescaler Submodule (GCP)	26-43
26.6	Reset	26-43
26.7	Interrupts	26-44
26.8	DMA Requests	26-44
26.9	Initialization/Application Information	26-44
26.9.1	Considerations	26-44
26.9.2	Application Information	26-44
26.9.3	Coherent Accesses	26-45

Chapter 27

Inter-Integrated Circuit Bus Controller Module (I²C)

27.1	Introduction	27-1
27.1.1	Block Diagram	27-1
27.1.2	DMA Interface	27-2
27.1.3	Features	27-3
27.1.4	Modes of Operation	27-4

27.2	External Signal Description	27-4
27.3	Memory Map and Registers	27-4
27.3.1	Module Memory Map	27-4
27.3.2	Register Descriptions	27-5
27.4	Functional Description	27-11
27.4.1	I-Bus Protocol	27-11
27.4.2	Interrupts	27-15
27.5	Initialization/Application Information	27-16
27.5.1	I ² C Programming Examples	27-16
27.5.2	DMA Application Information	27-20

Chapter 28

Periodic Interrupt Timer and Real Time Interrupt (PIT_RTI)

28.1	Introduction	28-1
28.1.1	Block Diagram	28-1
28.1.2	Features	28-2
28.1.3	Modes of Operation	28-3
28.2	Signal Description	28-3
28.2.1	External Signal Description	28-3
28.3	Memory Map and Registers	28-3
28.3.1	Module Memory Map	28-3
28.3.2	Register Descriptions	28-4
28.4	Functional Description	28-9
28.4.1	Timer / RTI	28-9
28.4.2	Debug Mode	28-10
28.4.3	Interrupts	28-10
28.5	Initialization and Application Information	28-11
28.5.1	Example Configuration	28-11

Chapter 29

External Bus Interface (EBI)

29.1	Introduction	29-1
29.1.1	Block Diagram	29-1
29.1.2	Features	29-2
29.1.3	Modes of Operation	29-3
29.2	Signal Description	29-5
29.2.1	External Signal Description	29-5
29.2.2	Signal Function and Direction by Mode	29-7
29.2.3	Signal Pad Configuration by Mode	29-8
29.3	Memory Map and Registers	29-8
29.3.1	Module Memory Map	29-8
29.3.2	Register Descriptions	29-9
29.4	Functional Description	29-16
29.4.1	External Bus Interface Features	29-16

29.4.2	External Bus Operations	29-22
29.5	Initialization/Application Information	29-48
29.5.1	Bootting from External Memory (for Factory Test only)	29-48
29.5.2	Running with Single Data Rate (SDR) Burst Memories	29-48
29.5.3	Running with Asynchronous Memories	29-48
29.5.4	Connecting an MCU to Multiple Memories	29-50
29.5.5	Dual-MCU Operation with Reduced Pinout MCUs	29-51

Chapter 30 FlexRay Communication Controller (FLEXRAY)

30.1	Introduction	30-1
30.1.1	Reference	30-1
30.1.2	Glossary	30-1
30.1.3	Color Coding	30-2
30.1.4	Overview	30-2
30.1.5	Features	30-4
30.1.6	Modes of Operation	30-5
30.2	External Signal Description	30-6
30.2.1	Detailed Signal Descriptions	30-6
30.3	Controller Host Interface Clocking	30-7
30.4	Protocol Engine Clocking	30-7
30.4.1	Oscillator Clocking	30-8
30.4.2	PLL Clocking	30-8
30.5	Memory Map and Register Description	30-8
30.5.1	Memory Map	30-8
30.5.2	Register Descriptions	30-11
30.6	Functional Description	30-78
30.6.1	Message Buffer Concept	30-78
30.6.2	Physical Message Buffer	30-78
30.6.3	Message Buffer Types	30-79
30.6.4	FlexRay Memory Layout	30-84
30.6.5	Physical Message Buffer Description	30-86
30.6.6	Individual Message Buffer Functional Description	30-95
30.6.7	Individual Message Buffer Search	30-119
30.6.8	Individual Message Buffer Reconfiguration	30-122
30.6.9	Receive FIFO	30-123
30.6.10	Channel Device Modes	30-127
30.6.11	External Clock Synchronization	30-129
30.6.12	Sync Frame ID and Sync Frame Deviation Tables	30-129
30.6.13	MTS Generation	30-132
30.6.14	Sync Frame and Startup Frame Transmission	30-133
30.6.15	Sync Frame Filtering	30-134
30.6.16	Strobe Signal Support	30-135
30.6.17	Timer Support	30-136
30.6.18	Slot Status Monitoring	30-137
30.6.19	Interrupt Support	30-140

30.6.20	Lower Bit Rate Support	30-144
30.7	Application Information	30-145
30.7.1	Initialization Sequence	30-145
30.7.2	Shut Down Sequence	30-146
30.7.3	Number of Usable Message Buffers	30-146
30.7.4	Protocol Control Command Execution	30-147
30.7.5	Protocol Reset Command	30-148
30.7.6	Message Buffer Search on Simple Message Buffer Configuration	30-149

Chapter 31

Enhanced Queued Analog-to-Digital Converter (eQADC)

31.1	Introduction	31-1
31.1.1	Block Diagram	31-2
31.1.2	Features	31-3
31.1.3	Modes of Operation	31-4
31.1.4	Normal Mode	31-4
31.1.5	Debug Mode	31-4
31.2	External Signal Description	31-5
31.3	Memory Map and Registers	31-5
31.3.1	Module Memory Map	31-6
31.3.2	Register Descriptions	31-9
31.3.3	eQADC Register Descriptions	31-9
31.3.4	On-Chip ADC Registers	31-25
31.4	Functional Description	31-31
31.4.1	Data Flow in the eQADC	31-32
31.4.2	Command/Result Queues	31-41
31.4.3	eQADC Command FIFOs	31-41
31.4.4	Result FIFOs	31-56
31.4.5	On-Chip ADC Configuration and Control	31-59
31.4.6	Internal/External Multiplexing	31-65
31.4.7	eQADC eDMA/Interrupt Request	31-70
31.4.8	Analog Submodule	31-71
31.5	Initialization/Application Information	31-74
31.5.1	Multiple Queues Control Setup Example	31-74
31.5.2	eQADC/eDMA Controller Interface	31-77
31.5.3	Sending Immediate Command Setup Example	31-78
31.5.4	Modifying Queues	31-79
31.5.5	Command Queue and Result Queue Usage	31-80
31.5.6	ADC Result Calibration	31-81

Chapter 32

Boot Assist Module (BAM)

32.1	Introduction	32-1
32.1.1	Features	32-1

32.1.2	Modes of Operation	32-2
32.1.3	Normal Mode	32-2
32.1.4	Debug Mode	32-2
32.1.5	Internal Boot Mode	32-2
32.1.6	Serial Boot Mode	32-2
32.2	Memory Map and Registers	32-2
32.2.1	Module Memory Map	32-2
32.2.2	Register Descriptions	32-3
32.3	Functional Description	32-3
32.3.1	BAM Program Resources	32-3
32.3.2	BAM Program Operation	32-3
32.3.3	Features	32-5

Chapter 33 Media Local Bus (MLB)

33.1	Introduction	33-1
33.1.1	Block Diagram	33-1
33.1.2	Features	33-2
33.1.3	Modes of Operation	33-2
33.2	External Signal Description	33-3
33.3	Memory Map and Registers	33-4
33.3.1	Register Descriptions	33-5
33.4	Functional Description	33-18
33.4.1	SoftMLB Interface Logic Description	33-21
33.4.2	SoftMLB Interface Logic Signal Description	33-22

Appendix A Revision History

A.1	Changes Between Revisions 0 and 1	A-1
-----	-----------------------------------	-----

Chapter 1

Overview

1.1 Introduction

The MPC5510 is a family of next generation microcontrollers built on the Power Architecture™ embedded category. This document describes the proposed features of the family and potential options available within the planned family members, and highlights the important electrical and physical characteristics of the device. This is a preliminary document for a product family that is still in development. Its purpose is to communicate information on the intended features of the family members. Information contained within this document is subject to change without notice.

NOTE: Bit and Field Numbering Conventions

In this reference manual, register bits and fields are generally numbered according to the convention used in the Power Architecture standard (MSB=0); however, in some instances the bit/field numbering may appear to be reversed. This is due to the fact that some of the modules were designed for use on devices that use either the MSB=0 numbering convention or the alternative convention (LSB=0), for example, the HC12 and 68K families, and simple reversing of bit/field numbers is not possible.

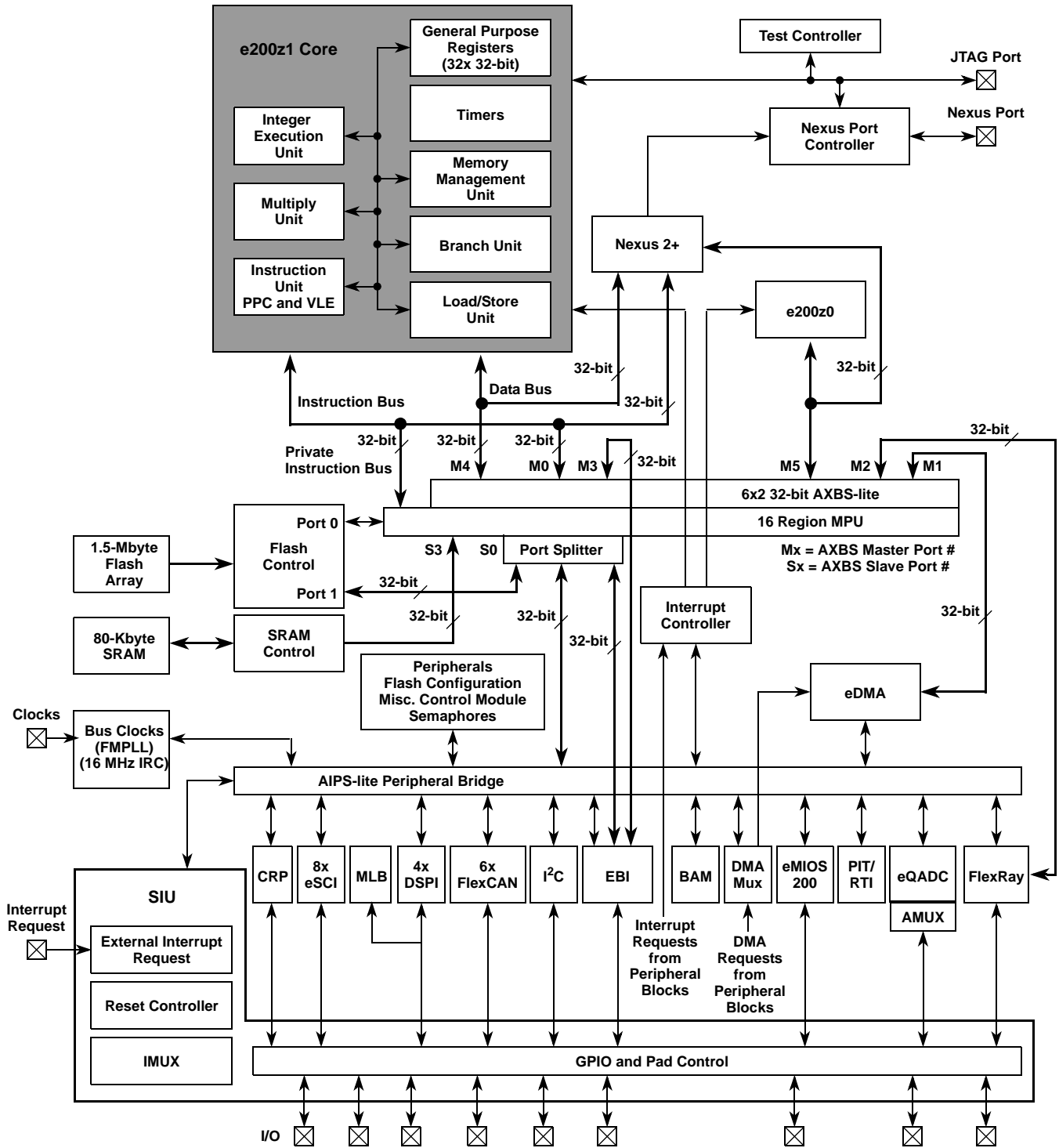
In the Nexus standard, register bits are numbered according to the alternative convention (LSB=0). As the CPU core on the MPC5510 family cannot access Nexus registers directly (they are accessed through external tools), register bits are numbered according to the LSB=0 convention in the Nexus chapter.

The MPC5510 family of 32-bit microcontrollers is Freescale Semiconductor's latest achievement in integrated automotive application controllers. It belongs to an expanding family of automotive-focused products designed to address the next wave of central body and gateway applications within the vehicle. Freescale's advanced and cost-efficient host processor core of the MPC5510 automotive controller family is compatible with the Power Architecture Book E architecture. It operates at speeds of up to 80 MHz and offers high-performance processing optimized for low-power consumption. It capitalizes on the available development infrastructure of the current Power Architecture devices and will be supported with software drivers, operating systems, and configuration code to assist with user implementations.

The MPC5510 platform has a single level of memory hierarchy and can support up to 80 KB of on-chip static random access memory (SRAM) and 1.5 MB of internal flash memory. Refer to [Table 1-1](#) for specific memory and feature sets of the proposed roadmap product members.

1.2 Block Diagram

Figure 1-1 illustrates the functionality and interdependence of major blocks of the MPC5516.



Note: The e200z1 is called Processor 0, and the e200z0 is called Processor 1 throughout this document

Figure 1-1. MPC5516 Block Diagram

1.3 MPC5510 Family Comparison

[Table 1-1](#) provides a summary of the different members of the MPC5510 family and their proposed features. This information is intended to provide an understanding of the range of functionality offered by this family.

Table 1-1. MPC5510 Family Comparison, Maximum Feature Set¹

Feature	MPC5517G	MPC5517E		MPC5517S		MPC5516G		MPC5516E		MPC5516S		MPC5515S		MPC5514G	MPC5514E
Package	208-BGA	144-LQFP	208-BGA/ 176-LQFP	144-LQFP	208-BGA/ 176-LQFP	144-LQFP	208-BGA	144-LQFP	208-BGA/ 176-LQFP	144-LQFP	176-LQFP	144-LQFP	176-LQFP	144-LQFP	144-LQFP
Main CPU	e200z1														
Maximum Execution Speed ²	80 MHz at Ta=105C 75 MHz at Ta=125C	80 MHz at Ta=105C 75 MHz at Ta=125C		66 MHz		80 MHz at Ta=105C 75 MHz at Ta=125C		80 MHz at Ta=105C 75 MHz at Ta=125C		66 MHz		66 MHz		66 MHz	
Flash ³	1.5 MB	1.5 MB		1.5 MB		1 MB		1 MB		1 MB		768 KB		512 KB	
RAM	80 KB	80 KB		64 KB		64 KB		64 KB		48 KB		48 KB		32 KB	
I/O Processor	e200z0	e200z0		—		e200z0		e200z0		—		—		e200z0	
DMA	Yes	Yes		Yes		Yes		Yes		Yes		Yes		Yes	
MPU	16 entry	16 entry		8 entry		16 entry		16 entry		8 entry		8 entry		16 entry	
ADC ⁴	40 channels, 12-bit (16 channels input only; 24 channels bidirectional)														
Total Timed I/O ⁵ eMIOS200	24 channels, 16-bit (8 channels IC/OC; 16 channels PWM, IC/OC)														
Real-Time Clock	ext 32 KHz Crystal	ext 32 KHz Crystal		—		ext 32 KHz Crystal		ext 32 KHz Crystal		—		—		ext 32 KHz Crystal	
SCI	6x eSCI	6x eSCI	8x eSCI	6x eSCI		6x eSCI		6x eSCI	8x eSCI	6x eSCI		6x eSCI		6x eSCI	
SPI	4x DSPI	4x DSPI		4x DSPI		4x DSPI		4x DSPI		3x DSPI		3x DSPI		4x DSPI	
SPI Chip Selects	24	23 ⁶	24	23 ⁶	24	23 ⁶	24	23 ⁶	24	18		18		23 ⁶	
CAN	6x FlexCAN	5x FlexCAN		4x FlexCAN	5x FlexCAN	6x FlexCAN		5x FlexCAN		4x FlexCAN	5x FlexCAN	4x FlexCAN	5x FlexCAN	6x FlexCAN	
FlexRay	Yes	—		—		Yes		—		—		—		Yes	
MLB ⁷	Yes	Yes		—		Yes		Yes		—		—		Yes	
I ² C	1														
EBI ⁸	Yes ⁹	Yes ¹⁰	Yes ⁹	Yes ¹⁰	Yes ⁹	Yes ¹⁰	Yes ⁹	Yes ¹⁰	Yes ⁹	—		—		Yes ¹⁰	
GPIO ¹¹	144	111	144/137	111	144/137	111	144/137	111	144/137	111	137	111	137	111	111

¹ Maximum feature set displayed for each family member. Feature set depends on selected peripheral multiplexing.
² Maximum speed is 66 MHz on 144-LQFP and 176-LQFP package options.
³ EEPROM emulation supported by small flash blocks with read-while-write operation as part of main array space.
⁴ ADC channel accuracy greater for input-only channel, bidirectional channels offer the ability for unused channels to be used as outputs.
⁵ IC—input capture; O/C—output compare; PWM—pulse-width modulation.
⁶ For devices with four DSPI modules, in the 144-pin package, it is not possible to bring out all 24 DSPI chip selects. Hence, three modules can have six chip selects, but one module can have only five.
⁷ MLB is emulated in software and requires the following resources: I/O Processor, 2xDSPI, 4x eDMA channels, RAM, SoftMLB Interface Logic.
⁸ In the 208-pin package, there can be up to 24 address bits with 32-bit data and four chip selects. In the 144-pin and 176-pin packages, there are 24 address bits with 16-bit data and four chip selects.
⁹ 16-bit or 32-bit multiplexed data bus supported. EBI multiplexed with other functions shown as available.
¹⁰ 16-bit multiplexed data bus supported. EBI multiplexed with other functions shown as available.
¹¹ Estimated I/O count for proposed packages based on multiplexing with peripherals.

1.3.1 Family Feature Set Scaling

The MPC5510 family supports multiple functions on most of the pins. This allows flexibility in the positioning and the availability of device features. It is the user's choice what trade-offs are made between the feature set used for the available pin count through this device pin multiplexing. The available features implemented on silicon will be incrementally added as the family functionality increases. [Table 1-2](#) provides a summary of the flash array address space supported by the different device memory sizes. [Table 1-3](#) provides a summary of the RAM array address space supported by the different device memory sizes. [Table 1-4](#) provides a summary of the available peripheral functionality of each family member.

Evaluation of the pin list for each device will be necessary as it may not be possible to retain all modules sequentially, depending on the selected pin multiplexing trade-offs on each device.

NOTE

The RAppID™ initialization tool provides a pin allocation wizard that allows users to graphically configure I/O to meet the requirements of the peripheral functions. More information on this tool can be found at <http://www.freescale.com/mpc55xx>.

Table 1-2. Flash Memory Scaling

Memory Size	Start Address	End Address
1.5 MB	0x0000_0000	0x0017_FFFF
1 MB	0x0000_0000	0x000F_FFFF
768 KB	0x0000_0000	0x000B_FFFF
512 KB	0x0000_0000	0x0007_FFFF

Table 1-3. RAM Memory Scaling

Memory Size	Start Address	End Address
80 KB	0x4000_0000	0x4001_3FFF
64 KB	0x4000_0000	0x4000_FFFF
48 KB	0x4000_0000	0x4000_BFFF
32 KB	0x4000_0000	0x4000_7FFF

Table 1-4. Peripheral Scaling

		MPC5517					MPC5516					MPC5515		MPC5514	
		G	E		S		G	E		S		S		G	E
	Package	208	144	176/ 208	144	176	144/208	144	176/ 208	144	176	144	176	144	144
MPU	Regions	16	16	16	8	8	16	16	16	8	8	8	8	16	16
SCI	Number	6	6	8	6	6	6	6	8	6	6	6	6	6	6
	Module	A,B,C,D, E,F	A,B,C,D, E,F	A,B,C,D, E,F,G,H	A,B,C,D, E,F	A,B,C,D, E,F	A,B,C,D, E,F	A,B,C,D, E,F	A,B,C,D, E,F,G,H	A,B,C,D, E,F	A,B,C,D, E,F	A,B,C,D, E,F	A,B,C,D, E,F	A,B,C,D, E,F	A,B,C,D, E,F
SPI	Number	4	4	4	4	4	4	4	4	3	3	3	3	3	4
	Module	A,B,C,D	A,B,C,D	A,B,C,D	A,B,C,D	A,B,C,D	A,B,C,D	A,B,C,D	A,B,C,D	A,B,C	A,B,C	A,B,C	A,B,C	A,B,C	A,B,C,D
CAN	Number	6	5	5	4	5	6	5	5	4	5	4	5	6	5
	Module	A,B,C,D, E,F	A,C,D,E, F	A,C,D,E, F	A,C,D,E	A,C,D,E, F	A,B,C,D, E,F	A,C,D,E, F	A,C,D,E, F	A,C,D,E, F	A,C,D,E	A,C,D,E, F	A,C,D,E, F	A,C,D,E, F	A,B,C,D, E,F

1.4 Chip-Level Features

On-chip modules available within the family include the following features:

- Single issue, 32-bit CPU core complex (e200z1)
 - Compliant with the Power Architecture embedded category
 - Includes an instruction set enhancement allowing variable length encoding (VLE) for code size footprint reduction. With the optional encoding of mixed 16-bit and 32-bit instructions, it is possible to achieve significant code-size footprint reduction.
- Up to 1.5 MB of on-chip flash with flash control unit (FCU)
- Up to 80 KB on-chip SRAM
- Memory protection unit (MPU) with up to 16 region descriptors and 32-byte region granularity
- Interrupt controller (INTC) capable of handling selectable-priority interrupt sources
- Frequency modulated Phase-locked loop (FMPLL)
- Crossbar switch architecture for concurrent access to peripherals, flash, or RAM from multiple bus masters
- A 16-channel enhanced direct memory access controller (eDMA)
- Boot assist module (BAM) supports internal flash programming via a serial link (CAN or SCI)
- Timer supports input/output channels providing a range of 16-bit input capture, output compare, and pulse-width modulation functions (eMIOS200)
- A 12-bit analog-to-digital converter (ADC)
- Up to four serial peripheral interface (DSPI) modules
- Media Local Bus (MLB) emulation logic which works in conjunction with two DSPI, the e200z0, the eDMA, and system RAM to create a 3-pin or 5-pin 256Fs Media Local Bus interface
- Up to eight serial communication interface (eSCI) modules
- Up to six enhanced full CAN (FlexCAN) modules with configurable buffers
- One inter IC communication interface (I²C) module
- Up to 144 configurable general-purpose pins supporting input and input/output operations
- Real-time counter (RTC_API) with clock source from external 32 kHz crystal oscillator, internal 32 kHz or 16 MHz oscillator and supporting wakeup with selectable 1 sec. resolution and >1 hour timeout, or 1 mS resolution with max timeout of 1 sec.
- Up to eight periodic interrupt timers (PIT) with 32-bit counter resolution
- Nexus development interface (NDI) per IEEE-ISTO 5001-2003 Class Two Plus standard
- Device/board test support per joint test action group (JTAG) of IEEE (IEEE 1149.1)
- On-chip voltage regulator (VREG) regulation of input supply for all internal levels
- Optional e200z0, second I/O processor built on Power Architecture technology with VLE instruction set
- Optional FlexRay controller
- Optional external bus interface (EBI) module

1.5 Low-Power Operation

The MPC5510 has two dynamic-power modes and three static-power modes:

- Low-power modes use clock gating to halt the clock for all or part of the device.
- The lowest power modes also use power gating to automatically turn off the power supply to parts of the device to minimize leakage.
- Dynamic-power mode is RUN:
 - RUN mode is the main full performance operating mode where the entire device is powered and clocked. The user can configure the device operating speed through selection of the clock source and the phase-locked loop (FMPLL) frequency. Clock gating can be performed on a peripheral by peripheral basis to select which device features have their clock halted to save power. When implemented, the I/O processor can optionally be enabled, allowing execution of code and access to the memory and peripherals of the device.
- Static-power modes are STOP and SLEEP:
 - STOP mode maintains power to the entire device allowing the retention of all on-chip registers and memory, and providing a fast recovery low-power mode with no need to reconfigure the device. The clocks are halted to the cores and peripherals, with the exception of the RTC, and can be optionally stopped to the oscillator or FMPLL at the expense of a slower start-up time. STOP is entered from RUN mode. On exiting STOP mode the device returns to the RUN mode.
 - SLEEP mode halts the clock to the entire device, with the exception of the RTC, and turns off the power to the majority of the chip to offer the lowest power consumption modes of the MPC5510. SLEEP mode retains the output levels on the pins, but power gating means that the contents of the cores, on-chip peripheral registers, and some of the volatile memory are not held. The device can be awakened from selected I/O pins, a reset, or from a periodic wakeup using a low-power oscillator. If required by the user, it is possible to enable the internal 16 MHz or 32 kHz RC oscillator or external 32 kHz oscillator. The user can select the desired level of RAM to be retained as the following: full contents of the on-chip SRAM, 64K, 32K, 16K, 8K, no RAM retained.
 - Fast wake-up using the on-chip 16 MHz internal RC oscillator allowing rapid execution on exit from low-power modes.
- 16 MHz internal RC oscillator supports low-speed code execution and clocking of peripherals

1.6 Memory Map

Table 1-5. Detailed MPC5510 Family Memory Map

Address Range ¹	Allocated Size ¹ (bytes)	Use
0x0000_0000–0x0017_FFFF	1.5 M	Flash Memory Array
0x0018_0000–0x00FF_7FFF	14.5 M – 32K	Reserved
0x00FF_8000–0x00FF_FFFF	32 K	Flash Shadow Row
0x0100_0000–0x1FFF_FFFF	496 M	Emulation mapping of Flash Array

Table 1-5. Detailed MPC5510 Family Memory Map (continued)

Address Range ¹	Allocated Size ¹ (bytes)	Use
0x2000_0000–0x3FFF_FFFF	512 M	External Memory
0x4000_0000–0x4000_1FFF	8 K	Internal SRAM Array. Powered during Sleep when CRP_PSCR[RAMSEL] = 1 to 7
0x4000_2000–0x4000_3FFF	8 K	Internal SRAM Array. Powered during Sleep when CRP_PSCR[RAMSEL] = 2 to 7
0x4000_4000–0x4000_7FFF	16 K	Internal SRAM Array. Powered during Sleep when CRP_PSCR[RAMSEL] = 3 to 7
0x4000_8000–0x4000_FFFF	32 K	Internal SRAM Array. Powered during Sleep when CRP_PSCR[RAMSEL] = 6 to 7
0x4001_0000–0x4001_3FFF	16 K	Internal SRAM Array. Powered during Sleep when CRP_PSCR[RAMSEL] = 7
0x4001_4000–0xDFFF_FFFF	2560 M – 80 K	Reserved
Peripherals		
0xE000_0000–0xFBFF_FFFF	512 M – 64 M	Reserved
0xFC00_0000–0xFFFF0_FFFF	63 M + 64 K	Reserved
0xFFF1_0000–0xFFF1_3FFF	16 K	Semaphores
0xFFF1_4000–0xFFF1_7FFF	16 K	Memory Protection Unit (MPU)
0xFFF1_8000–0xFFF3_FFFF	160 K	Reserved
0xFFF4_0000–0xFFF4_3FFF	16 K	Miscellaneous Control Module (MCM)
0xFFF4_4000–0xFFF4_7FFF	16 K	Enhanced Direct Memory Access Controller (eDMA)
0xFFF4_8000–0xFFF4_BFFF	16 K	Interrupt Controller (INTC)
0xFFF4_C000–0xFFF7_FFFF	208 K	Reserved
0xFFF8_0000–0xFFF8_3FFF	16 K	Enhanced Queued Analog-to-Digital Converter (eQADC)
0xFFF8_4000–0xFFF8_7FFF	16 K	SoftMLB Interface Logic
0xFFF8_8000–0xFFF8_BFFF	16 K	I ² C Controller (I2C_A)
0xFFF8_C000–0xFFF8_FFFF	16 K	Reserved
0xFFF9_0000–0xFFF9_3FFF	16 K	Deserial Serial Peripheral Interface (DSPI_A)
0xFFF9_4000–0xFFF9_7FFF	16 K	Deserial Serial Peripheral Interface (DSPI_B)
0xFFF9_8000–0xFFF9_BFFF	16 K	Deserial Serial Peripheral Interface (DSPI_C)
0xFFF9_C000–0xFFF9_FFFF	16 K	Deserial Serial Peripheral Interface (DSPI_D)
0xFFFFA_0000–0xFFFFA_3FFF	16 K	Serial Communications Interface (eSCI_A)
0xFFFFA_4000–0xFFFFA_7FFF	16 K	Serial Communications Interface (eSCI_B)
0xFFFFA_8000–0xFFFFA_BFFF	16 K	Serial Communications Interface (eSCI_C)
0xFFFFA_C000–0xFFFFA_FFFF	16 K	Serial Communications Interface (eSCI_D)
0xFFFFB_0000–0xFFFFB_3FFF	16 K	Serial Communications Interface (eSCI_E)

Table 1-5. Detailed MPC5510 Family Memory Map (continued)

Address Range ¹	Allocated Size ¹ (bytes)	Use
0xFFFFB_4000–0xFFFFB_7FFF	16 K	Serial Communications Interface (eSCI_F)
0xFFFFB_8000–0xFFFFB_FFFF	16 K	Serial Communications Interface (eSCI_G)
0xFFFFB_C000–0xFFFFB_FFFF	16 K	Serial Communications Interface (eSCI_H)
0xFFFFC_0000–0xFFFFC_3FFF	16 K	Controller Area Network (FlexCAN_A)
0xFFFFC_4000–0xFFFFC_7FFF	16 K	Controller Area Network (FlexCAN_B)
0xFFFFC_8000–0xFFFFC_BFFF	16 K	Controller Area Network (FlexCAN_C)
0xFFFFC_C000–0xFFFFC_FFFF	16 K	Controller Area Network (FlexCAN_D)
0xFFFFD_0000–0xFFFFD_3FFF	16 K	Controller Area Network (FlexCAN_E)
0xFFFFD_4000–0xFFFFD_7FFF	16 K	Controller Area Network (FlexCAN_F)
0xFFFFD_8000–0xFFFFD_BFFF	16 K	FlexRay Controller (FlexRay)
0xFFFFD_C000–0xFFFFD_FFFF	16 K	DMA Multiplexer (DMA_MUX)
0xFFFFE_0000–0xFFFFE_3FFF	16 K	Programmable Interrupt / Real Time Interrupt (PIT_RTI)
0xFFFFE_4000–0xFFFFE_7FFF	16 K	Enhanced Modular I/O Subsystem (eMIOS200)
0xFFFFE_8000–0xFFFFE_BFFF	16K	System Integration Unit (SIU)
0xFFFFE_C000–0xFFFFE_FFFF	16 K	Clocks, Reset and Power (CRP)
0xFFFFF_0000–0xFFFFF_3FFF	16 K	FMPLL Registers (FMPLL)
0xFFFFF_4000–0xFFFFF_7FFF	16 K	External Bus Interface Configuration Registers (EBI)
0xFFFFF_8000–0xFFFFF_BFFF	16 K	Flash Configuration Registers (FLASH)
0xFFFFF_C000–0xFFFFF_FFFF	16 K	Boot Assist Module (BAM)

¹ Refer to the individual module chapters for a description of how the allocated size is used.

Chapter 2

Signal Descriptions

2.1 Introduction

This chapter describes signals that connect off-chip. It includes a signal properties summary, power and ground segmentation summary, package pinouts, and detailed descriptions of signals. Because the MPC5510 comes in multiple packages, some signals will not be available on every package. Refer to the *MPC5510 Microcontroller Family Data Sheet* for electrical characteristics.

2.2 Signal Properties Summary

Table 2-1 shows the signals properties for each pin on MPC5510. For all port pins, which have an associated SIU_PCRx register to control its pin properties, the supported functions column lists the functions associated with the programming of the SIU_PCRx[PA] bit in the order: general-purpose input/output (GPIO), function 1, function 2, and function 3. If fewer than three functions and GPIO is supported by a given pin, then the unused functions begin with function 3, then function 2, then function 1 (see Figure 2-1).



Figure 2-1. Supported Functions Example

Table 2-1. MPC5510 Signal Properties

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
Port A (16) (Section/Page: 2.7.1/2-16)											
PA0	0	PA[0] AN[0]	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	9	9	E3
PA1	1	PA[1] AN[1]	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	8	8	E2
PA2	2	PA[2] AN[2]	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	7	7	E1
PA3	3	PA[3] AN[3]	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	6	6	D3

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
PA4	4	PA[4] AN[4]	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	5	5	D2
PA5	5	PA[5] AN[5]	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	4	4	D1
PA6	6	PA[6] AN[6]	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	3	3	C2
PA7	7	PA[7] AN[7]	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	2	2	C1
PA8	8	PA[8] AN[8]/ANW	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	143	175	A3
PA9	9	PA[9] AN[9]/ANX	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	142	174	C4
PA10	10	PA[10] AN[10]/ANY	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	140	172	D5
PA11	11	PA[11] AN[11]/ANZ	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	139	171	C5
PA12	12	PA[12] AN[12]	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	138	170	B5
PA13	13	PA[13] AN[13]	GPI eQADC Analog Input	I I	V _{DDA}	AE + IH	—	—	137	169	A5
PA14	14	PA[14] AN[14] EXTAL32 ⁵	GPI eQADC Analog Input 32 kHz Crystal Oscillator Input	I I I	V _{DDA}	AE + IH	—	—	136	167	D6
PA15	15	PA[15] AN[15] XTAL32 ⁵	GPI eQADC Analog Input 32 kHz Crystal Oscillator Output	I I O	V _{DDA}	AE + IH	—	—	135	165	C6
Port B (16) (Section/Page: 2.7.2/2-17)											
PB0	16	PB[0] AN[28] eMIOS[16] PCS_C[5]	GPIO eQADC Analog Input ⁶ eMIOS Channel DSPI_C Peripheral Chip Select	I/O I O O	V _{DDE1}	A + SH	—	—	134	162	C7
PB1	17	PB[1] AN[29] eMIOS[17] PCS_C[4]	GPIO eQADC Analog Input ⁶ eMIOS Channel DSPI_C Peripheral Chip Select	I/O I O O	V _{DDE1}	A + SH	—	—	133	161	D7
PB2	18	PB[2] AN[30] eMIOS[18] PCS_C[3]	GPIO eQADC Analog Input ⁶ eMIOS Channel DSPI_C Peripheral Chip Select	I/O I O O	V _{DDE1}	A + SH	—	—	132	160	A8
PB3	19	PB[3] AN[31] PCS_C[2]	GPIO eQADC Analog Input ⁶ DSPI_C Peripheral Chip Select	I/O I O	V _{DDE1}	A + SH	—	—	131	159	B8
PB4	20	PB[4] AN[32] PCS_C[1]	GPIO eQADC Analog Input ⁶ DSPI_C Peripheral Chip Select	I/O I O	V _{DDE1}	A + SH	—	—	130	158	C8

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
PB5	21	PB[5] AN[33] PCS_C[0]	GPIO eQADC Analog Input ⁶ DSPI_C Peripheral Chip Select	I/O I O	V _{DDE1}	A + SH	—	—	129	157	D8
PB6	22	PB[6] AN[34] SCK_C	GPIO eQADC Analog Input ⁶ DSPI_C Clock	I/O I I/O	V _{DDE1}	A + SH	—	—	128	156	A9
PB7	23	PB[7] AN[35] SOUT_C	GPIO eQADC Analog Input ⁶ DSPI_C Data Output	I/O I O	V _{DDE1}	A + SH	—	—	127	153	B9
PB8	24	PB[8] AN[36] SIN_C	GPIO eQADC Analog Input ⁶ DSPI_C Data Input	I/O I I	V _{DDE1}	A + SH	—	—	126	152	C9
PB9	25	PB[9] AN[37] CNTX_D PCS_B[4]	GPIO eQADC Analog Input ⁶ CAN_D Transmit DSPI_B Peripheral Chip Select	I/O I O O	V _{DDE1}	A + SH	—	—	125	151	D9
PB10	26	PB[10] AN[38] CNRX_D PCS_B[3]	GPIO eQADC Analog Input ⁶ CAN_D Receive DSPI_B Peripheral Chip Select	I/O I I O	V _{DDE1}	A + SH	—	—	124	150	A10
PB11	27	PB[11] AN[39] eMIOS[19] PCS_B[5]	GPIO eQADC Analog Input ⁶ eMIOS Channel DSPI_B Peripheral Chip Select	I/O I O O	V _{DDE1}	A + SH	—	—	123	149	B10
PB12	28	PB[12] TXD_G PCS_B[4]	GPIO SCI_G Transmit DSPI_B Peripheral Chip Select	I/O O O	V _{DDE1}	SH	—	—	—	164	A7
PB13	29	PB[13] RXD_G PCS_B[3]	GPIO SCI_G Receive DSPI_B Peripheral Chip Select	I/O I O	V _{DDE1}	SH	—	—	—	163	B7
PB14	30	PB[14] TXD_H	GPIO SCI_H Transmit	I/O O	V _{DDE1}	SH	—	—	—	148	C10
PB15	31	PB[15] RXD_H	GPIO SCI_H Receive	I/O I	V _{DDE1}	SH	—	—	—	147	A11
Port C (16) (Section/Page: 2.7.3/2-19)											
PC0	32	PC[0] eMIOS[0] FR_A_TX_EN AD[24]	GPIO eMIOS Channel FlexRay Channel A Transmit Enable EBI Multiplexed Address/Data	I/O I/O O I/O	V _{DDE1}	MH	—	—	122	146	B11
PC1	33	PC[1] eMIOS[1] FR_A_TX AD[16]	GPIO eMIOS Channel FlexRay Channel A Transmit EBI Multiplexed Address/Data	I/O I/O O I/O	V _{DDE1}	MH	—	—	121	145	C11
PC2	34	PC[2] eMIOS[2] FR_A_RX TS	GPIO eMIOS Channel FlexRay Channel A Receive EBI Transfer Start	I/O I/O I I/O	V _{DDE1}	MH	—	—	120	144	D11

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
PC3	35	PC[3] eMIOS[3] FR_DBG0	GPIO eMIOS Channel FlexRay Debug	I/O I/O O	V _{DDE1}	MH	—	—	117	141	A12
PC4	36	PC[4] eMIOS[4] FR_DBG1	GPIO eMIOS Channel FlexRay Debug	I/O I/O O	V _{DDE1}	SH	—	—	116	140	B12
PC5	37	PC[5] eMIOS[5] FR_DBG2	GPIO eMIOS Channel FlexRay Debug	I/O I/O O	V _{DDE1}	SH	—	—	115	139	C12
PC6	38	PC[6] eMIOS[6] FR_DBG3	GPIO eMIOS Channel FlexRay Debug	I/O I/O O	V _{DDE1}	SH	—	—	114	138	D12
PC7	39	PC[7] eMIOS[7] FR_B_RX	GPIO eMIOS Channel FlexRay Channel B Receive	I/O I/O I	V _{DDE1}	SH	—	—	113	137	A13
PC8	40	PC[8] eMIOS[8] FR_B_TX AD[15]	GPIO eMIOS Channel FlexRay Channel B Transmit EBI Multiplexed Address/Data	I/O I/O O I/O	V _{DDE1}	MH	—	—	112	136	B13
PC9	41	PC[9] eMIOS[9] FR_B_TX_EN AD[14]	GPIO eMIOS Channel FlexRay Channel B Transmit Enable EBI Muxed Address/Data	I/O I/O O I/O	V _{DDE1}	MH	—	—	111	135	C13
PC10	42	PC[10] eMIOS[10] PCS_C[5] SCK_D	GPIO eMIOS Channel DSPI_C Peripheral Chip Select DSPI_D Clock	I/O I/O O I/O	V _{DDE1}	SH	—	—	110	134	A14
PC11	43	PC[11] eMIOS[11] PCS_C[4] SOUT_D	GPIO eMIOS Channel DSPI_C Peripheral Chip Select DSPI_D Serial Out	I/O I/O O O	V _{DDE1}	SH	—	—	109	133	B14
PC12	44	PC[12] eMIOS[12] PSC_C[3] SIN_D	GPIO eMIOS Channel DSPI_C Peripheral Chip Select DSPI_D Serial In	I/O I/O O I	V _{DDE1}	SH	—	—	108	132	B16
PC13	45	PC[13] eMIOS[13] PCS_A[5] PCS_D[0]	GPIO eMIOS Channel DSPI_A Peripheral Chip Select DSPI_D Peripheral Chip Select	I/O I/O O O	V _{DDE1}	SH	—	—	107	131	C15
PC14	46	PC[14] eMIOS[14] PCS_A[4] PCS_D[1]	GPIO eMIOS Channel DSPI_A Peripheral Chip Select DSPI_D Peripheral Chip Select	I/O I/O O O	V _{DDE1}	SH	—	—	106	130	C16
PC15	47	PC[15] eMIOS[15] PCS_A[3] PCS_D[2]	GPIO eMIOS Channel DSPI_A Peripheral Chip Select DSPI_D Peripheral Chip Select	I/O I/O O O	V _{DDE1}	SH	—	—	105	129	D14

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
Port D (16) (Section/Page: 2.7.4/2-21)											
PD0	48	PD[0] CNTX_A PCS_D[3]	GPIO CAN_A Transmit DSPI_D Peripheral Chip Select	I/O O O	V _{DDE1}	SH	—	—	104	128	D15
PD1	49	PD[1] CNRX_A PCS_D[4]	GPIO CAN_A Receive DSPI_D Peripheral Chip Select	I/O I O	V _{DDE1}	SH	—	—	103	127	D16
PD2	50	PD[2] CNRX_B eMIOS[10] PCS_D[5] BOOTCFG ⁷	GPIO CAN_B Receive eMIOS Channel DSPI_D Peripheral Chip Select Boot Configuration	I/O I O O I	V _{DDE1}	SH	BOOTCFG (Pulldown)	GPI (Pulldown)	102	126	E14
PD3	51	PD[3] CNTX_B eMIOS[11]	GPIO CAN_B Transmit eMIOS Channel	I/O O O	V _{DDE1}	SH	—	—	101	125	E15
PD4	52	PD[4] CNTX_C eMIOS[12]	GPIO CAN_C Transmit eMIOS Channel	I/O O O	V _{DDE1}	SH	—	—	100	124	E16
PD5	53	PD[5] CNRX_C eMIOS[13]	GPIO CAN_C Receive eMIOS Channel	I/O I O	V _{DDE1}	SH	—	—	99	123	F13
PD6	54	PD[6] TXD_A eMIOS[14]	GPIO SCI_A Transmit eMIOS Channel	I/O O O	V _{DDE1}	SH	—	—	98	122	F14
PD7	55	PD[7] RXD_A eMIOS[15]	GPIO SCI_A Receive eMIOS Channel	I/O I O	V _{DDE1}	SH	—	—	97	121	F15
PD8	56	PD[8] TXD_B SCL_A	GPIO SCI_B Transmit I ² C Serial Clock Line	I/O O I/O	V _{DDE1}	SH	—	—	94	118	G13
PD9	57	PD[9] RXD_B SDA_A	GPIO SCI_B Receive I ² C Serial Data Line	I/O I I/O	V _{DDE1}	SH	—	—	93	117	F16
PD10	58	PD[10] PCS_B[2] CNTX_F NMI0	GPIO DSPI_B Peripheral Chip Select CAN_F Transmit NMI Input for Z1 Core	I/O O O I	V _{DDE1}	SH	—	—	92	116	G14
PD11	59	PD[11] PCS_B[1] CNRX_F NMI1	GPIO DSPI_B Peripheral Chip Select CAN_F Receive NMI Input for Z0 Core	I/O O I I	V _{DDE1}	SH	—	—	91	115	G15
PD12	60	PD[12] PCS_B[0] eMIOS[9]	GPIO DSPI_B Peripheral Chip Select eMIOS Channel	I/O I/O O	V _{DDE1}	SH	—	—	90	114	H14
PD13	61	PD[13] SCK_B eMIOS[8]	GPIO DSPI_B Clock eMIOS Channel	I/O I/O O	V _{DDE1}	SH	—	—	89	113	H15

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
PD14	62	PD[14] SOUT_B eMIO[7]	GPIO DSPI_B Data Output eMIO Channel	I/O O O	V _{DDE1}	SH	—	—	88	110	J14
PD15	63	PD[15] SIN_B eMIO[6]	GPIO DSPI_B Data Input eMIO Channel	I/O I O	V _{DDE1}	SH	—	—	87	107	K14
Port E (16) (Section/Page: 2.7.5/2-24)											
PE0	64	PE[0] PCS_A[2] eMIO[5] MLBCLK	GPIO DSPI_A Peripheral Chip Select eMIO Channel MLB Clock	I/O O O I	V _{DDE1}	SH	—	—	86	106	K16
PE1	65	PE[1] PCS_A[1] eMIO[4] MLBSI / MLBSIG	GPIO DSPI_A Peripheral Chip Select eMIO Channel MLB Signal In (5-pin) / MLB Bidirectional Signal (3-pin)	I/O O O I I/O	V _{DDE1}	MH	—	—	85	103	L14
PE2	66	PE[2] PCS_A[0] eMIO[3] MLBDI / MLBDAT	GPIO DSPI_A Peripheral Chip Select eMIO Channel MLB Data In (5-pin) / MLB Bidirectional Data (3-pin)	I/O I/O O I I/O	V _{DDE1}	MH	—	—	84	101	L15
PE3	67	PE[3] SCK_A eMIO[2] MLBSO / MLBSIG_BUFEN	GPIO DSPI_A Clock eMIO Channel MLB Signal Out (5-pin) / MLB Signal Level Shifter Enable (3-pin)	I/O I/O O O O O	V _{DDE1}	MH	—	—	83	100	M13
PE4	68	PE[4] SOUT_A eMIO[1] MLBDO / MLBDAT_BUFEN	GPIO DSPI_A Data Out eMIO Channel MLB Data Out (5-pin) / MLB Data Level Shifter Enable (3-pin)	I/O O O O O O	V _{DDE1}	MH	—	—	82	98	N14
PE5	69	PE[5] SIN_A eMIO[0] MLB_SLOT / MLB_SIGOBS / MLB_DATOBS	GPIO DSPI_A Data In eMIO Channel MLB Slot Debug / MLB Clock Adjust Observe Signal / MLB Clock Adjust Observe Data	I/O I O O O O O	V _{DDE1}	MH	—	—	81	97	M15
PE6	70	PE[6] CLKOUT	GPIO System Clock Output	I/O O	V _{DDE3}	MH	—	—	67	83	P13
PE7	71	PE[7]	GPIO	I/O	V _{DDE1}	SH	—	—	—	—	H13
PE8	72	PE[8]	GPIO	I/O	V _{DDE1}	SH	—	—	—	—	H16
PE9	72	PE[9]	GPIO	I/O	V _{DDE1}	SH	—	—	—	—	J13
PE10	74	PE[10]	GPIO	I/O	V _{DDE1}	SH	—	—	—	112	J16
PE11	75	PE[11]	GPIO	I/O	V _{DDE1}	SH	—	—	—	111	J15
PE12	76	PE[12]	GPIO	I/O	V _{DDE1}	SH	—	—	—	109	K13
PE13	77	PE[13]	GPIO	I/O	V _{DDE1}	SH	—	—	—	108	L13

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
PE14	78	PE[14]	GPIO	I/O	V _{DDE1}	SH	—	—	—	102	L16
PE15	79	PE[15]	GPIO	I/O	V _{DDE1}	SH	—	—	—	99	M14
Port F (16) (Section/Page: 2.7.6/2-25)											
PF0	80	PF[0] RD_WR EVTI ⁸	GPIO EBI Read/Write Nexus Event In	I/O I/O I	V _{DDE3}	MH	—	—	66	82	N12
PF1	81	PF[1] TA MLBCLK EVTO ⁸	GPIO EBI Transfer Acknowledge MLB Clock Nexus Event Out	I/O I/O I O	V _{DDE3}	MH	—	—	65	81	P12
PF2	82	PF[2] AD[8] ADDR[8] MLBSI / MLBSIG MSEO ⁸	GPIO EBI Muxed Address/Data EBI Non Muxed Address MLB Signal In (5-pin) / MLB Bidirectional Signal (3-pin) Nexus Message Start/End Out	I/O I/O O I I/O O	V _{DDE3}	MH	—	—	64	80	R12
PF3	83	PF[3] AD[9] ADDR[9] MLBDI / MLBDAT MCKO ⁸	GPIO EBI Muxed Address/Data EBI Non Muxed Address MLB Data In (5-pin) / MLB Bidirectional Data (3-pin) Nexus Message Clock Out	I/O I/O O I I/O O	V _{DDE3}	MH	—	—	63	79	T12
PF4	84	PF[4] AD[10] ADDR[10] MLBSO / MLBSIG_BUFEN MDO[0] ⁸	GPIO EBI Muxed Address/Data EBI Non Muxed Address MLB Signal Out (5-pin) / MLB Signal Level Shifter Enable (3-pin) Nexus Message Data Out	I/O I/O O O O O O	V _{DDE3}	MH	—	—	59	74	T10
PF5	85	PF[5] AD[11] ADDR[11] MLBDO / MLBDAT_BUFEN MDO[1] ⁸	GPIO EBI Muxed Address/Data EBI Non Muxed Address MLB Data Out (5-pin) / MLB Data Level Shifter Enable (3-pin) Nexus Message Data Out	I/O I/O O O O O O	V _{DDE3}	MH	—	—	58	72	R9
PF6	86	PF[6] AD[12] ADDR[12] MLB_SLOT / MLB_SIGOBS / MLB_DATOBS MDO[2] ⁸	GPIO EBI Muxed Address/Data EBI Non Muxed Address MLB Slot Debug / MLB Clock Adjust Observe Signal / MLB Clock Adjust Observe Data Nexus Message Data Out	I/O I/O O O O O O O	V _{DDE3}	MH	—	—	57	68	T8
PF7	87	PF[7] AD[13] ADDR[13] MDO[3] ⁸	GPIO EBI Muxed Address/Data EBI Non Muxed Address Nexus Message Data Out	I/O I/O O O	V _{DDE3}	MH	—	—	56	66	P8
PF8	88	PF[8] AD[14] ADDR[14] MDO[4] ⁸	GPIO EBI Muxed Address/Data EBI Non Muxed Address Nexus Message Data Out	I/O I/O O O	V _{DDE2}	MH	—	—	55	65	N8

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
PF9	89	PF[9] AD[15] ADDR[15] MDO[5] ⁸	GPIO EBI Muxed Address/Data EBI Non Muxed Address Nexus Message Data Out	I/O I/O O O	V _{DDE2}	MH	—	—	54	64	T7
PF10	90	PF[10] CS[1] TXD_C MDO[6] ⁸	GPIO EBI Chip Select SCI_C Transmit Nexus Message Data Out	I/O O O O	V _{DDE2}	MH	—	—	52	62	R7
PF11	91	PF[11] CS[0] RXD_C MDO[7] ⁸	GPIO EBI Chip Select SCI_C Receive Nexus Message Data Out	I/O O I O	V _{DDE2}	MH	—	—	51	61	P7
PF12	92	PF[12] TS TXD_D ALE	GPIO EBI Transfer Start SCI_D Transmit EBI Address Latch Enable	I/O I/O O O	V _{DDE2}	MH	—	—	50	60	N7
PF13	93	PF[13] OE RXD_D	GPIO EBI Output Enable SCI_D Receive	I/O O I	V _{DDE2}	MH	—	—	49	59	R6
PF14	94	PF[14] WE[0] BDIP CNTX_D	GPIO EBI Write Enable EBI Burst Data In Progress CAN_D Transmit	I/O O O O	V _{DDE2}	MH	—	—	45	55	P6
PF15	95	PF[15] WE[1] TEA CNRX_D	GPIO EBI Write Enable EBI Transfer Error Acknowledge CAN_D Receive	I/O O I/O I	V _{DDE2}	MH	—	—	44	54	N6
Port G (16) (Section/Page: 2.7.7/2-28)											
PG0	96	PG[0] AD[16] eMIOS[16]	GPIO EBI Muxed Address/Data eMIOS Channel	I/O I/O I/O	V _{DDE2}	MH	—	—	43	51	P5
PG1	97	PG[1] AD[17] eMIOS[17] SIN_C	GPIO EBI Muxed Address/Data eMIOS Channel DSPI_C Serial In	I/O I/O I/O I	V _{DDE2}	MH	—	—	42	50	T4
PG2	98	PG[2] AD[18] eMIOS[18] SOUT_C	GPIO EBI Muxed Address/Data eMIOS Channel DSPI_C Serial Out	I/O I/O I/O O	V _{DDE2}	MH	—	—	41	49	R4
PG3	99	PG[3] AD[19] eMIOS[19] SCK_C	GPIO EBI Muxed Address/Data eMIOS Channel DSPI_C Serial Clock	I/O I/O I/O I/O	V _{DDE2}	MH	—	—	40	48	P4
PG4	100	PG[4] AD[20] eMIOS[20] PCS_C[0]	GPIO EBI Muxed Address/Data eMIOS Channel DSPI_C Peripheral Chip Select	I/O I/O I/O I/O	V _{DDE2}	MH	—	—	39	47	T3

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
PG5	101	PG[5] AD[21] eMIOS[21]	GPIO EBI Muxed Address/Data eMIOS Channel	I/O I/O I/O	V _{DDE2}	MH	—	—	38	46	R3
PG6	102	PG[6] AD[22] eMIOS[22]	GPIO EBI Muxed Address/Data eMIOS Channel	I/O I/O I/O	V _{DDE2}	MH	—	—	37	45	T2
PG7	103	PG[7] AD[23] eMIOS[23] RXD_C	GPIO EBI Muxed Address/Data eMIOS Channel SCI_C Receive	I/O I/O I/O I	V _{DDE2}	MH	—	—	36	44	R1
PG8	104	PG[8] AD[24] PCS_A[4]	GPIO EBI Muxed Address/Data DSPI_A Peripheral Chip Select	I/O I/O O	V _{DDE2}	MH	—	—	35	43	P2
PG9	105	PG[9] AD[25] PCS_A[3] TXD_C	GPIO EBI Muxed Address/Data DSPI_A Peripheral Chip Select SCI_C Transmit	I/O I/O O O	V _{DDE2}	MH	—	—	34	42	N3
PG10	106	PG[10] AD[26] PCS_A[2]	GPIO EBI Muxed Address/Data DSPI_A Peripheral Chip Select	I/O I/O O	V _{DDE2}	MH	—	—	30	38	N2
PG11	107	PG[11] AD[27] PCS_A[1]	GPIO EBI Muxed Address/Data DSPI_A Peripheral Chip Select	I/O I/O O	V _{DDE2}	MH	—	—	29	37	N1
PG12	108	PG[12] AD[28] PCS_A[0]	GPIO EBI Muxed Address/Data DSPI_A Peripheral Chip Select	I/O I/O I/O	V _{DDE2}	MH	—	—	28	36	M4
PG13	109	PG[13] AD[29] SCK_A	GPIO EBI Muxed Address/Data DSPI_A Clock	I/O I/O I/O	V _{DDE2}	MH	—	—	27	35	M3
PG14	110	PG[14] AD[30] SOUT_A	GPIO EBI Muxed Address/Data DSPI_A Data Out	I/O I/O O	V _{DDE2}	MH	—	—	26	34	M2
PG15	111	PG[15] AD[31] SIN_A	GPIO EBI Muxed Address/Data DSPI_A Data In	I/O I/O I	V _{DDE2}	MH	—	—	25	33	M1
Port H (16) (Section/Page: 2.7.8/2-30)											
PH0	112	PH[0] AN[27] eMIOS[20] SCL_A	GPIO eQADC Analog Input ⁶ eMIOS Channel I ² C_A Serial Clock	I/O I O I/O	V _{DDE2}	A + SH	—	—	24	32	L3
PH1	113	PH[1] AN[26] eMIOS[21] SDA_A	GPIO eQADC Analog Input ⁶ eMIOS Channel I ² C_A Serial Data	I/O I O I/O	V _{DDE2}	A + SH	—	—	23	31	L2

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
PH2	114	PH[2] AN[25] eMIOS[22] CS[3]	GPIO eQADC Analog Input ⁶ eMIOS Channel EBI Chip Select	I/O I O O	V _{DDE2}	A + MH	—	—	22	30	L1
PH3	115	PH[3] AN[24] eMIOS[23] CS[2]	GPIO eQADC Analog Input ⁶ eMIOS Channel EBI Chip Select	I/O I O O	V _{DDE2}	A + MH	—	—	21	29	K4
PH4	116	PH[4] AN[23] TXD_E MA[2]	GPIO eQADC Analog Input ⁶ SCI_E Transmit eQADC External Mux Address	I/O I O O	V _{DDE2}	A + SH	—	—	20	28	K3
PH5	117	PH[5] AN[22] RXD_E MA[1]	GPIO eQADC Analog Input ⁶ SCI_E Receive eQADC External Mux Address	I/O I I O	V _{DDE2}	A + SH	—	—	19	24	J3
PH6	118	PH[6] AN[21] TXD_F	GPIO eQADC Analog Input ⁶ SCI_F Transmit	I/O I O	V _{DDE2}	A + SH	—	—	18	23	J2
PH7	119	PH[7] AN[20] RXD_F	GPIO eQADC Analog Input ⁶ SCI_F Receive	I/O I I	V _{DDE2}	A + SH	—	—	17	22	J1
PH8	120	PH[8] AN[19] CNTX_E MA[0]	GPIO eQADC Analog Input ⁶ CAN_E Transmit eQADC External Mux Address	I/O I O O	V _{DDE2}	A + SH	—	—	14	17	H1
PH9	121	PH[9] AN[18]/ANT CNRX_E	GPIO eQADC Analog Input ⁶ CAN_E Receive	I/O I I	V _{DDE2}	A + SH	—	—	13	14	G2
PH10	122	PH[10] AN[17]/ANS CNRX_F	GPIO eQADC Analog Input ⁶ CAN_F Receive	I/O I I	V _{DDE2}	A + SH	—	—	12	12	F4
PH11	123	PH[11] AN[16]/ANR CNTX_F	GPIO eQADC Analog Input ⁶ CAN_F Transmit	I/O I O	V _{DDE2}	A + SH	—	—	11	11	F3
PH12	124	PH[12] PCS_D[5]	GPIO DSPI_D Peripheral Chip Select	I/O O	V _{DDE2}	SH	—	—	—	—	F2
PH13	125	PH[13]	GPIO	I/O	V _{DDE2}	SH	—	—	—	—	F1
PH14	126	PH[14] WE[2]	GPIO EBI Write Enable	I/O O	V _{DDE2}	MH	—	—	—	53	T5
PH15	127	PH[15] WE[3]	GPIO EBI Write Enable	I/O O	V _{DDE2}	MH	—	—	—	52	R5
Port J (16) (Section/Page: 2.7.9/2-32)											
PJ0	128	PJ[0] AD[0]	GPIO EBI Muxed Address/Data	I/O I/O	V _{DDE3}	MH	—	—	—	—	N11

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
PJ1	129	PJ[1] AD[1]	GPIO EBI Muxed Address/Data	I/O I/O	V _{DDE3}	MH	—	—	—	—	P11
PJ2	130	PJ[2] AD[2]	GPIO EBI Muxed Address/Data	I/O I/O	V _{DDE3}	MH	—	—	—	—	N10
PJ3	131	PJ[3] AD[3]	GPIO EBI Muxed Address/Data	I/O I/O	V _{DDE3}	MH	—	—	—	—	R10
PJ4	132	PJ[4] AD[4]	GPIO EBI Muxed Address/Data	I/O I/O	V _{DDE3}	MH	—	—	—	75	P10
PJ5	133	PJ[5] AD[5]	GPIO EBI Muxed Address/Data	I/O I/O	V _{DDE3}	MH	—	—	—	73	T9
PJ6	134	PJ[6] AD[6]	GPIO EBI Muxed Address/Data	I/O I/O	V _{DDE3}	MH	—	—	—	69	P9
PJ7	135	PJ[7] AD[7]	GPIO EBI Muxed Address/Data	I/O I/O	V _{DDE3}	MH	—	—	—	67	R8
PJ8	136	PJ[8] PCS_D[4]	GPIO DSPI_D Peripheral Chip Select	I/O I/O	V _{DDE2}	SH	—	—	—	27	K2
PJ9	137	PJ[9] PCS_D[3]	GPIO DSPI_D Peripheral Chip Select	I/O I/O	V _{DDE2}	SH	—	—	—	26	K1
PJ10	138	PJ[10] PCS_D[2]	GPIO DSPI_D Peripheral Chip Select	I/O I/O	V _{DDE2}	SH	—	—	—	25	J4
PJ11	139	PJ[11] PCS_D[1]	GPIO DSPI_D Peripheral Chip Select	I/O I/O	V _{DDE2}	SH	—	—	—	19	H3
PJ12	140	PJ[12] PCS_D[0]	GPIO DSPI_D Peripheral Chip Select	I/O I/O	V _{DDE2}	SH	—	—	—	18	H2
PJ13	141	PJ[13] SCK_D	GPIO DSPI_D Clock	I/O I/O	V _{DDE2}	SH	—	—	—	16	G4
PJ14	142	PJ[14] SOUT_D	GPIO DSPI_D Serial Out	I/O O	V _{DDE2}	SH	—	—	—	15	G3
PJ15	143	PJ[15] SIN_D	GPIO DSPI_D Serial In	I/O I	V _{DDE2}	SH	—	—	—	13	G1
Port K (2) (Section/Page: 2.7.10/2-33)											
PK0	144	PK[0] EXTAL32	GPIO 32 kHz Crystal Oscillator Input	I I	V _{DDA}	AE + IH	—	—	—	168	B6
PK1	145	PK[1] XTAL32	GPIO 32 kHz Crystal Oscillator Output	I O	V _{DDA}	AE + IH	—	—	—	166	A6
Miscellaneous Pins (9) (Section/Page: 2.7.11/2-33)											
EXTAL	—	EXTAL EXTCLK	Main Crystal Oscillator Input External Clock Input	I I	V _{DDSYN}	AE	EXTAL		75	91	N16
XTAL	—	XTAL	Main Crystal Oscillator Output	O	V _{DDSYN}	AE	XTAL		74	90	P16
TMS	—	TMS	JTAG Test Mode Select Input	I	V _{DDE3}	SH	TMS (Pull Up)		72	88	T15
TCK	—	TCK	JTAG Test Clock Input	I	V _{DDE3}	IH	TCK (Pull Down)		71	87	R14

Table 2-1. MPC5510 Signal Properties (continued)

Pin Name	GPIO (PCR) Num ¹	Supported Functions ²	Description	I/O Type	Voltage ³	Pad Type	Status During Reset ⁴	Status After Reset ⁴	Package Pin Locations		
									144	176	208
TDO	—	TDO	JTAG Test Data Output	O	V _{DDE3}	MH	TDO (Pull Up ⁹)	70	86	T14	
TDI	—	TDI	JTAG Test Data Input	I	V _{DDE3}	IH	TDI (Pull Up)	69	85	R13	
JCOMP	—	JCOMP	JTAG Compliancy	I	V _{DDE3}	IH	JCOMP (Pull Down)	68	84	T13	
TEST	—	TEST	Test Mode Select	I	V _{DDE3}	IH	TEST	62	78	R11	
$\overline{\text{RESET}}$	—	$\overline{\text{RESET}}$	External Reset	I/O	V _{DDE2}	SH	$\overline{\text{RESET}}$ (Pull Up)	10	10	E4	

¹ The GPIO number is the same as the corresponding pad configuration register (SIU_PCR n) number.

² This column lists the functions associated with the programming of the SIU_PCR n [PA] bit field in the following order: GPIO, function 1, function 2, and function 3. The unused functions by a given pin begin with function 3, then function 2, then function 1 (see Figure 2-1).

³ These are nominal voltages. Each segment provides the power and ground for the given set of I/O pins.

⁴ A dash for the function in this column denotes the input and output buffer are turned off.

⁵ Port A[14:15]—EXTAL32 and XTAL32 functions only apply on the 144LQFP. These functions are on PortK[0:1] for the 176LQFP and 208BGA.

⁶ This analog input pin has reduced analog-to-digital conversion accuracy compared to PA0–PA15. See the *MPC5510 Microcontroller Family Data Sheet* for values.

⁷ BOOTCFG is the pin function while the RESET pin is asserted. When the RESET pin is negated, the pin function is controlled by the associated PCR register.

⁸ The NEXUS function is selected when the JTAG TAP controller is enabled via the JCOMP pin. The value of the PA field in the associated PCR register has no effect on the pin function when the NEXUS function is selected.

⁹ Pullup is enabled only when JCOMP is negated.

2.3 Power and Ground Supply Summary

Refer to [Section 2.7.12, “Power and Ground Pins,”](#) for detailed descriptions of these pins.

Table 2-2. MPC5510 Power/Ground

Pin Name	Function Description	Voltage ¹	Package Pin Locations		
			144	176	208
V _{DDR}	Voltage Regulator Supply	5.0 V	46	56	T6
V _{DDA}	Analog Power	5.0 V	144	176	A2
V _{RH} ²	eQADC Voltage Reference High	5.0 V			B3
V _{SSA}	Analog Ground	—	141	173	A4
V _{RL} ³	eQADC Voltage Reference Low	—			B4
REFBYPC	eQADC Reference Bypass Capacitor	V _{SSA}	1	1	B1
V _{PP} ⁴	Flash Program/Erase Power	5.0 V	78	94	P15
V _{DDSYN}	Clock Synthesizer Power	3.3 V	73	89	R16
V _{SSSYN}	Clock Synthesizer Ground	—	76	92	M16

Table 2-2. MPC5510 Power/Ground (continued)

Pin Name	Function Description	Voltage ¹	Package Pin Locations		
			144	176	208
V _{DDE1}	External I/O Power	3.3 V – 5.0 V	96,119	105,120, 143,155	A15,D10,E13, G16,K15
V _{DDE2}			16,33,48	21,41,58	H4,L4,N5,P1
V _{DDE3}			61	71,77	N9,T11
V _{SSE1}	External I/O Ground	–	95,118	104,119, 142,154	Shorted to V _{SS} in the package
V _{SSE2}			32,47	20,40,57	Shorted to V _{SS} in the package
V _{SSE3}			60	70,76	Shorted to V _{SS} in the package
V _{DD33}	3.3 V I/O Power	3.3 V	77	93	N15
V _{F:ASH} ⁵	Flash Read Power				
V _{DD}	Internal Logic Power	1.5 V	31,53,79	39,63,95	A1,A16,B2,B15, R2,R15,T1,T16
V _{DDF}	Flash Internal Logic Power				79
V _{SS}	Ground	–	80	96	C3,C14,D4,D13, G7-G10,H7-H10, J7-J10,K7-K10, N4,N13,P3,P14
V _{SSF}	Flash Internal Logic Ground				Shorted to V _{SS} in the package

¹ These are nominal voltages.

² V_{RH} is shorted to V_{DDA} in the 144LQFP and 176LQFP packages.

³ V_{RL} is shorted to V_{SSA} in the 144LQFP and 176LQFP packages.

⁴ V_{PP} requires nominal 5V for program/erase operations, but may be 0-5V otherwise.

⁵ V_{FLASH} is shorted to V_{DD33} in the package.

2.4 Pinout – 144 LQFP

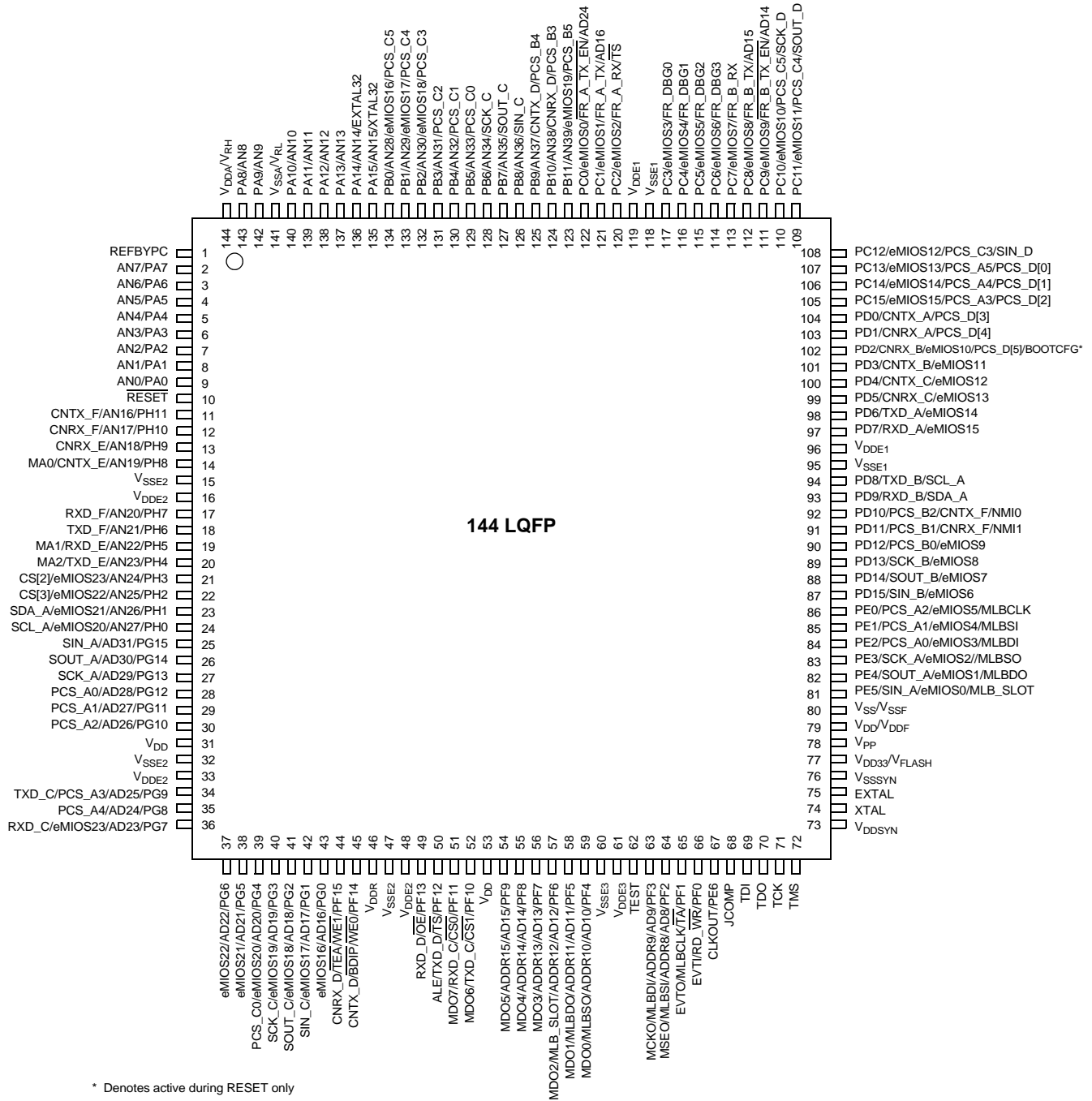


Figure 2-2. MPC5510 Pinout – 144 LQFP

2.5 Pinout – 176 LQFP

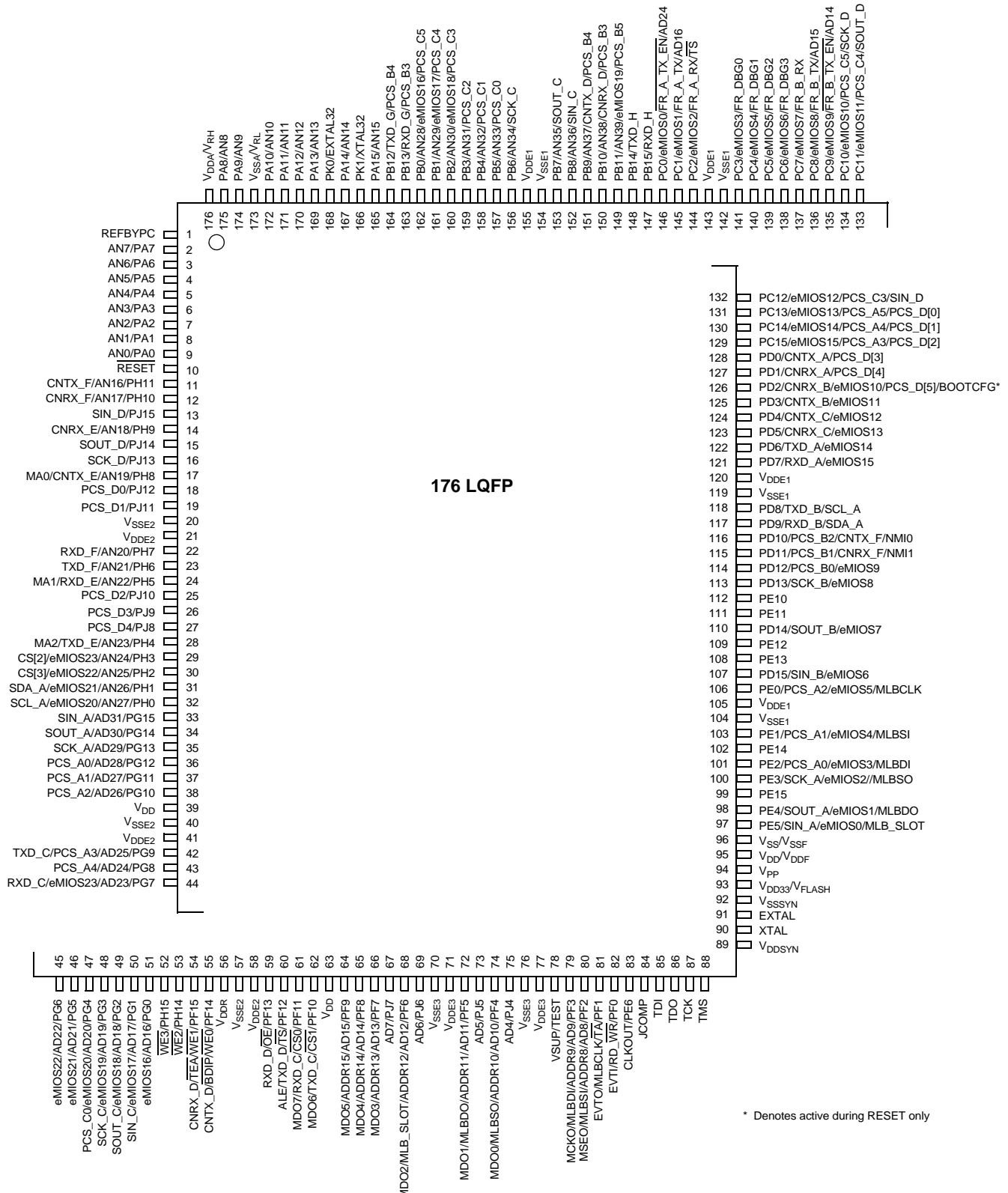


Figure 2-3. MPC5510 Pinout – 176 LQFP

2.6 Pinout – 208 BGA

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16																	
A	V _{DD}	V _{DDA}	PA8	V _{SSA}	PA13	PK1	PB12	PB2	PB6	PB10	PB15	PC3	PC7	PC10	V _{DDE1}	V _{DD}	A																
B	REF BYPC	V _{DD}	V _{RH}	V _{RL}	PA12	PK0	PB13	PB3	PB7	PB11	PC0	PC4	PC8	PC11	V _{DD}	PC12	B																
C	PA7	PA6	V _{SS}	PA9	PA11	PA15	PB0	PB4	PB8	PB14	PC1	PC5	PC9	V _{SS}	PC13	PC14	C																
D	PA5	PA4	PA3	V _{SS}	PA10	PA14	PB1	PB5	PB9	V _{DDE1}	PC2	PC6	V _{SS}	PC15	PD0	PD1	D																
E	PA2	PA1	PA0	RESET	<div style="text-align: center;"> 208 PBGA Ball Map (as viewed from top through the package) <table border="1" style="margin: 10px auto; width: 150px; height: 100px;"> <tr><td>V_{SS}</td><td>V_{SS}</td><td>V_{SS}</td><td>V_{SS}</td></tr> <tr><td>V_{SS}</td><td>V_{SS}</td><td>V_{SS}</td><td>V_{SS}</td></tr> <tr><td>V_{SS}</td><td>V_{SS}</td><td>V_{SS}</td><td>V_{SS}</td></tr> <tr><td>V_{SS}</td><td>V_{SS}</td><td>V_{SS}</td><td>V_{SS}</td></tr> </table> </div>								V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{SS}	V _{DDE1}	PD2	PD3	PD4	E
V _{SS}	V _{SS}	V _{SS}	V _{SS}																														
V _{SS}	V _{SS}	V _{SS}	V _{SS}																														
V _{SS}	V _{SS}	V _{SS}	V _{SS}																														
V _{SS}	V _{SS}	V _{SS}	V _{SS}																														
F	PH13	PH12	PH11	PH10									PD5	PD6	PD7	PD9	F																
G	PJ15	PH9	PJ14	PJ13									PD8	PD10	PD11	V _{DDE1}	G																
H	PH8	PJ12	PJ11	V _{DDE2}									PE7	PD12	PD13	PE8	H																
J	PH7	PH6	PH5	PJ10									PE9	PD14	PE11	PE10	J																
K	PJ9	PJ8	PH4	PH3									PE12	PD15	V _{DDE1}	PE0	K																
L	PH2	PH1	PH0	V _{DDE2}									PE13	PE1	PE2	PE14	L																
M	PG15	PG14	PG13	PG12									PE3	PE15	PE5	V _{SSSYN}	M																
N	PG11	PG10	PG9	V _{SS}	V _{DDE2}	PF15	PF12	PF8	V _{DDE3}	PJ2	PJ0	PF0	V _{SS}	PE4	V _{DD33}	EXTAL	N																
P	V _{DDE2}	PG8	V _{SS}	PG3	PG0	PF14	PF11	PF7	PJ6	PJ4	PJ1	PF1	PE6	V _{SS}	V _{PP}	XTAL	P																
R	PG7	V _{DD}	PG5	PG2	PH15	PF13	PF10	PJ7	PF5	PJ3	TEST	PF2	TDI	TCK	V _{DD}	V _{DDSYN}	R																
T	V _{DD}	PG6	PG4	PG1	PH14	V _{DDR}	PF9	PF6	PJ5	PF4	V _{DDE3}	PF3	JCOMP	TDO	TMS	V _{DD}	T																

Figure 2-4. MPC5510 Pinout – 208 PBGA

2.7 Detailed External Signal Descriptions

2.7.1 Port A Pins

2.7.1.1 PA0 to PA13 — GPI (PA[0:13]) / Analog Input (AN[0] – AN[13])

PA[0:13] are general-purpose input (GPI) pins. AN[0] to AN[13] are single-ended analog input pins.

2.7.1.2 PA14 — GPI (PA[14]) / Analog Input (AN[14]) / 32 kHz Crystal Input (EXTAL32)

PA[14] is a general-purpose input (GPI) pin. AN[14] is a single-ended analog input pin. EXTAL32 is the input pin for an external 32 kHz crystal oscillator (EXTAL32 function available on PA[14] pin on the 144LQFP package and PK[0] pin on the 176LQFP and 208BGA packages).

2.7.1.3 PA15 — GPI (PA[15]) / Analog Input (AN[15]) / 32 kHz Crystal Output (XTAL32)

PA[15] is a GPI pin. AN[15] is a single-ended analog input pin. XTAL32 is the output pin for an external 32 kHz crystal oscillator (XTAL32 function available on PA[15] pin on the 144LQFP package and PK[1] pin on the 176LQFP and 208BGA packages).

2.7.2 Port B Pins

2.7.2.1 PB0 — GPIO (PB[0]) / Analog Input (AN[28]) / eMIOS Channel (eMIOS[16]) / DSPI_C Peripheral Chip Select (PCS_C[5])

PB[0] is a GPIO pin. AN[28] is a single-ended analog input pin. eMIOS[16] is an output-only channel pin for the eMIOS200 module. PCS_C[5] is a peripheral chip select output pin for the DSPI C module.

2.7.2.2 PB1 — GPIO (PB[1]) / Analog Input (AN[29]) / eMIOS Channel (eMIOS[17]) / DSPI_C Peripheral Chip Select (PCS_C[4])

PB[1] is a GPIO pin. AN[29] is a single-ended analog input pin. eMIOS[17] is an output-only channel pin for the eMIOS200 module. PCS_C[4] is a peripheral chip select output pin for the DSPI C module.

2.7.2.3 PB2 — GPIO (PB[2]) / Analog Input (AN[30]) / eMIOS Channel (eMIOS[18]) / DSPI_C Peripheral Chip Select (PCS_C[3])

PB[2] is a GPIO pin. AN[30] is a single-ended analog input pin. eMIOS[18] is an output-only channel pin for the eMIOS200 module. PCS_C[3] is a peripheral chip select output pin for the DSPI C module.

2.7.2.4 PB3 — GPIO (PB[3]) / Analog Input (AN[31]) / DSPI_C Peripheral Chip Select (PCS_C[2])

PB[3] is a GPIO pin. AN[31] is a single-ended analog input pin. PCS_C[2] is a peripheral chip select output pin for the DSPI C module.

2.7.2.5 PB4 — GPIO (PB[4]) / Analog Input (AN[32]) / DSPI_C Peripheral Chip Select (PCS_C[1])

PB[4] is a GPIO pin. AN[32] is a single-ended analog input pin. PCS_C[1] is a peripheral chip select output pin for the DSPI C module.

2.7.2.6 PB5 — GPIO (PB[5]) / Analog Input (AN[33]) / DSPI_C Peripheral Chip Select (PCS_C[0])

PB[5] is a GPIO pin. AN[33] is a single-ended analog input pin. PCS_C[0] is a peripheral chip select output pin for the DSPI C module.

2.7.2.7 PB6 — GPIO (PB[6]) / Analog Input (AN[34]) / DSPI_C Clock (SCK_C)

PB[6] is a GPIO pin. AN[34] is a single-ended analog input pin. SCK_C is the SPI clock pin for the DSPI C module.

2.7.2.8 PB7 — GPIO (PB[7]) / Analog Input (AN[35]) / DSPI_C Data Output (SOUT_C)

PB[7] is a GPIO pin. AN[35] is a single-ended analog input pin. SOUT_C is the data output pin for the DSPI C module.

2.7.2.9 PB8 — GPIO (PB[8]) / Analog Input (AN[36]) / DSPI_C Data Input (SIN_C)

PB[8] is a GPIO pin. AN[36] is a single-ended analog input pin. SIN_C is the data input pin for the DSPI C module.

2.7.2.10 PB9 — GPIO (PB[9]) / Analog Input (AN[37]) / CAN_D Transmit (CNTX_D) / DSPI_B Peripheral Chip Select (PCS_B[4])

PB[9] is a GPIO pin. AN[37] is a single-ended analog input pin. CNTX_D is the transmit pin for the FlexCan D module. PCS_B[4] is a peripheral chip select output pin for the DSPI B module.

2.7.2.11 PB10 — GPIO (PB[10]) / Analog Input (AN[38]) / CAN_D Receive (CNRX_D) / DSPI_B Peripheral Chip Select (PCS_B[3])

PB[10] is a GPIO pin. AN[38] is a single-ended analog input pin. CNRX_D is the receive pin for the FlexCan D module. PCS_B[3] is a peripheral chip select output pin for the DSPI B module.

2.7.2.12 PB11 — GPIO (PB[11]) / Analog Input (AN[39]) / eMIOS Channel (eMIOS[19]) / DSPI_B Peripheral Chip Select (PCS_B[5])

PB[11] is a GPIO pin. AN[39] is a single-ended analog input pin. eMIOS[19] is an output-only channel pin for the eMIOS200 module. PCS_B[5] is a peripheral chip select output pin for the DSPI B module.

2.7.2.13 PB12 — GPIO (PB[12]) / SCI_G Transmit (TXD_G) / DSPI_B Peripheral Chip Select (PCS_B[4])

PB[12] is a GPIO pin. TXD_G is the transmit pin for the eSCI G module. PCS_B[4] is a peripheral chip select output pin for the DSPI B module.

2.7.2.14 PB13 — GPIO (PB[13]) / SCI_G Receive (RXD_G) / DSPI_B Peripheral Chip Select (PCS_B[3])

PB[13] is a GPIO pin. RXD_G is the receive pin for the eSCI G module. PCS_B[3] is a peripheral chip select output pin for the DSPI B module.

2.7.2.15 PB14 — GPIO (PB[14]) / SCI_H Transmit (TXD_H)

PB[14] is a GPIO pin. TXD_H is the transmit pin for the eSCI H module.

2.7.2.16 PB15 — GPIO (PB[15]) / SCI_H Receive (RXD_H)

PB[15] is a GPIO pin. RXD_H is the receive pin for the eSCI H module.

2.7.3 Port C Pins

2.7.3.1 PC0 — GPIO (PC[0]) / eMIOS Channel (eMIOS[0]) / FlexRay Channel A Transmit Enable (FR_A_TX_EN) / EBI Multiplexed Address/Data (AD[24])

PC[0] is a GPIO pin. eMIOS[0] is an input/output channel pin for the eMIOS200 module. $\overline{\text{FR_A_TX_EN}}$ in the FlexRay Channel A transmit enable pin. AD[24] is the external bus interface (EBI) multiplexed address and data bus.

2.7.3.2 PC1 — GPIO (PC[1]) / eMIOS Channel (eMIOS[1]) / FlexRay Channel A Transmit (FR_A_TX) / EBI Multiplexed Address/Data (AD[16])

PC[1] is a GPIO pin. eMIOS[1] is an input/output channel pin for the eMIOS200 module. FR_A_TX in the FlexRay Channel A transmit pin. AD[16] is the EBI multiplexed address and data bus

2.7.3.3 PC2 — GPIO (PC[2]) / eMIOS Channel (eMIOS[2]) / FlexRay Channel A Receive (FR_A_RX) / EBI Transfer Start ($\overline{\text{TS}}$)

PC[2] is a GPIO pin. eMIOS[2] is an input/output channel pin for the eMIOS200 module. FR_A_RX in the FlexRay Channel A receive pin. $\overline{\text{TS}}$ is the EBI transfer start signal.

2.7.3.4 PC3 — GPIO (PC[3]) / eMIOS Channel (eMIOS[3]) / FlexRay Debug 0 (FR_DBG0)

PC[3] is a GPIO pin. eMIOS[3] is an input/output channel pin for the eMIOS200 module. FR_DBG0 is one of the FlexRay debug port pins.

2.7.3.5 PC4 — GPIO (PC[4]) / eMIOS Channel (eMIOS[4]) / FlexRay Debug 1 (FR_DBG1)

PC[4] is a GPIO pin. eMIOS[4] is an input/output channel pin for the eMIOS200 module. FR_DBG1 is one of the FlexRay debug port pins.

2.7.3.6 PC5 — GPIO (PC[5]) / eMIOS Channel (eMIOS[5]) / FlexRay Debug 2 (FR_DBG2)

PC[5] is a GPIO pin. eMIOS[5] is an input/output channel pin for the eMIOS200 module. FR_DBG2 is one of the FlexRay debug port pins.

2.7.3.7 PC6 — GPIO (PC[6]) / eMIOS Channel (eMIOS[6]) / FlexRay Debug 3 (FR_DBG3)

PC[6] is a GPIO pin. eMIOS[6] is an input/output channel pin for the eMIOS200 module. FR_DBG3 is one of the FlexRay debug port pins.

2.7.3.8 PC7 — GPIO (PC[7]) / eMIOS Channel (eMIOS[7]) / FlexRay Channel B Receive (FR_B_RX)

PC[7] is a GPIO pin. eMIOS[7] is an input/output channel pin for the eMIOS200 module. FR_B_RX is the FlexRay Channel B receive pin.

2.7.3.9 PC8 — GPIO (PC[8]) / eMIOS Channel (eMIOS[8]) / FlexRay Channel B Transmit (FR_B_TX) / Multiplexed Address/Data (AD[15])

PC[8] is a GPIO pin. eMIOS[8] is an input/output channel pin for the eMIOS200 module. FR_B_TX is the FlexRay Channel B transmit pin. AD[15] is the EBI multiplexed address and data bus.

2.7.3.10 PC9 — GPIO (PC[9]) / eMIOS Channel (eMIOS[9]) / FlexRay Channel B Transmit Enable (FR_B_TX_EN) / Multiplexed Address/Data (AD[14])

PC[9] is a GPIO pin. eMIOS[9] is an input/output channel pin for the eMIOS200 module. FR_B_TX_EN is the FlexRay Channel B transmit enable pin. AD[14] is the EBI multiplexed address and data bus.

2.7.3.11 PC10 — GPIO (PC[10]) / eMIOS Channel (eMIOS[10]) / DSPI_C Peripheral Chip Select (PCS_C[5]) / DSPI_D Clock (SCK_D)

PC[10] is a GPIO pin. eMIOS[10] is an input/output channel pin for the eMIOS200 module. PCS_C[5] is a peripheral chip select output pin for the DSPI C module. SCK_D is the SPI clock pin of the DSPI_D module.

2.7.3.12 PC11 — GPIO (PC[11]) / eMIOS Channel (eMIOS[11]) / DSPI_C Peripheral Chip Select (PCS_C[4]) / DSPI_D Serial Data Out (SOUT_D)

PC[11] is a GPIO pin. eMIOS[11] is an input/output channel pin for the eMIOS200 module. PCS_C[4] is a peripheral chip select output pin for the DSPI C module. SOUT_D is the serial data output from the DSPI_D module.

2.7.3.13 PC12 — GPIO (PC[12]) / eMIOS Channel (eMIOS[12]) / DSPI_C Peripheral Chip Select (PCS_C[3]) / DSPI_D Serial Data Input (SIN_D)

PC[12] is a GPIO pin. eMIOS[12] is an input/output channel pin for the eMIOS200 module. PCS_C[3] is a peripheral chip select output pin for the DSPI C module. SIN_D is the serial data input for the DSPI_D module.

2.7.3.14 PC13 — GPIO (PC[13]) / eMIOS Channel (eMIOS[13]) / DSPI_A Peripheral Chip Select (PCS_A[5]) / DSPI_D Peripheral Chip Select (PCS_D[0])

PC[13] is a GPIO pin. eMIOS[13] is an input/output channel pin for the eMIOS200 module. PCS_A[5] is a peripheral chip select output pin for the DSPI A module. PCS_D[0] is a peripheral chip select output pin for the DSPI_D module.

2.7.3.15 PC14 — GPIO (PC[14]) / eMIOS Channel (eMIOS[14]) / DSPI_A Peripheral Chip Select (PCS_A[4]) / DSPI_D Peripheral Chip Select (PCS_D[1])

PC[14] is a GPIO pin. eMIOS[14] is an input/output channel pin for the eMIOS200 module. PCS_A[4] is a peripheral chip select output pin for the DSPI A module. PCS_D[1] is a peripheral chip select output pin for the DSPI D module.

2.7.3.16 PC15 — GPIO (PC[15]) / eMIOS Channel (eMIOS[15]) / DSPI_A Peripheral Chip Select (PCS_A[3]) / DSPI_D Peripheral Chip Select (PCS_D[2])

PC[15] is a GPIO pin. eMIOS[15] is an input/output channel pin for the eMIOS200 module. PCS_A[3] is a peripheral chip select output pin for the DSPI A module. PCS_D[2] is a peripheral chip select output pin for the DSPI D module.

2.7.4 Port D Pins

2.7.4.1 PD0 — GPIO (PD[0]) / CAN_A Transmit (CNTX_A) / DSPI_D Peripheral Chip Select (PCS_D[3])

PD[0] is a GPIO pin. CNTX_A is the transmit pin for the FlexCan A module. PCS_D[3] is a peripheral chip select for the DSPI_D module.

2.7.4.2 PD1 — GPIO (PD[1]) / CAN_A Receive (CNRX_A) / DSPI_D Peripheral Chip Select (PCS_D[4])

PD[1] is a GPIO pin. CNRX_A is the receive pin for the FlexCan A module. PCS_D[4] is a peripheral chip select for the DSPI_D module.

2.7.4.3 PD2 — GPIO (PD[2]) / CAN_B Receive (CNRX_B) / eMIOS Channel (eMIOS[10]) / Boot Configuration (BOOTCFG) / DSPI_D Peripheral Chip Select (PCS_D[5])

PD[2] is a GPIO pin. CNRX_B is the receive pin for the FlexCan B module. eMIOS[10] is an output-only channel pin for the eMIOS200 module. The BOOTCFG pin is sampled before the negation of the RESET pin. The value is used by the BAM program to determine the boot configuration. PCS_D[5] is a peripheral chip select output pin for the DSPI_D module.

2.7.4.4 PD3 — GPIO (PD[3]) / CAN_B Transmit (CNTX_B) / eMIOS Channel (eMIOS[11])

PD[3] is a GPIO pin. CNTX_B is the transmit pin for the FlexCan B module. eMIOS[11] is an output-only channel pin for the eMIOS200 module.

2.7.4.5 PD4 — GPIO (PD[4]) / CAN_C Transmit (CNTX_C) / eMIOS Channel (eMIOS[12])

PD[4] is a GPIO pin. CNTX_C is the transmit pin for the FlexCan C module. eMIOS[12] is an output-only channel pin for the eMIOS200 module.

2.7.4.6 PD5 — GPIO (PD[5]) / CAN_C Receive (CNRX_C) / eMIOS Channel (eMIOS[13])

PD[5] is a GPIO pin. CNRX_C is the receive pin for the FlexCan C module. eMIOS[13] is an output-only channel pin for the eMIOS200 module.

2.7.4.7 PD6 — GPIO (PD[6]) / SCI_A Transmit (TXD_A) / eMIOS Channel (eMIOS[14])

PD[6] is a GPIO pin. TXD_A is the transmit pin for the eSCI_A module. eMIOS[14] is an output-only channel pin for the eMIOS200 module.

2.7.4.8 PD7 — GPIO (PD[7]) / SCI_A Receive (RXD_A) / eMIOS Channel (eMIOS[15])

PD[7] is a GPIO pin. RXD_A is the receive pin for the eSCI_A module. eMIOS[15] is an output-only channel pin for the eMIOS200 module.

2.7.4.9 PD8 — GPIO (PD[8]) / SCI_B Transmit (TXD_B) / I²C Serial Clock Line (SCL_A)

PD[8] is a GPIO pin. TXD_B is the transmit pin for the eSCI_B module. SCL_A is the serial clock signal for the I²C_A module.

2.7.4.10 PD9 — GPIO (PD[9]) / SCI_B Receive (RXD_B) / I²C Serial Data Line (SDA_A)

PD[9] is a GPIO pin. RXD_B is the receive pin for the eSCI_B module. SDA_A is the serial data line for the I²C_A module.

2.7.4.11 PD10 — GPIO (PD[10]) / DSPI_B Peripheral Chip Select (PCS_B[2]) / CAN_F Transmit (CNTX_F) / e200z1 Critical Interrupt (NMI0)

PD[10] is a GPIO pin. PCS_B[2] is a peripheral chip select output pin for the DSPI B module. CNTX_F is the transmit pin for the FlexCan F module. NMI0 is the critical interrupt input for the e200z1 core.

2.7.4.12 PD11 — GPIO (PD[11]) / DSPI_B Peripheral Chip Select (PCS_B[1]) / CAN_F Receive (CNRX_F) / e200z0 Critical Interrupt (NMI1)

PD[11] is a GPIO pin. PCS_B[1] is a peripheral chip select output pin for the DSPI B module. CNRX_F is the receive pin for the FlexCan F module. NMI1 is the critical interrupt input for the e200z0 core.

2.7.4.13 PD12 — GPIO (PD[12]) / DSPI_B Peripheral Chip Select (PCS_B[0]) / eMIOS Channel (eMIOS[9])

PD[12] is a GPIO pin. PCS_B[0] is a peripheral chip select output pin for the DSPI B module. eMIOS[9] is an output-only channel pin for the eMIOS200 module.

2.7.4.14 PD13 — GPIO (PD[13]) / DSPI_B Clock (SCK_B) / eMIOS Channel (eMIOS[8])

PD[13] is a GPIO pin. SCK_B is the SPI clock pin for the DSPI B module. eMIOS[8] is an output-only channel pin for the eMIOS200 module.

2.7.4.15 PD14 — GPIO (PD[14]) / DSPI_B Data Output (SOUT_B) / eMIOS Channel (eMIOS[7])

PD[14] is a GPIO pin. SOUT_B is the data output pin for the DSPI B module. eMIOS[7] is an output-only channel pin for the eMIOS200 module.

2.7.4.16 PD15 — GPIO (PD[15]) / DSPI_B Data Input (SIN_B) / eMIOS Channel (eMIOS[6])

PD[15] is a GPIO pin. SIN_B is the data input pin for the DSPI B module. eMIOS[6] is an output-only channel pin for the eMIOS200 module.

2.7.5 Port E Pins

2.7.5.1 PE0 — GPIO (PE[0]) / DSPI_A Peripheral Chip Select (PCS_A[2]) / eMIOS Channel (eMIOS[5]) / MLB Clock (MLBCLK)

PE[0] is a GPIO pin. PCS_A[2] is a peripheral chip select output pin for the DSPI A module. eMIOS[5] is an output-only channel pin for the eMIOS200 module. MLBCLK is the clock pin for the emulated MLB module.

2.7.5.2 PE1 — GPIO (PE[1]) / DSPI_A Peripheral Chip Select (PCS_A[1]) / eMIOS Channel (eMIOS[4]) / MLB Signal In / Signal (MLBSI / MLBSIG)

PE[1] is a GPIO pin. PCS_A[1] is a peripheral chip select output pin for the DSPI A module. eMIOS[4] is an output-only channel pin for the eMIOS200 module. In a 3-pin MLB interface, MLBSIG is the bidirectional signal line that transfers bus management data to/from the MOST network controller. In a 5-pin interface, MLBSI carries signal line data from the MOST network controller to the emulated MLB module.

2.7.5.3 PE2 — GPIO (PE[2]) / DSPI_A Peripheral Chip Select (PCS_A[0]) / eMIOS Channel (eMIOS[3]) / MLB Data In / Data (MLBDI / MLBDAT)

PE[2] is a GPIO pin. PCS_A[0] is a peripheral chip select output pin for the DSPI A module. eMIOS[3] is an output-only channel pin for the eMIOS200 module. In a 3-pin MLB interface, MLBDAT is the bidirectional data line that transfers user data to/from the MOST network controller. In a 5-pin MLB interface, MLBDI carries user data from the MOST network controller to the emulated MLB module.

2.7.5.4 PE3 — GPIO (PE[3]) / DSPI_A Clock (SCK_A) / eMIOS Channel (eMIOS[2]) / MLB Signal Out / Level Shifter Enable (MLBSO / MLBSIG_BUFEN)

PE[3] is a GPIO pin. SCK_A is the SPI clock pin for the DSPI A module. eMIOS[2] is an output-only channel pin for the eMIOS200 module. In a 3-pin MLB interface, MLBSIG_BUFEN controls the external level shifter for the MLBSIG pin. In a 5-pin MLB interface, MLBSO carries signal data from the emulated MLB module to the MOST network controller.

2.7.5.5 PE4 — GPIO (PE[4]) / DSPI_A Data Output (SOUT_A) / eMIOS Channel (eMIOS[1]) / MLB Data Out / Level Shifter Enable (MLBDO / MLBDAT_BUFEN)

PE[4] is a GPIO pin. SOUT_A is the data output pin for the DSPI A module. eMIOS[1] is an output-only channel pin for the eMIOS200 module. In a 3-pin MLB interface, MLBDAT_BUFEN controls the external level shifter for the MLBDAT pin. In a 5-pin MLB interface, MLBDO carries user data from the emulated MLB module to the MOST network controller.

2.7.5.6 PE5 — GPIO (PE[5]) / DSPI_A Data Input (SIN_A) / eMIOS Channel (eMIOS[0]) / MLB SLOT / Signal Observe / Data Observe (MLB_SLOT / MLB_SIGOBS / MLB_DATOBS)

PE[5] is a GPIO pin. SIN_A is the data input pin for the DSPI A module. eMIOS[0] is an output-only channel pin for the eMIOS200 module. MLB_SLOT, MLB_SIGOBS, and MLB_DATOBS are debug signals for the MLB module.

2.7.5.7 PE6 — GPIO (PE[6]) / Clock Output (CLKOUT)

PE[6] is a GPIO pin. CLKOUT is the external bus interface clock output.

2.7.5.8 PE7 to PE15 — GPIO (PE[7:15])

PE[7:15] are GPIO pins.

2.7.6 Port F Pins

2.7.6.1 PF0 — GPIO (PF[0]) / EBI Read/Write (RD_ \overline{WR}) / Nexus Event In (\overline{EVTI})

PF[0] is a GPIO pin. RD_ \overline{WR} indicates whether an external bus transfer is a read or write operation. \overline{EVTI} is an input that is read on the assertion of JCOMP to enable or disable the Nexus Debug port. After reset, the \overline{EVTI} pin initiates program and data trace synchronization messages or generates a breakpoint.

2.7.6.2 PF1 — GPIO (PF[1]) / EBI Transfer Acknowledge (\overline{TA}) / Nexus Event Out (EVTO) / MLB Clock (MLBCLK)

PF[1] is a GPIO pin. \overline{TA} indicates to the external bus master that the slave has completed the current transfer. EVTO is an output providing timing to a development tool for a single watch point or breakpoint occurrence. MLBCLK is the clock pin for the emulated MLB module.

2.7.6.3 PF2 — GPIO (PF[2]) / EBI Multiplex Address/Data (AD[8]) / EBI Non Muxed Address (ADDR[8]) / MLB Signal In / Signal (MLBSI / MLBSIG) / Nexus Message Start/End Out (\overline{MSEO})

PF[2] is a GPIO pin. AD[8] is the EBI multiplexed address and data bus. ADDR[8] is the EBI non multiplexed address bus. In a 3-pin MLB interface, MLBSIG is the bidirectional signal line that transfers bus management data to/from the MOST network controller. In a 5-pin interface, MLBSI carries signal line data from the MOST network controller to the emulated MLB module. \overline{MSEO} is an output that indicates when messages start and end on the MDO pins.

2.7.6.4 PF3 — GPIO (PF[3]) / EBI Multiplex Address/Data (AD[9]) / EBI Non Muxed Address (ADDR[9]) / MLB Data In / Data (MLBDI / MLBDAT) / Nexus Message Clock Out (MCKO)

PF[3] is a GPIO pin. AD[9] is the EBI multiplexed address and data bus. ADDR[9] is the EBI non multiplexed address bus. In a 3-pin MLB interface, MLBDAT is the bidirectional data line that transfers user data to/from the MOST network controller. In a 5-pin MLB interface, MLBDI carries user data from the MOST network controller to the emulated MLB module. MCKO is a free running clock output to the development tools that is used for timing of the MDO and MSEO signals.

2.7.6.5 PF4 — GPIO (PF[4]) / EBI Multiplex Address/Data (AD[10]) / EBI Non Muxed Address (ADDR[10]) / MLB Signal Out / Level Shifter Enable (MLBSO / MLBSIG_BUFEN) / Nexus Message Data Out (MDO[0])

PF[4] is a GPIO pin. AD[10] is the EBI multiplexed address and data bus. ADDR[10] is the EBI non multiplexed address bus. In a 3-pin MLB interface, MLBSIG_BUFEN controls the external level shifter for the MLBSIG pin. In a 5-pin MLB interface, MLBSO carries signal data from the emulated MLB module to the MOST network controller. MDO[0] is a trace message output to the development tools.

2.7.6.6 PF5 — GPIO (PF[5]) / EBI Multiplex Address/Data (AD[11]) / EBI Non Muxed Address (ADDR[11]) / MLB Data Out / Level Shifter Enable (MLBDO / MLBDAT_BUFEN) / Nexus Message Data Out (MDO[1])

PF[5] is a GPIO pin. AD[11] is the EBI multiplexed address and data bus. ADDR[11] is the EBI non multiplexed address bus. In a 3-pin MLB interface, MLBDAT_BUFEN controls the external level shifter for the MLBDAT pin. In a 5-pin MLB interface, MLBDO carries user data from the emulated MLB module to the MOST network controller. MDO[1] is a trace message output to the development tools.

2.7.6.7 PF6 — GPIO (PF[6]) / EBI Multiplex Address/Data (AD[12]) / EBI Non Muxed Address (ADDR[12]) / MLB SLOT / Signal Observe / Data Observe (MLB_SLOT / MLB_SIGOBS / MLB_DATOBS) / Nexus Message Data Out (MDO[2])

PF[6] is a GPIO pin. AD[12] is the EBI multiplexed address and data bus. ADDR[12] is the EBI non multiplexed address bus. MLB_SLOT, MLB_SIGOBS, and MLB_DATOBS are debug signals for the MLB module. MDO[2] is a trace message output to the development tools.

2.7.6.8 PF7 — GPIO (PF[7]) / EBI Multiplex Address/Data (AD[13]) / EBI Non Muxed Address (ADDR[13]) / Nexus Message Data Out (MDO[3])

PF[7] is a GPIO pin. AD[13] is the EBI multiplexed address and data bus. ADDR[13] is the EBI non multiplexed address bus. MDO[3] is a trace message output to the development tools.

2.7.6.9 PF8 — GPIO (PF[8]) / EBI Multiplex Address/Data (AD[14]) / EBI Non Muxed Address (ADDR[14]) / Nexus Message Data Out (MDO[4])

PF[8] is a GPIO pin. AD[14] is the EBI multiplexed address and data bus. ADDR[14] is the EBI non multiplexed address bus. MDO[4] is a trace message output to the development tools.

2.7.6.10 PF9 — GPIO (PF[9]) / EBI Multiplex Address/Data (AD[15]) / EBI Non Muxed Address (ADDR[15]) / Nexus Message Data Out (MDO[5])

PF[9] is a GPIO pin. AD[15] is the EBI multiplexed address and data bus. ADDR[15] is the EBI non multiplexed address bus. MDO[5] is a trace message output to the development tools.

2.7.6.11 PF10 — GPIO (PF[10]) / EBI Chip Select (\overline{CS} [1]) / SCI_C Transmit (TXD_C) / Nexus Message Data Out (MDO[6])

PF[10] is a GPIO pin. \overline{CS} [1] is the EBI chip select output signals. TXD_C is the transmit pin for the eSCI C module. MDO[6] is a trace message output to the development tools.

2.7.6.12 PF11 — GPIO (PF[11]) / EBI Chip Select (\overline{CS} [0]) / SCI_C Receive (RXD_C) / Nexus Message Data Out (MDO[7])

PF[11] is a GPIO pin. \overline{CS} [0] is the EBI chip select output signals. RXD_C is the receive pin for the eSCI C module. MDO[7] is a trace message output to the development tools.

2.7.6.13 PF12 — GPIO (PF[12]) / EBI Transfer Start (\overline{TS}) / SCI_D Transmit (TXD_D) / EBI Address Latch Enable

PF[12] is a GPIO pin. \overline{TS} is the EBI transfer start output signals. TXD_D is the transmit pin for the eSCI D module. ALE is the EBI address latch enable.

2.7.6.14 PF13 — GPIO (PF[13]) / EBI Output Enable (\overline{OE}) / SCI_D Receive (RXD_D)

PF[13] is a GPIO pin. \overline{OE} is the EBI chip select output signals. RXD_D is the receive pin for the eSCI D module.

2.7.6.15 PF14 — GPIO (PF[14]) / EBI Write Enable (\overline{WE} [0]) / EBI Burst Data In Progress (\overline{BDIP}) / CAN_D Transmit (CNTX_D)

PF[14] is a GPIO pin. \overline{WE} [0] specifies which data pins contain valid data for an external bus transfer. \overline{BDIP} indicates that an EBI burst transfer is in progress. CNTX_D is the transmit pin for the FlexCan D module.

2.7.6.16 PF15 — GPIO (PF[15]) / EBI Write Enable (\overline{WE} [1]) / EBI Transfer Error Acknowledge (\overline{TEA}) / CAN_D Receive (CNRX_D)

PF[15] is a GPIO pin. \overline{WE} [1] specifies which data pins contain valid data for an external bus transfer. \overline{TEA} indicates that an error occurred in the current external bus transfer. CNRX_D is the receive pin for the FlexCan D module.

2.7.7 Port G Pins

2.7.7.1 PG0 — GPIO (PG[0]) / EBI Multiplex Address/Data (AD[16]) / eMIOS Channel (eMIOS[16])

PG[0] is a GPIO pin. AD[16] is the EBI multiplexed address and data bus. eMIOS[16] is an input/output channel pin for the eMIOS200 module.

2.7.7.2 PG1 — GPIO (PG[1]) / EBI Multiplex Address/Data (AD[17]) / eMIOS Channel (eMIOS[17]) / DSPI_C Data In (SIN_C)

PG[1] is a GPIO pin. AD[17] is the EBI multiplexed address and data bus. eMIOS[17] is an input/output channel pin for the eMIOS200 module. SIN_C is the data input pin for the DSPI C module.

2.7.7.3 PG2 — GPIO (PG[2]) / EBI Multiplex Address/Data (AD[18]) / eMIOS Channel (eMIOS[18]) / DSPI_C Data Out (SOUT_C)

PG[2] is a GPIO pin. AD[18] is the EBI multiplexed address and data bus. eMIOS[18] is an input/output channel pin for the eMIOS200 module. SOUT_C is the data output pin for the DSPI C module.

2.7.7.4 PG3 — GPIO (PG[3]) / EBI Multiplex Address/Data (AD[19]) / eMIOS Channel (eMIOS[19]) / DSPI_C Serial Clock (SCK_C)

PG[3] is a GPIO pin. AD[19] is the EBI multiplexed address and data bus. eMIOS[19] is an input/output channel pin for the eMIOS200 module. SCK_C is the SPI clock pin for the DSPI C module.

2.7.7.5 PG4 — GPIO (PG[4]) / EBI Multiplex Address/Data (AD[20]) / eMIOS Channel (eMIOS[20]) / DSPI_C Peripheral Chip Select (PCS_C[0])

PG[4] is a GPIO pin. AD[20] is the EBI multiplexed address and data bus. eMIOS[20] is an input/output channel pin for the eMIOS200 module. PCS_C[0] is a peripheral chip select output pin for the DSPI C module.

2.7.7.6 PG5 — GPIO (PG[5]) / EBI Multiplex Address/Data (AD[21]) / eMIOS Channel (eMIOS[21])

PG[5] is a GPIO pin. AD[21] is the EBI multiplexed address and data bus. eMIOS[21] is an input/output channel pin for the eMIOS200 module.

2.7.7.7 PG6 — GPIO (PG[6]) / EBI Multiplex Address/Data (AD[22]) / eMIOS Channel (eMIOS[22])

PG[6] is a GPIO pin. AD[22] is the EBI multiplexed address and data bus. eMIOS[22] is an input/output channel pin for the eMIOS200 module.

2.7.7.8 PG7 — GPIO (PG[7]) / EBI Multiplex Address/Data (AD[23]) / eMIOS Channel (eMIOS[23]) / SCI_C Receive (RXD_C)

PG[7] is a GPIO pin. AD[23] is the EBI multiplexed address and data bus. eMIOS[23] is an input/output channel pin for the eMIOS200 module. RXD_C is the receive pin for the eSCI C module.

2.7.7.9 PG8 — GPIO (PG[8]) / EBI Multiplex Address/Data (AD[24]) / DSPI_A Peripheral Chip Select (PCS_A[4])

PG[8] is a GPIO pin. AD[24] is the EBI multiplexed address and data bus. PCS_A[4] is a peripheral chip select output pin for the DSPI A module.

2.7.7.10 PG9 — GPIO (PG[9]) / EBI Multiplex Address/Data (AD[25]) / DSPI_A Peripheral Chip Select (PCS_A[3]) / SCI_C Transmit (TXD_C)

PG[9] is a GPIO pin. AD[25] is the EBI multiplexed address and data bus. PCS_A[3] is a peripheral chip select output pin for the DSPI A module. TXD_C is the transmit pin for the eSCI_C module.

2.7.7.11 PG10 — GPIO (PG[10]) / EBI Multiplex Address/Data (AD[26]) / DSPI_A Peripheral Chip Select (PCS_A[2])

PG[10] is a GPIO pin. AD[26] is the EBI multiplexed address and data bus. PCS_A[2] is a peripheral chip select output pin for the DSPI A module.

2.7.7.12 PG11 — GPIO (PG[11]) / EBI Multiplex Address/Data (AD[27]) / DSPI_A Peripheral Chip Select (PCS_A[1])

PG[11] is a GPIO pin. AD[27] is the EBI multiplexed address and data bus. PCS_A[1] is a peripheral chip select output pin for the DSPI A module.

2.7.7.13 PG12 — GPIO (PG[12]) / EBI Multiplex Address/Data (AD[28]) / DSPI_A Peripheral Chip Select (PCS_A[0])

PG[12] is a GPIO pin. AD[28] is the EBI multiplexed address and data bus. PCS_A[0] is a peripheral chip select output pin for the DSPI A module.

2.7.7.14 PG13 — GPIO (PG[13]) / EBI Multiplex Address/Data (AD[29]) / DSPI_A Serial Clock (SCK_A)

PG[13] is a GPIO pin. AD[29] is the EBI multiplexed address and data bus. SCK_A is the SPI clock pin for the DSPI A module.

2.7.7.15 PG14 — GPIO (PG[14]) / EBI Multiplex Address/Data (AD[30]) / DSPI_C Data Out (SOUT_A)

PG[14] is a GPIO pin. AD[24] is the EBI multiplexed address and data bus. SOUT_A is the data output pin for the DSPI A module.

2.7.7.16 PG15 — GPIO (PG[15]) / EBI Multiplex Address/Data (AD[31]) / DSPI_C Data In (SIN_A)

PG[15] is a GPIO pin. AD[31] is the EBI multiplexed address and data bus. SIN_A is the data input pin for the DSPI A module.

2.7.8 Port H Pins

2.7.8.1 PH0 — GPIO (PH[0]) / Analog Input (AN[27]) / eMIOS Channel (eMIOS[20]) / I²C Serial Clock Line (SCL_A)

PH[0] is a GPIO pin. AN[27] is a single-ended analog input pin. eMIOS[20] is an output-only channel pin for the eMIOS200 module. SCL_A is the serial clock signal for the I²C_A module.

2.7.8.2 PH1 — GPIO (PH[1]) / Analog Input (AN[26]) / eMIOS Channel (eMIOS[21]) / I²C Serial Data Line (SDA_A)

PH[1] is a GPIO pin. AN[26] is a single-ended analog input pin. eMIOS[21] is an output-only channel pin for the eMIOS200 module. SDA_A is the serial data signal for the I²C_A module.

2.7.8.3 PH2 — GPIO (PH[2]) / Analog Input (AN[25]) / eMIOS Channel (eMIOS[22]) / EBI Chip Select (CS[3])

PH[2] is a GPIO pin. AN[25] is a single-ended analog input pin. eMIOS[22] is an output-only channel pin for the eMIOS200 module. CS[3] is an EBI chip select output.

2.7.8.4 PH3 — GPIO (PH[3]) / Analog Input (AN[24]) / eMIOS Channel (eMIOS[23]) / EBI Chip Select (CS[2])

PH[3] is a GPIO pin. AN[24] is a single-ended analog input pin. eMIOS[23] is an output-only channel pin for the eMIOS200 module. CS[2] is an EBI chip select output.

2.7.8.5 PH4 — GPIO (PH[4]) / Analog Input (AN[23]) / SCI_E Transmit (TXD_E) / External Analog Mux Address Output (MA[2])

PH[4] is a GPIO pin. AN[23] is a single-ended analog input pin. TXD_E is the transmit pin for the eSCI_E module. MA[2] is a address output for an external analog mux used to select the mux input channel to connect to the QADC.

2.7.8.6 PH5 — GPIO (PH[5]) / Analog Input (AN[22]) / SCI_E Receive (RXD_E) / External Analog Mux Address Output (MA[1])

PH[5] is a GPIO pin. AN[22] is a single-ended analog input pin. RXD_E is the receive pin for the eSCI_E module. MA[1] is a address output for an external analog mux used to select the mux input channel to connect to the QADC.

2.7.8.7 PH6 — GPIO (PH[6]) / Analog Input (AN[21]) / SCI_E Transmit (TXD_F)

PH[6] is a GPIO pin. AN[21] is a single-ended analog input pin. TXD_F is the transmit pin for the eSCI_F module.

2.7.8.8 PH7 — GPIO (PH[7]) / Analog Input (AN[20]) / SCI_F Receive (RXD_F)

PH[7] is a GPIO pin. AN[20] is a single-ended analog input pin. RXD_F is the receive pin for the eSCI_F module.

2.7.8.9 PH8 — GPIO (PH[8]) / Analog Input (AN[19]) / CAN_E Transmit (CNTX_E) / External Analog Mux Address Output (MA[0])

PH[8] is a GPIO pin. AN[19] is a single-ended analog input pin. CNTX_E is the transmit pin for the FlexCAN_E module. MA[0] is an address output for an external analog mux used to select the mux input channel to connect to the QADC.

2.7.8.10 PH9 — GPIO (PH[9]) / Analog Input (AN[18]) / CAN_E Receive (CNRX_E)

PH[9] is a GPIO pin. AN[18] is a single-ended analog input pin. CNRX_E is the receive pin for the FlexCAN_E module.

2.7.8.11 PH10 — GPIO (PH[10]) / Analog Input (AN[17]) / CAN_F Receive (CNRX_F)

PH[10] is a GPIO pin. AN[17] is a single-ended analog input pin. CNRX_F is the receive pin for the FlexCAN_F module.

2.7.8.12 PH11 — GPIO (PH[11]) / Analog Input (AN[16]) / CAN_F Transmit (CNTX_F)

PH[11] is a GPIO pin. AN[16] is a single-ended analog input pin. CNTX_F is the transmit pin for the FlexCAN_F module.

2.7.8.13 PH12 — GPIO (PH[12]) / DSPI_D Peripheral Chip Select (PCS_D[5])

PH[12] is a GPIO pin. PCS_D[5] is a peripheral chip select output pin for the DSPI_D module.

2.7.8.14 PH13 — GPIO (PH[13])

PH[13] is a GPIO pin.

2.7.8.15 PH14 — GPIO (PH[14]) / EBI Write Enable (\overline{WE} [2])

PH[14] is a GPIO pin. \overline{WE} [2] specifies which data pins contain valid data for an external bus transfer.

2.7.8.16 PH15 — GPIO (PH[15]) / EBI Write Enable (\overline{WE} [3])

PH[15] is a GPIO pin. \overline{WE} [3] specifies which data pins contain valid data for an external bus transfer.

2.7.9 Port J Pins**2.7.9.1 PJ0 to PJ7 — GPIO (PJ[0:7]) / EBI Multiplex Address/Data (AD[0:7])**

PJ[0:7] are GPIO pins. AD[0:7] are EBI multiplexed address and data bus pins.

2.7.9.2 PJ8 — GPIO (PJ8) / DSPI_D Peripheral Chip Select (PCS_D[4])

PJ8 is a GPIO pin. PCS_D[4] is a peripheral chip select output pin for the DSPI_D module.

2.7.9.3 PJ9 — GPIO (PJ9) / DSPI_D Peripheral Chip Select (PCS_D[3])

PJ9 is a GPIO pin. PCS_D[3] is a peripheral chip select output pin for the DSPI_D module.

2.7.9.4 PJ10 — GPIO (PJ10) / DSPI_D Peripheral Chip Select (PCS_D[2])

PJ10 is a GPIO pin. PCS_D[2] is a peripheral chip select output pin for the DSPI_D module.

2.7.9.5 PJ11 — GPIO (PJ11) / DSPI_D Peripheral Chip Select (PCS_D[1])

PJ11 is a GPIO pin. PCS_D[1] is a peripheral chip select output pin for the DSPI_D module.

2.7.9.6 PJ12 — GPIO (PJ12) / DSPI_D Peripheral Chip Select (PCS_D[0])

PJ12 is a GPIO pin. PCS_D[0] is a peripheral chip select output pin for the DSPI_D module.

2.7.9.7 PJ13 - GPIO (PJ13) / DSPI_D Clock (SCK_D)

PJ13 is a GPIO pin. SCK_D is the SPI clock pin of the DSPI_D module.

2.7.9.8 PJ14 - GPIO (PJ14) / DSPI_D Serial Data Out (SOUT_D)

PJ14 is a GPIO pin. SOUT_D is the SPI serial data out for the DSPI_D module.

2.7.9.9 PJ15 - GPIO (PJ15) / DSPI_D Serial Data In (SIN_D)

PJ15 is a GPIO pin. SIN_D is the SPI serial data in for the DSPI_D module.

2.7.10 Port K Pins

2.7.10.1 PK0 — GPI (PK0) / 32 kHz Crystal Input (EXTAL32)

PK0 is a GPI pin. EXTAL32 is the input pin for an external 32 kHz crystal oscillator. (The EXTAL32 function is available on the PA14 pin on the 144LQFP package and on the PK0 pin on the 176LQFP and 208BGA packages.)

2.7.10.2 PK1 — GPI (PK[1]) / 32 kHz Crystal Output (XTAL32)

PK1 is a GPI pin. XTAL32 is the output pin for an external 32 kHz crystal oscillator (XTAL32 function available on A15 pin on the 144LQFP package and PK1 pin on the 176LQFP and 208BGA packages).

2.7.11 Miscellaneous Pins

2.7.11.1 XTAL — Crystal Oscillator Output

XTAL is the output pin for an external crystal oscillator.

2.7.11.2 EXTAL — Crystal Oscillator Input / External Clock Input

EXTAL is the input pin for an external crystal oscillator or an external clock source. The alternate function is the external clock input.

2.7.11.3 $\overline{\text{RESET}}$ — External Reset Input

The $\overline{\text{RESET}}$ pin is a bidirectional I/O pin. It is asserted by an external device to reset all modules of the MCU, except the RTC counter. It is also an open drain output signal that is asserted during an internal reset. See [Chapter 7, “Reset,”](#) for more detail.

2.7.11.4 TCK — JTAG Test Clock Input

TCK provides the clock input for the on-chip test logic.

2.7.11.5 TDI — JTAG Test Data Input

TDI provides the serial test instruction and data input for the on-chip test logic.

2.7.11.6 TDO — JTAG Test Data Output

TDO provides the serial test data output for the on-chip test logic.

2.7.11.7 TMS — JTAG Test Mode Select Input

TMS controls test mode operations for the on-chip test logic.

2.7.11.8 JCOMP — JTAG Compliance Input

The JCOMP pin is used to enable the JTAG TAP controller.

2.7.11.9 TEST — Test Mode Enable Input

The TEST pin is used to place the chip in test mode. It must be negated for normal operation, and should be connected to ground in all customer applications.

2.7.12 Power and Ground Pins

2.7.12.1 Voltage Regulator Reference (VDDR)

VDDR is the voltage reference to the internal voltage regulator.

2.7.12.2 VDDA — Analog-to-Digital Converter Analog Supply

VDDA is the analog supply for the eQADC.

2.7.12.3 VSSA — Analog-to-Digital Converter Analog Ground

VSSA is the analog ground for the eQADC.

2.7.12.4 VRH — Analog-to-Digital Converter Reference High

VRH is the reference high input for the eQADC.

2.7.12.5 VRL — Analog-to-Digital Converter Reference Low

VRL is the reference low input for the eQADC.

2.7.12.6 REFBYPC — Reference Bypass Capacitor

REFBYPC is a bypass capacitor input for the eQADC. The REFBYPC pin is used to connect an external bias capacitor between the REFBYPC pin and VRL.

2.7.12.7 VDDSYN — Clock Synthesizer Supply

VDDSYN is the supply power for the FMPLL.

2.7.12.8 VSSSYN — Clock Synthesizer Ground

VSSSYN is the ground reference for the FMPLL.

2.7.12.9 VDD33 — 3.3 V I/O and Flash Read Supply (VFLASH)

VDD33 is the 3.3 V internal supply used for the external I/O control logic. It is intended only for the connection of bypass capacitors, and must not be connect to load or power. (VFLASH is the on-chip flash read supply.)

2.7.12.10 VPP — Flash Program/Erase Supply

VPP is the on-chip flash program/erase supply.

2.7.12.11 VDD — Internal Logic Supply and Flash Logic Supply (VDDF)

VDD is the 1.5 V logic and flash supply.

2.7.12.12 VSS — Internal Logic and Flash (VSSF) Ground

VSS is the ground reference for internal logic and the flash.

2.7.12.13 VDDEx — External I/O Supply

VDDEx is the 3.3 V to 5.0 V external I/O supply independently controlling the level for one of three groups of I/O pins. (x=1,2,3.)

2.7.12.14 VSSEx — External I/O Ground

VSSEx is the external I/O ground for one of three groups of I/O pins. (x=1,2,3.)

Chapter 3

System Clock Description

3.1 Introduction

The MPC5510 supports several clock sources that include an internal phase-locked loop (PLL), an external high-frequency crystal (XOSC), an external low-frequency crystal (32kOSC), an internal high-frequency RC oscillator (IRC), and an internal low-frequency RC oscillator (32kRC).

The availability of the clock sources vary, depending on the run, stop, and power mode selected. During low power modes, the PLL and XOSC are not available as clock sources.

The internal system clock may be generated in several ways:

- Internal 16 MHz IRC
- PLL: Normal mode with crystal clock reference for XOSC
- PLL: Normal mode with external clock reference for XOSC
- XOSC with external clock reference (PLL bypass mode)
- XOSC with crystal clock reference

There are two clock output pins driven by programmable clock dividers: CLKOUT and MCKO.

The oscillator clock can be selected as the clock source for the FlexCAN interface in the FlexCAN blocks resulting in very low jitter performance. The oscillator clock can also be selected as the clock source for the FlexRay interface in the FlexRay block.

The default clock source after reset is the 16 MHz IRC.

3.2 Clock Sources

The various clock sources that are available on MPC5510 are shown in [Figure 3-1](#) and discussed in more detail in subsequent sections.

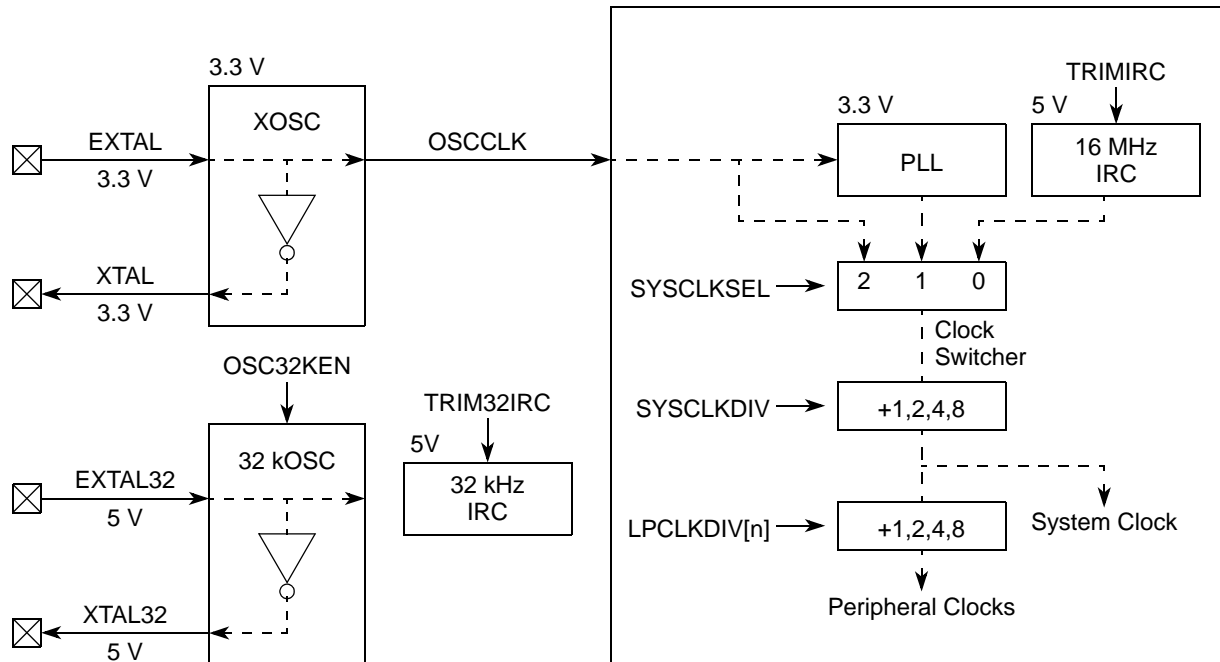


Figure 3-1. MPC5510 Available Clock Sources

3.2.1 External High-Frequency Crystal (XOSC)

The MPC5510 features an internal automatic level control (ALC) oscillator. The oscillator is designed for optimal startup margin with typical crystal oscillators. Oscillator power is supplied from its own 3.3 V PLL supply voltage generated by the voltage regulator to minimize noise. The oscillator provides the reference clock for the entire chip. As such, it may be used to drive the system clock directly (when the PLL is bypassed), or as the input reference clock for the PLL.

A square-wave input can also be supplied to the device through the oscillator by connecting the external clock source to the EXTAL pin with the oscillator operating in external-clock mode.

Features:

- Supports external high-frequency resonator or crystal in the range of f_{ref} (see MPC5510 data sheet for reference frequency specification)
- Pierce oscillator
- Two external pins are dedicated for this function (EXTAL, XTAL)
- 40 MHz max required to support FlexRay
- Clock input to PLL
- This clock source is capable of supporting FlexCAN communications (jitter < 0.5%)
- This clock source is capable of supporting FlexRay communications (jitter < 0.5%) (duty cycle = $50 \pm 10\%$)

- Clock for the RTI
- Optional disable
- Enabled by default after reset

3.2.2 External Low-Frequency Crystal (32kXOSC)

The MPC5510 supports an external 32 kHz crystal to provide accurate wake-up and time-keeping functions.

Features:

- Two external pins required: EXTAL32 and XTAL32
- Supports external low frequency crystal in the range of $f_{\text{ref}32}$ (see MPC5510 data sheet for reference frequency specification)
- Option to clock the API to provide a more accurate wakeup
- Option to clock the RTC to provide accurate time keeping
- Powered from 5 V
- Optional disable

3.2.3 Internal High-Frequency RC Oscillator (IRC)

The MPC5510 includes a 16 MHz IRC as the default system clock out of reset.

Features:

- Fast stabilization, enabling fast recovery
- Frequency trimmable for accuracy
- Option to clock software watchdog timer
- Powered from 5 V
- Always enabled except optionally disabled in sleep modes when not being used

3.2.4 Internal Low-Frequency RC Oscillator (32kRC)

The MPC5510 includes a 32 kHz internal RC oscillator that is intended to be used as a highly reliable clock source during low-power modes.

Features:

- Frequency trimmable for accuracy
- Option to clock the API to provide a wakeup
- Option to clock the RTC to provide time keeping
- Powered from 5 V
- Optionally enabled

3.3 System Clock Architecture Block Diagram

To optimize system power consumption, the MPC5510 supports both system- and peripheral-level clock dividers, and static clock gating using peripheral-level module disable (MDIS) bits and a system-level halt mechanism. Figure 3-2 shows the device-level clock gating mechanism for the MPC5510. These features are detailed in subsequent sections.

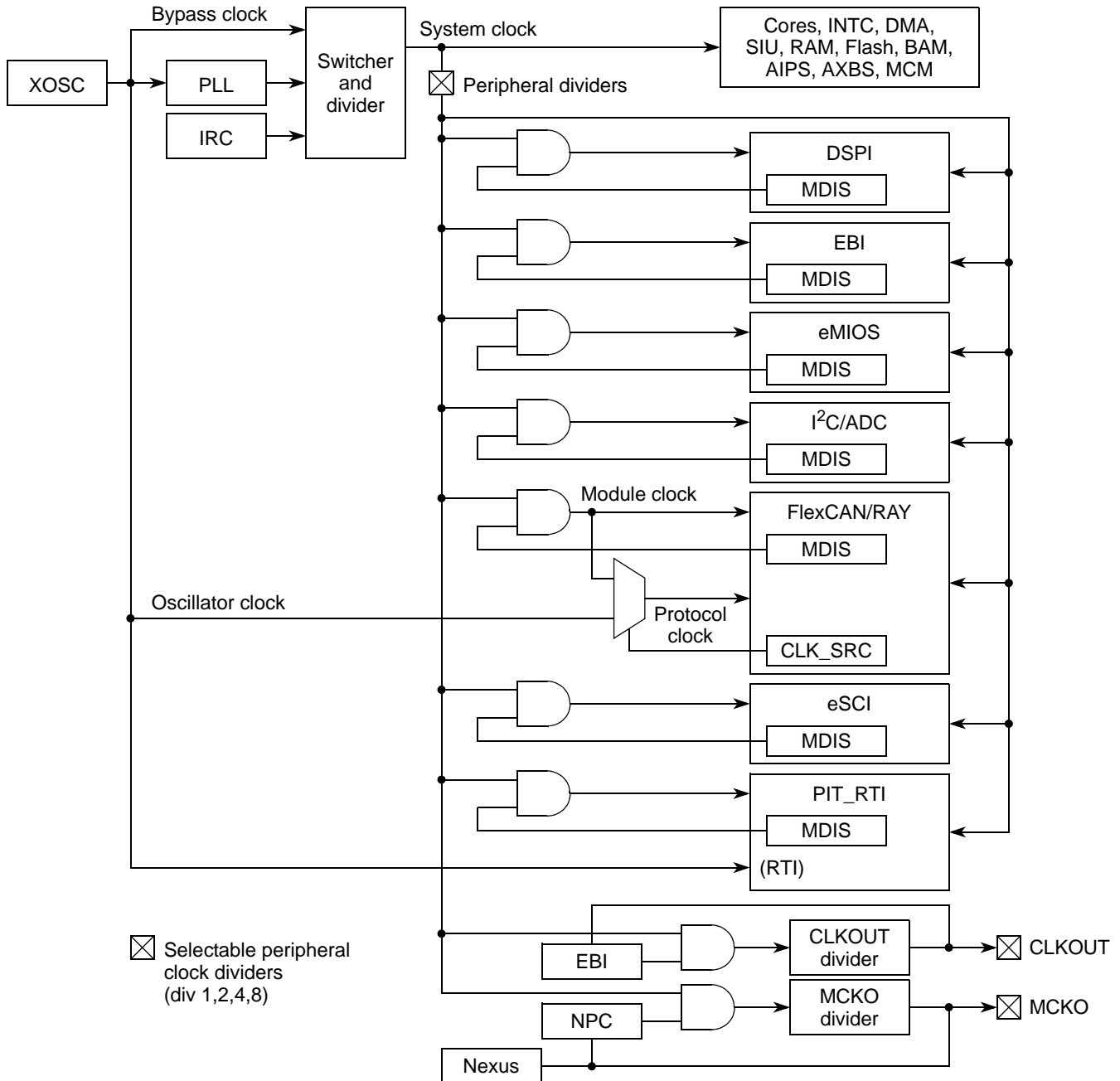


Figure 3-2. System Clock Architecture

3.4 Clock Dividers

3.4.1 System Clock Select

The source for the system clock can be selected by the `SYSCLKSEL` field of the SIU system clock register (`SIU_SYSCLK`) to be the 16 MHz IRC, the XOSC, or the PLL.

3.4.2 System Clock Dividers

The system clock dividers can be programmed to create a system clock, which is created from the selected clock source divided by 1, 2, 4, or 8, based on the setting of the `SYSCLKDIV` field in the SIU system clock register (`SIU_SYSCLK`).

3.4.3 External Bus Clock (CLKOUT) Divider

The external bus clock (`CLKOUT`) divider can be programmed to create a `CLKOUT`, which is created from the system clock divided by 1, 2, or 4, based on the settings of the `EBDF` bit field in the SIU external clock control register (`SIU_ECCR`). The reset value of `EBDF` selects a `CLKOUT` frequency of one half of the system clock frequency. The EBI supports gating of the `CLKOUT` signal when there are no external bus accesses in progress.

The `CLKOUT` divider provides a nominal 50% duty cycle clock. There is no guaranteed phase relationship between `CLKOUT` and `MCKO`.

3.4.4 Nexus Message Clock (MCKO) Divider

The Nexus message clock (`MCKO`) divider can be programmed to divide the system clock by one, two, four, or eight based on the `MCKO_DIV` bit field in the port configuration register (`PCR`) in the Nexus port controller (`NPC`). The reset value of `MCKO_DIV` selects an `MCKO` clock frequency one half of the system clock frequency. The `MCKO` divider is configured by writing to the `NPC` through the JTAG port. The `MCKO_EN` bit may be used to disable the `MCKO` clock. The `MCKO_GT` bit may be used to disable the `MCKO` clock when Nexus is not actively transmitting messages on the Nexus port.

The `MCKO` divider provides a nominal 50% duty cycle clock. There is no guaranteed phase relationship between `CLKOUT` and `MCKO`.

3.4.5 Peripheral Clock Dividers

The peripheral clock dividers provide a mechanism to reduce run power when it is not necessary to clock peripherals at the full system clock frequency.

The SIU's SIU_SYSCLK[LPCLKDIV n] bits control the clock divide value for each grouping of modules. The divide values may be independently selected for each grouping and support a divide by 1, 2, 4, or 8. [Figure 3-1](#) defines which peripherals are affected by which LPCLKDIV n bits.

Table 3-1. LPCLKDIV Module Groups

LPCLKDIV n	Modules
LPCLKDIV0	FlexCAN_A, DSPI_A
LPCLKDIV1	ESCI_A, I ² C_A, PIT
LPCLKDIV2	FlexCAN_B-F
LPCLKDIV3	DSPI_B-D
LPCLKDIV4	ESCI_B-H
LPCLKDIV5	eMIOS
LPCLKDIV6	MLB
LPCLKDIV7	Reserved

The MPC5510 implements a single clock divider circuit that uses the system clock as its source. The LPCLKDIV bits control which clock divide tap is used for each module grouping clock gate enable. The resultant gated clocks will be at the desired frequency but are clock pulses instead of a 50% duty cycle (the high clock pulse width is half the system clock period). The high-order clock taps will be disabled if not being used. Individual modules should be disabled when changing the LPCLKDIV values affecting the module.

The user is responsible for adjusting the module function, prescalers, protocol timings, etc. based on the LPCLKDIV values. Register accesses will be proportionally longer along with other basic module functions such as interrupts, DMA, etc.

3.5 Software-Controlled Power Management

3.5.1 Module Disable (MDIS) Clock Gating

Static clock gating is enabled by software writes to configuration bits for the modules to disable the modules. Modules are re-enabled by software to ungate the module clocks.

The modules support software controlled clock gating where the application software can disable the non-memory-mapped portions of the blocks by writing to module disable (MDIS) bits in registers within the blocks. (The memory-mapped portions of the blocks are clocked by the system clock only when they are accessed.) The Nexus port controller (NPC) can be configured to disable the MCKO signal when there are no Nexus messages pending. The flash array can be disabled by writing to the STOP bit in the flash's module configuration register (MCR).

The modules that support software-controlled power management/clock gating are listed in [Table 3-2](#) along with the registers and bits that disable each block. Default out of reset disables the software-controlled clocks.

Table 3-2. Software-Controlled Clock Gating Support

Block Name	Register Name	Bit Name
DSPI	MCR	MDIS
ESCI	MCR	MDIS
FlexCAN	MCR	MDIS
EMIOS	MCR	MDIS
EBI	MCR	MDIS
MLB	MCR	MDIS
PIT_RTI	MCR	MDIS ¹
I ² C	IBCR	MDIS
NPC	MCR	MCKO_EN, MCKO_GT
Flash Array	MCR	STOP

¹ Only the PIT timers are disabled by MDIS. The RTI is not affected by MDIS.

3.5.2 Halt Clock Gating

System clock gating is forced via the centralized halt mechanism. The SIU_HLT register's bits corresponding to individual modules are configured to determine which modules are clock gated.

The HLT bits are used to drive the stop inputs to the modules. After the module completes a clean shutdown, the module asserts the stop acknowledge handshake. The stop acknowledge is visible in the SIU_HLTACK read-only register bits. The modules are individually controlled and halted.

The halted module recovers when the HLT bit is cleared by software. After HLT is cleared, the device's logic will re-enable the clocks to the modules and negate the stop signal after the required timing has been met.

There is no hardware disable for the eDMA and FlexRay modules. Thus before setting the HLT bits for these masters, software should take actions to prepare for the eDMA and FlexRay clocks to be stopped. Then software sets the HLT bits for the eDMA and FlexRay to indicate to the clock logic that the clocks to these modules can now be stopped.

When the HLT bits for the eDMA and FlexRay are set and when the Z0 and Z1 have executed WAIT instructions, then the clocks to the platform are also gated. The platform logic includes the MPU, AXBS, AIPS, and MCM. The INTC and SIU are not clock gated to allow for an interrupt to be used to exit WAIT.

3.5.3 Core WAIT Clock Gating

Core clock gating is enabled via the CPU WAIT instruction (or, if the core is in reset, by the CRP Core Reset bit).

The Z1 and Z0 cores may be idled by their WAIT instructions. The WAIT instructions are used as a power-saving feature to halt the core. Executing the WAIT instruction puts the corresponding core in an idle state at a clean transition point. When the core stops, clocks to the core are gated off, and the core

asserts a signal indicating it is waiting for an interrupt. The state of this signal is software accessible via the appropriate SIU_HLTACK register's bits.

An interrupt to the corresponding core exits the WAIT instruction and the core continues to the appropriate interrupt service routine (ISR).

3.6 Alternate Module Clock Domains

3.6.1 FlexCAN Clock Domains

The FlexCAN blocks have two distinct software-controlled clock domains. One of the clock domains is always derived from the system clock. This clock domain includes the message buffer logic. The source for the second clock domain can be the system clock or the XOSC output. The logic in the second clock domain controls the CAN interface pins. The CLK_SRC bit in the FlexCAN CTRL register selects between the system clock and the oscillator clock as the clock source for the second domain. Selecting the oscillator as the clock source ensures low jitter on the CAN bus. System software can gate both clocks by writing to the MDIS bit in the FlexCAN MCR register.

NOTE

To prevent improper FlexCAN behavior when switching of the system clock or the CAN protocol engine clock source, or before the desired clock source has stabilized, the FlexCAN module must first be disabled by setting the FlexCAN_x_MCR[MDIS] = 1.

If the oscillator clock source is selected, the frequency of the peripheral clock needs to be the same or greater than the oscillator clock frequency.

If the XOSC is used as the system clock source and is divided down, then the clock source selected for the CAN interface must be the system clock (i.e. divided XOSC) to keep the system clock not slower than the CAN interface clock.

3.6.2 FlexRay Clock Domains

The FlexRay block has two distinct software-controlled clock domains. One of the clock domains is always derived from the system clock. The source for the second clock domain can be the system clock or the XOSC output. The logic in the second clock domain controls the FlexRay interface pins. The CLK_SRC bit in the FlexRay CTRL register selects between the system clock and the oscillator clock as the clock source for the second domain. Selecting the oscillator as the clock source ensures low jitter on the FlexRay bus.

NOTE

To prevent improper FlexRay behavior, the system clock or the FlexRay protocol engine clock source must be switched and stable before enabling the FlexRay module. After it is enabled, the FlexRay module can be disabled only by asserting $\overline{\text{RESET}}$.

If the oscillator clock source is selected for the FlexRay interface, then a divided down XOSC cannot be selected as the source for the system clock.

3.6.3 RTC Clock Domain

The clock source for the RTC can be selected as one of the following: the 32 kHz IRC, the 32 kHz OSC, or the 16 MHz IRC.

NOTE

To prevent improper real-time clock (RTC) behavior when switching the system clock source, or before the desired clock source has stabilized, the RTC must first be disabled by clearing the `CRP_RTCSC[CNTEN] = 0`.

3.6.4 SWT Clock Domain

The clock source for the SWT is selectable as the system clock or the 16 MHz IRC.

NOTE

To prevent improper software watchdog timer (SWT) behavior when switching the system clock source, or before the desired clock source has stabilized, the SWT must first be disabled by clearing the `MCM_MSWTCR[SWE] = 0`.

Chapter 4

Frequency Modulated Phase Locked Loop (FMPLL)

4.1 Introduction

The FMPLL module is a frequency modulated phase-locked loop that has been optimized to generate voltage controlled oscillator (VCO) frequencies from 192 MHz to 500 MHz based on an input clock range of 4 MHz to 40 MHz. The frequency multiplication, output dividers, and the frequency modulation waveform are register-programmable through a peripheral bus interface.

NOTE

Although this PLL is basically the same PLL that is used on other Power PC parts, its implementation is different, owing to the use of an internal 16 MHz IRC, low-power modes, and other features specific to the 5510 family.

4.1.1 Block Diagram

A simplified block diagram of the FMPLL illustrates the functionality and interdependence of major blocks (see Figure 4-1). Shaded blocks represent analog circuit components that make up the core analog portion of the FMPLL. The complete FMPLL closed-loop system contains the feedback divider (EMFD) and output divider (ERFD), which are implemented with standard cell core logic elements. Refer to Section 4.4.3.3, “PLL Normal Mode Without FM,” for details on each sub-block.

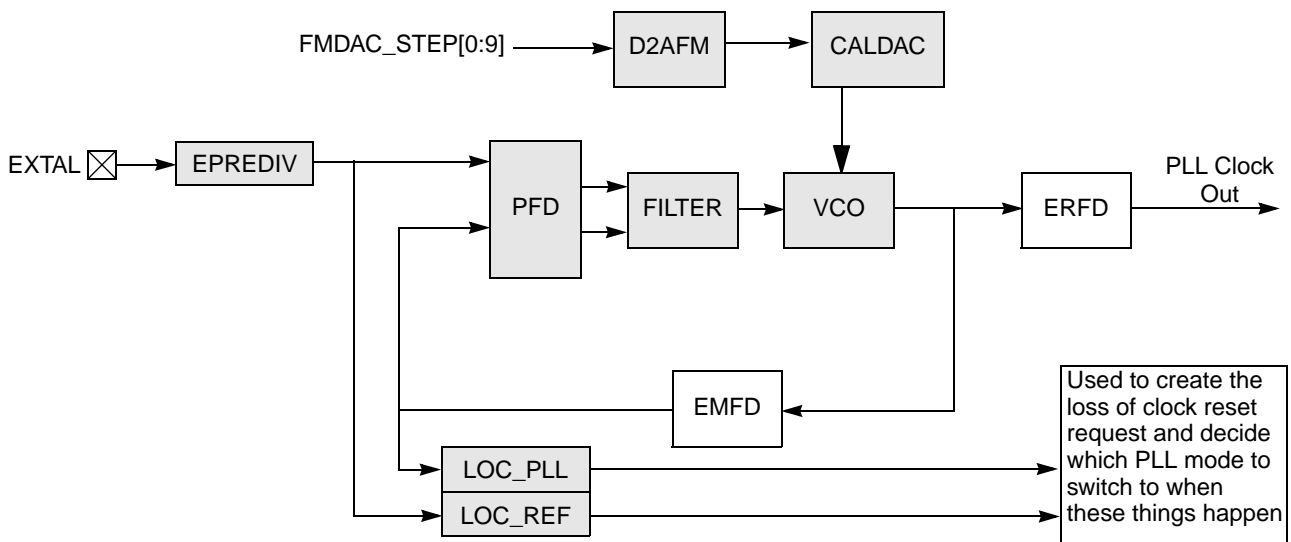


Figure 4-1. FMPLL Block Diagram

4.1.2 Features

The FMPLL has these major features:

- Input clock frequency range: 4 MHz to 40 MHz (EXTAL pin)
- Because the MPC5510 uses a 16 MHz IRC as its default system clock, the FMPLL will be put in PLL Off mode during reset, so that power dissipation is minimized by disabling the FMPLL until needed by the system.
- Programmable frequency multiplication factor settings generating VCO frequencies of 192 MHz – 500 MHz
- PLL Off mode (low-power stop)
- Register programmable output clock divider (ERFD)
- Programmable frequency modulation
 - Modulation applied as a triangle waveform
 - Peak-to-peak register programmable modulation depths of 0.5%, 1%, 1.5%, and 2% of the system frequency
 - Register programmable modulation rates of $F_{\text{extal}}/80$, $F_{\text{extal}}/40$, and $F_{\text{extal}}/20$
- Lock detect circuitry provides a signal indicating the FMPLL has acquired lock and continuously monitors the FMPLL output for any loss of lock
- Loss-of-clock circuitry monitors input reference and FMPLL output clocks with programmable ability to select a backup clock source as well as generate a reset or interrupt in the event of a failure

4.1.3 Modes of Operation

There are two main modes of FMPLL: PLL Off mode and normal mode. These modes are briefly described in this section.

When PLL Off mode is selected, the FMPLL is off, and the end-system user must have selected a different SIU MUX source than the PLL Output. The lock detector is not functional and will not indicate that the FMPLL is in a locked state. Frequency modulation is not available and the FMPLL is put into a low-power, idle state. This operating mode is described in [Section 4.4.2, “PLL Off Mode.”](#)

When normal mode is selected, the FMPLL is fully programmable. The FMPLL reference clock source can be a crystal oscillator or an external clock generator. The lock detector will function and indicate the lock status of the FMPLL and frequency modulation of the output clock can be enabled. This operating mode is described in [Section 4.4.3, “Normal Mode.”](#)

4.2 External Signal Description

Refer to [Table 2-1](#) and [Section 2.7, “Detailed External Signal Descriptions,”](#) for detailed signal descriptions.

4.3 Memory Map and Registers

This section provides a detailed description of all FMPLL registers.

4.3.1 Module Memory Map

Table 4-1 shows the FMPLL memory map. The address of each register is given as an offset to the FMPLL base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 4-1. FMPLL Memory Map

Offset from FMPLL_BASE_ADDR (0xFFFF_0000)	Register	Access	Reset Value	Section/Page
0x0000	Reserved			
0x0004	SYNSR—FMPLL Synthesizer Status Register	R/W	— ¹	4.3.2.1/4-3
0x0008	ESYNCR1—FMPLL Enhanced Synthesizer Control Register 1	R/W	0x8001_0053	4.3.2.2/4-5
0x000C	ESYNCR2—FMPLL Enhanced Synthesizer Control Register 2	R/W	0x0000_0005	4.3.2.3/4-8
0x0010–0x0014	Reserved			

¹ See specific register description.

4.3.2 Register Descriptions

This section lists the FMPLL registers in address order and describes the registers and their bit fields.

4.3.2.1 FMPLL Synthesizer Status Register (SYNSR)

FMOffset: PLL_BASE_ADDR + 0x0004

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	LOLF	LOC	MODE	PLL SEL	PLL REF	LOCKS	LOCK	LOCF	CAL DONE	CAL PASS
W							w1c							w1c		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-2. FMPLL Synthesizer Status Register (SYNSR)

Table 4-2. SYNSR Register Field Descriptions

Field	Description
bits 0–21	Reserved.

Table 4-2. SYNSR Register Field Descriptions (continued)

Field	Description
LOLF	<p>Loss-of-Lock Flag. This bit provides the interrupt request flag. To clear the flag, write a 1 to the bit. Writing 0 has no effect. This flag will not be set, and an interrupt will not be requested, if the loss-of-lock condition was caused by a system reset, enabling frequency modulation, or write to the ESYNCR1 which modifies the ESYNCR1[EMFD] bits. If the flag is set due to a system failure, writing the ESYNCR1[EMFD] bits or enabling FM will not clear the flag. Assert reset to clear the flag. If lock is reacquired, the bit will remain set until either a write 1 or reset is asserted.</p> <p>1 Interrupt service requested 0 Interrupt service not requested</p>
LOC	<p>Loss-Of-Clock Status. The LOC bit is an indication of whether a loss-of-clock condition is present when operating in normal PLL mode. If LOC=0, the system clocks are operating normally. If LOC=1, the system clocks have failed due to a reference failure or a PLL failure. If the read of the LOC bit and the loss-of-clock condition occur simultaneously, the bit does not reflect the current loss-of-clock condition. If a loss-of-clock condition occurs that sets this bit and the clocks later return to normal, this bit will be cleared. LOC is always zero in PLL Off mode.</p> <p>1 Clocks are not operating normally 0 Clocks are operating normally</p>
MODE	<p>Clock Mode. The initial value for the MODE bit is determined at reset. The state of this bit, along with PLLSEL and PLLREF, indicates which clock mode the PLL is operating in (see Table 4-3). The value of ESYNCR1[CLKCFG0] will be reflected in this location.</p> <p>1 PLL clock mode 0 PLL Off mode</p>
PLLSEL	<p>PLL Mode Select. The initial value for the PLLSEL bit is determined at reset. The state of this bit, along with MODE and PLLREF, indicates which mode the PLL operates in (see Table 4-3). This bit is cleared in PLL Off mode. The value of ESYNCR1[CLKCFG1] will be reflected in this location.</p> <p>1 Normal PLL mode 0 PLL Off mode</p>
PLLREF	<p>PLL Clock Reference Source. The initial value for the PLLREF bit is determined at reset. The state of this bit, along with MODE and PLLSEL, indicates which reference source has been chosen for normal PLL mode (see Table 4-3). This bit is cleared in PLL Off mode. The value of ESYNCR1[CLKCFG2] will be reflected in this location.</p> <p>1 Crystal clock reference chosen 0 External clock reference chosen</p> <p>Note: User must also use the XOSC bit in the CRP register (CRP_CLKSRC) to enable the 4 to 40 MHz oscillator.</p>
LOCKS	<p>Sticky PLL Lock Status Bit. The LOCKS bit is a sticky indication of PLL lock status. LOCKS is set by the lock detect circuitry when the PLL acquires lock after: 1) a system reset, or 2) a write to the ESYNCR2 which modifies the ESYNCR2[EMFD] bits, or 3) frequency modulation is enabled. Whenever the PLL loses lock, LOCKS is cleared. LOCKS remains cleared after the PLL re-locks, until one of the three conditions occurs. Furthermore, if the LOCKS bit is read when the PLL simultaneously loses lock, the bit does not reflect the current loss-of-lock condition.</p> <p>If operating in PLL Off mode, LOCKS remains cleared after reset.</p> <p>1 PLL has not lost lock since last system reset, a write to ESYNCR1 to modify the ESYNCR1[EMFD] bit field, or frequency modulation enabled 0 PLL has lost lock since last system reset, a write to ESYNCR1 to modify the ESYNCR1[EMFD] bit field, or frequency modulation enabled</p>

Table 4-2. SYNSR Register Field Descriptions (continued)

Field	Description
LOCK	PLL Lock Status Bit. The LOCK bit indicates whether the PLL has acquired lock. PLL lock occurs when the synthesized frequency matches to within approximately 0.75% of the programmed frequency. The PLL loses lock when a frequency deviation of greater than approximately 1.5% occurs. If the LOCK bit is read when the PLL simultaneously loses lock or acquires lock, the bit does not reflect the current condition of the PLL. If operating in PLL Off mode, LOCK remains cleared after reset. 1 PLL is locked 0 PLL is unlocked
LOCF	Loss-of-Clock Flag. This bit provides the interrupt request flag. To clear the flag, write a 1 to the bit. Writing 0 has no effect. Asserting reset will clear the flag. If clocks return to normal after the flag has been set, the bit will remain set until cleared by either writing 1 or asserting reset. A loss-of-clock condition can only be detected if LOCEN=1. 1 Interrupt service requested 0 Interrupt service not requested
CALDONE	Calibration Complete. The CALDONE bit is an indication of whether the calibration sequence has been completed since the last time modulation was enabled. If CALDONE=0 then the calibration sequence is in progress or modulation is disabled. If CALDONE=1 then the calibration sequence has been completed, and frequency modulation is operating. 1 Calibration complete 0 Calibration not complete
CALPASS	Calibration Passed. The CALPASS bit tells whether the calibration routine was successful. If CALPASS=1 and CALDONE=1 then the routine was successful. If CALPASS=0 and CALDONE=1, then the routine was unsuccessful. When the calibration routine is initiated the CALPASS is asserted. CALPASS remains asserted until modulation is disabled by clearing the EDEPTH bits in the ESYNCR2 register or a failure occurs within the FMPLL calibration sequence. 1 Calibration successful 0 Calibration unsuccessful If calibration is unsuccessful, then actual depth is not guaranteed to match the desired depth

Table 4-3. System Clock Status Per Mode

MODE	PLLSEL	PLLREF	Clock Mode
0	X	X	PLL Off mode
1	0	0	Reserved
1	1	0	Normal PLL mode with external clock reference
1	1	1	Normal PLL mode with crystal clock reference

4.3.2.2 FMPLL Enhanced Synthesizer Control Register 1 (ESYNCR1)

This is one of two FMPLL synthesizer control registers that are used to access enhanced features in the FMPLL. The bit fields in the ESYNCR1 behave as described in [Figure 4-3](#).

Frequency Modulated Phase Locked Loop (FMPLL)

FMOffset: PLL_BASE_ADDR + 0x0008

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	1	CLKCFG[0:2]			0	0	0	0	0	0	0	0	EPREDIV			
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	EMFD							
W																
Reset	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1

Figure 4-3. FMPLL Enhanced Synthesizer Control Register 1 (ESYNCR1)

Table 4-4. ESYNCR1 Register Field Descriptions

Field	Description
bit 0	Reserved. Note: This bit is set to 1 on reset and always reads as 1.
CLKCFG[0:2]	Clock Configuration. The CLKCFG[0:2] bits are writable versions of the MODE, PLLSEL, and PLLREF bits in the SYNSR. These change the clock mode, after reset has negated, via software. CLKCFG[0:2] map directly to MODE, PLLSEL, and PLLREF to control the system clock mode (see Table 4-3). Note: CLKCFG[0:2] = 0b101 can produce an unpredictable clock output. Note: The ESYNCR2[LOLRE] and ESYNCR2[LOCRES] should be set to 0 before changing the PLL mode, so that a reset is not immediately generated upon the write to CLKCFG[0:2]
bits 4–11	Reserved.
EPREDIV	Enhanced Pre-Divider. The EPREDIV bits control the value of the divider on the input clock. The output of the pre-divider circuit generates the reference clock to the PLL analog loop. The decimal equivalent of the EPREDIV binary number is substituted into the equation from Table 4-11 . Note: Setting the EPREDIV to any of the invalid states in Table 4-5 will cause the PLL to produce an unpredictable output clock, and the output frequency of the divider must equal the PLL reference frequency, f_{pllref} (see MPC5510 data sheet). When the EPREDIV bits are changed, the PLL will immediately lose lock. If the EPREDIV bits are changed during FM calibration, the current calibration sequence is terminated and the DEPTH bits are cleared. The PLL will re-lock to the new EPREDIV value you must manually re-enable modulation. To prevent an immediate reset, clear the LOLRE bit before writing the EPREDIV bits. In PLL Off mode the EPREDIV bits have no affect. The available enhanced pre-divider ratios are given in Table 4-5 .

Table 4-4. ESYNCR1 Register Field Descriptions (continued)

Field	Description
bits 16–23	Reserved.
EMFD	<p>Enhanced Multiplication Factor Divider. The EMFD bits control the value of the divider in the PLL feedback loop. The value specified by the EMFD bits establish the multiplication factor applied to the reference frequency. The decimal equivalent of the EMFD binary number is substituted into the equation from Table 4-11 for F_{sys} to determine the equivalent multiplication factor. The range of settings is $32 \leq EMFD \leq 132$.</p> <p>Note: EMFD values less than 32 and greater than 132 are invalid and will cause the PLL to produce an unpredictable clock output. The VCO frequency must be within the f_{VCO} specification (see MPC5510 data sheet)</p> <p>When the EMFD bits are changed, the PLL loses lock. If the EMFD bits are changed during FM calibration, the current calibration sequence is terminated and the DEPTH bits are cleared. The PLL will re-lock to the new EMFD value you must manually re-enable modulation. To prevent an immediate reset, clear the LOLRE bit before writing the EMFD bits.</p> <p>In PLL Off mode the EMFD bits have no affect.</p> <p>Table 4-6 shows the available divide ratios.</p>

Table 4-5. Enhanced Pre-divider Ratios

EPREDIV	Input Divide Ratio (EPREDIV+1)
0000	1
0001	2 (default for MPC5510)
0010	3
0011	4
0100	5
0101	6
0110	Invalid
0111	8
1000	Invalid
1001	10
1010–1111	Invalid

Table 4-6. Enhanced Feedback Divide Ratios

EMFD	Feedback Divide Ratio (EMFD+16)
0000_0000–0001_1111	Invalid
0010_0000	48
0010_0001	49
0010_0010	50
0010_0011	51
0010_0100	52

Table 4-6. Enhanced Feedback Divide Ratios

EMFD	Feedback Divide Ratio (EMFD+16)
0010_0101	53
0101_0011	99 (default for MPC5510)
1000_0100	148
1000_0101–1111_1111	Invalid

4.3.2.3 FMPLL Enhanced Synthesizer Control Register 2 (ESYNCR2)

This is the second of two enhanced versions of the FMPLL synthesizer control register used to access enhanced features in the FMPLL. The bit fields in the ESYNCR2 behave as described in [Figure 4-4](#).

FMOffset: PLL_BASE_ADDR + 0x000C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	LOCEN	LOLRE	LOCRE	LOL IRQ	LOC IRQ	0	ERATE	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	EDEPTH			0	0	ERFD					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Figure 4-4. FMPLL Enhanced Synthesizer Control Register 2 (ESYNCR2)

Table 4-7. ESYNCR2 Field Descriptions

Field	Description
bits 0–7	Reserved.
LOCEN	Loss-of-Clock Enable. The LOCEN bit determines whether the loss-of-clock function is operational along with backup clock modes, and interrupt and reset functions. See Section 4.4.3.2, “Loss-of-Clock Detection,” for more information. In PLL Off mode, this bit has no affect. LOCEN does not affect the loss-of-lock circuitry. 1 Loss-of-clock enabled. 0 Loss-of-clock disabled.
LOLRE	Loss-of-Lock Reset Enable. The LOLRE bit determines how the integration module handles a loss-of-lock indication. See Section 4.4.3.1, “PLL Lock Detection,” for more information. When operating in normal PLL mode, the PLL must be locked before setting the LOLRE bit. Otherwise reset is immediately asserted. The LOLRE bit has no affect in PLL Off mode. 1 Assert reset on loss of lock enabled. 0 Assert reset on loss of lock disabled.

Table 4-7. ESYNCR2 Field Descriptions (continued)

Field	Description
LOCRE	<p>Loss-of-Clock Reset Enable. The LOCRE bit determines how the integration module handles a loss-of-clock condition when LOCEN is equal to 1. LOCRE has no effect when LOCEN is equal to 0.</p> <p>If the LOCF bit in the SYNSR indicates a loss-of-clock condition, setting the LOCRE bit causes an immediate reset.</p> <p>In PLL Off mode LOCRE has no affect.</p> <p>1 Assert reset on loss of clock enabled 0 Assert reset on loss of clock disabled.</p>
LOLIRQ	<p>Loss-of-Lock Interrupt Request. The LOLIRQ bit determines how the integration module handles a loss-of-lock indication. See Section 4.6.1, “Loss-of-Lock Interrupt Request,” for more information.</p> <p>When operating in normal mode, the PLL must be locked before setting the LOLIRQ bit. Otherwise an interrupt is immediately requested.</p> <p>The LOLIRQ bit has no affect in PLL Off mode.</p> <p>1 Request interrupt enabled 0 Request interrupt disabled</p>
LOCIRQ	<p>Loss- of-Clock Interrupt Request. The LOCIRQ bit determines how the integration module handles a loss-of-clock condition when LOCEN=1. LOCIRQ has no effect when LOCEN=0.</p> <p>If the LOCF bit in the SYNSR indicates a loss-of-clock condition, setting (or having previously set) the LOCIRQ bit causes an interrupt request.</p> <p>In PLL Off mode LOCIRQ has no affect.</p> <p>1 Request interrupt on loss of clock enabled. 0 Request interrupt on loss of clock disabled</p>
bit 13	Reserved.
ERATE	Enhanced Modulation Rate. The ERATE bits control the rate of frequency modulation applied to the system frequency. Table 4-8 shows the allowable modulation rates.
bits 16–20	Reserved.
EDEPTH	Enhanced Modulation Depth. The EDEPTH bit field controls the frequency modulation depth and enables the frequency modulation. When programmed to a value other than 0x0 the frequency modulation is automatically enabled. Table 4-9 shows are the programmable frequency deviations from the system frequency. Upon a change in the depth value to other than 0x0, the calibration sequence will be re initialized.
bits 24–25	Reserved.
ERFD	<p>Enhanced Reduced Frequency Divider. The ERFD bits control a divider at the output of the PLL. The value specified by the ERFD bits establish the divisor applied to the PLL frequency. The ERFD divides the output clock by the quantity (ERFD + 1). Even-numbered RFD settings, which would result in odd divide ratios, are not allowed.</p> <p>The decimal equivalent of the ERFD binary number is substituted into the equation from Table 4-11.</p> <p>Note: The ERFD divides the output clock by the quantity (ERFD + 1). Even numbered ERFD settings, which would result in odd divide ratios, are invalid and cause the PLL to produce an unpredictable output clock. The PLL output clock must be within the f_{pll} specification (see MPC5510 data sheet).</p> <p>Changing the ERFD bits does not affect the PLL, hence, no re-lock delay is incurred. Resulting changes in clock frequency are synchronized to the next falling edge of the current system clock. These bits should be written only when the lock bit (LOCK) is set, to avoid surpassing the allowable system operating frequency. In PLL Off mode the ERFD bits have no affect.</p> <p>The available enhanced output divider ratios are given in Table 4-10.</p>

Table 4-8. Programmable Modulation Rates

ERATE	Modulation Rate (Hz)
00	$F_{\text{mod}} = F_{\text{extal}}/80$
01	$F_{\text{mod}} = F_{\text{extal}}/40$
10	$F_{\text{mod}} = F_{\text{extal}}/20$
11	Invalid

Table 4-9. Programmable Modulation Depths

EDEPTH	Modulation Depth (% of F_{sys})
000	0
001	0.25% – 0.5%
010	0.75% – 1.0%
011	1.25% – 1.5%
100	1.75% – 2.0%
101 – 111	Reserved

Table 4-10. Enhanced Output Divide Ratios

ERFD	Output Divide Ratio (ERFD+1)
00_0000	1
00_0001	2
00_0010	Invalid
00_0011	4
00_0100	Invalid
00_0101	6 (default value for MPC5510)
00_0110	Invalid
00_0111	8
.	.
.	.
.	.
11_1100	Invalid
11_1101	62
11_1110	Invalid
11_1111	64

4.4 Functional Description

The FMPLL module contains the frequency modulated phase lock loop (FMPLL), enhanced frequency divider (ERFD), enhanced synthesizer control registers (ESYNCR1 and ESYNCR2), synthesizer status register (SYNSR), and clock/PLL control logic. The block also contains a reference frequency pre-divider controlled by the EPREDIV bits in the ESYNCR1. This enables the user to use a high frequency crystal or external clock generator and obtain finer frequency synthesis resolution than would be available if the raw input clock were used directly by the analog loop. For the remainder of this chapter, the term “reference frequency” and the symbol F_{ref} indicate the output of the pre-divider circuit. This is the clock on which frequency multiplication will be performed.

4.4.1 General

At reset, the system clock is driven by the internal oscillator (16 MHz IRC) and the module is in bypass mode. After reset, software can change the PLL mode (see [Section 4.5.1, “Clock Mode Selection”](#)).

[Table 4-11](#) shows the PLL-clock to input-clock frequency relationships for the available clock modes.

Table 4-11. Clock-Out vs. Clock-In Relationships

Clock Mode	Frequency Equation
Normal PLL Mode	$F_{\text{sys}} = \frac{F_{\text{extal}} \cdot (\text{EMFD} + 16)}{(\text{EPREDIV} + 1)(\text{ERFD} + 1)}$

4.4.2 PLL Off Mode

When PLL Off mode is selected, the PLL is off and either the 16 MHz IRC must be selected as the system clock or the user must supply an external clock or crystal on the EXTAL pin, and select that clock source before entering PLL Off mode. The selected clock is directly used to produce the various system clocks. Refer to *MPC5510 Microcontroller Family Data Sheet* for external clock input requirements. In bypass mode, the analog portion of the PLL is disabled, the frequency modulation capability is not available, and no clocks are generated at the PLL output. The pre-divider is bypassed and has no effect on the system clock frequency in bypass mode.

4.4.3 Normal Mode

When normal PLL mode is selected, the PLL is fully programmable. The PLL can synthesize frequencies ranging from 48x to 148x the reference frequency of the output of the predivider, with or without frequency modulation enabled. The post-divider is capable of reducing the PLL clock frequency without forcing a re-lock. The PLL reference can be a crystal oscillator reference or an external clock reference. This clock will be divided by the pre-divider circuit to create the PLL reference clock.

4.4.3.1 PLL Lock Detection

The lock detect logic monitors the reference frequency and the PLL feedback frequency to determine when frequency lock has been achieved. Phase lock is inferred by the frequency relationship, but is not

guaranteed. The PLL lock status is reflected in the LOCK status bit in the SYNSR. A sticky lock status indication, LOCKS, is also provided.

The lock detect function uses two counters, which are clocked by the reference and PLL feedback respectively. When the reference counter has counted N cycles, the feedback counter's count is compared. If the feedback counter has also counted N cycles, the process is repeated for N + K counts. Then if the two counters' counts match, the lock criteria is relaxed by one count and the system is notified that the PLL has achieved frequency lock. Then takes three successful compares before tolerance is relaxed.

After lock has been detected, the lock circuitry continues to monitor the reference and feedback frequencies using the alternate count and compare process. If the counters do not match at any comparison time, then the LOCK status bit is cleared to indicate that the PLL has lost lock. At this point, the lock criteria is tightened and the lock detect process is repeated.

The alternate count sequences prevent false lock detects due to frequency aliasing while the PLL tries to lock. Alternating between a tight and relaxed lock criteria prevents the lock detect function from randomly toggling between locked and not locked status due to phase sensitivities. Figure 4-5 illustrates the sequence for detecting locked and not-locked conditions.

When the frequency modulation is enabled, the loss of lock continues to function as described but with the lock and loss of lock criteria reduced to ensure that false loss of lock conditions are not detected.

In PLL Off mode, the PLL cannot lock because the PLL is disabled.

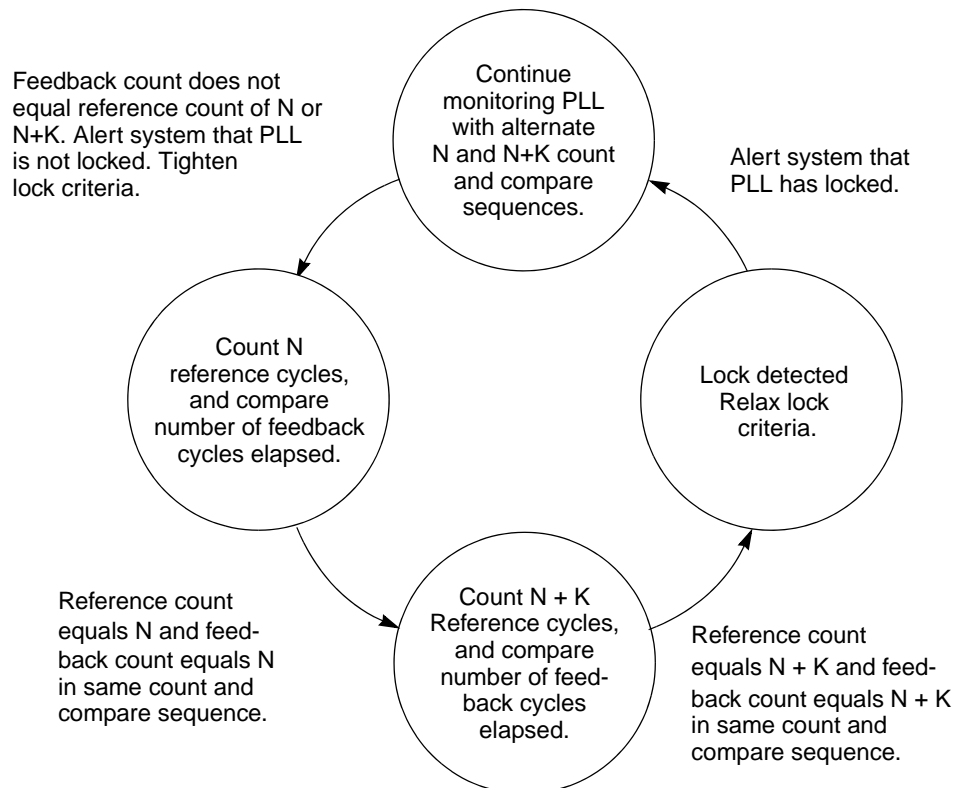


Figure 4-5. Lock Detect Sequence

After the PLL acquires lock after reset, the LOCK and LOCKS status bits are set. If the EPREDIV or EMFD are changed, or if an unexpected loss-of-lock condition occurs, the LOCK and LOCKS status bits are negated. While the PLL is in an unlocked condition, the system clocks continue to be sourced from the PLL as the PLL attempts to re-lock. Consequently, during the re-locking process, the system clock frequency is not well defined and may exceed the maximum system frequency violating the system clock timing specifications. Because of this condition, using the loss-of-lock reset function is recommended.

After the PLL has re-locked, the LOCK bit is set. The LOCKS bit remains cleared if the loss of lock was unexpected. The LOCKS bit is set to one when the loss of lock was caused by changing the EPREDIV or EMFD fields.

4.4.3.2 Loss-of-Clock Detection

When enabled by the LOCEN bit in the ESYNCR2, the loss-of-clock (LOC) detection circuit monitors the input clocks to the phase/frequency detector (PFD) (see [Figure 4-1](#)). When the reference or feedback clock frequency falls below a minimum frequency, the LOC circuitry considers the clock to have failed and a loss-of-clock status is reflected by the sticky LOCF bit, and non-sticky LOC bit in the SYNSR. See *MPC5510 Microcontroller Family Data Sheet* for the minimum clock frequency. In PLL Off mode, the loss-of-clock circuitry is disabled.

Depending on which clock source has failed, the LOC circuitry switches the PLL's output clock source to the remaining operational clock, if enabled by LOCEN. The PLL's output clocks are derived from the alternate clock source until reset is asserted. If the reference fails, the PLL goes out of lock and into self-clocked mode (SCM) (see [Table 4-12](#)). The PLL remains in SCM until the next reset. When the PLL is operating in SCM, the PLL will run open loop at a default VCO frequency. The RFD will set to divide-by-4 to ensure the clock presented to the system will be well below the maximum allowable frequency for the device. If the loss-of-clock condition is due to a PLL failure (i.e., loss of feedback clock), the PLL reference becomes the system clocks source until the next reset, even if the PLL regains itself and re-locks.

Table 4-12. Loss-of-Clock Summary

Clock Mode	System Clock Source before Failure	REFERENCE FAILURE Alternate Clock Selected by LOC Circuitry until Reset	PLL FAILURE Alternate Clock Selected by LOC Circuitry until Reset
PLL	PLL	PLL self-clocked mode	PLL reference
PLL bypass	Ext. Clock(s)	None	NA

Note: The LOC circuit monitors the inputs to the PFD: reference and feedback clocks (see [Figure 4-1](#)).

A special loss-of-clock condition occurs when both the reference and the PLL fail. The failures may be simultaneous or the PLL may fail first. In either case, the reference clock failure takes priority and the PLL attempts to operate in SCM. If successful, the PLL remains in SCM until the next reset. During SCM, modulation is always disabled. If the PLL cannot operate in SCM, the system remains static until the next reset. If a loss-of-clock reset is enabled, the reset switches the bus clocks over to the 16 MHz IRC (and switches off the PLL).

4.4.3.3 PLL Normal Mode Without FM

In PLL mode, the system clocks are synthesized by the FMPLL by multiplying up the reference clock frequency. It is critical that the system clock frequency remain within the range for the device (see *MPC5510 Microcontroller Family Data Sheet*). The output of the FMPLL can be divided down in powers of two up to 128 to reduce the system frequency with the ERFD. The ERFD is not contained in the feedback loop of the PLL, so changing the ERFD bits does not affect FMPLL operation. Finally, the PLL can be frequency modulated to reduce electromagnetic interference often associated with clock circuitry. [Figure 4-1](#) shows the overall block diagram for the PLL. Each of the major blocks is discussed briefly in the following sections.

4.4.3.3.1 Phase/Frequency Detector

The phase/frequency detector (PFD) is a dual-latch phase-frequency detector. It compares both the phase and frequency of the reference clock and the feedback clock. The reference clock comes from the crystal oscillator or an external clock source. The feedback clock comes from the VCO output divided down by the EMFD in normal PLL mode.

When the frequency of the feedback clock equals the frequency of the reference clock (i.e. the PLL is frequency locked), the PFD will pulse the UP or DOWN signals depending on the relative phase of the two clocks. If the falling edge of the reference clock leads the falling edge of the feedback clock, then the UP signal is pulsed. If the falling edge of the feedback clock leads the falling edge of the reference clock, then the DOWN signal is pulsed. The width of these pulses relative to the reference clock is dependent on how much the two clocks lead or lag each other. After phase lock is achieved, the PFD continues to pulse the UP and DOWN signals for a very short duration during each reference clock cycle. These short pulses force the PLL to continually update and prevent a frequency drift phenomena referred to as “dead-banding.” Dead-band describes the minimum amount of phase error between the reference and feedback clocks that a phase detector cannot correct.

4.4.3.3.2 Charge Pump/Loop Filter

Operation of the charge pump is controlled by the UP and DOWN signals from the PFD. They control whether the charge pumps apply or remove charge, respectively, from the loop filter.

4.4.3.3.3 VCO

The voltage into the VCO controls the frequency of its output. The frequency-to-voltage relationship (VCO gain) is positive.

4.4.3.3.4 EMFD

The MFD divides down the output of the VCO and feeds it back to the PFD. The PFD controls the VCO frequency (via the charge pump and loop filter) such that the reference and feedback clocks have the same frequency and phase. Thus, the input to the MFD, which is also the output of the VCO, is at a frequency that is the reference frequency multiplied by the same amount the MFD divides by. For example, if the MFD divides the VCO frequency by 48, then the PLL will be frequency locked when the VCO frequency is 48 times the reference frequency. The presence of the MFD in the loop allows the PLL to perform frequency multiplication, or synthesis.

4.4.3.3.5 Programming System Clock Frequency

In normal PLL clock mode, the default system frequency is determined by the default EPREDIV, EMFD, and ERFD values.

When programming the PLL, do not to violate the maximum system clock frequency or max/min VCO frequency specifications. Based on the desired system clock frequency, EPREDIV, EMFD, and ERFD must be calculated for the given crystal or external reference frequency. See *MPC5510 Microcontroller Family Data Sheet* for the max/min VCO frequency range and the maximum allowable system frequency.

Frequency modulation should be disabled prior to changing the EPREDIV, EMFD, or RATE bit fields. After enabling frequency modulation a new calibration sequence is performed. A change to EPREDIV, EMFD, DEPTH, or RATE while modulation is enabled will invalidate the previous calibration results.

Use these directions to accommodate the frequency overshoot that occurs when the EPREDIV or EMFD bits are changed. If frequency modulation is going to be enabled the maximum allowable frequency must be reduced by the programmed ΔF_m .

1. Determine the appropriate value for the EPREDIV, EMFD, and ERFD fields in the synthesizer control register(s), remember to include the ΔF_m if frequency modulation is to be enabled. The amount of jitter in the system clocks can be minimized by selecting the maximum EMFD factor that can be paired with an ERFD factor to provide the desired frequency. The maximum EMFD value that can be used is determined by the VCO and EMFD range.
2. Write a value of ERFD = ERFD (from step 1) + 1 to the ERFD field of the ESYNCR2. Not increasing the ERFD when changing the EPREDIV or EMFD could subject the device to clock frequencies beyond the range specified for the device due to the PLL's unlocked state.
3. If frequency modulation is currently enabled, disable it by writing 00 to the EDEPTH field of the ESYNCR2.
4. If programming the EPREDIV and/or EMFD, write the value(s) determined in step 1 to the appropriate field(s) in the ESYNCR1.
5. Monitor the synthesizer lock bit (LOCK) in the synthesizer status register (SYNSR). When the PLL achieves lock, write the ERFD value determined in step 1 to the ERFD field of the ESYNCR2. This changes the system clocks frequency to the desired frequency. If frequency modulation is desired, leave ERFD programmed to ERFD + 1 until after completing the steps in [Section 4.4.3.4.2, "Programming System Clock Frequency With Frequency Modulation."](#)
6. If frequency modulation was enabled initially, it can be re-enabled following the steps listed in [Section 4.4.3.4.2, "Programming System Clock Frequency With Frequency Modulation."](#)

4.4.3.4 PLL Normal Mode With Frequency Modulation

In normal PLL clock mode, frequency modulation is not enabled in the default synthesis mode. When frequency modulation is enabled two parameters must be set to generate the desired level of modulation. The parameters to be programmed are the RATE and DEPTH bit fields of the ESYNCR2 register. The RATE bit controls the frequency of modulation, F_{mod} . The DEPTH bits work to control the modulation depth, F_m . The available modulation rates and depths are given in [Table 4-8](#) and [Table 4-9](#), respectively. The modulation waveform is always a triangle wave and its shape is not programmable. An example of one period of the modulation waveform is shown in [Figure 4-6](#).

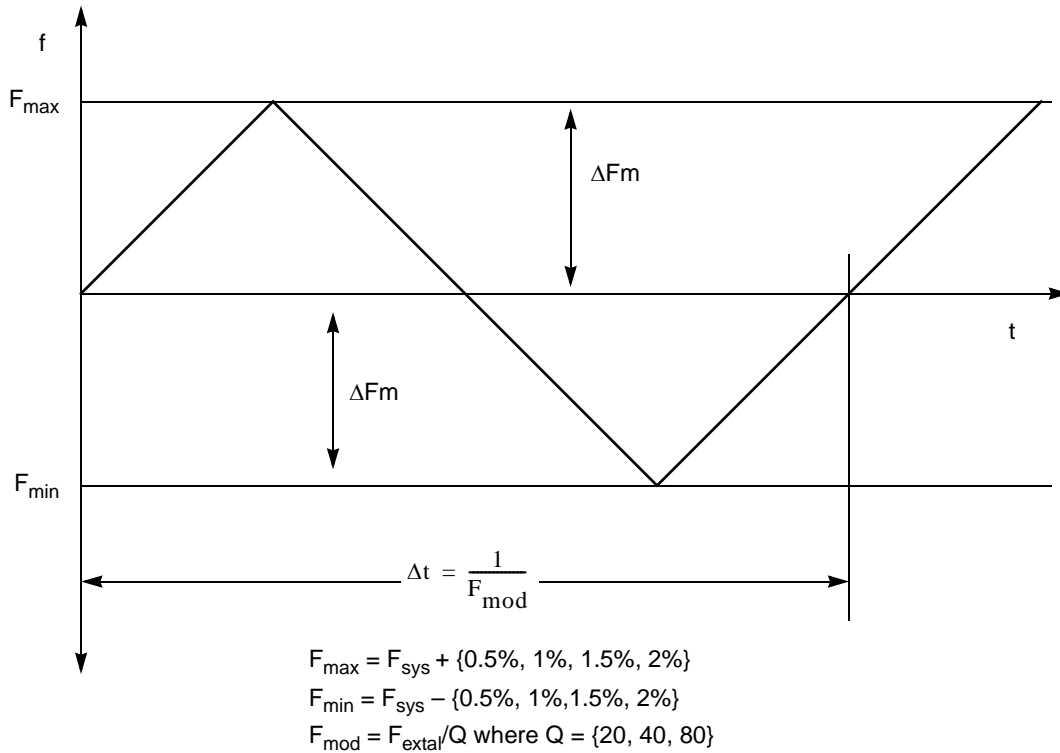


Figure 4-6. Frequency Modulation Waveform

4.4.3.4.1 Frequency Modulation Depth Calibration

The frequency modulation calibration system tunes a reference current into the modulation D/A so that the modulation depth (F_{\max} and F_{\min}) remains within specification. Disable frequency modulation prior to changing the EPREDIV, EMFD, or ERATE bit fields. Upon enabling frequency modulation a new calibration sequence is performed. A change to EPREDIV, EMFD, or ERATE while modulation is active will invalidate calibration results.

This routine will correct for process variations, but because temperature can change after the calibration has been performed, variation due to temperature drift is not eliminated. This system is also voltage dependent, so if the supply changes after the sequence takes place, error incurred will not be corrected. The calibration system reuses the two counters in the lock detect circuit, the reference and feedback counters. The reference counter remains clocked by the reference clock, but the feedback counter is clocked by the VCO clock.

When the calibration routine is initiated by writing to the EDEPTH bits, the CALPASS and CALDONE status bits are immediately cleared.

When calibration is induced the VCO is given time to settle before the feedback and reference counters start counting. Full VCO clock cycles are counted by the feedback counter during this time to give the initial center frequency count. When the reference counter has counted to the programmed number of reference count cycles, the input to the feedback counter is disabled and the result is placed in the COUNT0 register. The calibration system then enables modulation at programmed ΔF_m and the VCO gets time to settle. Both counters are reset and restarted. The feedback counter begins to count full VCO clock

cycles again to obtain the delta-frequency count. The counter will run only during the high phase of the triangular modulation waveform. Several half-modulation periods will be measured during the calibration routine to increase the resolution of the frequency measurement. This will result in a measurement of the average frequency during the high phase of the modulation waveform which under ideal circumstances will be equivalent to one-half of the desired modulation depth. When the reference counter has counted to the new programmed number of reference count cycles, the feedback counter is stopped again.

The delta-frequency count minus the center frequency count (COUNT0) results in a delta count proportional to the reference current into the modulation D/A. That delta count is subtracted from the expected value for the selected depth resulting in an error count. The sign of this error count determines the direction taken by the calibration D/A to update the calibration current. After obtaining the error count for the present iteration, both counters are cleared. The stored count of COUNT0 is preserved while a new feedback count is obtained, and the process to determine the error count is repeated. The calibration system repeats this process eight times, once for each bit of the calibration D/A.

After the last decision is made, a 1 is written to the CALDONE bit of the SYNSR. If an error occurs during the calibration routine, then CALPASS remains 0. If the routine completed successfully, CALPASS is set to 1.

4.4.3.4.2 Programming System Clock Frequency With Frequency Modulation

The following steps illustrate proper programming of the frequency modulation mode. These steps ensure proper operation of the calibration routine and prevent frequency overshoot from the sequence. The PLL should be programmed and allowed to lock in non-FM mode at the desired frequency as outlined in [Section 4.4.3.3.5, “Programming System Clock Frequency.”](#)

1. Monitor LOCK bit. Do not proceed until the PLL is locked in non-modulation mode.
2. Write a value of $ERFD = ERFD + 1$ to the ERFD field of the ESYNCR2 to ensure the maximum system frequency is not exceeded during the calibration routine. This should have been done when allowing the PLL to lock in non-FM mode.
3. Program the desired modulation rate and depth to the ERATE and EDEPTH fields in the ESYNCR2. This action initiates the calibration sequence.
4. Allow time for the calibration sequence. Wait for the PLL to lock (i.e. the LOCK bit to set in the SYNSR). At this time CALDONE should be asserted. CALPASS will be asserted if the calibration was successful. If not, the calibration can be re-initiated by repeating from step 3. When the PLL achieves lock, write the ERFD value desired.

The frequency modulation system is dependent on several factors. The accuracies of the VDDSYN/VSSSYN voltage, of the crystal oscillator frequency, and of the manufacturing variation.

For example, if a 5 percent accurate supply voltage is used, then a 5 percent modulation depth error will result. If the crystal oscillator frequency is skewed from the nominal operating frequency, the resulting modulation frequency will be proportionally skewed. Finally, the error due to the manufacturing and environment variation alone can cause the frequency modulation depth error to be greater than 20 percent.

4.5 Resets

This section describes the reset operation of the PLL, including power-on reset and normal resets. The reset values of registers and signals are provided in other sections.

4.5.1 Clock Mode Selection

The initial clock mode is reflected in the MODE, PLLSEL, and PLLREF bits of the synthesizer status register (SYNSR) as well as the ESYNCR1[CLKCFG] bit field. The clock mode can be modified by writing to the CLKCFG bit field. The synthesizer status register will then reflect the newly-selected PLL clock mode. [Table 4-13](#) shows the clock mode encoding.

The clock mode selection configuration is summarized in [Table 4-13](#).

Table 4-13. Clock Mode Selection

Clock Mode	Synthesizer Status Register (SYNSR) MODE, PLLSEL, and PLLREF Bits		
	MODE/ CLKCFG2	PLLSEL/ CLKCFG1	PLLREF/ CLKCFG0
Bypass mode	0	X	X
Normal mode with external reference	1	1	0
Normal mode with crystal reference	1	1	1
Reserved	1	0	0

4.5.1.1 Power-On Reset (POR)

The PLL will not operate until the POR signal has negated and the CLKCFG set for PLL mode. Refer to *MPC5510 Microcontroller Family Data Sheet* for these thresholds. At this point, the PLL will operate in self-clocked mode (SCM) until a valid reference clock is detected by the internal clock monitor circuit.

Internal to the PLL, the VCO will be held in reset until the negation of the POR signal. This prevents the PLL from attempting to lock before its supplies are within specification which can cause VCO/loop gain to be lower than what the analog loop is designed for.

4.5.1.2 External Reset

After POR has negated, the PLL defaults to Bypass mode and the default clock source for the system clock is the 16 MHz IRC. After reset exit, the PLL may be configured for operation and after lock may be selected as the system clock source.

After the initial lock with the default MFD (assuming normal mode was selected), ESYNCR1 may be written to modify the MFD for the desired operating frequency. The PLL might not lock with an MFD and crystal frequency combination that attempts to force the VCO outside its operating range.

CAUTION

When running in an unlocked state, the clocks generated by the PLL are not guaranteed stable and may exceed the maximum specified operating frequency of the device. The RFD should always be used as described in [Section 4.4.3.3.5, “Programming System Clock Frequency,”](#) to insulate the system from any potential frequency overshoot of the PLL clocks.

4.5.2 PLL Loss-of-Lock Reset

By programming the LOLRE bit in the ESYNCR2, the PLL can assert reset when a loss-of-lock condition occurs. Because the LOCK and LOCKS bits in the SYNSR are re-initialized after reset, the SIU reset status register (SIU_RSR) ([Section 6.3.2.2, “Reset Status Register \(SIU_RSR\)”](#)) must be read to determine a loss-of-lock condition occurred.

In PLL Off mode, the PLL cannot lock; therefore a loss-of-lock condition cannot occur and LOLRE has no affect.

4.5.3 PLL Loss-of-Clock Reset

When a loss-of-clock condition is recognized, $\overline{\text{RESET}}$ is asserted if the LOCRE bit in the SYNCR is set. The LOCF and LOC bits in the SYNSR are cleared after reset, therefore, the LOC bit must be read in the SIU_RSR to determine that a loss-of-clock condition occurred. LOCRE has no affect in PLL Off mode.

4.6 Interrupts

This section describes the interrupt requests that the PLL can generate.

4.6.1 Loss-of-Lock Interrupt Request

By setting the LOLIRQ bit in the ESYNCR2, the PLL can request an interrupt when a loss-of-lock condition occurs.

In PLL Off mode, the PLL cannot lock; therefore a loss-of-lock condition cannot occur and the LOLIRQ has no affect.

4.6.2 Loss-of-Clock Interrupt Request

When a loss-of-clock condition is recognized, the PLL will request an interrupt if the LOCIRQ bit in the SYNCR is set. The LOCIRQ bit has no affect in bypass mode or if LOCEN is equal to 0.

Chapter 5

Clock, Reset, and Power Control (CRP)

5.1 Introduction

The primary function of the clock, reset, and power (CRP) block is to maintain all of the control logic that requires power when other portions of the SoC are powered down in power-saving modes. The CRP manages entry into, operation during, and exit from power-saving modes.

The CRP consists of the input isolation block, the RTC/API, the wakeup and power status block, the clock and reset control block, low-power state machine, and bus interface unit. The input isolation block allows inputs from external blocks to be driven to known states when the logic driving the input is powered down. The RTC/API block implements a real-time counter and periodic interrupt. The wakeup and power status block implements the logic to select power mode operation and wakeup sources. The clock and reset control block implements miscellaneous logic related to PLL and oscillator operation, and reset gating for power-saving modes. The low-power state machine controls the transitions into and out of the power-saving modes. The bus interface unit allows for slave read/write register access from the device's core. There are also several miscellaneous integration functions included in the CRP that are discussed in detail in later sections of this chapter.

5.1.1 Block Diagram

A simplified block diagram of the CRP illustrates the functionality and interdependence of major blocks (see [Figure 5-1](#)).

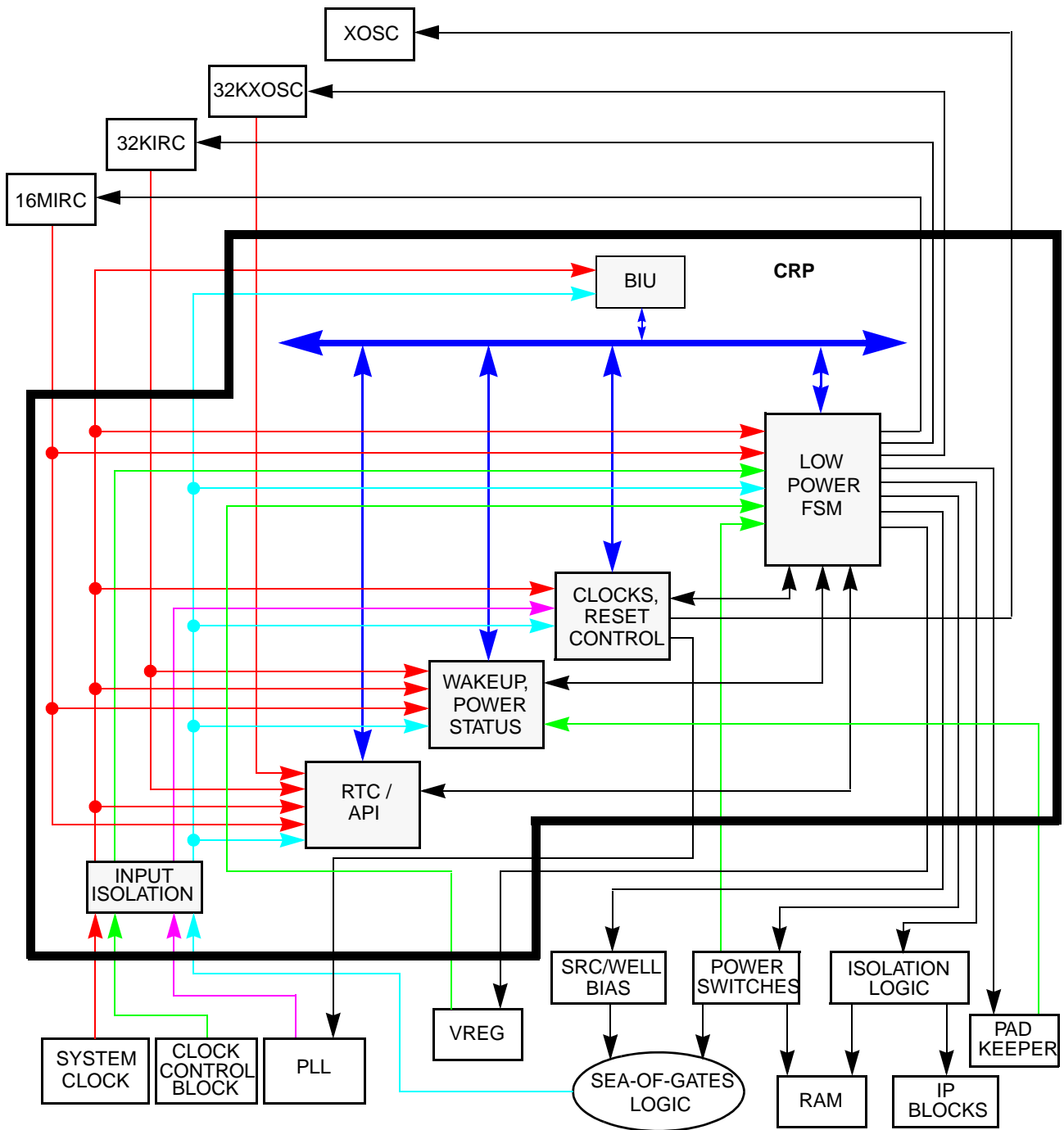


Figure 5-1. CRP Block Diagram

5.1.2 Features

The CRP has these major features:

- Real-time clock (RTC/API):
 - 32-bit counter
 - Four selectable counter clock sources
 - Fixed divide by 32 prescaler to provide 1.0 ms resolution with 32 kHz clock source
 - Fixed divide by 512 prescaler feeding the divide by 32 prescaler to provide 1.0 ms resolution with 16 MHz clock source
 - Option to bypass the divide by 512 prescaler with the 16 MHz clock source
 - 12-bit RTC compare, with minimum 1 second resolution (2 ms resolution with bypassed 16 MHz clock source)
 - 10-bit API compare with minimum 1 ms resolution (2 us resolution with bypassed 16 MHz clock source)
 - API compare value can be modified while RTC is running
 - Optional low-power wakeup and/or interrupt for RTC match, API match, and RTC rollover
 - Counters and dividers can be disabled to minimize power consumption
- Low-power mode management:
 - Provides control of voltage regulator, LVI circuits, isolation enables, power switches, and pin output state retention for both sleep and stop modes
 - FSM clock gates itself off when waiting for asynchronous wakeup signal for power savings
 - Eight selections available for blocks sizes for RAM data retention
- Low-power wakeup:
 - Wakeup sources can be either the RTC, API, RTC rollover, or external pin
 - All wakeup sources can be enabled at any given time (first to occur generates wakeup)
 - Eight pin-wakeup sources can be selected from 64 pins total (eight groups of eight)
 - Pin wakeup occurs on either rising edge, falling edge or both
 - Two clock-source inputs for pin wakeup to allow for lower power or faster wakeup
 - System level reset control to ensure clean recovery from sleep and stop modes
- Miscellaneous:
 - All functional logic inputs isolated in low-power modes
 - All logic with multiple clock sources internally synchronized
 - All CRP logic is reset asynchronously, but exits reset synchronously

5.1.3 Modes of Operation

There are three functional modes of operation for the CRP: normal operation, sleep mode, and stop mode. In normal operation, all CRP registers can be read or written. The input isolation, low-power FSM, and wakeup logic is disabled. The voltage regulator, LVI, and power switch outputs are in the enabled state. The RTC/API and associated interrupts are optionally enabled. In sleep and stop modes, the bus interface

is disabled and the input isolation is enabled. The RTC/API is enabled if enabled prior to entry into sleep and stop. The voltage regulator, LVI, and power switch control are dependent on whether in sleep or stop mode (see [Section 5.3, “Functional Description.”](#))

5.2 Memory Map and Registers

This section provides a detailed description of all CRP registers.

5.2.1 Module Memory Map

The CRP memory map is shown in [Table 5-1](#). The address of each register is given as an offset to the CRP base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 5-1. CRP Memory Map

Offset from CRP_BASE (0xFFFFE_C000)	Register	Access	Reset Value	Section/Page
0x0000	CRP_CLKSRC — Clock Source Register	R/W	0x0004_DF8F	5.2.2.1/5-5
0x0004–0x000F	Reserved			
0x0010	CRP_RTCSC — RTC Status and Control Register	R/W	0x0000_0000	5.2.2.2/5-6
0x0014	CRP_RTCCNT — RTC Counter Register	R	0x0000_0000	5.2.2.3/5-8
0x0018–0x003F	Reserved			
0x0040	CRP_WKPINSEL — Wakeup Pin Source Select Register	R/W	0x0000_0000	5.2.2.4/5-8
0x0044	CRP_WKSE — Wakeup Source Enable Register	R/W	0x0000_0000	5.2.2.5/5-9
0x0048–0x004F	Reserved			
0x0050	CRP_Z1VEC — Z1 Reset Vector Register	R/W	0xFFFF_FFFD	5.2.2.6/5-10
0x0054	CRP_Z0VEC — Z0 Reset Vector Register	R/W	0xFFFF_FFFE	5.2.2.7/5-11
0x0058	CRP_RECPTN — Recovery Pointer Register	R/W	0xFFFF_FFFC	5.2.2.8/5-12
0x005C–0x005F	Reserved			
0x0060	CRP_PSCR — Power Status and Control Register	R/W	0x0000_0000	5.2.2.9/5-13
0x0064–0x006F	Reserved			
0x0070	CRP_SOCSC — SoC Status and Control Register	R/W	0x0000_0000	5.2.2.10/5-15

5.2.2 Register Descriptions

This section lists the CRP registers in address order and describes the registers and their bit fields.

5.2.2.1 Clock Source Register (CRP_CLKSRC)

The CRP_CLKSRC contains:

- enable bits for the 32 kHz IRC (32KIRC), the 32 kHz XOSC (32KOSC), and the main external oscillator (XOSC)
- the trim values for the 16 MHz IRC and 32 kHz IRC

Offset: CRP_BASE + 0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	32KIRC	XOSC	0	32KOSC
W													EN	EN		EN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0 ¹	1	0	0 ¹
	16	17	18	19	20	21	22	23	24 ²	25 ²	26 ²	27	28	29	30	31
R	TRIM32IRC[0:7]								TRIMIRC[0:7]							
W																
Reset	1 ²	1 ²	0	1	1	1	1	1	1	0	0	0	1	1	1	1

¹ These bits are only reset by power-on, VDD15 LVI, VDD33 LVI, VDDSYN LVI, VDD5 Low LVI, and VDD5 LVI.

² These bits must not be changed.

³ These bits must remain set to a value of 1. Only the six least significant bits of TRIM32IRC are used.

Figure 5-2. Clock Source Register (CRP_CLKSRC)

Table 5-2. CRP_CLKSRC Field Descriptions

Field	Description
bits 0–11	Reserved.
IRC32KEN	32 kHz IRC Enable. The IRC32KEN bit enables the 32K IRC. 0 32 kHz IRC disabled 1 32 kHz IRC enabled
XOSCEN	External Oscillator Enable. The XOSCEN bit enables the external oscillator. 0 XOSC disabled. 1 XOSC enabled. Note: During sleep and stop mode with XOSCEN=1, the XOSC will still actively drive an external crystal but the XOSC clock to the system is disabled.
bit 14	Reserved.
OSC32KEN	32 kHz OSC Enable. The OSC32KEN bit enables the 32K oscillator. 0 32K OSC disabled 1 32K OSC enabled Note: After enabling the 32K OSC, software needs to wait the required crystal startup/stabilization time before making use of the 32K OSC.

Table 5-2. CRP_CLKSRC Field Descriptions (continued)

Field	Description
TRIM32IRC[0:7] ¹	Trim Value for 32 kHz IRC. The TRIM32IRC bits control the 32 kHz IRC internal reference clock frequency by controlling the internal reference clock period. The bits' effect are binary weighted (i.e. bit 6 adjusts twice as much as bit 7). Increasing the binary value decreases the period and decreasing the value increases the period. Note: A trim value of 0xff is reserved and is not a valid trim value.
TRIMIRC[0:7] ¹	Trim Value for 16 MHz IRC. The TRIMIRC bits control the 16 MHz IRC internal reference clock frequency by controlling the internal reference clock period. The bits' effect are binary weighted (i.e. bit 6 adjusts twice as much as bit 7). Increasing the binary value decreases the period and decreasing the value increases the period. Bits 0–2 control the bandgap voltage trim. Note: Do not change bits 0–2 to any values other than the Factory Trim values or the default reset values. Bits 3–7 control the IRC current reference.

¹ See Chapter 22, "Flash Array and Control" for factory trim value locations in memory.

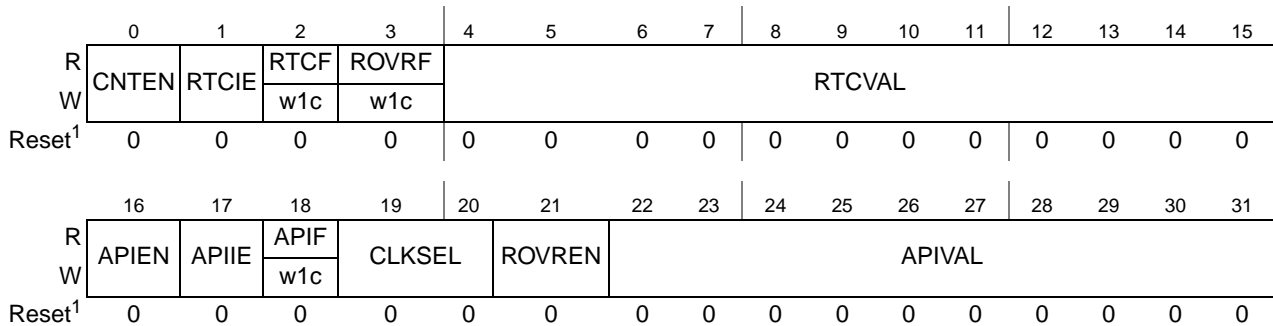
5.2.2.2 RTC Status and Control Register (CRP_RTCSC)

The CRP_RTCSC register contains:

- RTC counter enable
- RTC interrupt enable
- RTC interrupt flag
- RTC counter roll over interrupt flag
- RTC clock source select
- RTC compare value
- API enable
- API interrupt enable
- API interrupt flag
- API compare value

Offset: CRP_BASE + 0x0010

Access: User read/write



¹ These bits are only reset by power-on, VDD15 LVI, VDD33 LVI, and VDDSYN LVI, VDD5 low LVI, and VDD5 LVI.

Figure 5-3. RTC Status and Control Register (CRP_RTCSC)

Table 5-3. CRP_RTCSC Field Descriptions

Field	Description
CNTEN	Counter Enable. The CNTEN bit enable the RTC counter. CNTEN asserted has the effect of asynchronous resetting (synchronous reset negation) all the RTC logic. This allows for the RTC configuration and clock source selection to be updated without causing synchronization issues. 0 Counter disabled 1 Counter enabled
RTCIE	RTC Interrupt Enable. The RTCIE bit enables interrupts requests to the system if RTCF is asserted. 0 RTC interrupts disabled 1 RTC interrupts enabled
RTCF	RTC Interrupt Flag. The RTCF bit indicates that the RTC counter has reached the counter value matching RTCVAL. RTCF is cleared by writing a 1 to RTCF. Writing a 0 to RTCF has no effect. Note that the RTCF bit must be cleared before entering SLEEP or STOP mode, if the RTC is to be used as the wakeup source. 0 No RTC interrupt 1 RTC interrupt
ROVRF	Counter Roll Over Interrupt Flag. The ROVRF bit indicates that the RTC has rolled over from 0xFFFF_FFFF to 0x0000_0000. ROVRF is cleared by writing a 1 to ROVRF. Writing a 0 to ROVRF has no effect. Note that the ROVRF bit must be cleared before entering SLEEP or STOP mode, if the RTC rollover is to be used as the wakeup source. 0) RTC has not rolled over 1) RTC has rolled over
RTCVAL	RTC Compare Value. The RTCVAL bits are compared to bits 10–21 of the RTC counter and if match sets RTCF. RTCVAL may only be updated when CNTEN is 0.
APIEN	Autonomous Periodic Interrupt Enable. The APIEN bit enables the autonomous periodic interrupt function. 0 API disabled 1 API enabled
APIIE	API Interrupt Enable. The APIIE bit enables interrupts requests to the system if APIF is asserted. 0 API interrupts disabled 1 API interrupts enabled
APIF	API Interrupt Flag. The APIF bit indicates that the RTC counter has reached the counter value matching API offset value. APIF is cleared by writing a 1 to APIF. Writing a 0 to APIF has no effect. Note that the APIF bit must be cleared before entering SLEEP or STOP mode, if the API is to be used as the wakeup source. 0 No API interrupt 1 API interrupt.
CLKSEL	Clock Select. The CLKSEL bits select the clock source for the RTC. CLKSEL may be updated when CNTEN is 0 only. Note: The 32 kHz IRC or 32 kHz OSC are not automatically enabled if selected; therefore, they must be enabled before either one is selected for use. 00 32 kHz IRC 01 32 kHz OSC 10 16 MHz IRC with 512 prescaler divide 11 16 MHz IRC without 512 prescaler divide

Table 5-3. CRP_RTCSK Field Descriptions (continued)

Field	Description
ROVREN	Counter Roll Over Interrupt Enable. The ROVREN bit enables wakeup and/or interrupt requests when the RTC has rolled over from 0xFFFF_FFFF to 0x0000_0000. The RTCIE bit must also be set in order to generate an interrupt request on a counter rollover. 0) RTC rollover wakeup/interrupts disabled 1) RTC rollover wakeup/interrupt enabled
APIVAL	API Compare Value. The APIVAL bits are compared to an offset value based on bits 22–31 of the RTC counter and if match asserts a interrupt/wakeup request. APIVAL may only be updated when APIEN is 0 or API function is undefined. Note: The compare value will be the number in the API + 1. Numbers less than 3 should not be used as synchronization requires up to 2 clocks.

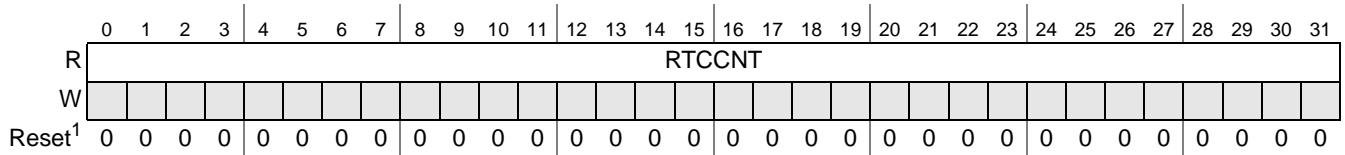
5.2.2.3 RTC Counter Register (CRP_RTCCNT)

The CRP_RTCCNT register contains:

- RTC counter value

Offset: CRP_BASE + 0x0014

Access: User read only



¹ These bits are only reset by power-on, VDD15 LVI, VDD33 LVI, and VDDSYN LVI, VDD5 low LVI, and VDDLVI.

Figure 5-4. RTC Counter Register (CRP_RTCCNT)

Table 5-4. CRP_RTCCNT Field Descriptions

Field	Description
RTCCNT	RTC Counter Value. The RTCCNT bits reflect the current value of the RTC counter.

5.2.2.4 Wakeup Pin Source Select Register (CRP_WKPINSEL)

The CRP_WKPINSEL register has eight fields, each of which controls which external pin will be used as one of the eight external wakeup sources.

Offset: CRP_BASE + 0x0040

Access: User read/write

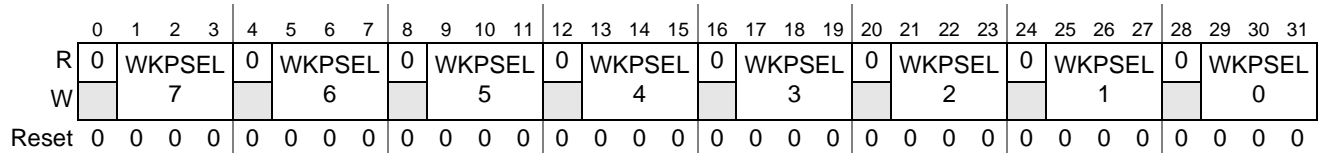


Figure 5-5. Wakeup Pin Source Select Register (CRP_WKPINSEL)

Table 5-5. CRP_WKPINSEL Field Descriptions

Field	Description
bits 0, 4, 8, 12, 16, 20, 24, 28	Reserved.
WKPSELn[0:2]	Wakeup Pin Source Select. The WKPSELn[0:2] bits select the external pin to be used as one of the eight external pin wakeup sources (see Table 5-6).

Table 5-6. Wakeup Source Selects

	111	110	101	100	011	010	001	000
WKPSEL0	PG11	PD15	PD10	PD0	PC2	PB13	PA4	PA0
WKPSEL1	PJ12	PG15	PD14	PD13	PC4	PB15	PA5	PA1
WKPSEL2	PF10	PD6	PD1	PC0	PB8	PB5	PA6	PA2
WKPSEL3	PG6	PD8	PD7	PC5	PC1	PB14	PB10	PA3
WKPSEL4	PH5	PH4	PG12	PG5	PD12	PD5	PB9	PB6
WKPSEL5	PH8	PH7	PG10	PF12	PE2	PD2	PB7	PA7
WKPSEL6	PH9	PG13	PG7	PF14	PF13	PD9	PD3	PC6
WKPSEL7	PH6	PG14	PG9	PF15	PF11	PD11	PD4	PB12

NOTE

Program any pins that are to be used as wakeup sources as inputs in the associated SIU_PCRx register prior to entering a low-power mode.

5.2.2.5 Wakeup Source Enable Register (CRP_WKSE)

The CRP_WKSE register contains:

- Wakeup source enables
- Wakeup clock select

Offset: CRP_BASE + 0x0044

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R	WKPDET7				WKPDET6		WKPDET5		WKPDET4		WKPDET3		WKPDET2		WKPDET1		WKPDETO
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	0	0	0	0	0	RTCOVR EN	RTCWK EN	APIWK EN	0	0	0	0	0	0	0	0	WKCLK SEL
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-6. Wakeup Source Enable Register (CRP_WKSE)

Table 5-7. CRP_WKSE Field Descriptions

Field	Description
WKPDEn	Wakeup Pin Edge Detection Select. The WKPDEn bits enable the external pin wakeup sources and define which edge transition is used for the wakeup. (The corresponding inputs must be enabled through the SIU registers, to allow them to be used as wakeup sources.) These bits map directly to WKPSELn bits in the CRP_WKPINSEL register. 00 External pin wakeup source disabled 01 Positive edge of selected external pin triggers the wakeup request 10 Negative edge of selected external pin triggers the wakeup request 11 Positive or negative edge of selected external pin triggers the wakeup request
bits 16–20	Reserved.
RTCOVREN	RTC Rollover Wakeup Enable. The RTCOVREN bit enables a rollover of the RTC counter to be a wakeup source for exit from low-power modes. 0 RTC rollover will not generate a wakeup request from low-power mode 1 RTC rollover will generate a wakeup request from low-power modes.
RTCWKEN	RTC Wakeup Enable. The RTCWKEN bit enables the RTC to be a wakeup source for exit from low-power modes. 0 RTC not enabled as a wakeup source 1 RTC enabled as a wakeup source
APIWKEN	API Wakeup Enable. The APIWKEN bit enables the API to be a wakeup source for exit from low-power modes. 0 The API will not generate a wakeup request from low-power mode 1 The API will generate a wakeup request from low-power modes.
bits 24–30	Reserved.
WKCLKSEL	Wakeup Clock Select. The WKCLKSEL bit selects the clock source used for the wakeup logic synchronizer and edge detect. WKCLKSEL should be switched only when all wakeup sources are disabled. 0 Clock source for wakeup logic is the 32 kHz IRC. 1 Clock source for wakeup logic is the 16 MHz IRC. Note: The 32 kHz IRC is not automatically enabled if selected; therefore, it must be enabled before it is selected for use. Note: When using the 32 kHz IRC to wake up from SLEEP, the application software must wait at least one 32 kHz clock cycle after entering SLEEP before waking up. Note: The wakeup flag cannot be cleared until at least three 32 kHz cycles after it has been set.

5.2.2.6 Z1 Reset Vector Register (CRP_Z1VEC)

The CRP_Z1VEC register contains:

- Recovery vector for the Z1 core
- Reset for the Z1 core
- VLE select for the Z1 core

Offset: CRP_BASE + 0x0050

Access: User read/write



Figure 5-7. Z1 Reset Vector Register (CRP_Z1VEC)

Table 5-8. CRP_Z1VEC Field Descriptions

Field	Description
Z1VEC	Z1 Recovery Vector. The Z1VEC value determines the initial program counter for the Z1 upon exiting reset. On POR, the value contained in the register defaults to 0xFFFF_FFFD, so that the Z1 fetches VLE code from the BAM starting at address 0xFFFF_FFFC. The user may change this value to point to a different memory location for system reinitialization upon low-power sleep mode exit.
30 Z1RST	Controls the assertion of RESET to the Z1 core. Writes to this bit cause the Z1 to immediately enter/exit reset. Reads of this bit indicate if the core is being held in reset. 0 Z1 not in reset 1 Z1 in reset
31 VLE	VLE Select. The VLE bit selects whether the Z1 recovers into VLE or Book E mode. 0 Book E 1 VLE

NOTE

The user may attempt to set both the CRP_Z1VEC[Z1RST] and CRP_Z0VEC[Z0RST] bits to 1, but if one of these bits is already set to a value of 1, the write to the other bit will be blocked.

If both cores are running, either core can stop the other core by writing to the other core’s reset bit.

5.2.2.7 Z0 Reset Vector Register (CRP_Z0VEC)

The CRP_Z0VEC register contains:

- Recovery vector for the Z0 core
- Reset for the Z0 core

Offset: CRP_BASE + 0x0054

Access: User read/write

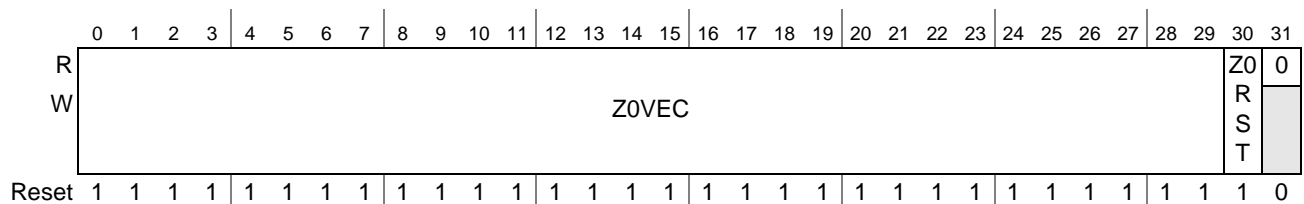


Figure 5-8. Z0 Reset Vector Register (CRP_Z0VEC)

Table 5-9. CRP_Z0VEC Field Descriptions

Field	Description
Z0VEC	Z0 Recovery Vector. The Z0VEC value determines the initial program counter for the Z0 upon exiting reset. On POR, the value contained in the register defaults to 0xFFFF_FFFE, and the Z0 is held in reset. Change this value to point to a different memory location for Z0 specific initialization upon negation of the Z0RST bit, or to a location for the Z0 to start running code when exiting Low Power modes (if it was not in RESET before entering the low power mode).
Z0RST	Controls the assertion of RESET to the Z0 core. Writes to this bit cause the Z0 to immediately enter/exit reset. Reads of this bit indicate if the core is being held in reset. 0 Z0 not in reset. 1 Z0 in reset.
bit 31	Reserved.

NOTE

The user may attempt to set the CRP_Z1VEC[Z1RST] and CRP_Z0VEC[Z0RST] bits to 1, but if one of these bits is already set to a value of 1, the write to the other bit will be blocked.

If both cores are running, either core can stop the other core by writing to the other core’s reset bit.

5.2.2.8 Reset Recovery Pointer Register (CRP_RECPTTR)

The CRP_RECPTTR register contains:

- Recovery pointer
- Fast recovery enable

Offset: CRP_BASE + 0x0058

Access: User read only

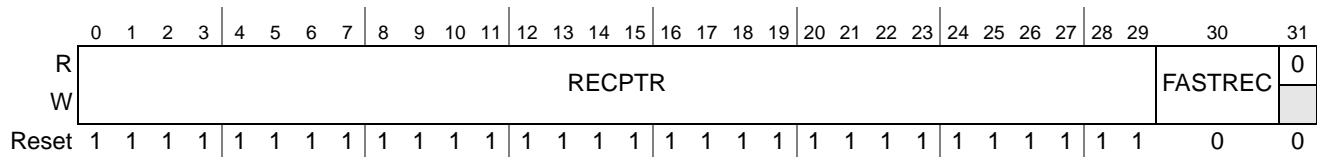


Figure 5-9. Reset Recovery Pointer (CRP_RECPTTR)

Table 5-10. CRP_RECPTTR Field Descriptions

Field	Description
RECPTTR	Recovery Pointer. The RECPTTR value is a generic 30 bit register available to the user application which retains a value during all low-power modes. This register may be used by the user software to indicate where in RAM a recovery routine exists. On reset, this register defaults to 0xFFFF_FFFC so that it points to the same location as the Z1VEC and Z0VEC registers.

Table 5-10. CRP_RECPTN Field Descriptions (continued)

Field	Description
FASTREC	Fast Reset Recovery. Allows the reset sequence generated at the exit of a sleep mode to be shortened to 64 clocks. This bit may be used when the Z1VEC or Z0VEC register of the core(s) executing code after a sleep mode points to a memory other than the flash. This allows code to be executed from those other memories while the flash completes its internal initialization. 0 Reset occurs for 2400 or 9600 clocks, depending on PLL configuration 1 Reset occurs for 64 clocks
bit 31	Reserved.

5.2.2.9 Power Status and Control Register (CRP_PSCR)

The power status and control register (CRP_PSCR) contains:

- Wakeup mode and source flags
- Pin wakeup selects
- Sleep and stop mode enables
- Pad keeper release
- Sleep RAM retention select

Offset: CRP_BASE + 0x0060

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLEEP	STOPF	0	0	0	WKRLOVRF	WKAPIF	WKRT CF	PWKSRCF							
W	w1c	w1c				w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLEEP	STOP	0	0	SLP12EN	RAMSEL			PWKSRIE[0:7]							
W				PKREL												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-10. Power Status and Control Register (CRP_PSCR)

Table 5-11. CRP_PSCR Field Descriptions

Field	Description
SLEEPF	SLEEP Flag. The SLEEPF bit indicates whether recovery from the last low-power modes was sleep. A write of 1 clears this status flag and a write of 0 has no effect. 0 Last low-power mode was not sleep 1 Last low-power mode was sleep
STOPF	STOP Flag. The STOPF bit indicates whether recovery from the last low-power modes was stop. A write of 1 clears this status flag and a write of 0 has no effect. 0 Last low-power mode was not stop 1 Last low-power mode was stop
bits 2–4	Reserved
WKRLLOVRF	Counter Rollover Wakeup Flag. The WKRLLOVRF bit indicates that a RTC/API counter rollover was the wakeup source. A write of 1 clears the interrupt flag and a write of 0 has no effect. 0 The RTC/API counter did not cause the last wakeup 1 The RTC/API counter caused the last wakeup
WKAPIF	API Wakeup Flag. The WKAPIF bit indicates the API was the wakeup source. A write of 1 clears the interrupt flag and a write of 0 has no effect. 0 The API did not cause the last wakeup 1 The API caused the last wakeup
WKRTCF	RTC Wakeup Flag. The WKRTCF bit indicates that the RTC was the wakeup source. A write of 1 clears the interrupt flag and a write of 0 has no effect. 0 The RTC did not cause the last wakeup 1 The RTC caused the last wakeup
PWKSRCF	Pin Wakeup Source Flag. The PWKSRCF bits indicate which external pin wakeup source event caused the wakeup. More than one external wakeup source can be asserted at a time if the wakeup events happened simultaneously. The eight wakeup sources are ORed together at chiptop to present one interrupt request to the cores. A write of 1 clears the interrupt flag and a write of 0 has no effect. 0 PWKSRCF[x] did not cause the last wakeup 1 PWKSRCF[x] caused the last wakeup
SLEEP	SLEEP Request. The SLEEP bit indicates a request to enter the sleep low-power mode. This bit is cleared automatically upon exit from SLEEP. 0 No request to enter the sleep low-power mode 1 Request to enter the sleep low-power mode Note: If SLEEP and STOP are set at the same time, the SLEEP bit has priority.
STOP	STOP Request. The STOP bit indicates a request to enter the stop low-power mode. This bit is cleared automatically upon exit from STOP. 0 No request to enter the stop low-power mode 1 Request to enter the stop low-power mode Note: If SLEEP and STOP are set at the same time, the SLEEP bit has priority.
bit 18	Reserved.
PKREL	Pad Keeper Release. The PKREL bit releases the held I/O states by the pad keepers after recovery from a low-power sleep mode. The PKREL bit is write only and always reads 0. 0 No effect 1 The I/O states held by the pad keepers are released back to normal functions

Table 5-11. CRP_PSCR Field Descriptions (continued)

Field	Description
SLP12EN	SLP12EN Sleep 1.2V Enable. The SLP12EN bit enables the use of the 1.2V internal regulator during Sleep mode instead of the default 1.5V internal regulator for the VDD supply. 0 Sleep 1.2V regulator disabled during Sleep mode. 1 Sleep 1.2V regulator enabled during Sleep mode.
RAMSEL	RAM Selects. The RAMSEL bits select which ram configuration retains power during the sleep mode. 000 All RAMs powered down 001 8K RAM retains power (0x4000_0000 - 0x4000_1FFF) 010 16K RAM retains power (0x4000_0000 - 0x4000_3FFF) 011 32K RAM retains power (0x4000_0000 - 0x4000_7FFF) 110 64K RAM retains power (0x4000_0000 - 0x4000_FFFF) 111 80K RAM retains power (0x4000_0000 - 0x4001_3FFF) Other reserved (defaults to 80K on MPC5510).
PWKSRIE[0:7]	Pin Wakeup Source Interrupt Enable. The PWKSRIE bits enable interrupt requests to the system if the corresponding PWKSRCF bit is asserted. (Note: PWKSRIEn = WKPSELn, n = 0 to 7.) 0 Wakeup source interrupt disabled 1 Wakeup source interrupt enabled

5.2.2.10 SoC Status and Control Register (CRP_SOCSC)

The CRP_SOCSC register contains:

- LVI interrupt flags
- LVI interrupt enables
- LVI reset enables
- LVI lock bit

Offset: CRP_BASE + 0x0070

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LVI5IE	LVI5HIE	LVI5F	LVI5HF	LVI5 LOCK	LVI5RE	0	0	0	0	0	0	0	0	0	BYP
W			w1c	w1c												DIS
Reset	0	0	0	0	0 ¹	1 ¹	0	0	0	0	0	0	0	0	0	0 ²
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

¹ These bits are only reset by power on, VDD15 LVI, VDD33 LVI, VDDSYN LVI, and VDD5 Low LVI.

² These bits are only reset by power on, VDD15 LVI, VDD33 LVI, VDDSYN LVI, VDD5 Low LVI, and VDD5 LVI.

Figure 5-11. LVI Status and Control Register (CRP_SOCSC)

Table 5-12. CRP_SOCSC Field Descriptions

Field	Description
LVI5IE	LVI5 Interrupt Enable. TheLVI5IE bit enables interrupts requests to the system if LVI5F is asserted. 0 LVI5 interrupts disabled 1 LVI5 interrupts enabled
LVI5HIE	LVI5 High Interrupt Enable. TheLVI5HIE bit enables interrupts requests to the system if LVI5HF is asserted. 0 LVI5H interrupts disabled 1 LVI5H interrupts enabled
LVI5F	LVI 5V Interrupt Flag. The LVI5F bit indicates that the LVI5 LVI circuit has detected that the 5 V supply is below the defined nominal limit. LVI5F is cleared by writing a 1 to LVI5F. Writing a 0 to LVI5F has no effect. 0 No LVI5 interrupt 1 LVI5 interrupt
LVI5HF	LVI 5V High Interrupt Flag. The LVI5HF bit indicates that the LVI5H LVI circuit has detected that the 5V supply is below the defined high limit. LVI5HF is cleared by writing a 1 to LVI5HF. Writing a 0 to LVI5HF has no effect. 0 No LVI5H interrupt 1 LVI5H interrupt
LVI5LOCK	LVI5 Lock. The LVI5LOCK bit disables writes to the LVI5RE register bit. After it is set, this bit will remain set until the next POR. 0 LVI5RE writeable 1 LVI5RE not writeable
LVI5RE	LVI5 Reset Enable. The LVI5RE bit enables the reset function of the LVI5. 0 LVI5 does not generate a reset when LVI5F is set 1 LVI5 generates a reset when the LVI5F is set
bits 6–14	Reserved.
BYPDIS	REFBYPC Disable. The BYPDIS bit disables the REFBYPC pin which allows for a faster eQADC recovery time after exit from a low power stop or sleep mode. 0 REFBYPC enabled 1 REFBYPC disabled
bits 16–31	Reserved.

5.3 Functional Description

5.3.1 Low-Power Modes

The CRP support two low power modes of operation, SLEEP and STOP. The primary difference between these modes is the standard cell logic is powered down in SLEEP mode, but remains powered and static in STOP mode. In order to achieve the functional requirements of these low power modes, the CRP provides the following functionality: control of the on-chip voltage regulator, LVI circuits, power gates, and well/source bias circuitry; control of external pin output state retention circuitry; wakeup monitoring on external pins or internal RTC; external reset pin monitoring to allow user to abort the low power mode; system recovery on wakeup; and support for JTAG and Nexus debug capability. The following sections discuss in detail the entry sequence, the operation, and the exit sequence for the low power modes.

5.3.1.1 External Pin Configuration

The CRP enables external pin state retention in both SLEEP and STOP modes. The user must have the external I/O in the desired configuration prior to entry. The CRP only provides the controls to latch the current pin state, select the latched data instead of the normal output data, and enable isolation logic in the external pin retention state circuitry. The specific external pins and pad configurations controlled by the CRP signals is determined at the SoC level.

5.3.1.2 External SoC Debug Tool Configuration

For STOP mode, the SoC debug configuration must be complete prior to entering the mode. To wakeup from STOP mode with the pre-STOP debug capability enabled, the LP_DBG bit in the NPC PCR register must be set prior to halting the SoC cores and entering the mode.

For SLEEP mode, the SoC debug configuration is lost and must be restored after wakeup. However, the configuration of the external debug pins must be set prior to entering the mode in order to be used after wakeup. This includes setting the Nexus pin functionality in the NPC PCR register, the external pin multiplexing selection and electrical characteristics, etc.

5.3.2 Low-Power Mode Entry

The sequence to enter the low-power stop or sleep modes is for the user to disable the DMA and FlexRay masters. Then halt all modules via the SIU_HLT register. If desired, the RTI should be shut down, because it is not affected by the SIU_HLT bits. The system clock source should be set to the 16 MHz IRC prior to disabling the PLL or powering down the XOSC. The PLL should then be disabled since it does not clock any logic in sleep or stop modes.

The main external oscillator (XOSC) can be optionally powered down in sleep or stop modes by setting the XOSCEN bit in the CRP_CLKSRC register. The XOSC will be automatically disabled when entering sleep or stop mode if it is left powered up (XOSCEN=1). If the XOSC is powered down for either low-power mode, the crystal oscillator will have to start up again on the exit from the low-power mode. If the XOSC powered down option is chosen, the user must be sure to first disable any logic that is being clocked directly by the XOSC to prevent glitches.

All program and erase operations on the flash array need to be completed before entering sleep or stop modes. In order to reduce power consumption in stop mode, the flash array should be placed in stop mode via the STOP bit in the flash Module Configuration Register. Also, for minimum power consumption, the external VPP pin should be grounded in sleep mode, and at 3.3V in stop mode

Prior to entry into either low-power stop or sleep modes, the EQADC halt bit must be set or the EQADC must be disabled. Upon exit from the low-power mode, the required recovery time must elapse before the EQADC can be enabled or the EQADC halt bit is cleared. The recovery time allows the EQADC circuits to stabilize, and must include a time for the REFBYPC pin to charge if the REFBPYC is populated with a capacitor. See the Data Sheet for recovery times.

Sleep or stop mode selection is done by setting the appropriate bits in the CRP_PSCR register. With one or both bits set, to enter sleep or stop mode, each active core should individually execute the WAIT instruction. If only one core is active, and one is held in reset by the user, then executing the WAIT

instruction on the active core will initiate entry into the low-power mode. At this point, the CRP takes over operation of the SoC until a wakeup event occurs.

5.3.2.1 CRP Clock Selection

In both sleep or stop modes, the CRP control logic is clocked by the 16 MHz IRC. The RTC/API can be clocked by either the 32 kHz IRC, the 32 kHz XOSC, or the 16 MHz IRC. The pin wakeup logic can be clocked by either the 32 kHz IRC, or the 16 MHz IRC. These clock source selections must be made prior to executing both WAIT instructions to the cores.

5.3.2.2 Sleep Mode RAM Retention

The RAMSEL bits in the CRP_PSCR register determine the amount of RAM that remains powered in sleep mode. This selection must be made prior to executing the WAIT instructions to the cores with the CRP_PSCR[SLEEP] bit set.

5.3.3 Low-Power Operation

After the WAIT instructions have been executed with either the sleep or stop bit set, and the cores have cleanly halted, the clock control block signals the CRP to enter the selected low-power mode. Note that if both the sleep and stop bits are set, sleep will be the mode entered.

At this point, the CRP has complete control of the SoC. [Figure 5-12](#) gives the sequence to transition from RUN mode to SLEEP/STOP. [Figure 5-13](#) and [Figure 5-14](#) give the transition diagram for going from RUN mode to sleep, and then back to RUN mode. [Figure 5-15](#) and [Figure 5-16](#) give the same diagram for RUN mode to stop, and back to RUN mode. The CRP does not support going directly to/from Sleep mode from/to stop mode.

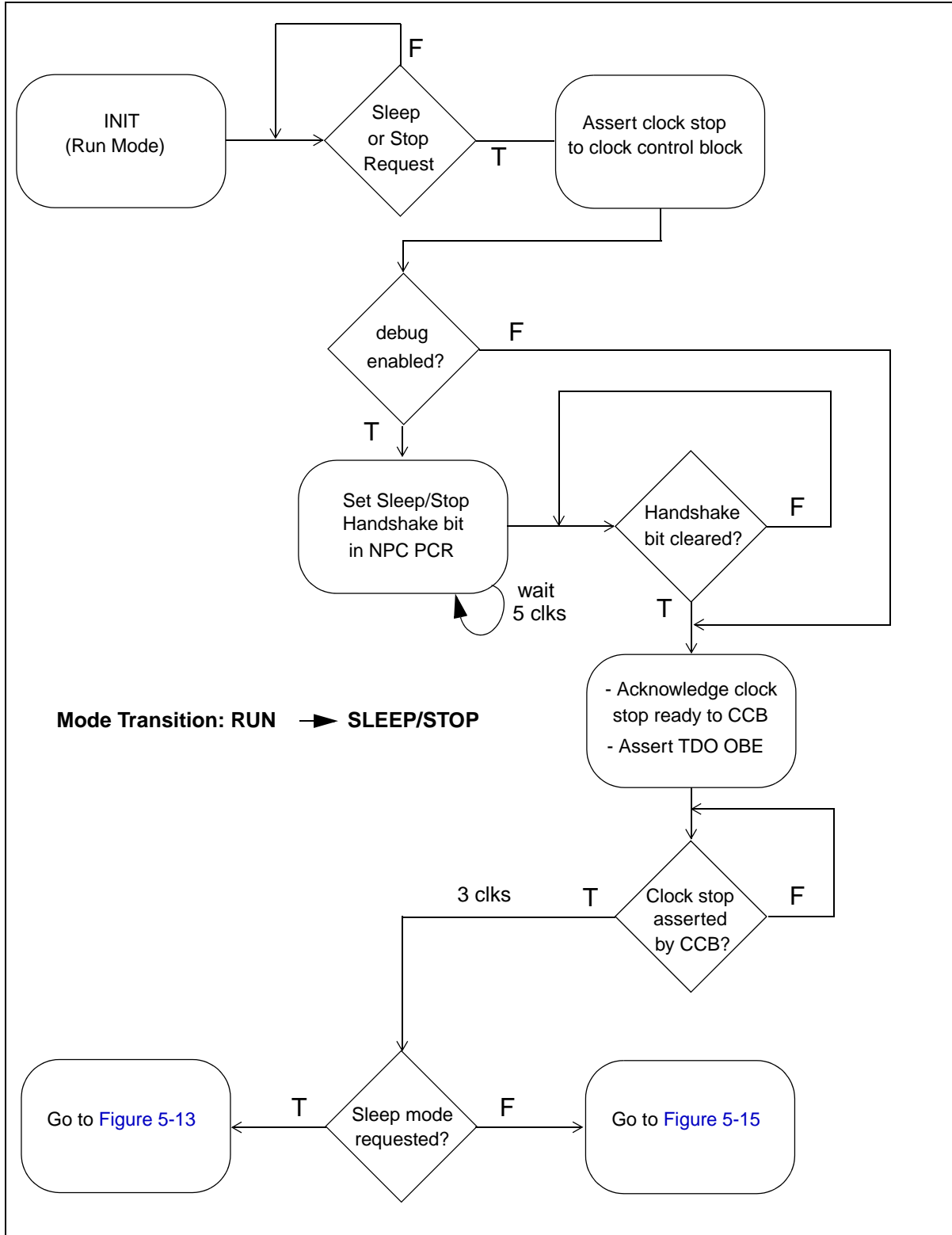


Figure 5-12. SLEEP/STOP Mode Entry Diagram

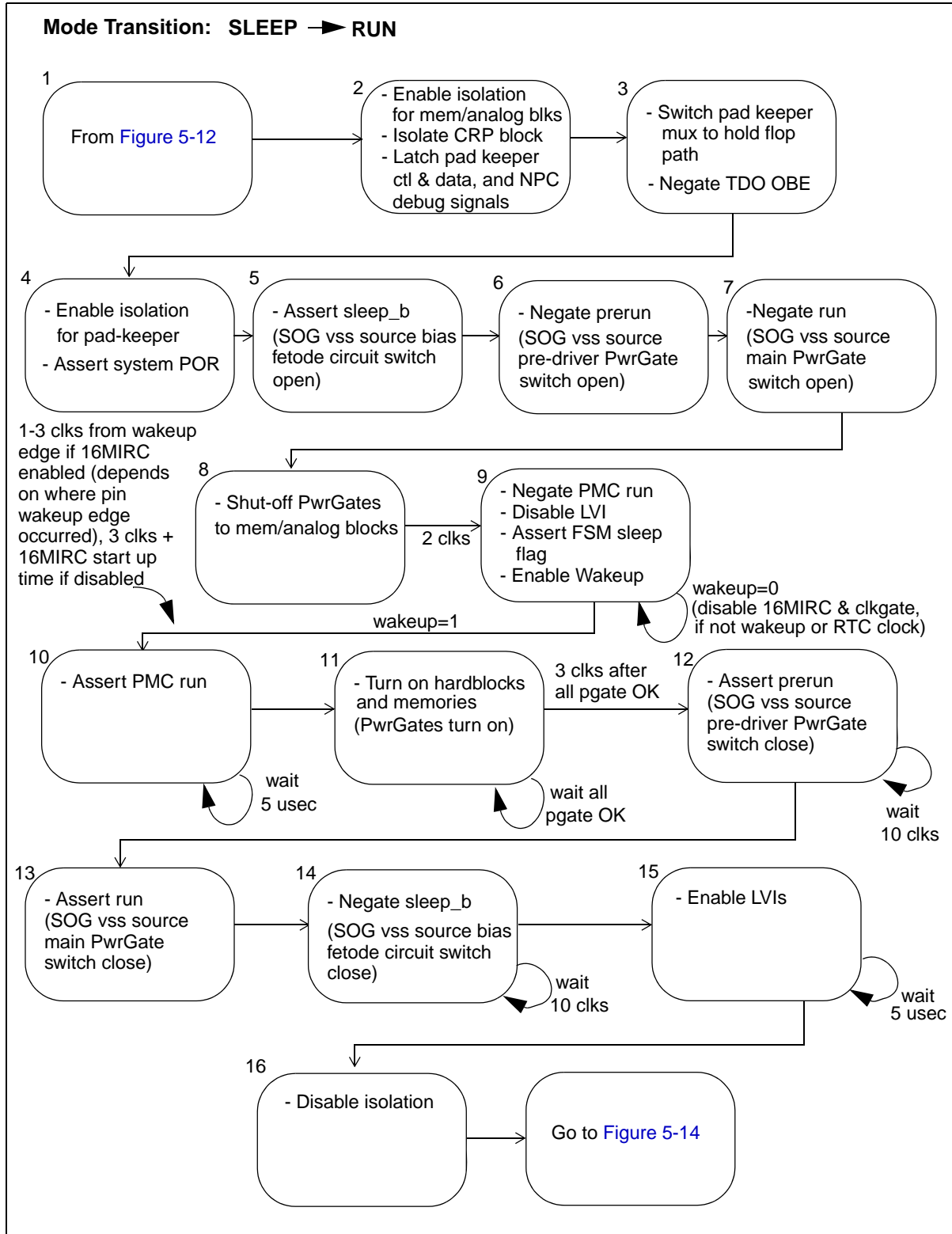


Figure 5-13. SLEEP Mode Transition Diagram (Part 1)

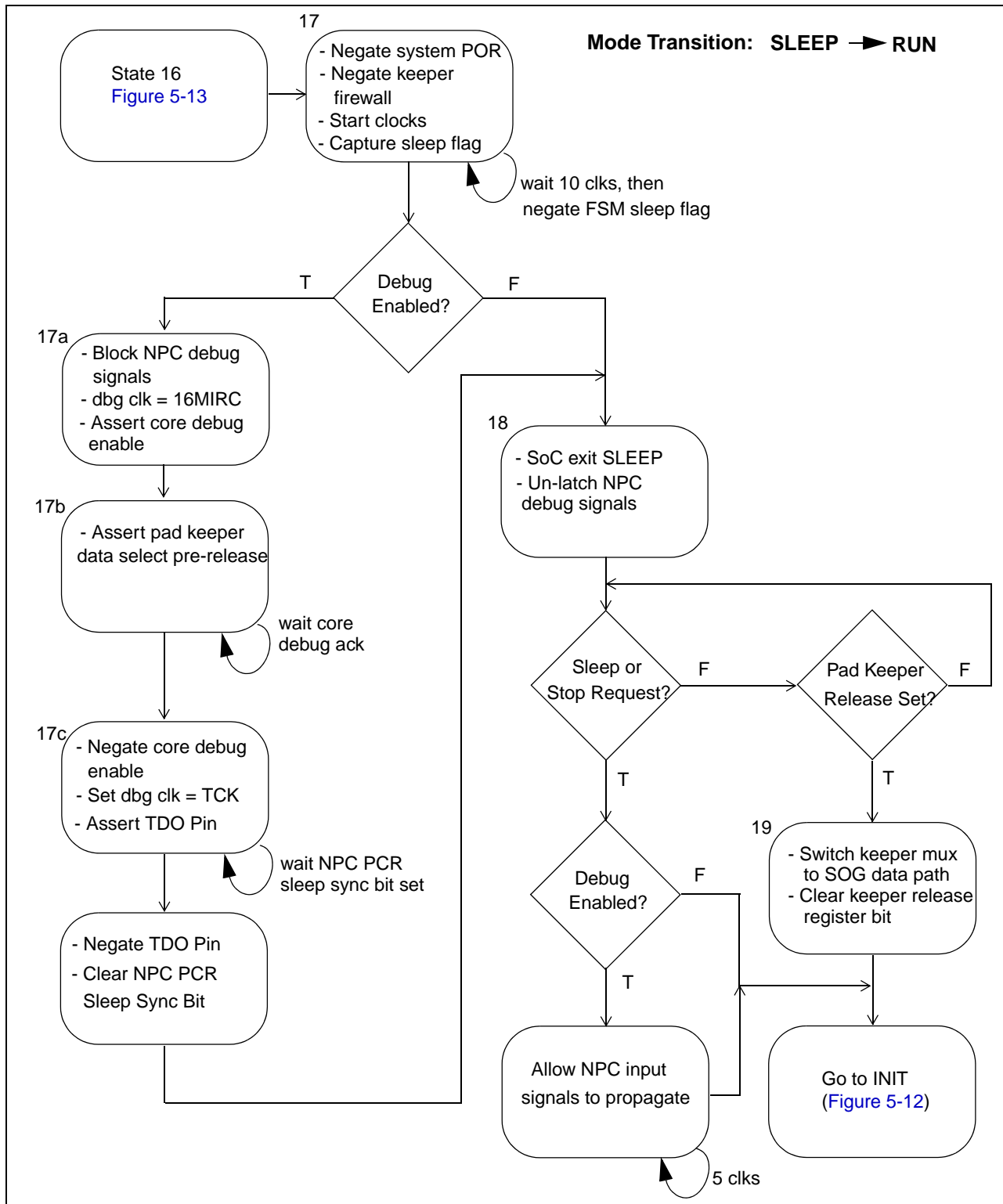


Figure 5-14. SLEEP Mode Transition Diagram (Part 2)

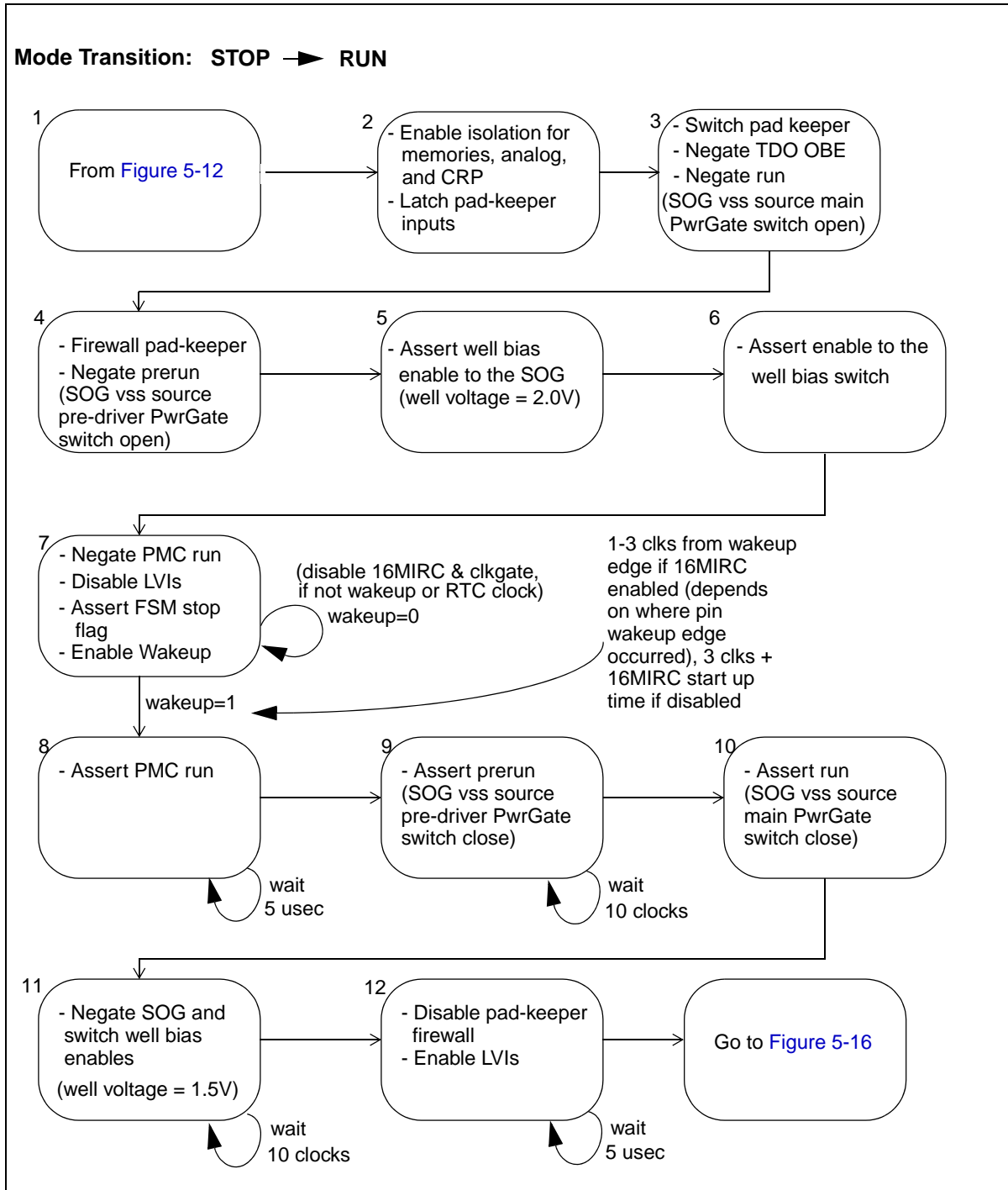


Figure 5-15. STOP Mode Transition Diagram (Part 1)

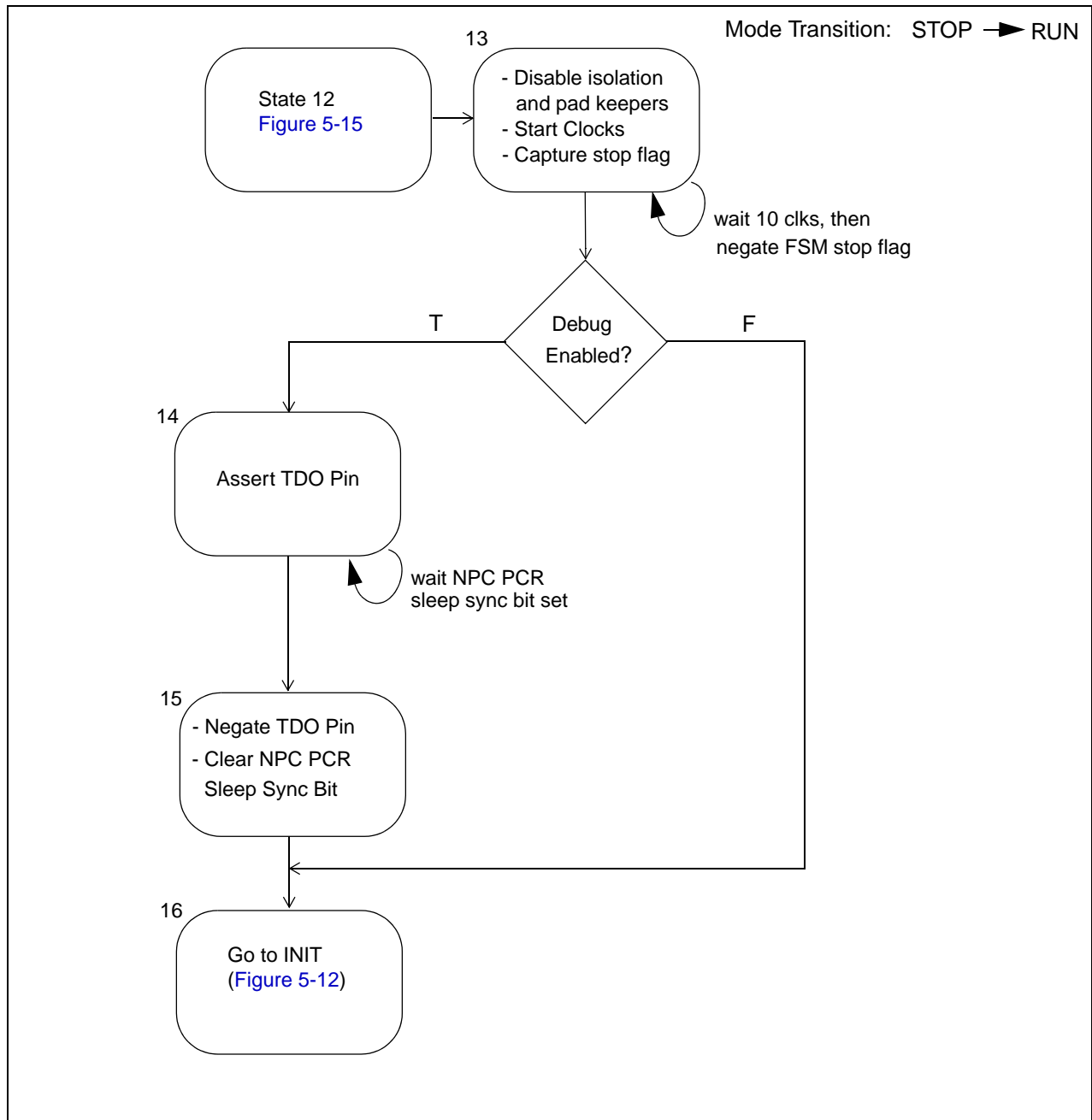


Figure 5-16. Stop Mode Transition Diagram (Part 2)

5.3.3.1 Pad Keeper Control

The pad keeper blocks are controlled by the CRP, but are not contained within it. The pad keepers maintain output pin states through the sleep and stop modes for pins configured as outputs in the SIU PCR registers. Pins that are to be used for wakeup from sleep/stop modes, must have the IBE enabled in the SIU PCR prior to sleep/stop entry. If a pullup/down is enabled on an input pin prior to entry into sleep or stop mode, it will remain enabled during the low-power mode.

The pad keepers are released automatically on the exit from stop mode. After exiting sleep mode, the user must restore the external pin configuration in the SIU and enable the IP blocks that drive external pins before releasing the pad keepers. The pad keepers are released by writing a 1 to the CRP_PSCR[PKREL] bit.

If the device enters sleep or stop mode, and the pad keepers are not released from a prior sleep mode, the pad keepers will automatically be released. In this case, the current pin configuration in the SIU PCR and the values driven by the IP blocks will be latched in the pad keepers and driven through the newly entered low power mode.

5.3.3.2 Sleep/Stop Mode Reset Operation

The reset controller in the SIU controls the normal reset sequences from POR, LVI, and other resets when the device is in RUN mode. The CRP controls reset operation for the device in sleep and stop modes.

The external RESET pin is enabled in all modes. Assertion of the RESET pin or a POR during sleep or stop modes causes the device to restart in RUN mode.

Upon power up from sleep mode, POR and reset is asserted to all logic that was powered down. The SIU will process the sleep recovery POR in the same manner as a normal POR. The RSR[PORS] bit in the SIU will be set after the reset controller sequence completes. The CRP_PSCR[SLEEPF] bit will be set in this case to indicate that the POR came from a sleep recovery.

Note that when powering up from sleep mode, the BOOTCFG pin is not read and the BAM boot sequence is bypassed since the Z1 and Z0 cores will branch to the appropriate reset vector set in the CRP_Z0VEC and CRP_Z1VEC registers.

5.3.4 Low-Power Wakeup

A POR or assertion of the external $\overline{\text{RESET}}$ pin causes exit from sleep and stop modes as a reset condition, and not a wakeup. A POR or external reset is captured in the SIU Reset Status Register. All CRP registers are reset for a POR, but some like the CRP_RTCSC are maintained for an external reset. Note that there are no internal reset sources active in sleep and stop modes. The internal power supply monitors are disabled during sleep and stop modes.

There are four methods for waking up the device from sleep or stop modes:

- RTC counter match
- RTC counter rollover
- API counter match
- External pin transition

All wakeup methods are independently enabled and function identically in either sleep or stop modes. The RTC, RTC rollover, and API wakeup logic is discussed in [Section 5.4, “Real-Time Counter \(RTC\).”](#)

Wakeup from sleep and stop modes can be enabled from transitions on up to eight external pins. Each of the eight external pin wakeup sources can be selected from a group of eight external pins, which gives a total of 64 possible external pin wakeup sources. External pin wakeup source selection is done in the CRP_WKPINSEL register, and [Table 5-8](#) gives the I/O pin mapping to the eight external pin wakeup

sources. To be used as a low-power mode wakeup, pins must be configured with the output buffer disabled in the SIU_PCR registers prior to entry into the low-power mode.

Each external pin wakeup source is enabled by the corresponding CRP_WKSE[WKPDETN] field. External pin wakeup generation can be selected for either a rising edge event on the pin, falling edge, or both. Using the CRP_WKSE[WKCLKSEL] bit, the edge capture logic can be selectively clocked from either the 16 MHz IRC clock for faster wakeup, or the 32 kHz IRC clock for lower average power. For wakeup, the pad assignment in the SIU_PCR does not matter. This enables a pin, such as a CAN receive pin, to wake up the device on a transition. For example, WKPSEL7 could select PF15, which could be assigned to CNRX_D.

The corresponding CRP_PSCR[PWKSRCF] flag bit will be set when a selected and enabled event occurs on an external pin wakeup source. An interrupt request can be generated for an external pin wakeup by setting the corresponding CRP_PSCR[PWKSRIE] bit. This interrupt request will be pending once the device recovers from the previous low-power mode.

On exiting stop mode, the PC continues with the location of interrupt service routine of the interrupt that exits the WAIT instruction. On exiting sleep mode, the PC value is loaded with the value contained in the Z1VEC/Z0VEC registers. The RECPTR register is a general purpose register which retains a value during stop and sleep modes and thus may be used by software to hold a generic value used by recovery routines.

A block diagram for the external pin wakeup logic is given in [Figure 5-17](#).

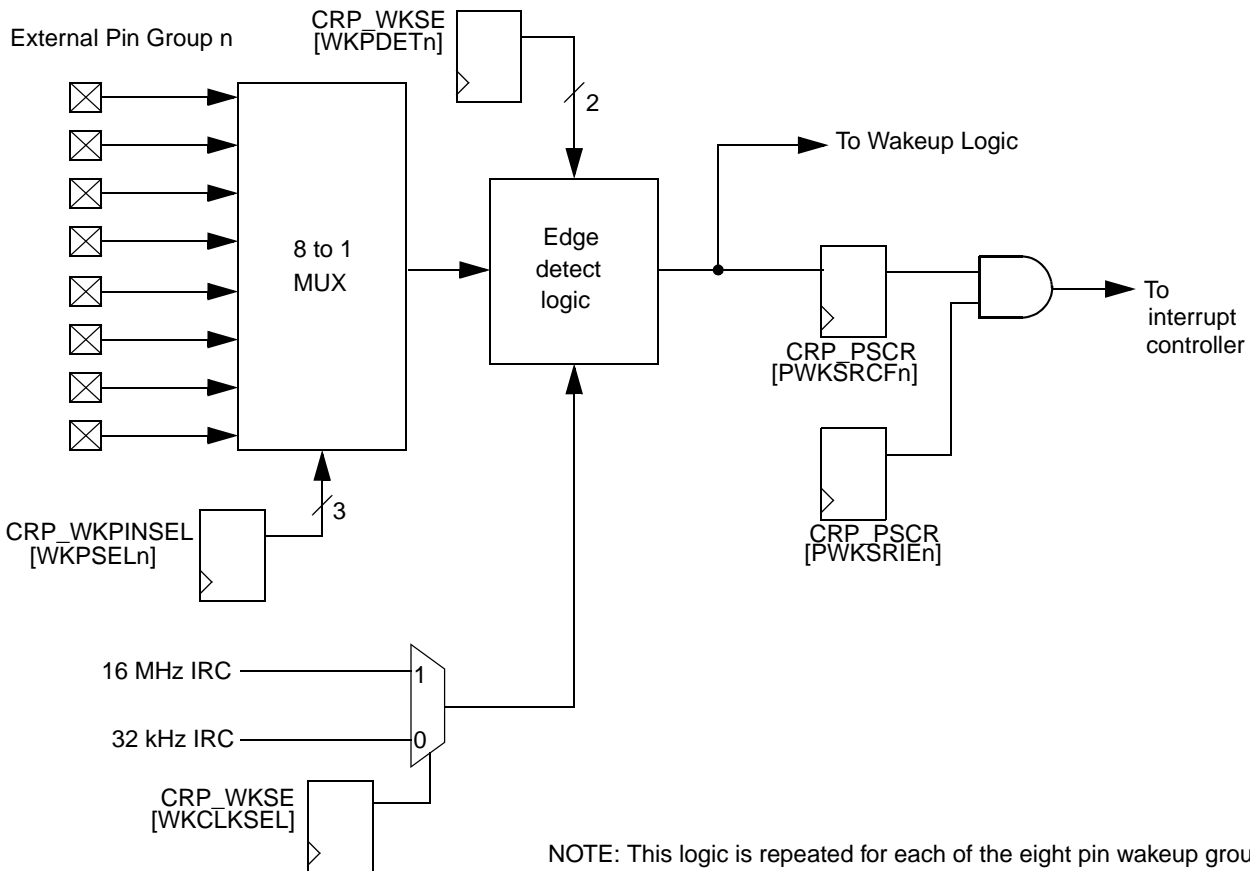


Figure 5-17. External Pin Wakeup Logic

5.3.4.1 Low Power Mode Debug Support

The CRP supports debug after exit from both SLEEP and STOP modes for both Nexus and JTAG debug tools. This function is enabled by setting the NPC PCR LP_DBG bit prior to entry into SLEEP/STOP modes.

On entry into STOP mode, if the NPC PCR LP_DBG bit is set, the CRP sets the NPC PCR STOP_SYNC bit to inform the debug tool that STOP mode is being entered. The CRP waits for this bit to be cleared before proceeding into STOP mode. During STOP mode, the entire SOC remains powered. The pad keepers are released immediately on wakeup from STOP mode. Any debug functionality that was enabled prior to STOP mode will be enabled after waking up from STOP mode. On exit from STOP mode, after the system clock is started, the CRP asserts the TDO pin in order to inform the debug tool of STOP mode exit. The TDO pin remains asserted until the debug tool sets the STOP_SYNC bit in the NPC PCR register. In order for the debug tool not to miss instruction execution, the CRP does not assert the wakeup interrupt to the Z0 and Z1 cores until after the debug tool has acknowledged the TDO assertion.

On entry into SLEEP mode, if the NPC PCR LP_DBG bit is set, the CRP sets the NPC PCR SLEEP_SYNC bit to inform the debug tool that SLEEP mode is being entered. The CRP waits for this bit to be cleared before proceeding into SLEEP mode. During SLEEP mode, most of the SOC is powered down, and the contents of the debug registers are lost. The CRP supports restoration of the debug registers on wakeup from SLEEP mode. The CRP latches the NPC PCR LP_DBG bit upon entry into SLEEP mode. On wakeup from SLEEP mode, if the latched bit is set, the CRP will place both the Z0 and Z1 cores into debug mode. The CRP selects the 16 MHz IRC to clock the core debug logic, so the development tool does not need to drive a clock on the TCK pin at this point. Once both cores have acknowledged that they have entered debug mode, the CRP allows the TCK pin to drive the debug logic, enables the JTAG pins, release the pad keepers for the Nexus pins, and drives the assertion of the TDO pin. The assertion of the TDO pin indicates to the debug tool that it can now restore the debug register contents via the JTAG interface. Although their pad keepers are released, the Nexus pins cannot be used until the NPC configuration is restored. The TDO pin remains asserted until the debug tool sets the SLEEP_SYNC bit in the NPC PCR register. At that point, TDO is negated, control of the pin given back to the JTAG controller, and the wakeup interrupt is asserted to the Z0 and Z1 cores. A block diagram of the SOC blocks and the connections between them to support debug on SLEEP wakeup is given in [Figure 5-18](#).

Note that the CRP will only enable the debug pins that were enabled prior to SLEEP mode entry. For example, if Nexus reduced port mode was enabled prior to SLEEP entry, then only the reduced port pins will be enabled on the wakeup from SLEEP.

When SLEEP or STOP mode is entered from RUN mode with the pad keepers still enabled from a previous SLEEP mode, the debug configuration is re-sampled and applied to the new low power mode. This includes sampling the NPC PCR LP_DBG bit to determine if debug should be enabled in the low power mode, and updating the Nexus pin configuration. However, if debug was not enabled when the pad keepers were initially enabled, the CRP will not be able to assert the TDO pin to synchronize with the debug tool. Therefore, the CRP does not support the sequence of entering SLEEP mode with debug disabled, and then re-entering SLEEP/STOP with the pad keepers enabled, and debug enabled. In this case, the low power mode will function as normal, but there is no capability for synchronization with the debug tool.

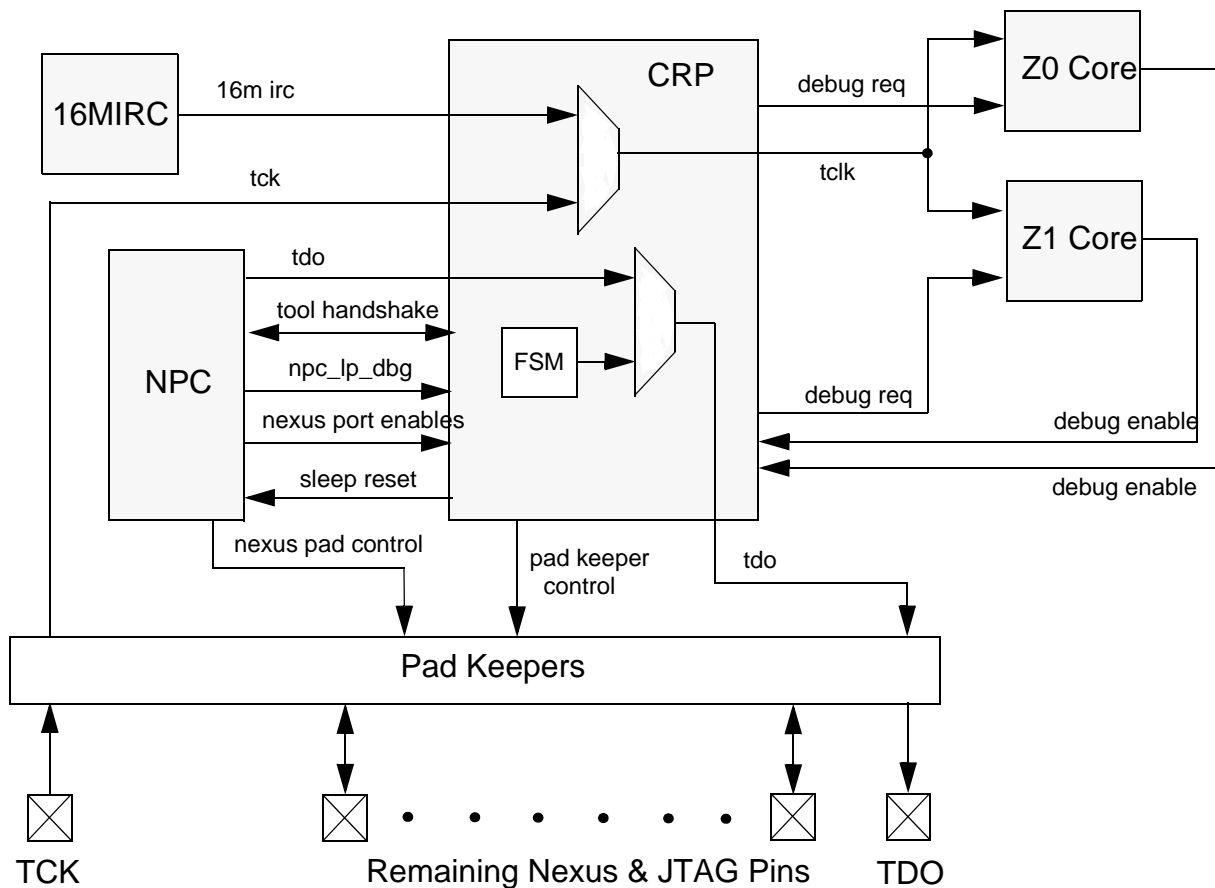


Figure 5-18. Sleep Mode Debug Block Integration

5.4 Real-Time Counter (RTC)

The RTC is a free-running counter used for time-keeping applications. The RTC may be configured to generate an interrupt at a pre-defined interval independent of the mode of operation (run, stop, and sleep). If in a low-power stop or sleep mode when the RTC interval is reached, the RTC will first generate a wakeup and then assert the interrupt request.

The RTC also supports an autonomous periodic interrupt function used to generate a periodic wakeup request to exit a low-power sleep mode or an interrupt request.

5.4.1 RTC Features

Features of the RTC include:

- 32-bit counter
- Selectable counter clock sources

- 32 kHz IRC
- 32 kHz OSC
- 16 MHz IRC
- Fixed divide by 32 prescaler to provide 1.0 ms resolution at 32 kHz
- Option to include or bypass a divide by 512 prescaler for the 16 MHz IRC. When not used, the divide by 512 prescaler is disabled to save power
- 32-bit counter supports gives roughly 1.5 month rollover with 1 ms resolution (2 μ s resolution with bypassed 16 MHz IRC)
- 12-bit compare value supports intervals of 1s up to ~1 hour with 1second resolution (2 ms resolution with bypassed 16 MHz IRC)
- RTC interrupt with interrupt enable
- Optionally enabled interrupt on RTC rollover
- Autonomous periodic interrupt support includes:
 - 10-bit compare value to support wakeup intervals of 1 ms to 1 second (2 μ s to 2 ms with bypassed 16 MHz IRC)
 - Wakeup logic has separate enable to support changing compare value while RTC running
 - Wakeup request flag captured for future reference
 - API interrupt with interrupt enable
 - Optionally operates in all modes of operation
- RTC continues to count through all resets except POR, VDD15 LVI, VDD33 LVI, VDDSYN LVI, VDD5 low LVI and VDD5 LVI

5.4.2 RTC Functional Description

The RTC consists of a 32-bit free-running counter enabled with CNTEN. (CNTEN when negated asynchronously resets the counter and synchronously enables the counter when enabled.) The value of the counter may be read via the RTCCNT register. Due to the clock synchronization, the RTCCNT value may actually represent a previous counter value.

The clock source to the counter is selected with CLKSEL and may be either the 32 kHz IRC, the 32 kHz OSC, or 16 MHz IRC. The 16 MHz IRC can optionally be divided by 512 to normalize it to the 32 kHz clock sources. Note that the 32 kHz OSC must be enabled before being selected. The 32 kHz OSC is selected to give a more accurate wakeup than the 32 kHz IRC. (CNTEN must be disabled when the clock sources are switched.) There is a fixed divide by 32 prescaler to support a 1.0 ms count resolution when using the 32 kHz input clock frequency.

When the counter value for counter bits 10–21 match the 12-bit value in RTCVAL then the RTCF interrupt flag is set (after proper clock synchronization). If the RTCIE interrupt enable bit is set, the RTC interrupt request is generated. The RTCF flag can be cleared by writing a 1 to RTCF. The RTCF supports interrupt requests in the range of 1 s to 4096 s (> 1 hr) with a 1 s resolution. RTCVAL may be updated when CNTEN is cleared to disable the counter only. If there is a match while in a sleep or stop mode, and the CRP_WKSE[RTCWKEN] bit is set, then the RTC will first generate a wakeup request to force a wakeup

to run mode, then the RTCF flag will be set. The RTC wakeup signal is captured in the CRP_PSCR[WKRTCF] flag bit.

A rollover wakeup and/or interrupt can be generated when the RTC transitions from a count of 0xFFFF_FFFF to 0x0000_0000. The rollover flag is enabled by setting the CRP_RTCSC[ROVREN] bit. An RTC counter rollover with this bit and the CRP_WKSE[RTCOVREN] bit set will cause a wakeup from both sleep and stop modes. The rollover wakeup flag is captured in the CRP_PSCR[WKRLLOVRF] bit. An interrupt request is generated for an RTC counter rollover when both the CRP_RTCSC[ROVREN] and CRP_RTCSC[RTCIE] bits are set.

Setting APIEN enables the autonomous interrupt function. The 10 bit APIVAL selects the time interval for triggering an interrupt and/or wakeup event. Since the RTC is a free-running counter, the APIVAL is added to the current count to calculate an offset. When the counter reaches the offset count, a interrupt and/or wakeup request is generated. Then the offset value is recalculated and again retriggers a new request when the new value is reached. APIVAL may only be updated when APIEN is disabled. When a compare is reached, the APIF interrupt flag is set (after proper clock synchronization). If the APIIE interrupt enable bit is set, then the API interrupt request is generated. The APIF flag can be cleared by writing a 1 to APIF. If there is a match while in sleep or stop mode, and the CRP_WKSE[APIWKEN] bit is set, then the API will first generate a wakeup request to force a wakeup to RUN mode, then the APIF flag will be set. The API wakeup flag is captured in the CRP_PSCR[WKAPIF] bit.

The RTC counter is unaffected during debug mode.

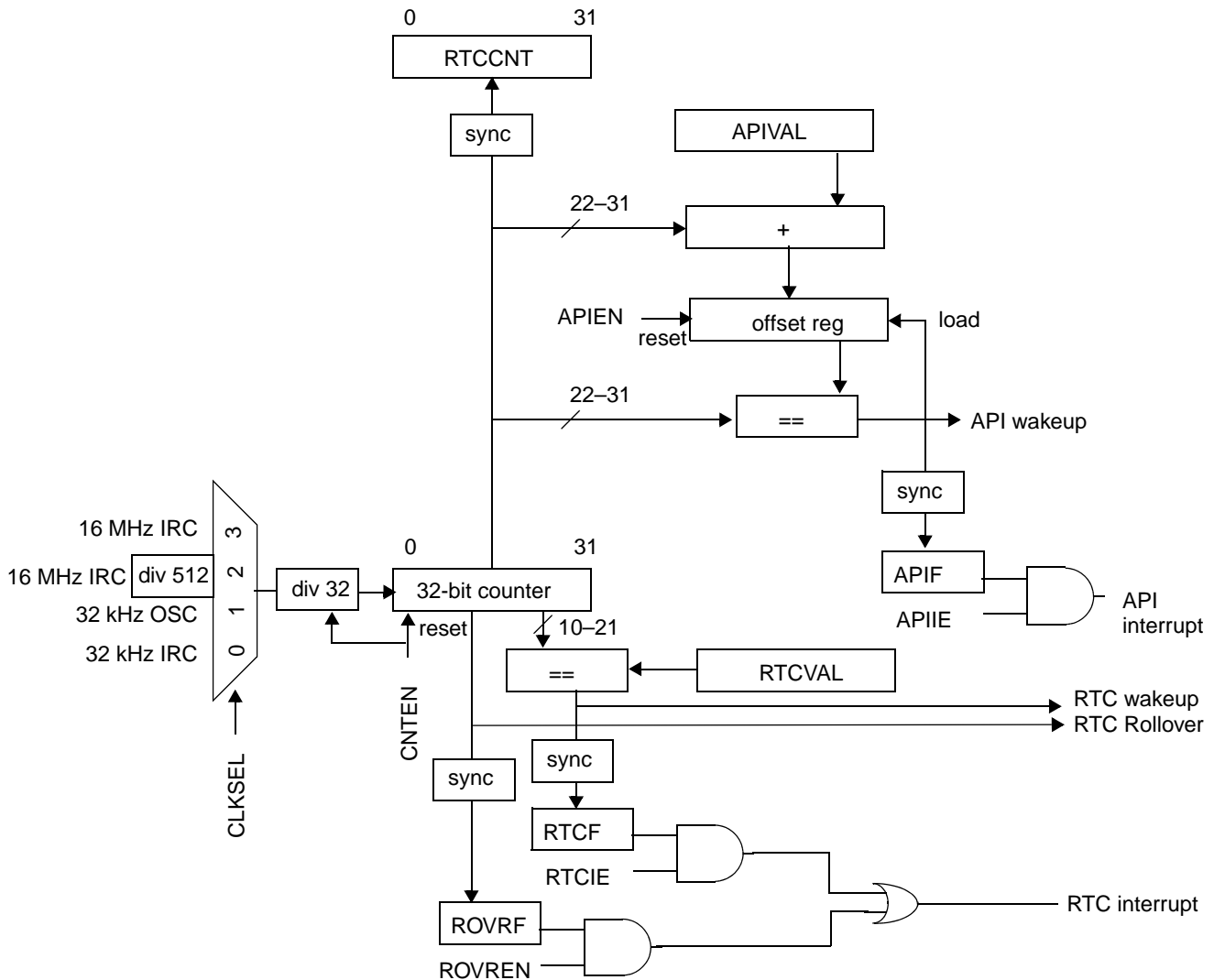


Figure 5-19. RTC/API Block Diagram

5.4.3 Register Description

The RTC registers control and monitor operation of the RTC. The registers that are relevant to the use of the RTC are as follows.

- RTC status and control register (Section 5.2.2.2, “RTC Status and Control Register (CRP_RTCSC)”)
- RTC counter register (Section 5.2.2.3, “RTC Counter Register (CRP_RTCCNT)”)

5.5 Power Supply Monitors

5.5.1 Power-On Reset (POR)

The internal power-on reset circuit monitors the voltage on the 5V VDDA supply and asserts a reset when the supply is below defined values. The POR is always enabled.

5.5.2 Low-Voltage Monitors (LVI)

The internal LVI circuits monitor when the voltage on the corresponding supply is below defined values and either assert a reset or an interrupt. All LVI circuits are enabled in RUN mode, but are disabled in sleep and stop modes. The LVIs also support hysteresis in the falling and rising trip points. There are the following LVIs:

- LVI15 — 1.5 V supply: generate reset if the 1.5 V VREG output is out of limit
- LVI33 — 3.3 V supply: generate reset if the 3.3 V VREG output (3.3 V I/O and flash domain) is out of limit
- LVI33SYN — 3.3V VDDSYN supply: generate reset if the 3.3 V VREG output for VDDSYN is out of limit
- LVI5 — 5 V VDDA supply: generate reset or interrupt if the 5 V VREG output is out of limit (nominal 4.5 V)
- LVI5L— 5 V VDDA supply: generate reset if the 5 V VREG output is out of limit (nominal 4.0V)
- LVI5H - 5 V VDDA supply: generate interrupt if the 5 V VREG output is out of limit (nominal 4.8 V)

The LVI15, LVI33, LVI33SYN, and LVI5L only generate resets. The reset request will cause a system reset and the appropriate bits set in the RSR status register.

The LVI5H only generates an interrupt request. The trigger event sets the CRP_SOCSC[LVI5HF] interrupt status flag and will generate an interrupt request to the system if the CRP_SOCSC[LVI5HIE] interrupt enable bit is set.

When a LVI5 trigger event occurs, the CRP_SOCSC[LVI5F] flag bit will be set, and either a reset or an interrupt may be generated, depending on the configuration of the CRP_SOCSC[LVI5IE] and CRP_SOCSC[LVI5RE] bits in the CRP. The CRP_SOCSC[LVI5RE] is always writable as long as the CRP_SOCSC[LVI5LOCK] bit is cleared. When CRP_SOCSC[LVI5LOCK] is set, then writes to CRP_SOCSC[LVI5RE] have no effect. The CRP_SOCSC[LVI5LOCK] bit is write-once and cleared only with POR.

There is no LVI monitoring of the individual VDDE I/O segments or the stop regulators during sleep and stop modes.

5.6 Low-Voltage Operation

The LVI5 is normally configured to generate a reset if the supply voltage is below 4.5 V. If this is always the desired function, then set the CRP_SOCSC[LVI5RE] to enable the reset function and set the write-once CRP_SOCSC[LVI5LOCK] bit to prevent any unintentional changes to the CRP_SOCSC[LVI5RE] bit.

Low-voltage or crank operation is the time when the 5 V supply voltage is pulled down below the minimum 4.5 V limit and the device is expected to be partially functional. The supply ramp is assumed to be relatively slow. Thus, the CRP_SOCSC[LVI5HF] is used as an early interrupt warning indication that the supply voltage is falling. Based on the LVI5HF (either polling the CRP_SOCSC[LVI5HF] status bit or by the CRP_SOCSC[LVI5HF] interrupt request), the LVI5 is configured for an interrupt function instead of a reset function. Low-voltage operation below 4.0 V is not supported, as the LVI5L will force a reset at this point.



Chapter 6

System Integration Unit (SIU)

6.1 Introduction

The system integration unit (SIU) controls MCU reset configuration, the system reset operation, pad configuration, external interrupt, general purpose I/O (GPIO), internal peripheral multiplexing, clock frequency divider configuration, peripheral clock disable configuration, and peripheral clock disable acknowledge. The reset configuration block contains the external pin boot configuration logic. The pad configuration block controls the static electrical characteristics of I/O pins. The GPIO block provides uniform and discrete input/output control of the MCU I/O pins. The reset controller performs reset monitoring of internal and external reset sources, and drives the $\overline{\text{RESET}}$ pin. The core accesses the SIU through the peripheral bus.

6.1.1 Block Diagram

[Figure 6-1](#) is a block diagram of the SIU. The signals shown are external pins to the device. The SIU registers are accessed through the crossbar switch. The power-on reset (POR) detection block, pad interface/pad ring block, and peripheral I/O channels are external to the SIU.

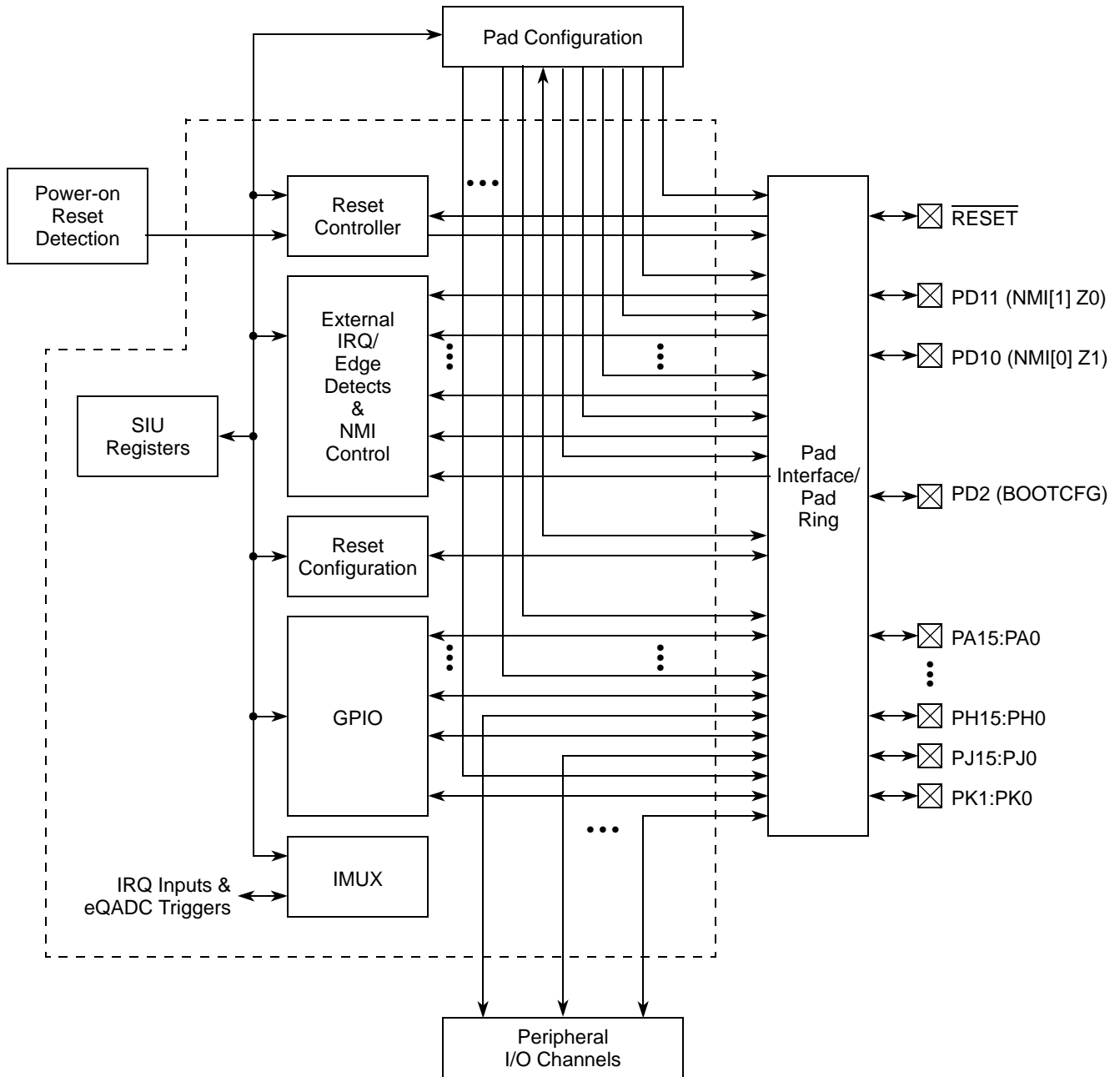


Figure 6-1. SIU Block Diagram

6.1.2 Features

Features include the following:

- System configuration
 - MCU reset configuration via external pins
 - Pad configuration control

- System reset monitoring and generation
 - Power-on reset support
 - Reset status register providing last reset source to software
 - Software controlled reset assertion
- External interrupt
 - 16 interrupt requests
 - Rising or falling edge event detection
 - Programmable digital filter for glitch rejection
- GPIO
 - GPIO function on up to 146 I/O pins (208 BGA, number varies per package type)
 - Dedicated input and output registers for each GPIO pin.
 - Parallel input and output registers with pins grouped into 16-bit ports
 - Read/Write data is coherent with data written/read using dedicated input/output registers.
- Internal multiplexing
 - Allows flexible selection of eQADC trigger inputs
 - Allows selection of interrupt requests among external pins
 - Allows selection of eMIOS inputs between external pins and deserialized DSPI outputs.
- System clock control
 - Clock divider control for individual peripherals or peripheral groups for lower power operation
 - Halt request register to disable clocks to unused peripherals for lower power operation
 - Halt acknowledge register to determine when peripheral clocks are disabled

6.1.3 Modes of Operation

6.1.3.1 Normal Mode

In normal mode, the SIU provides the register interface and logic that controls system configuration, the reset controller, GPIO, clock divider control, and peripheral clock disable/acknowledge.

6.1.3.2 Debug Mode

SIU operation in debug mode is identical to normal mode operation.

6.2 External Signal Description

Refer to [Table 2-1](#) and [Section 2.7](#), “Detailed External Signal Descriptions,” for signal properties.

6.2.1 Detailed Signal Descriptions

6.2.1.1 Reset ($\overline{\text{RESET}}$)

To reset all MCU modules, an external device asserts the $\overline{\text{RESET}}$ pin. The $\overline{\text{RESET}}$ pin is also an open-drain-output signal asserted during an internal reset. Assertion of the $\overline{\text{RESET}}$ pin when the device is in reset restarts the reset cycle (see [Chapter 7](#), “Reset”).

6.2.1.2 General-Purpose I/O Pins

The GPIO pins provide general-purpose input and output function. The GPIO pins are generally multiplexed with other I/O pin functions. An input (SIU_GPDI) or output (SIU_GPDO) register controls each GPIO input and output separately. See [Section 6.3.2.14](#), “GPIO Pin Data Output Registers (SIU_GPDO16_19–SIU_GPDO140_143),” [Section 6.3.2.15](#), “GPIO Pin Data Input Registers (SIU_GPDI0_3–SIU_GPDI144_145),” [Section 6.3.2.28](#), “Parallel GPIO Pin Data Output Register 0 (SIU_PGPDO0),”—[Section 6.3.2.37](#), “Parallel GPIO Pin Data Input Register 4 (SIU_PGPDI4),” and [Section 6.3.2.38](#), “Masked Parallel GPIO Pin Data Output Registers.”

6.2.1.3 Boot Configuration Pin (PD[2])

PD[2] is a GPIO pin. CNRX_B is the receive pin for the FlexCAN B module. eMIOS[10] is an input/output channel pin for the eMIOS200 module. The BOOTCFG pin is sampled before the negation of the $\overline{\text{RESET}}$ pin. The BAM program uses the value to determine the boot configuration.

6.2.1.4 Core Non Maskable Interrupt Pins (PD10 and PD11)

PD[10] is a GPIO pin. NMI0 is the critical interrupt input for the e200z1 core.

PD[11] is a GPIO pin. NMI1 is the critical interrupt input for the e200z0 core.

6.3 Memory Map and Registers

This section provides a detailed description of all SIU registers.

6.3.1 Module Memory Map

Table 6-1 is the address map for the SIU registers. All register addresses are given as an offset of the SIU base address.

Table 6-1. SIU Memory Map

Offset from SIU_BASE (0xFFFE_8000)	Register	Access	Reset Value	Section/Page
0x0000–0x0003	Reserved			
0x0004–0x0007	SIU_MIDR — MCU ID Register	R	— ¹	6.3.2.1/6-11
0x0008–0x000B	Reserved			
0x000C–0x000F	SIU_RSR — Reset Status Register	R/W	0x8000_000U	6.3.2.2/6-12
0x0010–0x0013	SIU_SRCR — System Reset Control Register	R/W	0x0800_C000	6.3.2.3/6-14
0x0014–0x0017	SIU_EISR — SIU External Interrupt Status Register	R/W	0x0000_0000	6.3.2.4/6-15
0x0018–0x001B	SIU_DIRER — DMA/Interrupt Request Enable Register	R/W	0x0000_0000	6.3.2.5/6-16
0x001C–0x001F	SIU_DIRSR — DMA/Interrupt Request Select Register	R/W	0x0000_0000	6.3.2.6/6-16
0x0020–0x0023	SIU_OSR — Overrun Status Register	R/W	0x0000_0000	6.3.2.7/6-17
0x0024–0x0027	SIU_ORER — Overrun Request Enable Register	R/W	0x0000_0000	6.3.2.8/6-18
0x0028–0x002B	SIU_IREER — External IRQ Rising-Edge Event Enable Register	R/W	0x0000_0000	6.3.2.9/6-18
0x002C–0x002F	SIU_IFEER — External IRQ Falling-Edge Event Enable Register	R/W	0x0000_0000	6.3.2.10/6-19
0x0030–0x0033	SIU_IDFR — External IRQ Digital Filter Register	R/W	0x0000_0000	6.3.2.11/6-20
0x0034–0x0037	SIU_IFIR — External IRQ Filtered Input Register	R/W	0x0000_0000	6.3.2.12/6-20
0x0038–0x0039	Reserved			
0x0040–0x0163	SIU_PCR0 – SIU_PCR145 — Pad Configuration Register 0 – Pad Configuration Register 145 ²	R/W	— ¹	6.3.2.13/6-21
0x0164–0x060F	Reserved			
0x0610–0x0689	SIU_GPDO0_16_19 – SIU_GPDO140_143 — GPIO Pin Data Output Register 16-19–GPIO Pin Data Output Register 140–143 ²	R/W	0x0000_0000	6.3.2.14/6-24
0x0690–0x07FF	Reserved			
0x0800–0x0891	SIU_GPDIO_3 – SIU_GPDI144_145 — GPIO Pin Data Input Register 0-3 –GPIO Pin Data Input Register 144-145 ²	R/W	— ¹	6.3.2.15/6-26
0x0892–0x08FF	Reserved			
0x0900–0x0903	SIU_ISEL0 — IMUX Select Register 0	R/W	0x0000_0000	6.3.2.16/6-27
0x0904–0x0907	SIU_ISEL1 — IMUX Select Register 1	R/W	0x0000_0000	6.3.2.17/6-28

Table 6-1. SIU Memory Map

Offset from SIU_BASE (0xFFFE_8000)	Register	Access	Reset Value	Section/Page
0x0908–0x090B	SIU_ISEL2 — IMUX Select Register 2	R/W	0x0000_0000	6.3.2.18/6-31
0x090C–0x097F	Reserved			
0x0980–0x0983	SIU_CCR — Chip Configuration Register	R/W	0x000U_0000	6.3.2.19/6-34
0x0984–0x0987	SIU_ECCR — External Clock Control Register	R/W	0x0000_1001	6.3.2.20/6-34
0x0988–0x098B	SIU_CMPAH — Compare A High Register	R/W	0x0000_0000	6.3.2.21/6-35
0x098C–0x098F	SIU_CMPAL — Compare A Low Register	R/W	0x0000_0000	6.3.2.22/6-36
0x0990–0x0993	SIU_CMPBH — Compare B High Register	R/W	0x0000_0000	6.3.2.23/6-36
0x0994–0x0997	SIU_CMPBL — Compare B Low Register	R/W	0x0000_0000	6.3.2.23/6-36
0x0998–0x099B	Reserved			
0x09A0–0x09A3	SIU_SYSCLK — System Clock Register	R/W	0x0000_0000	6.3.2.25/6-37
0x09A4–0x09A7	SIU_HLT — Halt Request	R/W	0x0000_0000	6.3.2.26/6-38
0x09A8–0x09AB	SIU_HLTACK — Halt Acknowledge	R	0x0000_0000	6.3.2.27/6-39
0x09AC–0x0BFF	Reserved			
0x0C00–0x0C13	SIU_PGPDO0 – SIU_PGPDO4 — Parallel GPIO Pin Data Output Register 0 – Parallel GPIO Pin Data Output Register 4	R/W	0x0000_0000	6.3.2.28/6-40
0x0C14–0x0C3F	Reserved			
0x0C40–0x0C53	SIU_PGPDIO – SIU_PGPDIO4 — Parallel GPIO Pin Data Input Register 0 – Parallel GPIO Pin Data Input Register 4	R/W	— ¹	6.3.2.33/6-43
0x0C54–0x0C83	Reserved			
0x0C84–0x0CA3	SIU_MPGPDO1 – SIU_MPGPDO8 — Masked Parallel GPIO Pin Data Output Register 1 – Masked Parallel GPIO Pin Data Output Register 8	R	0x0000_0000	6.3.2.38/6-45

¹ See register description for reset value.

² Gaps exist in this memory space where I/O pins are not available in the specified package.

Table 6-2 provides absolute addresses for the SIU_PCR and SIU_GPDO registers.

Table 6-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PA0	0	FFFE8040		FFFE8800
PA1	1	FFFE8042		FFFE8801
PA2	2	FFFE8044		FFFE8802
PA3	3	FFFE8046		FFFE8803
PA4	4	FFFE8048		FFFE8804

Table 6-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PA5	5	FFFE804A		FFFE8805
PA6	6	FFFE804C		FFFE8806
PA7	7	FFFE804E		FFFE8807
PA8	8	FFFE8050		FFFE8808
PA9	9	FFFE8052		FFFE8809
PA10	10	FFFE8054		FFFE880A
PA11	11	FFFE8056		FFFE880B
PA12	12	FFFE8058		FFFE880C
PA13	13	FFFE805A		FFFE880D
PA14	14	FFFE805C		FFFE880E
PA15	15	FFFE805E		FFFE880F
PB0	16	FFFE8060	FFFE8610	FFFE8810
PB1	17	FFFE8062	FFFE8611	FFFE8811
PB2	18	FFFE8064	FFFE8612	FFFE8812
PB3	19	FFFE8066	FFFE8613	FFFE8813
PB4	20	FFFE8068	FFFE8614	FFFE8814
PB5	21	FFFE806A	FFFE8615	FFFE8815
PB6	22	FFFE806C	FFFE8616	FFFE8816
PB7	23	FFFE806E	FFFE8617	FFFE8817
PB8	24	FFFE8070	FFFE8618	FFFE8818
PB9	25	FFFE8072	FFFE8619	FFFE8819
PB10	26	FFFE8074	FFFE861A	FFFE881A
PB11	27	FFFE8076	FFFE861B	FFFE881B
PB12	28	FFFE8078	FFFE861C	FFFE881C
PB13	29	FFFE807A	FFFE861D	FFFE881D
PB14	30	FFFE807C	FFFE861E	FFFE881E
PB15	31	FFFE807E	FFFE861F	FFFE881F
PC0	32	FFFE8080	FFFE8620	FFFE8820
PC1	33	FFFE8082	FFFE8621	FFFE8821
PC2	34	FFFE8084	FFFE8622	FFFE8822
PC3	35	FFFE8086	FFFE8623	FFFE8823
PC4	36	FFFE8088	FFFE8624	FFFE8824

Table 6-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PC5	37	FFFE808A	FFFE8625	FFFE8825
PC6	38	FFFE808C	FFFE8626	FFFE8826
PC7	39	FFFE808E	FFFE8627	FFFE8827
PC8	40	FFFE8090	FFFE8628	FFFE8828
PC9	41	FFFE8092	FFFE8629	FFFE8829
PC10	42	FFFE8094	FFFE862A	FFFE882A
PC11	43	FFFE8096	FFFE862B	FFFE882B
PC12	44	FFFE8098	FFFE862C	FFFE882C
PC13	45	FFFE809A	FFFE862D	FFFE882D
PC14	46	FFFE809C	FFFE862E	FFFE882E
PC15	47	FFFE809E	FFFE862F	FFFE882F
PD0	48	FFFE80A0	FFFE8630	FFFE8830
PD1	49	FFFE80A2	FFFE8631	FFFE8831
PD2	50	FFFE80A4	FFFE8632	FFFE8832
PD3	51	FFFE80A6	FFFE8633	FFFE8833
PD4	52	FFFE80A8	FFFE8634	FFFE8834
PD5	53	FFFE80AA	FFFE8635	FFFE8835
PD6	54	FFFE80AC	FFFE8636	FFFE8836
PD7	55	FFFE80AE	FFFE8637	FFFE8837
PD8	56	FFFE80B0	FFFE8638	FFFE8838
PD9	57	FFFE80B2	FFFE8639	FFFE8839
PD10	58	FFFE80B4	FFFE863A	FFFE883A
PD11	59	FFFE80B6	FFFE863B	FFFE883B
PD12	60	FFFE80B8	FFFE863C	FFFE883C
PD13	61	FFFE80BA	FFFE863D	FFFE883D
PD14	62	FFFE80BC	FFFE863E	FFFE883E
PD15	63	FFFE80BE	FFFE863F	FFFE883F
PE0	64	FFFE80C0	FFFE8640	FFFE8840
PE1	65	FFFE80C2	FFFE8641	FFFE8841
PE2	66	FFFE80C4	FFFE8642	FFFE8842
PE3	67	FFFE80C6	FFFE8643	FFFE8843
PE4	68	FFFE80C8	FFFE8644	FFFE8844

Table 6-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PE5	69	FFFE80CA	FFFE8645	FFFE8845
PE6	70	FFFE80CC	FFFE8646	FFFE8846
PE7	71	FFFE80CE	FFFE8647	FFFE8847
PE8	72	FFFE80D0	FFFE8648	FFFE8848
PE9	73	FFFE80D2	FFFE8649	FFFE8849
PE10	74	FFFE80D4	FFFE864A	FFFE884A
PE11	75	FFFE80D6	FFFE864B	FFFE884B
PE12	76	FFFE80D8	FFFE864C	FFFE884C
PE13	77	FFFE80DA	FFFE864D	FFFE884D
PE14	78	FFFE80DC	FFFE864E	FFFE884E
PE15	79	FFFE80DE	FFFE864F	FFFE884F
PF0	80	FFFE80E0	FFFE8650	FFFE8850
PF1	81	FFFE80E2	FFFE8651	FFFE8851
PF2	82	FFFE80E4	FFFE8652	FFFE8852
PF3	83	FFFE80E6	FFFE8653	FFFE8853
PF4	84	FFFE80E8	FFFE8654	FFFE8854
PF5	85	FFFE80EA	FFFE8655	FFFE8855
PF6	86	FFFE80EC	FFFE8656	FFFE8856
PF7	87	FFFE80EE	FFFE8657	FFFE8857
PF8	88	FFFE80F0	FFFE8658	FFFE8858
PF9	89	FFFE80F2	FFFE8659	FFFE8859
PF10	90	FFFE80F4	FFFE865A	FFFE885A
PF11	91	FFFE80F6	FFFE865B	FFFE885B
PF12	92	FFFE80F8	FFFE865C	FFFE885C
PF13	93	FFFE80FA	FFFE865D	FFFE885D
PF14	94	FFFE80FC	FFFE865E	FFFE885E
PF15	95	FFFE80FE	FFFE865F	FFFE885F
PG0	96	FFFE8100	FFFE8660	FFFE8860
PG1	97	FFFE8102	FFFE8661	FFFE8861
PG2	98	FFFE8104	FFFE8662	FFFE8862
PG3	99	FFFE8106	FFFE8663	FFFE8863
PG4	100	FFFE8108	FFFE8664	FFFE8864

Table 6-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PG5	101	FFFE810A	FFFE8665	FFFE8865
PG6	102	FFFE810C	FFFE8666	FFFE8866
PG7	103	FFFE810E	FFFE8667	FFFE8867
PG8	104	FFFE8110	FFFE8668	FFFE8868
PG9	105	FFFE8112	FFFE8669	FFFE8869
PG10	106	FFFE8114	FFFE866A	FFFE886A
PG11	107	FFFE8116	FFFE866B	FFFE886B
PG12	108	FFFE8118	FFFE866C	FFFE886C
PG13	109	FFFE811A	FFFE866D	FFFE886D
PG14	110	FFFE811C	FFFE866E	FFFE886E
PG15	111	FFFE811E	FFFE866F	FFFE886F
PH0	112	FFFE8120	FFFE8670	FFFE8870
PH1	113	FFFE8122	FFFE8671	FFFE8871
PH2	114	FFFE8124	FFFE8672	FFFE8872
PH3	115	FFFE8126	FFFE8673	FFFE8873
PH4	116	FFFE8128	FFFE8674	FFFE8874
PH5	117	FFFE812A	FFFE8675	FFFE8875
PH6	118	FFFE812C	FFFE8676	FFFE8876
PH7	119	FFFE812E	FFFE8677	FFFE8877
PH8	120	FFFE8130	FFFE8678	FFFE8878
PH9	121	FFFE8132	FFFE8679	FFFE8879
PH10	122	FFFE8134	FFFE867A	FFFE887A
PH11	123	FFFE8136	FFFE867B	FFFE887B
PH12	124	FFFE8138	FFFE867C	FFFE887C
PH13	125	FFFE813A	FFFE867D	FFFE887D
PH14	126	FFFE813C	FFFE867E	FFFE887E
PH15	127	FFFE813E	FFFE867F	FFFE887F
PJ0	128	FFFE8140	FFFE8680	FFFE8880
PJ1	129	FFFE8142	FFFE8681	FFFE8881
PJ2	130	FFFE8144	FFFE8682	FFFE8882
PJ3	131	FFFE8146	FFFE8683	FFFE8883
PJ4	132	FFFE8148	FFFE8684	FFFE8884

Table 6-2. Detailed Memory Map for SIU_PCR, SIU_GPDO, and SIU_GPDI

Pad ID	Pad #	SIU_PCR Address	SIU_GPDO Address	SIU_GPDI Address
PJ5	133	FFFE814A	FFFE8685	FFFE8885
PJ6	134	FFFE814C	FFFE8686	FFFE8886
PJ7	135	FFFE814E	FFFE8687	FFFE8887
PJ8	136	FFFE8150	FFFE8688	FFFE8888
PJ9	137	FFFE8152	FFFE8689	FFFE8889
PJ10	138	FFFE8154	FFFE868A	FFFE888A
PJ11	139	FFFE8156	FFFE868B	FFFE888B
PJ12	140	FFFE8158	FFFE868C	FFFE888C
PJ13	141	FFFE815A	FFFE868D	FFFE888D
PJ14	142	FFFE815C	FFFE868E	FFFE888E
PJ15	143	FFFE815E	FFFE868F	FFFE888F
PK0	144	FFFE8160		FFFE8890
PK1	145	FFFE8162		FFFE8891

6.3.2 Register Descriptions

This section lists the SIU registers in address order and describes the registers and their bit fields.

6.3.2.1 MCU ID Register (SIU_MIDR)

The SIU_MIDR contains the part identification number, package type, and mask revision number specific to the device. The part number is a read-only field mask-programmed with the device part number. It is not changed for bug fixes or process changes. The package type is a read-only field that reflects the device package type. The mask number is a read-only field mask-programmed with the device's specific mask revision level.

Offset: SIU_BASE + 0x0004

Access: User read

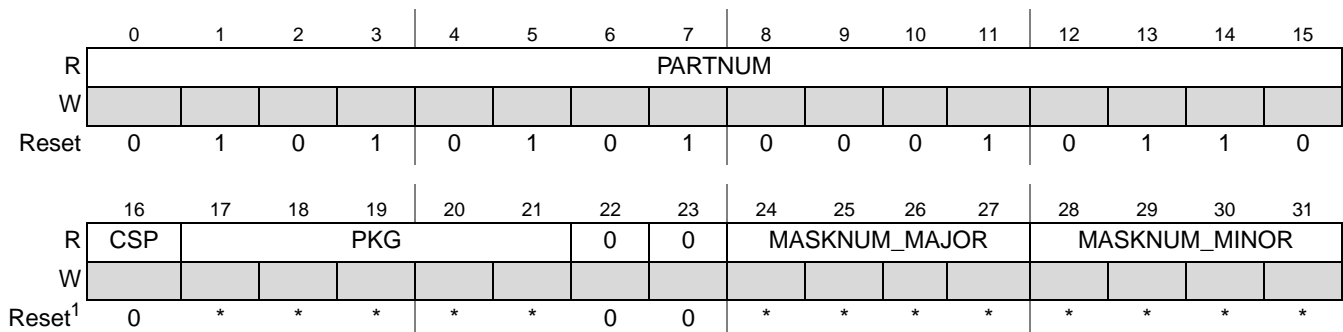


Figure 6-2. MCU ID Register (SIU_MIDR)

- ¹ PKG default value reflects the device package type as defined in [Table 6-3](#).
 MASKNUM_MAJOR default value is 0x0 for the device's initial mask set and changes for each major mask set revision.
 MASKNUM_MINOR default value is 0x0 for the device's initial mask set and changes for each minor mask set revision.

Table 6-3. SIU_MIDR Field Descriptions

Field	Description
PARTNUM	MCU Part Number. Read-only, mask-programmed part identification number of the MCU. Reads 0x5516 for the MPC5516.
CSP	Chip Scale Package. The CSP bit indicates whether the die is mounted in a chip scale package. 0 Not a chip scale package. 1 Chip scale package.
PKG	Package Configuration. These values set the pin package used for each MPC5510 device. 01101 144-pin LQFP 10001 176-pin LQFP 10000 208-pin BGA All other combinations are reserved
bits 22–23	Reserved
MASKNUM_MAJOR	Major Mask Revision Number. Read-only, mask-programmed mask number of the MCU. Reads 0x0 for the device's initial mask set and changes for each major mask set revision.
MASKNUM_MINOR	Minor Mask Revision Number. Read-only, mask-programmed mask number of the MCU. Reads 0x0 for the device's initial mask set and changes for each minor mask set.

6.3.2.2 Reset Status Register (SIU_RSR)

The SIU_RSR reflects the most recent source, or reset sources, and the pins' configuration state at reset. This register contains one bit for each reset source, indicating the last reset was power-on reset (POR), external, software system, watchdog, loss of PLL lock, loss of clock or checkstop reset. A reset status bit set to logic one indicates the reset type that occurred. After it is set, the reset source status bits in the SIU_RSR remain set until another reset occurs. In the following cases more than one reset bit is set:

1. If a power-on reset request has negated and the device is still in the resulting reset, and then an external reset is requested, both the power-on and external reset status bits will be set. In this case, the device started the reset sequence due to a power-on reset request but ended the reset sequence after an external reset request.
2. If any of the loss of clock, loss of lock, watchdog or checkstop reset requests occur on the same clock cycle, and no other higher priority reset source is requesting reset ([Table 6-4](#)), the reset status bits for all of the requesting resets are set.

Simultaneous reset requests are prioritized. When reset requests of different priorities occur on the same clock cycle, the lower priority reset request is ignored. Only the highest priority reset request's status bit is set. Except for a power-on reset request and condition 1 above, all reset requests of any priority are ignored until the device exits reset.

Table 6-4. Reset Source Priorities

Reset Source	Priority
Power on reset (POR) and external reset (Group 0)	Highest
Software system reset (Group1)	
Loss of clock, loss of lock, watchdog, checkstop (Group2)	
Software external reset (Group 3)	Lowest

Offset: SIU_BASE + 0x000C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PORS	ERS	LLRS	LCRS	WDRS	CRS	0	0	0	0	0	0	0	0	SSRS	0
W																
Reset ¹	1 ²	0 ³	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	BOOT CFG	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U ⁴	0

¹ The reset status register receives its reset values during power-on reset.

² The PORS bit is also set upon recovery from low-power sleep mode.

³ The ERS bit is set if the $\overline{\text{RESET}}$ pin was held low during POR.

⁴ Before the rising edge of $\overline{\text{RESET}}$, the PD2 pin state sets the BOOTCFG bit value. During sleep mode recovery, this bit will take on the state of the PD2 pad keeper when internal reset is negated.

Figure 6-3. Reset Status Register (SIU_RSR)

Table 6-5. SIU_RSR Field Descriptions

Field	Description
PORS	Power-on Reset Status. (Also set upon recover from sleep mode) 0 The reset controller acknowledged another reset source since the last assertion of the power-on reset input. 1 The power-on reset input to the reset controller is asserted, and no other reset source has been acknowledged since that assertion of the power-on reset input except an external reset.
ERS	External Reset Status. 0 Last reset source the reset controller acknowledged was not a valid assertion of the $\overline{\text{RESET}}$ pin. 1 Last reset source the reset controller acknowledged was a valid assertion of the $\overline{\text{RESET}}$ pin.
LLRS	Loss-of-Lock Reset Status. (Asynchronous reset source) 0 Last reset source the reset controller acknowledged was not a loss of PLL lock reset. 1 Last reset source the reset controller acknowledged was a loss of PLL lock reset.
LCRS	Loss-of-Clock Reset Status. (Asynchronous reset source) 0 Last reset source the reset controller acknowledged was not a loss of clock reset. 1 Last reset source the reset controller acknowledged was a loss of clock reset.
WDRS	Watchdog Timer/Debug Reset Status. 0 Last reset source the reset controller acknowledged was not a watchdog timer or debug reset. 1 Last reset source the reset controller acknowledged was a watchdog timer or debug reset.

Table 6-5. SIU_RSR Field Descriptions (continued)

Field	Description
CRS	Checkstop Reset Status. 0 Last reset source the reset controller acknowledged was not an enabled checkstop reset. 1 Last reset source the reset controller acknowledged was an enabled checkstop reset.
bits 6–13	Reserved.
SSRS	Software System Reset Status. 0 Last reset source the reset controller acknowledged was not a software system reset. 1 Last reset source the reset controller acknowledged was a software system reset.
bits 15–29	Reserved.
BOOTCFG	Status of BOOTCFG pin at negation of $\overline{\text{RESET}}$.
bit 31	Reserved.

6.3.2.3 System Reset Control Register (SIU_SRCR)

Offset: SIU_BASE + 0x0010

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SSR ¹	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CRE0	CRE1	0	0	0	0	0	0	SSRL ³	0	0	0	0	0	0	0
W																
Reset	1 ²	1 ²	0	0	0	0	0	0	0	0	0	0	0	0	0	0

¹ The SSR bit always reads as zero. A write of zero to this bit has no effect.

² The CRE0/1 bits are reset to 0b1 by POR. Other resets sources do not reset the bit value.

³ Once written to a 1, the SSRL bit can be reset only to zero by POR.

Figure 6-4. System Reset Control Register (SIU_SRCR)

Table 6-6. SIU_SRCR Field Descriptions

Field	Description
SSR	Software System Reset. Used to generate a software system reset. Writing a 1 to this bit causes an internal reset. The software system reset is processed as a synchronous reset. The bit is automatically cleared on the assertion of any other reset source except a software external reset. 0 Do not generate a software system reset. 1 Generate a software system reset.
bits 1–15	Reserved.

Table 6-6. SIU_SRCR Field Descriptions (continued)

Field	Description
CRE0	Checkstop Reset Enable (enable primary CPU, Z1, checkstop to generate reset). Writing a 1 to this bit enables a reset when the e200z1 checkstop reset request input is asserted. The checkstop reset request input is a synchronous internal reset source. The CRE0 bit defaults to checkstop reset enabled at POR. If this bit is cleared, it remains cleared until the next POR. 0 No reset occurs when the e200z1 checkstop reset input to the reset controller is asserted. 1 A reset occurs when the e200z1 checkstop reset input to the reset controller is asserted.
CRE1	Checkstop Reset Enable (enable secondary CPU, Z0, checkstop to generate reset). Writing a 1 to this bit enables a reset when the e200z0 checkstop reset request input is asserted. The checkstop reset request input is a synchronous internal reset source. The CRE1 bit defaults to checkstop reset enabled at POR. If this bit is cleared, it remains cleared until the next POR. 0 No reset occurs when the e200z0 checkstop reset input to the reset controller is asserted. 1 A reset occurs when the e200z0 checkstop reset input to the reset controller is asserted.
bits 18–23	Reserved.
SSRL	Software System Reset Lock. This bit is used to disable the software system reset. This bit defaults to 0. A write of 1 disables the SSR bit until the next POR (write once). 0 Enable the SSR bit. 1 Disable the SSR bit.
bits 25–31	Reserved.

6.3.2.4 External Interrupt Status Register (SIU_EISR)

The external interrupt status register is used to record edge-triggered events on the IRQ0–IRQ15 and NMI0–NMI1 inputs to the SIU. When an edge-triggered event is enabled in the SIU_IREER or SIU_IFEER for an IRQ_n input and then sensed, the corresponding SIU_EISR flag bit is set. The IRQ flag bit is set, regardless of the state of the corresponding DMA/IRQ enable bit in SIU_DIRER. The IRQ flag bit remains set until cleared by software or through the servicing of a DMA request. The IRQ flag bits are cleared by writing a 1 to the bits (w1c). A write of 0 has no effect.

Offset: SIU_BASE + 0x0014

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMI0	NMI1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c	w1c														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF15	EIF14	EIF13	EIF12	EIF11	EIF10	EIF9	EIF8	EIF7	EIF6	EIF5	EIF4	EIF3	EIF2	EIF1	EIF0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-5. SIU External Interrupt Status Register (SIU_EISR)

Table 6-7. SIU_EISR Field Descriptions

Field	Description
NMI n	Non Maskable Interrupt Flag for primary CPU (Z1) or secondary CPU (Z0). NMI0 is for the primary core. NMI1 is for the secondary core. This bit is set when an edge-triggered event occurs on the corresponding NMI n input. 0 No edge-triggered event occurred on the corresponding NMI n input. 1 An edge-triggered event occurred on the corresponding NMI n input.
bits 2–15	Reserved
EIF n	External Interrupt Request Flag n . Set when an edge-triggered event occurs on the corresponding IRQ n input. 0 No edge triggered event occurred on the corresponding IRQ n input. 1 An edge triggered event occurred on the corresponding IRQ n input.

6.3.2.5 DMA/Interrupt Request Enable Register (SIU_DIRER)

The SIU_DIRER allows the assertion of a DMA or interrupt request if the corresponding flag bit is set in the SIU_EISR. The external interrupt request enable bits enable the interrupt or DMA request. There are five interrupt requests from the SIU to the interrupt controller: IRQ0, IRQ1, IRQ2, IRQ3, and IRQ4, plus IRQ5 to IRQ15 on one interrupt request. The EIRE bits allow selection of which external interrupt request flag bits cause assertion of the one interrupt request signal for IRQ5 to IRQ15.

Offset: SIU_BASE + 0x0018

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIRE	EIRE	EIRE	EIRE	EIRE	EIRE	EIRE9	EIRE8	EIRE7	EIRE6	EIRE5	EIRE4	EIRE3	EIRE2	EIRE1	EIRE0
W	15	14	13	12	11	10										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-6. SIU DMA/Interrupt Request Enable Register (SIU_DIRER)

Table 6-8. SIU_DIRER Field Descriptions

Field	Description
bits 0–15	Reserved.
EIRE n	External Interrupt Request Enable n . Enables assertion of the interrupt request from the SIU to the interrupt controller when an edge triggered event occurs on the IRQ n pin. 0 External interrupt request disabled. 1 External interrupt request enabled.

6.3.2.6 DMA/Interrupt Request Select Register (SIU_DIRSR)

The SIU_DIRSR allows selection between a DMA or interrupt request for events on the IRQ4–IRQ1 inputs. The SIU_DIRSR selects between DMA and interrupt requests. If the corresponding bits are set in SIU_EISR and the SIU_DIRER, then the DMA/interrupt request select bit determines whether a DMA or interrupt request is asserted.

Offset: SIU_BASE + 000x1C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	DIRS4	DIRS3	DIRS2	DIRS1	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 ¹

¹ Do not write a 1 to bit 31 as it will block interrupt function of IRQ0.

Figure 6-7. DMA/Interrupt Request Select Register (SIU_DIRSR)

Table 6-9. SIU_DIRER Field Descriptions

Field	Description
bits 0–26	Reserved.
DIRS n	DMA/Interrupt Request Select n . Selects between a DMA or interrupt request when an edge triggered event occurs on the corresponding IRQ n pin. 0 Interrupt request selected. 1 DMA request selected.
bit 31	Reserved. Note: Reserved bit 31 is writeable, but setting this bit will block the interrupt function of IRQ0. Thus, this bit should not be written to a one.

6.3.2.7 Overrun Status Register (SIU_OSR)

The SIU_OSR contains flag bits that record an overrun. These flag bits are cleared by writing 1 to the bits (w1c); writing 0 has no effect.

Offset: SIU_BASE + 0x0020

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVF15	OVF14	OVF13	OVF12	OVF11	OVF10	OVF9	OVF8	OVF7	OVF6	OVF5	OVF4	OVF3	OVF2	OVF1	OVF0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-8. Overrun Status Register (SIU_OSR)

Table 6-10. SIU_OSR Field Descriptions

Field	Function
bits 0–15	Reserved.
OVFn	Overrun Flag <i>n</i> . This bit is set when an overrun occurs on the corresponding IRQn pin. 0 No overrun occurred on the corresponding IRQn pin. 1 An overrun occurred on the corresponding IRQn pin.

6.3.2.8 Overrun Request Enable Register (SIU_ORER)

The SIU_ORER contains bits to enable an overrun if the corresponding flag bit is set in the SIU_OSR. If any overrun request enable bit and the corresponding flag bit are set, the single combined overrun request from the SIU to the interrupt controller is asserted.

Offset: SIU_BASE + 0x0024

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ORE	ORE	ORE	ORE	ORE	ORE	ORE	ORE	ORE	ORE	ORE	ORE	ORE	ORE	ORE	ORE
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-9. Overrun Request Enable Register (SIU_ORER)

Table 6-11. SIU_ORER Field Descriptions

Field	Function
bits 0–15	Reserved.
OREn	Overrun Request Enable <i>n</i> . Enables the corresponding overrun request when an overrun occurs on the corresponding IRQn pin. 0 Overrun request disabled. 1 Overrun request enabled.

6.3.2.9 IRQ Rising-Edge Event Enable Register (SIU_IREER)

The SIU_IREER allows rising-edge-triggered events to be enabled on the corresponding IRQn pins. Setting the corresponding bits in the SIU_IREER and SIU_IFEER enables rising- and falling-edge events.

Offset: SIU_BASE + 0x0028

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NREE0	NREE1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IREE	IREE	IREE	IREE	IREE	IREE	IREE	IREE	IREE	IREE	IREE	IREE	IREE	IREE	IREE	IREE
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-10. IRQ Rising-Edge Event Enable Register (SIU_IREER)

Table 6-12. SIU_IREER Field Descriptions

Field	Function
NREE n	NREE n - NMI Rising-Edge Event Enable n . These write-once bits enable rising-edge-triggered events on the corresponding NMI n input. 0 Rising edge event disabled. 1 Rising edge event enabled.
bits 2–15	Reserved.
IREE n	IRQ Rising-Edge Event Enable n . Enables rising-edge triggered events on the corresponding IRQ n pin. 0 Rising edge event disabled. 1 Rising edge event enabled.

6.3.2.10 IRQ Falling-Edge Event Enable Register (SIU_IFEER)

The SIU_IFEER allows falling-edge-triggered events to be enabled on the corresponding IRQ n pins. Setting the corresponding bits in the SIU_IREER and SIU_IFEER enables rising- and falling-edge events.

Offset: SIU_BASE + 0x002C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NFEE0	NFEE1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE	IFEE
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-11. IRQ Falling-Edge Event Enable Register (SIU_IFEER)

Table 6-13. SIU_IFEER Field Descriptions

Field	Function
NFEEn	NMI Falling-Edge Event Enable <i>n</i> . These write-once bits enable rising-edge triggered events on the corresponding NMI <i>n</i> input. 0 Falling edge event disabled. 1 Falling edge event enabled.
bits 2–15	Reserved.
IFEE <i>n</i>	IRQ Falling-Edge Event Enable <i>n</i> . Enables falling-edge triggered events on the corresponding IRQ <i>n</i> pin. 0 Falling edge event disabled. 1 Falling edge event enabled.

6.3.2.11 External IRQ Digital Filter Register (SIU_IDFR)

The SIU_IDFR specifies the amount of digital filtering on the IRQ0–IRQ15 pins. The digital filter length field specifies the number of system clocks that define the period of the digital filter and the minimum time a signal must be held in the active state on the IRQ pins to be recognized as an edge-triggered event.

Offset: SIU_BASE + 0x0030

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DFL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-12. External IRQ Digital Filter Register (SIU_IDFR)

Table 6-14. SIU_IDFR Field Descriptions

Field	Function
0–27	Reserved.
28–31 DFL	Digital Filter Length. Defines digital filter period on the IRQ <i>n</i> inputs according to the following equation: $\text{Filter Period} = (\text{SystemClockPeriod} \times 2^{\text{DFL}}) + 1(\text{SystemClockPeriod})$ For a 66 MHz system clock, this gives a range of 30 ns to 491.5 μs. The minimum time of two clocks accounts for synchronization of the IRQ input pins with the system clock.

6.3.2.12 IRQ Filtered Input Register (SIU_IFIR)

This is a read only register that captures the output of the NMI*n* and IRQ*n* digital input filters.

Offset: SIU_BASE + 0x0034

Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FNMI	FNMI	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	0	1														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FI15	FI14	FI13	FI12	FI11	FI10	FI9	FI8	FI7	FI6	FI5	FI4	FI3	FI2	FI1	FI0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-13. External IRQ Filtered Input Register (SIU_IFIR)

Table 6-15. SIU_IFIR Field Descriptions

Field	Function
FNMI0	Filtered Non Maskable Interrupt 0. This bit is set/cleared for the corresponding NMI pin: 0 A logic one has passed through the NMI digital filter for NMIO pin. 1 A logic zero has passed through the NMI digital filter for NMIO pin.
FNMI1	Filtered Non Maskable Interrupt 1. This bit is set/cleared for the corresponding NMI pin: 0 A logic one has passed through the NMI digital filter for NMI1 pin. 1 A logic zero has passed through the NMI digital filter for NMI1 pin.
bits 2–15	Reserved.
FI <i>n</i>	Filtered Input <i>n</i> . This bit is set/cleared for the corresponding filtered IRQ pin: 0 A logic one has passed through the IRQ digital filter for the corresponding IRQ pin. 1 A logic zero has passed through the IRQ digital filter for the corresponding IRQ pin.

6.3.2.13 Pad Configuration Registers (SIU_PCR)

The following subsections define the SIU_PCRs for all device pins that allow configuration of the pin function, direction, and static electrical attributes. The information presented pertains to which bits and fields are active for a given pin or group of pins, and the register reset state. The reset state of SIU_PCRs in the following sections is prior to executing the boot-assist module (BAM) program. The BAM program may change SIU_PCRs based on reset configuration. See the BAM section of the manual for more detail.

For all PCR:

- If the pin is configured as an input only, the ODE and SRC bits do not apply.
- If the pin is configured as an output only, the HYS bit does not apply.
- When a pin is configured as an output, the weak internal pull up/down is disabled, regardless of the WPE or WPS settings in the PCR.

IBE and OBE bit definitions are specific to each PCR. When an I/O function is input- or output-only, the IBE and OBE bits do not have to be set to enable the input or output. When an I/O function can be either an input and output, the IBE and OBE bits must be set accordingly (IBE = 1 for input, and OBE = 1 for output). For I/O functions that change direction dynamically, such as the external data bus, switching between input and output is handled internally, and the IBE and OBE bits have no effect.

For all PCRs where GPIO function is available on the pin, if the pin is configured as an output and the IBE bit is set, the actual pin value is reflected in the corresponding GPDIx_x register. Negating the IBE bit when the pin is configured as an output reduces noise and power consumption. Reads from the GPDIx_x registers are undefined when the corresponding IBE bit is negated.

The SIU_PCRs are 16-bit registers that may be read or written as 32-bit values aligned on 32-bit address boundaries. [Table 6-16](#) describes the SIU_PCR fields.

NOTE

Not all of the fields may be present in a given SIU_PCR, depending on the type of pad it controls. See the specific SIU_PCR definition.

For all SIU_PCRs, the associated pin supports GPIO and up to three functions. The PA field is defined in [Table 6-16](#). For all PCRs of this type, a value of 0b11 selects Function 3, a value of 0b10 selects Function 2, the value 0b01 selects Function 1, and a value 0b00 selects GPIO.

All pins are named according to their associated parallel port name and associated bit number. For example, the Port A pins are named PA0 to PA15 (these pin names should not be confused with the bit field name.) See [Chapter 2, “Signal Descriptions,”](#) for a list of pins and their functions. The MCU is available in different package configurations. Some of the I/O controlled by the SIU PCR are not available in the smaller package. The port-enable logic for these PCR is the same for PCR that control I/O available in all packages. For the smaller package where some of the I/O is not available, the pad drivers are disabled in the pad interface logic. Do not select the unavailable functions via the PA field.

Some PCR contain a slew rate control (SRC) field. Slew rate control pertains to pins with slow or medium I/O pad types. The SRC field for all PCR with slew rate control is defined in [Table 6-16](#).

Table 6-16. SIU_PCR Field Descriptions

Field	Description										
bits 0–3	Reserved.										
PA	<p>Pin Assignment. Selects a multiplexed pad function. A separate port enable output signal from the SIU is asserted for each register value.</p> <table border="1" data-bbox="651 1367 1092 1619"> <thead> <tr> <th>PA Field</th> <th>Pin Function</th> </tr> </thead> <tbody> <tr> <td>0b00</td> <td>GPIO</td> </tr> <tr> <td>0b01</td> <td>Function 1</td> </tr> <tr> <td>0b10</td> <td>Function 2</td> </tr> <tr> <td>0b11</td> <td>Function 3</td> </tr> </tbody> </table>	PA Field	Pin Function	0b00	GPIO	0b01	Function 1	0b10	Function 2	0b11	Function 3
PA Field	Pin Function										
0b00	GPIO										
0b01	Function 1										
0b10	Function 2										
0b11	Function 3										
OBE	<p>Output Buffer Enable. Enables the pad as an output and drives the output buffer enable signal.</p> <p>0 Output buffer for the pad disabled. 1 Output buffer for the pad enabled.</p>										

Table 6-16. SIU_PCR Field Descriptions (continued)

Field	Description
IBE	Input Buffer Enable. Enables the pad as an input and drives the input buffer enable signal. 0 Input buffer for the pad disabled. 1 Input buffer for the pad enabled.
bits 8–9	Reserved.
ODE	Open Drain Output Enable. Controls output driver configuration for the pads. Either open drain or push/pull driver configurations can be selected. This feature applies only when pins are configured as outputs. 0 Open drain disabled for the pad (push/pull driver enabled). 1 Open drain enabled for the pad.
HYS	Input Hysteresis. Controls whether hysteresis is enabled for the pad. 0 Hysteresis disabled for the pad. 1 Hysteresis enabled for the pad.
SRC	Slew Rate Control. Controls slew rate for the pad. Slew rate control pertains to pins with slow or medium I/O pad types, and the output signals are driven according to the value of this field. Actual slew rate is dependent on the pad type and load. See the <i>MPC5510 Microcontroller Family Data Sheet</i> for this information. 00 Minimum slew rate (slowest) 01 Medium slew rate 10 Reserved 11 Maximum slew rate (fastest)
WPE	Weak Pullup/Down Enable. Controls whether the weak pullup/down devices are enabled/disabled for the pad. Pullup/down devices are enabled by default. 0 Weak pull device is disabled for the pad. 1 Weak pull device is enabled for the pad.
WPS	Weak Pullup/Down Select. Controls whether weak pullup or weak pulldown devices are used for the pad when weak pullup/down devices are enabled. The WKPCFG pin determines whether pullup or pulldown devices are enabled at reset. The WPS bit determines whether weak pullup or pulldown devices are used after reset, or for pads in which the WKPCFG pin does not determine the reset weak pullup/down state. 0 Pulldown value enabled for the pad. 1 Pullup value enabled for the pad.

6.3.2.13.1 Pad Configuration Registers 0–15 (SIU_PCR0–SIU_PCR15)

The SIU_PCR0 to SIU_PCR15 registers control the pin function and static electrical attributes of the Port A pins PA0 to PA15 (input only). For each pin, [Table 2-1](#) lists the signals available as the PA settings for Function1, Function2 and Function3.

Offset: SIU_BASE+0x0040–SIU_BASE+0x005F Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		0	IBE ¹	0	0	ODE	HYS	0	0	WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

¹ The IBE bit should be 0 when analog input function is selected.

Figure 6-14. Port A Pad Configuration Registers (SIU_PCR0 - SIU_PCR15)

See [Table 6-16](#) for bit field definitions.

6.3.2.13.2 Pad Configuration Registers 16–143 (SIU_PCR16–SIU_PCR143)

The SIU_PCR16 to SIU_PCR143 registers control the pin function, direction, and static electrical attributes of the Port B (PB0-PB15), Port C (PC0-PC15), Port D (PD0-PD15), Port E (PE0-PE15), Port F (PF0-PF150), Port G (PG0-PG15), Port H (PH0-PH15), and Port J (PJ0-PJ15) pins. For each pin, [Table 2-1](#) lists the signals that are available as the PA settings for Function1, Function2 and Function3.

Offset: SIU_BASE+0x0060–SIU_BASE+0x015F Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		OBE	IBE	0	0	ODE	HYS	SRC		WPE	WPS
W																
Reset	0	0	0	0	0	0	0	U ¹	0	0	0	0	0	0	U ¹	0

¹ The reset value is 1 for PCR50 (BOOTCFG), 0 for all other PCRs in this range

Figure 6-15. Port B to Port K Pad Configuration Registers (SIU_PCR16 - SIU_PCR145)

See [Table 6-16](#) for bit field definitions.

6.3.2.13.3 Pad Configuration Registers 144–145 (SIU_PCR144–SIU_PCR145)

The SIU_PCR144 and SIU_PCR145 registers control the pin function and static electrical attributes of the Port K pins PK0 and PK1 (input only). For each pin, [Table 2-1](#) lists the signals available as the PA settings for Function1, Function2 and Function3.

Offset: SIU_BASE+0x0160–SIU_BASE+0x0163 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PA		0	IBE ¹	0	0	ODE	HYS	0	0	WPE	WPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

¹ The IBE bit should be 0 when analog input function is selected.

Figure 6-16. Port K Pad Configuration Registers (SIU_PCR144–SIU_PCR145)

See [Table 6-16](#) for bit field definitions.

6.3.2.14 GPIO Pin Data Output Registers (SIU_GPDO16_19–SIU_GPDO140_143)

The SIU_GPDO16_19 register definition is in [Figure 6-17](#). All other SIU_GPDO_{x_x} registers follow the same pattern where four GPDO bits are placed in a 32-bit word, with one bit per byte. Each of the 128 PDO bits corresponds to a port pin in the order given in [Table 6-18](#). Gaps exist in this memory space where the pin is not available in the package.

NOTE

On MPC5510, the Port A and Port K pins are only general-purpose inputs. Therefore, there are no output data registers associated with these pins.

The SIU_GPDO_{x_x} registers are written to by software to drive data out on the external GPIO pin. Each byte of a register drives a single external GPIO pin, which allows the pin state to be controlled independently from other GPIO pins. Writes to the SIU_GPDO_{x_x} registers do not affect pin states if the pins are configured as inputs or as non-GPIO function by the associated pad configuration registers. The

SIU_GPDOx_x register values are automatically driven to the GPIO pins without software update if the GPIO pins' direction changes from input to output.

Offset: SIU_BASE + 0x0610–SIU_BASE+0x068F

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
R	0	0	0	0	0	0	0	PDO 16	0	0	0	0	0	0	0	0	PDO 17
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
R	0	0	0	0	0	0	0	PDO 18	0	0	0	0	0	0	0	0	PDO 19
W																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-17. GPIO Pin Data Out Register 16 - 19 (SIU_GPDO16_19)

Table 6-17. SIU_GPDO_n Field Descriptions

Field	Description
PDO _n	Pin Data Out. Stores the data to be driven out on the external GPIO pin associated with the register. If the register is read, it returns the value written. 0 V _{OL} driven on the external GPIO pin when the pin is configured as an output. 1 V _{OH} driven on the external GPIO pin when the pin is configured as an output.

Table 6-18. Pin Data Output Register to Pin Mapping

SIU_GPDOx_x	Address Offset	Pin
16_19	0x0610	PB0-PB3
20_23	0x0614	PB4-PB7
24_27	0x0618	PB8-PB11
28_31	0x061C	PB12-PB15
32_35	0x0620	PC0-PC3
36_39	0x0624	PC4-PC7
40_43	0x0628	PC8-PC11
44_47	0x062C	PC12-PC15
48_51	0x0630	PD0-PD3
52_55	0x0634	PD4-PD7
56_59	0x0638	PD8-PD11
60_63	0x063C	PD12-PD15
64_67	0x0640	PE0-PE3
68_71	0x0644	PE4-PE7
72_75	0x0648	PE8-PE11
76_79	0x064C	PE12-PE15
80_83	0x0650	PF0-PF3
84_87	0x0654	PF4-PF7
88_91	0x0658	PF8-PF11
92_95	0x065C	PF12-PF15

Table 6-18. Pin Data Output Register to Pin Mapping (*continued*)

SIU_GPDO _x _x	Address Offset	Pin
96_99	0x0660	PG0-PG3
100_103	0x0664	PG4-PG7
104_107	0x0668	PG8-PG11
108_111	0x066C	PG12-PG15
112_115	0x0670	PH0-PH3
116_119	0x0674	PH4-PH7
120_123	0x0678	PH8-PH11
124_127	0x067C	PH12-PH15
128_131	0x0680	PJ0-PJ3
132_135	0x0684	PJ4-PJ7
136_139	0x0688	PJ8-PJ11
140_143	0x068C	PJ12-PJ15

6.3.2.15 GPIO Pin Data Input Registers (SIU_GPDI0_3–SIU_GPDI144_145)

The definition of the SIU_GPDI0_3 register is given in [Figure 6-18](#). All other SIU_GPDI_x_x registers follow the same pattern where 4 GPDI bits are placed in a 32-bit word, with one bit per byte. Each of the 146 GPDI bits correspond to the port pin ([Table 6-20](#)). Gaps exist in this memory space where the pin is not available in the package.

The SIU_GPDI_x_x registers are read-only registers that allow software to read the input state of an external GPIO pin. Each byte of a register represents the input state of a single external GPIO pin. If the GPIO pin is configured as an output, and the input buffer enable (IBE) bit is set in the associated Pad Configuration Register, the SIU_GPDI_x_x register reflects the actual state of the output pin.

Offset: SIU_BASE + 0x0800–SIU_BASE+0x0891

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDI0	0	0	0	0	0	0	0	PDI1
W																
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	U
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDI2	0	0	0	0	0	0	0	PDI3
W																
Reset	0	0	0	0	0	0	0	U	0	0	0	0	0	0	0	U

Figure 6-18. GPIO Pin Data Input Register 0–3 (SIU_GPDI0_3)

Table 6-19. SIU_GPDI_n Field Description

Field	Description
PDI _n	Pin Data In. This bit reflects the input state on the external GPIO pin associated with the register. 0 Signal on pin is less than or equal to V _{IL} . 1 Signal on pin is greater than or equal to V _{IH} .

Table 6-20. GPIO Pin Data Input Register to Pin Mapping

SIU_GPDIx_x	Address Offset	Pin
0_3	0x0800	PA0–PA3
4_7	0x0804	PA4–PA7
8_11	0x0808	PA8–PA11
12_15	0x080C	PA12–PA15
16_19	0x0810	PB0–PB3
20_23	0x0814	PB4–PB7
24_27	0x0818	PB8–PB11
28_31	0x081C	PB12–PB15
32_35	0x0820	PC0–PC3
36_39	0x0824	PC4–PC7
40_43	0x0828	PC8–PC11
44_47	0x082C	PC12–PC15
48_51	0x0830	PD0–PD3
52_55	0x0834	PD4–PD7
56_59	0x0838	PD8–PD11
60_63	0x083C	PD12–PD15
64_67	0x0840	PE0–PE3
68_71	0x0844	PE4–PE7
72_75	0x0848	PE8–PE11
76_79	0x084C	PE12–PE15
80_83	0x0850	PF0–PF3
84_87	0x0854	PF4–PF7
88_91	0x0858	PF8–PF11
92_95	0x085C	PF12–PF15
96_99	0x0860	PG0–PG3
100_103	0x0864	PG4–PG7
104_107	0x0868	PG8–PG11
108_111	0x086C	PG12–PG15
112_115	0x0870	PH0–PH3
116_119	0x0874	PH4–PH7
120_123	0x0878	PH8–PH11
124_127	0x087C	PH12–PH15
128_131	0x0880	PJ0–PJ3
132_135	0x0884	PJ4–PJ7
136_139	0x0888	PJ8–PJ11
140_143	0x088C	PJ12–PJ15
144_145	0x0890	PK0–PK1

6.3.2.16 IMUX Select Register 0 (SIU_ISEL0)

The SIU_ISEL0 register selects the source for the EQADC trigger inputs.

NOTE

If a PIT trigger is selected as the source of the trigger, the trigger pulse width is two PIT clocks long. The PIT clock may be the system clock divided by 1, 2, 4, or 8, as selected by the SIU_SYSCLK[LPCLKDIV1] register. Thus the eQADC digital filtering needs to be set appropriately.

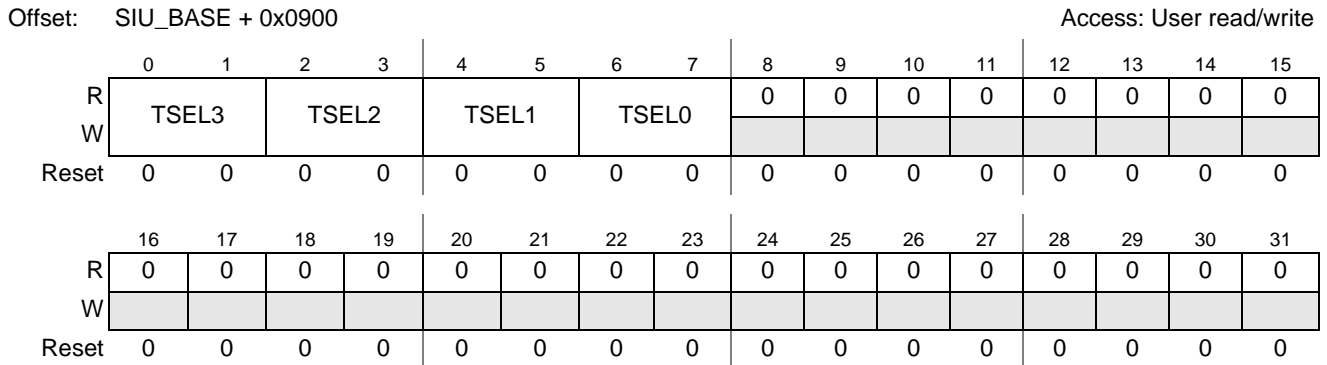


Figure 6-19. IMUX Select Register 0 (SIU_ISEL0)

Table 6-21. SIU_ISEL0 Field Descriptions

Field	Description
TSEL3	eQADC Trigger Input Select 3. Specifies input for eQADC trigger 3. 00 PC1 pin 01 PG1 pin 10 PIT 7 11 PIT 8
TSEL2	eQADC Trigger Input Select 2. Specifies input for eQADC trigger 2. 00 PC2 pin 01 PG2 pin 10 PIT 7 11 PIT 8
TSEL1	eQADC Trigger Input Select 1. Specifies input for eQADC trigger 1. 00 PE2 pin 01 PG3 pin 10 PIT 7 11 PIT 8
TSEL0	eQADC Trigger Input Select 0. Specifies input for eQADC trigger 0. 00 PC4 pin 01 PG4 pin 10 PIT 7 11 PIT 8
bits 8–31	Reserved.

6.3.2.17 IMUX Select Register 1 (SIU_ISEL1)

The SIU_ISEL1 selects the source for the external interrupt/DMA inputs.

Offset: SIU_BASE + 0x0904

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																
R	ESEL15				ESEL14				ESEL13				ESEL12				ESEL11				ESEL10				ESEL9				ESEL8			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31																
R	ESEL7				ESEL6				ESEL5				ESEL4				ESEL3				ESEL2				ESEL1				ESEL0			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-20. IMUX Select Register 1 (SIU_ISEL1)

Table 6-22. SIU_ISEL1 Field Descriptions

Field	Description ¹
ESEL15	External IRQ Input Select 15. Specifies input for IRQ15. 00 PD4 01 PF15 10 PG14 11 PH6
ESEL14	External IRQ Input Select 14. Specifies input for IRQ14. 00 PD3 01 PF13 10 PG13 11 PH9
ESEL13	External IRQ Input Select 13. Specifies input for IRQ13. 00 PB7 01 PE2 10 PG10 11 PH8
ESEL12	External IRQ Input Select 12. Specifies input for IRQ12. 00 PB6 01 PD12 10 PG12 11 PH4
ESEL11	External IRQ Input Select 11. Specifies input for IRQ11. 00 PB12 01 PD11 10 PF11 11 PG9
ESEL10	External IRQ Input Select 10. Specifies input for IRQ10. 00 PD9 01 PF14 10 PG7 11 PC6

Table 6-22. SIU_ISEL1 Field Descriptions (continued)

Field	Description ¹
ESEL9	External IRQ Input Select 9. Specifies input for IRQ9. 00 PA7 01 PD2 10 PF12 11 PH7
ESEL8	External IRQ Input Select 8. Specifies input for IRQ8. 00 PB9 01 PD5 10 PG5 11 PH5
ESEL7	External IRQ Input Select 7. Specifies input for IRQ7. 00 PB14 01 PD7 10 PG6 11 PC5
ESEL6	External IRQ Input Select 6. Specifies input for IRQ6. 00 PA6 01 PB8 10 PD1 11 PF10
ESEL5	External IRQ Input Select 5. Specifies input for IRQ5. 00 PA5 01 PB15 10 PD13 11 PC4
ESEL4	External IRQ Input Select 4. Specifies input for IRQ4. 00 PA4 01 PB13 10 PD15 11 PC2
ESEL3	External IRQ Input Select 3. Specifies input for IRQ3. 00 PA3 01 PB10 10 PD8 11 PC1
ESEL2	External IRQ Input Select 2. Specifies input for IRQ2. 00 PA2 01 PB5 10 PD6 11 PC0

Table 6-22. SIU_ISEL1 Field Descriptions (continued)

Field	Description ¹
ESEL1	External IRQ Input Select 1. Specifies input for IRQ1. 00 PA1 01 PD14 10 PG15 11 PJ12
ESEL0	External IRQ Input Select 0. Specifies input for IRQ0. 00 PA0 01 PD0 10 PG11 11 PD10

¹ Pins specified in this table must be configured as general purpose inputs to be used as external IRQs.

6.3.2.18 IMUX Select Register 2 (SIU_ISEL2)

The SIU_ISEL2 register specifies the source for the eMIOS[15:0] input channels, thus allowing the timer input channels to come from the pins, or from the deserialized output of one of three DSPI modules. Each 2-bit field in this register individually controls the setting for one eMIOS input channel, but typically all channels receive their inputs from the same source.

Offset: SIU_BASE + 0x0908

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS	SELEMIOS
W	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-21. IMUX Select Register 2 (SIU_ISEL2)

Table 6-23. SIU_ISEL2 Field Descriptions

Field	Description
SELEMIOS15	eMIOS[15] Input Select. The source of the input for the eMIOS[15] timer channel is selected according to the SELEMIOS15 field. 00 eMIOS[15] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIOS14	eMIOS[14] Input Select. The source of the input for the eMIOS[14] timer channel is selected according to the SELEMIOS14 field. 00 eMIOS[14] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output

Table 6-23. SIU_ISEL2 Field Descriptions (continued)

Field	Description
SELEMIO13	eMIOS[13] Input Select. The source of the input for the eMIOS[13] timer channel is selected according to the SELEMIO13 field. 00 eMIOS[13] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO12	eMIOS[12] Input Select. The source of the input for the eMIOS[12] timer channel is selected according to the SELEMIO12 field. 00 eMIOS[12] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO11	eMIOS[11] Input Select. The source of the input for the eMIOS[11] timer channel is selected according to the SELEMIO11 field. 00 eMIOS[11] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO10	eMIOS[10] Input Select. The source of the input for the eMIOS[10] timer channel is selected according to the SELEMIO10 field. 00 eMIOS[10] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO9	eMIOS[9] Input Select. The source of the input for the eMIOS[9] timer channel is selected according to the SELEMIO9 field. 00 eMIOS[9] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO8	eMIOS[8] Input Select. The source of the input for the eMIOS[8] timer channel is selected according to the SELEMIO8 field. 00 eMIOS[8] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO7	eMIOS[7] Input Select. The source of the input for the eMIOS[7] timer channel is selected according to the SELEMIO7 field. 00 eMIOS[7] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO6	eMIOS[6] Input Select. The source of the input for the eMIOS[6] timer channel is selected according to the SELEMIO6 field. 00 eMIOS[6] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output

Table 6-23. SIU_ISEL2 Field Descriptions (continued)

Field	Description
SELEMIO5	eMIOS[5] Input Select. The source of the input for the eMIOS[5] timer channel is selected according to the SELEMIO5 field. 00 eMIOS[5] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO4	eMIOS[4] Input Select. The source of the input for the eMIOS[4] timer channel is selected according to the SELEMIO4 field. 00 eMIOS[4] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO3	eMIOS[3] Input Select. The source of the input for the eMIOS[3] timer channel is selected according to the SELEMIO3 field. 00 eMIOS[3] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO2	eMIOS[2] Input Select. The source of the input for the eMIOS[2] timer channel is selected according to the SELEMIO2 field. 00 eMIOS[2] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO1	eMIOS[1] Input Select. The source of the input for the eMIOS[1] timer channel is selected according to the SELEMIO1 field. 00 eMIOS[1] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output
SELEMIO0	eMIOS[0] Input Select. The source of the input for the eMIOS[0] timer channel is selected according to the SELEMIO0 field. 00 eMIOS[0] input pin 01 DSPI_A deserialized output 10 DSPI_B deserialized output 11 DSPI_C deserialized output

6.3.2.19 Chip Configuration Register (SIU_CCR)

Offset: SIU_BASE + 0x0980

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MATCH	DISNEX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U	U
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	TESTLOCK	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-22. Chip Configuration Register (SIU_CCR)

Table 6-24. SIU_CCR Field Descriptions

Field	Description
bits 0–13	Reserved.
MATCH	Compare Register Match. The MATCH bit is a read-only bit that holds the value of the match input signal to the SIU. The match input is asserted if the values in the SIU_CMPAH/SIU_CMPAL and SIU_CMPBH/SIU_CMPBL are equal. 0 Match input signal is negated. 1 Match input signal is asserted.
DISNEX	Disable Nexus. The DISNEX bit is a read-only bit that holds the value of the Nexus disable input signal to the SIU. When system reset negates, the value in this bit depends on the censorship control word and the boot configuration bits. 0 Nexus disable input signal negated. 1 Nexus disable input signal asserted.
bits 16–23	Reserved.
TESTLOCK	TEST Lock. The TESTLOCK bit prevents access to Freescale internal test features. These internal test features are enabled by writing to reserved test bits in the device. Setting the TESTLOCK bit locks the test bits so that they cannot be changed inadvertently by runaway code. Customer initialization code should always set this bit. 0 Internal test features could be enabled. 1 Internal test features are disabled.
bits 25–31	Reserved. Note: Reserved bit 30 is writeable, but writing to this bit has no effect other than to update the value of the register. For future compatibility, this bit should be written to zero. This bit is reset with POR only.

6.3.2.20 External Clock Control Register (SIU_ECCR)

The SIU_ECCR controls the timing relationship between the system clock and the external clocks, CLKOUT. All bits and fields in the SIU_ECCR are read/write and reset by the asynchronous reset signal.

Offset: SIU_BASE + 0x0984

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EBDF	
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

Figure 6-23. External Clock Control Register (SIU_ECCR)

Table 6-25. SIU_ECCR Field Descriptions

Field	Description
bits 0–29	Reserved. Note: Reserved bits 16–24 and 28 are writeable, but writing to these bits has no effect other than to update the value of the register. For future compatibility, these bits should be written to zeros.
EBDF	External Bus Division Factor. Specifies frequency ratio between system clock and external clock, CLKOUT. The EBDF field must not be changed during an external bus access or while an access is pending. The CLKOUT frequency is divided from the system clock frequency according to the descriptions below. 00 Divide by 1 01 Divide by 2 10 Reserved 11 Divide by 4 Note: The reset value of the EBDF field is divide-by-2. Note: The EBDF field must not be modified while an external bus transaction is in progress. Note: If EBDF is equal to 0x00 and SYSCLKDIV is not equal to 0x00, then the CLKOUT pin will not have a nominal 50% duty cycle.

6.3.2.21 Compare A High Register (SIU_CMPAH)

The SIU_CMPAH register holds the 32-bit value that is compared against the value in the SIU_CMPBH register. The CMPAH field is read/write and reset by the asynchronous reset signal.

Offset: SIU_BASE + 0x0988

Access: User read-only

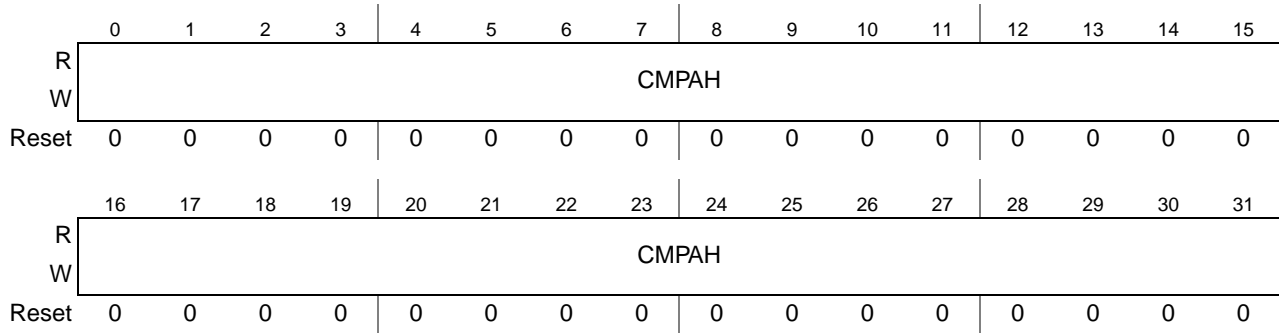


Figure 6-24. Compare A High Register (SIU_CMPAH)

6.3.2.22 Compare A Low Register (SIU_CMPAL)

The SIU_CMPAL register holds the 32-bit value that is compared against the value in the SIU_CMPBL register. The CMPAL field is read/write and reset by the asynchronous reset signal.

Offset: SIU_BASE + 0x098C

Access: User read-only

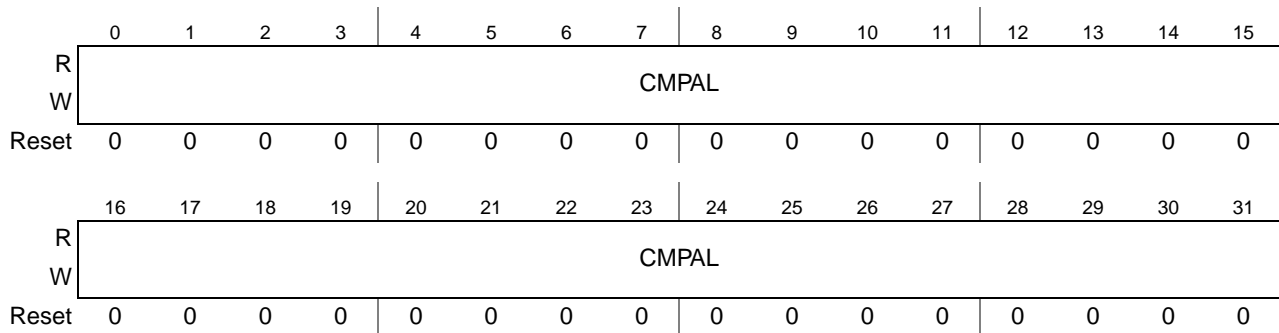


Figure 6-25. Compare A Low Register (SIU_CMPAL)

6.3.2.23 Compare B High Register (SIU_CMPBH)

The SIU_CMPBH register holds the 32-bit value that is compared against the value in the SIU_CMPAH register. The CMPBH field is read/write and reset by the asynchronous reset signal.

Offset: SIU_BASE + 0x0990

Access: User read-only

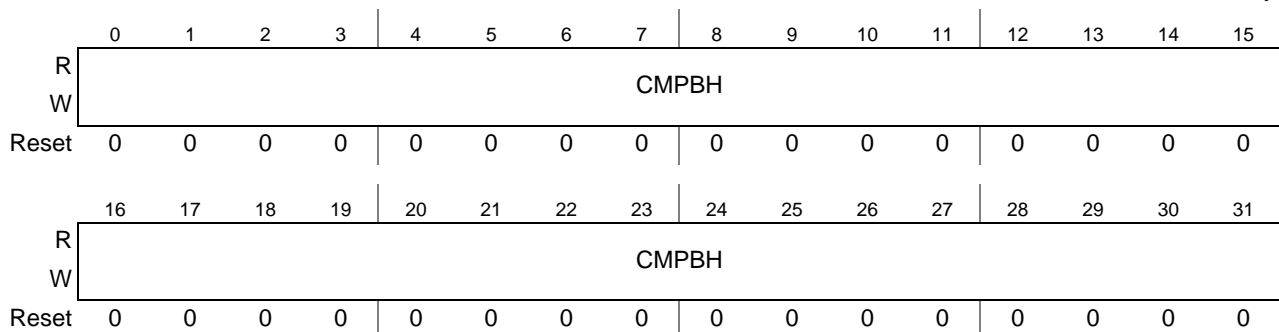


Figure 6-26. Compare B High Register (SIU_CMPBH)

6.3.2.24 Compare B Low Register (SIU_CMPBL)

The SIU_CMPBL register holds the 32-bit value that is compared against the value in the SIU_CMPAL register. The CMPBL field is read/write and reset by the asynchronous reset signal.

Offset: SIU_BASE + 0x0994

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CMPBL															
W	CMPBL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CMPBL															
W	CMPBL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-27. Compare B Low Register (SIU_CMPBL)

6.3.2.25 System Clock Register (SIU_SYSCLK)

The SIU_SYSCLK register controls the source for the system clock, the divider for the system clock, and eight fields that control the clock divider for groups of peripherals. For a listing of which peripherals are associated with which LPCLKDIV bit on MPC5510, see [Section 3.4.5, “Peripheral Clock Dividers.”](#)

Offset: SIU_BASE + 0x09A0

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SYSCLKSEL		SYSCLKDIV		SWT CLKSEL	0	0	0	0	0	0	0	0	0	0	0
W	SYSCLKSEL		SYSCLKDIV													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LPCLKDIV7		LPCLKDIV6		LPCLKDIV5		LPCLKDIV4		LPCLKDIV3		LPCLKDIV2		LPCLKDIV1		LPCLKDIV0	
W	LPCLKDIV7		LPCLKDIV6		LPCLKDIV5		LPCLKDIV4		LPCLKDIV3		LPCLKDIV2		LPCLKDIV1		LPCLKDIV0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-28. System Clock Register (SIU_SYSCLK)

Table 6-26. SIU_SYSCLK Field Descriptions

Field	Description
SYSCLKSEL	System Clock Select. The SYSCLKSEL bit selects the source for the system clock. 00 System clock supplied by 16 MHz IRC 01 System clock supplied by XOSC 10 System clock supplied by PLL 11 Reserved (defaults to 16 MHz IRC)
SYSCLKDIV	System Clock Divide. The SYSCLKDIV bits select the divider value for the system clock. The SYSCLKDIV divider is required in addition to the RFD to allow the other sources for the system clock (16 MHz IRC and OSC) to be divided to slowest frequencies to improve power. 00 Divide by 1 01 Divide by 2 10 Divide by 4 11 Divide by 8
SWTCLKSEL	Software Watchdog Timer Clock Select. The SWTCLKSEL bit determines whether the software watchdog timer counter uses 16 MHz IRC or the system clock. 0 System Clock (Note: out of reset, the system clock is driven by the 16 MHz IRC) 1 16 MHz IRC
bits 5–15	Reserved.
LPCLKDIV _n	Low-Power Peripheral Clock Divides. The LPCLKDIV bits select the divider values for each peripheral group. Table 6-27 defines the module groups that are affect by LPCLKDIV _n . 00 Divide by 1 01 Divide by 2 10 Divide by 4 11 Divide by 8

Table 6-27. LPCLKDIV Module Groups

LPCLKDIV _n	Modules
LPCLKDIV0	FlexCAN_A, DSPI_A
LPCLKDIV1	ESCI_A, I ² C_A, PIT
LPCLKDIV2	FlexCAN_B-F
LPCLKDIV3	DSPI_B-D
LPCLKDIV4	ESCI_B-H
LPCLKDIV5	eMIOS
LPCLKDIV6	MLB
LPCLKDIV7	Reserved

6.3.2.26 Halt Register (SIU_HLT)

The SIU_HLT register is used to disable the clocks to various modules. Each bit drives a separate halt request to the associated peripheral. [Table 6-28](#) shows these connected outputs.

Offset: SIU_BASE + 0x09A4

Access: User read/write

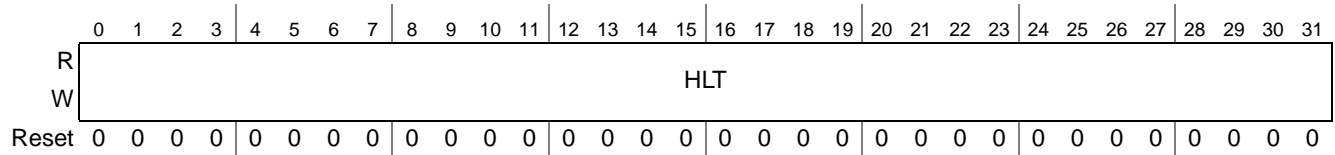


Figure 6-29. Halt Register (SIU_HLT)

Table 6-28. HALT Register Field Descriptions

Field	Description
HLT	<p>Halt Selects. The HLT bits halt specific modules. Each bit corresponds to a separate module, as mapped below.</p> <ul style="list-style-type: none"> 0 Reserved 1 Reserved 2 FLEXRAY 3 DMA 4 Reserved 5 Reserved 6 NPC 7 EBI 8 EQADC 9 MLB 10 EMIOS200 11 Reserved 12 I²C_A 13 PIT 14 FLEXCAN_F 15 FLEXCAN_E 16 FLEXCAN_D 17 FLEXCAN_C 18 FLEXCAN_B 19 FLEXCAN_A 20 DSPI_D 21 DSPI_C 22 DSPI_B 23 DSPI_A 24 ESCI_H 25 ESCI_G 26 ESCI_F 27 ESCI_E 28 ESCI_D 29 ESCI_C 30 ESCI_B 31 ESCI_A <p>Note: Writes to reserved HLT bits 4, 5, and 11 are reflected in the reserved HLTACK bits 4, 5, and 11.</p>

6.3.2.27 Halt Acknowledge Register (SIU_HLTACK)

The SIU_HLTACK bits indicate that the peripheral requested to halt via the HLT bit has completed the halt process and has entered a halted state with the peripheral clocks disabled. The HLTACK bits are read-only

and writes have no effect. The halt acknowledge from each peripheral is connected, as shown in Table 6-29.

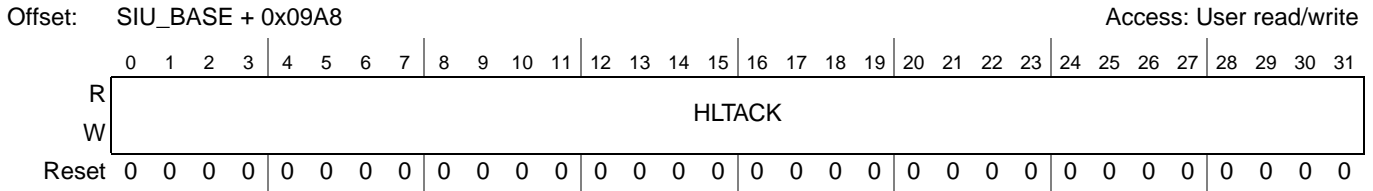


Figure 6-30. Halt Acknowledge Register (SIU_HLTACK)

Table 6-29. HLTACK Register Field Descriptions

Field	Description
HLTACK	<p>Halt Flags. Each bit corresponds to a separate module, as mapped below.</p> <ul style="list-style-type: none"> 0 e200z1 1 e200z0 2 FLEXRAY 3 DMA 4 Reserved 5 Reserved 6 NPC 7 EBI 8 EQADC 9 MLB 10 EMIOS200 11 Reserved 12 I²C_A 13 PIT 14 FLEXCAN_F 15 FLEXCAN_E 16 FLEXCAN_D 17 FLEXCAN_C 18 FLEXCAN_B 19 FLEXCAN_A 20 DSPI_D 21 DSPI_C 22 DSPI_B 23 DSPI_A 24 ESCI_H 25 ESCI_G 26 ESCI_F 27 ESCI_E 28 ESCI_D 29 ESCI_C 30 ESCI_B 31 ESCI_A <p>Note: Writes to reserved HLT bits 4, 5, and 11 are reflected in the reserved HLTACK bits 4, 5, and 11.</p>

6.3.2.28 Parallel GPIO Pin Data Output Register 0 (SIU_PGPDO0)

The SIU_PGPDO0 register contains the parallel GPIO pin data output for PB[0:15].

Reads and writes to this register are coherent with the registers SIU_GPDO16_19, SIU_GPDO20_23, SIU_GPDO24_27, and SIU_GPDO28_31.

NOTE

On MPC5510, the port A pins are general-purpose inputs only. Therefore, there are no parallel GPIO pin data output register bits for port A.

Offset: SIU_BASE + 0xC00 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PB0:PB15															
W	PB0:PB15															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-31. Parallel GPIO Pin Data Output Register 0 (SIU_PGPDO0)

6.3.2.29 Parallel GPIO Pin Data Output Register 1 (SIU_PGPDO1)

The SIU_PGPDO1 register contains the parallel GPIO pin data output for PC0:PC15 and PD0:PD15.

Reads and writes to this register are coherent with the registers SIU_GPDO32_35, SIU_GPDO36_39, SIU_GPDO40_43, SIU_GPDO44_47, SIU_GPDO48_51, SIU_GPDO52_55, SIU_GPDO56_59, and SIU_GPDO60_63.

Offset: SIU_BASE + 0x0C04 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PC0:PC15															
W	PC0:PC15															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PD0:PD15															
W	PD0:PD15															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-32. Parallel GPIO Pin Data Output Register 1 (SIU_PGPDO1)

6.3.2.30 Parallel GPIO Pin Data Output Register 2 (SIU_PGPDO2)

The SIU_PGPDO2 register contains the Parallel GPIO Pin Data Output for PE0:PE15 and PF0:PF15.

Reads and writes to this register are coherent with the registers SIU_GPDO64_67, SIU_GPDO68_71, SIU_GPDO72_75, SIU_GPDO76_79, SIU_GPDO80_83, SIU_GPDO84_87, SIU_GPDO88_91, and SIU_GPDO92_95.

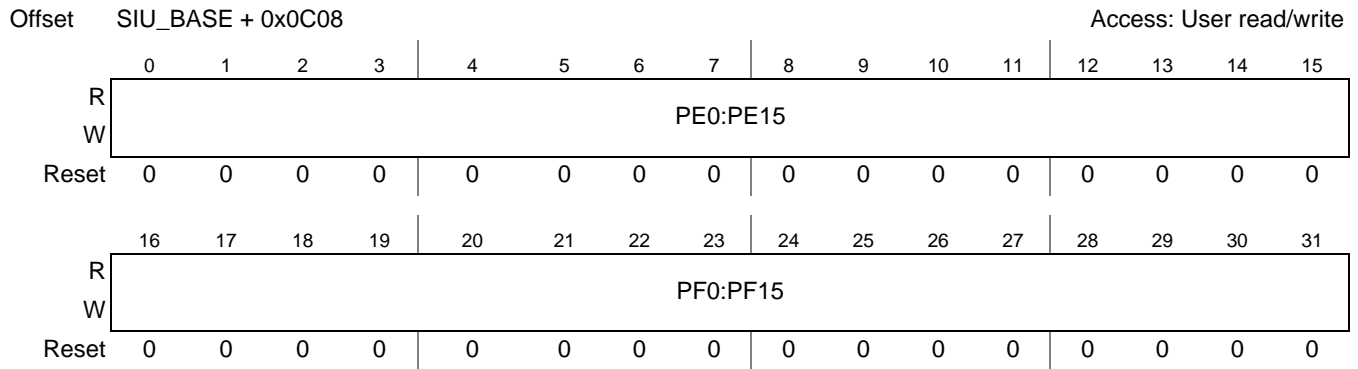


Figure 6-33. Parallel GPIO Pin Data Output Register 2 (SIU_PGPDO2)

6.3.2.31 Parallel GPIO Pin Data Output Register 3 (SIU_PGPDO3)

The SIU_PGPDO3 register contains the parallel GPIO pin data output for PG0:PG15 and PH0:PH15.

Reads and writes to this register are coherent with the registers SIU_GPDO96_99, SIU_GPDO100_103, SIU_GPDO104_107, SIU_GPDO108_111, SIU_GPDO112_115, SIU_GPDO116_119, SIU_GPDO120_123, and SIU_GPDO124_127.

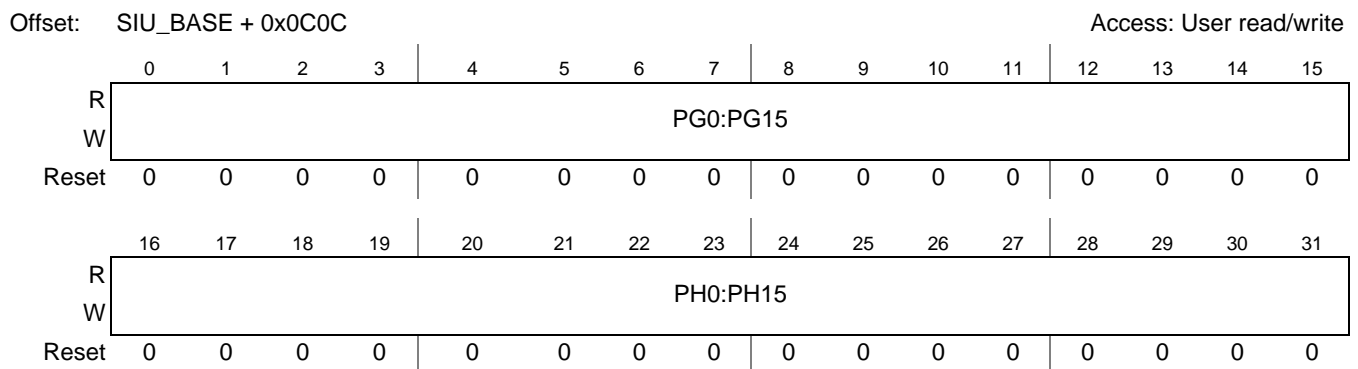


Figure 6-34. Parallel GPIO Pin Data Output Register 3 (SIU_PGPDO3)

6.3.2.32 Parallel GPIO Pin Data Output Register 4 (SIU_PGPDO4)

The SIU_PGPDO4 register contains the parallel GPIO pin data output for PJ0:PJ15.

Reads and writes to this register are coherent with the registers SIU_GPDO18_131, SIU_GPDO132_135, SIU_GPDO136_139, and SIU_GPDO140_143.

NOTE

On MPC5510, the port K pins are only inputs. Therefore, there are no parallel GPIO pin data output bits associated with port K.

Offset: SIU_BASE + 0x0C10

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PJ0:PJ15															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-35. Parallel GPIO Pin Data Output Register 4 (SIU_PGPDO4)

6.3.2.33 Parallel GPIO Pin Data Input Register 0 (SIU_PGPDIO)

Reads to the SIU_PGPDIO register provide the parallel GPIO pin data input for PA0:PA15 and PB0:PB15. Writes have no effect.

Reads of this register are coherent with the registers SIU_GPDI0_3, SIU_GPDI4_7, SIU_GPDI8_11, SIU_GPDI12_15, SIU_GPDI16_19, SIU_GPDI20_23, SIU_GPDI24_27, and SIU_GPDI28_31.

Offset: SIU_BASE + 0x0C40

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PA0:PA15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PB0:PB15															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 6-36. Parallel GPIO Pin Data Input Register 0 (SIU_PGPDIO)

6.3.2.34 Parallel GPIO Pin Data Input Register 1 (SIU_PGPDIO1)

Reads to the SIU_PGPDIO1 register provide the parallel GPIO pin data input for PC0:PC15 and PD0:PD15. Writes have no effect.

Reads of this register are coherent with the registers SIU_GPDI32_35, SIU_GPDI36_39, SIU_GPDI40_43, SIU_GPDI44_47, SIU_GPDI48_51, SIU_GPDI52_55, SIU_GPDI56_59, and SIU_GPDI60_63.

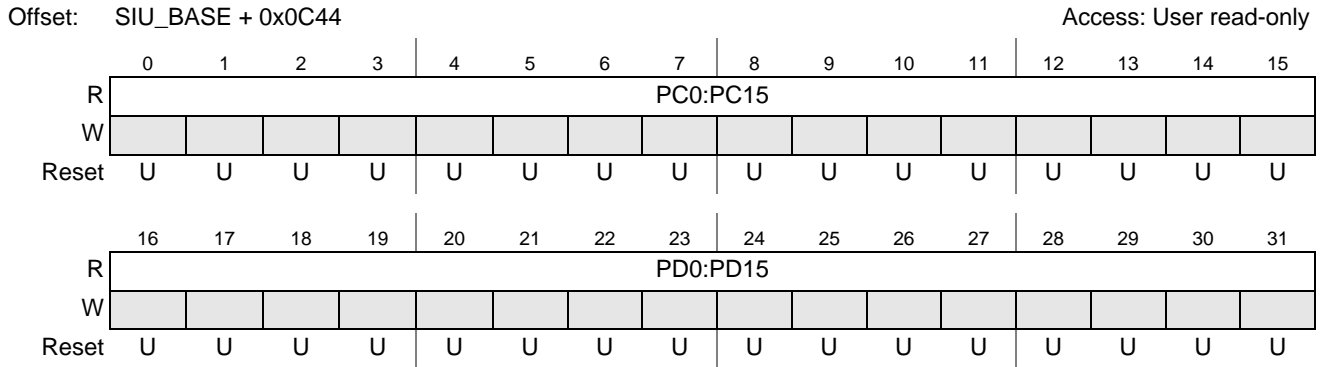


Figure 6-37. Parallel GPIO Pin Data Input Register 1 (SIU_PGPDI1)

6.3.2.35 Parallel GPIO Pin Data Input Register 2 (SIU_PGPDI2)

Reads to the SIU_PGPDI2 register provide the parallel GPIO pin data input for PE0:PE15 and PF0:PF15. Writes have no effect.

Reads of this register are coherent with the registers SIU_GPDI64_67, SIU_GPDI68_71, SIU_GPDI72_75, SIU_GPDI76_79, SIU_GPDI80_83, SIU_GPDI84_87, SIU_GPDI88_91, and SIU_GPDI92_95.

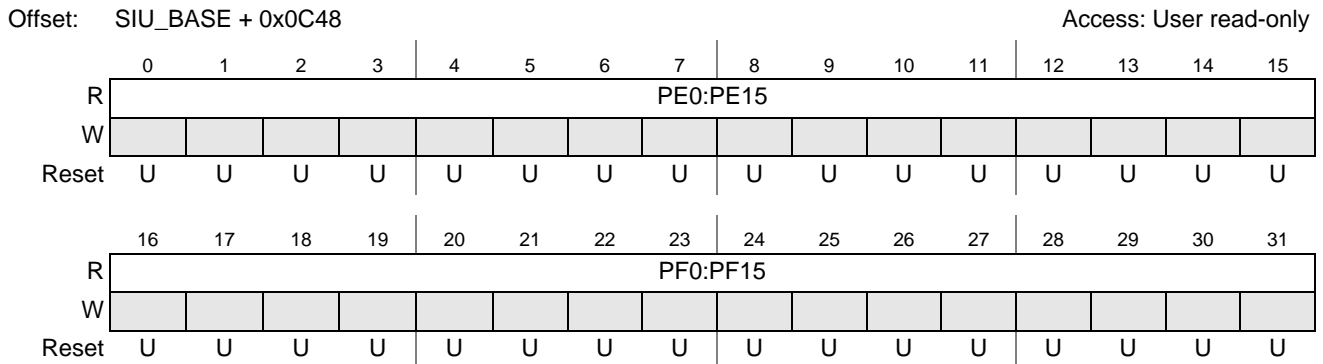


Figure 6-38. Parallel GPIO Pin Data Input Register 2 (SIU_PGPDI2)

6.3.2.36 Parallel GPIO Pin Data Input Register 3 (SIU_PGPDI3)

Reads to the SIU_PGPDI2 register provide the parallel GPIO pin data input for PG0:PG15 and PH0:PH15. Writes have no effect.

Reads of this register are coherent with the registers SIU_GPDI96_99, SIU_GPDI100_103, SIU_GPDI104_107, SIU_GPDI108_111, SIU_GPDI112_115, SIU_GPDI116_119, SIU_GPDI120_123, and SIU_GPDI124_127.

Offset: SIU_BASE + 0x0C4C

Access: User read-only

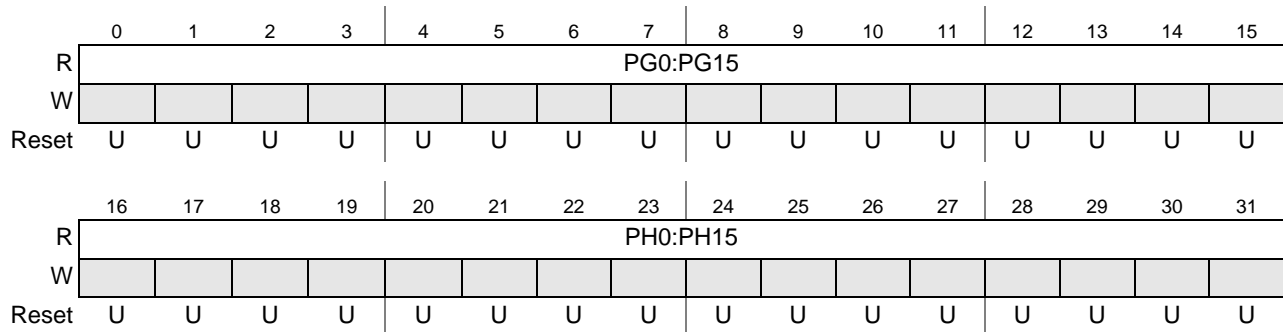


Figure 6-39. Parallel GPIO Pin Data Input Register 3 (SIU_PGPD13)

6.3.2.37 Parallel GPIO Pin Data Input Register 4 (SIU_PGPD14)

Reads to the SIU_PGPD14 register provide the parallel GPIO pin data input for PJ0:PJ15 and PK0:PK1. Writes have no effect.

Reads of this register are coherent with the registers SIU_GPDI128_131, SIU_GPDI132_135, SIU_GPDI136_139, SIU_GPDI140_143, and SIU_GPDI144_145.

Offset: SIU_BASE + 0x0C50

Access: User read-only

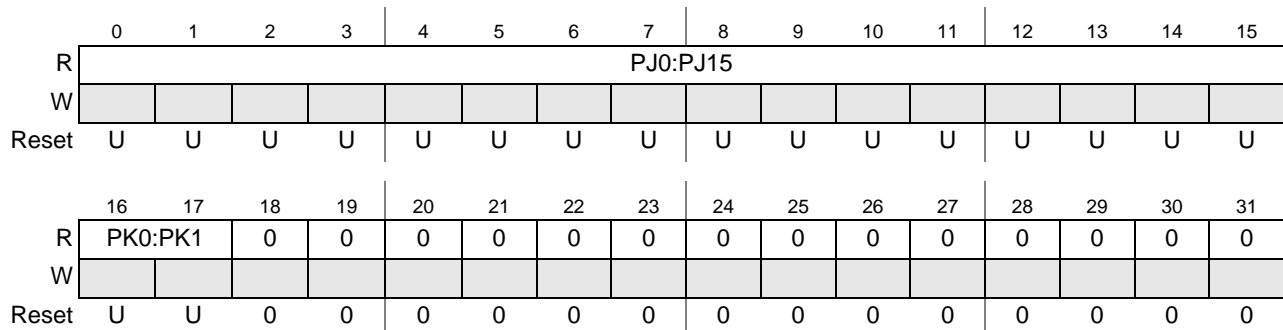


Figure 6-40. Parallel GPIO Pin Data Input Register 4 (SIU_PGPD14)

6.3.2.38 Masked Parallel GPIO Pin Data Output Registers

The purpose of these registers is to allow any combination of bits in a 16-bit parallel GPIO pin data output port to be updated in a single 32-bit write operation, while allowing other bits to maintain their previous state. This is accomplished by grouping each 16-bit port with a 16-bit mask register, and only updating those bits in the data register for which the corresponding mask bit is set.

For example, if the current state of the port B parallel GPIO pin data output register is 0x1234 and you want to change only bits [12:15] (i.e., the 4) to be an 8, then a 32-bit write with a mask value of 0x000C and data value of 0x0008 (i.e. 0x000C_0008) would be performed.

This register always reads as 0.

6.3.2.38.1 Masked Parallel GPIO Pin Data Output Register 1 (SIU_MPGPDO1)

The SIU_MPGPDO1 register contains the Masked Parallel GPIO Pin Data Output for PB[0:15].

Writes to this register are coherent with the registers SIU_GPDO16_19, SIU_GPDO20_23, SIU_GPDO24_27, and SIU_GPDO28_31.

Offset: SIU_BASE + 0x0C84 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PB_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PB[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-41. Masked Parallel GPIO Pin Data Output Register 1 (SIU_MPGPDO1)

6.3.2.38.2 Masked Parallel GPIO Pin Data Output Register 2 (SIU_MPGPDO2)

The SIU_MPGPDO2 register contains the masked parallel GPIO pin data output for PC[0:15].

Writes to this register are coherent with the registers SIU_GPDO32_35, SIU_GPDO36_39, SIU_GPDO40_43, and SIU_GPDO44_47.

Offset: SIU_BASE + 0x0C88 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PC_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PC[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-42. Masked Parallel GPIO Pin Data Output Register 2 (SIU_MPGPDO2)

6.3.2.38.3 Masked Parallel GPIO Pin Data Output Register 3 (SIU_MPGPDO3)

The SIU_MPGPDO3 register contains the masked parallel GPIO pin data output for PD[0:15].

Writes to this register are coherent with the registers SIU_GPDO48_51, SIU_GPDO52_55, SIU_GPDO56_59, and SIU_GPDO60_63.

Offset: SIU_BASE + 0x0C8C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PD_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PD[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-43. Masked Parallel GPIO Pin Data Output Register 3 (SIU_MPGPDO3)

6.3.2.38.4 Masked Parallel GPIO Pin Data Output Register 4 (SIU_MPGPDO4)

The SIU_MPGPDO4 register contains the masked parallel GPIO pin data output for PE[0:15].

Writes to this register are coherent with registers SIU_GPDO64_67, SIU_GPDO68_71, SIU_GPDO72_75, and SIU_GPDO76_79.

Offset: SIU_BASE + 0x0C90

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PE_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PE[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-44. Masked Parallel GPIO Pin Data Output Register 4 (SIU_MPGPDO4)

6.3.2.38.5 Masked Parallel GPIO Pin Data Output Register 5 (SIU_MPGPDO5)

The SIU_MPGPDO5 register contains the masked parallel GPIO pin data output for PF[0:15].

Writes to this register are coherent with registers SIU_GPDO80_83, SIU_GPDO84_87, SIU_GPDO88_91, and SIU_GPDO92_95.

Offset: SIU_BASE + 0x0C94

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PF_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PF[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-45. Masked Parallel GPIO Pin Data Output Register 5 (SIU_MPGPDO5)

6.3.2.38.6 Masked Parallel GPIO Pin Data Output Register 6 (SIU_MPGPDO6)

The SIU_MPGPDO6 register contains the masked parallel GPIO pin data output for PG[0:15]

Writes to this register are coherent with registers SIU_GPDO96_99, SIU_GPDO100_103, SIU_GPDO104_107, and SIU_GPDO108_111.

Offset: SIU_BASE + 0x0C98

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PG_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PG[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-46. Masked Parallel GPIO Pin Data Output Register 6 (SIU_MPGPDO6)

6.3.2.38.7 Masked Parallel GPIO Pin Data Output Register 7 (SIU_MPGPDO7)

The SIU_MPGPDO7 register contains the masked parallel GPIO pin data output for PH[0:15].

Writes to this register are coherent with registers SIU_GPDO112_115, SIU_GPDO116_119, SIU_GPDO120_123, and SIU_GPDO124_127.

Offset: SIU_BASE + 0xC9C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PH_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PH[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-47. Masked Parallel GPIO Pin Data Output Register 7 (SIU_MPGPDO7)

6.3.2.38.8 Masked Parallel GPIO Pin Data Output Register 8 (SIU_MPGPDO8)

The SIU_MPGPDO8 register contains the masked parallel GPIO pin data output for PJ[0:15].

Writes to this register are coherent with registers SIU_GPDO128_131, SIU_GPDO132_135, SIU_GPDO136_139, and SIU_GPDO140_143.

Offset: SIU_BASE + 0x0CA0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PJ_MASK[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PJ[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-48. Masked Parallel GPIO Pin Data Output Register 8 (SIU_MPGPDO8)

6.4 Functional Description

The following sections provide an overview of the SIU operation.

6.4.1 System Configuration

6.4.1.1 Boot Configuration

During the assertion of $\overline{\text{RESET}}$, the BOOTCFG pin is used to load a value into the SIU_RSR[BOOTCFG] bit, so the BAM program can determine the location of the reset configuration half word (RCHW), the boot mode to be initiated, and whether to initiate a CAN or SCI boot. See [Section 32.3.3.1.1, “Reset Configuration Halfword Read”](#) of the BAM chapter for detail on the RCHW. [Table 6-30](#) defines the boot modes specified by the SIU_RST[BOOTCFG] field.

Table 6-30. SIU_RSR[BOOTCFG] Configuration

Value	Meaning
0b0	Boot from internal flash memory
0b1	CAN/SCI boot

6.4.1.2 Pad Configuration

The pad configuration registers (SIU_PCR) in the SIU allow software control of the static electrical characteristics of external pins. The PCRs can select the multiplexed function of a pin, selection of pullup or pulldown devices, the slew rate of I/O signals, open drain mode for output pins, and hysteresis.

6.4.2 Reset Control

The reset controller logic is located in the SIU. See [Section 7.4, “Reset Configuration,”](#) for reset operation details.

6.4.3 External Interrupt

There are sixteen external interrupt inputs, IRQ0–IRQ15, to the SIU. The IRQ n inputs can be configured for rising- or falling-edge events or both. Each IRQ n input has a corresponding flag bit in the external interrupt status register (SIU_EISR). The flag bits for the IRQ4–IRQ15 inputs are ORed together to form one interrupt request to the interrupt controller. The flag bits for the IRQ1–IRQ4 inputs can generate an interrupt request to the interrupt controller or a DMA transfer request to the DMA controller. The flag bit for IRQ0 can generate an interrupt request if SIU_DIRSR[31] is 0, or is disabled if SIU_DIRSR[0] is 1. [Figure 6-49](#) shows the DMA and interrupt request connections to the interrupt and DMA controllers.

Any pin used as an external interrupt must be configured in its SIU_PCR as a GPIO in input mode. In addition, either rising and/or falling edge must be enabled in the SIU_IREER, or SIU_IFEER.

Two external inputs from pins PD11 and PD10 connect through the SIU to the critical interrupt input to the Z0 and Z1 cores, respectively. These signals should be used as non maskable interrupt (NMI) inputs.

The SIU contains an overrun interrupt enable for each IRQ and one combined overrun interrupt request to the interrupt controller which is the logical OR of the individual overrun requests' flags. Only the combined overrun interrupt request is used in the device, and the individual overrun requests are not connected.

Each IRQ pin has a programmable filter for rejecting glitches on the IRQ signals. The filter length for the IRQ pins is specified in the external IRQ digital filter register (SIU_IDFR).

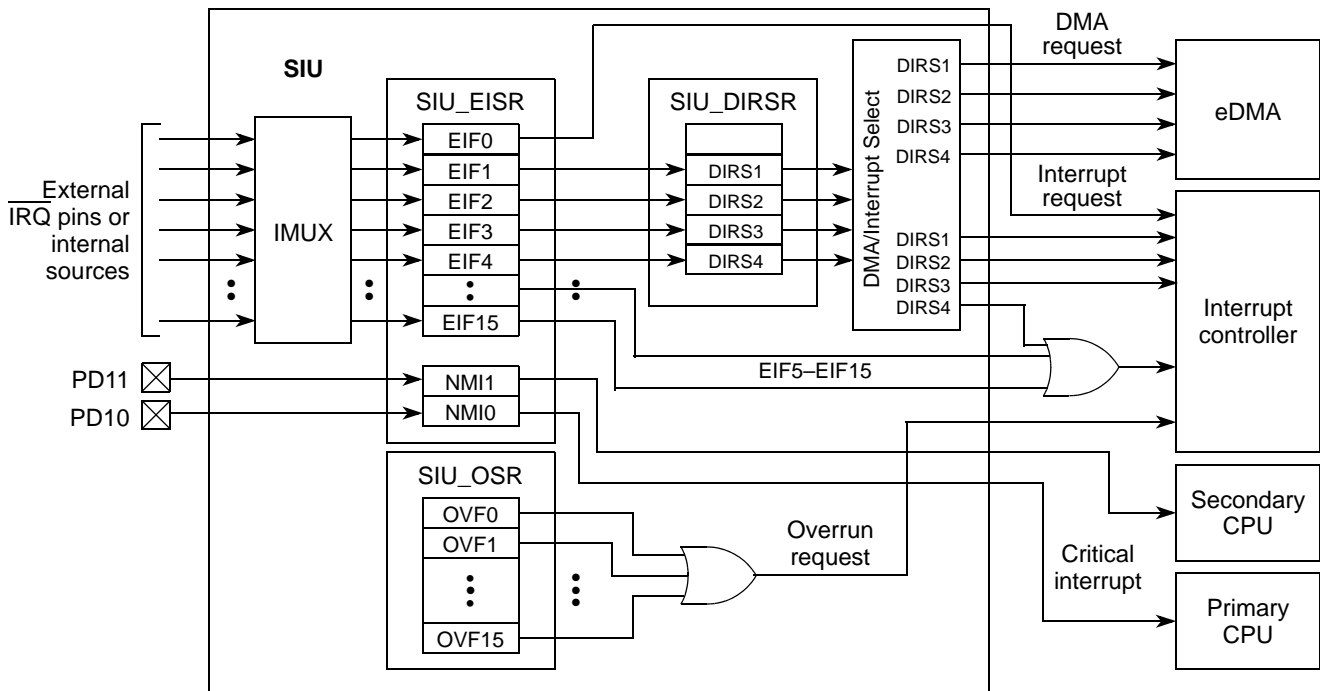


Figure 6-49. SIU DMA/Interrupt Request Diagram

6.4.4 GPIO Operation

All GPIO functionality is provided by the SIU. Each pin that has GPIO functionality has an associated Pin Configuration Register in the SIU where the GPIO function is selected for the pin. In addition, each pin with GPIO functionality has an input data register (SIU_GPDI x_x) and an output data register (SIU_GPDO x_x). The SIU also implements several parallel GPIO registers (SIU_PGPDO x_x and SIU_PGPDIX x_x) that can be used to access up to 32 GPIO bits in a single- and word-sized accesses. The values read/written to these parallel register is coherent with the data read/written to the SIU_GPDO x_x and SIU_GPDI x_x registers.

6.4.5 Internal Multiplexing

The IMUX Select Registers (SIU_ISEL x) provide selection of the input source for the eQADC external trigger inputs and the SIU external interrupts.

6.4.5.1 eQADC External Trigger Input Multiplexing

The four eQADC external trigger inputs can be connected to two different external pins or one of two PIT channels. The input source for each eQADC external trigger is individually specified in the IMUX Select Register 0 (SIU_ISEL0). [Figure 6-50](#) gives an example of the multiplexing of an eQADC external trigger input. As shown in the figure, the ETRIG[0] input of the eQADC can be connected to the PC4 pin, the PG4 pin, the PIT7 channel, or the PIT8 channel. Remaining ETRIG inputs are multiplexed in the same manner.

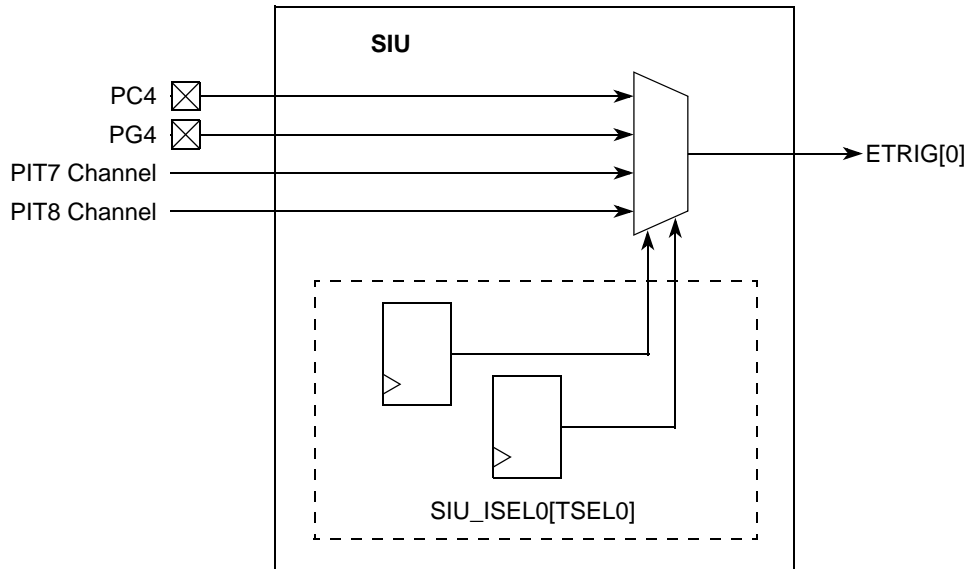


Figure 6-50. Four-to-One Internal Multiplexing Block Diagram

6.4.5.2 SIU External Interrupt Input Multiplexing

The 16 SIU external interrupt inputs can be connected to one of four external pins. The input source for each SIU external interrupt is individually specified in the IMUX Select Register 1 (SIU_ISEL1).

Figure 6-51 shows an example of the multiplexing of an SIU external interrupt input. As shown in the figure, the IRQ[0] input of the SIU can be connected to the PA0 pin, PD0 pin, PD10, or PG11 pin. The remaining IRQ inputs are multiplexed in the same manner.

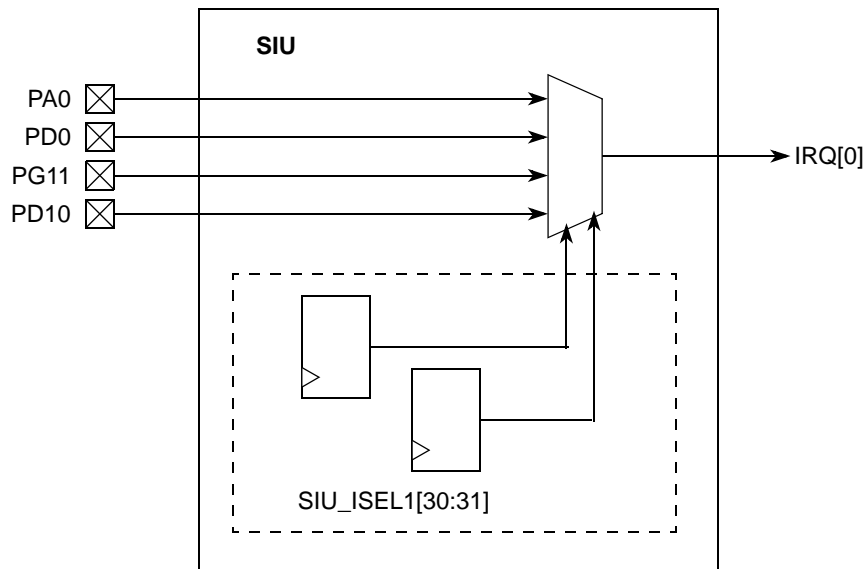


Figure 6-51. SIU External Interrupt Input Multiplexing

Chapter 7

Reset

7.1 Introduction

The reset sources supported in the MPC5510 are:

- Power-on reset (POR)
- Low-voltage inhibit (LVI) reset
- External reset
- Loss-of-lock reset
- Loss-of-clock reset
- Watchdog timer
- JTAG reset
- Checkstop reset (both Z1 and Z0 cores)
- Software-system reset

All reset sources are processed by the reset controller, which is located in the SIU module ([Chapter 6, “System Integration Unit \(SIU\)”](#)). The reset controller monitors the reset input sources and, upon detection of a reset event, resets internal logic and controls the assertion of the $\overline{\text{RESET}}$ pin.

The MCU is clocked by the 16 MHz IRC clock after any reset.

The reset status register (SIU_RSR) gives the source, or sources, of the last reset and is updated for all reset sources except JTAG reset

The BOOTCFG pin controls the MCU boot sequence after the POR or if the Z1 reset vector points to the BAM. If the pin is driven low during the MCU reset, the MCU boots from internal flash and the Reset Configuration Halfword (RCHW) controls the boot sequence. The RCHW needs to be programmed by user in internal flash in one of predefined locations together with the user application start address.

If the pin is driven high, the BAM executes serial boot sequence.

See [Chapter 32, “Boot Assist Module \(BAM\)”](#) for more details about the boot procedures.

7.2 External Signal Description.

Refer to [Table 2-1](#) and [Section 2.7, “Detailed External Signal Descriptions,”](#) for signal properties.

7.2.1 Reset ($\overline{\text{RESET}}$)

This pin provides the system reset. It is an open-drain, active-low bidirectional pin. It acts as an input to initialize the MCU to a known start-up state, and an output when an internal MCU function causes a reset. Externally asserting the $\overline{\text{RESET}}$ pin will asynchronously reset the chip. The chip will remain in reset as long as the external $\overline{\text{RESET}}$ pin is asserted. Any internal reset event will assert the $\overline{\text{RESET}}$ pin for as long as the reset event is active. When the internal reset sources are negated, the $\overline{\text{RESET}}$ pin will be asserted by the reset controller for a predefined time (2400 clocks), then the reset controller will stop asserting the $\overline{\text{RESET}}$ pin. After another predefined time, the $\overline{\text{RESET}}$ pin is sampled, and if still asserted then an external reset request is assumed. When the $\overline{\text{RESET}}$ pin is sampled high (the pin is no longer being driven low by the MPC5510 reset logic or by external logic that might be requesting reset), the reset configuration pin (pin PD2) is sampled and the internal reset to the chip negates.

7.2.2 Boot Configuration (BOOTCFG)

The BOOTCFG pin (pin name PD2 in package diagrams and signal lists) is used to determine the boot mode initiated by the BAM program. The pin state during reset is latched in the SIU_RSR[BOOTCFG] field. The BAM program uses the BOOTCFG field to determine whether initiate internal flash boot mode or a CAN or SCI “serial” boot.

Refer to [Section 6.3.2.2, “Reset Status Register \(SIU_RSR\),”](#) and [Section 6.4.1.1, “Boot Configuration,”](#) for more information.

NOTE

The reset controller latches the state of the BOOTCFG pin into the SIU_RSR register 4 clock cycles prior to the negation of $\overline{\text{RESET}}$.

7.3 Functional Description

7.3.1 Z1, Z0 Cores Reset Vectors

The reset vectors for the Z1 and Z0 cores in the MPC5510 MCU are controlled via the Z1VEC and Z0VEC registers in the Clock, Reset, and Power control (CRP) module. The power-on reset values for the Z1VEC and Z0VEC registers point to the first instruction of the BAM program.

The Z0 core is disabled after the reset occurs and Z1 is active. Thus, following the reset, the Z1 core starts to execute the BAM code. See [Chapter 32, “Boot Assist Module \(BAM\)”](#) for more details about the boot process.

The values in the Z1VEC and Z0VEC registers can be changed so the Z1 and Z0 cores begin code execution at any desired memory location after any exit from Sleep. This can reduce the start-up time upon a low-power mode exit by pointing the cores directly to a low-power mode recovery routine.

7.3.2 Reset Sources

7.3.2.1 Power-on Reset (POR)

The internal power-on reset signal is asserted when the voltage on the 5 V VDDA supply is below defined values. See the *MPC5510 Microcontroller Family Data Sheet* and [Section 7.3.2.2, “Low-Voltage Inhibit \(LVI\) Resets.”](#)

7.3.2.2 Low-Voltage Inhibit (LVI) Resets

The internal LVI reset signals are asserted when the voltage on the corresponding supply is below defined values. The following are the LVI resets:

- LVI15S: LVI on 1.5 V supply
- LVI33S: LVI on 3.3 V supply (used for 3.3 V power to internal I/O pad logic)
- LVI33SYNS: LVI on 3.3 V supply (used for VDDSYN)
- LVI5S: LVI on 5 V VDDA supply (nominal trip point V_{LV5A})
- LVI5CS: LVI on 5 V VDDA supply (nominal trip point 4.0 V used during crank operation)

7.3.2.3 External Reset

When the reset controller detects assertion of the $\overline{\text{RESET}}$ pin, the internal reset signal is asserted. The SIU_RSR[ERS] bit is set, and all other reset status bits in the SIU_RSR are cleared.

7.3.2.4 Loss-of-Lock Reset

A loss-of-lock reset occurs when the PLL loses lock and the loss-of-lock reset enable (LOLRE) bit in the PLL enhanced synthesizer control register 2 (ESYNCR2) is set. The internal reset signal and $\overline{\text{RESET}}$ pin are asserted. The SIU_RSR[LLRS] bit is set, and all other reset status bits in the SIU_RSR are cleared.

7.3.2.5 Loss-of-Clock Reset

A loss-of-clock reset occurs when a failure is detected in either the reference clock signal or PLL output when the PLL is enabled. The internal reset signal and $\overline{\text{RESET}}$ pin are asserted. The SIU_RSR[LCRS] bit is set, and all other reset status bits in the SIU_RSR are cleared.

7.3.2.6 Watchdog Timer

A watchdog timer reset occurs when the miscellaneous controller module (MCM) SWT watchdog timer is enabled and is not serviced properly. The affect of a watchdog timer reset is the same for the reset controller. The internal reset signal and $\overline{\text{RESET}}$ pin are asserted. The SIU_RSR[WTRS] bit is set, and all other reset status bits in the SIU_RSR are cleared.

7.3.2.7 Checkstop Reset

When the Z1 or Z0 core enters a checkstop state, and the checkstop reset is enabled (SIU_SRCR[CRE0] bit for Z1 and the SIU_SRCR[CRE1] bit for Z0), a checkstop reset occurs. The internal reset signal and

$\overline{\text{RESET}}$ pin are asserted. The SIU_RSR[CRS] bit is set and all other reset status bits in the SIU_RSR are cleared.

7.3.2.8 JTAG Reset

A system reset occurs when JTAG is enabled and either the EXTEST, CLAMP, or HIGHZ instructions are executed by the JTAG controller. The internal reset signal is asserted. The state of the $\overline{\text{RESET}}$ pin is determined by the JTAG instruction. The reset status bits in the SIU_RSR are unaffected.

7.3.2.9 Software System Reset

A software system reset is caused by writing to the SIU_RCR[SSR] bit. Setting the SSR bit causes an internal reset of the MCU. The internal reset signal and $\overline{\text{RESET}}$ pin are asserted. The SIU_RSR[SSRS] bit is set, and all other reset status bits in the SIU_RSR are cleared.

7.4 Reset Configuration

The reset state of the system is:

- All pads on ports A–K are placed in a disabled mode with output enables, input enables, and pull devices all disabled, except PD2.
- TDI pad is an input with pullup enabled.
- TDO pad is an output with fastest slew rate selected.
- TCK pad is an input with pulldown enabled.
- TMS pad is an input with pullup enabled.
- JCOMP pad is an input with pulldown enabled.
- $\overline{\text{RESET}}$ pin is configured as open drain output with pullup disabled and initially driven low, but switched to an input with pullup enabled after the reset sequence.
- BOOTCFG pin is an input, the pin data is latched 4 clock cycles before the $\overline{\text{RESET}}$ signal is negated (high).

7.4.1 Reset Configuration Timing

The timing diagram in [Figure 7-1](#) shows the sampling of the BOOTCFG (PD2) pin for a power-on reset. The timing diagram is also valid for internal/external resets assuming VDD, VDD33, and VDDA are within valid operating ranges. The value of the BOOTCFG pin is latched 4 clock cycles before the negation of the $\overline{\text{RESET}}$ pin and stored in the reset status register.

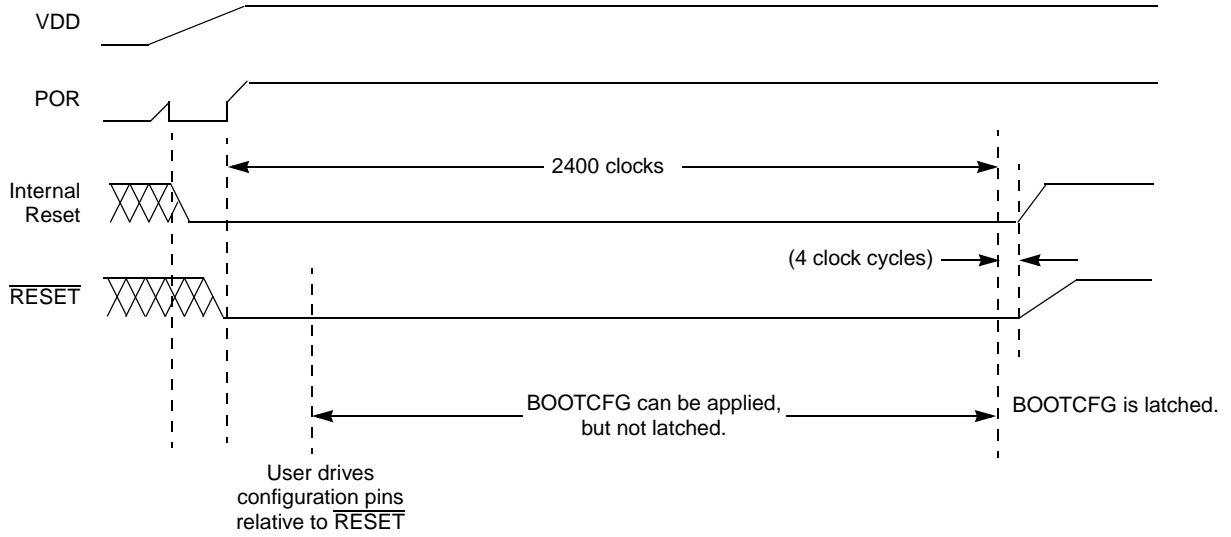


Figure 7-1. Reset Configuration Timing

Chapter 8

Interrupts

8.1 Introduction

Interrupt functionality is handled between the e200z1 (Z1) and e200z0 (Z0) cores and the interrupt controller (INTC). The INTC has a set of configuration bits that allows any interrupt source to generate an interrupt request to either the Z1 or Z0 or to both the Z1 and Z0 cores. The INTC has two independent sets of priority arbitration/comparison, request selection, vector encoder and acknowledge logic—one set for each CPU. This allows each CPU to handle its software-assigned interrupt requests independently of the other CPU's operation, and provides flexibility for the user to decide which core should handle which interrupt sources in the application.

Here is an example of an interrupt handling approach between the e200z1 and e200z0 cores.

- Z0 handles all interrupts associated with eSCI, DSPI, FlexCAN, and FlexRay modules.
 - Z0 can also handle eMIOS and eQADC interrupts if there is available bandwidth.
- Z1 handles interrupts associated with application code — eMIOS and eQADC.
 - Z1 can handle communication interrupts if desired.
- By allowing the user to choose which core handles which interrupt, the user is not restricted to using the Z0 core only as Freescale has envisioned its use.
- INTC has two independent interrupt request outputs — one for each core.
 - Implies two sets of priority selection, vector encoding, priority level FIFOs, etc.
 - Priority level configuration register also contains bits to select which core is interrupted when a source (including software interrupts) asserts an interrupt request. (Note that an interrupt source can be selected to interrupt both cores; in such cases, the user must take special care not to cause spurious interrupts on the other core when servicing the interrupt.)
 - Out of RESET, all interrupt requests are steered to Z1 (for compatibility with parts without Z0).
 - User decides which core will handle which interrupts in their application.
 - Care must be taken when dynamically switching core handling interrupts (i.e. make sure interrupt source is disabled before switching so that both cores don't become interrupted).
 - Software interrupts would be used for inter processor signaling.
 - A use-case for the MPC5510: use the DMA for simple data movement, the Z0 for data movement with some intermediate processing, and the Z1 for the main algorithm.

The details of the INTC operation are given in [Chapter 9, “Interrupt Controller \(INTC\).”](#)

Two types of modes are used to learn the interrupt request source's vector number: software vector mode and hardware vector mode. Software vector mode is the mode that conforms to Power Architecture technology. The e200z1/z0 branches to a common interrupt exception handler to service the interrupt

request. The interrupt exception handler reads the INTC_IACKR to learn the vector of the source of the interrupt request. In hardware vector mode, the interrupt exception handler is unique to the interrupt request source’s vector.

8.2 Interrupt Vectors

The core interrupt vectors are located on a 4 KB boundary in the memory map, with the hardware interrupt vectors located 2 KB above the core interrupt vectors (see [Figure 8-1](#)).

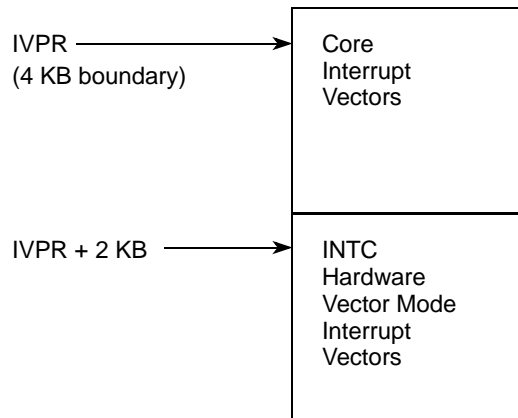


Figure 8-1. MPC5510 Interrupt Vector Memory Map

8.2.1 Core Interrupts

Table 8-1. MPC5510 Core Interrupt Vector Memory Map

Core Interrupt Type	IVOR # ¹	VPR Offset	Enables ²	State Saved In	Examples
Critical Input	IVOR 0	0x000	CE	CSRR[0:1]	Non maskable interrupt (pins PD[10], PD[11])
Machine Check	IVOR 1	0x010	ME	CSRR[0:1]	ISI, ITLB error on first instruction of exception handler
Data Storage	IVOR 2	0x020	—	SRR[0:1]	Incorrect privilege mode for R/W access
Instruction Storage	IVOR 3	0x030	—	SRR[0:1]	Incorrect privilege mode for instruction
External Input ³	IVOR 4	0x040	EE, src	SRR[0:1]	Peripherals, IRQ pins, software
Alignment	IVOR 5	0x050	—	SRR[0:1]	Load or store operand not word aligned
Program	IVOR 6	0x060	—	SRR[0:1]	Illegal instruction, trap
Floating Point Unavailable ⁴	IVOR 7	0x070	—	SRR[0:1]	FP instruction attempt with MSR[FP]=0
System Call	IVOR 8	0x080	—	SRR[0:1]	System call, “sc”, instruction
Decrementer ⁴	IVOR 10	0x0A0	EE, DIE	SRR[0:1]	Decrementer timeout
Fixed Interval Timer ⁴	IVOR 11	0x0B0	EE, FIE	SRR[0:1]	Fixed-interval timer timeout
Watchdog Timer ⁴	IVOR 12	0x0C0	CE, WIE	CSRR[0:1]	Watchdog timeout when ENW=1, WIS=0

Table 8-1. MPC5510 Core Interrupt Vector Memory Map (continued)

Core Interrupt Type	IVOR # ¹	VPR Offset	Enables ²	State Saved In	Examples
Data TLB Error ⁴	IVOR 13	0x0D0	—	SRR[0:1]	Data TLB miss in MMU
Instruction TLB Error ⁴	IVOR 14	0x0E0	—	SRR[0:1]	Instruction TLB miss in MMU
Debug	IVOR 15	0x0F0	DE, IDM	CSRR[0:1]	ROM Debugger when HID0[DAPUEN]=0
				DSRR[0:1]	ROM Debugger when HID0[DAPUEN]=1

¹ IVOR 9 (Offset 0x090) is not supported.

² CE, ME, EE, DE are in MSR. DIE, FIE, WIE are in TCR. “src” is individual enable for each INTC source. Debug interrupt IVOR15 also requires EDM = 0 (EDM and IDM are in DBCR0).

³ Software vector mode interrupts use IVOR 4. Hardware vectored mode interrupts supply an interrupt vector based upon the vector number given in [Table 8-2.](#), “Interrupt Summary for External Input to e200z1 or e200z0.

⁴ Only on e200z1; not implemented on e200z0.

8.2.2 External Input: Software Vector Mode

The IVPR acts as a base register for all types of exceptions. An IVOR, unique to each type of exception, determines the offset from the IVPR. The IVPR and IVOR are added to calculate the interrupt exception handler address. In software vector mode, IVOR4 is used for the external input, that is, the interrupt request to the e200z1 or e200z0 from the INTC. [Figure 8-2](#) shows the software vector mode interrupt exception handler address calculation.

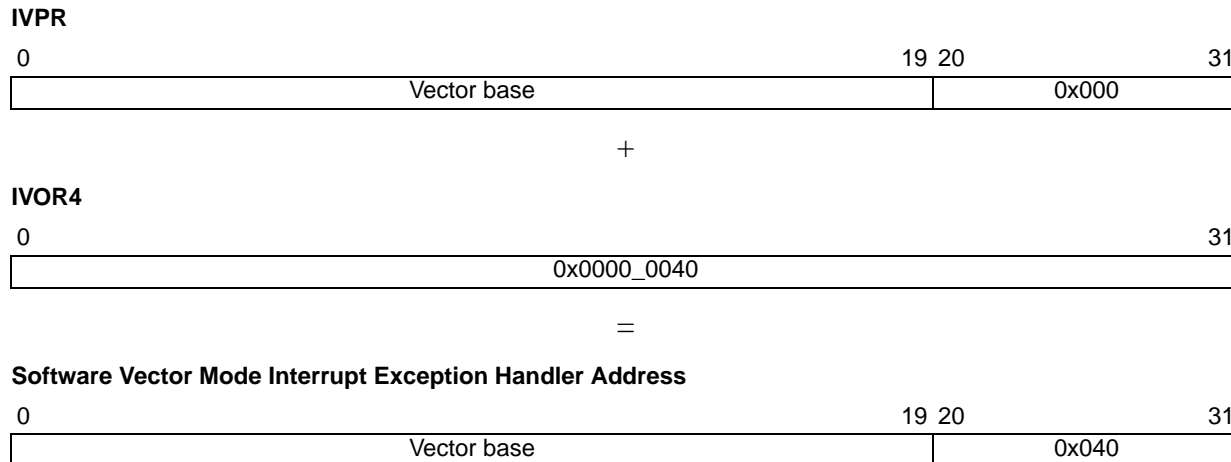


Figure 8-2. Software Vector Mode Interrupt Exception Handler Address Calculation

8.2.3 External Input: Hardware Vector Mode

In hardware vector mode, no IVOR is used, including IVOR4, which has no effect. The interrupt exception handler for each vector is offset from the IVPR. The vectors for each source are shown in [Table 8-2](#). The

amount of the offset is the vector number \times 4 bytes. Figure 8-3 shows the hardware vector mode interrupt exception handler address calculation.

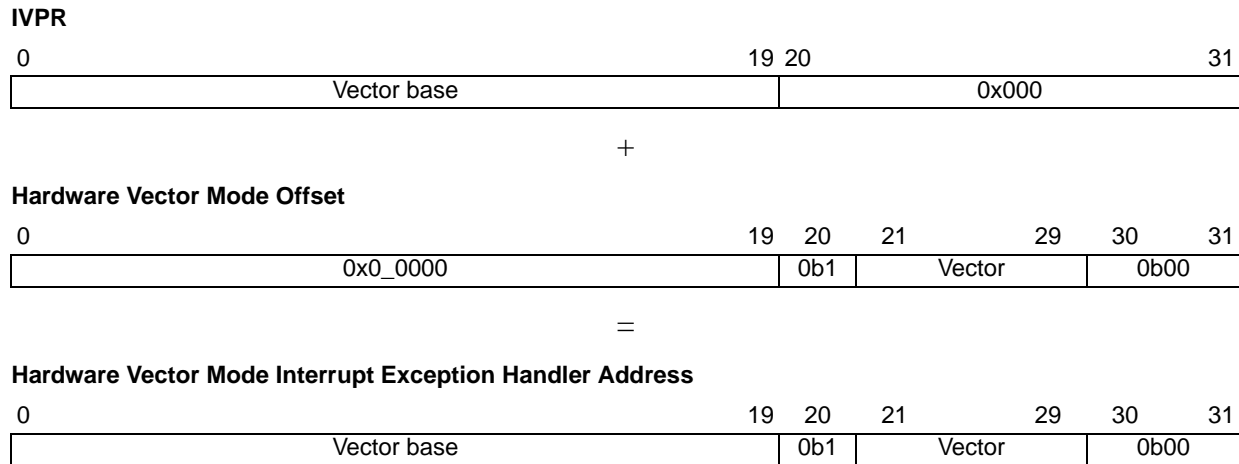


Figure 8-3. Hardware Vector Mode Interrupt Exception Handler Address Calculation

Any vector that is not reserved can have an interrupt exception handler that can be executed.

NOTE

In hardware vector mode, the IVORs for all other exceptions besides the external input must be configured to not use the interrupt exception handler addresses.

8.2.4 Critical Input

The external interrupt pins NMI0 (the third alternate function of pin PD[10]) and NMI1 (the third alternate function of pin PD[11]) can be used as critical interrupt sources to the e200z1 and e200z0 cores respectively. See Section 8.4.3, “Non Maskable Interrupt (NMI),” for more details on the usage and configuration of the critical interrupt input to the core as a pseudo non maskable interrupt (NMI).

For critical interrupts, IVOR0 is used for the exception handler address calculation as shown in Figure 8-4.

For critical interrupts, only software vector mode is supported. The INTC does not support a hardware vector mode for critical interrupts.

For a critical interrupt, the port pin must be configured for alternate function NMI_n mode in its SIU_PCR, and either rising-edge or falling-edge detect must be enabled in the SIU_IREER or SIU_IFEER. (Note that these bits are “write once” bits.) When the NMI is taken, the flag must be cleared in the SIU_EISR.

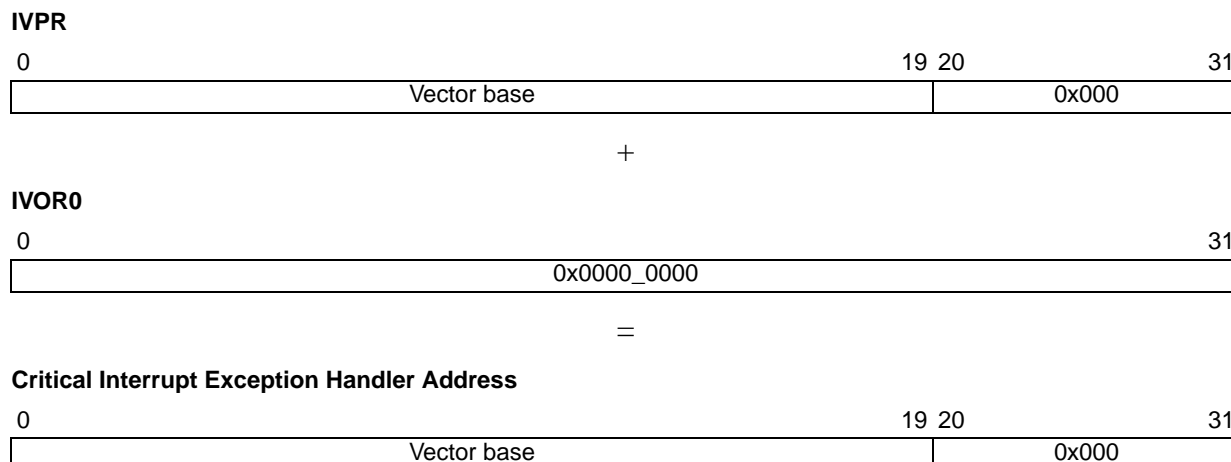


Figure 8-4. Critical Interrupt Exception Handler Address Calculation

8.3 Interrupt Sources

8.3.1 Interrupt Source Summary Table

The assignments between the interrupt requests from the blocks to the vectors for the external input to either the e200z1 or e200z0 are shown in Table 8-2. The source column is written in C language syntax. The syntax is `block_instance.register[bit]`. The syntax ‘||’ represents the ORing of individual interrupt requests from the block.

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 1 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
INTC_SSCIR0_3_CLR0	0x0800	0		INTC.INTC_SSCIR0_3[CLR0]	INTC software settable clear flag 0
INTC_SSCIR0_3_CLR1	0x0804	1		INTC.INTC_SSCIR0_3[CLR1]	INTC software settable clear flag 1
INTC_SSCIR0_3_CLR2	0x0808	2		INTC.INTC_SSCIR0_3[CLR2]	INTC software settable clear flag 2
INTC_SSCIR0_3_CLR3	0x080C	3		INTC.INTC_SSCIR0_3[CLR3]	INTC software settable clear flag 3
INTC_SSCIR4_7_CLR4	0x0810	4		INTC.INTC_SSCIR4_7[CLR4]	INTC software settable clear flag 4
INTC_SSCIR4_7_CLR5	0x0814	5		INTC.INTC_SSCIR4_7[CLR5]	INTC software settable clear flag 5
INTC_SSCIR4_7_CLR6	0x0818	6		INTC.INTC_SSCIR4_7[CLR6]	INTC software settable clear flag 6
INTC_SSCIR4_7_CLR7	0x081C	7		INTC.INTC_SSCIR4_7[CLR7]	INTC software settable clear flag 7
MCM_MSWTIR_SWTIC	0x0820	8		MCM.MSWTIR[SWTIC]	MCM software watchdog interrupt flag
MCM_ESR_COMB	0x0824	9		MCM.ESR[PRNCE] MCM.ESR[PFNCE]	MCM combined interrupt request of the platform RAM non-correctable error and platform flash non-correctable error interrupt requests

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 2 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
eDMA_ERRL_ERR31_0	0x0828	10		eDMA.DMAERRL[ERR31:ERR0]	eDMA channel error flags 31 - 0
eDMA_INTL_INT0	0x082C	11		eDMA.DMAINTL[INT0]	eDMA channel interrupt 0
eDMA_INTL_INT1	0x0830	12		eDMA.DMAINTL[INT1]	eDMA channel interrupt 1
eDMA_INTL_INT2	0x0834	13		eDMA.DMAINTL[INT2]	eDMA channel interrupt 2
eDMA_INTL_INT3	0x0838	14		eDMA.DMAINTL[INT3]	eDMA channel interrupt 3
eDMA_INTL_INT4	0x083C	15		eDMA.DMAINTL[INT4]	eDMA channel interrupt 4
eDMA_INTL_INT5	0x0840	16		eDMA.DMAINTL[INT5]	eDMA channel interrupt 5
eDMA_INTL_INT6	0x0844	17		eDMA.DMAINTL[INT6]	eDMA channel interrupt 6
eDMA_INTL_INT7	0x0848	18		eDMA.DMAINTL[INT7]	eDMA channel interrupt 7
eDMA_INTL_INT8	0x084C	19		eDMA.DMAINTL[INT8]	eDMA channel interrupt 8
eDMA_INTL_INT9	0x0850	20		eDMA.DMAINTL[INT9]	eDMA channel interrupt 9
eDMA_INTL_INT10	0x0854	21		eDMA.DMAINTL[INT10]	eDMA channel interrupt 10
eDMA_INTL_INT11	0x0858	22		eDMA.DMAINTL[INT11]	eDMA channel interrupt 11
eDMA_INTL_INT12	0x085C	23		eDMA.DMAINTL[INT12]	eDMA channel interrupt 12
eDMA_INTL_INT13	0x0860	24		eDMA.DMAINTL[INT13]	eDMA channel interrupt 13
eDMA_INTL_INT14	0x0864	25		eDMA.DMAINTL[INT14]	eDMA channel interrupt 14
eDMA_INTL_INT15	0x0868	26		eDMA.DMAINTL[INT15]	eDMA channel interrupt 15
Reserved	0x086C	27		Reserved	Reserved
Reserved	0x0870	28		Reserved	Reserved
Reserved	0x0874	29		Reserved	Reserved
Reserved	0x0878	30		Reserved	Reserved
Reserved	0x087C	31		Reserved	Reserved
Reserved	0x0880	32		Reserved	Reserved
Reserved	0x0884	33		Reserved	Reserved
Reserved	0x0888	34		Reserved	Reserved
Reserved	0x088C	35		Reserved	Reserved
Reserved	0x0890	36		Reserved	Reserved
Reserved	0x0894	37		Reserved	Reserved
Reserved	0x0898	38		Reserved	Reserved
Reserved	0x089C	39		Reserved	Reserved
Reserved	0x08A0	40		Reserved	Reserved
Reserved	0x08A4	41		Reserved	Reserved

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 3 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
Reserved	0x08A8	42		Reserved	Reserved
Semaphore Int 0	0x08AC	43		Semaphore Int 0	Z1 requested semaphore has unlocked
Semaphore Int 1	0x08B0	44		Semaphore Int 1	Z0 requested semaphore has unlocked
Reserved	0x08B4	45		Reserved	Reserved
CRP Interrupt	0x08B8	46		CRP Interrupt	Combined pin wakeup, API, RTC match, and RTC rollover
LVI Interrupt	0x08BC	47		LVI Interrupt	Low-voltage inhibit interrupt
IIC_A_IBSR_IBIF	0x08C0	48		IIC_A.IBSR[IBIF]	IIC_A arbitration lost, or byte transfer complete, or addressed as slave, or no acknowledge from slave.
Reserved	0x08C4	49		Reserved	Reserved
PLL_SYNSR_LOCF	0x08C8	50		PLL.SYNSR[LOCF]	FNPLL loss-of-clock flag
PLL_SYNSR_LOLF	0x08CC	51		PLL.SYNSR[LOLF]	FNPLL loss-of-lock flag
SIU_OSR_OVER	0x08D0	52		SIU.SIU_OSR[OVF15:OVF0]	SIU combined overrun interrupt request of the external interrupt overrun flags.
SIU_EISR_EIF0	0x08D4	53		SIU.SIU_EISR[EIF0]	SIU external interrupt flag 0
SIU_EISR_EIF1	0x08D8	54		SIU.SIU_EISR[EIF1]	SIU external interrupt flag 1
SIU_EISR_EIF2	0x08DC	55		SIU.SIU_EISR[EIF2]	SIU external interrupt flag 2
SIU_EISR_EIF3	0x08E0	56		SIU.SIU_EISR[EIF3]	SIU external interrupt flag 3
SIU_EISR_EIF15_4	0x08E4	57		SIU.SIU_EISR[EIF15:EIF4]	SIU external interrupt flags 15–4
eMIOS200_FLAG_F0	0x08E8	58		eMIOS200.eMIOS200FLAG[F0]	eMIOS200 channel 0 flag
eMIOS200_FLAG_F1	0x08EC	59		eMIOS200.eMIOS200FLAG[F1]	eMIOS200 channel 1 flag
eMIOS200_FLAG_F2	0x08F0	60		eMIOS200.eMIOS200FLAG[F2]	eMIOS200 channel 2 flag
eMIOS200_FLAG_F3	0x08F4	61		eMIOS200.eMIOS200FLAG[F3]	eMIOS200 channel 3 flag
eMIOS200_FLAG_F4	0x08F8	62		eMIOS200.eMIOS200FLAG[F4]	eMIOS200 channel 4 flag
eMIOS200_FLAG_F5	0x08FC	63		eMIOS200.eMIOS200FLAG[F5]	eMIOS200 channel 5 flag
eMIOS200_FLAG_F6	0x0900	64		eMIOS200.eMIOS200FLAG[F6]	eMIOS200 channel 6 flag
eMIOS200_FLAG_F7	0x0904	65		eMIOS200.eMIOS200FLAG[F7]	eMIOS200 channel 7 flag
eMIOS200_FLAG_F8	0x0908	66		eMIOS200.eMIOS200FLAG[F8]	eMIOS200 channel 8 flag
eMIOS200_FLAG_F9	0x090C	67		eMIOS200.eMIOS200FLAG[F9]	eMIOS200 channel 9 flag
eMIOS200_FLAG_F10	0x0910	68		eMIOS200.eMIOS200FLAG[F10]	eMIOS200 channel 10 flag
eMIOS200_FLAG_F11	0x0914	69		eMIOS200.eMIOS200FLAG[F11]	eMIOS200 channel 11 flag
eMIOS200_FLAG_F12	0x0918	70		eMIOS200.eMIOS200FLAG[F12]	eMIOS200 channel 12 flag

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 4 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
eMIOS200_FLAG_F13	0x091C	71		eMIOS200.eMIOS200FLAG[F13]	eMIOS200 channel 13 flag
eMIOS200_FLAG_F14	0x0920	72		eMIOS200.eMIOS200FLAG[F14]	eMIOS200 channel 14 flag
eMIOS200_FLAG_F15	0x0924	73		eMIOS200.eMIOS200FLAG[F15]	eMIOS200 channel 15 flag
eMIOS200_FLAG_F16	0x0928	74		eMIOS200.eMIOS200FLAG[F16]	eMIOS200 channel 16 flag
eMIOS200_FLAG_F17	0x092C	75		eMIOS200.eMIOS200FLAG[F17]	eMIOS200 channel 17 flag
eMIOS200_FLAG_F18	0x0930	76		eMIOS200.eMIOS200FLAG[F18]	eMIOS200 channel 18 flag
eMIOS200_FLAG_F19	0x0934	77		eMIOS200.eMIOS200FLAG[F19]	eMIOS200 channel 19 flag
eMIOS200_FLAG_F20	0x0938	78		eMIOS200.eMIOS200FLAG[F20]	eMIOS200 channel 20 flag
eMIOS200_FLAG_F21	0x093C	79		eMIOS200.eMIOS200FLAG[F21]	eMIOS200 channel 21 flag
eMIOS200_FLAG_F22	0x0940	80		eMIOS200.eMIOS200FLAG[F22]	eMIOS200 channel 22 flag
eMIOS200_FLAG_F23	0x0944	81		eMIOS200.eMIOS200FLAG[F23]	eMIOS200 channel 23 flag
eQADC_FISR_OVER	0x0948	82		eQADC.eQADC_FISRx[TORF] eQADC.eQADC_FISRx[RFOF] eQADC.eQADC_FISRx[CFUF]	eQADC combined overrun interrupt request of the trigger overrun, receive FIFO overflow, and command FIFO underflow interrupt requests from all of the FIFOs
eQADC_FISR0_NCF0	0x094C	83		eQADC.eQADC_FISR0[NCF0]	eQADC command FIFO 0 non-coherency flag
eQADC_FISR0_PFO	0x0950	84		eQADC.eQADC_FISR0[PFO]	eQADC command FIFO 0 pause flag
eQADC_FISR0_EOQF0	0x0954	85		eQADC.eQADC_FISR0[EOQF0]	eQADC command FIFO 0 command queue end-of-queue flag
eQADC_FISR0_CFFF0	0x0958	86		eQADC.eQADC_FISR0[CFFF0]	eQADC command FIFO 0 fill flag
eQADC_FISR0_RFDF0	0x095C	87		eQADC.eQADC_FISR0[RFDF0]	eQADC receive FIFO 0 drain flag
eQADC_FISR1_NCF1	0x0960	88		eQADC.eQADC_FISR1[NCF1]	eQADC command FIFO 1 non-coherency flag
eQADC_FISR1_PFO	0x0964	89		eQADC.eQADC_FISR1[PFO]	eQADC command FIFO 1 pause flag
eQADC_FISR1_EOQF1	0x0968	90		eQADC.eQADC_FISR1[EOQF1]	eQADC command FIFO 1 command queue end-of-queue flag
eQADC_FISR1_CFFF1	0x096C	91		eQADC.eQADC_FISR1[CFFF1]	eQADC command FIFO 1 fill flag
eQADC_FISR1_RFDF1	0x0970	92		eQADC.eQADC_FISR1[RFDF1]	eQADC receive FIFO 1 drain flag
eQADC_FISR2_NCF2	0x0974	93		eQADC.eQADC_FISR2[NCF2]	eQADC command FIFO 2 non-coherency flag
eQADC_FISR2_PFO	0x0978	94		eQADC.eQADC_FISR2[PFO]	eQADC command FIFO 2 pause flag
eQADC_FISR2_EOQF2	0x097C	95		eQADC.eQADC_FISR2[EOQF2]	eQADC command FIFO 2 command queue end-of-queue flag

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 5 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
eQADC_FISR2_CFFF2	0x0980	96		eQADC.eQADC_FISR2[CFFF2]	eQADC command FIFO 2 fill flag
eQADC_FISR2_RFDF2	0x0984	97		eQADC.eQADC_FISR2[RFDF2]	eQADC receive FIFO 2 drain flag
eQADC_FISR3_NCF3	0x0988	98		eQADC.eQADC_FISR3[NCF3]	eQADC command FIFO 3 non-coherency flag
eQADC_FISR3_PF3	0x098C	99		eQADC.eQADC_FISR3[PF3]	eQADC command FIFO 3 pause flag
eQADC_FISR3_EOQF3	0x0990	100		eQADC.eQADC_FISR3[EOQF3]	eQADC command FIFO 3 command queue end-of-queue flag
eQADC_FISR3_CFFF3	0x0994	101		eQADC.eQADC_FISR3[CFFF3]	eQADC command FIFO 3 fill flag
eQADC_FISR3_RFDF3	0x0998	102		eQADC.eQADC_FISR3[RFDF3]	eQADC receive FIFO 3 drain flag
eQADC_FISR4_NCF4	0x099C	103		eQADC.eQADC_FISR4[NCF4]	eQADC command FIFO 4 non-coherency flag
eQADC_FISR4_PF4	0x09A0	104		eQADC.eQADC_FISR4[PF4]	eQADC command FIFO 4 pause flag
eQADC_FISR4_EOQF4	0x09A4	105		eQADC.eQADC_FISR4[EOQF4]	eQADC command FIFO 4 command queue end-of-queue flag
eQADC_FISR4_CFFF4	0x09A8	106		eQADC.eQADC_FISR4[CFFF4]	eQADC command FIFO 4 fill flag
eQADC_FISR4_RFDF4	0x09AC	107		eQADC.eQADC_FISR4[RFDF4]	eQADC receive FIFO 4 drain flag
eQADC_FISR5_NCF5	0x09B0	108		eQADC.eQADC_FISR5[NCF5]	eQADC command FIFO 5 non-coherency flag
eQADC_FISR5_PF5	0x09B4	109		eQADC.eQADC_FISR5[PF5]	eQADC command FIFO 5 pause flag
eQADC_FISR5_EOQF5	0x09B8	110		eQADC.eQADC_FISR5[EOQF5]	eQADC command FIFO 5 command queue end-of-queue flag
eQADC_FISR5_CFFF5	0x09BC	111		eQADC.eQADC_FISR5[CFFF5]	eQADC Command FIFO 5 fill flag
eQADC_FISR5_RFDF5	0x09C0	112		eQADC.eQADC_FISR5[RFDF5]	eQADC receive FIFO 5 drain flag
SCI_A_COMB	0x09C4	113		SCI_A.SCISR1[TDRE] SCI_A.SCISR1[TC] SCI_A.SCISR1[RDRF] SCI_A.SCISR1[IDLE] SCI_A.SCISR1[OR] SCI_A.SCISR1[NF] SCI_A.SCISR1[FE] SCI_A.SCISR1[PF] SCI_A.SCISR2[BERR] SCI_A.LINSTAT1[RXRDY] SCI_A.LINSTAT1[TXRDY] SCI_A.LINSTAT1[LWAKE] SCI_A.LINSTAT1[STO] SCI_A.LINSTAT1[PBERR] SCI_A.LINSTAT1[CERR] SCI_A.LINSTAT1[CKERR] SCI_A.LINSTAT1[FRC] SCI_A.LINSTAT2[OVFL]	SCI_A combined interrupt request of the SCI status register 1 transmit data register empty, transmit complete, receive data register full, idle line, overrun, noise, frame error, and parity error interrupt requests, SCI status register 2 bit error interrupt request, LIN status register 1 receive data ready, transmit data ready, received LIN wakeup signal, slave timeout, physical bus error, CRC error, checksum error, frame complete interrupts requests, and LIN status register 2 receive register overflow interrupt request

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 6 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
SCI_B_COMB	0x09C8	114		SCI_B.SCIISR1[TDRE] SCI_B.SCIISR1[TC] SCI_B.SCIISR1[RDRF] SCI_B.SCIISR1[IDLE] SCI_B.SCIISR1[OR] SCI_B.SCIISR1[NF] SCI_B.SCIISR1[FE] SCI_B.SCIISR1[PF] SCI_B.SCIISR2[BERR] SCI_B.LINSTAT1[RXRDY] SCI_B.LINSTAT1[TXRDY] SCI_B.LINSTAT1[LWAKE] SCI_B.LINSTAT1[STO] SCI_B.LINSTAT1[PBERR] SCI_B.LINSTAT1[CERR] SCI_B.LINSTAT1[CKERR] SCI_B.LINSTAT1[FRC] SCI_B.LINSTAT2[OVFL]	SCI_B combined interrupt request of the SCI status register 1 transmit data register empty, transmit complete, receive data register full, idle line, overrun, noise, frame error, and parity error interrupt requests, SCI status register 2 bit error interrupt request, LIN status register 1 receive data ready, transmit data ready, received LIN wakeup signal, slave timeout, physical bus error, CRC error, checksum error, frame complete interrupts requests, and LIN status register 2 receive register overflow interrupt request
SCI_C_COMB	0x09CC	115		SCI_C.SCIISR1[TDRE] SCI_C.SCIISR1[TC] SCI_C.SCIISR1[RDRF] SCI_C.SCIISR1[IDLE] SCI_C.SCIISR1[OR] SCI_C.SCIISR1[NF] SCI_C.SCIISR1[FE] SCI_C.SCIISR1[PF] SCI_C.SCIISR2[BERR] SCI_C.LINSTAT1[RXRDY] SCI_C.LINSTAT1[TXRDY] SCI_C.LINSTAT1[LWAKE] SCI_C.LINSTAT1[STO] SCI_C.LINSTAT1[PBERR] SCI_C.LINSTAT1[CERR] SCI_C.LINSTAT1[CKERR] SCI_C.LINSTAT1[FRC] SCI_C.LINSTAT2[OVFL]	SCI_C combined interrupt request of the SCI status register 1 transmit data register empty, transmit complete, receive data register full, idle line, overrun, noise, frame error, and parity error interrupt requests, SCI status register 2 bit error interrupt request, LIN status register 1 receive data ready, transmit data ready, received LIN wakeup signal, slave timeout, physical bus error, CRC error, checksum error, frame complete interrupts requests, and LIN status register 2 receive register overflow interrupt request
SCI_D_COMB	0x09D0	116		SCI_D.SCIISR1[TDRE] SCI_D.SCIISR1[TC] SCI_D.SCIISR1[RDRF] SCI_D.SCIISR1[IDLE] SCI_D.SCIISR1[OR] SCI_D.SCIISR1[NF] SCI_D.SCIISR1[FE] SCI_D.SCIISR1[PF] SCI_D.SCIISR2[BERR] SCI_D.LINSTAT1[RXRDY] SCI_D.LINSTAT1[TXRDY] SCI_D.LINSTAT1[LWAKE] SCI_D.LINSTAT1[STO] SCI_D.LINSTAT1[PBERR] SCI_D.LINSTAT1[CERR] SCI_D.LINSTAT1[CKERR] SCI_D.LINSTAT1[FRC] SCI_D.LINSTAT2[OVFL]	SCI_D combined interrupt request of the SCI status register 1 transmit data register empty, transmit complete, receive data register full, idle line, overrun, noise, frame error, and parity error interrupt requests, SCI status register 2 bit error interrupt request, LIN status register 1 receive data ready, transmit data ready, received LIN wakeup signal, slave timeout, physical bus error, CRC error, checksum error, frame complete interrupts requests, and LIN status register 2 receive register overflow interrupt request

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 7 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
DSPI_A_ISR_OVER	0x09D4	117		DSPI_A.DSPI_ISR[TFUF] DSPI_A.DSPI_ISR[RFOF]	DSPI_A combined overrun interrupt request of the transmit FIFO underflow and receive FIFO overflow interrupt requests
DSPI_A_ISR_EOQF	0x09D8	118		DSPI_A.DSPI_ISR[EOQF]	DSPI_A transmit FIFO end-of-queue flag
DSPI_A_ISR_TFFF	0x09DC	119		DSPI_A.DSPI_ISR[TFFF]	DSPI_A transmit FIFO fill flag
DSPI_A_ISR_TCF	0x09E0	120		DSPI_A.DSPI_ISR[TCF]	DSPI_A transfer complete flag
DSPI_A_ISR_RFDF	0x09E4	121		DSPI_A.DSPI_ISR[RFDF]	DSPI_A receive FIFO drain flag
DSPI_B_ISR_OVER	0x09E8	122		DSPI_B.DSPI_ISR[TFUF] DSPI_B.DSPI_ISR[RFOF]	DSPI_B combined overrun interrupt request of the transmit FIFO underflow and receive FIFO overflow interrupt requests
DSPI_B_ISR_EOQF	0x09EC	123		DSPI_B.DSPI_ISR[EOQF]	DSPI_B transmit FIFO end-of-queue flag
DSPI_B_ISR_TFFF	0x09F0	124		DSPI_B.DSPI_ISR[TFFF]	DSPI_B transmit FIFO fill flag
DSPI_B_ISR_TCF	0x09F4	125		DSPI_B.DSPI_ISR[TCF]	DSPI_B transfer complete flag
DSPI_B_ISR_RFDF	0x09F8	126		DSPI_B.DSPI_ISR[RFDF]	DSPI_B receive FIFO drain flag
FLEXCAN_A_ESR_BOFF_INT	0x09FC	127		FLEXCAN_A.ESR[BOFF_INT] FLEXCAN_A.ESR[TWRN_INT] FLEXCAN_A.ESR[RWRN_INT]	FLEXCAN_A bus off interrupt, FLEXCAN_A transmit warning interrupt, FLEXCAN_A receive warning interrupt
FLEXCAN_A_ESR_ERR_INT	0x0A00	128		FLEXCAN_A.ESR[ERR_INT]	FLEXCAN_A error interrupt
Reserved	0x0A04	129		Reserved	Reserved
FLEXCAN_A_IFLAG1_BUF0I	0x0A08	130		FLEXCAN_A.IFLAG1[BUF0I]	FLEXCAN_A buffer 0 interrupt
FLEXCAN_A_IFLAG1_BUF1I	0x0A0C	131		FLEXCAN_A.IFLAG1[BUF1I]	FLEXCAN_A buffer 1 interrupt
FLEXCAN_A_IFLAG1_BUF2I	0x0A10	132		FLEXCAN_A.IFLAG1[BUF2I]	FLEXCAN_A buffer 2 interrupt
FLEXCAN_A_IFLAG1_BUF3I	0x0A14	133		FLEXCAN_A.IFLAG1[BUF3I]	FLEXCAN_A buffer 3 interrupt
FLEXCAN_A_IFLAG1_BUF4I	0x0A18	134		FLEXCAN_A.IFLAG1[BUF4I]	FLEXCAN_A buffer 4 interrupt
FLEXCAN_A_IFLAG1_BUF5I	0x0A1C	135		FLEXCAN_A.IFLAG1[BUF5I]	FLEXCAN_A buffer 5 interrupt
FLEXCAN_A_IFLAG1_BUF6I	0x0A20	136		FLEXCAN_A.IFLAG1[BUF6I]	FLEXCAN_A buffer 6 interrupt
FLEXCAN_A_IFLAG1_BUF7I	0x0A24	137		FLEXCAN_A.IFLAG1[BUF7I]	FLEXCAN_A buffer 7 interrupt
FLEXCAN_A_IFLAG1_BUF8I	0x0A28	138		FLEXCAN_A.IFLAG1[BUF8I]	FLEXCAN_A buffer 8 interrupt
FLEXCAN_A_IFLAG1_BUF9I	0x0A2C	139		FLEXCAN_A.IFLAG1[BUF9I]	FLEXCAN_A buffer 9 interrupt
FLEXCAN_A_IFLAG1_BUF10I	0x0A30	140		FLEXCAN_A.IFLAG1[BUF10I]	FLEXCAN_A buffer 10 interrupt
FLEXCAN_A_IFLAG1_BUF11I	0x0A34	141		FLEXCAN_A.IFLAG1[BUF11I]	FLEXCAN_A buffer 11 interrupt
FLEXCAN_A_IFLAG1_BUF12I	0x0A38	142		FLEXCAN_A.IFLAG1[BUF12I]	FLEXCAN_A buffer 12 interrupt
FLEXCAN_A_IFLAG1_BUF13I	0x0A3C	143		FLEXCAN_A.IFLAG1[BUF13I]	FLEXCAN_A buffer 13 interrupt

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 8 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
FLEXCAN_A_IFLAG1_BUF14I	0x0A40	144		FLEXCAN_A.IFLAG1[BUF14I]	FLEXCAN_A buffer 14 interrupt
FLEXCAN_A_IFLAG1_BUF15I	0x0A44	145		FLEXCAN_A.IFLAG1[BUF15I]	FLEXCAN_A buffer 15 interrupt
FLEXCAN_A_IFLAG1_BUF31_16I	0x0A48	146		FLEXCAN_A.IFLAG1 [BUF31I:BUF16I]	FLEXCAN_A buffers 31–16 interrupts
FLEXCAN_A_IFLAG2_BUF63_32I	0x0A4C	147		FLEXCAN_A.IFLAG2 [BUF63I:BUF32I]	FLEXCAN_A buffers 63–32 interrupts
PIT_PITFLG_RTIF	0x0A50	148		PIT.PITFLG[RTIF]	Real-time counter interrupt
PIT_PITFLG_PIT1	0x0A54	149		PIT.PITFLG[PIT1]	Programmable interrupt timer 1 interrupt
PIT_PITFLG_PIT2	0x0A58	150		PIT.PITFLG[PIT2]	Programmable interrupt timer 2 interrupt
PIT_PITFLG_PIT3	0x0A5C	151		PIT.PITFLG[PIT3]	Programmable interrupt timer 3 interrupt
PIT_PITFLG_PIT4	0x0A60	152		PIT.PITFLG[PIT4]	Programmable interrupt timer 4 interrupt
PIT_PITFLG_PIT5	0x0A64	153		PIT.PITFLG[PIT5]	Programmable interrupt timer 5 interrupt
PIT_PITFLG_PIT6	0x0A68	154		PIT.PITFLG[PIT6]	Programmable interrupt timer 6 interrupt
PIT_PITFLG_PIT7	0x0A6C	155		PIT.PITFLG[PIT7]	Programmable interrupt timer 7 interrupt
PIT_PITFLG_PIT8	0x0A70	156		PIT.PITFLG[PIT8]	Programmable interrupt Timer 8 interrupt
FLEXCAN_B_ESR_BOFF_INT	0x0A74	157		FLEXCAN_B.ESR[BOFF_INT] FLEXCAN_B.ESR[TWRN_INT] FLEXCAN_B.ESR[RWRN_INT]	FLEXCAN_B bus off interrupt, FLEXCAN_B transmit warning interrupt, FLEXCAN_B receive warning interrupt
FLEXCAN_B_ESR_ERR_INT	0x0A78	158		FLEXCAN_B.ESR[ERR_INT]	FLEXCAN_B error interrupt
Reserved	0x0A7C	159		Reserved	Reserved
FLEXCAN_B_IFLAG1_BUF0I	0x0A80	160		FLEXCAN_B.IFLAG1[BUF0I]	FLEXCAN_B buffer 0 interrupt
FLEXCAN_B_IFLAG1_BUF1I	0x0A84	161		FLEXCAN_B.IFLAG1[BUF1I]	FLEXCAN_B buffer 1 interrupt
FLEXCAN_B_IFLAG1_BUF2I	0x0A88	162		FLEXCAN_B.IFLAG1[BUF2I]	FLEXCAN_B buffer 2 interrupt
FLEXCAN_B_IFLAG1_BUF3I	0x0A8C	163		FLEXCAN_B.IFLAG1[BUF3I]	FLEXCAN_B buffer 3 interrupt
FLEXCAN_B_IFLAG1_BUF4I	0x0A90	164		FLEXCAN_B.IFLAG1[BUF4I]	FLEXCAN_B buffer 4 interrupt
FLEXCAN_B_IFLAG1_BUF5I	0x0A94	165		FLEXCAN_B.IFLAG1[BUF5I]	FLEXCAN_B buffer 5 interrupt
FLEXCAN_B_IFLAG1_BUF6I	0x0A98	166		FLEXCAN_B.IFLAG1[BUF6I]	FLEXCAN_B buffer 6 interrupt
FLEXCAN_B_IFLAG1_BUF7I	0x0A9C	167		FLEXCAN_B.IFLAG1[BUF7I]	FLEXCAN_B buffer 7 interrupt
FLEXCAN_B_IFLAG1_BUF8I	0x0AA0	168		FLEXCAN_B.IFLAG1[BUF8I]	FLEXCAN_B buffer 8 interrupt
FLEXCAN_B_IFLAG1_BUF9I	0x0AA4	169		FLEXCAN_B.IFLAG1[BUF9I]	FLEXCAN_B buffer 9 interrupt
FLEXCAN_B_IFLAG1_BUF10I	0x0AA8	170		FLEXCAN_B.IFLAG1[BUF10I]	FLEXCAN_B buffer 10 interrupt
FLEXCAN_B_IFLAG1_BUF11I	0x0AAC	171		FLEXCAN_B.IFLAG1[BUF11I]	FLEXCAN_B buffer 11 interrupt
FLEXCAN_B_IFLAG1_BUF12I	0x0AB0	172		FLEXCAN_B.IFLAG1[BUF12I]	FLEXCAN_B buffer 12 interrupt

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 9 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
FLEXCAN_B_IFLAG1_BUF13I	0x0AB4	173		FLEXCAN_B.IFLAG1[BUF13I]	FLEXCAN_B buffer 13 interrupt
FLEXCAN_B_IFLAG1_BUF14I	0x0AB8	174		FLEXCAN_B.IFLAG1[BUF14I]	FLEXCAN_B buffer 14 interrupt
FLEXCAN_B_IFLAG1_BUF15I	0x0ABC	175		FLEXCAN_B.IFLAG1[BUF15I]	FLEXCAN_B buffer 15 interrupt
FLEXCAN_B_IFLAG1_BUF31_16I	0x0AC0	176		FLEXCAN_B.IFLAG1 [BUF31I:BUF16I]	FLEXCAN_B buffers 31–16 Interrupts
FLEXCAN_B_IFLAG2_BUF63_32I	0x0AC4	177		FLEXCAN_B.IFLAG2 [BUF63I:BUF32I]	FLEXCAN_B buffers 63– 32 interrupts
FLEXCAN_C_ESR_BOFF_INT	0x0AC8	178		FLEXCAN_C.ESR[BOFF_INT] FLEXCAN_C.ESR[TWRN_INT] FLEXCAN_C.ESR[RWRN_INT]	FLEXCAN_C bus off interrupt, FLEXCAN_C transmit warning interrupt, FLEXCAN_C receive warning interrupt
FLEXCAN_C_ESR_ERR_INT	0x0ACC	179		FLEXCAN_C.ESR[ERR_INT]	FLEXCAN_C error interrupt
Reserved	0x0AD0	180		Reserved	Reserved
FLEXCAN_C_IFLAG1_BUF0I	0x0AD4	181		FLEXCAN_C.IFLAG1[BUF0I]	FLEXCAN_C buffer 0 Interrupt
FLEXCAN_C_IFLAG1_BUF1I	0x0AD8	182		FLEXCAN_C.IFLAG1[BUF1I]	FLEXCAN_C buffer 1 interrupt
FLEXCAN_C_IFLAG1_BUF2I	0x0ADC	183		FLEXCAN_C.IFLAG1[BUF2I]	FLEXCAN_C buffer 2 interrupt
FLEXCAN_C_IFLAG1_BUF3I	0x0AE0	184		FLEXCAN_C.IFLAG1[BUF3I]	FLEXCAN_C buffer 3 interrupt
FLEXCAN_C_IFLAG1_BUF4I	0x0AE4	185		FLEXCAN_C.IFLAG1[BUF4I]	FLEXCAN_C buffer 4 interrupt
FLEXCAN_C_IFLAG1_BUF5I	0x0AE8	186		FLEXCAN_C.IFLAG1[BUF5I]	FLEXCAN_C buffer 5 interrupt
FLEXCAN_C_IFLAG1_BUF6I	0x0AEC	187		FLEXCAN_C.IFLAG1[BUF6I]	FLEXCAN_C buffer 6 interrupt
FLEXCAN_C_IFLAG1_BUF7I	0x0AF0	188		FLEXCAN_C.IFLAG1[BUF7I]	FLEXCAN_C buffer 7 interrupt
FLEXCAN_C_IFLAG1_BUF8I	0x0AF4	189		FLEXCAN_C.IFLAG1[BUF8I]	FLEXCAN_C buffer 8 interrupt
FLEXCAN_C_IFLAG1_BUF9I	0x0AF8	190		FLEXCAN_C.IFLAG1[BUF9I]	FLEXCAN_C buffer 9 interrupt
FLEXCAN_C_IFLAG1_BUF10I	0x0AFC	191		FLEXCAN_C.IFLAG1[BUF10I]	FLEXCAN_C buffer 10 interrupt
FLEXCAN_C_IFLAG1_BUF11I	0x0B00	192		FLEXCAN_C.IFLAG1[BUF11I]	FLEXCAN_C buffer 11 interrupt
FLEXCAN_C_IFLAG1_BUF12I	0x0B04	193		FLEXCAN_C.IFLAG1[BUF12I]	FLEXCAN_C buffer 12 interrupt
FLEXCAN_C_IFLAG1_BUF13I	0x0B08	194		FLEXCAN_C.IFLAG1[BUF13I]	FLEXCAN_C buffer 13 interrupt
FLEXCAN_C_IFLAG1_BUF14I	0x0B0C	195		FLEXCAN_C.IFLAG1[BUF14I]	FLEXCAN_C buffer 14 interrupt
FLEXCAN_C_IFLAG1_BUF15I	0x0B10	196		FLEXCAN_C.IFLAG1[BUF15I]	FLEXCAN_C buffer 15 interrupt
FLEXCAN_C_IFLAG1_BUF31_16I	0x0B14	197		FLEXCAN_C.IFLAG1 [BUF31I:BUF16I]	FLEXCAN_C buffers 31–16 interrupts
FLEXCAN_C_IFLAG2_BUF63_32I	0x0B18	198		FLEXCAN_C.IFLAG2 [BUF63I:BUF32I]	FLEXCAN_C buffers 63–32 interrupts
FLEXCAN_D_ESR_BOFF_INT	0x0B1C	199		FLEXCAN_D.ESR[BOFF_INT] FLEXCAN_D.ESR[TWRN_INT] FLEXCAN_D.ESR[RWRN_INT]	FLEXCAN_D bus off interrupt, FLEXCAN_D transmit warning interrupt, FLEXCAN_D receive warning interrupt

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 10 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
FLEXCAN_D_ESR_ERR_INT	0x0B20	200		FLEXCAN_D.ESR[ERR_INT]	FLEXCAN_D error interrupt
Reserved	0x0B24	201		Reserved	Reserved
FLEXCAN_D_IFLAG1_BUF0I	0x0B28	202		FLEXCAN_D.IFLAG1[BUF0I]	FLEXCAN_D buffer 0 interrupt
FLEXCAN_D_IFLAG1_BUF1I	0x0B2C	203		FLEXCAN_D.IFLAG1[BUF1I]	FLEXCAN_D buffer 1 interrupt
FLEXCAN_D_IFLAG1_BUF2I	0x0B30	204		FLEXCAN_D.IFLAG1[BUF2I]	FLEXCAN_D buffer 2 interrupt
FLEXCAN_D_IFLAG1_BUF3I	0x0B34	205		FLEXCAN_D.IFLAG1[BUF3I]	FLEXCAN_D buffer 3 interrupt
FLEXCAN_D_IFLAG1_BUF4I	0x0B38	206		FLEXCAN_D.IFLAG1[BUF4I]	FLEXCAN_D buffer 4 interrupt
FLEXCAN_D_IFLAG1_BUF5I	0x0B3C	207		FLEXCAN_D.IFLAG1[BUF5I]	FLEXCAN_D buffer 5 interrupt
FLEXCAN_D_IFLAG1_BUF6I	0x0B40	208		FLEXCAN_D.IFLAG1[BUF6I]	FLEXCAN_D buffer 6 interrupt
FLEXCAN_D_IFLAG1_BUF7I	0x0B44	209		FLEXCAN_D.IFLAG1[BUF7I]	FLEXCAN_D buffer 7 interrupt
FLEXCAN_D_IFLAG1_BUF8I	0x0B48	210		FLEXCAN_D.IFLAG1[BUF8I]	FLEXCAN_D buffer 8 interrupt
FLEXCAN_D_IFLAG1_BUF9I	0x0B4C	211		FLEXCAN_D.IFLAG1[BUF9I]	FLEXCAN_D buffer 9 interrupt
FLEXCAN_D_IFLAG1_BUF10I	0x0B50	212		FLEXCAN_D.IFLAG1[BUF10I]	FLEXCAN_D buffer 10 interrupt
FLEXCAN_D_IFLAG1_BUF11I	0x0B54	213		FLEXCAN_D.IFLAG1[BUF11I]	FLEXCAN_D buffer 11 interrupt
FLEXCAN_D_IFLAG1_BUF12I	0x0B58	214		FLEXCAN_D.IFLAG1[BUF12I]	FLEXCAN_D buffer 12 interrupt
FLEXCAN_D_IFLAG1_BUF13I	0x0B5C	215		FLEXCAN_D.IFLAG1[BUF13I]	FLEXCAN_D buffer 13 interrupt
FLEXCAN_D_IFLAG1_BUF14I	0x0B60	216		FLEXCAN_D.IFLAG1[BUF14I]	FLEXCAN_D buffer 14 interrupt
FLEXCAN_D_IFLAG1_BUF15I	0x0B64	217		FLEXCAN_D.IFLAG1[BUF15I]	FLEXCAN_D buffer 15 interrupt
FLEXCAN_D_IFLAG1_BUF31_16I	0x0B68	218		FLEXCAN_D.IFLAG1 [BUF31I:BUF16I]	FLEXCAN_D buffers 31–16 interrupts
FLEXCAN_D_IFLAG2_BUF63_32I	0x0B6C	219		FLEXCAN_D.IFLAG2 [BUF63I:BUF32I]	FLEXCAN_D buffers 63–32 interrupts
FLEXCAN_E_ESR_BOFF_INT	0x0B70	220		FLEXCAN_E.ESR[BOFF_INT] FLEXCAN_E.ESR[TWRN_INT] FLEXCAN_E.ESR[RWRN_INT]	FLEXCAN_E bus off interrupt, FLEXCAN_E transmit warning interrupt, FLEXCAN_E receive warning interrupt
FLEXCAN_E_ESR_ERR_INT	0x0B74	221		FLEXCAN_E.ESR[ERR_INT]	FLEXCAN_E error interrupt
Reserved	0x0B78	222		Reserved	Reserved
FLEXCAN_E_IFLAG1_BUF0I	0x0B7C	223		FLEXCAN_E.IFLAG1[BUF0I]	FLEXCAN_E buffer 0 interrupt
FLEXCAN_E_IFLAG1_BUF1I	0x0B80	224		FLEXCAN_E.IFLAG1[BUF1I]	FLEXCAN_E buffer 1 interrupt
FLEXCAN_E_IFLAG1_BUF2I	0x0B84	225		FLEXCAN_E.IFLAG1[BUF2I]	FLEXCAN_E buffer 2 interrupt
FLEXCAN_E_IFLAG1_BUF3I	0x0B88	226		FLEXCAN_E.IFLAG1[BUF3I]	FLEXCAN_E buffer 3 interrupt
FLEXCAN_E_IFLAG1_BUF4I	0x0B8C	227		FLEXCAN_E.IFLAG1[BUF4I]	FLEXCAN_E buffer 4 interrupt
FLEXCAN_E_IFLAG1_BUF5I	0x0B90	228		FLEXCAN_E.IFLAG1[BUF5I]	FLEXCAN_E buffer 5 interrupt

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 11 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
FLEXCAN_E_IFLAG1_BUF6I	0x0B94	229		FLEXCAN_E.IFLAG1[BUF6I]	FLEXCAN_E buffer 6 interrupt
FLEXCAN_E_IFLAG1_BUF7I	0x0B98	230		FLEXCAN_E.IFLAG1[BUF7I]	FLEXCAN_E buffer 7 interrupt
FLEXCAN_E_IFLAG1_BUF8I	0x0B9C	231		FLEXCAN_EC.IFLAG1[BUF8I]	FLEXCAN_E buffer 8 interrupt
FLEXCAN_E_IFLAG1_BUF9I	0x0BA0	232		FLEXCAN_E.IFLAG1[BUF9I]	FLEXCAN_E buffer 9 interrupt
FLEXCAN_E_IFLAG1_BUF10I	0x0BA4	233		FLEXCAN_E.IFLAG1[BUF10I]	FLEXCAN_E buffer 10 interrupt
FLEXCAN_E_IFLAG1_BUF11I	0x0BA8	234		FLEXCAN_E.IFLAG1[BUF11I]	FLEXCAN_E buffer 11 interrupt
FLEXCAN_E_IFLAG1_BUF12I	0x0BAC	235		FLEXCAN_E.IFLAG1[BUF12I]	FLEXCAN_E buffer 12 interrupt
FLEXCAN_E_IFLAG1_BUF13I	0x0BB0	236		FLEXCAN_E.IFLAG1[BUF13I]	FLEXCAN_E buffer 13 interrupt
FLEXCAN_E_IFLAG1_BUF14I	0x0BB4	237		FLEXCAN_E.IFLAG1[BUF14I]	FLEXCAN_E buffer 14 interrupt
FLEXCAN_E_IFLAG1_BUF15I	0x0BB8	238		FLEXCAN_E.IFLAG1[BUF15I]	FLEXCAN_E buffer 15 interrupt
FLEXCAN_E_IFLAG1_BUF31_16I	0x0BBC	239		FLEXCAN_E.IFLAG1 [BUF31I:BUF16I]	FLEXCAN_E buffers 31–16 interrupts
FLEXCAN_E_IFLAG2_BUF63_32I	0x0BC0	240		FLEXCAN_E.IFLAG2 [BUF63I:BUF32I]	FLEXCAN_E buffers 63–32 interrupts
FLEXCAN_F_ESR_BOFF_INT	0x0BC4	241		FLEXCAN_F.ESR[BOFF_INT] FLEXCAN_F.ESR[TWRN_INT] FLEXCAN_F.ESR[RWRN_INT]	FLEXCAN_F bus off interrupt, FLEXCAN_F transmit warning interrupt, FLEXCAN_F receive warning interrupt
FLEXCAN_F_ESR_ERR_INT	0x0BC8	242		FLEXCAN_F.ESR[ERR_INT]	FLEXCAN_F error interrupt
Reserved	0x0BCC	243		Reserved	Reserved
FLEXCAN_F_IFLAG1_BUF0I	0x0BD0	244		FLEXCAN_F.IFLAG1[BUF0I]	FLEXCAN_F buffer 0 interrupt
FLEXCAN_F_IFLAG1_BUF1I	0x0BD4	245		FLEXCAN_F.IFLAG1[BUF1I]	FLEXCAN_F buffer 1 interrupt
FLEXCAN_F_IFLAG1_BUF2I	0x0BD8	246		FLEXCAN_F.IFLAG1[BUF2I]	FLEXCAN_F buffer 2 interrupt
FLEXCAN_F_IFLAG1_BUF3I	0x0BDC	247		FLEXCAN_F.IFLAG1[BUF3I]	FLEXCAN_F buffer 3 interrupt
FLEXCAN_F_IFLAG1_BUF4I	0x0BE0	248		FLEXCAN_F.IFLAG1[BUF4I]	FLEXCAN_F buffer 4 interrupt
FLEXCAN_F_IFLAG1_BUF5I	0x0BE4	249		FLEXCAN_F.IFLAG1[BUF5I]	FLEXCAN_F buffer 5 interrupt
FLEXCAN_F_IFLAG1_BUF6I	0x0BE8	250		FLEXCAN_F.IFLAG1[BUF6I]	FLEXCAN_F buffer 6 interrupt
FLEXCAN_F_IFLAG1_BUF7I	0x0BEC	251		FLEXCAN_F.IFLAG1[BUF7I]	FLEXCAN_F buffer 7 interrupt
FLEXCAN_F_IFLAG1_BUF8I	0x0BF0	252		FLEXCAN_F.IFLAG1[BUF8I]	FLEXCAN_F buffer 8 interrupt
FLEXCAN_F_IFLAG1_BUF9I	0x0BF4	253		FLEXCAN_F.IFLAG1[BUF9I]	FLEXCAN_F buffer 9 interrupt
FLEXCAN_F_IFLAG1_BUF10I	0x0BF8	254		FLEXCAN_F.IFLAG1[BUF10I]	FLEXCAN_F buffer 10 interrupt
FLEXCAN_F_IFLAG1_BUF11I	0x0BFC	255		FLEXCAN_F.IFLAG1[BUF11I]	FLEXCAN_F buffer 11 interrupt
FLEXCAN_F_IFLAG1_BUF12I	0x0C00	256		FLEXCAN_F.IFLAG1[BUF12I]	FLEXCAN_F buffer 12 interrupt
FLEXCAN_F_IFLAG1_BUF13I	0x0C04	257		FLEXCAN_F.IFLAG1[BUF13I]	FLEXCAN_F buffer 13 interrupt

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 12 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
FLEXCAN_F_IFLAG1_BUF14I	0x0C08	258		FLEXCAN_F.IFLAG1[BUF14I]	FLEXCAN_F buffer 14 interrupt
FLEXCAN_F_IFLAG1_BUF15I	0x0C0C	259		FLEXCAN_F.IFLAG1[BUF15I]	FLEXCAN_F buffer 15 interrupt
FLEXCAN_F_IFLAG1_BUF31_16I	0x0C10	260		FLEXCAN_F.IFLAG1 [BUF31I:BUF16I]	FLEXCAN_F buffers 31–16 interrupts
FLEXCAN_F_IFLAG2_BUF63_32I	0x0C14	261		FLEXCAN_F.IFLAG2 [BUF63I:BUF32I]	FLEXCAN_F buffers 63–32 interrupts
Reserved	0x0C18	262		Reserved	Reserved
Reserved	0x0C1C	263		Reserved	Reserved
Reserved	0x0C20	264		Reserved	Reserved
Reserved	0x0C24	265		Reserved	Reserved
Reserved	0x0C28	266		Reserved	Reserved
Reserved	0x0C2C	267		Reserved	Reserved
Reserved	0x0C30	268		Reserved	Reserved
Reserved	0x0C34	269		Reserved	Reserved
SCI_E_COMB	0x0C38	270		SCI_E.SCISR1[TDRE] SCI_E.SCISR1[TC] SCI_E.SCISR1[RDRF] SCI_E.SCISR1[IDLE] SCI_E.SCISR1[OR] SCI_E.SCISR1[NF] SCI_E.SCISR1[FE] SCI_E.SCISR1[PF] SCI_E.SCISR2[BERR] SCI_E.LINSTAT1[RXRDY] SCI_E.LINSTAT1[TXRDY] SCI_E.LINSTAT1[LWAKE] SCI_E.LINSTAT1[STO] SCI_E.LINSTAT1[PBERR] SCI_E.LINSTAT1[CEERR] SCI_E.LINSTAT1[CKERR] SCI_E.LINSTAT1[FRC] SCI_E.LINSTAT2[OVFL]	SCI_E combined interrupt request of the SCI status register 1 transmit data register empty, transmit complete, receive data register full, idle line, overrun, noise, frame error, and parity error interrupt requests, SCI status register 2 bit error interrupt request, LIN status register 1 receive data ready, transmit data ready, received LIN wakeup signal, slave timeout, physical bus error, CRC error, checksum error, frame complete interrupts requests, and LIN status register 2 receive register overflow interrupt request

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 13 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
SCI_F_COMB	0x0C3C	271		SCI_F.SCIISR1[TDRE] SCI_F.SCIISR1[TC] SCI_F.SCIISR1[RDRF] SCI_F.SCIISR1[IDLE] SCI_F.SCIISR1[OR] SCI_F.SCIISR1[NF] SCI_F.SCIISR1[FE] SCI_F.SCIISR1[PF] SCI_F.SCIISR2[BERR] SCI_F.LINSTAT1[RXRDY] SCI_F.LINSTAT1[TXRDY] SCI_F.LINSTAT1[LWAKE] SCI_F.LINSTAT1[STO] SCI_F.LINSTAT1[PBERR] SCI_F.LINSTAT1[CERR] SCI_F.LINSTAT1[CKERR] SCI_F.LINSTAT1[FRC] SCI_F.LINSTAT2[OVFL]	SCI_F combined interrupt request of the SCI status register 1 transmit data register empty, transmit complete, receive data register full, idle line, overrun, noise, frame error, and parity error interrupt requests, SCI status register 2 bit error interrupt request, LIN status register 1 receive data ready, transmit data ready, received LIN wakeup signal, slave timeout, physical bus error, CRC error, checksum error, frame complete interrupts requests, and LIN status register 2 receive register overflow interrupt request
SCI_G_COMB	0x0C40	272		SCI_G.SCIISR1[TDRE] SCI_G.SCIISR1[TC] SCI_G.SCIISR1[RDRF] SCI_G.SCIISR1[IDLE] SCI_G.SCIISR1[OR] SCI_G.SCIISR1[NF] SCI_G.SCIISR1[FE] SCI_G.SCIISR1[PF] SCI_G.SCIISR2[BERR] SCI_G.LINSTAT1[RXRDY] SCI_G.LINSTAT1[TXRDY] SCI_G.LINSTAT1[LWAKE] SCI_G.LINSTAT1[STO] SCI_G.LINSTAT1[PBERR] SCI_G.LINSTAT1[CERR] SCI_G.LINSTAT1[CKERR] SCI_G.LINSTAT1[FRC] SCI_G.LINSTAT2[OVFL]	SCI_G combined interrupt request of the SCI status register 1 transmit data register empty, transmit complete, receive data register full, idle line, overrun, noise, frame error, and parity error interrupt requests, SCI status register 2 bit error interrupt request, LIN status register 1 receive data ready, transmit data ready, received LIN wakeup signal, slave timeout, physical bus error, CRC error, checksum error, frame complete interrupts requests, and LIN status register 2 receive register overflow interrupt request
SCI_H_COMB	0x0C44	273		SCI_H.SCIISR1[TDRE] SCI_H.SCIISR1[TC] SCI_H.SCIISR1[RDRF] SCI_H.SCIISR1[IDLE] SCI_H.SCIISR1[OR] SCI_H.SCIISR1[NF] SCI_H.SCIISR1[FE] SCI_H.SCIISR1[PF] SCI_H.SCIISR2[BERR] SCI_H.LINSTAT1[RXRDY] SCI_H.LINSTAT1[TXRDY] SCI_H.LINSTAT1[LWAKE] SCI_H.LINSTAT1[STO] SCI_H.LINSTAT1[PBERR] SCI_H.LINSTAT1[CERR] SCI_H.LINSTAT1[CKERR] SCI_H.LINSTAT1[FRC] SCI_H.LINSTAT2[OVFL]	SCI_H combined interrupt request of the SCI status register 1 transmit data register empty, transmit complete, receive data register full, idle line, overrun, noise, frame error, and parity error interrupt requests, SCI status register 2 bit error interrupt request, LIN status register 1 receive data ready, transmit data ready, received LIN wakeup signal, slave timeout, physical bus error, CRC error, checksum error, frame complete interrupts requests, and LIN status register 2 receive register overflow interrupt request

Table 8-2. Interrupt Summary for External Input to e200z1 or e200z0 (Sheet 14 of 14)

Interrupt	Offset ¹	Vector	Priority ²	Source	Description
DSPI_C_ISR_OVER	0x0C48	274		DSPI_C.DSPI_ISR[TFUF] DSPI_C.DSPI_ISR[RFOF]	DSPI_C combined overrun interrupt request of the transmit FIFO underflow and receive FIFO overflow interrupt requests
DSPI_C_ISR_EOQF	0x0C4C	275		DSPI_C.DSPI_ISR[EOQF]	DSPI_C transmit FIFO end-of-queue flag
DSPI_C_ISR_TFFF	0x0C50	276		DSPI_C.DSPI_ISR[TFFF]	DSPI_C transmit FIFO fill flag
DSPI_C_ISR_TCF	0x0C54	277		DSPI_C.DSPI_ISR[TCF]	DSPI_C transfer complete flag
DSPI_C_ISR_RFDF	0x0C58	278		DSPI_C.DSPI_ISR[RFDF]	DSPI_C receive FIFO drain flag
DSPI_D_ISR_OVER	0x0C5C	279		DSPI_D.DSPI_ISR[TFUF] DSPI_D.DSPI_ISR[RFOF]	DSPI_D combined overrun interrupt request of the Transmit FIFO Underflow and Receive FIFO Overflow interrupt requests
DSPI_D_ISR_EOQF	0x0C60	280		DSPI_D.DSPI_ISR[EOQF]	DSPI_D transmit FIFO End Of Queue flag
DSPI_D_ISR_TFFF	0x0C64	281		DSPI_D.DSPI_ISR[TFFF]	DSPI_D Transmit FIFO Fill flag
DSPI_D_ISR_TCF	0x0C68	282		DSPI_D.DSPI_ISR[TCF]	DSPI_D Transfer Complete flag
DSPI_D_ISR_RFDF	0x0C6C	283		DSPI_D.DSPI_ISR[RFDF]	DSPI_D Receive FIFO Drain flag
FlexRay_GLOB	0x0C70	284		FLEXRAY.CIFRR[7]	Global FlexRay module interrupt flag
FlexRay_PRIF	0x0C74	285		FLEXRAY.CIFRR[6]	FlexRay protocol status and error interrupt flag
FlexRay_CHIF	0x0C78	286		FLEXRAY.CIFRR[5]	FlexRay controller host interface error interrupt flag
FlexRay_WUP_IF	0x0C7C	287		FLEXRAY.CIFRR[4]	FlexRay wakeup interrupt flag
FlexRay_FBNE_F	0x0C80	288		FLEXRAY.CIFRR[3]	FlexRay receive FIFO B not empty intr. flag
FlexRay_FANE_F	0x0C84	289		FLEXRAY.CIFRR[2]	FlexRay Receive FIFO A not Empty intr. flag
FlexRay_RBIF	0x0C88	290		FLEXRAY.CIFRR[1]	FlexRay combined receive buffer intr. flag
FlexRay_TBIF	0x0C8C	291		FLEXRAY.CIFRR[0]	FlexRay combined transmit buffer intr. flag.
Reserved	0x0C90	292		Reserved	Reserved
MLB_Service_Request	0x0C94	293		MLB.MSR[MSVRQS]	MLB Service Request

¹ Offsets are for hardware vector mode only. In software vector mode all interrupts have the same offset of 0x0040.

² The priorities are selected in INTC_PSRx_x, where the specific select register is assigned according to the vector. This column is for the user to fill in how they set their specific priorities.

8.4 Interrupt Operation

8.4.1 Software Vector Mode

In software vector mode, the interrupt exception handler acknowledges the interrupt request to the e200z1/0 from the INTC by reading the INTC_IACKR. The e200z1/0 is enabled again to recognize the external input by setting the EE bit of the MSR. The prolog of the interrupt exception handler must acknowledge the interrupt request before the e200z1/0 is enabled again to recognize the external input. Otherwise, the e200z1/0 will attempt to service the same source of the interrupt request.

The INTC's LIFO is popped by writing to the INTC_EOIR. The e200z1/0's recognition of the external input is disabled by clearing the EE bit of the MSR. In the epilog of the interrupt exception handler, the timing relationship between popping the LIFO and disabling recognition of the external input has no restrictions. The writes can happen in either order. However, disabling recognition of the external input before popping the LIFO eases the calculation of the maximum pipe depth at the cost of postponing the servicing of the next interrupt request.

8.4.2 Hardware Vector Mode

In hardware vector mode, the interrupt request to the e200z1/0 from the INTC is acknowledged before the e200z1/0 starts to execute the exception handler. The INTC_IACKR does not need to be read to acknowledge the interrupt request before the e200z1/0 is enabled again to recognize the external input.

As in software vector mode, the timing relationship between popping the LIFO and disabling recognition of the external input has no restrictions. Also, as in software vector mode, disabling recognition of the external input before popping the LIFO eases the calculation of the maximum pipe depth at the cost of postponing the servicing of the next interrupt request.

8.4.3 Non Maskable Interrupt (NMI)

The MPC5510 can be configured to use the pins PD[10] and PD[11] as non maskable interrupts (NMI) by providing a path to the critical interrupt input of the e200Z1 and e200z0 cores, respectively.

After the SIU is configured by user code, an NMI cannot be prevented from reaching the assigned core. The only possible way of disabling the critical interrupt is by clearing the critical interrupt enable (CE) bit in the core's machine state register (MSR). The NMI will have a higher priority than any interrupt request generated by the INTC, and will not be blocked or preempted by any other INTC interrupt request.

After the SIU is properly configured, the operation of the NMI always generates an interrupt request when the programmed edge transition occurs on the pin, regardless of the selected muxing on that pin. It is the user's responsibility to assign pin multiplexing correctly for use with an NMI, which would normally mean selecting it as a port pin rather than a peripheral function.

Figure 8-5 shows the various system level connections needed to create the NMI.

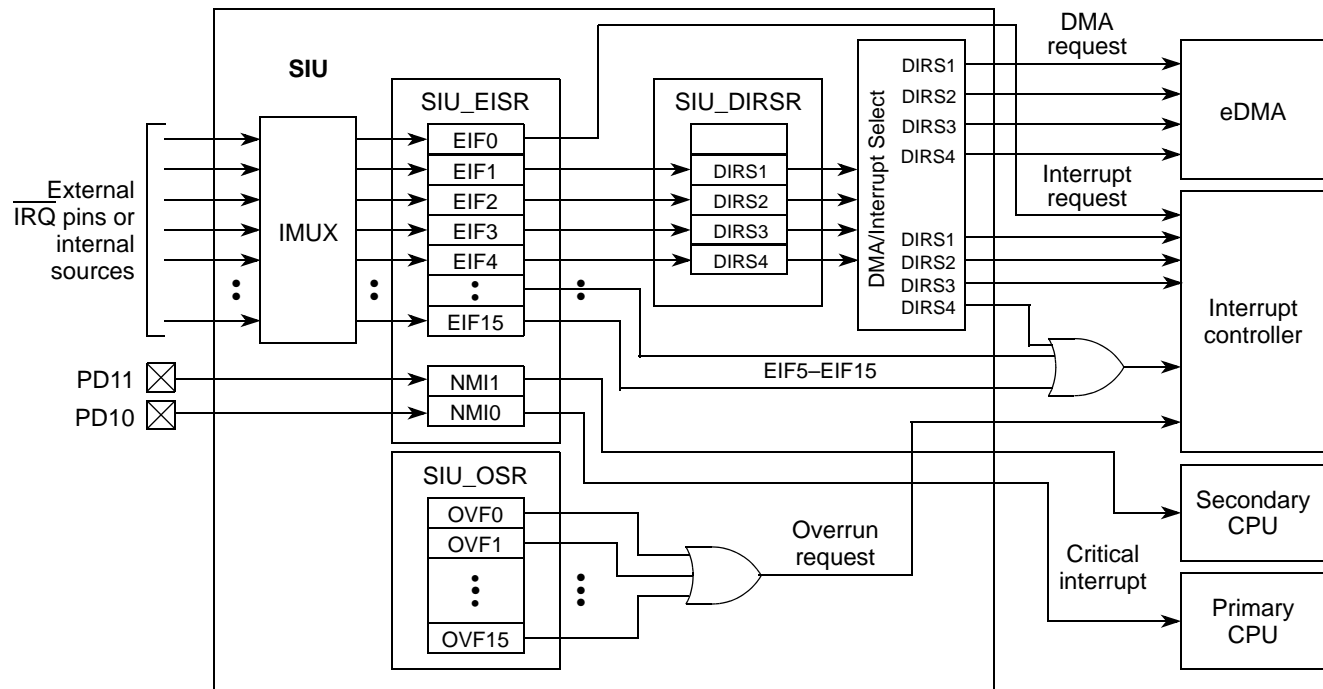


Figure 8-5. NMI Connections

8.4.4 Dynamic Priority Elevation

Zen Core processors support critical and external interrupts. Each processor can be configured to employ priority elevation on critical and/or external interrupt events. Critical interrupts come from external pins PD10 or PD11 or the SoftMLB interface logic, and are routed to the processor's critical interrupt input. External interrupts come from the peripherals and are routed through the interrupt controller. In addition to the interrupt notification signals, various processor specific configuration flags from the Zen processor's Machine Check Register (MCR[ee,ce]) and the Hardware Implementation register (HID1) are sent to the Miscellaneous Control Module (MCM) to determine when interrupt servicing is enabled and when high-priority elevation should be enabled. If the corresponding processor is configured to allow high-priority elevation on critical interrupt events, the MCM generates the high-priority signal upon critical interrupt detection and holds it active for the duration of interrupt servicing until a return from critical interrupt (rfci) is detected. If the corresponding processor is configured to allow high-priority elevation on external interrupt events, the MCM generates the high-priority signal upon external interrupt detection and holds it active for the duration of interrupt servicing, until a return from interrupt (rfi) is detected.

Great care must be taken when using the priority elevation as it can enable a master to starve the rest of the masters in the system.

8.4.4.1 Hardware Implementation Dependent Register 1

The HID1 register is used for bus configuration and system control. HID1 is shown in [Figure 8-6](#).

Zen Core SPR 1009												Access: User read/write				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BAECE	BAEEE	0	0	0	0	0	0	ATS	0	0	0	0	0	0	0
W	BAECE	BAEEE	0	0	0	0	0	0	ATS	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-6. Hardware Implementation Dependent Register 1 (HID1)

The HID1 fields are defined in [Table 8-3](#).

Table 8-3. HID1 Field Description

Field	Description
bits 0–15	Reserved. ¹
BAECE	Bus Arbitration Elevation for Critical Interrupts Enable. This bit enables the dynamic elevation of the processor's system bus arbitration priority during critical interrupt processing. If set, this bit enables the elevated arbitration priority during the time when the critical interrupt request is asserted. If cleared, the processor's arbitration priority does not change when the critical interrupt request is asserted.
BAEEE	Bus Arbitration Elevation for External Interrupts Enable. This bit enables the dynamic elevation of the processor's system bus arbitration priority during external interrupt processing. If set, this bit enables the elevated arbitration priority during the time when the external interrupt request is asserted. If cleared, the processor's arbitration priority does not change when the external interrupt request is asserted.
bits 18–23	Reserved. ¹
ATS	Atomic status (read-only). Indicates state of the reservation bit in the load/store unit.
bit 31	Reserved. ¹

¹ These bits are not implemented and should be written with zero for future compatibility.

Chapter 9

Interrupt Controller (INTC)

9.1 Introduction

The INTC provides priority-based preemptive scheduling of interrupt service requests (ISRs). This scheduling scheme is suitable for statically scheduled hard real-time systems. The INTC supports 294 interrupt requests. It is targeted to work with Power Architecture technology and automotive applications where the ISRs nest to multiple levels, but it also can be used with other processors and applications.

For high-priority interrupt requests in these target applications, the time from the assertion of the peripheral's interrupt request to when the processor is performing useful work to service the interrupt request needs to be minimized. The INTC supports this goal by providing a unique vector for each interrupt request source. It also provides 16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. Because each individual application will have different priorities for each source of interrupt request, the priority of each interrupt request is configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks which share the resource can not preempt each other.

Multiple processors can assert interrupt requests to each other through software settable interrupt requests. These software settable interrupt requests can also be used to separate the work involved in servicing an interrupt request into a high-priority portion and a low-priority portion. The high-priority portion is initiated by a peripheral interrupt request, but then the ISR can assert a software settable interrupt request to finish the servicing in a lower priority ISR. Therefore these software settable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

9.1.1 Features

- Supports 286 peripheral and eight software-settable interrupt request sources.
- Each interrupt source can be steered by software to processor 0 (Z1), processor 1 (Z0) or both processors interrupt request outputs.

NOTE

By default, processor 0 (Z1) receives all interrupt requests, so backward compatibility with single processor systems is maintained.

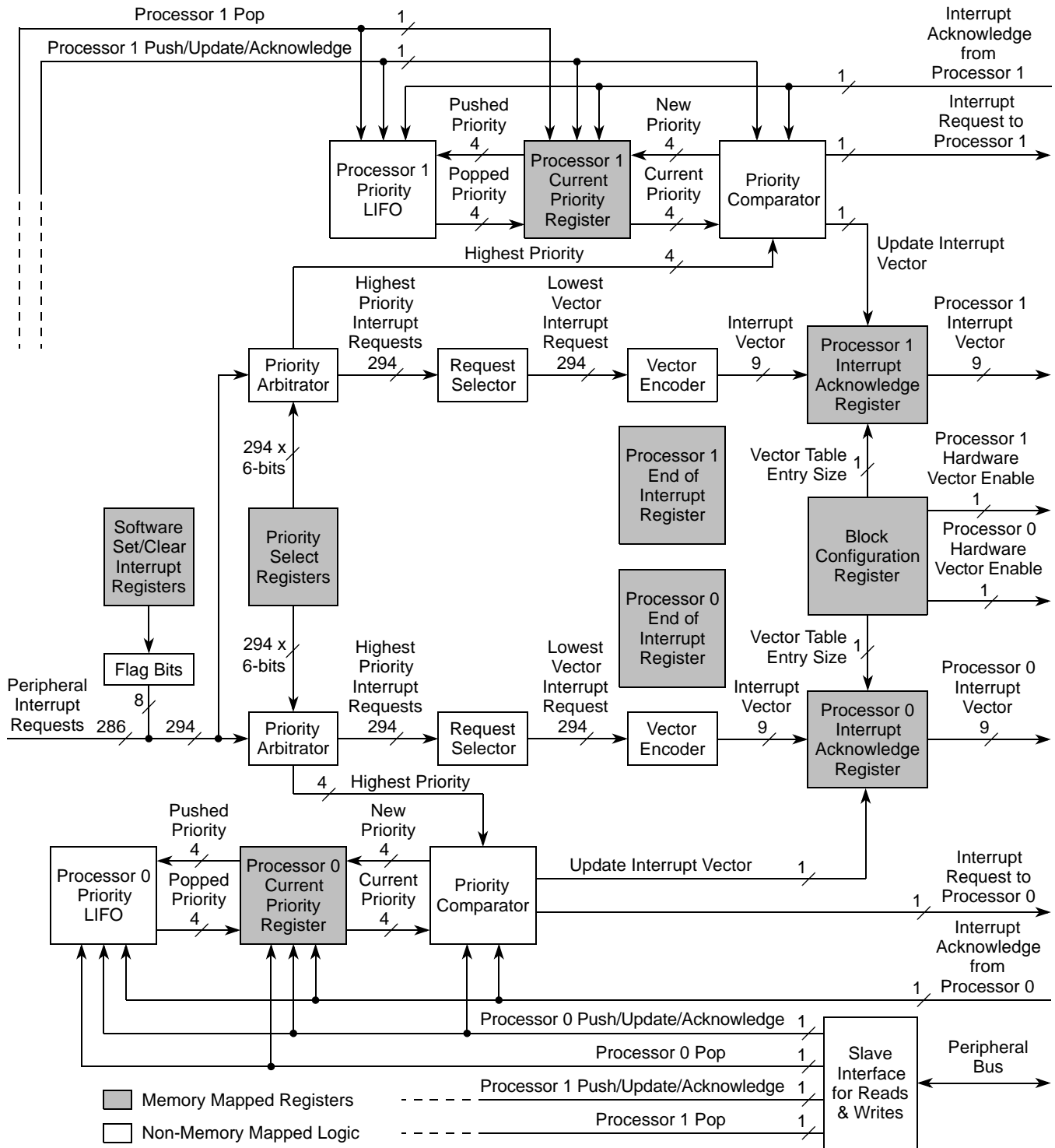
When sending an interrupt to both cores, the user must take care to prevent the interrupt from going away from the other core when not expected.

- 9-bit vector
 - Unique vector for each interrupt request source

- Hardware connection to processor or read from register
- Each interrupt source can be programmed to one of 16 priorities
- Preemption
 - Preemptive prioritized interrupt requests to processor
 - ISR at a higher priority preempts ISRs or tasks at lower priorities
 - Automatic pushing or popping of preempted priority to or from a LIFO
 - Ability to modify the ISR or task priority; modifying the priority can be used to implement the priority ceiling protocol for accessing shared resources.
- Low latency—three clocks from receipt of interrupt request from peripheral to interrupt request to processor.

9.1.2 Block Diagram

Figure 9-1 is a block diagram of the interrupt controller (INTC).



NOTE: Processor 0 is Z1 and Processor 1 is Z0.

Figure 9-1. INTC Block Diagram

9.1.3 Modes of Operation

9.1.3.1 Normal Mode

In normal mode, the INTC has two handshaking modes with the processor: software vector mode and hardware vector mode.

9.1.3.1.1 Software Vector Mode

In software vector mode, the interrupt exception handler software must read a register in the INTC to obtain the vector associated with the interrupt request to the processor. The INTC will use software vector mode for a given processor when its associated HVEN_PRC0 or HVEN_PRC1 bit in INTC_MCR is negated. The hardware vector enable signal to processor 0 or processor 1 is driven as negated when its associated HVEN_PRC n bit is negated. The vector is read from INC_IACKR_PRC0 or INTC_IACKR_PRC1. Reading the INTC_IACKR_PRC n negates the interrupt request to the associated processor. Even if a higher priority interrupt request arrived while waiting for this interrupt acknowledge, the interrupt request to the processor will negate for at least one clock. The reading also pushes the PRI value in INTC_CPR_PRC0 or INTC_CPR_PRC1 onto the associated LIFO and updates PRI in the associated INTC_CPR_PRC n with the new priority.

Furthermore, the interrupt vector to the processor is driven as all 0s. The interrupt acknowledge signal from the associated processor is ignored.

9.1.3.1.2 Hardware Vector Mode

In hardware vector mode, the hardware signals the interrupt vector from the INTC in conjunction with a processor that can use that vector. This hardware causes the first instruction to be executed in handling the interrupt request to the processor to be specific to that vector. Therefore, the interrupt exception handler is specific to a peripheral or software settable interrupt request rather than being common to all of them. The INTC uses hardware vector mode for a given processor when the associated HVEN_PRC0 or HVEN_PRC1 bit in the INTC_MCR is asserted. The hardware vector enable signal to the associated processor is driven as asserted. When the interrupt request to the associated processor asserts, the interrupt vector signal is updated. The value of that interrupt vector is the unique vector associated with the preempting peripheral or software settable interrupt request. The vector value matches the value of the INTVEC_PRC0 field in the INTC_IACKR_PRC0 or the INTVEC_PRC1 field in the INTC_IACKR_PRC1, depending on which processor was assigned to handle a given interrupt source.

The processor negates the interrupt request to the processor driven by the INTC by asserting the interrupt acknowledge signal for one clock. Even if a higher priority interrupt request arrived while waiting for the interrupt acknowledge, the interrupt request to the processor will negate for at least one clock.

The assertion of the interrupt acknowledge signal for a given processor pushes the associated PRI value in the associated INTC_CPR_PRC n register onto the associated LIFO and updates the associated PRI in the associated INTC_CPR_PRC n register with the new priority. This pushing of the PRI value onto the associated LIFO and updating PRI in the associated INTC_CPR_PRC n does not occur when the associated interrupt acknowledge signal asserts and INTC_SSCIR0_3–INTC_SSCIR4_7 is written at a time such that the PRI value in the associated INTC_CPR_PRC n register would need to be pushed and the previously

last pushed PRI value would need to be popped simultaneously. In this case, PRI in the associated INTC_CPR_PRC n is updated with the new priority, and the associated LIFO is neither pushed or popped.

9.1.3.2 Debug Mode

The INTC operation in debug mode is identical to its operation in normal mode.

9.1.3.3 Stop Mode

The INTC supports stop mode. The INTC can have its clock input disabled at any time by the clock driver on the device. While its clocks are disabled, the INTC registers are not accessible.

The INTC requires clocking in order for a peripheral interrupt request to generate an interrupt request to the processor. Since the INTC is not clocked in stop mode, peripheral interrupt requests can not be used as a wakeup source, unless the clock, reset, and power module (CRP) supports that interrupt request as a wakeup source.

9.2 Signal Description

The INTC has no external signals.

9.3 Memory Map and Registers

9.3.1 Module Memory Map

Table 9-1 shows the INTC memory map.

Table 9-1. INTC Memory Map

Offset from INTC_BASE_ADDR (0xFFFF4_8000)	Register	Access	Reset Value	Section/Page
0x0000	INTC_MCR—INTC module configuration register	R/W	0x0000_0000	9.3.2.1/9-6
0x0004	Reserved	—	—	—
0x0008	INTC_CPR_PRC0—INTC current priority register for processor 0 (Z1)	R/W	0x0000_000F	9.3.2.2/9-7
0x000C	INTC_CPR_PRC1—INTC current priority register for processor 1 (Z0)	R/W	0x0000_000F	9.3.2.3/9-8
0x0010	INTC_IACKR_PRC0—INTC interrupt acknowledge register for processor 0 (Z1)	R ¹ /W	0x0000_0000	9.3.2.4/9-9
0x0014	INTC_IACKR_PRC1—INTC interrupt acknowledge register for processor 1 (Z0)	R ¹ /W	0x0000_0000	9.3.2.5/9-10
0x0018	INTC_EOIR_PRC0—INTC end of interrupt register for processor 0 (Z1)	W	0x0000_0000	9.3.2.6/9-10
0x001C	INTC_EOIR_PRC1—INTC end of interrupt register for processor 1 (Z0)	W	0x0000_0000	9.3.2.7/9-11

Table 9-1. INTC Memory Map (continued)

Offset from INTC_BASE_ADDR (0xFFFF4_8000)	Register	Access	Reset Value	Section/Page
0x0020–0x0024	INTC_SSCIR0_3—INTC software set/clear interrupt register 0–3 INTC_SSCIR4_7—INTC software set/clear interrupt register 4–7	R/W	0x0000_0000	9.3.2.8/9-11
0x0028–0x003C	Reserved	—	—	—
0x0040–0x0164	INTC_PSR0_3—INTC priority select register 0–3 — INTC_PSR292_293—INTC priority select register 292–293	R/W	0x0000_0000	9.3.2.9/9-12

¹ When the HVEN bit in the INTC module configuration register (INTC_MCR) is asserted, a read of the INTC_IACKR has no side effects.

9.3.2 Register Descriptions

All registers are 32 bits in width. Any combination of accessing the four bytes of a register with a single access is supported, provided that the access does not cross a register boundary. These supported accesses include types and sizes of eight bits, aligned 16 bits, misaligned 16 bits to the middle two bytes, and aligned 32 bits.

In software vector mode, the side effects of a read of INTC_IACKR_PRC0 and INTC_IACR_PRC1 are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to either INTC_SSCIR0_3–INTC_SSCIR4_7 or INTC_EOIR_PRC0–INTC_EOIR_PRC1 does not affect the operation of the write.

9.3.2.1 INTC Module Configuration Register (INTC_MCR)

The module configuration register is used to configure options of the INTC.

Offset: 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	VTES_PRC1	0	0	0	0	HVEN_PRC1	0	0	VTES_PRC0	0	0	0	0	HVEN_PRC0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-2. INTC Module Configuration Register (INTC_MCR)

Table 9-2. INTC_MCR Field Descriptions

Field	Description
VTES_PRC1 VTES_PRC0	For software mode only, the Vector Table Entry Size for Processor 0 (Z1) and Processor 1 (Z0). The VTES_PRC0 bit controls the number of 0s to the right of INTVEC_PRC0 in INTC_IACKR_PRC0. The VTES_PRC1 bit controls the number of 0s to the right of INTVEC_PRC1 in INTC_IACKR_PRC1. If the contents of INTC_IACKR_PRC0 or INTC_IACKR_PRC1 are used as an address of an entry in a vector table, then the number of rightmost 0s will determine the size of each vector table entry. 0 4 bytes of address offset between vectors. 1 8 bytes of address offset between vectors. Note: A larger table may be useful if the interrupt service routines (ISR) require very few instructions; however, more typically, the smaller 4-byte size is used as a jump table to the actual ISRs.
HVEN_PRC1 HVEN_PRC0	Hardware Vector Enable for Processor 0 (Z1) and Processor 1 (Z0). The HVEN bit controls whether the INTC is in hardware vector mode or software vector mode. Refer to Section 9.1.3.1, “Normal Mode” for details of handshaking with the processor in each mode. 0 Software vector mode. 1 Hardware vector mode.

9.3.2.2 INTC Current Priority Register for Processor 0 (Z1) (INTC_CPR_PRC0)

Offset: 0x0008

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PRI			
W																																				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 9-3. INTC Current Priority Register for Processor 0 (Z1) (INTC_CPR_PRC0)

Table 9-3. INTC_CPR_PRC0 Field Descriptions

Field	Description
PRI	Priority. PRI is the priority of the currently executing ISR according to the field values defined in Table 9-4 .

The current priority register masks any peripheral or software settable interrupt request at the same or lower priority of the current value than the PRI field in INTC_CPR_PRC0 from generating an interrupt request to processor 0 (Z1). When INTC_IACKR_PRC0 is read in software vector mode, or the interrupt acknowledge signal from the processor is asserted in hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When INTC_SSCIR0_3–INTC_SSCIR4_7 is written, the LIFO is popped into the INTC_CPR_PRC0's PRI field. An exception case in hardware vector mode to this behavior is described in [Section 9.1.3.1.2, “Hardware Vector Mode”](#).

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to [Section 9.5.5, “Priority Ceiling Protocol.”](#)

NOTE

A store to modify the PRI field that closely precedes or follows an access to a shared resource can result in a non-coherent access to the resource. Refer to [Section 9.5.5.2, “Ensuring Coherency”](#) for example code to ensure coherency.

Table 9-4. PRI Values

PRI	Meaning
1111	Priority 15—highest priority
1110	Priority 14
1101	Priority 13
1100	Priority 12
1011	Priority 11
1010	Priority 10
1001	Priority 9
1000	Priority 8
0111	Priority 7
0110	Priority 6
0101	Priority 5
0100	Priority 4
0011	Priority 3
0010	Priority 2
0001	Priority 1
0000	Priority 0—lowest priority

9.3.2.3 INTC Current Priority Register for Processor 1 (Z0) (INTC_CPR_PRC1)

Offset: 0x000C

Access: User read/write

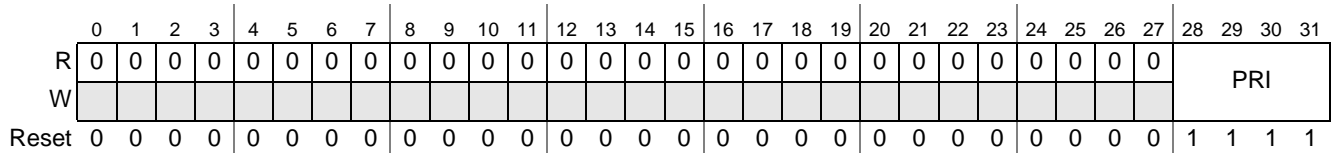


Figure 9-4. INTC Current Priority Register for Processor 1 (Z0) (INTC_CPR_PRC1)

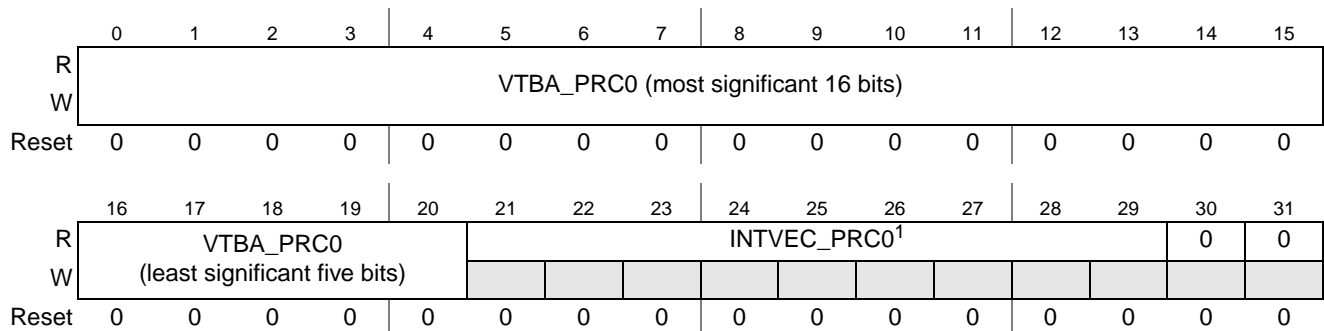
Table 9-5. INTC_CPR_PRC1 Field Descriptions

Field	Description
PRI	Priority. The function of this register is the same as described for processor 0 (Z1) in Section 9.3.2.2, “INTC Current Priority Register for Processor 0 (Z1) (INTC_CPR_PRC0)”

9.3.2.4 INTC Interrupt Acknowledge Register for Processor 0 (Z1) (INTC_IACKR_PRC0)

Offset: 0x0010

Access: User read/write



¹ When the VTES_PRC0 bit in INTC_MCR is asserted, INTVEC_PRC0 is shifted to the left one bit. Bit 29 is read as a 0. VTBA_PRC0 is narrowed to 20 bits in width.

Figure 9-5. INTC Interrupt Acknowledge Register for Processor 0 (Z1) (INTC_IACKR_PRC0)

Table 9-6. INTC_IACKR_PRC0 Field Descriptions

Field	Description
VTBA_PRC0	Vector Table Base Address for Processor 0 (Z1). VTBA_PRC0 can be the base address of a vector table of addresses of ISRs for processor 0 (Z1). The VTBA_PRC0 only uses the leftmost 20 bits when the VTES_PRC0 bit in INTC_MCR is asserted.
INTVEC_PRC0	Interrupt Vector for Processor 0 (Z1). INTVEC_PRC0 is the vector of the peripheral or software settable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC_PRC0 is updated, whether the INTC is in software or hardware vector mode.

The interrupt acknowledge register for processor 0 (Z1) provides a value that can be used to load the address of an ISR from a vector table. The vector table can be composed of addresses of the ISRs specific to their respective interrupt vectors.

In software vector mode, the INTC_IACKR_PRC0 has side effects from reads. Therefore, it must not be speculatively read while in this mode. The side effects are the same regardless of the size of the read. Reading the INTC_IACKR_PRC0 does not have side effects in hardware vector mode.

9.3.2.5 INTC Interrupt Acknowledge Register for Processor 1 (Z0) (INTC_IACKR_PRC1)

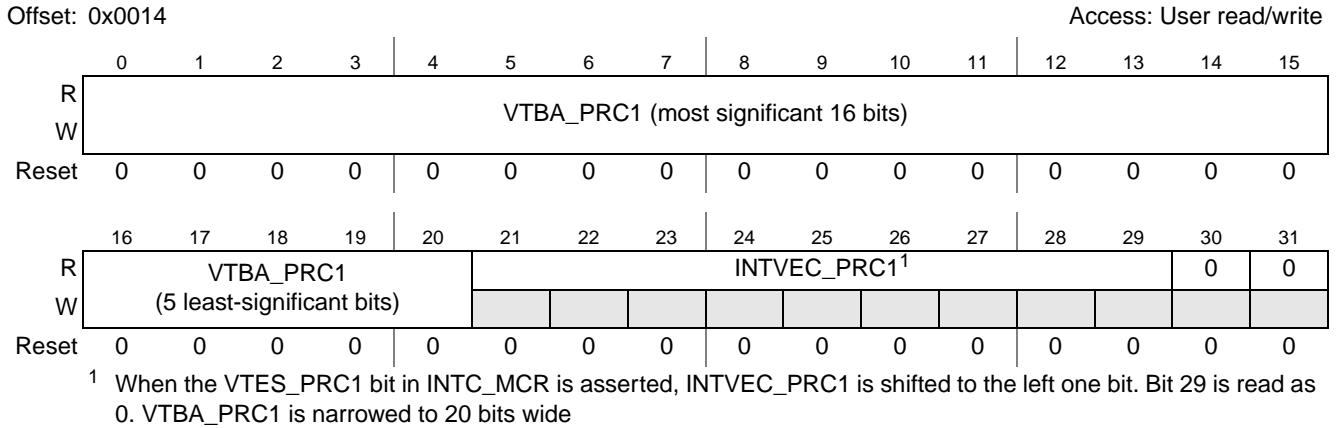


Figure 9-6. INTC Interrupt Acknowledge Register for Processor 1 (Z0) (INTC_IACKR_PRC1)

Table 9-7. INTC_IACKR_PRC1 Field Descriptions

Field	Description
VTBA_PRC1	The register's function is the same as described for processor 0 (Z1) in Section 9.3.2.4, "INTC Interrupt Acknowledge Register for Processor 0 (Z1) (INTC_IACKR_PRC0)"
INTVEC_PRC1	

9.3.2.6 INTC End-of-Interrupt Register for Processor 0 (Z1) (INTC_EOIR_PRC0)

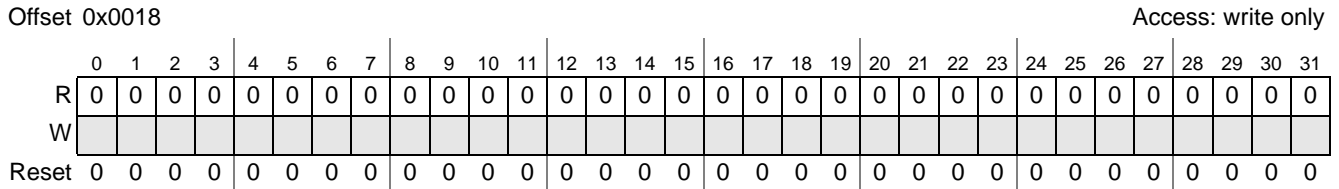


Figure 9-7. INTC End-of-Interrupt Register for Processor 0 (Z1) (INTC_EOIR_PRC0)

Writing to the end-of-interrupt register signals the end of the servicing of the interrupt request. When the INTC_EOIR_PRC0 is written, the priority last pushed on the LIFO is popped into INTC_CPR_PRC0. An exception to this behavior is described in [Section 9.1.3.1.2, "Hardware Vector Mode."](#) The values and size of data written to the INTC_EOIR_PRC0 are ignored. The values and sizes written to this register neither update the INTC_EOIR_PRC0 contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR_PRC0.

9.3.2.7 INTC End-of-Interrupt Register for Processor 1 (Z0) (INTC_EOIR_PRC1)

Offset: 0x001C

Access: write only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 9-8. INTC End-of-Interrupt Register for Processor 1 (Z0) (INTC_EOIR_PRC1)

The register's function is the same as for processor 0 (Z1) as described in [Section 9.3.2.6, "INTC End-of-Interrupt Register for Processor 0 \(Z1\) \(INTC_EOIR_PRC0\)."](#)

9.3.2.8 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)

Offset: 0x0020

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR0	0	0	0	0	0	0	0	CLR1
W							SET0	w1c							SET1	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR2	0	0	0	0	0	0	0	CLR3
W							SET2	w1c							SET3	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-9. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3])

Offset: 0x0024

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR4	0	0	0	0	0	0	0	CLR5
W							SET4	w1c							SET5	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR6	0	0	0	0	0	0	0	CLR7
W							SET6	w1c							SET7	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-10. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7])

Table 9-8. INTC_SSCIR[0:7] Field Descriptions

Field	Description
SETn	Set Flag Bits. Writing a 1 sets the corresponding CLRn bit. Writing a 0 has no effect. Each SETn always will be read as a 0.
CLRn	Clear Flag Bits. CLRn is the flag bit. Writing a 1 to CLRn clears it provided that a 1 is not written simultaneously to its corresponding SETn bit. Writing a 0 to CLRn has no effect. 0 Interrupt request not pending within INTC. 1 Interrupt request pending within INTC.

The software set/clear interrupt registers support the setting or clearing of software settable interrupt request. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. Except for being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC like a peripheral interrupt request. Writing a 1 to SETn will leave SETn unchanged at 0 but sets CLRn. Writing a 0 to SETn has no effect. CLRn is the flag bit. Writing a 1 to CLRn clears it. Writing a 0 to CLRn has no effect. If a 1 is written simultaneously to a pair of SETn and CLRn bits, CLRn will be asserted, regardless of whether CLRn was asserted before the write.

9.3.2.9 INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR292_293)

Offset: 0x0040

Access: User read/write

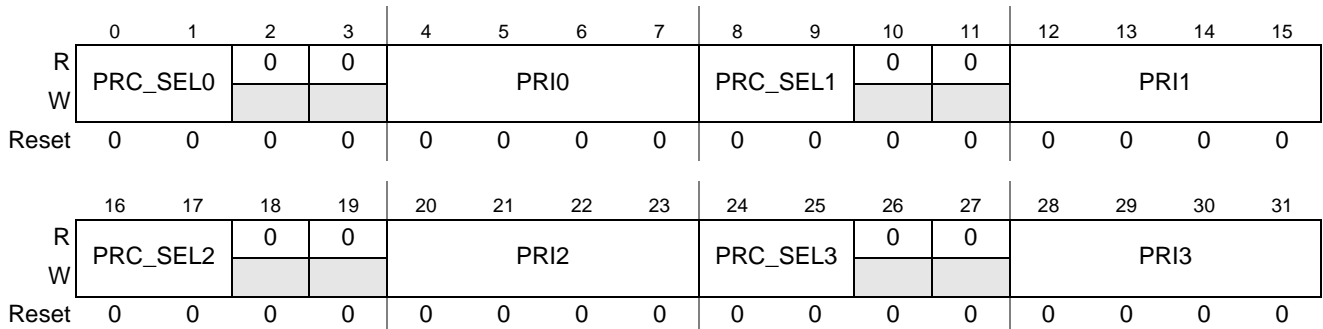


Figure 9-11. INTC Priority Select Register 0–3 (INTC_PSR0–3)

Offset: 0x0164

Access: User read/write

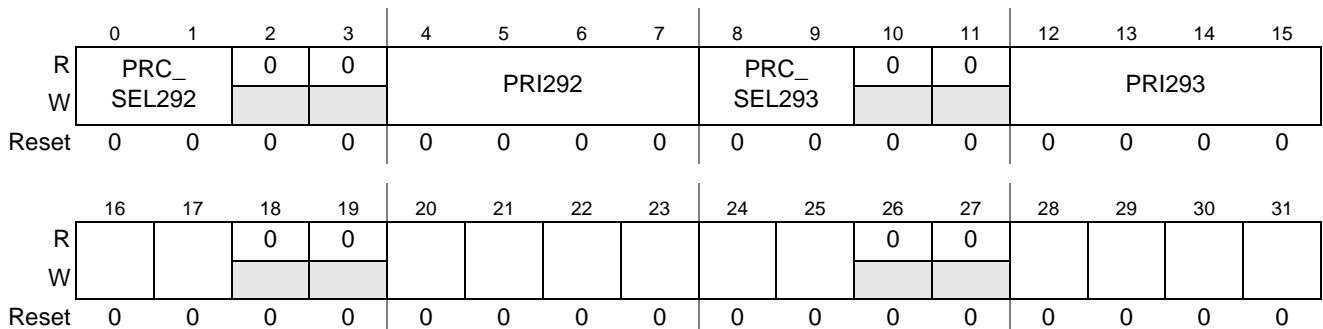


Figure 9-12. INTC Priority Select Register 292–293 (INTC_PSR292–293)

Table 9-9. INTC_PSR0_3–INTC_PSR292_293 Field Descriptions

Field	Description
PRC_SEL0– PRC_SEL293	Processor Select. If an interrupt source is enabled, PRC_SEL n selects whether the interrupt request is to be sent to processor 0 (Z1), processor 1 (Z0), or both. See Table 9-11 .
PRI0– PRI293	Priority Select. PRI n selects the priority for interrupt requests. Refer to Section 9.4, “Functional Description.” Note: In INTC_PSR292_293, bits 16, 17, 20–25, 28–31 can be read and written; however, writing has no effect other than to set or clear the bits. Reading returns the values written to the bits.

Table 9-10. INTC Priority Select Register Address Offsets

INTC_PSR n_n	Offset Address	INTC_PSR n_n	Offset Address
INTC_PSR0_3	0x0040	INTC_PSR148_151	0x00D4
INTC_PSR4_7	0x0044	INTC_PSR152_155	0x00D8
INTC_PSR8_11	0x0048	INTC_PSR156_159	0x00DC
INTC_PSR12_15	0x004C	INTC_PSR160_163	0x00E0
INTC_PSR16_19	0x0050	INTC_PSR164_167	0x00E4
INTC_PSR20_23	0x0054	INTC_PSR168_171	0x00E8
INTC_PSR24_27	0x0058	INTC_PSR172_175	0x00EC
INTC_PSR28_31	0x005C	INTC_PSR176_179	0x00F0
INTC_PSR32_35	0x0060	INTC_PSR180_183	0x00F4
INTC_PSR36_39	0x0064	INTC_PSR184_187	0x00F8
INTC_PSR40_43	0x0068	INTC_PSR188_191	0x00FC
INTC_PSR44_47	0x006C	INTC_PSR192_195	0x0100
INTC_PSR48_51	0x0070	INTC_PSR196_199	0x0104
INTC_PSR52_55	0x0074	INTC_PSR200_203	0x0108
INTC_PSR56_59	0x0078	INTC_PSR204_207	0x010C
INTC_PSR60_63	0x007C	INTC_PSR208_211	0x0110
INTC_PSR64_67	0x0080	INTC_PSR212_215	0x0114
INTC_PSR68_71	0x0084	INTC_PSR216_219	0x0118
INTC_PSR72_75	0x0088	INTC_PSR220_223	0x011C
INTC_PSR76_79	0x008C	INTC_PSR224_227	0x0120
INTC_PSR80_83	0x0090	INTC_PSR228_231	0x0124
INTC_PSR84_87	0x0094	INTC_PSR232_235	0x0128
INTC_PSR88_91	0x0098	INTC_PSR236_239	0x012C
INTC_PSR92_95	0x009C	INTC_PSR240_243	0x0130
INTC_PSR96_99	0x00A0	INTC_PSR244_247	0x0134

Table 9-10. INTC Priority Select Register Address Offsets

INTC_PSR n_n	Offset Address	INTC_PSR n_n	Offset Address
INTC_PSR100_103	0x00A4	INTC_PSR248_251	0x0138
INTC_PSR104_107	0x00A8	INTC_PSR252_255	0x013C
INTC_PSR108_111	0x00AC	INTC_PSR256_259	0x0140
INTC_PSR112_115	0x00B0	INTC_PSR260_263	0x0144
INTC_PSR116_119	0x00B4	INTC_PSR264_267	0x0148
INTC_PSR120_123	0x00B8	INTC_PSR268_271	0x014C
INTC_PSR124_127	0x00BC	INTC_PSR272_275	0x0150
INTC_PSR128_131	0x00C0	INTC_PSR276_279	0x0154
INTC_PSR132_135	0x00C4	INTC_PSR280_283	0x0158
INTC_PSR136_139	0x00C8	INTC_PSR284_287	0x015C
INTC_PSR140_143	0x00CC	INTC_PSR288_291	0x0160
INTC_PSR144_147	0x00D0	INTC_PSR292_293	0x0164

The priority select registers support the selection of an individual priority for each source of interrupt request, and whether the interrupt request is to be sent to processor 0 (Z1), processor 1, (Z0) or both. The unique vector of each peripheral or software settable interrupt request determines which INTC_PSR n_n is assigned to that interrupt request. The software settable interrupt requests 0–7 are assigned vectors 0–7, and their priorities are configured in INTC_PSR0_3 and INTC_PSR4_7, respectively. The peripheral interrupt requests are assigned vectors 8–293, and their priorities are configured in INTC_PSR8_11 through INTC_PSR292_293, respectively (see [Section 8.3.1, “Interrupt Source Summary Table”](#)).

NOTE

The PRC_SEL n or PRI n field of an INTC_PSR n_n must not be modified while the corresponding peripheral or software settable interrupt request is asserted.

Table 9-11. Selected Processor for Interrupt Request

PRC_SEL n	Meaning
00	Interrupt request sent to processor 0 (Z1)
01	Interrupt request sent to both processors
10	Reserved
11	Interrupt request sent to processor 1 (Z0)

NOTE

When sending an interrupt to both cores, the user must take care to prevent the interrupt from going away from the other core when not expected.

9.4 Functional Description

The functional description involves the areas of interrupt request sources, priority management, and handshaking with the processor.

9.4.1 Interrupt Request Sources

The INTC has two types of interrupt requests, peripheral and software settable. These interrupt requests can assert on any clock cycle.

NOTE

The INTC has no spurious vector support. Therefore, if an asserted peripheral or software settable interrupt request whose PRI_n value in INTC_PSR0_3–INTC_PSR292_293 is higher than the PRI value in INTC_CPR_PRC0 or INTC_CPR_PRC1 negates before the interrupt request to the processor for that peripheral or software settable interrupt request is acknowledged, the interrupt request to the processor can assert or remain asserted for that peripheral or software settable interrupt request. In this case, the interrupt vector will correspond to that peripheral or software settable interrupt request. Also, the PRI value in either the INTC_CPR_PRC0 or INTC_CPR_PRC1 will be updated with the corresponding PRI_n value in INTC_PSR n_n . Furthermore, clearing the peripheral interrupt request's enable bit in the peripheral or, alternatively, setting its mask bit has the same consequences as clearing its flag bit. Setting its enable bit or clearing its mask bit while its flag bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.

9.4.1.1 Peripheral Interrupt Requests

An interrupt event in a peripheral's hardware sets a flag bit that resides in the peripheral. The interrupt request from the peripheral is driven by that flag bit.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

Interrupt requests from devices external to the MPC5510 are classified as peripheral interrupt requests in this reference manual. External interrupts are handled by the SIU (see [Section 6.4.3, "External Interrupt"](#)).

9.4.1.2 Software Settable Interrupt Requests

An interrupt request is triggered by software by writing a 1 to a SET n bit in INTC_SSCIR0_3–INTC_SSCIR4_7. This write sets the corresponding flag bit, CLR n , resulting in the interrupt request. The interrupt request is cleared by writing a 1 to the CLR n bit.

The time from the write to the SET n bit to the time that the INTC starts to drive the interrupt request to the processor is four clocks.

9.4.1.3 Unique Vector for Each Interrupt Request Source

Each peripheral and software settable interrupt request is assigned a hardwired unique 9-bit vector. Software settable interrupts 0–7 are assigned vectors 0–7 respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to include all the peripheral interrupt requests. The peripheral interrupt request input ports at the boundary of the INTC block are assigned specific hardwired vectors within the INTC (see [Section 8.3.1, “Interrupt Source Summary Table”](#)).

9.4.2 Priority Management

The asserted interrupt requests are compared to each other based on their PRI_n and PRC_SEL_n values set in $INTC_PSR0_3$ – $INTC_PSR292_293$. The result is compared to PRI in the associated $INTC_CPR_PRC0$ or $INTC_CPR_PRC1$. The results of those comparisons manage the priority of the ISR executed by the associated processor. The associated LIFO also assists in managing that priority.

9.4.2.1 Current Priority and Preemption

The priority arbitrator, selector, encoder, and comparator sub-blocks shown in [Figure 9-1](#) compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software settable interrupt request is higher than the current priority for a given processor, then the interrupt request to the processor is asserted. A unique vector for the preempting peripheral or software settable interrupt request is generated for the associated $INTC_IACKR_PRC0$ or $INTC_IACKR_PRC1$ and, if in hardware vector mode, for the interrupt vector given to the processor.

9.4.2.1.1 Priority Arbitrator Sub-block

The priority arbitrator sub-block for each processor compares all the priorities of all of the asserted interrupt requests assigned to that processor, both peripheral and software settable. The output of the priority arbitrator sub-block is the highest of those priorities assigned to a given processor. Also, any interrupt requests which have this highest priority are output as asserted interrupt requests to the associated request selector sub-block.

9.4.2.1.2 Request Selector Sub-block

If only one interrupt request from the associated priority arbitrator sub-block is asserted, then it is passed as asserted to the associated vector encoder sub-block. If multiple interrupt requests from the associated priority arbitrator sub-block are asserted, only the one with the lowest vector passes as asserted to the associated vector encoder sub-block. The lower vector is chosen regardless of the time order of the assertions of the peripheral or software settable interrupt requests.

9.4.2.1.3 Vector Encoder Sub-block

The vector encoder sub-block generates the unique 9-bit vector for the asserted interrupt request from the request selector sub-block for the associated processor.

9.4.2.1.4 Priority Comparator Sub-block

The priority comparator sub-block compares the highest priority output from the associated priority arbitrator sub-block with PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1. If the priority comparator sub-block detects that the highest priority is higher than the current priority, then it asserts the interrupt request to the associated processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1, or the PRI value in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 is lowered below this highest priority. This highest priority becomes the new priority which is written to PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI_n in INTC_PSR $_{n-n}$ are 0 will not cause a preemption because their PRI_n will not be higher than PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1.

9.4.2.2 Last-In First-Out (LIFO)

The LIFO stores the preempted PRI values from the associated INTC_CPR_PRC0 or INTC_CPR_PRC1. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 does not need to be loaded from the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 and stored onto the context stack. Likewise, at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the associated INTC_CPR_PRC0 or INTC_CPR_PRC1.

The PRI value in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 is pushed onto the LIFO when the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1 is read in software vector mode or when the interrupt acknowledge signal from the associated processor is asserted in hardware vector mode. The priority is popped into PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 when the associated INTC_EOIR_PRC0 or INTC_EOIR_PRC1 is written. An exception case in hardware vector mode to this behavior is described in [Section 1.2.3.1.2, “Hardware Vector Mode.”](#)

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR_PRC0 or INTC_CPR_PRC1 equal to 15 will not be preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority of 0 is not needed because of how pushing onto a full LIFO and popping an empty LIFO are treated. If the LIFO is pushed 15 or more times than it is popped, the first priorities pushed are overwritten. A priority of 0 would be an overwritten priority. However, the LIFO will pop 0s if it is popped more times than pushed. Therefore, although a priority of 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory mapped, even in debug mode.

9.4.3 Handshaking with Processor

9.4.3.1 Software Vector Mode Handshaking

This section describes handshaking in software vector mode.

9.4.3.1.1 Acknowledging Interrupt Request to Processor

The software vector mode handshaking can be used with processors that support an interrupt request to them only, or processors that support both an interrupt request to them only and/or an interrupt request and interrupt vector to them. The software vector mode handshaking also supports processors which always expect an interrupt vector with the interrupt request to them

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode and the handshake near the end of the interrupt exception handler, is shown in [Figure 9-13](#). The INTC examines the peripheral and software settable interrupt requests. When it finds an asserted peripheral or software settable interrupt request with a higher priority than PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1, it asserts the interrupt request to the associated processor. The INTVEC field in the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1 is updated with the preempting interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The handshaking process is described in [Section 9.1.3.1.1, "Software Vector Mode"](#).

9.4.3.1.2 End of Interrupt Exception Handler

Before the interrupt exception handling completes, INTC_SSCIR0_3–INTC_SSCIR4_7 must be written. When written, the associated LIFO is popped so the preempted priority is restored into PRI of the associated INTC_CPR_PRC0 or INTC_CPR_PRC1. Before it is written, the peripheral or software settable flag bit must be cleared so that the peripheral or software settable interrupt request is negated.

NOTE

A store to clear the peripheral or software settable interrupt flag bit that closely precedes the store to the INTC_EOIR_PRC0 or INTC_EOIR_PRC1 can result in that peripheral or software settable interrupt request being serviced again. If this scenario happens, preventative measures can be used such as executing a Power Architecture isync instruction before the store to the INTC_EOIR_PRC0 or INTC_EOIR_PRC1 as shown in [Section 9.1.3.1.1, "Software Vector Mode"](#).

When returning from the preemption, the INTC does not search for the peripheral or software settable interrupt request whose ISR was preempted. Depending on how much the ISR progressed, that interrupt request may no longer be asserted. When PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 is lowered to the priority of the preempted ISR, the interrupt request for the preempted ISR or other asserted peripherals or software settable interrupt requests at or below that priority will not cause a preemption. Instead, after the restoration of the preempted context, the processor returns to the instruction address it was to execute before it was preempted. This next instruction is part of the preempted ISR or the interrupt exception handler's prolog or epilog.

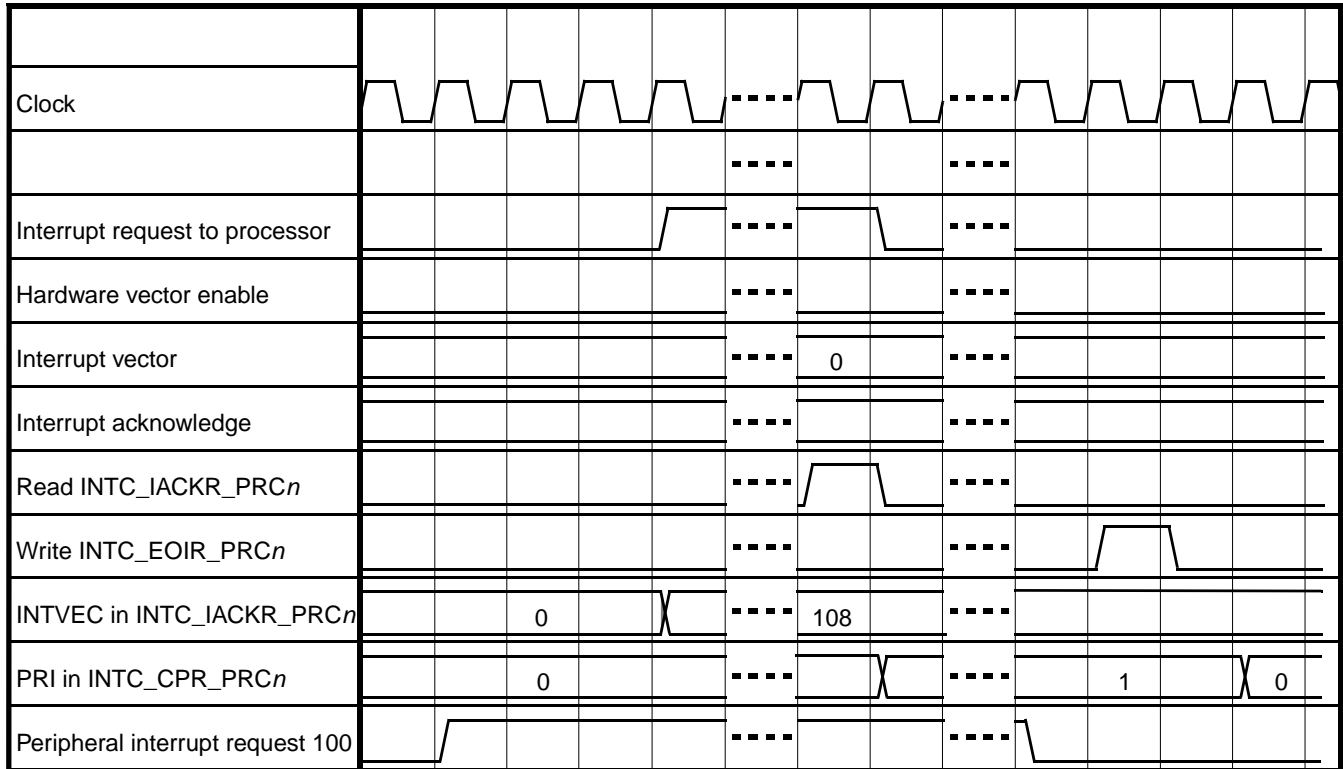


Figure 9-13. Software Vector Mode Handshaking Timing Diagram

9.4.3.2 Hardware Vector Mode Handshaking

A timing diagram of the interrupt request and acknowledge handshaking in hardware vector mode and the handshake near the end of the interrupt exception handler, is shown in [Figure 9-14](#). As in software vector mode, the INTC examines the peripheral and software settable interrupt requests and, when it finds one asserted with a higher priority than PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1, it asserts the interrupt request to the associated processor. The INTVEC field in the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1 is updated with the preempting peripheral or software settable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the associated processor is asserted. The value of the interrupt vector to the associated processor also matches the value of the INTVEC field in the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1. The handshake process is described in [Section 9.1.3.1.2, "Hardware Vector Mode."](#)

The handshaking near the end of the interrupt exception handler, that is written to the associated INTC_EOIR_PRC0 or INTC_EOIR_PRC1, is the same as in software vector mode (see [Section 9.4.3.1.2, "End of Interrupt Exception Handler"](#)).

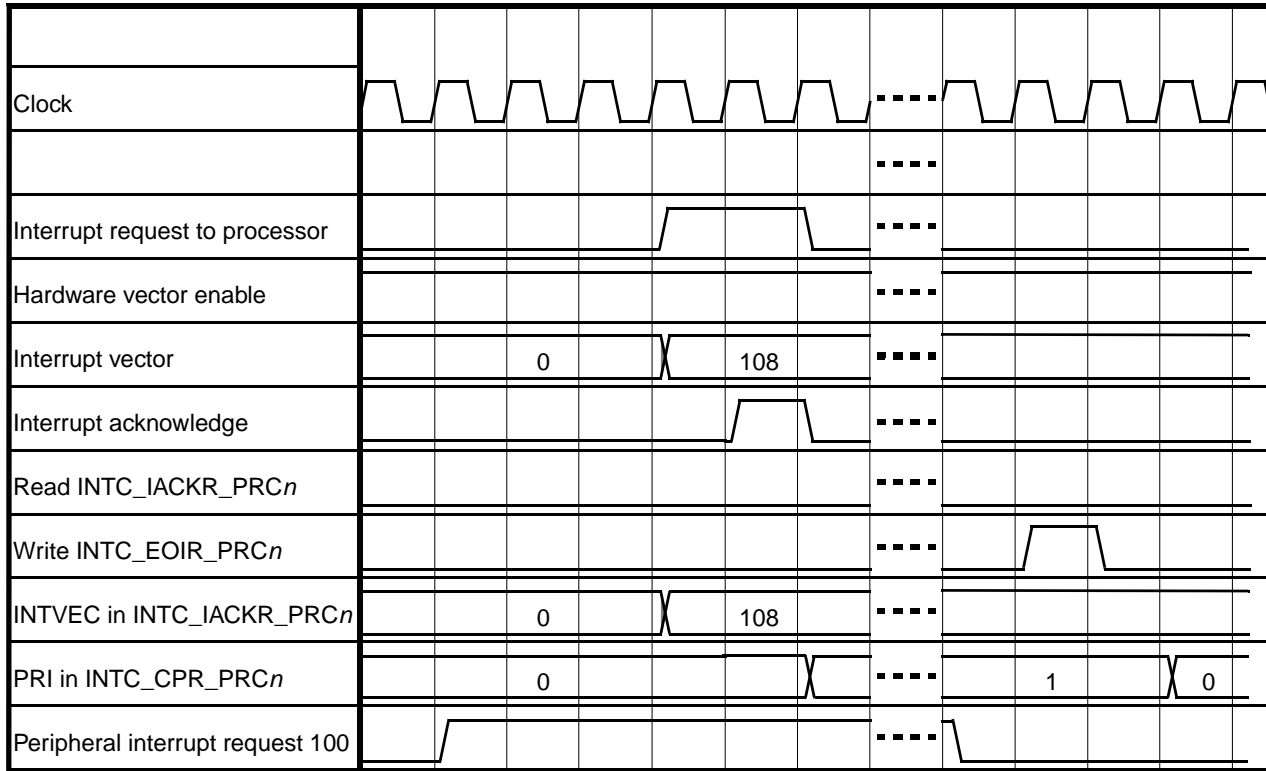


Figure 9-14. Hardware Vector Mode Handshaking Timing Diagram

9.5 Initialization/Application Information

9.5.1 Initialization Flow

After exiting reset, all of the $PRIn$ and PRC_SELn fields in $INTC_PSR0_3$ – $INTC_PSR292_293$ will be 0, and PRI in both $INTC_CPR_PRC0$ and $INTC_CPR_PRC1$ will be 15. These reset values prevent the INTC from asserting interrupt requests to the processors. Furthermore, the peripherals must have a bit to enable or mask peripheral interrupt request signals. An initialization sequence for allowing the peripheral and software settable interrupt requests to cause an interrupt request to the processor is:

```

interrupt_request_initialization:
configure VTES_PRC0,VTES_PRC1,HVEN_PRC0 and HVEN_PRC1 in INTC_MCR
configure VTBA_PRCn in INTC_IACKR_PRCn
raise the  $PRIn$  fields and set the  $PRC\_SELn$  fields to the desired processor in  $INTC\_PSRn_n$ 
set the enable bits or clear the mask bits for the peripheral interrupt requests
lower  $PRI$  in  $INTC\_CPR\_PRCn$  to zero
enable processor(s) recognition of interrupts
    
```

9.5.2 Interrupt Exception Handler

These example interrupt exception handlers use Power Architecture assembly code.

9.5.2.1 Software Vector Mode

```

interrupt_exception_handler:
code to save SRR0 and SRR1

lis      r3,hi(INTC_IACKR_PRCn)    # form INTC_IACKR_PRCn address
ori      r3,r3,lo(INTC_IACKR_PRCn)
lwz      r3,0x0(r3)                # load INTC_IACKR_PRCn, which clears request to processor
lwz      r3,0x0(r3)                # load address of ISR from vector table

code to enable processor recognition of interrupts and save context required by EABI

mtlr     r3                        # move INTC_IACKR_PRCn contents into link register
blrl     # branch to ISR; link register updated with epilog
# address

epilog:
lis      r3,hi(INTC_EOIR_PRCn)    # form INTC_EOIR_PRC0 address
ori      r3,r3,lo(INTC_EOIR_PRCn)
li       r4,0x0                    # form 0 to write to INTC_EOIR_PRCn
stw      r4,0x0(r3)               # store to INTC_EOIR_PRCn, informing INTC to lower priority

code to restore context required by EABI and disable processor recognition of interrupts
code to restore SRR0 and SRR1

rfi

vector_table_base_address:
address of ISR for interrupt with vector 0
address of ISR for interrupt with vector 1
.
.
.
address of ISR for interrupt with vector 510
address of ISR for interrupt with vector 511

ISRn:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                                     # return to epilog

```

9.5.2.2 Hardware Vector Mode

This interrupt exception handler is useful with processor and system bus implementations which support a hardware vector. This example assumes that each `interrupt_exception_handler_n` only has space for four instructions, and therefore a branch to `interrupt_exception_handler_continuedn` is needed.

```

interrupt_exception_handler_n:
b        interrupt_exception_handler_continuedn# 4 instructions available, branch to continue

interrupt_exception_handler_continuedn:
code to save SRR0 and SRR1
code to enable processor recognition of interrupts and save context required by EABI

```

Interrupt Controller (INTC)

```
bl      ISRn                # branch to ISR for interrupt with vector x

epilog:
lis     r3,hi(INTC_EOIR_PRCn) # form INTC_EOIR_PRCn address
ori     r3,r3,lo(INTC_EOIR_PRCn)
li      r4,0x0              # form 0 to write to INTC_EOIR_PRCn
stw     r4,0x0(r3)         # store to INTC_EOIR_PRCn, informing INTC to lower priority

code to restore context required by EABI and disable processor recognition of interrupts
code to restore SRR0 and SRR1

rfi

ISRn:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                    # branch to epilog
```

9.5.3 ISR, RTOS, and Task Hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC_CPR_PRC0 or INTC_CPR_PRC1 having a value of 0. The RTOS executes the tasks according to whatever priority scheme it may have, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC_CPR_PRCn priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR_PRCn priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR_PRCn priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task's priority can be elevated in the INTC_CPR_PRCn while the shared resource is being accessed.

An ISR whose PRIn in INTC_PSR0_3–INTC_PSR292_293 has a value of 0 will not cause an interrupt request to the selected processor, even if its peripheral or software settable interrupt request is asserted. For a peripheral interrupt request, not setting its enable bit or disabling the mask bit causes it to remain negated, which consequently does not cause an interrupt request to the processor. Because the ISRs are outside the control of the RTOS, this ISR will not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

9.5.4 Order of Execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software settable interrupt requests. However, if multiple peripheral or software settable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, the INTC selects the one with the lowest unique vector regardless of the order in time that they asserted. However, the ability to meet deadlines with this scheduling scheme is no less than if the ISRs execute in the time order that their peripheral or software settable interrupt requests asserted.

The example in [Table 9-12](#) shows the order of execution of both ISRs with different priorities and the same priority.

Table 9-12. Order of ISR Execution Example

Step#	Step Description	Code Executing at End of Step					Interrupt Exception Handler	PRI in INTC_CPR at End of Step
		RTOS	ISR108 ¹	ISR208	ISR308	ISR408		
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 is asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 is asserts.					X		4
5	Peripheral interrupt request 200 at priority 3 is asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR_PRC _n .						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR_PRC _n .						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR_PRC _n .						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR_PRC _n .						X	0
12	RTOS continues execution.	X						0

¹ ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software settable interrupt requests.

9.5.5 Priority Ceiling Protocol

9.5.5.1 Elevating Priority

The PRI field in INTC_CPR_PRC0 or INTC_CPR_PRC1 is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol allows coherent accesses of the ISRs to that shared resource.

For example, ISR1 has a priority of 1, ISR2 has a priority of 2, and ISR3 has a priority of 3. They share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC_CPR_PRC_n to 3, the ceiling of all of the ISR priorities. After they release the resource, the PRI

value in INTC_CPR_PRC n can be lowered. If they do not raise their priority, ISR2 can preempt ISR1, and ISR3 can preempt ISR1 or ISR2, possibly corrupting the shared resource. Another possible failure mechanism is deadlock if the higher priority ISR needs the lower priority ISR to release the resource before it can continue, but the lower priority ISR cannot release the resource until the higher priority ISR completes and execution returns to the lower priority ISR.

Using the PCP instead of disabling processor recognition of all interrupts eliminates the time when accessing a shared resource that all higher priority interrupts are blocked. For example, while ISR3 cannot preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

9.5.5.2 Ensuring Coherency

A scenario can cause non-coherent accesses to the shared resource. For example, ISR1 and ISR2 are both running on the same core and both share a resource. ISR1 has a lower priority than ISR2. ISR1 is executing and writes to the INTC_CPR_PRC n . The instruction following this store is a store to a value in a shared coherent data block. Either immediately before or at the same time as the first store, the INTC asserts the interrupt request to the processor because the peripheral interrupt request for ISR2 has asserted. As the processor is responding to the interrupt request from the INTC, and as it is aborting transactions and flushing its pipeline, it is possible that both stores will be executed. ISR2 thereby thinks that it can access the data block coherently, but the data block has been corrupted.

OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent corruption of a coherent data block, modifications to PRI in INTC_CPR_PRC n can be made by those system services with the following code sequences. Processor recognition of interrupts must be enabled before executing the GetResource code sequence.

```
GetResource:
    raise PRI
    mbar
    isync

ReleaseResource:
    mbar
    lower PRI
```

9.5.6 Selecting Priorities According to Request Rates and Deadlines

The selection of the priorities for the ISRs can be made using rate monotonic scheduling (RMS) or a superset of it, deadline monotonic scheduling (DMS). In RMS, the ISRs which have higher request rates have higher priorities. In DMS, if the deadline is before the next time the ISR is requested, then the ISR is assigned a priority according to the time from the request for the ISR to the deadline, not from the time of the request for the ISR to the next request for it.

For example, ISR1 executes every 100 μ s, ISR2 executes every 200 μ s, and ISR3 executes every 300 μ s. ISR1 has a higher priority than ISR2 which has a higher priority than ISR3; however, if ISR3 has a deadline of 150 μ s, then it has a higher priority than ISR2.

The INTC has 16 priorities, which may be less than the number of ISRs. In this case, the ISRs should be grouped with other ISRs that have similar deadlines. For example, a priority could be allocated for every

time the request rate doubles. ISRs with request rates around 1 ms would share a priority, ISRs with request rates around 500 μ s would share a priority, ISRs with request rates around 250 μ s would share a priority, etc. With this approach, a range of ISR request rates of 2^{16} could be included, regardless of the number of ISRs.

Reducing the number of priorities reduces the processor's ability to meet its deadlines. However, reducing the number of priorities can reduce the size and latency through the interrupt controller. It also allows easier management of ISRs with similar deadlines that share a resource. They do not need to use the PCP to access the shared resource.

9.5.7 Software Settable Interrupt Requests

The software settable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and they may also be used by processors to interrupt other processors in a multiple processor system.

9.5.7.1 Scheduling a Lower Priority Portion of an ISR

A portion of an ISR needs to be executed at the PRI_n value in INTC_PSR0_3–INTC_PSR292_293, which becomes the PRI value in either INTC_CPR_PRC0 or INTC_CPR_PRC1 with the interrupt acknowledge. The ISR, however, can have a portion that does not need to be executed at this higher priority. Therefore, executing the later portion that does not need to be executed at this higher priority can prevent the execution of ISRs which do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This preemptive scheduling inefficiency reduces the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a SET n bit in INTC_SSCIR0_3–INTC_SSCIR4_7." Writing a 1 to SET n causes a software settable interrupt request. This software settable interrupt request will usually have a lower PRI_n value in the INTC_PSR n_n and will not cause preemptive scheduling inefficiencies. After generating a software settable interrupt request, the higher priority ISR completes. The lower priority ISR is scheduled according to its priority. Execution of the higher priority ISR is not resumed after the completion of the lower priority ISR.

9.5.7.2 Scheduling an ISR on Another Processor

Because the SET n bits in the INTC_SSCIR n_n are memory mapped, processors in multiple-processor systems can schedule ISRs on the other processors. One application is that one processor wants to command another processor to perform a piece of work and the initiating processor does not need to use the results of that work. If the initiating processor is concerned that the processor executing the software settable ISR has not completed the work before asking it to again execute the ISR, it can check if the corresponding CLR n bit in INTC_SSCIR n_n is asserted before again writing a 1 to the SET n bit.

Another application is the sharing of a block of data. For example, a first processor has completed accessing a block of data and wants a second processor to then access it. Furthermore, after the second

processor has completed accessing the block of data, the first processor again wants to access it. The accesses to the block of data must be done coherently. To do this, the first processor writes a 1 to a SETn bit on the second processor. After accessing the block of data, the second processor clears the corresponding CLRn bit and then writes 1 to a SETn bit on the first processor, informing it that it can now access the block of data.

9.5.8 Lowering Priority Within an ISR

A common method for avoiding preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities (see [Section 9.5.7.1, “Scheduling a Lower Priority Portion of an ISR”](#)) is to lower the current priority. However, the INTC has a LIFO whose depth is determined by the number of priorities.

NOTE

Lowering the PRI value in either INTC_CPR_PRC0 or INTC_CPR_PRC1 within an ISR to below the ISR’s corresponding PRI value in INTC_PSR0_3–INTC_PSR292_293 allows more preemptions than the LIFO depth can support.

Therefore, the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.

9.5.9 Negating an Interrupt Request Outside of its ISR

9.5.9.1 Negating an Interrupt Request as a Side Effect of an ISR

Some peripherals have flag bits that can be cleared as a side effect of servicing a peripheral interrupt request. For example, reading a specific register can clear the flag bits and their corresponding interrupt requests. This clearing as a side effect of servicing a peripheral interrupt request can cause the negation of other peripheral interrupt requests besides the peripheral interrupt request whose ISR presently is executing. This negating of a peripheral interrupt request outside of its ISR can be a desired effect.

9.5.9.2 Negating Multiple Interrupt Requests in One ISR

An ISR can clear other flag bits besides its own. One reason that an ISR clears multiple flag bits is because it serviced those flag bits, and therefore the ISRs for these flag bits do not need to be executed.

9.5.9.3 Proper Setting of Interrupt Request Priority

Whether an interrupt request negates outside its own ISR due to the side effect of an ISR execution or the intentional clearing a flag bit, the priorities of the peripheral or software settable interrupt requests for these other flag bits must be selected properly. Their PRI_n values in INTC_PSR0_3–INTC_PSR292_293 must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits can cause the interrupt request to the processor to assert. Furthermore, the clearing of these other flag bits also has the same timing relationship to the writing to INTC_SSCIR0_3–INTC_SSCIR4_7 as the clearing of the flag bit that caused the present ISR to be executed (see [Section 9.4.3.1.2, “End of Interrupt Exception Handler”](#)).

A flag bit whose enable bit or mask bit negates its peripheral interrupt request can be cleared at any time, regardless of the peripheral interrupt request's PRI_n value in $INTC_PSR_n$.

9.5.10 Examining LIFO contents

In normal mode, the user does not need to know the contents of the LIFO. He may not even know how deeply the LIFO is nested. However, if he wants to read the contents, such as in debug mode, they are not memory mapped. The contents can be read by popping the LIFO and reading the PRI field in either $INTC_CPR_PRC0$ or $INTC_CPR_PRC1$. The code sequence is:

```
pop_lifo:
store to INTC_EOIR_PRCn
load INTC_CPR_PRCn, examine PRI, and store onto stack
if PRI is not zero or value when interrupts were enabled, branch to pop_lifo
```

When the examination is complete, the LIFO can be restored using this code sequence:

```
push_lifo:
load stacked PRI value and store to INTC_CPR_PRCn
load INTC_IACKR_PRCn
if stacked PRI values are not depleted, branch to push_lifo
```


Chapter 10

e200z1 Core (Z1)

10.1 Introduction

The e200 processor family is a set of CPU cores that implement low-cost versions of the Power Architecture Book E architecture. e200 processors are designed for deeply embedded control applications that require low cost solutions rather than maximum performance.

The e200z1 processors integrate an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable of sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by the branch unit to allow single-cycle branches in some cases.

The e200z1 core is a single-issue, 32-bit Power Architecture Book E compliant design with 32-bit general purpose registers (GPRs). Power Architecture Book E floating-point instructions are not supported by e200 in hardware, but are trapped and may be emulated by software. All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs).

In addition to the base Power Architecture Book E instruction set support, the core also implements the VLE (variable-length encoding) APU, providing improved code density. The VLE APU is further documented in “PowerPC VLE APU Definition, Version 1.01”, a separate document.

In the remainder of this document, the e200z1 core is also referred to as the ‘e200z1 core’ or ‘e200 core’.

10.1.1 Features

The following is a list of some of the key features of the e200z1 core:

- 32-bit Power Architecture Book E programmer’s model
- Single issue, 32-bit CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
 - Dedicated branch address calculation adder
 - Branch acceleration using Branch Target Buffer
- Supports independent instruction and data accesses to different memory subsystems, such as SRAM and Flash memory via independent Instruction and Data BIUs.
- Load/store unit
 - 1 cycle load latency

- Fully pipelined
- Big-and Little-endian support
- Misaligned access support
- Zero load-to-use pipeline bubbles for aligned transfers
- Power management
 - Low power design
 - Power saving modes: doze, nap, sleep, and wait
 - Dynamic power management of execution units

NOTE

The MPC5510 does not use the core's `HID0[DOZE,NAP,SLEEP]` bits to enter/exit low-power modes. Entry to and exit from low-power modes is managed by the CRP module.

10.2 Microarchitecture Summary

The execution pipeline four stages operate in an overlapped fashion, allowing single-clock instruction execution for most instructions. These stages are as follows:

1. The instruction fetch
2. Instruction decode/register file read/effective address calculation
3. Execute/memory access
4. Register writeback

The integer execution unit consists of a 32-bit arithmetic unit (AU), a logic unit (LU), a 32-bit barrel shifter (Shifter), a mask insertion unit (MIU), a condition register manipulation unit (CRU), a count-leading-zeros unit (CLZ), a 32x32 hardware multiplier array, result feed-forward hardware, and a hardware divider.

Arithmetic and logical operations are executed in a single cycle with the exception of the divide instructions. A count-leading-zeros unit operates in a single clock cycle.

The instruction unit contains a PC incremter and a dedicated branch address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching from the BTB is performed to accelerate certain taken branches. Prefetched instructions are placed into an instruction buffer with 62 entries, each capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Conditional branches which are not taken execute in a single clock. Branches with successful target prefetching have an effective execution time of one clock. All other taken branches have an execution time of two clocks.

Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The condition register unit supports the condition register (CR) and condition register operations defined by the Power Architecture. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

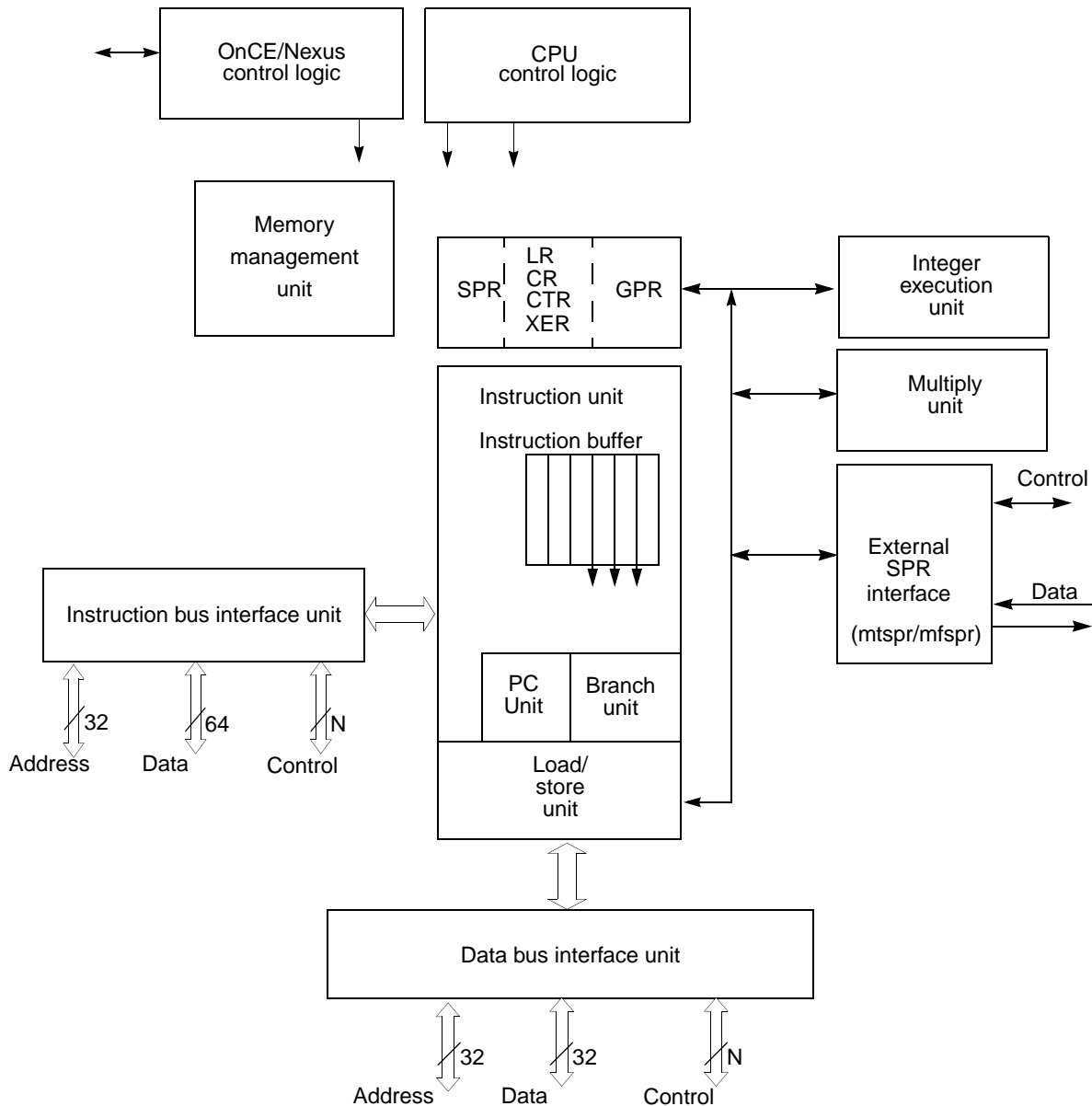


Figure 10-1. e200z1 Block Diagram

10.2.1 Instruction Unit Features

The features of the e200 instruction unit are:

- 32-bit instruction fetch path supports fetching of one 32-bit instruction per clock, or up to two 16-bit VLE instructions per clock.
- Instruction buffer with four entries, each holding a single 32-bit instruction, or a pair of 16-bit instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch unit with dedicated branch address adder, and small branch target buffer logic supporting single cycle of execution of certain branches, two cycles for all others

10.2.2 Integer Unit Features

The e200 integer unit supports single cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count leading zero's function
- 32-bit single cycle barrel shifter for shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in 6-16 clocks with minimized execution timing
- 32x32 hardware multiplier array supports single-cycle 32x32->32 multiply

10.2.3 Load/Store Unit Features

The e200 load/store unit supports load, store, and the load multiple / store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- 32-bit interface to memory (dedicated memory interface on e200z1)

10.2.4 e200z1 System Bus Features

The features of the e200z1 system bus interface are as follows:

- Independent instruction and data buses
- 32-bit address bus plus attributes and control on each bus
- 32-bit read data bus for instruction interface
- Separate uni-directional 32-bit read data bus and 32-bit write data bus for data interface
- Overlapped, in-order accesses

10.2.5 MMU Features

The features of the MMU are as follows:

- Virtual memory support
- 32-bit virtual and physical addresses
- 8-bit process identifier

- 8-entry fully-associative TLB
- Support for multiple page sizes from 4 Kbyte to 4 Gbyte
- Entry flush protection

NOTE

The maximum system frequency is only supported if the MMU control and status register 0 bypass bit (MMUCSR0[Bypass]) is set to 1, so that address translation is not performed. If the MMUCSR0[Bypass] bit is 0, then the maximum system frequency will be less than the maximum frequency listed in the *MPC5510 Microcontroller Family Data Sheet*.

10.3 Core Registers and Programmer's Model

This section describes the registers implemented in the e200z1 core. It includes an overview of registers defined by the Power Architecture Book E architecture, highlighting differences in how these registers are implemented in the e200 core, and provides a detailed description of e200-specific registers. Full descriptions of the architecture-defined register set are provided in Power Architecture Book E Specification.

The Power Architecture Book E defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions. Data is transferred between memory and registers with explicit load and store instructions only.

[Figure 10-2](#), [Figure 10-3](#), and [Figure 10-4](#) show the e200 register set including the registers which are accessible while in supervisor mode, and the registers which are accessible in user mode. The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register (for example, the integer exception register (XER) is SPR 1).

NOTE

e200z1 is a 32-bit implementation of the Power Architecture Book E specification. In this document, register bits are sometimes numbered from bit 0 (most significant bit) to 31 (least significant bit), rather than the Book E numbering scheme of 32:63, thus register bit numbers for some registers in Book E are 32 higher.

Where appropriate, the Book E defined bit numbers are shown in parentheses.

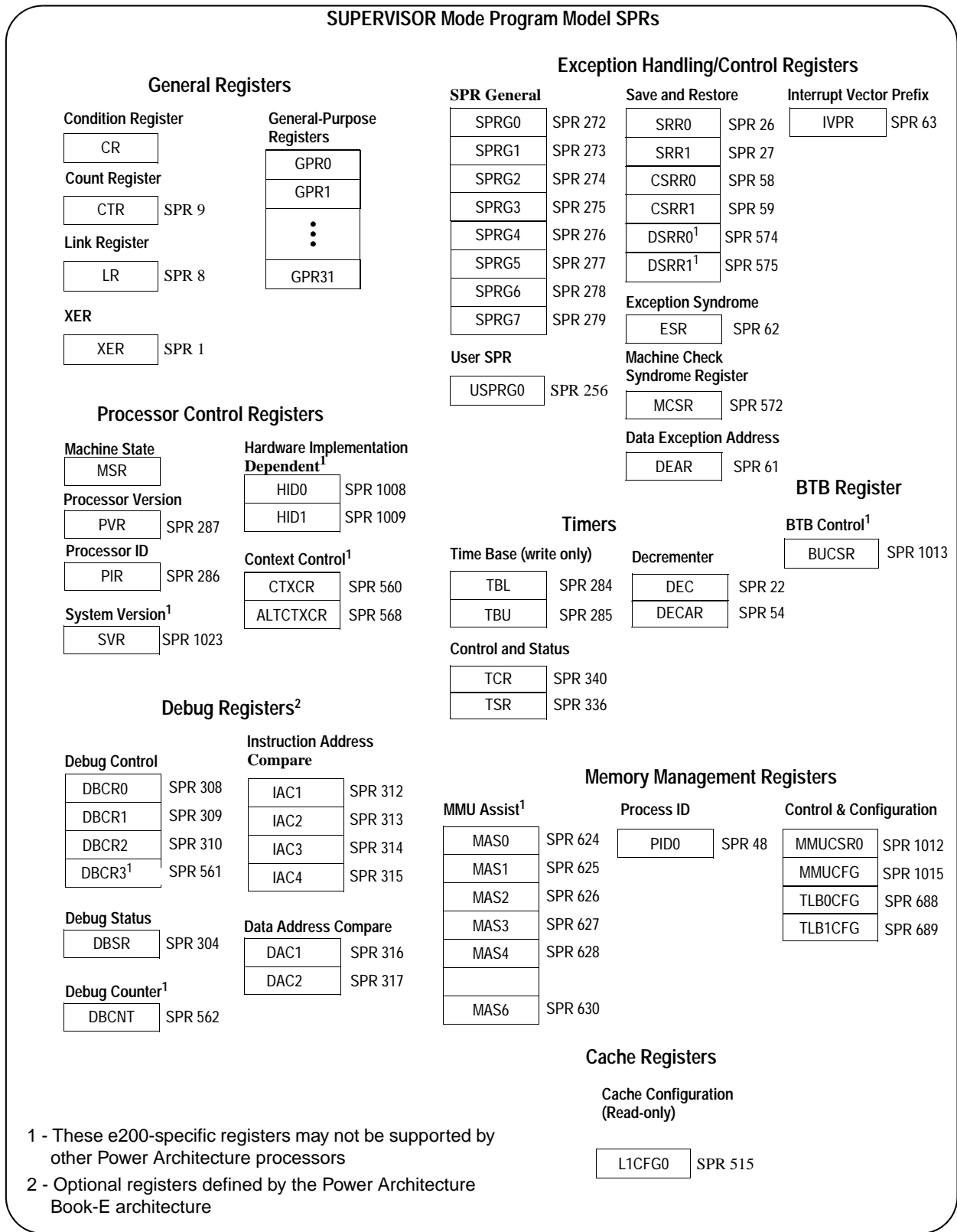


Figure 10-2. e200z1 Supervisor Mode Programmer's Model SPRs

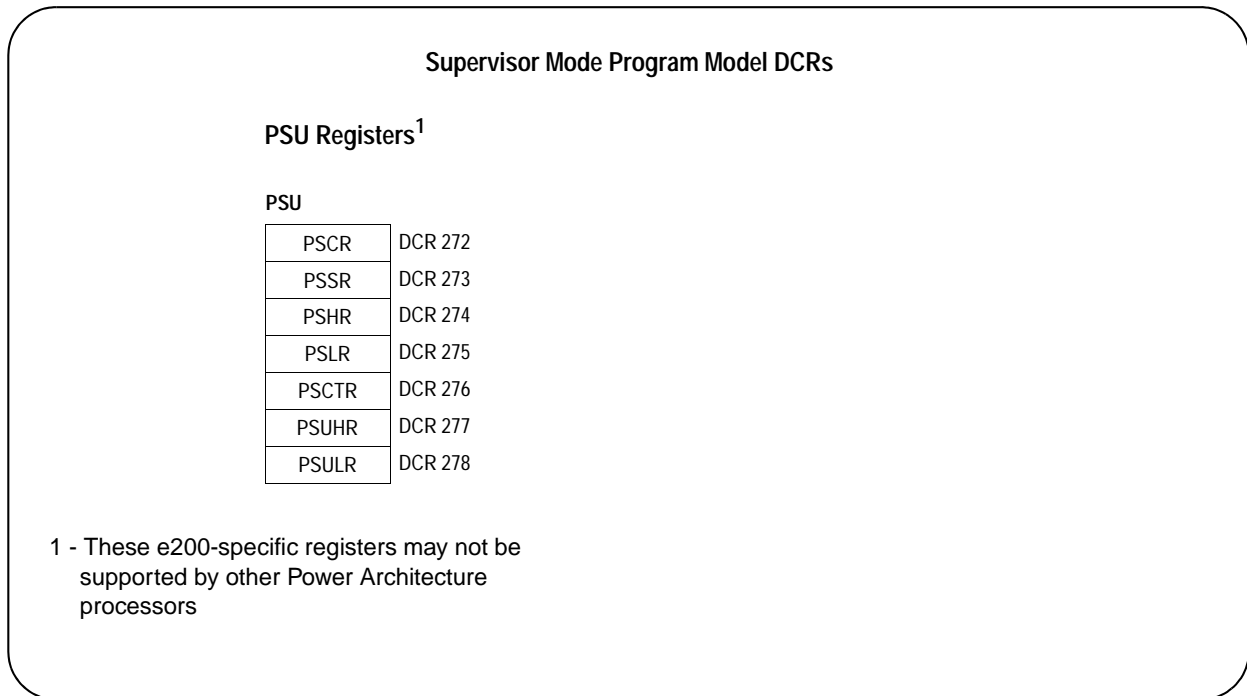


Figure 10-3. e200 Supervisor Mode Program Model Device Control Registers (DCRs)

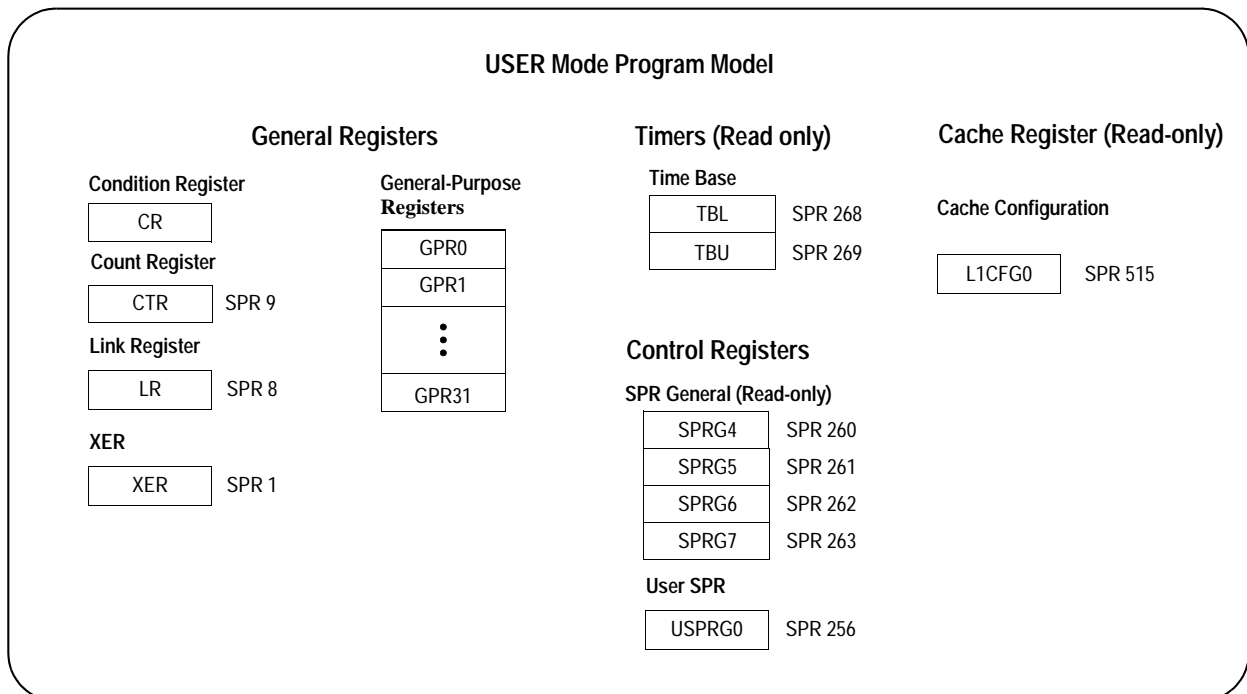


Figure 10-4. e200 User Mode Program Model

General purpose registers (GPRs) are accessed through instruction operands. Access to other registers can be explicit (by using instructions for that purpose such as Move to Special Purpose Register (**mtspr**) and

Move from Special Purpose Register (**mf spr**) instructions) or implicit as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

10.3.1 Power Architecture Book E Registers

e200 supports most of the registers defined by *Power Architecture™ Book E Specification*. Notable exceptions are the Floating Point registers FPR0-FPR31 and FPSCR. e200 does not support the Book E floating-point architecture in hardware. The e200-supported Power Architecture Book E registers are described as follows (e200-specific registers are described in the [Section 10.3.2, “e200-Specific Special Purpose Registers](#)):

10.3.1.1 User-Level Registers

The user-level registers can be accessed by all software with either user or supervisor privileges. They include the following:

- General-purpose registers (GPRs). The thirty-two 32-bit GPRs (GPR0–GPR31) serve as data source or destination registers for integer instructions and provide data for generating addresses.
- Condition register (CR). The 32-bit CR consists of eight 4-bit fields, CR0–CR7, that reflect results of certain arithmetic operations and provide a mechanism for testing and branching. See “Condition Register (CR),” in Chapter 3, “Branch and Condition Register Operations, Power Architecture Book E Specification.

The remaining user-level registers are SPRs. Note that the Power Architecture Book E provides the **mt spr** and **mf spr** instructions for accessing SPRs.

Integer exception register (XER). The XER indicates overflow and carries for integer operations. See “XER Register (XER),” in Chapter 4, “Integer Operations” of *Power Architecture Book E Specification* for more information.

- Link register (LR). The LR provides the branch target address for the Branch [Conditional] to Link Register (**blr**, **blrl**, **se_blr**, **se_blrl**) instructions, and is used to hold the address of the instruction that follows a branch and link instruction, typically used for linking to subroutines. See “Link Register (LR),” in Chapter 3, “Branch and Condition Register Operations” of *Power Architecture Book E Specification*.
- Count register (CTR). The CTR holds a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR also provides the branch target address for the Branch [Conditional] to Count Register (**bctr**, **bctrl**, **se_bctr**, **se_bctrl**) instructions. See “Count Register (CTR),” in Chapter 3, “Branch and Condition Register Operations” of *Power Architecture Book E Specification*.
- The Time Base facility (TB) consists of two 32-bit registers—Time Base Upper (TBU) and Time Base Lower (TBL). These two registers are accessible in a read-only fashion to user-level software. See “Time Base”, in Chapter 8, “Timer Facilities” of *Power Architecture Book E Specification*.
- SPRG4-SPRG7. The Power Architecture Book E architecture defines Software-Use Special Purpose Registers (SPRGs). SPRG4 through SPRG7 are accessible in a read-only fashion by user-level software. e200 does not allow user mode access to the SPRG3 register (defined as implementation dependent by Book E).

- USPRG0. The Power Architecture Book E architecture defines User Software-Use Special Purpose Register USPRG0 which is accessible in a read-write fashion by user-level software.

10.3.1.2 Supervisor-Level Registers

In addition to the registers accessible in user mode, Supervisor-level software has access to additional control and status registers used for configuration, exception handling, and other operating system functions. The Power Architecture Book E defines the following supervisor-level registers:

- Processor Control Registers
 - Machine State Register (MSR). The MSR defines the state of the processor. The MSR can be modified by the Move to Machine State Register (**mtmsr**), System Call (**sc**, **se_sc**), and Return from Exception (**rfi**, **rfdi**, **se_rfi**, **se_rfdi**) instructions. It can be read by the Move from Machine State Register (**mfmsr**) instruction. When an interrupt occurs, the contents of the MSR are saved to one of the machine state save/restore registers (SRR1, CSRR1, DSRR1).
 - Processor version register (PVR). This register is a read-only register that identifies the processor type and version (model) and the revision level of the processor. [Table 10-1](#) shows the PVR values and the corresponding processor type and version numbers for the cores used on the MPC5510 Family.

Table 10-1. PVR Values, and Processor Type and Version Numbers

Device	Core	PVR Value	Type	Version
MPC5516	e200Z1	0x8144_0000	0x14	0x4
MPC5516	e200Z0	0x8171_0000	0x17	0x1

- Processor Identification Register (PIR). This read-only register is provided to distinguish the processor from other processors in the system.
- Storage Control Register
 - Process ID Register (PID, also referred to as PID0). This register is provided to indicate the current process or task identifier. It is used by the MMU as an extension to the effective address, and by the Nexus3 module for Ownership Trace message generation. Although the Power Architecture Book E allows for multiple PIDs, e200z1 implements only one.
- Interrupt Registers
 - Data Exception Address Register (DEAR). After most Data Storage Interrupts (DSI), or on an Alignment Interrupt or Data TLB Miss Interrupt, the DEAR is set to the effective address (EA) generated by the faulting instruction.
 - SPRG0–SPRG7, USPRG0. The SPRG0–SPRG7 and USPRG0 registers are provided for operating system use. e200 does not allow user mode access to the SPRG3 register (defined as implementation dependent by Book E).
 - Exception Syndrome Register (ESR). The ESR register provides a syndrome to differentiate between the different kinds of exceptions that can generate the same interrupt.
 - Interrupt Vector Prefix Register (IVPR). This register together with hardwired offsets which replace the IVOR0-15 registers provide the address of the interrupt handler for different classes of interrupts.

- Save/Restore Register 0 (SRR0). The SRR0 register is used to save machine state on a non-critical interrupt, and contains the address of the instruction at which execution resumes when an **rfi** or **se_rfi** instruction is executed at the end of a non-critical class interrupt handler routine.
- Critical Save/Restore register 0 (CSRR0). The CSRR0 register is used to save machine state on a critical interrupt, and contains the address of the instruction at which execution resumes when an **rfdi** or **se_rfdi** instruction is executed at the end of a critical class interrupt handler routine.
- Save/Restore register 1 (SRR1). The SRR1 register is used to save machine state from the MSR on non-critical interrupts, and to restore machine state when **rfi** or **se_rfi** executes.
- Critical Save/Restore register 1 (CSRR1). The CSRR1 register is used to save machine state from the MSR on critical interrupts, and to restore machine state when **rfdi** or **se_rfdi** executes.
- Debug Facility Registers
 - Debug Control Registers (DBCR0-DBCR2). These registers provide control for enabling and configuring debug events.
 - Debug Status Register (DBSR). This register contains debug event status.
 - Instruction Address Compare registers (IAC1-IAC4). These registers contain addresses and/or masks which are used to specify Instruction Address Compare debug events.
 - Data address compare registers (DAC1-2). These registers contain addresses and/or masks which are used to specify Data Address Compare debug events.
 - e200 does **not** implement the Data Value Compare registers (DVC1 and DVC2).
- Timer Registers
 - Time base (TB). The TB is a 64-bit structure provided for maintaining the time of day and operating interval timers. The TB consists of two 32-bit registers, Time Base Upper (TBU) and Time Base Lower (TBL). The Time Base registers can be written to only by supervisor-level software, but can be read by both user and supervisor-level software.
 - Decrementer register (DEC). This register is a 32-bit decrementing counter that provides a mechanism for causing a decrementer exception after a programmable delay.
 - Decrementer Auto-Reload (DECAR). This register is provided to support the auto-reload feature of the Decrementer.
 - Timer Control Register (TCR). This register controls Decrementer, Fixed-Interval Timer, and Watchdog Timer options.
 - Timer Status Register (TSR). This register contains status on timer events and the most recent Watchdog Timer-initiated processor reset.

10.3.2 e200-Specific Special Purpose Registers

The Power Architecture Book E architecture allows implementation-specific special purpose registers. Those incorporated in the e200 core are as follows:

10.3.2.1 User-Level Registers

The user-level registers can be accessed by all software with either user or supervisor privileges. They include the following:

- The L1 Cache Configuration register (L1CFG0). This read-only register allows software to query the configuration of the L1 Cache. For the e200z1, this register returns all zeros indicating no cache is present.

10.3.2.2 Supervisor-level registers

The following supervisor-level registers are defined in e200 in addition to the Power Architecture Book E registers described above:

- Configuration Registers
 - Hardware implementation-dependent register 0 (HID0). This register controls various processor and system functions.
 - Hardware implementation-dependent register 1 (HID1). This register controls various processor and system functions.
- Exception Handling and Control Registers
 - Machine Check Syndrome register (MCSR). This register provides a syndrome to differentiate between the different kinds of conditions which can generate a Machine Check.
 - Debug Save/Restore register 0 (DSRR0). When enabled, the DSRR0 register is used to save the address of the instruction at which execution continues when **rfdi** or **se_rfdi** executes at the end of a debug interrupt handler routine.
 - Debug Save/Restore register 1 (DSRR1). When enabled, the DSRR1 register is used to save machine status on debug interrupts and to restore machine status when **rfdi** or **se_rfdi** executes.
- Debug Facility Registers
 - Debug Control Register 3 (DBCR3)—This register provides control for debug functions not described in Power Architecture Book E architecture.
 - Debug Counter Register (DBCNT)—This register provides counter capability for debug functions.
- L1 Cache Configuration Register (L1CFG0) is a read-only register that allows software to query the configuration of the L1 Cache. For the e200z1, this register returns all zeros.
- MMU Configuration Register (MMUCFG) is a read-only register that allows software to query the configuration of the MMU.
- Memory Management Registers
 - MMU Assist (MAS0-MAS4, MAS6) registers. These registers provide the interface to the e200 core from the Memory Management Unit.

- MMU Control and Status Register (MMUCSR0) controls invalidation of the MMU.
- TLB Configuration Registers (TLB0CFG, TLB1CFG) are read-only registers that allow software to query the configuration of the TLBs.
- System version register (SVR). This register is a read-only register that identifies the version (model) and revision level of the SoC which includes an e200 Power Architecture processor.

Note that it is not guaranteed that the implementation of e200 core-specific registers is consistent among Power Architecture processors, although other processors may implement similar or identical registers. All e200 SPR definitions are compliant with the Freescale EIS specification definitions.

10.3.3 e200z1 Core Complex Features Not Supported on the MPC5510

The MPC5510 implements a subset of the e200z1 core complex features. The e200z1 core complex features that are not supported in the MPC5510 are described in [Table 10-2](#).

Table 10-2. e200z1 Features Not Supported on the MPC5510 Family

Description	Function/Category
The less significant halfword of the Processor Version Register (PVR) provides the revision level which is comprised of the following three bit fields: Reserved = 0x00 Revision = 0x0 ID = 0x0 The more significant halfword of the Processor Version Register (PVR) provides the processor type and version number (see Table 10-1).	PVR Value
Nexus registers are not accessible by code running in User or Supervisor mode. Nexus registers can be accessed only by external tools via the Nexus port.	Debug

10.4 e200z1 Memory Management Unit

The e200z1 Memory Management Unit is a 32-bit Power Architecture Book E compliant implementation, with the following feature set:

- EIS MMU architecture compliant
- Translates from 32-bit effective to 32-bit real addresses
- 8-entry fully associative TLB with support for eleven page sizes (4K, 16K, 64K, 256K, 1M, 4M, 16M, 64M, 256M, 1G, 4G)
- Hardware assist for TLB miss exceptions
- Software managed by **tlbre**, **tlbwe**, **tlbsx**, **tlbsync**, and **tlbivax** instructions

10.4.1 Effective to Real Address Translation

10.4.1.1 Effective Addresses

Instruction accesses are generated by sequential instruction fetches or due to a change in program flow (branches and interrupts). Data accesses are generated by load, store, and cache management instructions. The e200 instruction fetch, branch, and load/store units generate 32-bit effective addresses. The MMU translates this effective address to a 32-bit real address which is then used for memory accesses.

The Power Architecture Book E architecture divides the effective (virtual) and real (physical) address space into pages. The page represents the granularity of effective address translation, permission control, and memory/cache attributes. The MMU supports eleven page sizes (4 KB, 16 KB, 64 KB, 256 KB, 1 MB, 4 MB, 16 MB, 64 MB, 256 MB, 1GB, 4GB). In order for an effective to real address translation to exist, a valid entry for the page containing the effective address must be in a Translation Lookaside Buffer (TLB). Addresses for which no TLB entry exists (a TLB miss) cause Instruction or Data TLB Errors.

10.4.1.2 Address Spaces

Instruction accesses are generated by sequential instruction fetches or due to a change in program flow (branches and interrupts). Data accesses are generated by load, store, and cache management instructions.

The Power Architecture Book E architecture defines two effective address spaces for instruction accesses and two effective address spaces for data accesses. The current effective address space for instruction or data accesses is determined by the value of MSR[IS] and MSR[DS], respectively. The address space indicator (the value of either MSR[IS] or MSR[DS], as appropriate) is used in addition to the effective address generated by the processor for translation into a physical address by the TLB mechanism. Because MSR[IS] and MSR[DS] are both cleared to '0' when an interrupt occurs, an address space value of 0b0 can be used to denote interrupt-related address spaces (or possibly all system software address spaces), and an address space value of 0b1 can be used to denote non interrupt-related (or possibly all user address spaces) address spaces.

The address space associated with an instruction or data access is included as part of the virtual address in the translation process (AS).

10.4.1.3 Process ID

The Power Architecture Book E architecture defines that a process ID (PID) value is associated with each effective address (instruction or data) generated by the processor. At the Book E level, a single PID register is defined as a 32-bit register, and it maintains the value of the PID for the current process. This PID value is included as part of the virtual address in the translation process (PID0). For the e200 MMU, the PID is 8 bits in length. The most-significant 24 bits are unimplemented and read as '0'.

10.4.1.4 Translation Flow

The effective address, concatenated with the address space value of the corresponding MSR bit (MSR[IS] or MSR[DS]), is compared to the appropriate number of bits of the EPN field (depending on the page size) and the TS field of TLB entries. If the contents of the effective address plus the address space bit matches the EPN field and TS bit of the TLB entry, that TLB entry is a candidate for a possible translation match. In addition to a match in the EPN field and TS, a matching TLB entry must match with the current Process ID of the access (in PID0), or have a TID value of '0', indicating the entry is globally shared among all processes.

[Figure 10-5](#) shows the translation match logic for the effective address plus its attributes, collectively called the virtual address, and how it is compared with the corresponding fields in the TLB entries.

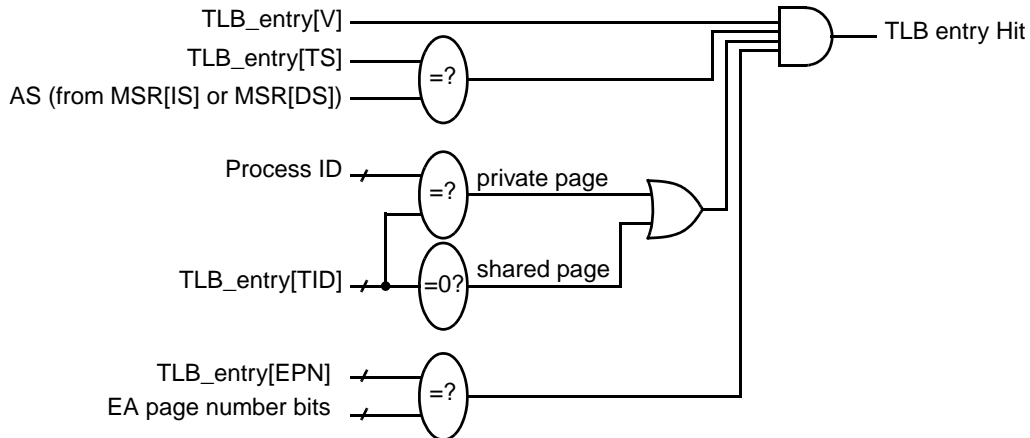


Figure 10-5. Virtual Address and TLB-Entry Compare Process

The page size defined for a TLB entry determines how many bits of the effective address are compared with the corresponding EPN field in the TLB entry as shown in [Table 10-3](#). On a TLB hit, the corresponding bits of the Real Page Number (RPN) field are used to form the real address.

Table 10-3. Page Size and EPN Field Comparison

SIZE Field	Page Size (4^{SIZE} Kbytes)	EA to EPN Comparison
0b0001	4 Kbyte	EA[0:19] =? EPN[0:19]
0b0010	16 Kbyte	EA[0:17] =? EPN[0:17]
0b0011	64 Kbyte	EA[0:15] =? EPN[0:15]
0b0100	256 Kbyte	EA[0:13] =? EPN[0:13]
0b0101	1 Mbyte	EA[0:11] =? EPN[0:11]
0b0110	4 Mbyte	EA[0:9] =? EPN[0:9]
0b0111	16 Mbyte	EA[0:7] =? EPN[0:7]
0b1000	64 Mbyte	EA[0:5] =? EPN[0:5]
0b1001	256 Mbyte	EA[0:3] =? EPN[0:3]
0b1010	1 Gbyte	EA[0:1] =? EPN[0:1]
0b1011	4Gbyte	(none)

On a TLB hit, the generation of the physical address occurs as shown in [Figure 10-6](#).

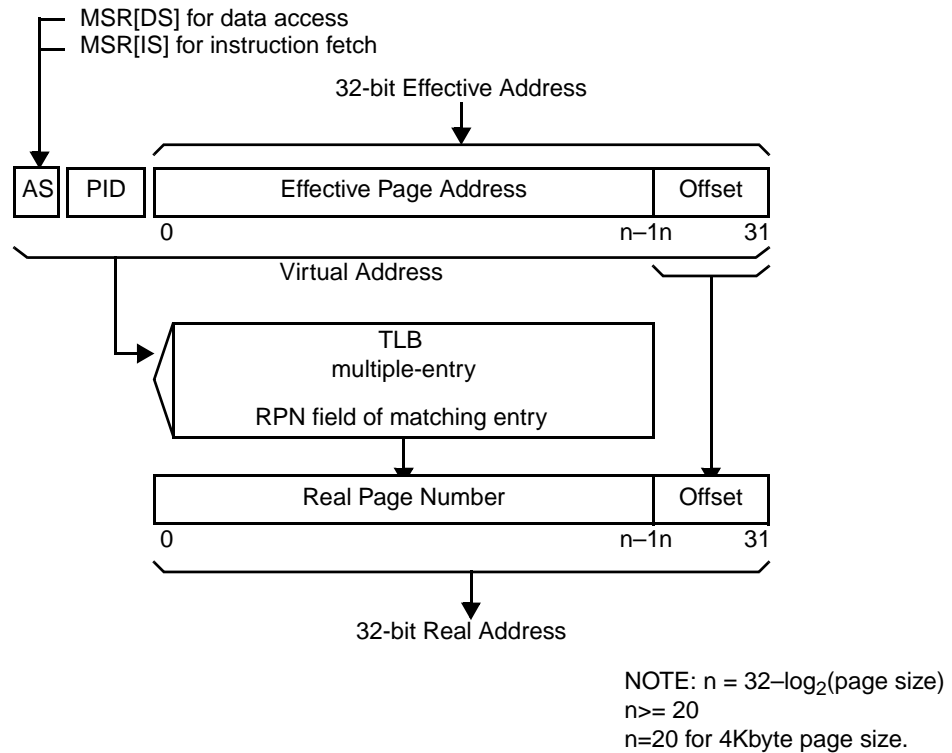


Figure 10-6. Effective to Real Address Translation Flow

Address mapping may be disabled via the `MMUCSR0_BYPASS` control bit. When bypassing is enabled, the translation flow is still followed, and translation misses may still occur, but the real address is driven directly from the effective address. Protection and attribute information is not affected by address bypassing.

10.4.1.5 Permissions

An operating system may restrict access to virtual pages by selectively granting permissions for user mode read, write, and execute, and supervisor mode read, write, and execute on a per page basis. These permissions can be set up for a particular system (for example, program code might be execute-only, data structures may be mapped as read/write/no-execute) and can also be changed by the operating system based on application requests and operating system policies.

The UX, SX, UW, SW, UR, and SR access control bits are provided to support selective permissions (access control):

- **SR**—Supervisor read permission. Allows load instructions to access the page while in supervisor mode (`MSR[PR=0]`).
- **SW**—Supervisor write permission. Allows store instructions to access the page while in supervisor mode (`MSR[PR=0]`).
- **SX**—Supervisor execute permission. Allows instruction fetches to access the page and instructions to be executed from the page while in supervisor mode (`MSR[PR=0]`).

- UR—User read permission. Allows load instructions to access the page while in user mode (MSR[PR=1]).
- UW—User write permission. Allows store instructions to access the page while in user mode (MSR[PR=1]).
- UX—User execute permission. Allows instruction fetches to access the page and instructions to be executed from the page while in user mode (MSR[PR=1]).

If the translation match was successful, the permission bits are checked as shown in Figure 10-7. If the access is not allowed by the access permission mechanism, the processor generates an Instruction or Data Storage interrupt (ISI or DSI).

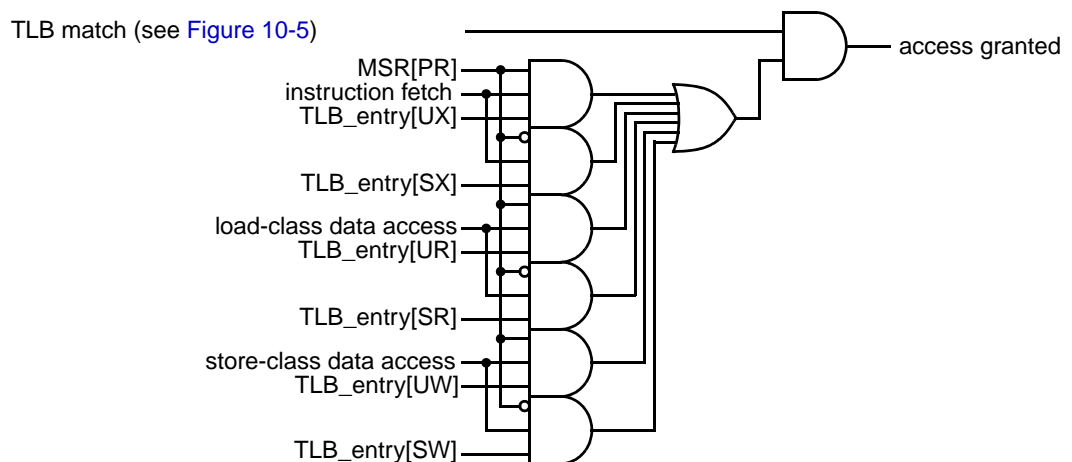


Figure 10-7. Granting of Access Permission

10.4.2 Translation Lookaside Buffer

The EIS architecture defines support for zero or more TLBs in an implementation, each with its own characteristics, and provides configuration information for software to query the existence and structure of the TLB(s) through a set of special purpose registers: MMUCFG, TLB0CFG, TLB1CFG, etc. By convention, TLB0 is used for a set associative TLB with fixed page sizes, TLB1 is used for a fully associative TLB with variable page sizes, and TLB2 is arbitrarily defined by an implementation. The e200z1 MMU supports a single TLB which is fully associative and supports variable page sizes, thus it corresponds to TLB1. For the rest of this document, TLB, TLBCAM, and TLB1 are used interchangeably.

The TLB consists of an 8-entry, fully associative CAM array with support for eleven page sizes. To perform a lookup, the CAM is searched in parallel for a matching TLB entry. The contents of this TLB entry are then concatenated with the page offset of the original effective address. The result constitutes the real (physical) address of the access.

A hit to multiple TLB entries is considered to be a programming error. If this occurs, the TLB generates an invalid address and TLB entries may be corrupted (an exception will not be reported).

Table 10-4. TLB Entry Bit Definitions

Field	Comments
V	Valid bit for entry
TS	Translation address space (compared against AS bit)
TID[0:7]	Translation ID (compared against PID0 or '0')
EPN[0:19]	Effective page number (compared against effective address)
RPN[0:19]	Real page number (translated address)
SIZE[0-3]	Page size (4K/16K/64K/256K/1M/4M/16M/64M/256M/1G/4G)
SX, SW, SR	Supervisor execute, write, and read permission bits
UX, UW, UR	User execute, write, and read permission bits
WIMGE	Translation attributes (write-through required, cache-inhibited, memory coherence required, guarded, endian)
U0-U3	User bits -- used only by software
IPROT	Invalidation protect
VLE	VLE page indicator

10.4.3 MMU Assist Registers (MAS)

e200 uses six special purpose registers (MAS0, MAS1, MAS2, MAS3, MAS4 and MAS6) to facilitate reading, writing, and searching the TLB. The MAS registers can be read or written using the **mf spr** and **mt spr** instructions. e200 does not implement the MAS5 register, present in other EIS Book E designs, because the **tlbsx** instruction only searches based on a single SPID value.

NOTE

e200z1 is a 32-bit implementation of the Power Architecture Book E specification. In this document, register bits are sometimes numbered from bit 0 (Most Significant Bit) to 31 (Least Significant Bit), rather than the Book E numbering scheme of 32:63, thus register bit numbers for some registers in Book E are 32 higher. Where appropriate, the Book E defined bit numbers are shown in parentheses.

The MAS0 register is shown in Figure 10-8. Fields are defined in [Table 10-5](#).

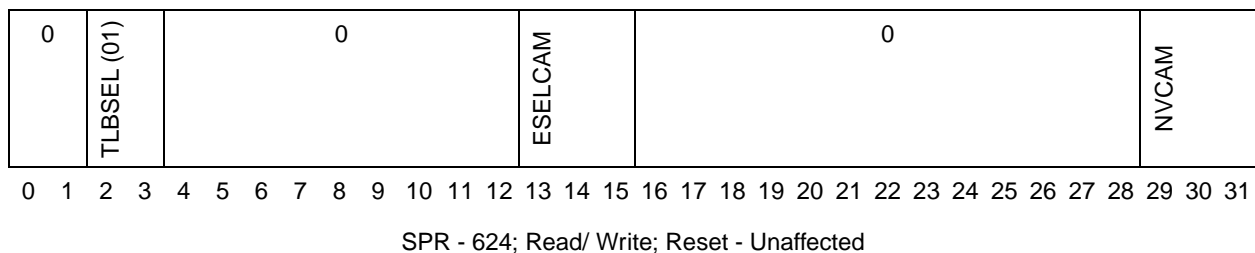
**Figure 10-8. MMU Assist Register 0 (MAS0)**

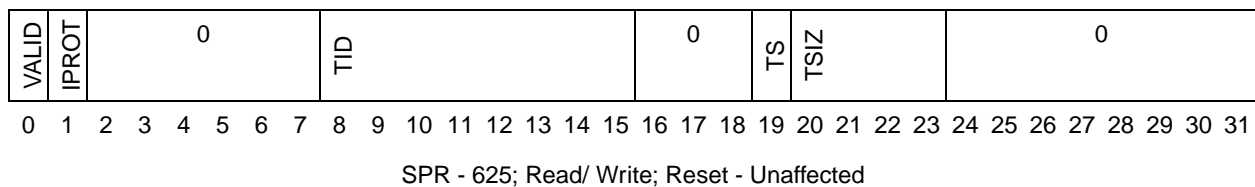
Table 10-5. MAS0 — MMU Read/Write and Replacement Control

Bit ¹	Name	Comments, or Function when Set
0:1 [32:33]	—	Reserved ²
2:3 [34:35]	TLBSEL	selects TLB for access: 01=TLBCAM (ignored by Zen, should be written to 01 for future compatibility)
4:11 [36:42]	—	Reserved ²
13:15 [44:47]	ESELCAM	Entry select for TLBCAM
16:27 [48:59]	—	Reserved ²
29:31 [61:63]	NVCAM	Next replacement victim for TLBCAM (software managed) Software updates this field; it is copied to the ESELCAM field on a TLB Error.

¹ Numbers shown in parentheses are the 64-bit register bit numbers defined in the Power Architecture Book Specification.

² These bits are not implemented, will be read as zero, and writes are ignored.

The MAS1 register is shown in [Figure 10-9](#). Fields are defined in [Table 10-6](#).

**Figure 10-9. MMU Assist Register 1 (MAS1)****Table 10-6. MAS1 —Descriptor Context and Configuration Control**

Bit ¹	Name	Comments, or Function when Set
0 [32]	VALID	TLB Entry Valid 0 - This TLB entry is invalid 1 - This TLB entry is valid
1 [33]	IPROT	Invalidation Protect 0 - Entry is not protected from invalidation 1 - Entry is protected from invalidation. Protects TLB entry from invalidation by tlbivax (TLBCAM only), or flash invalidates through MMUSCR0[TLBCAM_FI].
2:7 [34:39]	—	Reserved ²
8:15 [40:47]	TID	Translation ID bits This field is compared with the current process IDs of the effective address to be translated. A TID value of 0 defines an entry as global and matches with all process IDs.

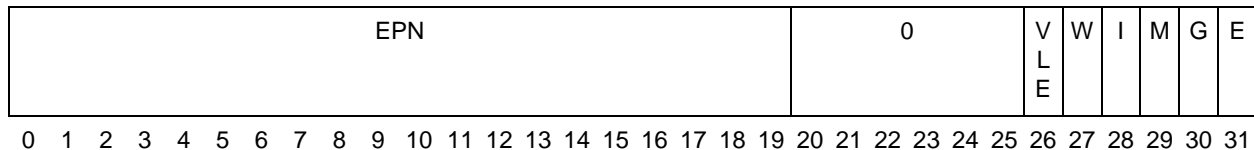
Table 10-6. MAS1 —Descriptor Context and Configuration Control

Bit ¹	Name	Comments, or Function when Set
16:18 [48:50]	—	Reserved ²
19 [51]	TS	Translation address space This bit is compared with the IS or DS fields of the MSR (depending on the type of access) to determine if this TLB entry may be used for translation.
20:23 [52:55]	TSIZE	Entry's page size Supported page sizes are: TSIZE(0:3) = 0b0001 4KB TSIZE(0:3) = 0b0010 16KB TSIZE(0:3) = 0b0011 64k TSIZE(0:3) = 0b0100 256k TSIZE(0:3) = 0b0101 1MB TSIZE(0:3) = 0b0110 4MB TSIZE(0:3) = 0b0111 16MB TSIZE(0:3) = 0b1000 64MB TSIZE(0:3) = 0b1001 256MB TSIZE(0:3) = 0b1010 1 GB TSIZE(0:3) = 0b1011 4 GB All other values are undefined
24:31 [56:63]	—	Reserved ²

¹ Numbers shown in parentheses are the 64-bit register bit numbers defined in the Power Architecture Book Specification.

² These bits are not implemented, will be read as zero, and writes are ignored.

The MAS2 register is shown in [Figure 10-10](#). Fields are defined in [Table 10-7](#).



SPR - 626; Read/ Write; Reset - Unaffected

Figure 10-10. MMU Assist Register 2 (MAS2)

Table 10-7. MAS2 - EPN and Page Attributes

Bit ¹	Name	Comments, or Function when Set
0:19 [32:51]	EPN	Effective page number [0:19]
20:25 [52:57]	—	Reserved ²

Table 10-7. MAS2 - EPN and Page Attributes

Bit ¹	Name	Comments, or Function when Set
26 [58]	VLE	Power Architecture VLE 0 - This page is a standard Book E page 1 - This page is a Power Architecture VLE page
27 [59]	W	Write-through Required 0 - This page is considered write-back with respect to the caches in the system 1 - All stores performed to this page are written through to main memory
28 [60]	I	Cache Inhibited 0 - This page is considered cacheable 1 - This page is considered cache-inhibited
29 [61]	M	Memory Coherence Required 0 - Memory Coherence is not required 1 - Memory Coherence is required Zen Z1 does <u>not</u> support the Memory Coherence required attribute, and thus it is ignored
30 [62]	G	Guarded 0 - Access to this page are not guarded, and can be performed before it is known if they are required by the sequential execution model 1 - All loads and stores to this page are performed without speculation (i.e. they are known to be required) Zen Z1 ignores the guarded attribute, since no speculative or out-of-order processing is performed.
31 [63]	E	Endianness 0 - The page is accessed in big-endian byte order. 1 - The page is accessed in true little-endian byte order. Determines endianness for the corresponding page.

¹ Numbers shown in parentheses are the 64-bit register bit numbers defined in the Power Architecture Book Specification.

² These bits are not implemented, will be read as zero, and writes are ignored.

The MAS3 register is shown in [Figure 10-11](#). Fields are defined in [Table 10-8](#).

RPN																				0	U	U	U	U	U	S	U	S	U	S	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

SPR - 627; Read/ Write; Reset - Unaffected

Figure 10-11. MMU Assist Register 3 (MAS3)

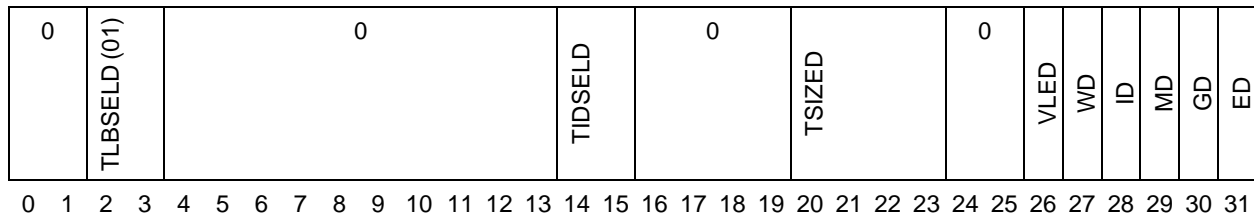
Table 10-8. MAS3 - RPN and Access Control

Bit ¹	Name	Comments, or Function when Set
0:19 [32:51]	RPN	Real page number [0:19] Only bits that correspond to a page number are valid. Bits that represent offsets within a page are ignored and should be zero.
20:21 [52:53]	—	Reserved ²
22:25 [54:57]	U0-U3	User bits [0-3]
26:31 [58:63]	PERMIS	Permission bits (UX, SX, UW, SW, UR, SR)

¹ Numbers shown in parentheses are the 64-bit register bit numbers defined in the Power Architecture Book Specification.

² These bits are not implemented, will be read as zero, and writes are ignored.

The MAS4 register is shown in [Figure 10-12](#). Fields are defined in [Table 10-9](#).



SPR - 628; Read/ Write; Reset - Unaffected

Figure 10-12. MMU Assist Register 4 (MAS4)

Table 10-9. MAS4 - Hardware Replacement Assist Configuration Register

Bit ¹	Name	Comments, or Function when Set
0:1 [32:33]	—	Reserved ²
2:3 [34:35]	TLBSELD	Default TLB selected: 01=TLBCAM (ignored by Zen, should be written to 01 for future compatibility)
4:13 [36:45]	—	Reserved ²
14:15 [46:47]	TIDSELD	Default PID# to load TID from 00 - PID0 01 - Reserved, do not use 10 - Reserved, do not use 11=TIDZ (8'h00) (Use all zeros, the globally shared value)
16:19 [48:51]	—	Reserved ²
20:23 [52:55]	TSIZED	Default TSIZE value

Table 10-9. MAS4 - Hardware Replacement Assist Configuration Register

Bit ¹	Name	Comments, or Function when Set
24:25 [56:57]	—	Reserved ²
26 [58]	VLED	Default VLE value
27:31 [59:63]	DWIMGE	Default WIMGE values

¹ Numbers shown in parentheses are the 64-bit register bit numbers defined in the Power Architecture Book Specification.

² These bits are not implemented, will be read as zero, and writes are ignored.

The MAS6 register is shown in [Figure 10-13](#). Fields are defined in [Table 10-10](#).

0	SPID	0	SAS																												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

SPR - 630; Read/ Write; Reset - Unaffected

Figure 10-13. MMU Assist Register 6 (MAS6)**Table 10-10. MAS6 - TLB Search Context Register 0**

Bit ¹	Name	Comments, or Function when Set
0:7 [32:39]	—	Reserved ²
8:15 [40:47]	SPID	PID value for searches
16:30 [48:62]	—	Reserved ²
31 [63]	SAS	AS value for searches

¹ Numbers shown in parentheses are the 64-bit register bit numbers defined in the Power Architecture Book Specification.

² These bits are not implemented, will be read as zero, and writes are ignored.

10.5 Interrupt Types

The interrupts implemented on the MPC5510 and the exception conditions that cause them are listed in [Table 10-11](#).

Table 10-11. Exceptions and Conditions

Interrupt Type	Interrupt Vector Offset Register	Causing Conditions
System reset	none, vector to address determined by CRP_Z1VEC	<ol style="list-style-type: none"> 1. Reset. 2. Watchdog Timer Reset Control. 3. Debug Reset Control.
Critical Input	IVOR 0 ¹	Non maskable interrupt request and MSR[CE]=1.
Machine check	IVOR 1	<ol style="list-style-type: none"> 1. Machine check error and MSR[ME] =1. 2. ISI, ITLB Error on first instruction fetch for an exception handler and current MSR[ME]=1. 3. Bus error (XTE) with MSR[EE]=0 and current MSR[ME]=1
Data Storage	IVOR 2	<ol style="list-style-type: none"> 1. Access control. 2. Byte ordering due to misaligned access across page boundary to pages with mismatched E bits. 3. Precise external termination error and MSR[EE]=1.
Instruction Storage	IVOR 3	<ol style="list-style-type: none"> 1. Access control. 2. Byte ordering due to misaligned instruction across page boundary to pages with mismatched VLE bits, or access to page with VLE set and E indicating little-endian. 3. Misaligned Instruction fetch due to a change of flow to an odd halfword instruction boundary on a Book E (non-VLE) instruction page 4. Precise external termination error and MSR[EE]=1.
External Input	IVOR 4 ¹	Interrupt request and MSR[EE]=1.
Alignment	IVOR 5	<ol style="list-style-type: none"> 1. lmw, stmw not word aligned. 2. dcbz with disabled cache or no cache present, or to W or I storage. 3. lwarx or stwcx. not word aligned.
Program	IVOR 6	Illegal, Privileged, Trap, FP enabled, AP enabled, Unimplemented Operation.
Floating-point unavailable	IVOR 7	MSR[FP]=0 and attempt to execute a Book E floating point operation.
System call	IVOR 8	Execution of the System Call (sc , se_sc) instruction
AP unavailable	IVOR 9	Unused
Decrementer	IVOR 10	As specified in <i>Book E: Enhanced PowerPC™ Architecture v0.99</i> , Ch. 8, pg. 190-191
Fixed Interval Timer	IVOR 11	As specified in <i>Book E: Enhanced PowerPC™ Architecture v0.99</i> , Ch. 8, pg. 191-192
Watchdog Timer	IVOR 12	As specified in <i>Book E: Enhanced PowerPC™ Architecture v0.99</i> , Ch. 8, pg. 192-194
Data TLB Error	IVOR 13	Data translation lookup did not match a valid entry in the TLB
Instruction TLB Error	IVOR 14	Instruction translation lookup did not match a valid entry in the TLB

Table 10-11. Exceptions and Conditions (continued)

Interrupt Type	Interrupt Vector Offset Register	Causing Conditions
Debug	IVOR 15	Trap, Instruction Address Compare, Data Address Compare, Instruction Complete, Branch Taken, Return from Interrupt, Interrupt Taken, Debug Counter, External Debug Event, Unconditional Debug Event
Reserved	IVOR 16-31	—

¹ Autovectored External and Critical Input interrupts use this IVOR. Vectored interrupts supply an interrupt vector offset directly.

10.6 Bus Interface Unit (BIU)

The BIU encompasses control and data signals supporting instruction and data transfers, support for interrupts, including vectored interrupt logic, reset support, power management interface signals, debug event signals, time base control and status information, processor state information, Nexus / OnCE / JTAG interface signals, and a test interface.

The memory portion of the e200 core interface is comprised of a pair of 32-bit wide system buses, one for instructions and the other for data. The data memory interface supports read and write transfers of 8, 16, 24, and 32 bits, supports misaligned transfers, supports true big- and little-endian operating modes, and operates in a pipelined fashion. The instruction memory interface supports read transfers of 16 and 32 bits, supports misaligned transfers, supports true big- and little-endian operating modes, and operates in a pipelined fashion.

Single-beat and misaligned transfers are supported for read and write cycles. Incrementing burst transfers are supported for instruction prefetch operations.

Chapter 11

e200z0 Core (Z0)

11.1 Introduction

The e200 processor family is a set of CPU cores that implement low-cost versions of the Power Architecture Book E architecture. e200 processors are designed for deeply embedded control applications that require low cost solutions rather than maximum performance.

The e200z0 processors integrate an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable of sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle.

The e200z0 core is a single-issue, 32-bit Power Architecture VLE-only design with 32-bit general purpose registers (GPRs). All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs).

Instead of the base Power Architecture Book E instruction set support, the e200z0 core implements only the VLE (variable-length encoding) APU, providing improved code density. The VLE APU is further documented in “PowerPC VLE APU Definition, Version 1.01”, a separate document.

In the remainder of this document, the e200z0 core is also referred to as the ‘e200z0 core’ or ‘e200 core’.

11.1.1 Features

The following is a list of some of the key features of the e200z0 core:

- 32-bit Power Architecture VLE-only programmer’s model
- Single issue, 32-bit CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
 - Dedicated branch address calculation adder
- Supports instruction and data access via a unified 32-bit Instruction/Data BIU (e200z0 only).
- Load/store unit
 - 1 cycle load latency
 - Fully pipelined
 - Big-endian support only
 - Misaligned access support

- Zero load-to-use pipeline bubbles for aligned transfers
- Power management
 - Low power design
 - Power saving modes: doze, nap, sleep, and wait
 - Dynamic power management of execution units

NOTE

The MPC5510 does not use the core's HID0[DOZE,NAP,SLEEP] bits to enter/exit low-power modes. Entry to and exit from low-power modes is managed by the CRP module.

11.2 Microarchitecture Summary

The execution pipeline four stages operate in an overlapped fashion, allowing single-clock instruction execution for most instructions. These stages are as follows:

1. The instruction fetch
2. Instruction decode/register file read/effective address calculation
3. Execute/memory access
4. Register writeback

The integer execution unit consists of a 32-bit arithmetic unit (AU), a logic unit (LU), a 32-bit barrel shifter (Shifter), a mask insertion unit (MIU), a condition register manipulation unit (CRU), a count-leading-zeros unit (CLZ), an 8x32 hardware multiplier array, result feed-forward hardware, and a hardware divider.

Arithmetic and logical operations are executed in a single cycle with the exception of the divide and multiply instructions. A count-leading-zeros unit operates in a single clock cycle.

The instruction unit contains a PC incrementer and a dedicated branch address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Prefetched instructions are placed into an instruction buffer with 62 entries, each capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Conditional branches which are not taken execute in a single clock. All taken branches have an execution time of two clocks.

Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The condition register unit supports the condition register (CR) and condition register operations defined by the Power Architecture. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

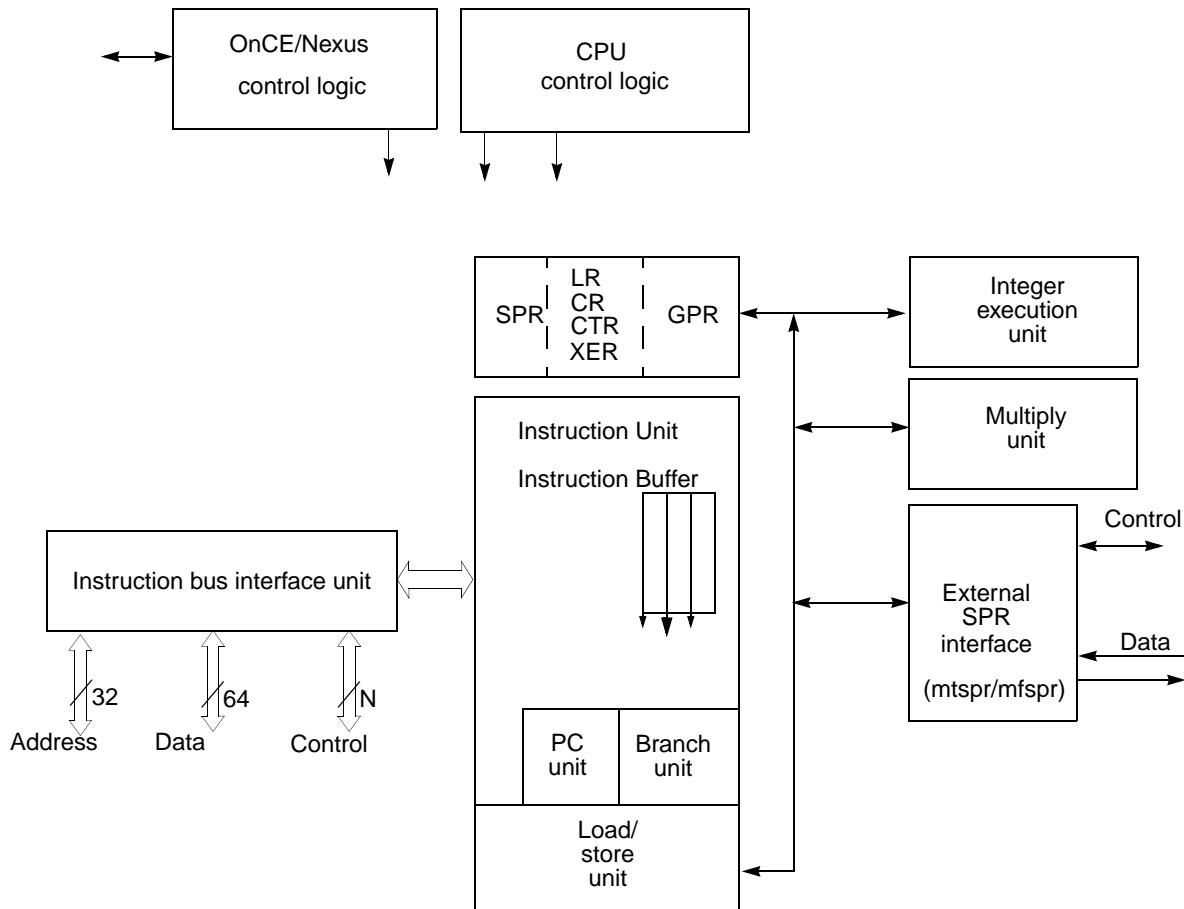


Figure 11-1. e200z0 Block Diagram

11.2.1 Instruction Unit Features

The features of the e200 Instruction unit are:

- 32-bit instruction fetch path supports fetching of one 32-bit instruction per clock, or up to two 16-bit VLE instructions per clock.
- Instruction buffer with two entries, each holding a single 32-bit instruction, or a pair of 16-bit instructions
- Dedicated PC incremter supporting instruction prefetches
- Branch unit with dedicated branch address adder supporting single cycle of execution of certain branches, two cycles for all others

11.2.2 Integer Unit Features

The e200 integer unit supports single cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count leading zero's function
- 32-bit single cycle barrel shifter for shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in 5 to 34 clocks with minimized execution timing
- 8x32 hardware multiplier array supports 1 to 4 cycle 32x32->32 multiply (early out)

11.2.3 Load/Store Unit Features

The e200 load/store unit supports load, store, and the load multiple / store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- 32-bit interface to memory

11.2.4 e200z0 System Bus Features

The features of the e200z0 System Bus interface are as follows:

- Unified instruction/data bus
- 32-bit address bus plus attributes and control
- Separate uni-directional 32-bit read data bus and 32-bit write data bus
- Overlapped, in-order accesses

11.3 Core Registers and Programmer's Model

This section describes the registers implemented in the e200z0 core. It includes an overview of registers defined by the Power Architecture Book E architecture, highlighting differences in how these registers are implemented in the e200 core, and provides a detailed description of e200-specific registers. Full descriptions of the architecture-defined register set are provided in Power Architecture Book E Specification.

The Power Architecture Book E defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions. Data is transferred between memory and registers with explicit load and store instructions only.

Figure 11-2 and Figure 11-3 show the e200 register set including the registers which are accessible while in supervisor mode, and the registers which are accessible in user mode. The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register (for example, the integer exception register (XER) is SPR 1).

NOTE

e200z0 is a 32-bit implementation of the Power Architecture Book E specification. In this document, register bits are sometimes numbered from bit 0 (most significant bit) to 31 (least significant bit), rather than the Book E numbering scheme of 32:63, thus register bit numbers for some registers in Book E are 32 higher.

Where appropriate, the Book E defined bit numbers are shown in parentheses.

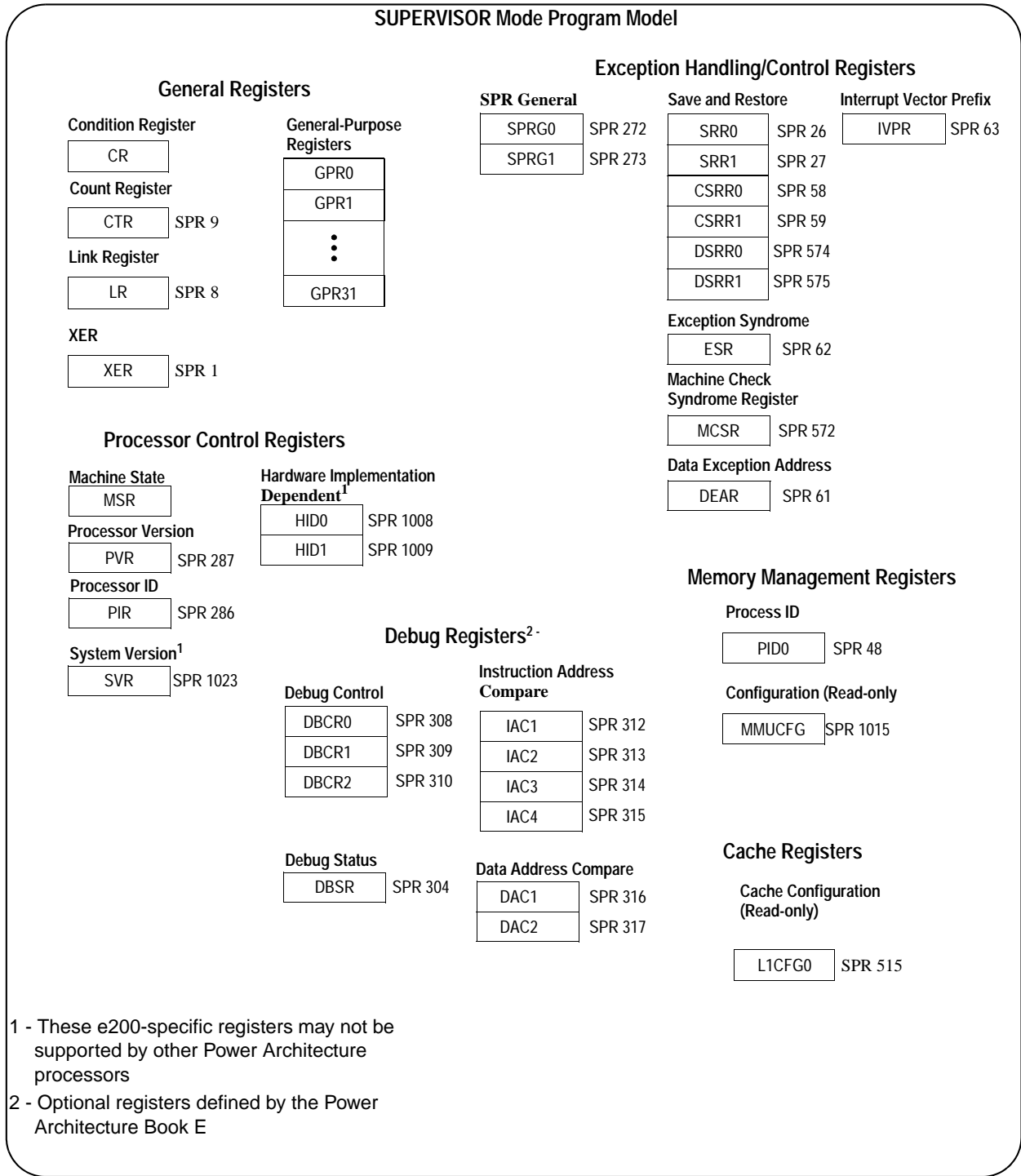


Figure 11-2. e200z0 Supervisor Mode Programmer's Model

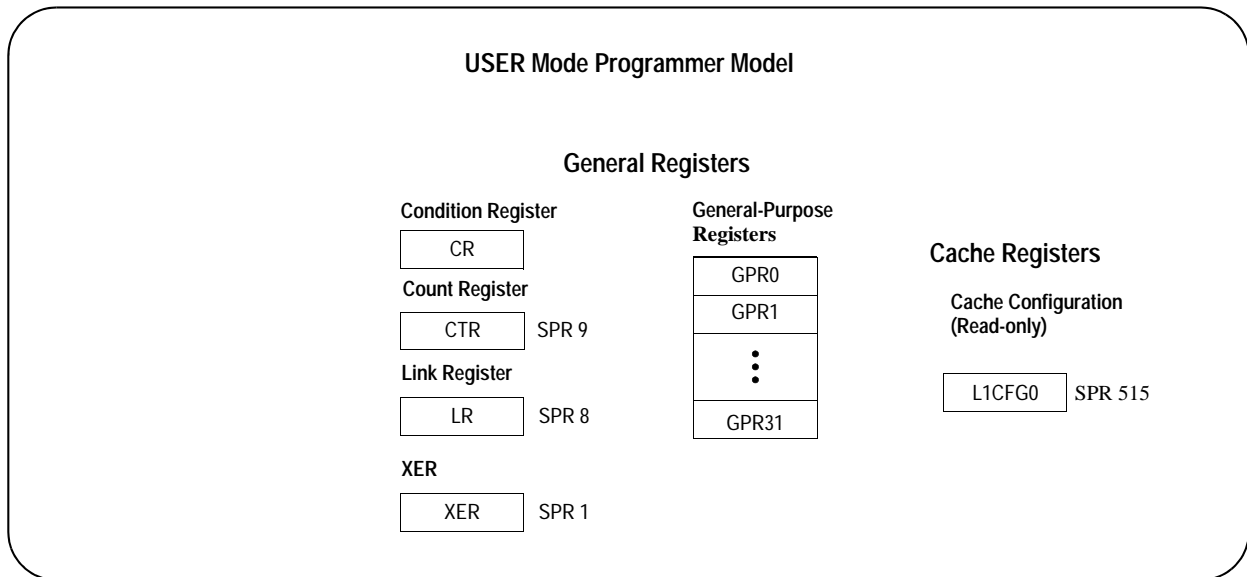


Figure 11-3. e200 User Mode Program Model

General purpose registers (GPRs) are accessed through instruction operands. Access to other registers can be explicit (by using instructions for that purpose such as Move to Special Purpose Register (**mtspr**) and Move from Special Purpose Register (**mfspir**) instructions) or implicit as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

11.3.1 Power Architecture Book E Registers

e200 supports a subset of the registers defined by *Power Architecture™ Book E Specification*. Notable exceptions are the Floating Point registers FPR0-FPR31 and FPSCR. e200z0 does not support the Book E floating-point architecture. The e200-supported Power Architecture Book E registers are described as follows (e200-specific registers are described in [Section 11.3.2, “e200-Specific Special Purpose Registers”](#)).

11.3.1.1 User-Level Registers

The user-level registers can be accessed by all software with either user or supervisor privileges. They include the following:

- General-purpose registers (GPRs). The thirty-two 32-bit GPRs (GPR0–GPR31) serve as data source or destination registers for integer instructions and provide data for generating addresses.
- Condition register (CR). The 32-bit CR consists of eight 4-bit fields, CR0–CR7, that reflect results of certain arithmetic operations and provide a mechanism for testing and branching. See “Condition Register (CR),” in Chapter 3, “Branch and Condition Register Operations, Power Architecture Book E Specification.

The remaining user-level registers are SPRs. Note that the Power Architecture Book E provides the **mtspr** and **mfspir** instructions for accessing SPRs.

Integer exception register (XER). The XER indicates overflow and carries for integer operations. See “XER Register (XER),” in Chapter 4, “Integer Operations” of *Power Architecture Book E Specification* for more information.

- Link register (LR). The LR provides the branch target address for the Branch to Link Register (`se_blr`, `se_btrl`) instructions, and is used to hold the address of the instruction that follows a branch and link instruction, typically used for linking to subroutines. See “Link Register (LR),” in Chapter 3, “Branch and Condition Register Operations” of *Power Architecture Book E Specification*.
- Count register (CTR). The CTR holds a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR also provides the branch target address for the Branch to Count Register (`se_bctr`, `se_bctrl`) instructions. See “Count Register (CTR),” in Chapter 3, “Branch and Condition Register Operations” of *Power Architecture Book E Specification*.

11.3.1.2 Supervisor-Level Registers

In addition to the registers accessible in user mode, Supervisor-level software has access to additional control and status registers used for configuration, exception handling, and other operating system functions. The Power Architecture Book E defines the following supervisor-level registers:

- Processor Control Registers
 - Machine State Register (MSR). The MSR defines the state of the processor. The MSR can be modified by the Move to Machine State Register (`mtmsr`), System Call (`se_sc`), and Return from Exception (`se_rfi`, `se_rfci`, `se_rfdi`) instructions. It can be read by the Move from Machine State Register (`mfmsr`) instruction. When an interrupt occurs, the contents of the MSR are saved to one of the machine state save/restore registers (SRR1, CSRR1, DSRR1).
 - Processor version register (PVR). This register is a read-only register that identifies the processor type and version (model) and the revision level of the processor. [Table 11-1](#) shows the PVR values and the corresponding processor type and version numbers for the cores used on the MPC5510 Family.

Table 11-1. PVR Values, and Processor Type and Version Numbers

Device	Core	PVR Value	Type	Version
MPC5516	e200Z1	0x8144_0000	0x14	0x4
MPC5516	e200Z0	0x8171_0000	0x17	0x1

- Processor Identification Register (PIR). This read-only register is provided to distinguish the processor from other processors in the system.
- Storage Control Register
 - Process ID Register (PID, also referred to as PID0). This register is provided to indicate the current process or task identifier. It is used by the Nexus2 module for Ownership Trace message generation. Although the Power Architecture Book E allows for multiple PIDs, e200z0 implements only one.
- Interrupt Registers

- Data Exception Address Register (DEAR). After most Data Storage Interrupts (DSI), or on an Alignment Interrupt, the DEAR is set to the effective address (EA) generated by the faulting instruction.
- SPRG0-SPRG1. The SPRG0-SPRG1 registers are provided for operating system or interrupt handler use.
- Exception Syndrome Register (ESR). The ESR register provides a syndrome to differentiate between the different kinds of exceptions which can generate the same interrupt.
- Interrupt Vector Prefix Register (IVPR). This register together with hardwired offsets which replace the IVOR0-15 registers provide the address of the interrupt handler for different classes of interrupts.
- Save/Restore Register 0 (SRR0). The SRR0 register is used to save machine state on a non-critical interrupt, and contains the address of the instruction at which execution resumes when an `se_rfi` instruction is executed at the end of a non-critical class interrupt handler routine.
- Critical Save/Restore register 0 (CSRR0). The CSRR0 register is used to save machine state on a critical interrupt, and contains the address of the instruction at which execution resumes when an `se_rfc` instruction is executed at the end of a critical class interrupt handler routine.
- Save/Restore register 1 (SRR1). The SRR1 register is used to save machine state from the MSR on non-critical interrupts, and to restore machine state when `se_rfi` executes.
- Critical Save/Restore register 1 (CSRR1). The CSRR1 register is used to save machine state from the MSR on critical interrupts, and to restore machine state when `se_rfc` executes.
- Debug Facility Registers
 - Debug Control Registers (DBCR0-DBCR2). These registers provide control for enabling and configuring debug events.
 - Debug Status Register (DBSR). This register contains debug event status.
 - Instruction Address Compare registers (IAC1-IAC4). These registers contain addresses and/or masks which are used to specify Instruction Address Compare debug events.
 - Data address compare registers (DAC1-2). These registers contain addresses and/or masks which are used to specify Data Address Compare debug events.
 - e200 does **not** implement the Data Value Compare registers (DVC1 and DVC2).

11.3.2 e200-Specific Special Purpose Registers

The Power Architecture Book E architecture allows implementation-specific special purpose registers. Those incorporated in the e200 core are as follows:

11.3.2.1 User-Level Registers

The user-level registers can be accessed by all software with either user or supervisor privileges. They include the following:

- The L1 Cache Configuration register (L1CFG0). This read-only register allows software to query the configuration of the L1 Cache. For the e200z0, this register returns all zeros indicating no cache is present.

11.3.2.2 Supervisor-level registers

The following supervisor-level registers are defined in e200 in addition to the Power Architecture Book E registers described above:

- Configuration Registers
 - Hardware implementation-dependent register 0 (HID0). This register controls various processor and system functions.
 - Hardware implementation-dependent register 1 (HID1). This register controls various processor and system functions.
- Exception Handling and Control Registers
 - Machine Check Syndrome register (MCSR). This register provides a syndrome to differentiate between the different kinds of conditions which can generate a Machine Check.
 - Debug Save/Restore register 0 (DSRR0). When enabled, the DSRR0 register is used to save the address of the instruction at which execution continues when `se_rfdi` executes at the end of a debug interrupt handler routine.
 - Debug Save/Restore register 1 (DSRR1). When enabled, the DSRR1 register is used to save machine status on debug interrupts and to restore machine status when `se_rfdi` executes.
- L1 Cache Configuration Register (L1CFG0) is a read-only register that allows software to query the configuration of the L1 Cache. For the e200z0, this register returns all zeros.
- System version register (SVR). This register is a read-only register that identifies the version (model) and revision level of the SoC which includes an e200 Power Architecture processor.

Note that it is not guaranteed that the implementation of e200 core-specific registers is consistent among Power Architecture processors, although other processors may implement similar or identical registers. All e200 SPR definitions are compliant with the Freescale EIS specification definitions.

11.3.3 e200z0 Core Complex Features Not Supported on the MPC5510

The MPC5510 implements a subset of the e200z0 core complex features. The e200z0 core complex features that are not supported in the MPC5510 are described in [Table 11-2](#).

Table 11-2. e200z0 Features Not Supported on the MPC5510 Family

Description	Function/Category
The less significant halfword of the Processor Version Register (PVR) provides the revision level which is comprised of the following three bit fields: Reserved = 0x00 Revision = 0x0 ID = 0x0 The more significant halfword of the Processor Version Register (PVR) provides the processor type and version number (see Table 11-1).	PVR Value
Nexus registers are not accessible by code running in User or Supervisor mode. Nexus registers can be accessed only by external tools via the Nexus port.	Debug

11.4 Interrupt Types

the interrupts implemented on the MPC5510 and the exception conditions that cause them are listed in [Table 11-3](#).

Table 11-3. Exceptions and Conditions

Interrupt Type	Interrupt Vector Offset Register	Causing Conditions
System reset	none, vector to address determined by CRP_ZOVEC	1. Reset. 2. Debug Reset Control.
Critical Input	IVOR 0 ¹	Non maskable interrupt request and MSR[CE]=1.
Machine check	IVOR 1	1. Machine check error and MSR[ME] =1. 2. Bus error (XTE) with MSR[EE]=0 and current MSR[ME]=1
Data Storage	IVOR 2	1. Access control. (unused on e200z0) 2. Precise external termination error and MSR[EE]=1.
Instruction Storage	IVOR 3	1. Access control. (unused on e200z0) 2. Precise external termination error and MSR[EE]=1.
External Input	IVOR 4 ¹	Interrupt request and MSR[EE]=1.
Alignment	IVOR 5	1. lmw , stmw not word aligned. 2. lwarx or stwcx . not word aligned.
Program	IVOR 6	Illegal, Privileged, Trap, Unimplemented Operation.
Floating-point unavailable	IVOR 7	Unused
System call	IVOR 8	Execution of the System Call (se_sc) instruction
AP unavailable	IVOR 9	Unused
Decrementer	IVOR 10	Unused

Table 11-3. Exceptions and Conditions (continued)

Interrupt Type	Interrupt Vector Offset Register	Causing Conditions
Fixed Interval Timer	IVOR 11	Unused
Watchdog Timer	IVOR 12	Unused
Data TLB Error	IVOR 13	Unused
Instruction TLB Error	IVOR 14	Unused
Debug	IVOR 15	Trap, Instruction Address Compare, Data Address Compare, Instruction Complete, Branch Taken, Return from Interrupt, Interrupt Taken, External Debug Event, Unconditional Debug Event
Reserved	IVOR 16-31	—

¹ Autovectored External and Critical Input interrupts use this IVOR. Vectored interrupts supply an interrupt vector offset directly.

11.5 Bus Interface Unit (BIU)

The BIU encompasses control and data signals supporting instruction and data transfers, support for interrupts, including vectored interrupt logic, reset support, power management interface signals, debug event signals, processor state information, Nexus /OnCE / JTAG interface signals, and a test interface.

The memory portion of the e200 core interface is comprised of a 32-bit wide system bus and a unified bus. The memory interface supports read and write transfers of 8, 16, 24, and 32 bits, supports misaligned transfers, and operates in a pipelined fashion.

Single-beat and misaligned transfers are supported for read and write cycles. Incrementing burst transfers are supported for instruction prefetch operations.

Chapter 12

Enhanced Direct Memory Access (eDMA)

12.1 Introduction

The enhanced direct memory access controller (eDMA) is a second-generation platform block capable of performing complex data movements through 16 programmable channels, with minimal intervention from the host processor. The hardware microarchitecture includes a DMA engine that performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels. This implementation minimizes the overall block size.

12.1.1 Block Diagram

A simplified block diagram of the eDMA illustrates the functionality and interdependence of major blocks (see [Figure 12-1](#)).

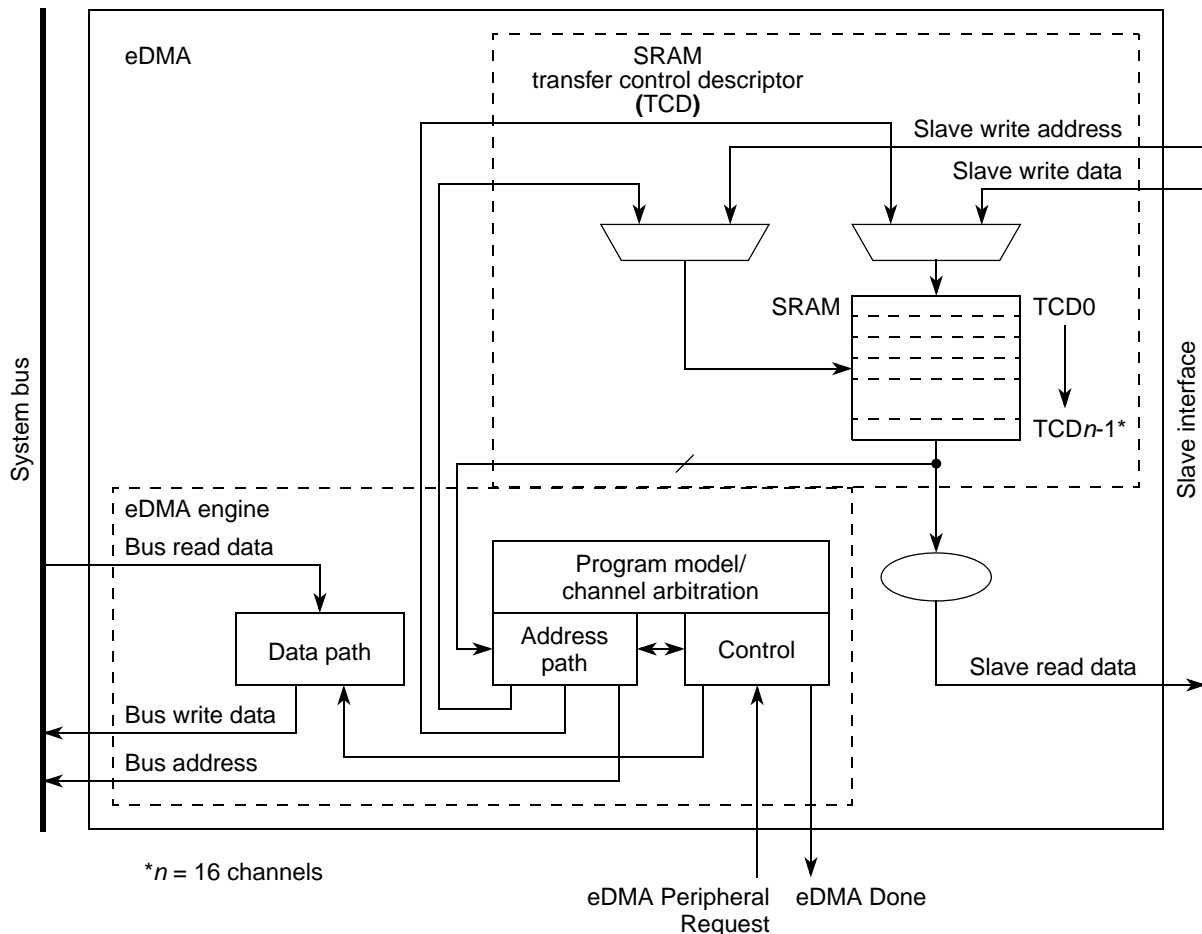


Figure 12-1. eDMA Block Diagram

12.1.2 Features

The eDMA has these major features:

- All data movement via dual-address transfers: read from source, write to destination
 - Programmable source, destination addresses, transfer size, and support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations
 - An inner data transfer loop defined by a minor byte transfer count
 - An outer data transfer loop defined by a major iteration count
- Channel activation via one of three methods:
 - Explicit software initiation
 - Initiation via a channel-to-channel linking mechanism for continuous transfers
 - Peripheral-paced hardware requests (one per channel)

All three methods require one activation per execution of the minor loop
- Support for fixed-priority and round-robin channel arbitration

- Channel completion reported via optional interrupt requests
 - One interrupt per channel, optionally asserted at completion of major iteration count
 - Error terminations are optionally enabled per channel and logically summed together to form a single error interrupt.
- Support for scatter-gather DMA processing
- Any channel can be programmed to be suspended by a higher priority channel's activation, before completion of a minor loop.

12.1.3 Modes of Operation

There are two main operating modes of eDMA: normal mode and debug mode. These modes are briefly described in this section.

12.1.3.1 Normal Mode

In normal mode, the eDMA is used to transfer data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

12.1.3.2 Debug Mode

In debug mode, the eDMA will not accept new transfer requests when its debug input signal is asserted. If the signal is asserted during transfer of a block of data described by a minor loop in the current active channel's TCD, the eDMA will continue operation until completion of the minor loop.

12.2 External Signal Description

The eDMA has no external signals.

12.3 Memory Map and Registers

This section provides a detailed description of all eDMA registers.

12.3.1 Module Memory Map

The eDMA memory map is shown in [Table 12-1](#). The address of each register is given as an offset to the eDMA base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed. [Table 12-2](#) shows a graphical representation of the same memory map.

The eDMA's programming model is partitioned into two regions: the first region defines a number of registers providing control functions; however, the second region corresponds to the local transfer control descriptor memory.

Some registers are implemented as two 32-bit registers, and include H and L suffixes, signaling the high and low portions of the control function.

Table 12-1. eDMA Memory Map

Offset from EDMA_BASE (0xFFF4_4000)	Register	Access	Reset Value	Section/Page	Size
0x0000	EDMA_CR — eDMA control register	R/W	0x0000_0000	12.3.2.1/12-7	32
0x0004	EDMA_ESR — eDMA error status register	R	0x0000_0000	12.3.2.2/12-8	32
0x0008	Reserved				
0x000E	EDMA_ERQRL — eDMA enable request low register (channels 15–00)	R/W	0x0000	12.3.2.3/12-10	16
0x0010	Reserved				
0x0016	EDMA_EEIRL — eDMA enable error interrupt low register (channels 15–00)	R/W	0x0000	12.3.2.4/12-11	16
0x0018	EDMA_SERQR — eDMA set enable request register	W	0x00	12.3.2.5/12-11	8
0x0019	EDMA_CERQR — eDMA clear enable request register	W	0x00	12.3.2.6/12-12	8
0x001A	EDMA_SEEIR — eDMA set enable error interrupt register	W	0x00	12.3.2.7/12-13	8
0x001B	EDMA_CEEIR — eDMA clear enable error interrupt register	W	0x00	12.3.2.8/12-13	8
0x001C	EDMA_CIRQR — eDMA clear interrupt request register	W	0x00	12.3.2.9/12-14	8
0x001D	EDMA_CER — eDMA clear error register	W	0x00	12.3.2.10/12-15	8
0x001E	EDMA_SSBP — eDMA set start bit register	W	0x00	12.3.2.11/12-15	8
0x001F	EDMA_CDSBR — eDMA clear done status bit register	W	0x00	12.3.2.12/12-16	8
0x0020	Reserved				
0x0026	EDMA_IRQRL — eDMA interrupt request low register	R/W	0x0000	12.3.2.13/12-16	16
0x0028	Reserved				
0x002E	EDMA_ERL — eDMA error low register	R/W	0x0000	12.3.2.14/12-17	16
0x0030	Reserved				
0x0100	EDMA_CPR0 — eDMA channel 0 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x0101	EDMA_CPR1 — eDMA channel 1 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x0102	EDMA_CPR2 — eDMA channel 2 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x0103	EDMA_CPR3 — eDMA channel 3 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x0104	EDMA_CPR4 — eDMA channel 4 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x0105	EDMA_CPR5 — eDMA channel 5 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x0106	EDMA_CPR6 — eDMA channel 6 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x0107	EDMA_CPR7 — eDMA channel 7 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x0108	EDMA_CPR8 — eDMA channel 8 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x0109	EDMA_CPR9 — eDMA channel 9 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x010A	EDMA_CPR10 — eDMA channel 10 priority register	R/W	— ¹	12.3.2.15/12-18	8

Table 12-1. eDMA Memory Map (continued)

Offset from EDMA_BASE (0xFFFF4_4000)	Register	Access	Reset Value	Section/Page	Size
0x010B	EDMA_CPR11 — eDMA channel 11 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x010C	EDMA_CPR12 — eDMA channel 12 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x010D	EDMA_CPR13 — eDMA channel 13 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x010E	EDMA_CPR14 — eDMA channel 14 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x010F	EDMA_CPR15 — eDMA channel 15 priority register	R/W	— ¹	12.3.2.15/12-18	8
0x0110	Reserved				
0x1000	TCD00 — eDMA transfer control descriptor 00	R/W	— ¹	12.3.2.16/12-19	256
0x1020	TCD01 — eDMA transfer control descriptor 01	R/W	— ¹	12.3.2.16/12-19	256
0x1040	TCD02 — eDMA transfer control descriptor 02	R/W	— ¹	12.3.2.16/12-19	256
0x1060	TCD03 — eDMA transfer control descriptor 03	R/W	— ¹	12.3.2.16/12-19	256
0x1080	TCD04 — eDMA transfer control descriptor 04	R/W	— ¹	12.3.2.16/12-19	256
0x10A0	TCD05 — eDMA transfer control descriptor 05	R/W	— ¹	12.3.2.16/12-19	256
0x10C0	TCD06 — eDMA transfer control descriptor 06	R/W	— ¹	12.3.2.16/12-19	256
0x10E0	TCD07 — eDMA transfer control descriptor 07	R/W	— ¹	12.3.2.16/12-19	256
0x1100	TCD08 — eDMA transfer control descriptor 08	R/W	— ¹	12.3.2.16/12-19	256
0x1120	TCD09 — eDMA transfer control descriptor 09	R/W	— ¹	12.3.2.16/12-19	256
0x1140	TCD10 — eDMA transfer control descriptor 10	R/W	— ¹	12.3.2.16/12-19	256
0x1160	TCD11 — eDMA transfer control descriptor 11	R/W	— ¹	12.3.2.16/12-19	256
0x1180	TCD12 — eDMA transfer control descriptor 12	R/W	— ¹	12.3.2.16/12-19	256
0x11A0	TCD13 — eDMA transfer control descriptor 13	R/W	— ¹	12.3.2.16/12-19	256
0x11C0	TCD14 — eDMA transfer control descriptor 14	R/W	— ¹	12.3.2.16/12-19	256
0x11E0	TCD15 — eDMA transfer control descriptor 15	R/W	— ¹	12.3.2.16/12-19	256
0x1200	Reserved				

¹ Refer to the register description for the reset value.

Table 12-2. eDMA 32-bit Memory Map—Graphical View

Address	Register	
0xFFFF4_4000	eDMA Control Register (EDMA_CR)	
0xFFFF4_4004	eDMA Error Status (EDMA_ESR)	
0xFFFF4_4008	Reserved	
0xFFFF4_400C	Reserved	eDMA Enable Request Low (EDMA_ERQRL, channels 15-00)

Table 12-2. eDMA 32-bit Memory Map—Graphical View (continued)

Address	Register			
0xFFFF4_4010	Reserved			
0xFFFF4_4014	Reserved		eDMA Enable Error Interrupt Low (EDMA_EEIRL, Channels 15-00)	
0xFFFF4_4018	eDMA Set Enable Request (EDMA_SERQR)	eDMA Clear Enable Request (EDMA_CERQR)	eDMA Set Enable Error Interrupt (EDMA_SEEIR)	eDMA Clear Enable Error Interrupt (EDMA_CEEIR)
0xFFFF4_401C	eDMA Clear Interrupt Request (EDMA_CIRQR)	eDMA Clear Error (EDMA_CER)	eDMA Set Start Bit, Activate Channel (EDMA_SSBP)	eDMA Clear Done Status Bit (EDMA_CDSBR)
0xFFFF4_4020	Reserved			
0xFFFF4_4024	Reserved		eDMA Interrupt Request Low (EDMA_IRQRL, Channels 15-00)	
0xFFFF4_4028	Reserved			
0xFFFF4_402C	Reserved		eDMA Error Low (EDMA_ERL, Channels 15-00)	
0xFFFF4_4030 – 0xFFFF4_40FC	Reserved			
0xFFFF4_4100	eDMA Channel 0 Priority (EDMA_CPR0)	eDMA Channel 1 Priority (EDMA_CPR1)	eDMA Channel 2 Priority (EDMA_CPR2)	eDMA Channel 3 Priority (EDMA_CPR3)
0xFFFF4_4104	eDMA Channel 4 Priority (EDMA_CPR4)	eDMA Channel 5 Priority (EDMA_CPR5)	eDMA Channel 6 Priority (EDMA_CPR6)	eDMA Channel 7 Priority (EDMA_CPR7)
0xFFFF4_4108	eDMA Channel 8 Priority (EDMA_CPR8)	eDMA Channel 9 Priority (EDMA_CPR9)	eDMA Channel 10 Priority (EDMA_CPR10)	eDMA Channel 11 Priority (EDMA_CPR11)
0xFFFF4_410C	eDMA Channel 12 Priority (EDMA_CPR12)	eDMA Channel 13 Priority (EDMA_CPR13)	eDMA Channel 14 Priority (EDMA_CPR14)	eDMA Channel 15 Priority (EDMA_CPR15)
0xFFFF4_4110	Reserved			
0xFFFF4_5000 – 0xFFFF4_51FC	TCD00-TCD15			
0xFFFF4_5200	Reserved			

12.3.2 Register Descriptions

This section lists the eDMA registers in address order and describes the registers and their bit fields.

Reading reserved bits in a register will return the value of zero. Writes to reserved bits in a register will be ignored. Reading or writing to a reserved memory location will generate a bus error.

Many of the control registers have a bit width that matches the number of channels implemented in the module, or 16-bits in size.

12.3.2.1 eDMA Control Register (EDMA_CR)

The 32-bit EDMA_CR defines the basic operating configuration of the eDMA.

Arbitration among the channels can be configured to use a fixed priority or a round robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers (see [Section 12.3.2.15, “eDMA Channel n Priority Registers \(EDMA_CPRn\)”](#)). In round-robin arbitration mode, the channel priorities are ignored and the channels are cycled through, from channel 15 down to channel 0, without regard to priority.

Offset: EDMA_BASE + 0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0		ERCA	EDBG	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-2. eDMA Control Register (EDMA_CR)

Table 12-3. EDMA_CR Field Descriptions

Field	Description
0–28, 31	Reserved. Note: Bits 28 and 31 can be read and written; however, writing has no effect other than to set or clear the bits. Reading returns the values written to the bits.
29 ERCA	Enable Round-Robin Channel Arbitration. 0 Fixed-priority arbitration is used for channel selection. 1 Round-robin arbitration is used for channel selection.
30 EDBG	Enable Debug. 0 The assertion of the system debug control input is ignored. 1 The assertion of the system debug control input causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution will resume when either the system debug control input is negated or the EDBG bit is cleared.

12.3.2.2 eDMA Error Status Register (EDMA_ESR)

The EDMA_ESR provides information about the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed-arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count, and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

In fixed-arbitration mode, a configuration error is generated when any two channel priority levels are equal and any channel is activated. The ERRCHN field is undefined for this type of error. All channel priority levels must be unique before any service requests are made.

If a scatter-gather operation is enabled on channel completion, a configuration error is reported if the scatter-gather address (DLAST_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled on channel completion, a configuration error is reported when the link is attempted if the TCD.CITER.E_LINK bit is not equal to the TCD.BITER.E_LINK bit. All configuration error conditions except scatter-gather and minor loop link error are reported as the channel is activated and assert an error interrupt request if enabled. When properly enabled, a scatter-gather configuration error is reported when the scatter-gather operation begins at major loop completion. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is immediately stopped and the appropriate bus error flag is set. In this case, the state of the channel's transfer control descriptor is updated by the DMA engine with the current source address, destination address, and minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write will execute using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence will execute before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the DMA engine to stop the active channel and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA_ESR. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected. After the error status has been updated, the DMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel will execute and terminate with the same error condition.

Offset: EDMA_BASE + 0x0004

Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	CPE	ERRCHN				SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-3. eDMA Error Status Register (EDMA_ESR)

Table 12-4. EDMA_ESR Field Descriptions

Field	Description
VLD	Valid Bit. Logical OR of all EDMA_ERL status bits. 0 No EDMA_ER bits are set. 1 At least one EDMA_ER bit is set indicating a valid error exists that has not been cleared.
bits 1–16	Reserved.
CPE	Channel-Priority Error. 0 No channel-priority error. 1 The last recorded error was a configuration error in the channel priorities, indicating not all channel priorities are unique.
ERRCHN	Error Channel Number. Channel number of the last recorded error (excluding CPE errors). Note: Do not rely on the number in the ERRCHN field for channel-priority errors. Channel-priority errors must be resolved by inspection. The application code must interrogate the priority registers to find channels with duplicate priority level.
SAE	Source Address Error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD.SADDR field, indicating TCD.SADDR is inconsistent with TCD.SSIZE.
SOE	Source Offset Error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.SOFF field, indicating TCD.SOFF is inconsistent with TCD.SSIZE.
DAE	Destination Address Error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD.DADDR field, indicating TCD.DADDR is inconsistent with TCD.DSIZE.
DOE	Destination Offset Error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.DOFF field, indicating TCD.DOFF is inconsistent with TCD.DSIZE.

Table 12-4. EDMA_ESR Field Descriptions (continued)

Field	Description
NCE	NBYTES/CITER Configuration Error. 0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the TCD.NBYTES or TCD.CITER fields, indicating the following conditions exist: <ul style="list-style-type: none"> • TCD.NBYTES is not a multiple of TCD.SSIZE and TCD.DSIZE, or • TCD.CITER is equal to zero, or • TCD.CITER.E_LINK is not equal to TCD.BITER.E_LINK.
SGE	Scatter-Gather Configuration Error. 0 No scatter-gather configuration error. 1 The last recorded error was a configuration error detected in the TCD.DLAST_SGA field, indicating TCD.DLAST_SGA is not on a 32-byte boundary. This field is checked at the beginning of a scatter-gather operation after major loop completion if TCD.E_SG is enabled.
SBE	Source Bus Error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
DBE	Destination Bus Error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

12.3.2.3 eDMA Enable Request Register (EDMA_ERQRL)

The EDMA_ERQRL provides a bit map for the 16 channels to enable the request signal for each channel. EDMA_ERQRL maps to channels 15–0.

The state of any given channel enable is directly affected by writes to this register; the state is also affected by writes to the EDMA_SERQR and EDMA_CERQR. The EDMA_CERQR and EDMA_SERQR are provided so that the request enable for a single channel can be modified without performing a read-modify-write sequence to the EDMA_ERQRL.

Both the eDMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does not affect a channel service request made through software or a linked channel request.

Offset: EDMA_BASE + 0x000E

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ	ERQ
W	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-4. eDMA Enable Request Low Register (EDMA_ERQRL)

Table 12-5. EDMA_ERQRL Field Descriptions

Field	Description
ERQ n	Enable eDMA Hardware Service Request n . 0 The eDMA request signal for channel n is disabled. 1 The eDMA request signal for channel n is enabled.

As a given channel completes processing its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the EDMA_ERQR bit for that channel. If the TCD.D_REQ bit is set, then the corresponding EDMA_ERQR bit is cleared after the major loop is complete, disabling the eDMA hardware request. Otherwise if the D_REQ bit is cleared, the state of the EDMA_ERQR bit is unaffected.

12.3.2.4 eDMA Enable Error Interrupt Register (EDMA_EEIRL)

The EDMA_EEIRL provides a bit map for the 16 channels to enable the error interrupt signal for each channel. EDMA_EEIRL maps to channels 15–0.

The state of any given channel's error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA_SEEIR and EDMA_CEEIR. The EDMA_SEEIR and EDMA_CEEIR are provided so that the error interrupt enable for a single channel can be modified without the performing a read-modify-write sequence to the EDMA_EEIRL.

Both the eDMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted.

Offset: EDMA_BASE + 0x0016

Access: User read/write

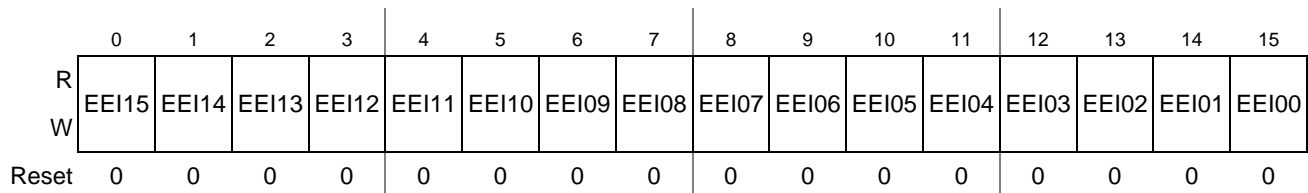


Figure 12-5. eDMA Enable Error Interrupt Low Register (EDMA_EEIRL)

Table 12-6. EDMA_EEIRL Field Descriptions

Field	Description
EEI n	Enable Error Interrupt n . 0 The error signal for channel n does not generate an error interrupt. 1 The assertion of the error signal for channel n generate an error interrupt request.

12.3.2.5 eDMA Set Enable Request Register (EDMA_SERQR)

The EDMA_SERQR provides a simple memory-mapped mechanism to set a given bit in the EDMA_ERQRL to enable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be set. Setting bit 1 (SERQ[0]) provides a global set

function, forcing the entire contents of EDMA_ERQRL to be asserted. Reads of this register return all zeroes.

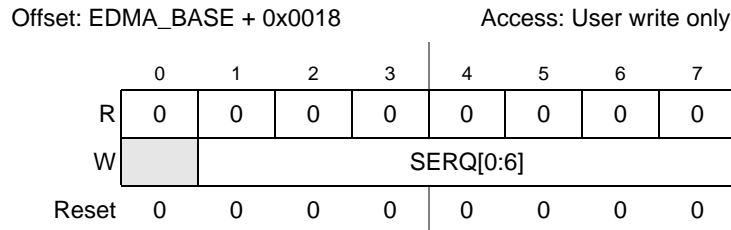


Figure 12-6. eDMA Set Enable Request Register (EDMA_SERQR)

Table 12-7. EDMA_SERQR Field Descriptions

Field	Descriptions
bit 0	Reserved
SERQ[0:6]	Set Enable Request. 0–15 Set corresponding bit in EDMA_ERQRL 16–63 Reserved 64–127 Set all bits in EDMA_ERQRL Note: Bits 2 and 3(SERQR[1:2]) are not used.

12.3.2.6 eDMA Clear Enable Request Register (EDMA_CERQR)

The EDMA_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERQRL to disable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be cleared. Setting bit 1 (CERQ[0]) provides a global clear function, forcing the entire contents of the EDMA_ERQRL to be zeroed, disabling all eDMA request inputs. Reads of this register return all zeroes.

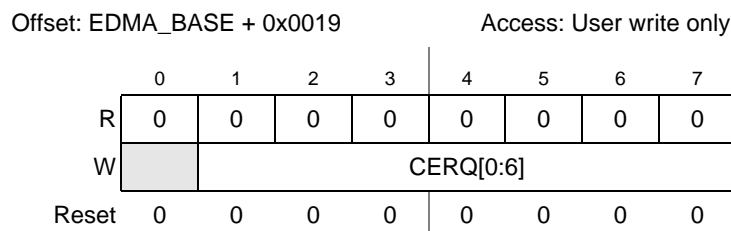


Figure 12-7. eDMA Clear Enable Request Register (EDMA_CERQR)

Table 12-8. EDMA_CERQR Field Descriptions

Field	Description
bit 0	Reserved.
CERQ[0:6]	Clear Enable Request. 0–15 Clear corresponding bit in EDMA_ERQRL 16–63 Reserved 64–127 Clear all bits in EDMA_ERQRL Note: Bits 2 and 3(CERQR[1:2]) are not used.

12.3.2.7 eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

The EDMA_SEEIR provides a memory-mapped mechanism to set a given bit in the EDMA_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be set. Setting bit 1 (SEEI[0]) provides a global set function, forcing the entire contents of EDMA_EEIRL to be asserted. Reads of this register return all zeroes.

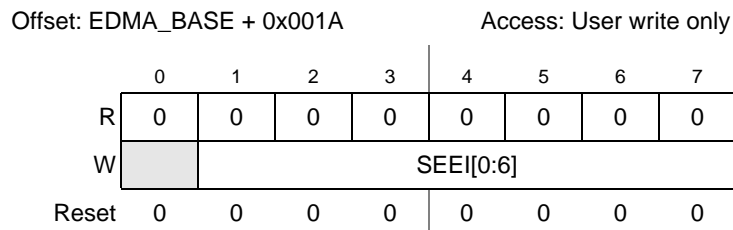


Figure 12-8. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

Table 12-9. EDMA_SEEIR Field Descriptions

Field	Description
bit 0	Reserved.
SEEI[0:6]	Set Enable Error Interrupt. 0–15 Set corresponding bit in EDMA_EIRRL 16–63 Reserved 64–127 Set all bits in EDMA_EEIRL Note: Bits 2 and 3(SEEIRR[1:2]) are not used.

12.3.2.8 eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

The EDMA_CEEIR provides a memory-mapped mechanism to clear a given bit in the EDMA_EEIRL to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be cleared. Setting bit 1 (CEEI[0]) provides a global clear function, forcing the entire contents of the EDMA_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register return all zeroes.

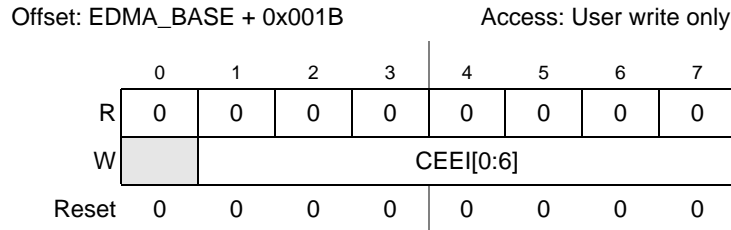


Figure 12-9. eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

Table 12-10. EDMA_CEEIR Field Descriptions

Field	Description
bit 0	Reserved.
CEEI[0:6]	Clear Enable Error Interrupt. 0–15 Clear corresponding bit in EDMA_EEIRL 16–63 Reserved 64–127 Clear all bits in EDMA_EEIRL Note: Bits 2 and 3(CEEIR[1:2]) are not used.

12.3.2.9 eDMA Clear Interrupt Request Register (EDMA_CIRQR)

The EDMA_CIRQR provides a memory-mapped mechanism to clear a given bit in the EDMA_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_IRQRL to be cleared. Setting bit 1 (CINT[0]) provides a global clear function, forcing the entire contents of the EDMA_IRQRL to be zeroed, disabling all eDMA interrupt requests. Reads of this register return all zeroes.

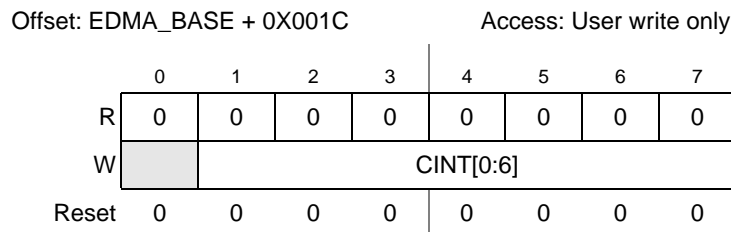


Figure 12-10. eDMA Clear Interrupt Request (EDMA_CIRQR)

Table 12-11. EDMA_CIRQR Field Descriptions

Field	Description
bit 0	Reserved.
CINT[0:6]	Clear Interrupt Request. 0–15 Clear corresponding bit in EDMA_IRQRL 16–63 Reserved 64–127 Clear all bits in EDMA_IRQRL Note: Bits 2 and 3(CIRQR[1:2]) are not used.

12.3.2.10 eDMA Clear Error Register (EDMA_CER)

The EDMA_CER provides a memory-mapped mechanism to clear a given bit in the EDMA_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_ERL to be cleared. Setting bit 1 (CERR[0]) provides a global clear function, forcing the entire contents of the EDMA_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes.

Offset: EDMA_BASE + 0x001D				Access: User write only				
	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	CERR[0:6]							
Reset	0	0	0	0	0	0	0	0

Figure 12-11. eDMA Clear Error Register (EDMA_CER)

Table 12-12. EDMA_CER Field Descriptions

Field	Description
bit 0	Reserved.
CERR[0:6]	Clear Error Indicator. 0–15 Clear corresponding bit in EDMA_ERL 16–63 Reserved 64–127 Clear all bits in EDMA_ERL Note: Bits 2 and 3(CER[1:2]) are not used.

12.3.2.11 eDMA Set START Bit Register (EDMA_SSBR)

The EDMA_SSBR provides a memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding transfer control descriptor to be set. Setting bit 1 (SSB[0]) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes.

Offset: EDMA_BASE + 0x001E				Access: User write only				
	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	SSB[0:6]							
Reset	0	0	0	0	0	0	0	0

Figure 12-12. eDMA Set START Bit Register (EDMA_SSBR)

Table 12-13. EDMA_SSBR Field Descriptions

Field	Description
bit 0	Reserved.
SSB[0:6]	Set START Bit (channel service request). 0–15 Set the corresponding channel's TCD START bit 16–63 Reserved 64–127 Set all TCD START bits Note: Bits 2 and 3(SSBR[1:2]) are not used.

12.3.2.12 eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

The EDMA_CDSBR provides a memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting bit 1 (CDSB[0]) provides a global clear function, forcing all DONE bits to be cleared.

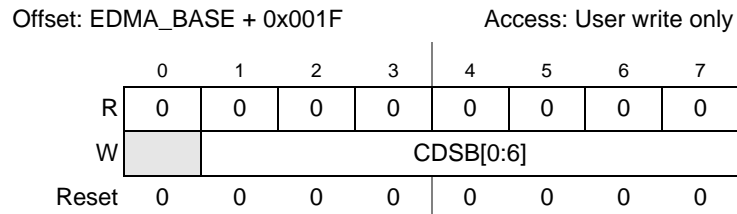


Figure 12-13. eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

Table 12-14. EDMA_CDSBR Field Descriptions

Field	Description
bit 0	Reserved.
CDSB[0:6]	Clear DONE Status Bit. 0–15 Clear the corresponding channel's DONE bit 16–63 Reserved 64–127 Clear all TCD DONE bits Note: Bits 2 and 3(CDSBR[1:2]) are not used.

12.3.2.13 eDMA Interrupt Request Register (EDMA_IRQRL)

The EDMA_IRQRL provides a bit map for the 16 channels signaling the presence of an interrupt request for each channel. EDMA_IRQRL maps to channels 15–0.

The DMA engine signals the occurrence of a programmed interrupt on the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, software must clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA_CIRQR in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA_CIRQR. On writes to the EDMA_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A 0 in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA_CIRQR is provided so the interrupt request for a single channel can be cleared without performing a read-modify-write sequence to the EDMA_IRQRL.

Offset: EDMA_BASE + 0x0026

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT	INT
W	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-14. eDMA Interrupt Request Low Register (EDMA_IRQRL)

Table 12-15. EDMA_IRQRL Field Descriptions

Field	Description
INT n	eDMA Interrupt Request n . 0 The interrupt request for channel n is cleared. 1 The interrupt request for channel n is active.

12.3.2.14 eDMA Error Register (EDMA_ERL)

The EDMA_ERL provides a bit map for the 16 channels signaling the presence of an error for each channel. EDMA_ERL maps to channels 15-0.

The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA_EEIR, then logically summed across 16 channels to form an error interrupt request, which is then routed to the interrupt controller. During the execution of the interrupt service routine associated with any eDMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the EDMA_CER in the interrupt service routine is used for this purpose. The normal eDMA channel completion indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the EDMA_EEIR. The EDMA_ESR[VLD] bit is a logical OR of all bits in this register and it provides a single bit indication of any errors. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the EDMA_CER. On writes to EDMA_ERL, a 1 in any bit position clears the corresponding channel's error status. A 0 in any bit position has no affect on the corresponding channel's current error status. The EDMA_CER is provided so the error indicator for a single channel can be cleared.

Offset: EDMA_BASE + 0x002E

Access: User read/write

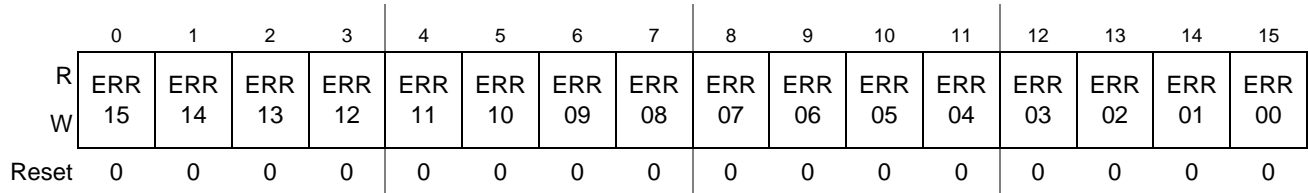


Figure 12-15. eDMA Error Low Register (EDMA_ERL)

Table 12-16. EDMA_ERL Field Descriptions

Field	Description
ERR n	eDMA Error n . 0 An error in channel n has not occurred. 1 An error in channel n has occurred.

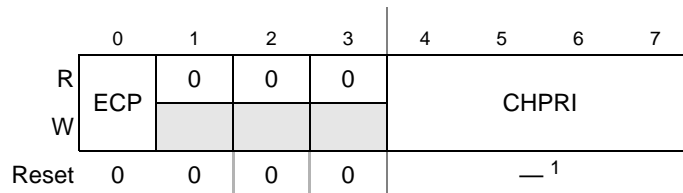
12.3.2.15 eDMA Channel n Priority Registers (EDMA_CPR n)

When the fixed-priority channel arbitration mode is enabled (EDMA_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software modifies channel priority values, then the software must ensure that the channel priorities contain unique values, otherwise a configuration error will be reported. The range of the priority value is limited to the values of 0 through 15. See Figure 12-2 and Table 12-3 for the EDMA_CR definition.

Channel preemption is enabled on a per-channel basis by setting the ECP bit in the EDMA_CPR n register. Channel preemption allows the executing channel’s data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel requests service, the restored channel will be suspended and the higher priority channel will be serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is available only when fixed arbitration is selected for channel arbitration mode.

Offset: EDMA_BASE + 0x0100 + n

Access: User read/write



¹ The reset value for the channel priority field, CHPRI[0–3], is equal to the corresponding channel number for each priority register; that is, EDMA_CPR0[CHPRI] = 0b0000 and EDMA_CPR15[CHPRI] = 0b1111.

Figure 12-16. eDMA Channel n Priority Register (EDMA_CPR n)

Table 12-17. EDMA_CPR n Field Descriptions

Field	Description
ECP	Enable Channel Preemption. 0 Channel n cannot be suspended by a higher priority channel's service request. 1 Channel n can be temporarily suspended by the service request of a higher priority channel.
bits 1–3	Reserved.
CHPRI	Channel n Arbitration Priority. Channel priority when fixed-priority arbitration is enabled. The reset value for the channel priority fields CHPRI[0–3], is equal to the corresponding channel number for each priority register; that is, EDMA_CPR31[CHPRI] = 0b1111.

12.3.2.16 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 15. The definitions of the TCD are presented as eight 32-bit values. Table 12-18 is a field list of the basic TCD structure.

Table 12-18. TCD n 32-bit Memory Structure

eDMA Offset	TCD n Field	
0x1000+(32 x n)+0x0000	Source address (saddr)	
0x1000+(32 x n)+0x0004	Transfer attributes	Signed source address offset (soff)
0x1000+(32 x n)+0x0008	Inner minor byte count (nbytes)	
0x1000+(32 x n)+0x000C	Last source address adjustment (slast)	
0x1000+(32 x n)+0x0010	Destination address (daddr)	
0x1000+(32 x n)+0x0014	Current major iteration count (citer)	Signed destination address offset (doff)
0x1000 (32 x n) 0x0018	Last destination address adjustment / scatter-gather address (dlast_sga)	
0x1000+(32 x n)+0x001c	Beginning major iteration count (biter)	Channel control/status

Figure 12-17 and Table 12-19 define the fields of the TCD n structure.

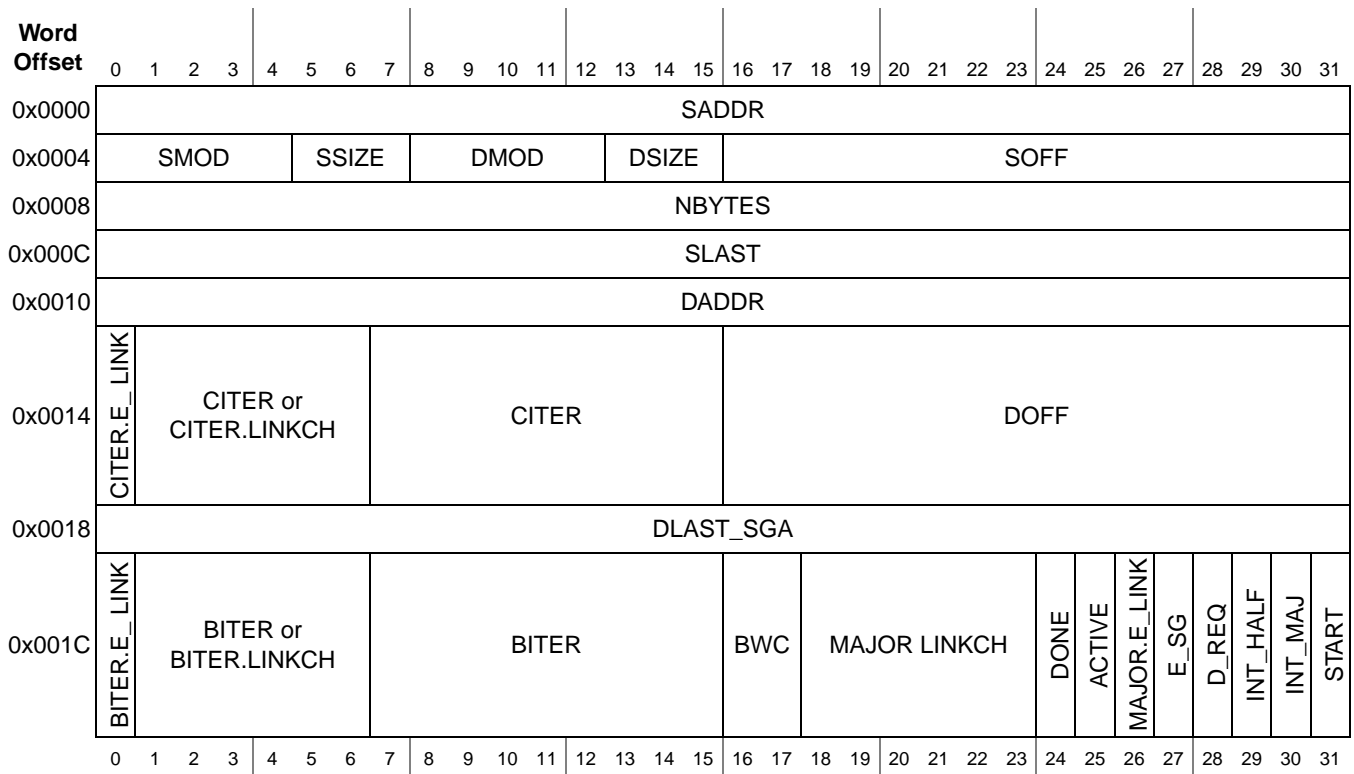


Figure 12-17. TCD Structure

NOTE

The TCD structures for the eDMA channels shown in [Figure 12-17](#) are implemented in internal SRAM. These structures are not initialized at reset; therefore, all channel TCD parameters must be initialized by the application code before activating that channel.

Table 12-19. TCD_n Field Descriptions

Bits / Word Offset [n:n]	Name	Description
0–31 / 0x0 [0:31]	SADDR [0:31]	Source address. Memory address pointing to the source data. Word 0x0, bits 0–31.
32–36 / 0x4 [0:4]	SMOD [0:4]	Source address modulo. 0 Source address modulo feature is disabled. non-0 This value defines a specific address range that is specified to be the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range.

Table 12-19. TCDn Field Descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
37–39 / 0x4 [5:7]	SSIZE [0:2]	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 Reserved 100 16-byte (32-bit, 4-beat, WRAP4 burst) 101 32-byte (32-bit, 8 beat, WRAP8 burst) 110 Reserved 111 Reserved The attempted specification of a reserved encoding causes a configuration error.
40–44 / 0x4 [8:12]	DMOD [0:4]	Destination address modulo. See the SMOD[0:5] definition.
45–47 / 0x4 [13:15]	DSIZE [0:2]	Destination data transfer size. See the SSIZE[0:2] definition.
48–63 / 0x4 [16:31]	SOFF [0:15]	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.
64–95 / 0x8 [0:31]	NBYTES [0:31]	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the DMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. After the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. Note: The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GB transfer.
96–127 / 0xC [0:31]	SLAST [0:31]	Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.
128–159 / 0x10 [0:31]	DADDR [0:31]	Destination address. Memory address pointing to the destination data.
160 / 0x14 [0]	CITER.E_LINK	Enable channel-to-channel linking on minor loop completion. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. Note: This bit must be equal to the BITER.E_LINK bit otherwise a configuration error will be reported.

Table 12-19. TCDn Field Descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
161–166 / 0x14 [1:6]	CITER [0:5] or CITER.LINKCH [0:5]	Current major iteration count or link channel number. If channel-to-channel linking is disabled (TCD.CITER.E_LINK = 0), then <ul style="list-style-type: none"> No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] are used to form a 15-bit CITER field. Otherwise, <ul style="list-style-type: none"> After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel's TCD.START bit.
167–175 / 0x14 [7:15]	CITER [6:14]	Current major iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field. Note: When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field. Note: If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.
176–191 / 0x14 [16:31]	DOFF [0:15]	Destination address signed Offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.
192–223 / 0x18 [0:31]	DLAST_SGA [0:31]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter-gather). If scatter-gather processing for the channel is disabled (TCD.E_SG = 0) then <ul style="list-style-type: none"> Adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to restore the destination address to the initial value, or adjust the address to reference the next data structure. Otherwise, <ul style="list-style-type: none"> This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter-gather address must be 0-modulo-32 byte, otherwise a configuration error is reported.
224 / 0x1C [0]	BITER.E_LINK	Enables channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. Note: When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.

Table 12-19. TCDn Field Descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
225–230 / 0x1C [1:6]	BITER [0:5] or BITER.LINKCH[0:5]	Starting major iteration count or link channel number. If channel-to-channel linking is disabled (TCD.BITER.E_LINK = 0), then <ul style="list-style-type: none"> No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] are used to form a 15-bit BITER field. Otherwise, <ul style="list-style-type: none"> After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel's TCD.START bit. Note: When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
231–239 / 0x1C [7:15]	BITER [6:14]	Starting major iteration count. As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field are reloaded into the CITER field. Note: If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.
240–241 / 0x1C [16:17]	BWC [0:1]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR). 00 No DMA engine stalls 01 Reserved 10 DMA engine stalls for 4 cycles after each r/w 11 DMA engine stalls for 8 cycles after each r/w
242–247 / 0x1C [18:23]	MAJOR.LINKCH [0:5]	Link channel number. If channel-to-channel linking on major loop complete is disabled (TCD.MAJOR.E_LINK = 0) then, <ul style="list-style-type: none"> No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise <ul style="list-style-type: none"> After the major loop counter is exhausted, the DMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's TCD.START bit.
248 / 0x1C [24]	DONE	Channel done. This flag indicates the eDMA has completed the outer major loop. It is set by the DMA engine as the CITER count reaches zero; it is cleared by software or hardware when the channel is activated (when the DMA engine has begun processing the channel, not when the first data transfer occurs). Note: This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.
249 / 0x1C [25]	ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the DMA engine as the inner minor loop completes or if any error condition is detected.

Table 12-19. TCDn Field Descriptions (continued)

Bits / Word Offset [n:n]	Name	Description
250 / 0x1C [26]	MAJOR.E_LINK	Enable channel-to-channel linking on major loop completion. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. NOTE: To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
251 / 0x1C [27]	E_SG	Enable scatter-gather processing. As the channel completes the outer major loop, this flag enables scatter-gather processing in the current channel. If enabled, the DMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. NOTE: To support the dynamic scatter-gather coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The current channel's TCD is normal format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.
252 / 0x1C [28]	D_REQ	Disable hardware request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQL bit when the current major iteration count reaches zero. 0 The channel's EDMA_ERQL bit is not affected. 1 The channel's EDMA_ERQL bit is cleared when the outer major loop is complete.
253 / 0x1C [29]	INT_HALF	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQL when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER >> 1)). This halfway point interrupt request is provided to support double-buffered (aka ping-pong) schemes, or other types of data movement where the processor needs an early indication of the transfer's progress. CITER = BITER = 1 with INT_HALF enabled will generate an interrupt as it satisfies the equation (CITER == (BITER >> 1)) after a single activation. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
254 / 0x1C [30]	INT_MAJ	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQL when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
255 / 0x1C [31]	START	Channel start. If this flag is set the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

12.4 Functional Description

This section provides an overview of the microarchitecture and functional operation of the eDMA block.

The eDMA module is partitioned into two major modules: the DMA engine and the transfer control descriptor local memory. The DMA engine is further partitioned into four submodules, which are detailed below.

- DMA engine
 - Address path: This module implements registered versions of two channel transfer control descriptors: channel x and channel y, and is responsible for all the master bus address calculations. All the implemented channels provide the same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by EDMA_CPR n [ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.

When another channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other address path channel{x,y}. After the inner minor loop completes execution, the address path hardware writes the new values for the TCD n .{SADDR, DADDR, CITER} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCD n .CITER field, and a possible fetch of the next TCD n from memory as part of a scatter-gather operation.
 - Data path: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The system read data bus is the primary input, and the system write data bus is the primary output.

The address and data path modules directly support the two-stage pipelined system bus. The address path module represents the 1st stage of the bus pipeline (the address phase), while the data path module implements the second stage of the pipeline (the data phase).
 - Program model/channel arbitration: This module implements the first section of eDMA's programming model and also the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request outputs are also connected to this module (via the control logic).
 - Control: This module provides all the control functions for the DMA engine. For data transfers where the source and destination sizes are equal, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner minor loop byte count has been moved.

A minor loop interaction is defined as the number of bytes to transfer (n bytes) divided by the transfer size. Transfer size is defined as:

```

if (SSIZE < DSIZE)
  transfer size = destination transfer size (# of bytes)
else
  transfer size = source transfer size (# of bytes)

```

Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, AND START. Major loop TCD variables are DLAST, SLAST, CITER, BITER, DONE, D_REQ, INT_MAJ, MAJOR_LNKCH, and INT_HALF.

For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. For example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- TCD local memory
 - Memory controller: This logic implements the required dual-ported controller, handling accesses from both the DMA engine as well as references from the slave bus. As noted earlier, in the event of simultaneous accesses, the DMA engine is given priority and the slave transaction is stalled. The hooks to a BIST controller for the local TCD memory are included in this module.
 - Memory array: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

12.4.1 eDMA Basic Data Flow

The eDMA transfers data based on a two-deep, nested flow. The basic flow of a data transfer can be partitioned into three segments. As shown in [Figure 12-18](#), the first segment involves the channel service request. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel n . Channel service request via software and the TCDn.START bit follows the same basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed to through the DMA engine, first through the control module, then into the program model/channel arbitration module. In the next cycle, the channel arbitration is performed using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the DMA engine address path channel{x,y} registers. The TCD memory is organized 64-bits in width to minimize the time needed to fetch the activated channel's descriptor and load it into the eDMA engine address path channel{x,y} registers.

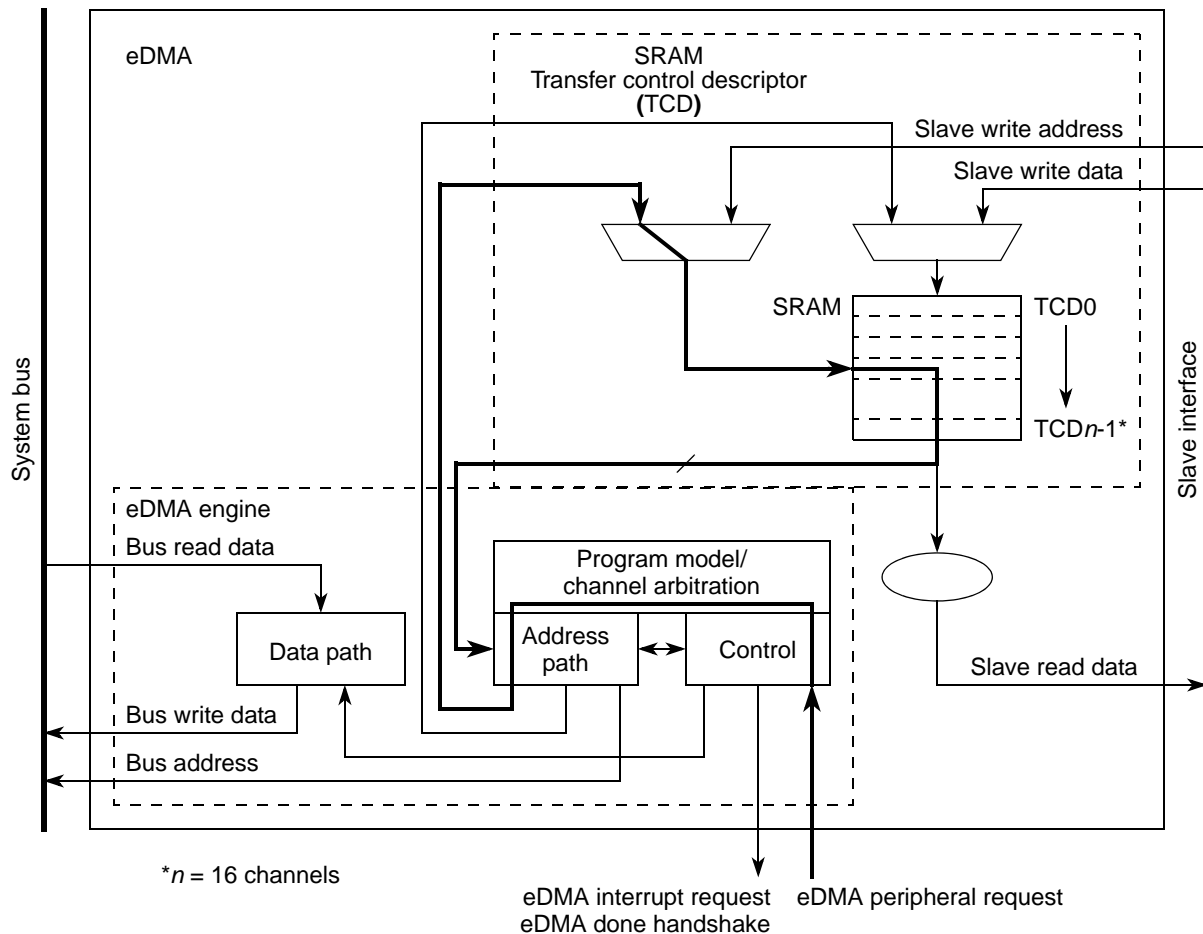


Figure 12-18. eDMA Operation, Part 1

In the second part of the basic data flow as shown in [Figure 12-19](#), the modules associated with the data transfer (address path, data path, and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the system bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA done handshake signal is asserted at the end of the minor byte count transfer.

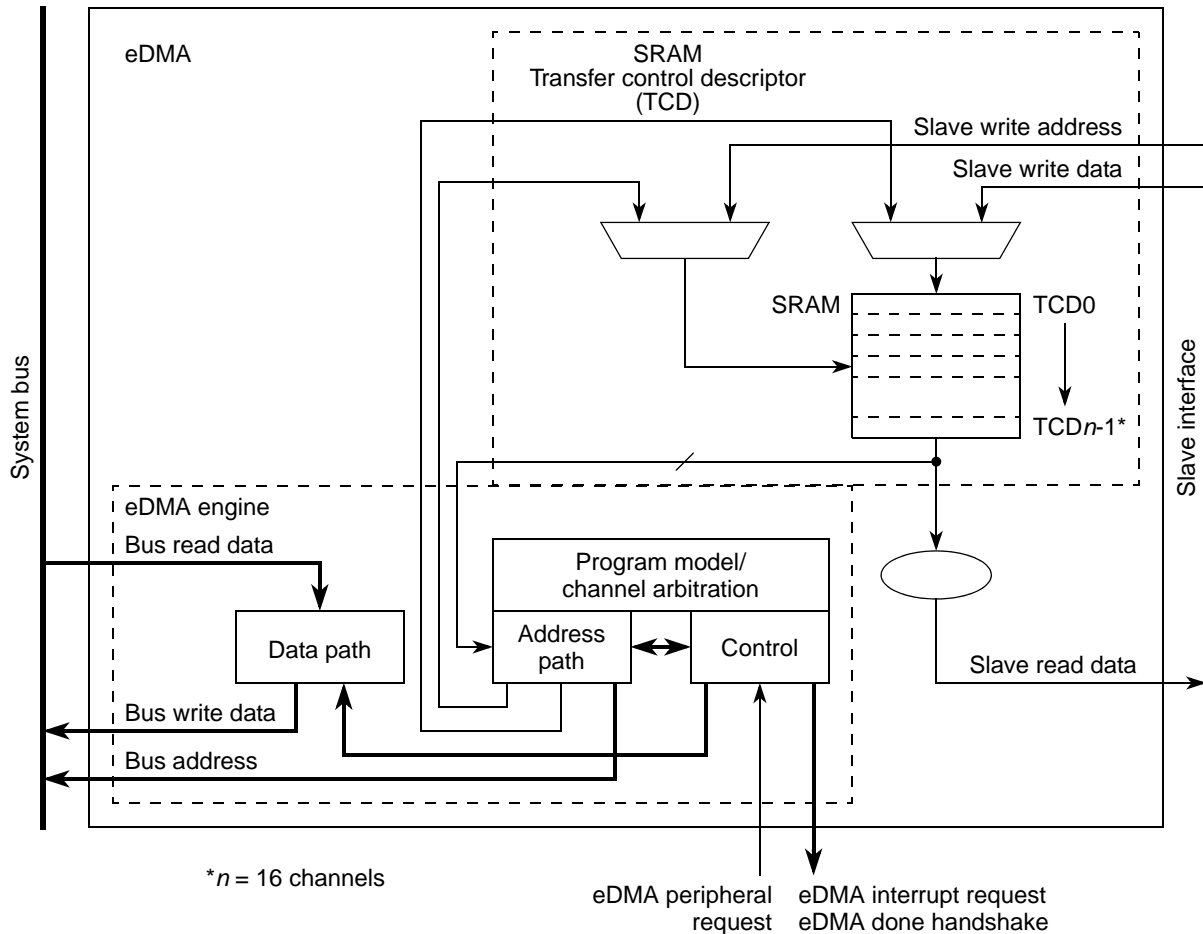


Figure 12-19. eDMA Operation, Part 2

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel's TCD; for example, SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then there are additional operations performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter-gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in [Figure 12-20](#).

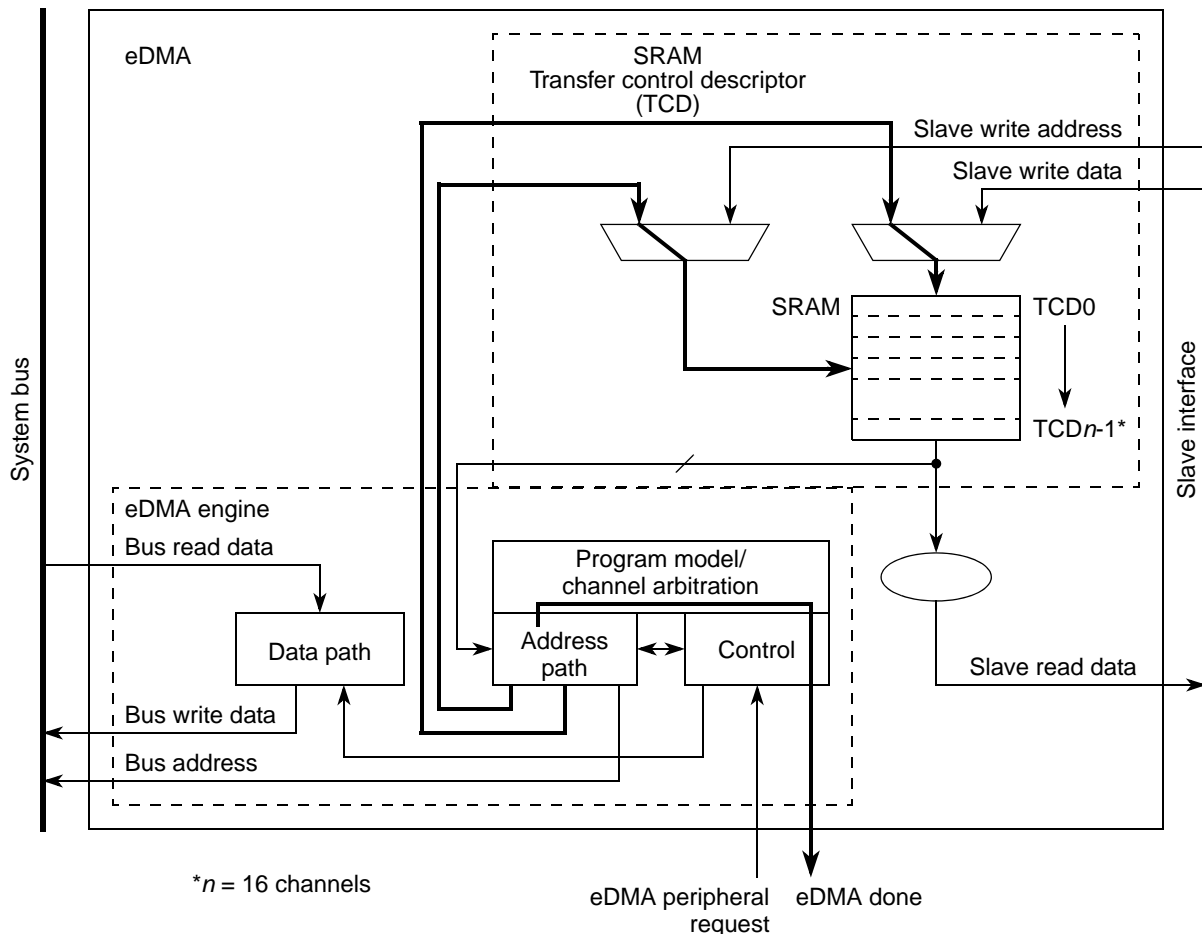


Figure 12-20. eDMA Operation, Part 3

12.5 Initialization / Application Information

12.5.1 eDMA Initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA_CPR n registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_EEIRL and/or EDMA_EEIRH registers if desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the EDMA_ERQRH and/or EDMA_ERQRL registers.
6. Request channel service by software (setting the TCD.START bit) or by hardware (slave device asserting its DMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA engine will read the entire TCD, including the

primary transfer control parameter shown in [Table 12-20](#), for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the system bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.SADDR) to the destination (as defined by the destination address, TCD.DADDR) continue until the specified number of bytes (TCD.NBYTES) have been transferred. When the transfer is complete, the DMA engine's local TCD.SADDR, TCD.DADDR, and TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed; for example, interrupts, major loop channel linking, and scatter-gather operations, if enabled.

Table 12-20. TCD Primary Control and Status Fields

TCD Field Name	Description
START	Control bit to start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service
BWC	Control bits for throttling bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

[Figure 12-21](#) shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).

Example memory array			Current major loop iteration count (CITER)		
DMA request		Minor loop	Major loop	3	
	⋮				
DMA request		Minor loop		Major loop	2
	⋮				
DMA request		Minor loop			Major loop
	⋮				

Figure 12-21. Example of Multiple Loop Iterations

Figure 12-22 lists the memory array terms and how the TCD settings interrelate.

xADDR: (Starting address)	xSIZE: (Size of one data transfer)	Minor loop (NBYTES in minor loop, often the same value as xSIZE)	Offset (xOFF): Number of bytes added to current address after each transfer (Often the same value as xSIZE)
⋮	⋮	Minor loop	Each DMA source (S) and destination (D) has its own: <ul style="list-style-type: none"> • Address (xADDR) • Size (xSIZE) • Offset (xOFF) • Modulo (xMOD) • Last address adjustment (xLAST) where x = S or D
xLAST: Number of bytes added to current address after major loop (typically used to loop back)	⋮	Last minor loop	Peripheral queues typically have size and offset equal to NBYTES

Figure 12-22. Memory Array Terms

12.5.2 DMA Programming Errors

The DMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per-channel basis with the exception of channel-priority error, or EDMA_ESR[CPE].

For all error types other than channel-priority errors, the channel number causing the error is recorded in the EDMA_ESR. If the error source is not removed before the next activation of the problem channel, the error will be detected and recorded again.

If priority levels are not unique, the highest (channel) priority that has an active request is selected, but the lowest numbered (channel) with that priority is selected by arbitration and executed by the DMA engine. The hardware service request handshake signals, error interrupts, and error reporting are associated with the selected channel.

12.5.3 DMA Request Assignments

The assignments between the DMA requests from the modules to the channels of the eDMA are shown in [Table 12-21](#). The source column is written in C language syntax. The syntax is `module_instance.register[bit]`.

Table 12-21. DMA Request Summary for eDMA

DMA Request	Channel	Source	Description
DMA_MUX_CHCONFIG0_SOURCE	0	DMA_MUX.CHCONFIG0[SOURCE]	DMA MUX channel 0 source
DMA_MUX_CHCONFIG1_SOURCE	1	DMA_MUX.CHCONFIG1[SOURCE]	DMA MUX channel 1 source
DMA_MUX_CHCONFIG2_SOURCE	2	DMA_MUX.CHCONFIG2[SOURCE]	DMA MUX channel 2 source
DMA_MUX_CHCONFIG3_SOURCE	3	DMA_MUX.CHCONFIG3[SOURCE]	DMA MUX channel 3 source
DMA_MUX_CHCONFIG4_SOURCE	4	DMA_MUX.CHCONFIG4[SOURCE]	DMA MUX channel 4 source
DMA_MUX_CHCONFIG5_SOURCE	5	DMA_MUX.CHCONFIG5[SOURCE]	DMA MUX channel 5 source
DMA_MUX_CHCONFIG6_SOURCE	6	DMA_MUX.CHCONFIG6[SOURCE]	DMA MUX channel 6 source
DMA_MUX_CHCONFIG7_SOURCE	7	DMA_MUX.CHCONFIG7[SOURCE]	DMA MUX channel 7 source
DMA_MUX_CHCONFIG8_SOURCE	8	DMA_MUX.CHCONFIG8[SOURCE]	DMA MUX channel 8 source
DMA_MUX_CHCONFIG9_SOURCE	9	DMA_MUX.CHCONFIG9[SOURCE]	DMA MUX channel 9 source
DMA_MUX_CHCONFIG10_SOURCE	10	DMA_MUX.CHCONFIG10[SOURCE]	DMA MUX channel 10 source
DMA_MUX_CHCONFIG11_SOURCE	11	DMA_MUX.CHCONFIG11[SOURCE]	DMA MUX channel 11 source
DMA_MUX_CHCONFIG12_SOURCE	12	DMA_MUX.CHCONFIG12[SOURCE]	DMA MUX channel 12 source
DMA_MUX_CHCONFIG13_SOURCE	13	DMA_MUX.CHCONFIG13[SOURCE]	DMA MUX channel 13 source
DMA_MUX_CHCONFIG14_SOURCE	14	DMA_MUX.CHCONFIG14[SOURCE]	DMA MUX channel 14 source
DMA_MUX_CHCONFIG15_SOURCE	15	DMA_MUX.CHCONFIG15[SOURCE]	DMA MUX channel 15 source

12.5.4 DMA Arbitration Mode Considerations

12.5.4.1 Fixed-Channel Arbitration

In this mode, the channel service request from the highest priority channel is selected to execute. Preemption is available in this scenario only.

12.5.4.2 Round-Robin Channel Arbitration

In this mode, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the assigned channel priority levels.

12.5.5 DMA Transfer

12.5.5.1 Single Request

To perform a simple transfer of n bytes of data with one activation, set the major loop to 1 (TCD.CITER = TCD.BITER = 1). The data transfer will begin after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, the TCD.DONE bit will be set and an interrupt will be generated if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

```
TCD.CITER = TCD.BITER = 1
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = -16
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA = -16
TCD.INT_MAJ = 1
TCD.START = 1 (Must be written last after all other fields have been initialized)
All other TCD fields = 0
```

This would generate the following sequence of events:

1. Slave write to the TCD.START bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
 - a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
 - b) write_word(0x2000) → first iteration of the minor loop
 - c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
 - d) write_word(0x2004) → second iteration of the minor loop
 - e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
 - f) write_word(0x2008) → third iteration of the minor loop

- g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
- h) write_word(0x200c) → last iteration of the minor loop → major loop complete
- 6. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 1 (TCD.BITER).
- 7. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR_n = 1.
- 8. The channel retires.

The eDMA goes idle or services the next channel.

12.5.5.2 Multiple Requests

The next example is the same as previous, excepting transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the EDMA_ERQR, channel service requests are initiated by the slave device (ERQR should be set after TCD). Note that TCD.START = 0.

```
TCD.CITER = TCD.BITER = 2
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = -32
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA = -32
TCD.INT_MAJ = 1
TCD.START = 0 (Must be written last after all other fields have been initialized)
All other TCD fields = 0
```

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
 - a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
 - b) write_word(0x2000) → first iteration of the minor loop
 - c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
 - d) write_word(0x2004) → second iteration of the minor loop

- e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
 - f) write_word(0x2008) → third iteration of the minor loop
 - g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
 - h) write_word(0x200c) → last iteration of the minor loop
6. eDMA engine writes: TCD.SADDR = 0x1010, TCD.DADDR = 0x2010, TCD.CITER = 1.
 7. eDMA engine writes: TCD.ACTIVE = 0.
 8. The channel retires → one iteration of the major loop.

The eDMA goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers are executed as follows:
 - a) read_byte(0x1010), read_byte(0x1011), read_byte(0x1012), read_byte(0x1013)
 - b) write_word(0x2010) → first iteration of the minor loop
 - c) read_byte(0x1014), read_byte(0x1015), read_byte(0x1016), read_byte(0x1017)
 - d) write_word(0x2014) → second iteration of the minor loop
 - e) read_byte(0x1018), read_byte(0x1019), read_byte(0x101a), read_byte(0x101b)
 - f) write_word(0x2018) → third iteration of the minor loop
 - g) read_byte(0x101c), read_byte(0x101d), read_byte(0x101e), read_byte(0x101f)
 - h) write_word(0x201c) → last iteration of the minor loop → major loop complete
14. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 2 (TCD.BITER).
15. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR_n = 1.
16. The channel retires → major loop complete.

The eDMA goes idle or services the next channel.

12.5.5.3 Modulo Feature

The modulo feature of the eDMA provides the ability to implement a circular data queue in which the size of the queue is a power of two. MOD is a 5-bit bitfield for both the source and destination in the TCD and specifies which lower address bits are allowed to increment from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. A setting of 0 for this field disables the modulo feature.

Table 12-22 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the mod field is set to 4, allowing for a 2⁴ byte (16-byte) size queue.

Table 12-22. Modulo Feature Example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

12.5.6 TCD Status

12.5.6.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.CITER field and test for a change. Another method may be extracted from the sequence below. The second method is to test the TCD.START bit AND the TCD.ACTIVE bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.START was written to a 1. Polling the TCD.ACTIVE bit may be inconclusive because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.START = 1, TCD.ACTIVE = 0, TCD.DONE = 0 (channel service request via software).
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing).
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel has completed the major loop and is idle).

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer's model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. eDMA peripheral request asserts (channel service request via hardware).
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing).
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel has completed the major loop and is idle).

For both activation types, the major loop complete status is explicitly indicated via the TCD.DONE bit.

The TCD.START bit is cleared automatically when the channel begins execution, regardless of how the channel was activated.

12.5.6.2 Active Channel TCD Reads

The eDMA will read back the true TCD.SADDR, TCD.DADDR, and TCD.NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

12.5.6.3 Preemption Status

Preemption is available only when fixed arbitration is selected for channel-arbitration mode. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed-channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCD.ACTIVE bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.ACTIVE bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

12.5.7 Channel Linking

Channel linking (or chaining) is a mechanism in which one channel sets the TCD.START bit of another channel (or itself), thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.CITER.E_LINK field are used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

```
TCD.CITER.E_LINK = 1
TCD.CITER.LINKCH = 0xC
TCD.CITER value = 0x4
TCD.MAJOR.E_LINK = 1
TCD.MAJOR.LINKCH = 0x7
```

will execute as:

1. Minor loop done → set channel 12 TCD.START bit
2. Minor loop done → set channel 12 TCD.START bit

3. Minor loop done → set channel 12 TCD.START bit
4. Minor loop done, major loop done → set channel 7 TCD.START bit

When minor loop linking is enabled (TCD.CITER.E_LINK = 1), the TCD.CITER field uses a nine bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.CITER.E_LINK = 0), the TCD.CITER field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD.CITER.LINKCH field are concatenated onto the CITER value to increase the range of the CITER.

NOTE

After configuration, the TCD.CITER.E_LINK bit and the TCD.BITER.E_LINK bit must be equal or a configuration error will be reported. The CITER and BITER vector widths must be equal to calculate the major loop, halfway done interrupt point.

[Table 12-23](#) summarizes how a DMA channel can link to another DMA channel, i.e, use another channel's TCD, at the end of a loop.

Table 12-23. Channel Linking Parameters

Desired Link Behavior	TCD Control Field Name	Description
Link at end of minor loop	citer.e_link	Enable channel-to-channel linking on minor loop completion (current iteration).
	citer.linkch	Link channel number when linking at end of minor loop (current iteration).
Link at end of major loop	major.e_link	Enable channel-to-channel linking on major loop completion.
	major.linkch	Link channel number when linking at end of major loop.

12.5.8 Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

12.5.8.1 Dynamic Channel Linking and Dynamic Scatter-Gather Operation

Dynamic channel linking and dynamic scatter-gather operation is the process of changing the TCD.MAJOR.E_LINK or TCD.E_SG bits during channel execution. These bits are read from the TCD local memory at the end of channel execution thus allowing the user to enable either feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider a scenario where the user attempts to execute a dynamic channel link by enabling the TCD.MAJOR.E_LINK bit at the same time the eDMA engine is retiring the channel. The TCD.MAJOR.E_LINK would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The following coherency model is recommended when executing a dynamic channel link or dynamic scatter-gather request:

1. Set the TCD.MAJOR.E_LINK bit.
2. Read back the TCD.MAJOR.E_LINK bit
3. Test the TCD.MAJOR.E_LINK request status:
 - a) If the bit is set, the dynamic link attempt was successful.
 - b) If the bit is cleared, the attempted dynamic link did not succeed, the channel was already retiring.

This same coherency model is true for dynamic scatter-gather operations. For both dynamic requests, the TCD local memory controller forces the TCD.MAJOR.E_LINK and TCD.E_SG bits to zero on any writes to a channel's TCD after that channel's TCD.DONE bit is set indicating the major loop is complete.

NOTE

The user must clear the TCD.DONE bit before writing the TCD.MAJOR.E_LINK or TCD.E_SG bits. The TCD.DONE bit is cleared automatically by the eDMA engine after a channel begins execution.

Chapter 13

DMA Channel Mux (DMA_MUX)

13.1 Introduction

The DMA_MUX allows for software selection of 16 out of 64 possible DMA sources. Up to fifty-five of these DMA sources are requests from peripherals, but four of the peripheral sources are reserved and will behave as always disabled sources. Eight sources are always enabled and will generate a DMA request as soon as that source is selected. One source (the default for all channels) is always disabled.

13.1.1 Block Diagram

A simplified block diagram of the DMA_MUX is shown in [Figure 13-1](#).

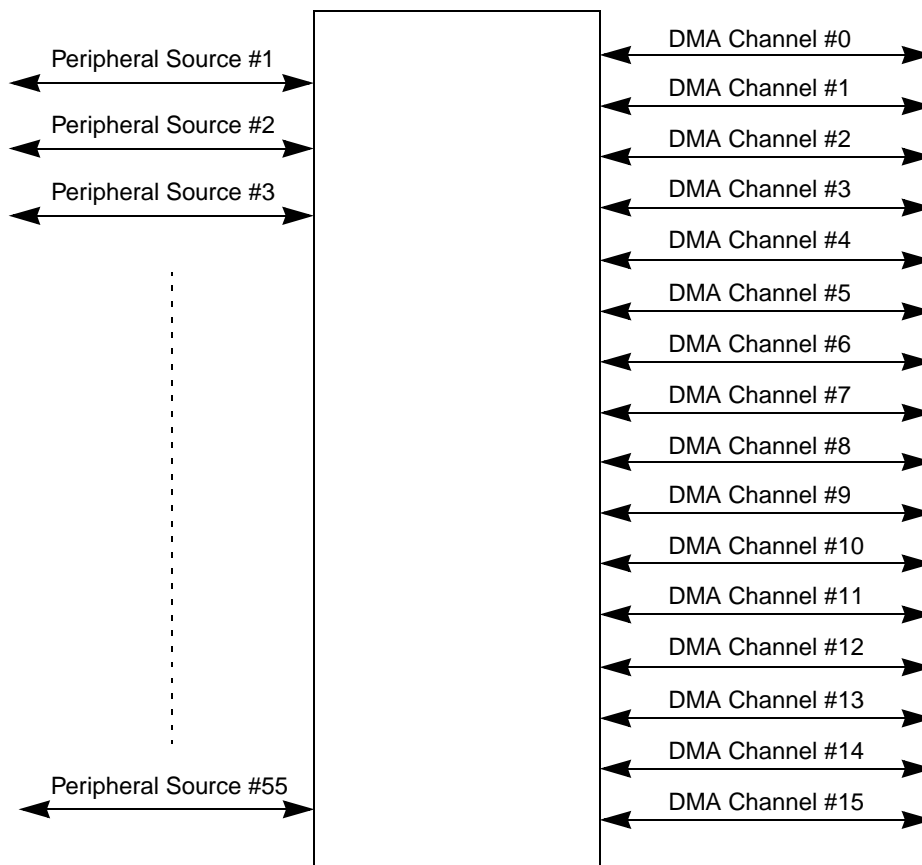


Figure 13-1. DMA_MUX Block Diagram

13.1.2 Features

The DMA_MUX has these major features:

- 16 independently selectable DMA channel routers
 - Eight channels with normal or periodic triggering capability
 - Eight channels with normal operation
 - Each channel router can be assigned to 1 of 55 possible peripheral DMA sources, eight always enabled sources, or one always disabled source.

13.1.3 Modes of Operation

DMA channels 0–7 may be used in the following modes, but channels 8–15 may only be configured to disabled or normal mode.

- Disabled mode

In this mode, the DMA channel is disabled. Because disabling and enabling of DMA channels is done primarily via the DMA registers, this mode is used mainly as the reset state for a DMA channel in the DMA channel mux. It may also be used to temporarily suspend a DMA channel while reconfiguration of the system takes place (changing the period of a DMA trigger, for example).
- Normal mode

In this mode, a DMA source (such as SCI transmit or SCI receive for example) is routed directly to the specified DMA channel. The operation of the DMA_MUX in this mode is completely transparent to the system.
- Periodic trigger mode

In this mode, a DMA source may only request a DMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. Configuration of the period is done in the registers of the periodic interrupt timer.

13.2 External Signal Description

The DMA_MUX has no external signals.

13.3 Memory Map and Registers

This section provides a detailed description of all DMA_MUX registers.

13.3.1 Module Memory Map

The DMA_MUX memory map is shown in [Table 13-1](#). The address of each register is given as an offset to the DMA_MUX base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

All registers are accessible via 8-bit, 16-bit, or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries and 32-bit accesses must be aligned to 32-bit boundaries. As an example,

CHCONFIG0 through CHCONFIG3 are accessible by a 32-bit READ/WRITE to address DMA_MUX_BASE + 0x00, but performing a 32-bit access to address DMA_MUX_BASE + 0x01 is illegal.

Table 13-1. DMA_MUX Memory Map

Offset from DMA_MUX_BASE (0xFFFFD_C000)	Register	Access	Reset Value	Section/Page
0x0000	CHCONFIG0 — Channel #0 configuration	R/W	0x00	13.3.2.1/13-3
0x0001	CHCONFIG1 — Channel #1 configuration	R/W	0x00	13.3.2.1/13-3
0x0002	CHCONFIG2 — Channel #2 configuration	R/W	0x00	13.3.2.1/13-3
0x0003	CHCONFIG3 — Channel #3 configuration	R/W	0x00	13.3.2.1/13-3
0x0004	CHCONFIG4 — Channel #4 configuration	R/W	0x00	13.3.2.1/13-3
0x0005	CHCONFIG5 — Channel #5 configuration	R/W	0x00	13.3.2.1/13-3
0x0006	CHCONFIG6 — Channel #6 configuration	R/W	0x00	13.3.2.1/13-3
0x0007	CHCONFIG7 — Channel #7 configuration	R/W	0x00	13.3.2.1/13-3
0x0008	CHCONFIG8 — Channel #8 configuration	R/W	0x00	13.3.2.1/13-3
0x0009	CHCONFIG9 — Channel #9 configuration	R/W	0x00	13.3.2.1/13-3
0x000A	CHCONFIG10 — Channel #10 configuration	R/W	0x00	13.3.2.1/13-3
0x000B	CHCONFIG11 — Channel #11 configuration	R/W	0x00	13.3.2.1/13-3
0x000C	CHCONFIG12 — Channel #12 configuration	R/W	0x00	13.3.2.1/13-3
0x000D	CHCONFIG13 — Channel #13 configuration	R/W	0x00	13.3.2.1/13-3
0x000E	CHCONFIG14 — Channel #14 configuration	R/W	0x00	13.3.2.1/13-3
0x000F	CHCONFIG15 — Channel #15 configuration	R/W	0x00	13.3.2.1/13-3

13.3.2 Register Descriptions

This section lists the DMA_MUX registers in address order and describes the registers and their bit fields.

13.3.2.1 Channel Configuration Registers (CHCONFIG n)

Each of the 16 DMA channels can be independently enabled/disabled and associated with one of the 64 DMA sources in the system.

Offset: DMA_MUX_BASE + n

Access: User read/write



Figure 13-2. Channel Configuration Registers (CHCONFIG n)

Table 13-2. CHCONFIG n Field Descriptions

Field	Description
ENBL	DMA Channel Enable. ENBL enables the DMA channel. 0 DMA channel is disabled. This mode is primarily used during configuration of the DMA_MUX. The DMA has separate channel enables/disables, which should be used to disable or re-configure a DMA channel. 1 DMA channel is enabled.
TRIG	DMA Channel Trigger Enable (channels 0–7 only). TRIG enables the periodic trigger capability for the DMA channel 0 Triggering is disabled. If triggering is disabled, and the ENBL bit is set, the DMA channel will simply route the specified source to the DMA channel. 1 Triggering is enabled.
SOURCE	DMA Channel Source. SOURCE specifies which DMA source, if any, is routed to a particular DMA channel, according to Table 13-4 .

Table 13-3. Channel and Trigger Enabling

ENBL	TRIG	Function	Mode
0	X	DMA channel is disabled	Disabled mode
1	0	DMA channel is enabled with no triggering (transparent)	Normal mode
1	1	DMA channel is enabled with triggering	Periodic trigger mode

NOTE

Setting multiple CHCONFIG registers with the same DMA source value will result in unpredictable behavior.

Table 13-4. DMA Source Configuration

DMA Request	DMA_MUX Source Input Number	DMA Source	Description
Channel disabled ¹	0x00	Channel disabled	Channel disabled
SCI_A_COMBTX	0x01	SCI_A.SCISR1[TDRE] SCI_A.SCISR1[TC] SCI_A.LINSTAT1[TXRDY]	SCI_A combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_A_COMBRX	0x02	SCI_A.SCISR1[RDRF] SCI_A.LINSTAT1[RXRDY]	SCI_A combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_B_COMBTX	0x03	SCI_B.SCISR1[TDRE] SCI_B.SCISR1[TC] SCI_B.LINSTAT1[TXRDY]	SCI_B combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_B_COMBRX	0x04	SCI_B.SCISR1[RDRF] SCI_B.LINSTAT1[RXRDY]	SCI_B combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_C_COMBTX	0x05	SCI_C.SCISR1[TDRE] SCI_C.SCISR1[TC] SCI_C.LINSTAT1[TXRDY]	SCI_C combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests

Table 13-4. DMA Source Configuration (continued)

DMA Request	DMA_MUX Source Input Number	DMA Source	Description
SCI_C_COMBRX	0x06	SCI_C.SCISR1[RDRF] SCI_C.LINSTAT1[RXRDY]	SCI_C combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_D_COMBTX	0x07	SCI_D.SCISR1[TDRE] SCI_D.SCISR1[TC] SCI_D.LINSTAT1[TXRDY]	SCI_D combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_D_COMBRX	0x08	SCI_D.SCISR1[RDRF] SCI_D.LINSTAT1[RXRDY]	SCI_D combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_E_COMBTX	0x09	SCI_E.SCISR1[TDRE] SCI_E.E.SCISR1[TC] SCI_E.LINSTAT1[TXRDY]	SCI_E combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_E_COMBRX	0x0A	SCI_E.SCISR1[RDRF] SCI_E.LINSTAT1[RXRDY]	SCI_E combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_F_COMBTX	0x0B	SCI_F.SCISR1[TDRE] SCI_F.SCISR1[TC] SCI_F.LINSTAT1[TXRDY]	SCI_F combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_F_COMBRX	0x0C	SCI_F.SCISR1[RDRF] SCI_F.LINSTAT1[RXRDY]	SCI_F combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_G_COMBTX	0x0D	SCI_G.SCISR1[TDRE] SCI_G.SCISR1[TC] SCI_G.LINSTAT1[TXRDY]	SCI_G combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_G_COMBRX	0x0E	SCI_G.SCISR1[RDRF] SCI_G.LINSTAT1[RXRDY]	SCI_G combined DMA request of the receive data register full and LIN receive data ready DMA requests
SCI_H_COMBTX	0x0F	SCI_H.SCISR1[TDRE] SCI_H.SCISR1[TC] SCI_H.LINSTAT1[TXRDY]	SCI_H combined DMA request of the transmit data register empty, transmit complete, and LIN transmit data ready DMA requests
SCI_H_COMBRX	0x10	SCI_H.SCISR1[RDRF] SCI_H.LINSTAT1[RXRDY]	SCI_H combined DMA request of the receive data register full and LIN receive data ready DMA requests
DSPI_A_SR_TFFF	0x11	DSPI_A.DSPI_SR[TFFF]	DSPI_A transmit FIFO fill flag
DSPI_A_SR_RFDF	0x12	DSPI_A.DSPI_SR[RFDF]	DSPI_A receive FIFO drain flag
DSPI_B_SR_TFFF	0x13	DSPI_B.DSPI_SR[TFFF]	DSPI_B transmit FIFO fill flag
DSPI_B_SR_RFDF	0x14	DSPI_B.DSPI_SR[RFDF]	DSPI_B receive FIFO drain flag
DSPI_C_SR_TFFF	0x15	DSPI_C.DSPI_SR[TFFF]	DSPI_C transmit FIFO fill flag
DSPI_C_SR_RFDF	0x16	DSPI_C.DSPI_SR[RFDF]	DSPI_C receive FIFO drain flag
DSPI_D_SR_TFFF	0x17	DSPI_D.DSPI_SR[TFFF]	DSPI_D transmit FIFO fill flag

Table 13-4. DMA Source Configuration (continued)

DMA Request	DMA_MUX Source Input Number	DMA Source	Description
DSPI_D_SR_RFDF	0x18	DSPI_D.DSPI_SR[RFDF]	DSPI_D receive FIFO drain flag
eMIOS200_FLAG_F0	0x19	eMIOS200.eMIOS200FLAG[F0]	eMIOS200 channel 0 flag
eMIOS200_FLAG_F1	0x1A	eMIOS200.eMIOS200FLAG[F1]	eMIOS200 channel 1 flag
eMIOS200_FLAG_F2	0x1B	eMIOS200.eMIOS200FLAG[F2]	eMIOS200 channel 2 flag
eMIOS200_FLAG_F3	0x1C	eMIOS200.eMIOS200FLAG[F3]	eMIOS200 channel 3 flag
eMIOS200_FLAG_F4	0x1D	eMIOS200.eMIOS200FLAG[F4]	eMIOS200 channel 4 flag
eMIOS200_FLAG_F5	0x1E	eMIOS200.eMIOS200FLAG[F5]	eMIOS200 channel 5 flag
eMIOS200_FLAG_F6	0x1F	eMIOS200.eMIOS200FLAG[F6]	eMIOS200 channel 6 flag
eMIOS200_FLAG_F7	0x20	eMIOS200.eMIOS200FLAG[F7]	eMIOS200 channel 7 flag
eMIOS200_FLAG_F8	0x21	eMIOS200.eMIOS200FLAG[F8]	eMIOS200 channel 8 flag
eMIOS200_FLAG_F9	0x22	eMIOS200.eMIOS200FLAG[F9]	eMIOS200 channel 9 flag
eMIOS200_FLAG_F10	0x23	eMIOS200.eMIOS200FLAG[F10]	eMIOS200 channel 10 flag
eMIOS200_FLAG_F11	0x24	eMIOS200.eMIOS200FLAG[F11]	eMIOS200 channel 11 flag
eMIOS200_FLAG_F12	0x25	eMIOS200.eMIOS200FLAG[F12]	eMIOS200 channel 12 flag
eMIOS200_FLAG_F13	0x26	eMIOS200.eMIOS200FLAG[F13]	eMIOS200 channel 13 flag
eMIOS200_FLAG_F14	0x27	eMIOS200.eMIOS200FLAG[F14]	eMIOS200 channel 14 flag
eMIOS200_FLAG_F15	0x28	eMIOS200.eMIOS200FLAG[F15]	eMIOS200 channel 15 flag
I ² C_A_TX	0x29	I ² C_A.TX_REQ	I ² C_A transmit
I ² C_A_RX	0x2A	I ² C_A.RX_REQ	I ² C_A receive
Reserved	0x2B	Reserved	Reserved
Reserved	0x2C	Reserved	Reserved
SIU_EISR{EIF1}	0x2D	SIU.SIU_EISR{EIF1}	SIU external interrupt flag 1
SIU_EISR{EIF2}	0x2E	SIU.SIU_EISR{EIF1}	SIU external interrupt flag 2
SIU_EISR{EIF3}	0x2F	SIU.SIU_EISR{EIF3}	SIU external interrupt flag 3
SIU_EISR{EIF4}	0x30	SIU.SIU_EISR{EIF4}	SIU external interrupt flag 4
eQADC_FISR0_RFDF0	0x31	eQADC.eQADC_FISR0[RFDF0]	eQADC receive FIFO 0 drain flag
eQADC_FISR0_CFFF0	0x32	eQADC.eQADC_FISR0[CFFF0]	eQADC command FIFO 0 fill flag
eQADC_FISR1_RFDF1	0x33	eQADC.eQADC_FISR1[RFDF1]	eQADC receive FIFO 1 drain flag
eQADC_FISR1_CFFF1	0x34	eQADC.eQADC_FISR1[CFFF1]	eQADC command FIFO 1 fill flag
MLB_DMA_REQ	0x35	MLB.MSR[MDATRQS]	MLB Data Request
Reserved	0x36	Reserved	Reserved

Table 13-4. DMA Source Configuration (continued)

DMA Request	DMA_MUX Source Input Number	DMA Source	Description
Reserved	0x37	Reserved	Reserved
Always enabled	0x38	Always enabled	Always enabled
Always enabled	0x39	Always enabled	Always enabled
Always enabled	0x3A	Always enabled	Always enabled
Always enabled	0x3B	Always enabled	Always enabled
Always enabled	0x3C	Always enabled	Always enabled
Always enabled	0x3D	Always enabled	Always enabled
Always enabled	0x3E	Always enabled	Always enabled
Always enabled	0x3F	Always enabled	Always enabled

¹ Configuring a DMA channel to select source 0 or any of the reserved sources will disable that DMA channel.

13.4 Functional Description

The primary purpose of the DMA_MUX is to provide flexibility in the system's use of the available DMA channels. As such, configuration of the DMA_MUX is intended to be a static procedure done during execution of the system boot code. However, if the procedure outlined in [Section 13.5.2, "Enabling and Configuring Sources,"](#) is followed, the configuration of the DMA_MUX may be changed during the normal operation of the system.

Functionally, the DMA_MUX channels may be divided into two classes: channels 0–7, which implement the normal routing functionality and periodic triggering capability, and channels 8–15, which implement only the normal routing functionality.

13.4.1 DMA Channels 0–7

In addition to the normal routing functionality, channels 0–7 of the DMA_MUX provide a special periodic triggering capability that can be used to provide an automatic mechanism to transmit bytes, frames, or packets at fixed intervals without the need for processor intervention. The trigger is generated by the periodic interrupt timer (PIT); as such, the configuration of the periodic triggering interval is done via configuration registers in the PIT. Please refer to the periodic interrupt timer block guide for more information on this topic.

NOTE

Because of the dynamic nature of the system (i.e. DMA channel priorities, bus arbitration, interrupt service routine lengths, etc.), the number of clock cycles between a trigger and the actual DMA transfer cannot be guaranteed.

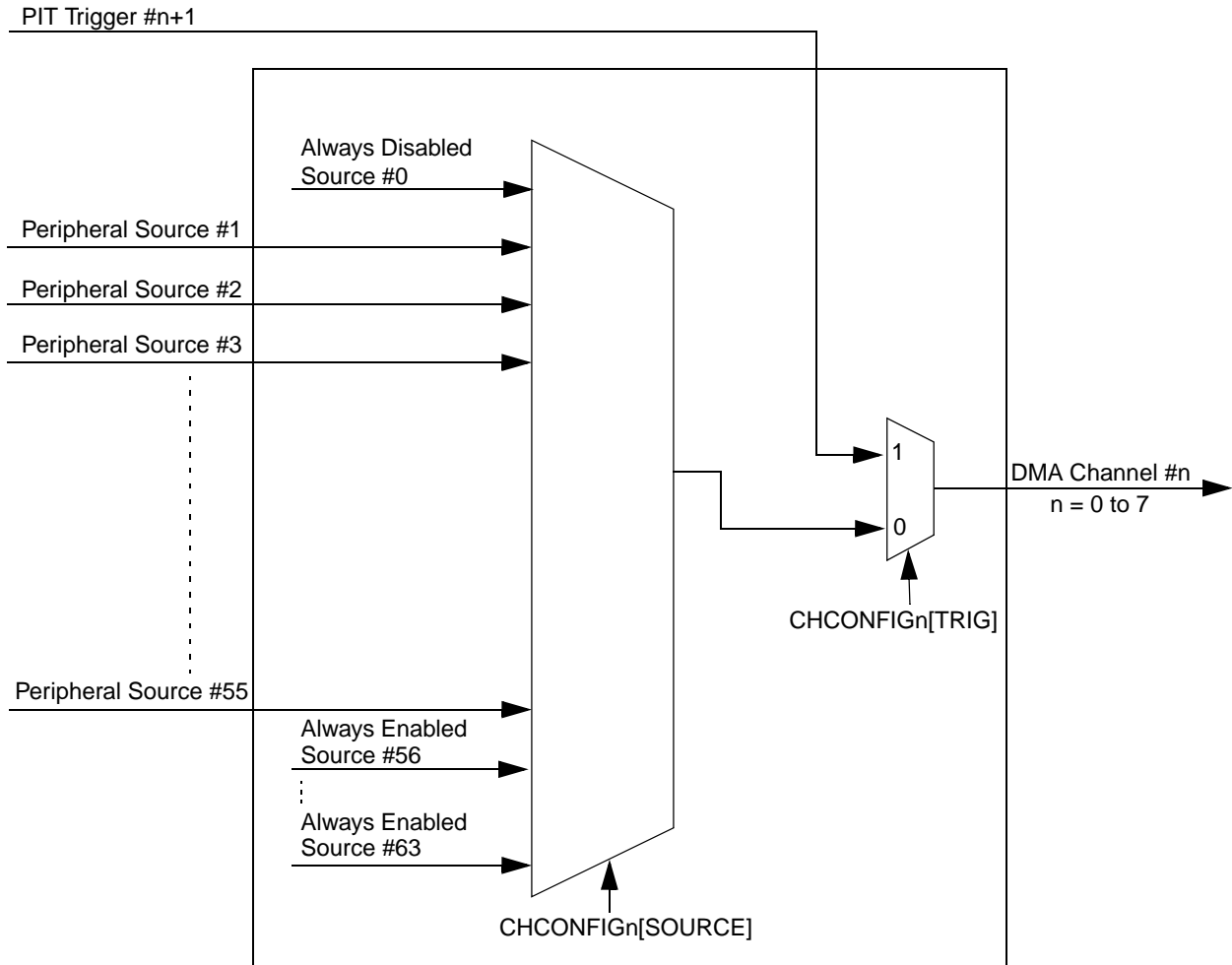


Figure 13-3. DMA_MUX Channel 0-7 Block Diagram

The DMA channel triggering capability allows the system to schedule regular DMA transfers, usually on the transmit side of certain peripherals, without the intervention of the processor. This trigger works by gating the request from the peripheral to the DMA until a trigger event has been seen. This is illustrated in Figure 13-4.

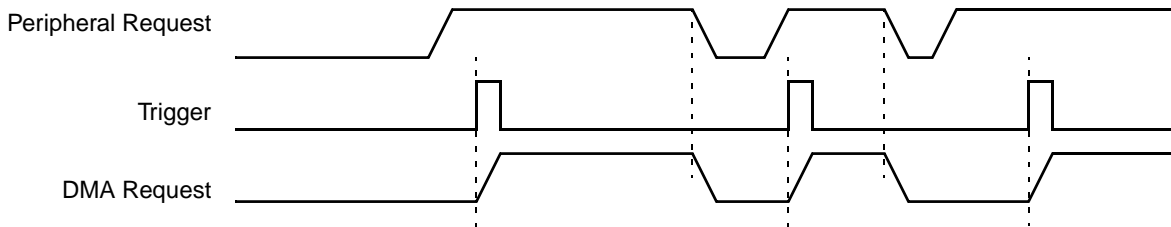


Figure 13-4. DMA_MUX Channel Triggering: Normal Operation

After the DMA request has been serviced, the peripheral negates its request, effectively resetting the gating mechanism until the peripheral re-asserts its request AND the next trigger event is seen. This means that

if a trigger is seen, but the peripheral is not requesting a transfer, that triggered will be ignored. This situation is illustrated in [Figure 13-5](#).

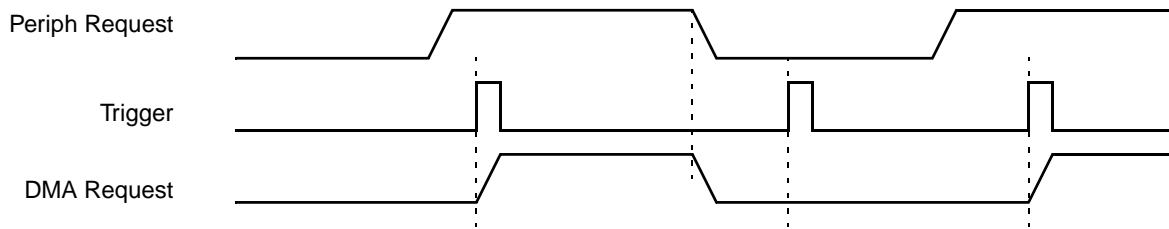


Figure 13-5. DMA_MUX Channel Triggering: Ignored Trigger

This triggering capability may be used with any peripheral that supports DMA transfers and is most useful for two types of situations:

- Periodically polling external devices on a particular bus. As an example, the transmit side of an SPI is assigned to a DMA channel with a trigger, as described above. After setup, the SPI requests DMA transfers (presumably from memory) as long as its transmit buffer is empty. By using a trigger on this channel, the SPI transfers can be automatically performed every 5 μ s (as an example). On the receive side of the SPI, the SPI and DMA can be configured to transfer receive data into memory, effectively implementing a method to periodically read data from external devices and transfer the results into memory without processor intervention.
- Using the GPIO ports to drive or sample waveforms. By configuring the DMA to transfer data to one or more GPIO ports, it is possible to create complex waveforms using tabular data stored in on-chip memory. Conversely, using the DMA to periodically transfer data from one or more GPIO ports, it is possible to sample complex waveforms and store the results in tabular form in on-chip memory.

A more detailed description of the capability of each trigger (i.e. resolution, range of values, etc.) may be found in [Chapter 28, “Periodic Interrupt Timer and Real Time Interrupt \(PIT_RTI\).”](#)

13.4.2 DMA Channels 8–15

Channels 8–15 of the DMA_MUX provide the normal routing functionality as described in [Section 13.1.3, “Modes of Operation.”](#)

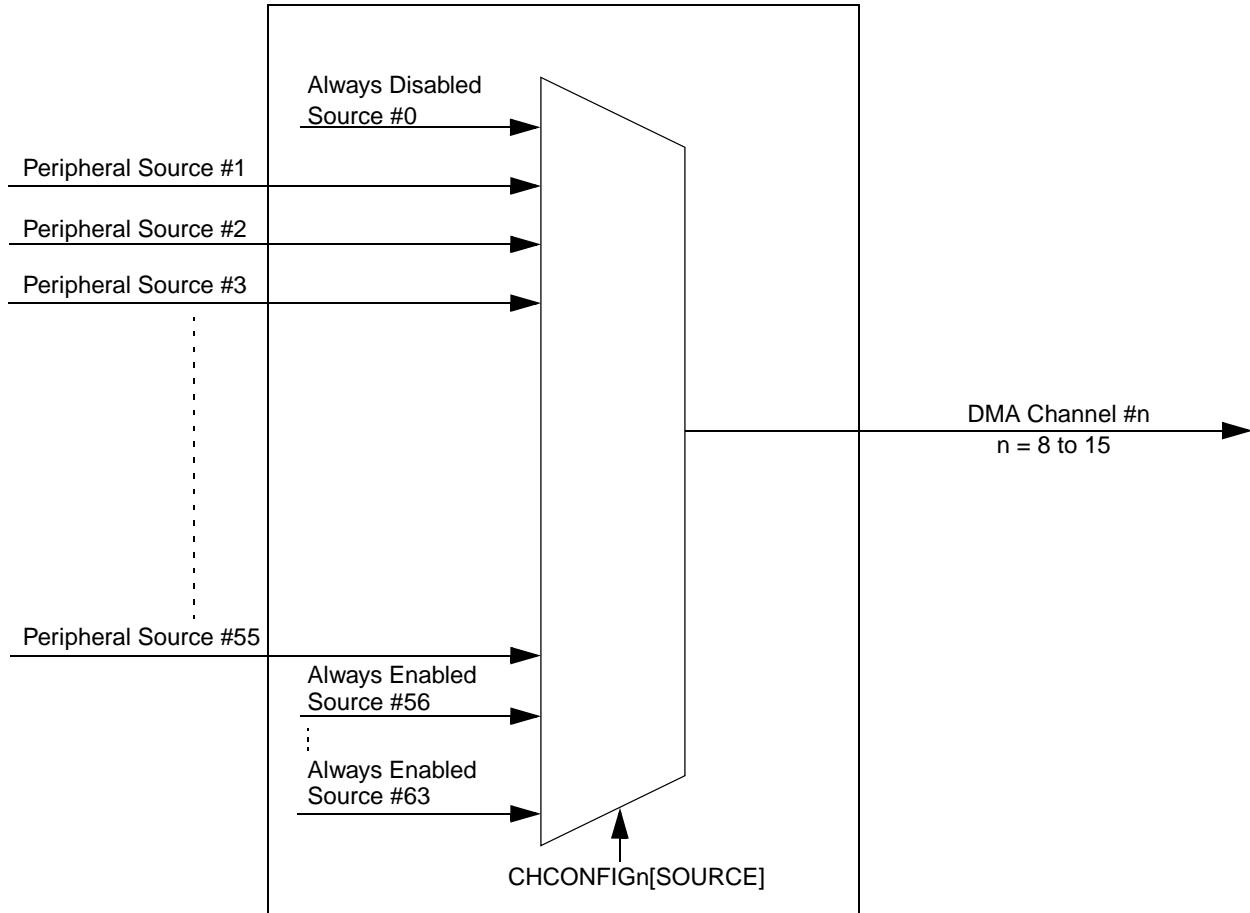


Figure 13-6. DMA_MUX Channel 8–15 Block Diagram

13.4.3 Always Enabled DMA Sources

In addition to the 55 peripherals that can be used as DMA sources, there are eight additional DMA sources that are always enabled. Unlike the peripheral DMA sources, where the peripheral controls the flow of data during DMA transfers, the always enabled sources provide no such throttling of the data transfers. These sources are most useful in the following cases:

- Doing DMA transfers to/from GPIO — Moving data from/to one or more GPIO pins, either un-throttled (i.e. as fast as possible), or periodically (using the DMA triggering capability).
- Doing DMA transfers from memory to memory — Moving data from memory to memory, typically as fast as possible, sometimes with software activation.
- Doing DMA transfers from memory to the external bus (or vice-versa) — Similar to memory to memory transfers, this is typically done as quickly as possible.
- Any DMA transfer that requires software activation — Any DMA transfer that should be explicitly started by software.

In cases where software should initiate the start of a DMA transfer, an always enabled DMA source can be used to provide maximum flexibility. When activating a DMA channel via software, subsequent

executions of the minor loop require a new start event be sent. This can either be a new software activation or a transfer request from the DMA channel mux. The options for doing this are:

- Transfer all data in a single minor loop. By configuring the DMA to transfer all of the data in a single minor loop (i.e. major loop counter = 1), no re-activation of the channel is necessary. The disadvantage to this option is the reduced granularity in determining the load that the DMA transfer will incur on the system. For this option, the DMA channel should be disabled in the DMA channel mux.
- Use explicit software re-activation. In this option, the DMA is configured to transfer the data using both minor and major loops, but the processor is required to re-activate the channel (by writing to the DMA registers) after every minor loop. For this option, the DMA channel should be disabled in the DMA channel mux.
- Use an always enabled DMA source. In this option, the DMA is configured to transfer the data using both minor and major loops, and the DMA channel mux does the channel re-activation. For this option, the DMA channel should be enabled and pointing to an always enabled source. Note that the re-activation of the channel can be continuous (DMA triggering is disabled) or can use the DMA triggering capability. In this manner, it is possible to execute periodic transfers of packets of data from one source to another without processor intervention.

13.5 Initialization/Application Information

13.5.1 Reset

The reset state of each individual bit is shown within the register description section ([Section 13.3.2, “Register Descriptions”](#)). After reset, all channels are disabled and must be explicitly enabled before use.

13.5.2 Enabling and Configuring Sources

13.5.2.1 Enabling a Source with Periodic Triggering

1. Determine with which DMA channel the source will be associated. Only DMA channels 0–7 have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point.
4. In the PIT, configure the corresponding timer.
5. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

Example 13-1. Configure DSPI_B Transmit for use with DMA Channel 2, with periodic triggering capability

1. Write 0x00 to CHCONFIG2 (base address + 0x02).
2. Configure channel 2 in the DMA, including enabling the channel.
3. Configure timer 3 in the periodic interrupt timer (PIT) for the desired trigger interval.
4. Write 0xC5 to CHCONFIG2 (base address + 0x02).

The following code example illustrates steps #1 and #4 above:

In File **registers.h**:

```
#define DMAMUX_BASE_ADDR      0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);
```

In File **main.c**:

```
#include "registers.h"
:
:
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0xC5;
```

13.5.2.2 Enabling a Source Without Periodic Triggering

1. Determine with which DMA channel the source will be associated. Only DMA channels 0–7 have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point.
4. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL is set and the TRIG bit is cleared.

Example 13-2. Configure DSPI_B Transmit for use with DMA Channel 2, with no periodic triggering capability.

1. Write 0x00 to CHCONFIG2 (base address + 0x02).
2. Configure channel 2 in the DMA, including enabling the channel.
3. Write 0x85 to CHCONFIG2 (base address + 0x02).

The following code example illustrates steps #1 and #3 above:

In File **registers.h**:

```
#define DMAMUX_BASE_ADDR      0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
```

```
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);
```

```
In File main.c:
#include "registers.h"
:
:
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0x85;
```

13.5.2.3 Disabling a Source

A particular DMA source may be disabled by not writing the corresponding source value into any of the CHCONFIG registers. Some module specific configuration may also be necessary. Refer to the appropriate section for more details.

13.5.2.4 Switching the Source of a DMA Channel

1. Disable the DMA channel in the DMA and re-configure the channel for the new source.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

Example 13-3. Switch DMA Channel 8 from DSPI_A transmit to ESCI_A transmit

1. In the DMA configuration registers, disable DMA channel 8 and re-configure it to handle the DSPI_A transmits.
2. Write 0x00 to CHCONFIG8 (base address + 0x08).
3. Write 0x87 to CHCONFIG8 (base address + 0x08). In this case, setting the TRIG bit has no effect because channels 8–15 do not support the periodic triggering functionality.

The following code example illustrates steps #2 and #4 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR      0xFC084000 /* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
```

DMA Channel Mux (DMA_MUX)

```
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);
```

In File **main.c**:

```
#include "registers.h"
    :
    :
*CHCONFIG8 = 0x00;
*CHCONFIG8 = 0x87;
```

13.6 Interrupts

The DMA channel mux does not generate interrupts.

Chapter 14

Peripheral Bridge (AIPS-lite)

14.1 Introduction

The AIPS-lite acts as an interface between the system bus and lower bandwidth peripherals.

14.1.1 Terminology

Table 14-1. Terms and Acronyms

Terms	Description
AHB 2.v6	AMBA AHB-lite version 2.0 with v6 extensions
AMBA AHB-Lite Interface	A standard AHB-lite bus interface
AIPS	AHB 2.v6 to IPS interface unit
IPS	IP slave interface—A Freescale Intellectual Property Interface standard used for interfacing to peripheral devices and control registers (slaves).
Pipeline	Act of initiating a bus cycle while another bus cycle is in progress. Thus, the bus can have multiple bus cycles pending at one time.
Slave	A bus slave is a device that responds to a bus transaction, but never initiates a cycle on the bus.
Transaction	A bus transaction consists of an address transfer (address phase) and one or more data transfer(s) (data phase).

14.1.2 Block Diagram

A simplified block diagram of the AIPS-lite illustrates the functionality and interdependence of major blocks (see [Figure 14-1](#)).

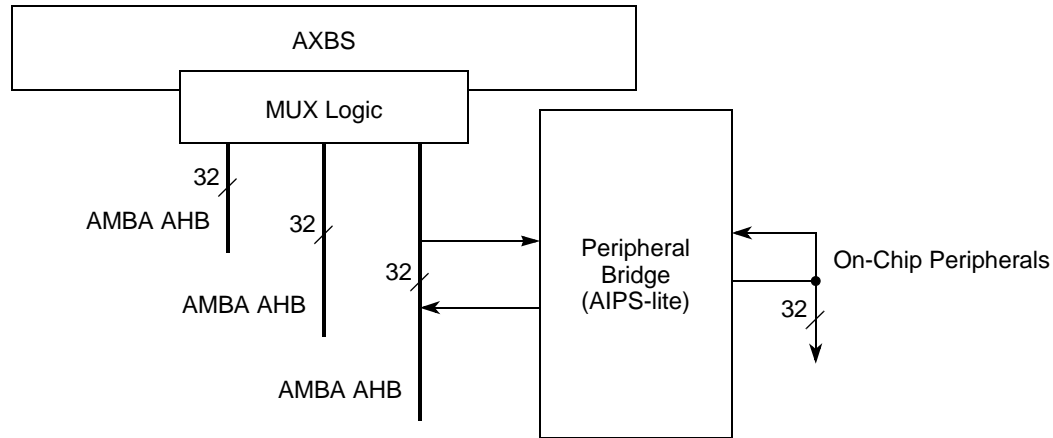


Figure 14-1. AIPS-lite Block Diagram

14.1.3 Features

The AIPS-lite has these major features:

- AIPS-lite supports the IPS slave interface signals. This interface is meant for slave peripherals only.
- AIPS-lite supports 32-bit IPS peripherals. (Byte, halfword, and word reads and write are supported to each.)
- Read and write accesses of 32 bits or less require two clocks, provided they do not cross a 32-bit boundary.
 - Read and write accesses that cross a 32-bit boundary are not supported.
- The peripherals connected to the AIPS-lite may be configured in groups to run at less than the system clock frequency. See [Section 3.4.5, “Peripheral Clock Dividers,”](#) in [Chapter 3, “System Clock Description,”](#) for a description of these groups.

14.1.4 Modes of Operation

The AIPS-lite has only one operating mode.

14.2 External Signal Description

The AIPS-lite has no external signals.

14.3 Memory Map and Registers

The AIPS-lite does not contain any user-programmable registers.

14.4 Functional Description

The AIPS-lite serves as an interface between an AHB 2.v6 system bus and the peripheral interface bus. It functions as a protocol translator.

Accesses that fall within the address space of the AIPS-lite are decoded to provide individual module selects for peripheral devices on the peripheral bus interface.

See the peripherals section of [Table 1-7](#) for a description of which peripherals are allocated to which 16 KB memory space in the AIPS-lite address map.

14.4.1 Read Cycles

Two-clock read accesses are possible with the AIPS-Lite when the reference size is 32 bits or smaller. This module does not support any type of misaligned read access crossing a 32-bit boundary.

14.4.2 Write Cycles

Two-clock write accesses are possible with the AIPS-Lite when the reference size is 32 bits or smaller. This module does not support any type of misaligned write access crossing a 32-bit boundary.

Chapter 15

Crossbar Switch (XBAR)

15.1 Introduction

The multi-port crossbar switch (XBAR) implements a hardware interconnection matrix supporting two simultaneous connections between six master ports and two slave ports. One slave port is used to access the system RAM. The other slave port is shared by the secondary flash port (port 1), the EBI, and the AIPS.

The XBAR supports a 32-bit address bus width and a 32-bit data bus width at all master and slave ports.

The XBAR allows for concurrent transactions to occur from any master port to any slave port. It is possible for both slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic will select the appropriate master and grant it ownership of the slave port. By default, requesting masters will be treated with round-robin priority and will be granted access to a slave port if it is the current higher priority master or when the current higher priority master has completed its operation. If operating with fixed-priority arbitration, all other masters requesting that slave port will be stalled until the current higher priority master completes its transactions. The XBAR can be configured to use fixed priority arbitration by clearing the MCM_MUDCR[PRI] bit.

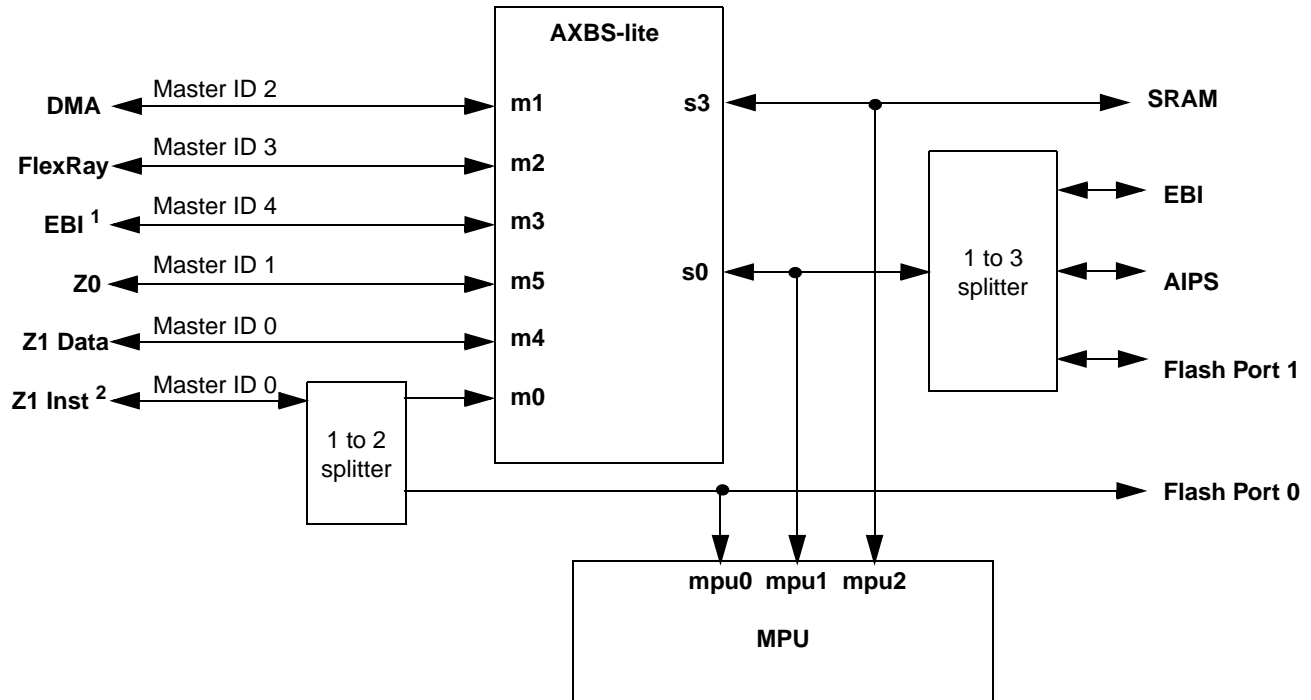
Table 15-1 lists the master IDs for each of the possible bus masters on MPC5510.

Table 15-1. Master IDs

Master	Master ID	XBAR port
e200z1 data	0	m4
e200z1 instr		m0
e200z0	1	m5
eDMA	2	m1
FlexRay	3	m2
EBI	4	m3
Nexus 2+ (e200z1)	8	—
Nexus 2+ (e200z0)	9	—

15.1.1 Block Diagram

A simplified block diagram of the XBAR illustrates the functionality and interdependence of major blocks (see Figure 15-1).



¹ For factory test only.

² For Z1, all instruction accesses to flash go through port P0. The path from the Z1 Instruction bus through the 1 to 2 splitter and AXBS port m0 is only used for non-Flash (i.e. RAM and BAM) instruction fetches.

Figure 15-1. XBAR Block Diagram

15.1.2 Features

The XBAR has these major features:

- Masters (listed in order of highest to lowest priority when the XBAR is configured for fixed priority arbitration, the logical master number and physical port connection are provided)
 - eDMA (master ID = 2, XBAR m1)
 - FlexRay (master ID = 3, XBAR m2)
 - EBI master (master ID = 4, XBAR m3)
 - The EBI is a master for test purposes only
 - Z0 core (master ID = 1, XBAR m5)
 - Z1 core—Data (master ID = 0, XBAR m4)
 - Z1 core—Instruction (master ID = 0, XBAR m0)
 - Nexus 2+ pretending to be Z0 core (master ID = 9)
 - Nexus 2+ pretending to be Z1 core (master ID = 8)
- Slaves
 - SRAM (XBAR s3)

- Shared between flash port 1 / EBI slave / AIPS peripheral bridge (XBAR s0)
- 32-bit address, 32-bit data paths
- Two concurrent transfers between independent master and slave ports
 - Only one device on the shared slave port can be accessed at a time. If the shared slave port is occupied by a master, then another master's access to a device on the shared slave port will be blocked, even if the second master is requesting access to one of the other devices on the shared slave port, until the first master's access has completed. For example, if the Z0 is accessing the AIPS, then the Z1 data bus will not be able to access flash port 1 until the Z0 access has completed.

15.1.3 Modes of Operation

The XBAR has two arbitration modes. See [Section 15.4.3.2, “Round-Robin Priority Operation,”](#) and [Section 15.4.3.3, “Fixed Priority Operation.”](#)

15.2 Signal Description

The XBAR has no external signals.

15.3 Memory Map and Registers

The XBAR has no memory mapped registers.

15.4 Functional Description

The main goal of the XBAR is to increase overall system performance by allowing up to two simultaneous transfers between master ports and slave ports.

When a master makes an access to the XBAR from an idle master state, the access will be immediately taken by the XBAR. If the targeted slave port of the access is available (i.e. the requesting master is currently granted ownership of the slave port), the access will be immediately presented on the slave port. If the targeted slave port of the access is busy or parked on a different master port, the requesting master will see wait states inserted until the targeted slave port can service the master's request. The latency in servicing the request will depend on each master's priority level and the responding slave's access time.

A master will be given control of the targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has an outstanding request to one slave port that has a long response time, has a pending access to a different slave port, and a lower priority master is also making a request to the same slave port as the pending access of the higher priority master.

After the master has control of the slave port it is targeting, the master will remain in control of that slave port until it gives up the slave port by running an IDLE cycle or by leaving that slave port for its next access. The master can also lose ownership of the slave port when a transfer boundary is reached if the arbitration logic has selected another master. In this context, the transfer boundary is defined as the completion of any “single” transfer, the completion of each transfer within an undefined-length burst, the

completion of all the transfers of a fixed-length burst or the negation of lock after a series of indivisible transfers.

When a slave bus is being idled by the XBAR, it is parked on a specific master port. On the MPC5510, the shared flash/AIPS/EBI slave port is parked on the e200z0, and the SRAM slave port is parked on the last master to access it.

15.4.1 Master Ports

A master access will be taken if the slave port to which the access decodes is either currently servicing the master or is parked on the master. In this case the XBAR will be completely transparent and the master's access will be immediately seen on the slave bus and no arbitration delays will be incurred. A master access will be stalled if the access decodes to a slave port that is busy serving another master or parked on another master.

If the slave port is currently parked on another master and no other master is requesting access to the slave port, then only one clock of arbitration will be incurred. If the slave port is currently serving another master and the arbitration logic selects a new master, then the new master gains controls over the slave port as soon as the data phase of the current access is completed. When operating with fixed-priority arbitration, if the slave port is currently servicing another master of a higher priority, then the lower-priority master gains control of the slave port once the other master releases control of the slave port as long as no other higher priority master is also waiting for the slave port.

A master access is terminated with an error if the access decodes to a location not occupied by a slave port. This is the only time the XBAR will directly respond with an error response. All other error responses received by the master are the result of error responses on the slave ports being passed through the XBAR.

15.4.2 Slave Ports

The goal of the XBAR with respect to the slave ports is to keep them 100% saturated when masters are actively making requests. In order to do this the XBAR must not insert any pipeline stall cycles onto the slave bus unless absolutely necessary.

There is only one instance when the XBAR will force a stall onto the slave bus when a master is actively making a request. This occurs when a handoff of bus ownership occurs and there are no wait states from the slave port. A requesting master which does not own the slave port will be granted access after a one clock delay.

The only other time the XBAR will have control of the slave port is when no masters are making access requests to the slave port and the XBAR is forced to park the slave port on a specific master. In this case, the XBAR will force IDLE for the transfer type.

15.4.3 Arbitration

The XBAR supports two arbitration schemes: a fixed-priority algorithm, and a round-robin fairness algorithm. The selected arbitration scheme is applied to all slave ports. On MPC5510, the XBAR defaults to round robin priority arbitration. The priority arbitration scheme is selectable via the MCM

miscellaneous user defined control register as described in [Section 16.2.2.4, “Miscellaneous User-Defined Control Register \(MUDCR\).”](#)

15.4.3.1 Arbitration During Undefined Length Bursts

The XBAR is explicitly configured to arbitrate after each transfer in an underfined-length burst.

15.4.3.2 Round-Robin Priority Operation

When operating in round-robin mode, each master is assigned a relative priority based on the master number. This relative priority is compared to the ID of the last master to perform a transfer on the slave bus. The highest priority requesting master will become owner of the slave bus at the next transfer boundary (accounting for locked and burst transfers). Priority is based on how far ahead the ID of the requesting master is to the ID of the last master (ID is defined by master port number).

Once granted access to a slave port, a master may perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line will be granted access to the slave port at the termination of the current bus access, or on the next clock cycle if the current master has no pending access request.

As an example of arbitration in round-robin mode, assume the XBAR is implemented with master ports 0, 1, 4, and 5. If the last master of the slave port was master 1, and masters 0, 4, and 5 make simultaneous requests, they will be serviced in the order 4, 5, and then 0.

Parking may still be used in a round-robin mode, but will not affect the round-robin pointer unless the parked master performs a transfer. Handoff will occur to the next master in line after one cycle of arbitration.

Each master port has a high priority request input signal. If a master port's high priority request input is enabled for a slave port programmed for round-robin mode, that master can force the slave port into fixed priority mode by asserting its high priority request input while making a request to that particular slave port. While that (or any enabled) master's high priority request input is asserted while making an access attempt to that particular slave port, the slave port will remain in fixed priority mode. After that (or any enabled) master's high priority request input is negated, or the master no longer attempts to make accesses to that particular slave port, the slave port will revert back to round-robin priority mode and the pointer will be set on the last master to access the slave port.

NOTE

When either the e200z1 or e200z0 request an external or critical interrupt, or for certain configurations of an active eDMA transfer control descriptor, a high priority access will be requested that puts the requesting master at the front of the queue. The ability to enable the high-priority request from the processors is programmable. This feature is enabled via the assertion of the appropriate HID1 control bits in the e200 core.

15.4.3.3 Fixed Priority Operation

When operating in fixed-priority mode, each master is assigned a unique priority level, as described in [Section 15.1.2, “Features.”](#) If two masters both request access to a slave port, the master with the highest priority in the selected priority parameter will gain control over the slave port.

Any time a master makes a request to a slave port, the slave port checks to see if the new requesting master’s priority level is higher than that of the master that currently has control over the slave port (unless the slave port is in a parked state). The slave port does an arbitration check at every clock cycle to ensure that the proper master (if any) has control of the slave port.

If the new requesting master’s priority level is higher than that of the master that currently has control of the slave port the new requesting master will be granted control over the slave port at the next clock cycle. The exception to this rule is if the master that currently has control over the slave port is running a burst transfer or a locked transfer. In this case the new requesting master will have to wait until the end of the burst transfer or locked transfer before it will be granted control of the slave port.

If the new requesting master’s priority level is lower than that of the master that currently has control of the slave port, the new requesting master will be forced to wait until the master that currently has control of the slave port either runs an IDLE cycle or runs a non IDLE cycle to a location other than the current slave port.

15.4.4 Slave Port State Machine

15.4.4.1 Slave Port State Machine Arbitration

The real work in the state machine is determining which master port will be in control of the slave port in the next clock cycle, the arbitration. Each master is programmed with a fixed 3 bit priority level equivalent to the master number. A fourth priority bit is derived from the master port’s high priority request inputs on the master ports, effectively making each master’s priority level a 4 bit field with master port’s high priority request being the MSB. The XBAR uses these bits in determining priority levels when programmed for fixed priority mode of operation or when one of the enabled master port’s high priority request inputs is asserted.

Arbitration always occurs on a clock cycle, but only occurs on edges when a change in mastership will not violate AHB-lite protocols. Valid arbitration points include any clock cycle in which current bus access is asserted (provide the master is not performing a burst or locked cycle) and any wait state in which the master owning the bus indicates a transfer type of IDLE (provided the master is not performing a locked cycle).

Since arbitration can occur on every clock cycle the slave port masks off all master requests if the current master is performing a locked transfer or a protected burst transfer, guaranteeing that no matter how low its priority level it will be allowed to finish its locked or protected portion of a burst sequence.

15.4.4.2 Slave Port State Machine Parking

If no master is currently making a request to the slave port then the slave port is parked on a given master port. When a slave port is parked on a master and that master accesses the slave port, the master does not

incur an arbitration penalty or delay. However, if any other master wishes to access the slave port, a one clock arbitration penalty is imposed.

Two types of parking are supported, park on a fixed master and park on the last master. When a slave port is configured for parking on a fixed master, the ownership returns to the fixed master whenever the slave port becomes idle. Figure 15-2 illustrates parking on a fixed master. When a slave port is configured for parking on the last master, ownership remains with the last master to access the slave port until another master requests an access. Figure 15-3 illustrates parking on the last master.

On MPC5510, the s0 port is “parked on the e200z0” and the s3 port is “parked on last”. This configuration is hardwired and cannot be changed by software.

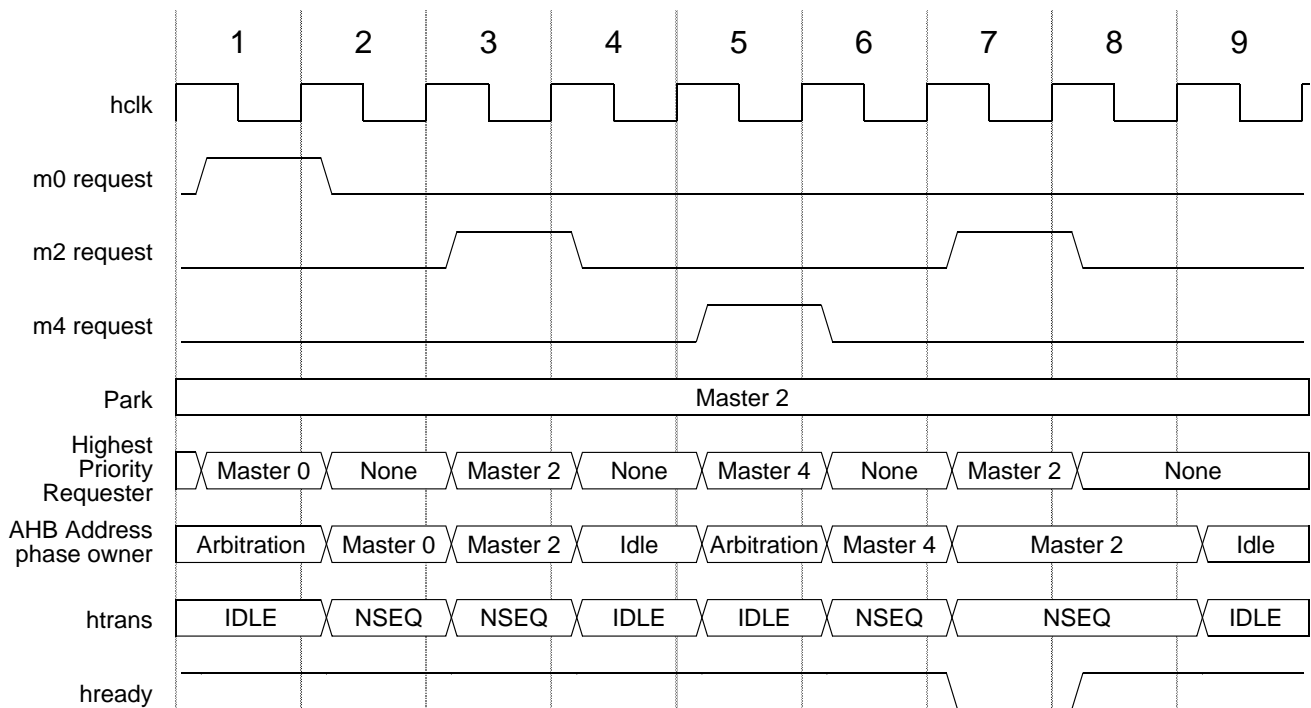


Figure 15-2. Parking on a Specific Master¹

1. Hardwired and not user changeable configuration for slave port s0.

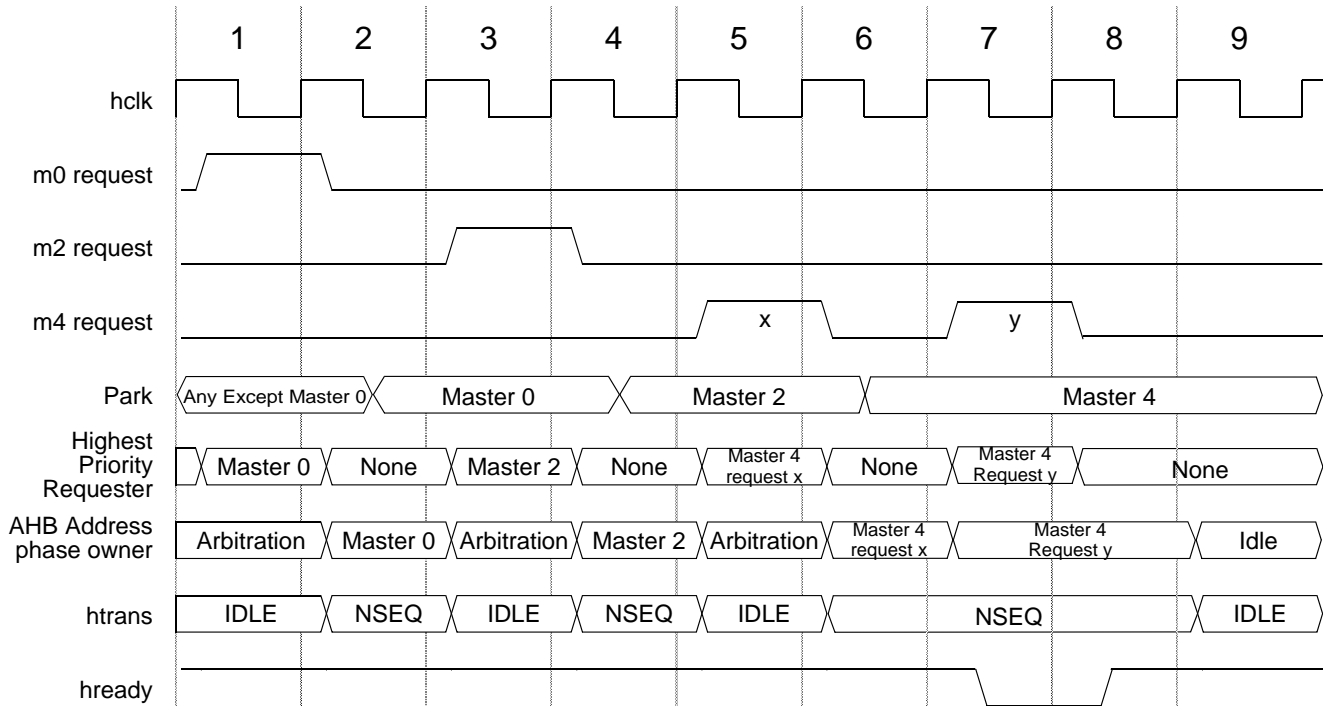


Figure 15-3. Parking on Last Master¹

15.5 DMA Requests

There are no DMA requests associated with the XBAR.

15.6 Interrupt Requests

There are no interrupt requests associated with the XBAR.

1. Hardwired and not user changeable configuration for slave port s3.

Chapter 16

Miscellaneous Control Module (MCM)

16.1 Introduction

The miscellaneous control module (MCM) provides control functions regarding a software watchdog timer (SWT) and information on memory errors reported by error-correcting codes (ECC).

The MCM provides a set of registers that configure and report ECC errors for the device including accesses to RAM and flash memory. The application may configure the device for the types of memory errors to be reported, and then query a set of read-only status and information registers to identify any errors that have been signaled.

There are two types of ECC errors: correctable and non-correctable. A correctable ECC error is generated when only one bit is wrong in a 64-bit doubleword. In this case, it is corrected automatically by hardware and no flags or other indication is set that the error occurred. A non-correctable ECC error is generated when two or more bits in a 64-bit doubleword are incorrect. Non-correctable ECC errors cause an interrupt, and if enabled, additional error details are available in the MCM.

Error correction is implemented on 64 bits of data at a time, using eight bits for ECC for every 64-bit doubleword. ECC is checked on reads and calculated on writes per the following:

1. 64 bits containing the desired byte / halfword / word or doubleword in memory is read and ECC checked.
2. If the access is a write, then
 - The new byte / halfword / word / doubleword is merged into the 64 bits.
 - New ECC bits are calculated.
 - The 64 bits and the new ECC bits are written back.

NOTE

Before using the SRAM, it must be initialized, even if the application does not use ECC reporting, see [Chapter 21, “Internal Static RAM \(SRAM\)”](#) for more information.

16.1.1 Features

The MCM has these major features:

- Software watchdog timer (SWT) with programmable interrupt response
 - The default state after reset of the SWT is enabled
 - The SWT count can be optionally held when system debug is enabled

- After reset, the SWT will be clocked by the 16 MHz IRC, with a default timeout of 2^{17} clocks. The clock source can be changed by the end user to the bus clock (see [Chapter 6, “System Integration Unit \(SIU\)”](#)). Note: the BAM will write to the SWT timeout, before going into serial boot mode, and change the timeout for the SWT to 2^{27} clocks.
- Registers for capturing information on memory errors if error-correcting codes (ECC) are implemented.

16.2 Memory Map and Registers

This section provides a detailed description of all MCM registers.

16.2.1 Module Memory Map

The MCM memory map is shown in [Table 16-1](#) (a graphical layout of the registers is shown in [Table 16-2](#) to better see Reserved areas in the memory map). The address of each register is given as an offset to the MCM base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 16-1. MCM Memory Map

Offset from MCM_BASE_ADDR (0xFFFF4_0000)	Register	Access	Reset Value ¹	Section/Page
0x0000–0x0015	Reserved			
0x0016	SWTCR—Software Watchdog Timer (SWT) Control		0x00D1	16.2.2.1/16-4
0x001B	SWTSR—SWT Service		U	16.2.2.2/16-6
0x001F	SWTIR—SWT Interrupt		0x00	16.2.2.3/16-7
0x0024	MUDCR—Miscellaneous User Defined Control Register		0x8000_0000	16.2.2.4/16-7
0x0043	ECR—ECC Configuration		0x00	16.2.2.5.1/16-8
0x0047	ESR—ECC Status		0x00	16.2.2.5.2/16-9
0x004A	EEGR—ECC Error Generation		0x0000	16.2.2.5.3/16-10
0x0050	FEAR—Flash ECC Address		U	16.2.2.5.4/16-12
0x0056	FEMR—Flash ECC Master		0x0U	16.2.2.5.5/16-12
0x0057	FEAT—Flash ECC Attributes		U	16.2.2.5.6/16-13
0x0058	Reserved			
0x005C	FEDR—Flash ECC Data		U	16.2.2.5.7/16-14
0x0060	REAR—RAM ECC Address		U	16.2.2.5.8/16-14
0x0066	REMR—RAM ECC Master		0x0U	16.2.2.5.9/16-15
0x0067	REAT—RAM ECC Attributes		U	16.2.2.5.10/16-15

Table 16-1. MCM Memory Map (continued)

Offset from MCM_BASE_ADDR (0xFFFF4_0000)	Register	Access	Reset Value ¹	Section/Page
0x0068	Reserved			
0x006C	REDR—RAM ECC Data		U	16.2.2.5.11/16-16

¹ Please refer to the register definition. U=undefined at reset.

Table 16-2. MCM Graphical Memory Map

MCM Offset	Register		
0x0000–0x00013	Reserved		
0x0014	Reserved	Software Watchdog Timer Control (SWTCR)	
0x0018	Reserved		SWT Service (SWTSR)
0x001C	Reserved		SWT Interrupt (SWTIR)
0x0020–0x0023	Reserved		
0x0024	Miscellaneous User Defined Control Register (MUDCR)		
0x0028–0x003C	Reserved		
0x0040	Reserved		ECC Configuration (ECR)
0x0044	Reserved		ECC Status (ESR)
0x000048	Reserved	ECC Error Generation (EEGR)	
0x0050	Flash ECC Address (FEAR)		
0x0054	Reserved	Flash ECC Master (FEMR)	Flash ECC Attributes (FEAT)
0x0058	Flash ECC Data High (FEDRH)		
0x005C	Flash ECC Data Low (FEDRL)		
0x0060	RAM ECC Address (REAR)		
0x0064	Reserved	RAM ECC Master (REMR)	RAM ECC Attributes (REAT)
0x0068	Reserved		
0x006C	RAM ECC Data (REDR)		

16.2.2 Register Descriptions

This section lists the MCM registers in address order and describes the registers and their bit fields. Attempted accesses to reserved addresses result in an error termination; however, attempted writes to read-only registers are ignored and do not terminate with an error.

NOTE

Unless noted otherwise, writes to the programming model must match the size of the register, e.g., an n -bit register only supports n -bit writes, etc. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.

16.2.2.1 Software Watchdog Timer Control Register (SWTCR)

The software watchdog timer (SWT) prevents system lockup if the software becomes trapped in a loop with no controlled exit or if a bus transaction becomes hung. The software watchdog timer can be enabled or disabled through the SWTCR[SWE]. The SWT operates in a separate, asynchronous clock domain and contains clock domain synchronizers as the communication mechanism between the system clock domain and the software watchdog timer domain. If enabled, the watchdog timer requires the periodic execution of a software watchdog servicing sequence. If this periodic servicing action does not occur, the timer expires, resulting in a watchdog timer interrupt or a hardware reset, as programmed in the SWTCR[SWRI].

There are three user-defined responses to a time-out:

- The SWTCR[SWRI] can specify the assertion of a watchdog timer interrupt.
- The SWTCR[SWRI] can specify the immediate assertion of a system reset.
- The SWTCR[SWRI] can specify a sequence of responses. Upon the first time-out, the watchdog timer interrupt is asserted. If the watchdog timer interrupt flag is not cleared before a second time-out occurs, the watchdog timer asserts the system reset signal in response to the second time-out. This configuration supports a more graceful response to watchdog time-outs: first attempting an interrupt to notify the system, if that fails, generating a system reset.

In addition to these three modes of operation, the watchdog timer also supports a windowed mode of operation. In this mode, the time-out period is divided into four equal segments and the servicing of the watchdog timer must occur during the last segment, i.e., during the last 25% of the time-out period. If the watchdog timer is serviced anytime in the first 75% of the time-out period, an immediate system reset occurs.

Throughout this section, there are many references to the generation of a system reset. The MCM's behavior during this process is detailed below. When the watchdog timer expires and SWTCR[SWRI] is programmed for a reset (either as the initial or secondary response), the MCM generates a watchdog timer reset output signal, which is driven to the SIU and will cause a system reset.

The watchdog timer logic also sends an interrupt request to the device's interrupt controller.

To prevent the watchdog timer from interrupting or resetting, the SWTSR must be serviced by performing the following sequence:

1. Write 0x55 to the SWTSR.
2. Write 0xAA to the SWTSR.

Both writes must occur in this order before the time-out, but any number of instructions can be executed between the two writes. This definition allows interrupts and exceptions to occur, if necessary, between the two writes. The timer value is constantly compared with the time-out period specified by SWTCR[SWT].

NOTE

Any write to the SWTCR resets the watchdog timer.

There is also a read-only control flag included in the SWTCR to prevent accidental updates to this control register from changing the desired system configuration.

If the second write occurs at the exact same cycle as the time-out condition is reached, the clear takes precedence and the time-out response suppressed.

The SWTCR controls the software watchdog timer, time-out periods, and time-out response. The register can be read or written at any time. At system reset, the software watchdog timer is enabled. See [Figure 16-1](#) and [Table 16-3](#) for the software watchdog timer control register definition.

Offset: MCM_BASE_ADDR + 0x0016

Access: User read/write

	0	1	2	3	4	5	6 ¹	7	8	9	10	11	12	13	14	15
R	RO	0	0	0	0	0	0	SW RWH	SWE	SWRI		SWT				
W																
Reset ²	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1

Figure 16-1. Software Watchdog Timer Control (SWTCR) Register

¹ Bit 6 is reserved and must never be set.

² The SWTCR default reset values may be modified during BAM execution. Please reference the BAM section for more details.

Table 16-3. SWTCR Field Descriptions

Field	Description
RO	Read-Only. 0 SWTCR can be read or written. 1 SWTCR can be read only. A system reset is required before this register can again be written. The setting of this bit is intended to prevent accidental writes of the SWTCR from changing the defined system watchdog configuration.
bits 1–6	Reserved. Note: Reserved bit 6 must never be set.
SWRWH	Software Watchdog Run While Halted. 0 SWT stops counting if the processor core is halted. 1 SWT continues to count even while the processor core is halted.

Table 16-3. SWTCR Field Descriptions (continued)

Field	Description
SWE	Software Watchdog Enable. 0 SWT is disabled. 1 SWT is enabled.
SWRI	Software Watchdog Reset/Interrupt. 00 If a time-out occurs, the SWTIC interrupt flag is set and the SWT generates an interrupt request to the system. The programming of the interrupt level for the SWT is system-specific. Typically, the highest priority interrupt level is used to signal the SWT. 01 The first time-out sets the SWTIC watchdog interrupt flag and the SWT generates an interrupt request to the system. If the SWTIC watchdog timer interrupt flag is not cleared before a second time-out occurs, the watchdog timer asserts the system reset signal in response to the second time-out. 10 If a time-out occurs, the SWT generates a system reset. 11 The SWT functions in a window mode of operation. For this mode, the servicing of the MSWSR must occur during the last 25% of the time-out period. Any writes to the MSWSR during the first 75% of the time-out period generate an immediate system reset. Likewise, if the MSWSR is not serviced during the last 25% of the time-out period, then a system reset is generated.
SWT	Software Watchdog Time-Out Period. Selects the time-out period for the SWT. At reset, this field is 0b10001 (2^{17} clocks). In general, the value in this field defines the bit position within the 32-bit counter that specifies the time-out period. Thus, if $SWT = n$, then the time-out period is 2^n system clock cycles. Since it is not practical to operate the software watchdog timer with very short time-out periods, data values of [0–7] are forced to a value of 8, defining a minimum time-out period of 256 system clock cycles. The logic which forces the minimum value to 8 does not affect the contents of this field in the register. For $SWT = n$, then time-out period = 2^n system clock cycles, $n = 8, 9, \dots, 31$.

16.2.2.2 Software Watchdog Timer Service Register (SWTSR)

- The SWTCR[SWRI] can specify a sequence of responses. Upon the first time-out, the watchdog timer interrupt is asserted. If the watchdog timer interrupt flag is not cleared before a second time-out occurs, the watchdog timer asserts the system reset signal in response to the second time-out. This configuration supports a more graceful response to watchdog time-outs: first attempting an interrupt to notify the system, if that fails, generating a system reset.

The software service sequence must be performed using the SWTSR as a data register to prevent a SWT time-out. The service sequence requires two writes to this data register: first a write of 0x55 followed by a write of 0xAA. Both writes must be performed in this order prior to the SWT time-out, but any number of instructions or accesses to the SWSR may occur between the two writes. If the SWT has already timed out, writing to this register has no effect in negating the SWT interrupt or reset. [Figure 16-2](#) illustrates the SWTSR.

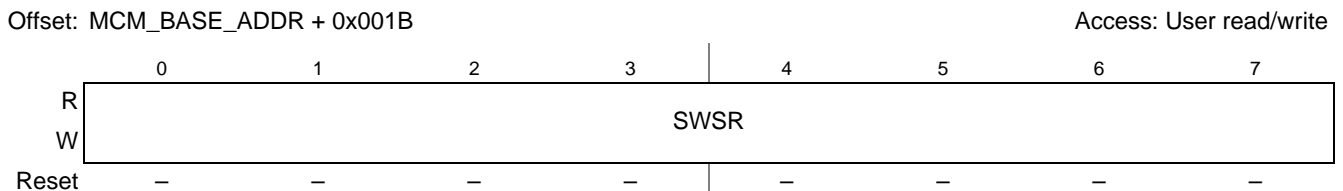


Figure 16-2. Software Watchdog Timer Service Register (SWTSR)

If the software watchdog timer is enabled (SWTCR[SWE] = 1), then any write of a data value other than 0x55 or 0xAA generates an immediate system reset, regardless of the value in the SWTCR[SWRI] field.

16.2.2.3 SWT Interrupt (SWTIR)

All interrupt requests associated with MCM are collected in the SWTIR register. This includes the software watchdog timer interrupt.

During the appropriate interrupt service routine handling these requests, the interrupt source contained in the SWTIR must be explicitly cleared (see [Figure 16-3](#) and [Table 16-4](#)).

For certain values of the SWTCR[SWRI] field, the software watchdog timer generates an interrupt response to a time-out. For these configurations, the SWTIR provides the program-visible interrupt request from the software watchdog timer.

Offset: MCM_BASE_ADDR + 0x001F

Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	SWTIC
W								w1c
Reset	0	0	0	0	0	0	0	0

Figure 16-3. SWT Interrupt (SWTIR)

Table 16-4. SWTIR Field Descriptions

Field	Description
bits 0–6	Reserved.
SWTIC	Software Watchdog Interrupt Flag. 0 A SWT interrupt has not occurred. 1 A SWT interrupt has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.

16.2.2.4 Miscellaneous User-Defined Control Register (MUDCR)

The MUDCR provides a program-visible register. On MPC5510, one bit is implemented. The PRI bit determines whether the AXBS-lite uses a fixed or round robin priority arbitration scheme for masters requesting access to AXBS-lite slave ports. See [Figure 16-4](#) and [Table 16-5](#) for the miscellaneous user-defined control register definition.

Offset: MCM_BASE_ADDR + 0x0024

Access: User read/write

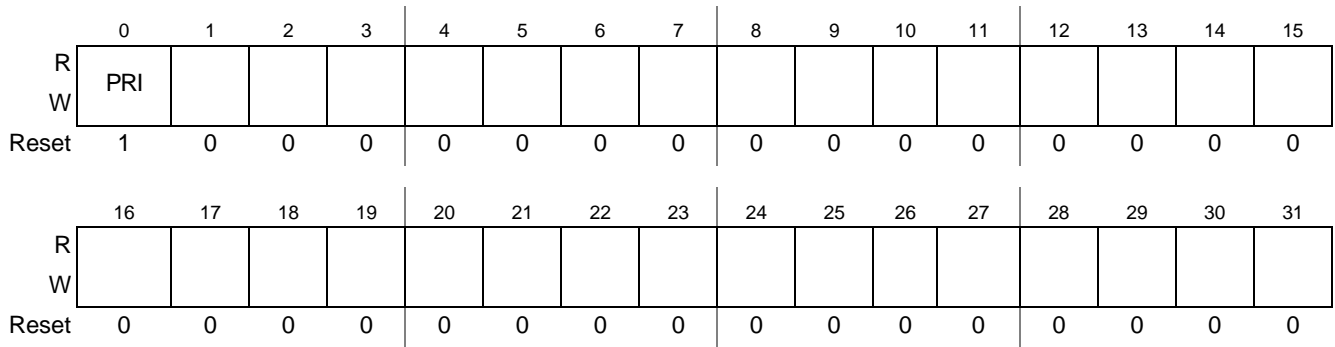


Figure 16-4. Miscellaneous User-Defined Control Register (MUDCR)

Table 16-5. MUDCR Field Descriptions

Field	Description
PRI	AXBS-lite arbitration priority scheme. 0 Fixed priority arbitration. 1 Round Robin priority arbitration.
bits 1–31	Reserved. Note: These bits can be read and written; however, writing has no effect other than to set or clear the bits. Reading returns the values written to the bits.

16.2.2.5 ECC Registers

There are a number of registers for the sole purpose of reporting and logging of memory failures. This section describes those registers.

16.2.2.5.1 ECC Configuration Register (ECR)

The ECC configuration register is an 8-bit control register for specifying which types of memory errors are reported. In all systems with ECC, the occurrence of a non-correctable error causes the current access to be terminated with an error condition. In many cases, this error termination is reported directly by the initiating bus master. However, there are certain situations where the occurrence of this type of non-correctable error is not reported by the master. Examples include speculative instruction fetches that are discarded due to a change-of-flow operation and buffered operand writes. The ECC reporting logic in the MCM provides an optional error interrupt mechanism to signal all non-correctable memory errors. In addition to the interrupt generation, the MCM captures specific information (memory address, attributes and data, bus master number, etc.) that may be useful for subsequent failure analysis.

See [Figure 16-5](#) and [Table 16-6](#) for the ECC configuration register definition.

Offset: MCM_BASE_ADDR + 0x0043

Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	ERNCR	EFNCR
W								
Reset	0	0	0	0	0	0	0	0

Figure 16-5. ECC Configuration (ECR) Register

Table 16-6. ECR Field Descriptions

Field	Description
bits 0–5	Reserved.
ERNCR	Enable RAM Non-Correctable Reporting. The occurrence of a non-correctable multi-bit RAM error generates a MCM ECC interrupt request as signaled by the assertion of ESR[RNCE]. The faulting address, attributes, and data are also captured in the REAR, RESR, REMR, REAT, and REDR registers. 0 Reporting of non-correctable RAM errors is disabled. 1 Reporting of non-correctable RAM errors is enabled.
EFNCR	Enable Flash Non-Correctable Reporting. The occurrence of a non-correctable multi-bit flash error generates a MCM ECC interrupt request as signaled by the assertion of ESR[FNCE]. The faulting address, attributes, and data are also captured in the FEAR, FEMR, FEAT, and FEDR registers. 0 Reporting of non-correctable flash errors is disabled. 1 Reporting of non-correctable flash errors is enabled.

16.2.2.5.2 ECC Status Register (ESR)

The ECC status register is an 8-bit control register for signaling which types of properly-enabled ECC events have been detected. The ESR signals the last properly-enabled memory event to be detected. An ECC interrupt request is asserted if any flag bit is asserted and its corresponding enable bit is asserted.

The MCM allows a maximum of one bit of the ESR to be asserted at any given time. This preserves the association between the ESR and the corresponding address and attribute registers, which are loaded on each occurrence of an properly-enabled ECC event. If there is a pending ECC interrupt and another properly-enabled ECC event occurs, the MCM hardware automatically handles the ESR reporting, clearing the previous data and loading the new state and thus guaranteeing that only a single flag is asserted.

To maintain the coherent software view of the reported event, the following sequence in the MCM error interrupt service routine is suggested:

1. Read the ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ESR and verify the current contents matches the original contents. If the two values are different, repeat from step one.
4. When the values are identical, write a 1 to the asserted ESR flag to negate the interrupt request.

See [Figure 16-6](#) and [Table 16-7](#) for the ECC status register definition.

Offset: MCM_BASE_ADDR + 0x0047

Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	RNCE	FNCE
W							w1c	w1c
Reset	0	0	0	0	0	0	0	0

Figure 16-6. ECC Status (ESR) Register

Table 16-7. ESR Field Descriptions

Field	Description
bits 0–5	Reserved.
RNCE	RAM Non-Correctable Error. The occurrence of a properly-enabled non-correctable RAM error generates a MCM ECC interrupt request. The faulting address, attributes, and data are also captured in the REAR, RESR, REMR, REAT, and REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable non-correctable RAM error has been detected. 1 A reportable non-correctable RAM error has been detected.
FNCE	Flash Non-Correctable Error. The occurrence of a properly-enabled non-correctable flash error generates a MCM ECC interrupt request. The faulting address, attributes and data are also captured in the FEAR, FEMR, FEAT, and FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable non-correctable flash error has been detected. 1 A reportable non-correctable flash error has been detected.

If both a flash and RAM non-correctable error occur at the same time, the MCM records the event with the highest priority, RNCE, and finally FNCE.

16.2.2.5.3 ECC Error Generation Register (EEGR)

The ECC error generation register is a 16-bit control register used to force the generation of single- and double-bit data inversions in the memories with ECC, most notably the RAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for injecting errors into the memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

The intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit noncorrectable errors that are terminated with an error response.

See [Figure 16-7](#) and [Table 16-8](#) for the ECC error generation register definition.

Offset: MCM_BASE_ADDR + 0x004A

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FRCNCI	FR1NCI	0	ERRBIT						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-7. ECC Error Generation (EEGR) Register

Table 16-8. EEGR Field Descriptions

Field	Description
bits 0–5	Reserved.
FRCNCI	<p>Force RAM Continuous Noncorrectable Data Inversions. The assertion of this bit forces the RAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT and the overall odd parity bit, continuously on every write operation.</p> <p>After this bit has been enabled to generate another continuous noncorrectable data inversion, it must be cleared before being set again to properly re-enable the error generation logic.</p> <p>The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM.</p> <p>0 No RAM continuous 2-bit data inversions are generated. 1 2-bit data inversions in the RAM are continuously generated.</p>
FR1NC	<p>Force RAM One Noncorrectable Data Inversions. The assertion of this bit forces the RAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT and the overall odd parity bit, on the first write operation after this bit is set.</p> <p>The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM.</p> <p>After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic.</p> <p>0 No RAM single 2-bit data inversions are generated. 1 One 2-bit data inversion in the RAM is generated.</p>
bit 8	Reserved.
ERRBIT	<p>Error Bit Position. The vector defines the bit position, which is complemented to create the data inversion on the write operation. For the creation of 2-bit data inversions, the bit specified by this field plus the odd parity bit of the ECC code are inverted.</p> <p>The RAM controller follows a vector bit ordering scheme where LSB=0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the RAM width. For example, consider a 32-bit RAM implementation.</p> <p>The 32-bit ECC approach requires seven code bits for a 32-bit word. For RAM data width of 32 bits, the actual SRAM (32b data + 7b for ECC= 39 bits). The following association between the ERRBIT field and the corrupted memory bit is defined:</p> <pre> if ERRBIT = 0, then RAM[0] is inverted if ERRBIT = 1, then RAM[1] is inverted ... if ERRBIT = 31, then RAM[31] is inverted if ERRBIT = 64, then ECC Parity[0] is inverted if ERRBIT = 65, then ECC Parity[1] is inverted ... if ERRBIT = 70, then ECC Parity[6] is inverted </pre> <p>Note: For ERRBIT values of 32 to 63 and greater than 70, no bit position is inverted.</p>

NOTE

If an attempt to force a non-correctable inversion (by asserting EEGR[FRCNCI] or EEGR[FRC1NCI]) and EEGR[ERRBIT] equals 64, then no data inversion will be generated.

The only allowable values for the 2 control bit enables {FRCNCI, FR1NCI} are {0,0}, {1,0} and {0,1}. All other values result in undefined behavior.

16.2.2.5.4 Flash ECC Address Register (FEAR)

The FEAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC configuration register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT, and FEDR registers and also the appropriate flag (F1BC or FNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See [Figure 16-8](#) and [Table 16-9](#) for the flash ECC address register definition.

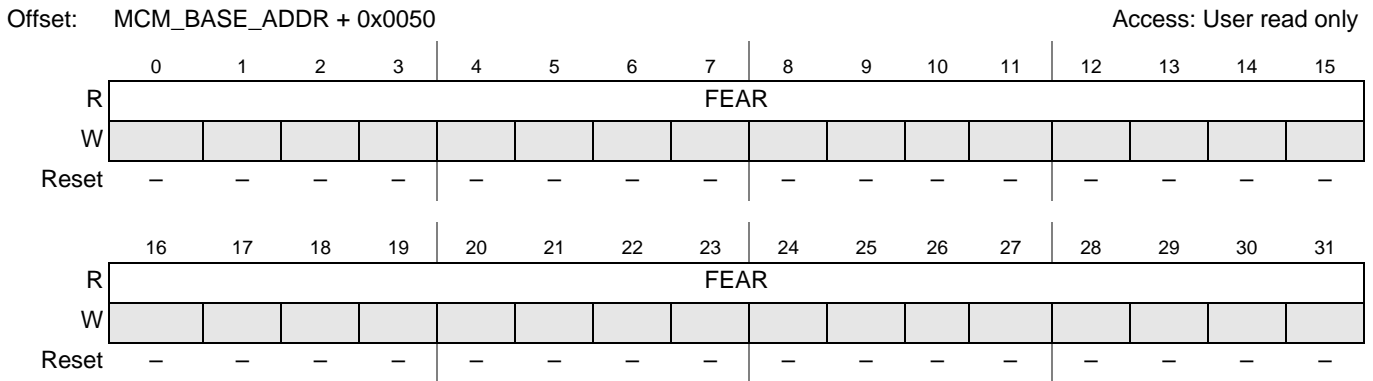


Figure 16-8. Flash ECC Address (FEAR) Register

Table 16-9. FEAR Field Descriptions

Field	Description
FEAR	Flash ECC Address Register. Contains the faulting access address of the last, properly-enabled flash ECC event.

16.2.2.5.5 Flash ECC Master Number Register (FEMR)

The FEMR is a 4-bit register for capturing the AXBS bus master number of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT, and FEDR registers and also the appropriate flag (FNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See [Figure 16-9](#) and [Table 16-10](#) for the flash ECC master number register definition.

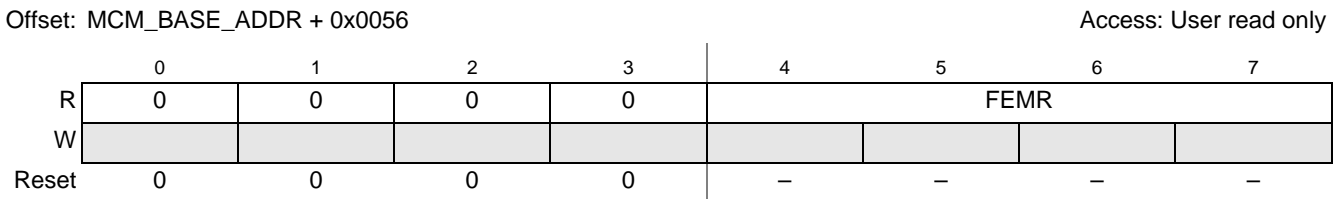


Figure 16-9. Flash ECC Master Number (FEMR) Register

Table 16-10. FEMR Field Descriptions

Field	Description
bits 0–3	Reserved.
FEMR	Flash CC Master Number Register. Contains the AXBS bus master number of the faulting access of the last, properly-enabled flash ECC event.

16.2.2.5.6 Flash ECC Attributes Register (FEAT)

The FEAT is an 8-bit register for capturing the AXBS bus master attributes of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT, and FEDR registers and also the appropriate flag (FNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See [Figure 16-10](#) and [Table 16-11](#) for the flash ECC attributes register definition.

Offset: MCM_BASE_ADDR + 0x0057

Access: User read only

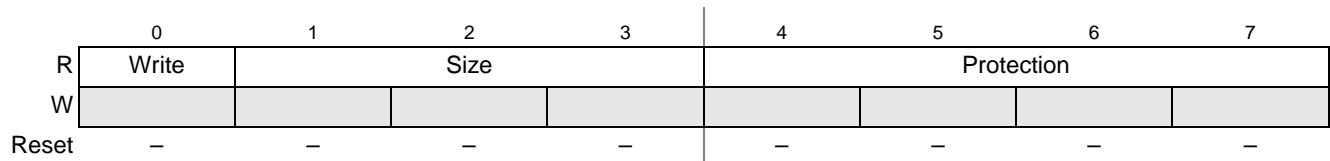


Figure 16-10. Flash ECC Attributes (FEAT) Register

Table 16-11. FEAT Field Descriptions

Field	Description
Write	0 Read access 1 Write access
Size	000 8-bit access 001 16-bit access 010 32-bit access 011 64-bit access 1xx Reserved
Protection	Cache: 0xxx Non-cacheable 1xxx Cacheable Buffer: x0xx Non-bufferable x1xx Bufferable Mode: xx0x User mode xx1x Supervisor mode Type: xxx0 I-Fetch xxx1 Data

16.2.2.5.7 Flash ECC Data Register (FEDR)

The FEDR is a 32-bit register for capturing the data associated with the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC configuration register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT, and FEDR registers and also the appropriate flag (FNCE) in the ECC status register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

Since the Flash performs ECC checking on a 64-bit double word, the 32-bit word captured in the FEDR register may not be the word that contained the error.

This register is read-only; any attempted write is ignored. See [Figure 16-11](#) and [Table 16-12](#) for the flash ECC data register definition.

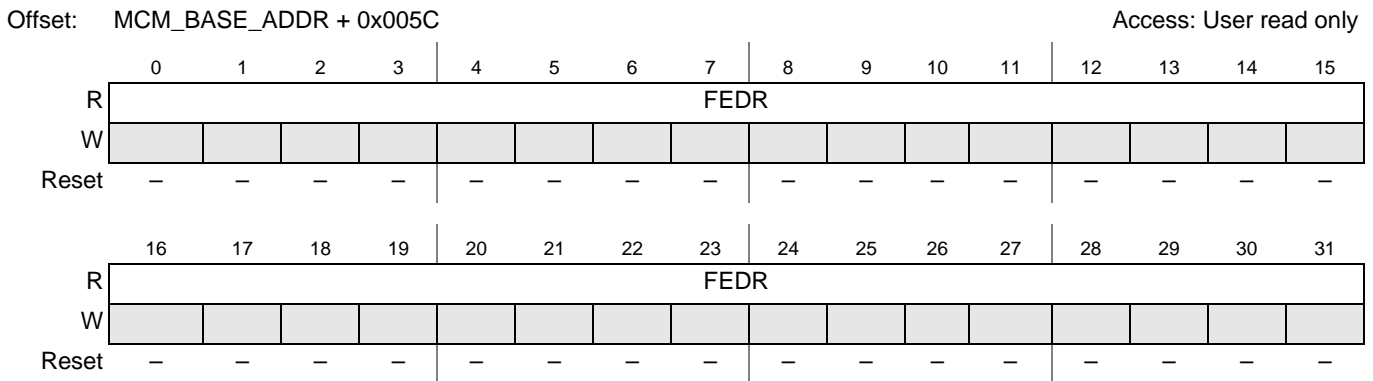


Figure 16-11. Flash ECC Data (FEDR) Register

Table 16-12. FEDR Field Descriptions

Field	Description
FEDR	Flash ECC Data Register. Contains the data associated with the faulting access of the last, properly-enabled flash ECC event. The register contains the data value taken directly from the data bus.

16.2.2.5.8 RAM ECC Address Register (REAR)

The REAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC configuration register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT, and REDR registers and also the appropriate flag (RNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See [Figure 16-12](#) and [Table 16-13](#) for the RAM ECC address register definition.

Offset: MCM_BASE_ADDR + 0x0060

Access: User read only

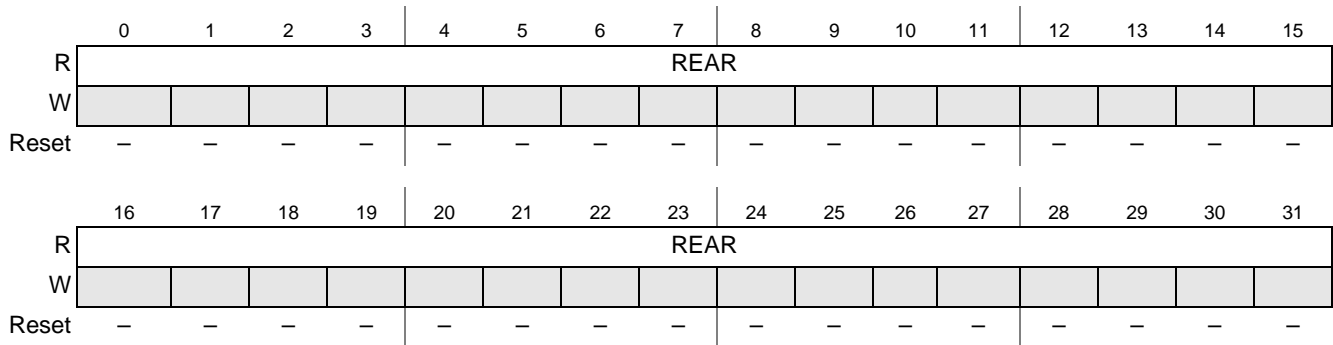


Figure 16-12. RAM ECC Address (REAR) Register

Table 16-13. REAR Field Descriptions

Field	Description
REAR	RAM ECC Address Register. Contains the faulting access address of the last, properly-enabled RAM ECC event.

16.2.2.5.9 RAM ECC Master Number Register (REMR)

The REMR is a 4-bit register for capturing the AXBS bus master number of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC configuration register, an ECC event in the RAM causes the address, attributes, and data associated with the access to be loaded into the REAR, RESR, REMR, REAT, and REDR registers and also the appropriate flag (RNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See [Figure 16-13](#) and [Table 16-14](#) for the RAM ECC master number register definition.

Offset: MCM_BASE_ADDR + 0x0066

Access: User read only

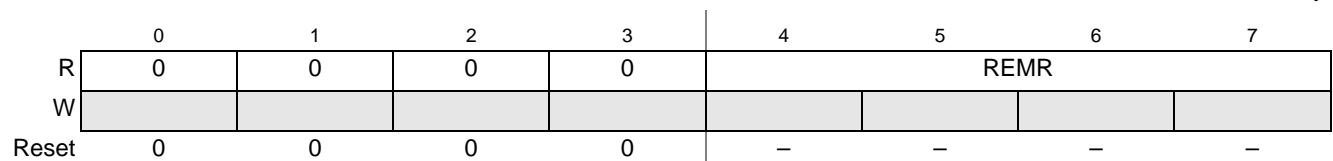


Figure 16-13. RAM ECC Master Number (REMR) Register

Table 16-14. REMR Field Descriptions

Field	Description
bits 0–3	Reserved
REMR	RAM ECC Master Number Register. Contains the AXBS bus master number of the faulting access of the last, properly-enabled RAM ECC event.

16.2.2.5.10 RAM ECC Attributes Register (REAT)

The REAT is an 8-bit register for capturing the AXBS bus master attributes of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC configuration register, an ECC event

in the RAM causes the address, attributes, and data associated with the access to be loaded into the REAR, RESR, REMR, REAT, and REDR registers and also the appropriate flag (RNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See [Figure 16-14](#) and [Table 16-15](#) for the RAM ECC attributes register definition.

Offset: MCM_BASE_ADDR + 0x0067

Access: User read only

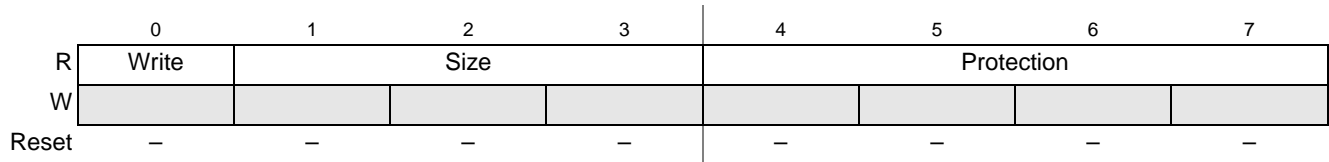


Figure 16-14. RAM ECC Attributes (REAT) Register

Table 16-15. REAT Field Descriptions

Field	Description
Write	0 Read access 1 Write access
Size	000 8-bit access 001 16-bit access 010 32-bit access 011 64-bit access 1xx Reserved
Protection	Cache: 0xxx Non-cacheable 1xxx Cacheable Buffer: x0xx Non-bufferable x1xx Bufferable Mode: xx0x User mode xx1x Supervisor mode Type: xxx0 I-Fetch xxx1 Data

16.2.2.5.11 RAM ECC Data Register (REDR)

The REDR is a 32-bit register for capturing the data associated with the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC configuration register, an ECC event in the RAM causes the address, attributes, and data associated with the access to be loaded into the REAR, RESR, REMR, REAT, and REDR registers and also the appropriate flag (RNCE) in the ECC status register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register is read-only; any attempted write is ignored. See [Figure 16-15](#) and [Table 16-16](#) for the RAM ECC data register definition.

Offset: MCM_BASE_ADDR + 0x006C

Access: User read only

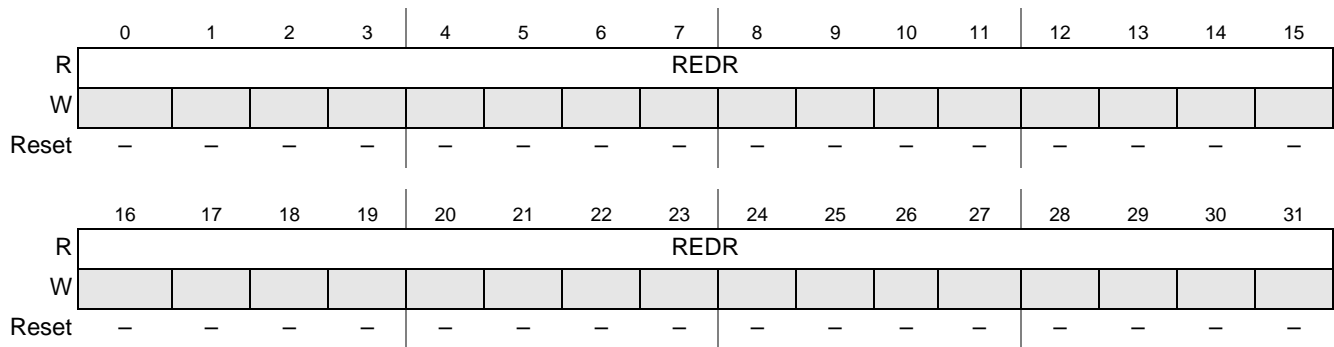


Figure 16-15. RAM ECC Data (REDR) Register

Table 16-16. REDR Field Descriptions

Field	Description
REDR	RAM ECC Data Register. Contains the data associated with the faulting access of the last, properly-enabled RAM ECC event. The register contains the data value taken directly from the data bus.

16.3 Functional Description

16.3.1 High-Priority Enables

The MCM contains an output to each core which are used with the AXBS-lite to elevate the priority of interrupt service routine accesses in the system bus controllers' arbitration schemes.

The core processors are configured to support critical and/or external interrupts. Furthermore, each processor can be configured to employ priority elevation on critical and/or external interrupt events. Critical interrupts come from outside the device and are routed directly to the processor's critical interrupt input. External interrupts are routed through the interrupt controller. In addition to the interrupt notification signals, various processor-specific configuration flags from the processor's machine check register (MCR[ee,ce]) and the hardware implementation register (HID1) are sent to the MCM to determine when interrupt servicing is enabled and when high-priority elevation should be enabled. If the corresponding processor is configured to allow high-priority elevation on critical interrupt events, the MCM generates the high-priority signal upon critical interrupt detection and holds it active throughout the duration of interrupt servicing. If the corresponding processor is configured to allow high-priority elevation on external interrupt events, the MCM generates the high-priority signal upon external interrupt detection and holds it active throughout the duration of interrupt servicing.

Be careful when using the priority elevation as it can enable a master to starve the rest of the masters in the system. Reference [Chapter 15, "Crossbar Switch \(XBAR\),"](#) for information on priority elevation and the Z1 and Z0 Core Reference Manual for information on the use of the interrupts.

Chapter 17

Memory Protection Unit (MPU)

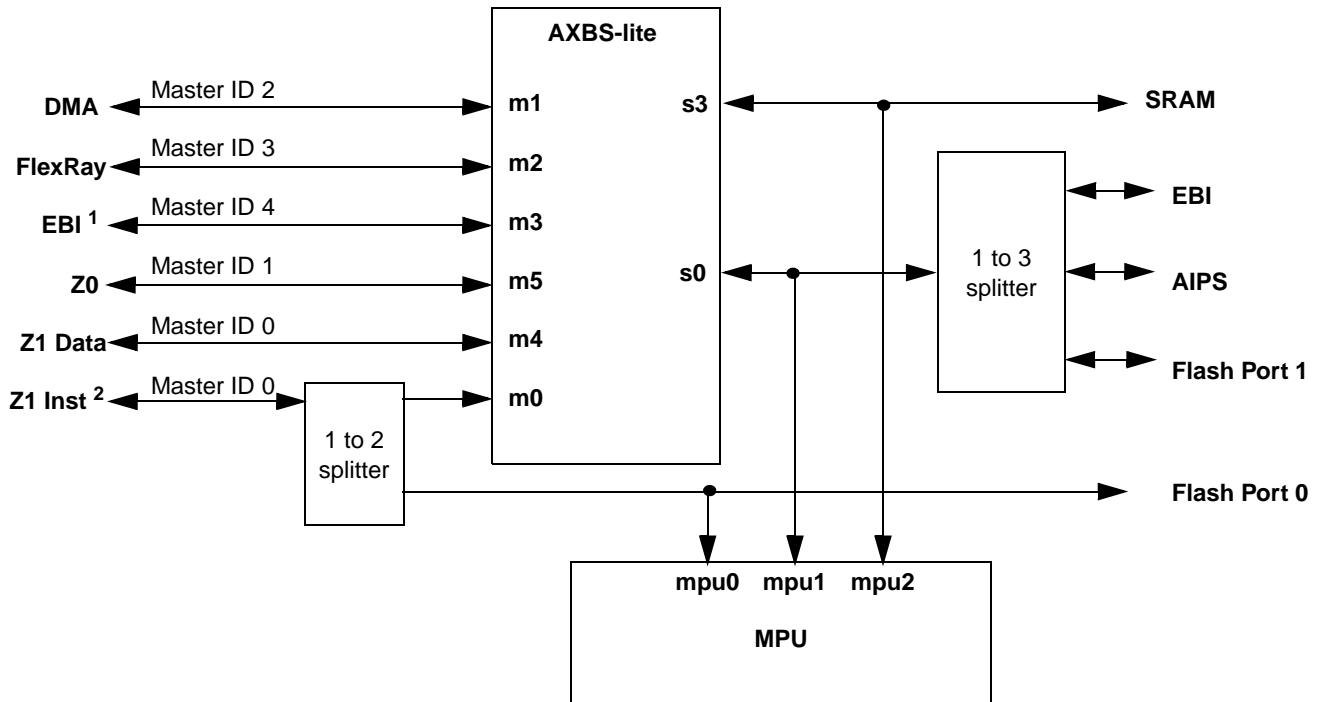
17.1 Introduction

The memory protection unit (MPU) provides hardware access control for all memory references generated in a device. Using pre-programmed region descriptors that define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. Memory references with sufficient access control rights are allowed to complete, but references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The MPU implements a set of program-visible region descriptors that monitor all system bus addresses. The result is a hardware structure with a two-dimensional connection matrix, where the region descriptors represent one dimension and the individual system bus addresses and attributes are the second dimension.

17.1.1 Block Diagram

A simplified block diagram illustrates how the MPU block is connected to the three AXBS-lite slave ports, one of them being the shared slave port splitter (see [Figure 17-1](#)).



¹ For factory test only.

² For Z1, all instruction accesses to flash go through port P0. The path from the Z1 Instruction bus through the 1 to 2 splitter and AXBS port m0 is only used for non-Flash (i.e. RAM and BAM) instruction fetches.

Figure 17-1. MPU Connections to AXBS-lite

17.1.2 Features

The MPU has these major features:

- Support for 16 memory region descriptors, each 128 bits in size
 - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB
 - MPU is invalid at reset, thus no access restrictions are enforced
 - Two types of access control definitions: two processor core bus masters (e200z1 and e200z0) support the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses; the remaining three non-core bus masters (DMA, FlexRay, and EBI) support {read, write} attributes
 - Automatic hardware maintenance of the region descriptor valid bit removes issues associated with maintaining a coherent image of the descriptor
 - Alternate memory view of the access control word for each descriptor provides an efficient mechanism to dynamically alter the access rights of a descriptor only

- For overlapping region descriptors, priority is given to permission granting over access denying as this approach provides more flexibility to system software
- Support for three AHB slave port connections
 - Flash port 0, shared flash port1/EBI/AIPS, and system RAM.
 - MPU hardware monitors every AHB slave port access using the pre-programmed memory region descriptors
 - An access protection error is detected if a memory reference does not hit in any memory region or the reference is flagged as illegal in all memory regions where it does hit; in the event of an access error, the AHB reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device
 - 64-bit error registers, one for each AHB slave port, capture the last faulting address, attributes, and detail information

17.1.3 Modes of Operation

The MPU does not support any special modes of operation.

17.2 Signal Description

The MPU does not include any external signals.

17.3 Memory Map and Registers

This section provides a detailed description of all MPU registers.

17.3.1 Module Memory Map

The MPU memory map is shown in [Table 17-1](#). The address of each register is given as an offset to the MPU base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

The MPU registers can be referenced using 32-bit (word) accesses only. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (for example, a write to a read-only register or a read of a write-only register) generate an error termination.

Table 17-1. MPU Memory Map

Offset from MPU_BASE (0xFFFF1_4000)	Register	Access	Reset Value	Section/Page
0x0000	MPU_CESR — MPU control/error status register	R/W	0x0080_3200	17.3.2.1/17-5
0x0004–0x000F	Reserved			
0x0010	MPU_EAR0 — MPU error address register, slave port 0	R	— ¹	17.3.2.2/17-6
0x0014	MPU_EDR0 — MPU error detail register, slave port 0	R	— ¹	17.3.2.3/17-7
0x0018	MPU_EAR1 — MPU error address register, slave port 1	R	— ¹	17.3.2.2/17-6

Table 17-1. MPU Memory Map (continued)

Offset from MPU_BASE (0xFFFF1_4000)	Register	Access	Reset Value	Section/Page
0x001C	MPU_EDR1 — MPU error detail register, slave port 1	R	— ¹	17.3.2.3/17-7
0x0020	MPU_EAR2 — MPU error address register, slave port 2	R	— ¹	17.3.2.2/17-6
0x0024	MPU_EDR2 — MPU error detail register, slave port 1	R	— ¹	17.3.2.3/17-7
0x0028–0x03FF	Reserved			
0x0400	MPU_RGD0 — MPU region descriptor 0	R/W	— ¹	17.3.2.4/17-7
0x0410	MPU_RGD1 — MPU region descriptor 1	R/W	— ¹	17.3.2.4/17-7
0x0420	MPU_RGD2 — MPU region descriptor 2	R/W	— ¹	17.3.2.4/17-7
0x0430	MPU_RGD3 — MPU region descriptor 3	R/W	— ¹	17.3.2.4/17-7
0x0440	MPU_RGD4 — MPU region descriptor 4	R/W	— ¹	17.3.2.4/17-7
0x0450	MPU_RGD5 — MPU region descriptor 5	R/W	— ¹	17.3.2.4/17-7
0x0460	MPU_RGD6 — MPU region descriptor 6	R/W	— ¹	17.3.2.4/17-7
0x0470	MPU_RGD7 — MPU region descriptor 7	R/W	— ¹	17.3.2.4/17-7
0x0480	MPU_RGD8 — MPU region descriptor 8	R/W	— ¹	17.3.2.4/17-7
0x0490	MPU_RGD9 — MPU region descriptor 9	R/W	— ¹	17.3.2.4/17-7
0x04A0	MPU_RGD10 — MPU region descriptor 10	R/W	— ¹	17.3.2.4/17-7
0x04B0	MPU_RGD11 — MPU region descriptor 11	R/W	— ¹	17.3.2.4/17-7
0x04C0	MPU_RGD12 — MPU region descriptor 12	R/W	— ¹	17.3.2.4/17-7
0x04D0	MPU_RGD13 — MPU region descriptor 13	R/W	— ¹	17.3.2.4/17-7
0x04E0	MPU_RGD14 — MPU region descriptor 14	R/W	— ¹	17.3.2.4/17-7
0x04F0	MPU_RGD15 — MPU region descriptor 15	R/W	— ¹	17.3.2.4/17-7
0x00500–0x07FF	Reserved			
0x0800	MPU_RGDAAC0 — MPU RGD alternate access control 0	W	— ¹	17.3.2.5/17-12
0x0804	MPU_RGDAAC1 — MPU RGD alternate access control 1	W	— ¹	17.3.2.5/17-12
0x0808	MPU_RGDAAC2 — MPU RGD alternate access control 2	W	— ¹	17.3.2.5/17-12
0x080C	MPU_RGDAAC3 — MPU RGD alternate access control 3	W	— ¹	17.3.2.5/17-12
0x0810	MPU_RGDAAC4 — MPU RGD alternate access control 4	W	— ¹	17.3.2.5/17-12
0x0814	MPU_RGDAAC5 — MPU RGD alternate access control 5	W	— ¹	17.3.2.5/17-12
0x0818	MPU_RGDAAC6 — MPU RGD alternate access control 6	W	— ¹	17.3.2.5/17-12
0x081C	MPU_RGDAAC7 — MPU RGD alternate access control 7	W	— ¹	17.3.2.5/17-12
0x0820	MPU_RGDAAC8 — MPU RGD alternate access control 8	W	— ¹	17.3.2.5/17-12
0x0824	MPU_RGDAAC9 — MPU RGD alternate access control 9	W	— ¹	17.3.2.5/17-12

Table 17-1. MPU Memory Map (continued)

Offset from MPU_BASE (0xFFFF1_4000)	Register	Access	Reset Value	Section/Page
0x0828	MPU_RGDAAC10 — MPU RGD alternate access control 10	W	— ¹	17.3.2.5/17-12
0x082C	MPU_RGDAAC11 — MPU RGD alternate access control 11	W	— ¹	17.3.2.5/17-12
0x0830	MPU_RGDAAC12 — MPU RGD alternate access control 12	W	— ¹	17.3.2.5/17-12
0x0834	MPU_RGDAAC13 — MPU RGD alternate access control 13	W	— ¹	17.3.2.5/17-12
0x0838	MPU_RGDAAC14 — MPU RGD alternate access control 14	W	— ¹	17.3.2.5/17-12
0x083C	MPU_RGDAAC15 — MPU RGD alternate access control 15	W	— ¹	17.3.2.5/17-12
0x0840–0x08FF	Reserved			

¹ See register definition.

17.3.2 Register Descriptions

This section lists the MPU registers in address order and describes the registers and their bit fields.

17.3.2.1 MPU Control/Error Status Register (MPU_CESR)

The MPU_CESR provides one byte of error status and three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Offset: MPU_BASE+0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		
R	MPERR ¹	0	0	0	0	0	0	1	0	0	0	HRL				NSP				NRGD				0	0	0	0	0	0	0	0	0	0	V
W	w1c																															L		
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	D		

Figure 17-2. MPU Control/Error Status Register (MPU_CESR)

¹ Each MPERR bit can be cleared by writing a one to the bit location.

Table 17-2. MPU_CESR Field Descriptions

Field	Description
MPERR	<p>MPU Port <i>n</i> Error, where the MPU port number matches the bit number. Each bit in this read-only field represents a flag maintained by the MPU for signaling the presence of a captured error contained in the MPU_EAR<i>n</i> and MPU_EDR<i>n</i> registers. The individual bit is set when the hardware detects an error and records the faulting address and attributes. It is cleared when the corresponding bit is written to a logical one. If another error is captured at the exact same cycle as a write of a logical one, this flag remains set. A find-first-one instruction (or equivalent) can be used to detect the presence of a captured error.</p> <p>0 The corresponding MPU_EAR<i>n</i>/MPU_EDR<i>n</i> registers do not contain an unread captured error 1 The corresponding MPU_EAR<i>n</i>/MPU_EDR<i>n</i> registers do contain an unread captured error</p> <p>Note: Bit 0 indicates a flash port 0 access protection error, bit 1 a combined Flash Port 1/EBI/peripheral bridge protection error, and bit 3 an SRAM protection error.</p>
HRL	<p>Hardware Revision Level. This 4-bit read-only field specifies the MPU's hardware and definition revision level. It can be read by software to determine the functional definition of the module. This field reads as 0 on MPC5510.</p>
NSP	<p>Number of MPU/Slave Ports. This 4-bit read-only field specifies the number of MPU/slave ports [1–8] connected to the MPU. This field reads as 0b0011 on MPC5510.</p>
NRGD	<p>Number of Region Descriptors. This 4-bit read-only field specifies the number of region descriptors implemented in the MPU. The defined encodings include: 0000 8 region descriptors 0010 16 region descriptors This field reads as 0b0010 on MPC5510</p>
VLD	<p>Valid. This bit provides a global enable/disable for the MPU. 0 The MPU is disabled 1 The MPU is enabled While the MPU is disabled, all accesses from all bus masters are allowed.</p>

17.3.2.2 MPU Error Address Register, MPU Port 0 to 2 (MPU_EAR*n*)

When the MPU detects an access error on MPU port *n*, the 32-bit reference address is captured in this read-only register and the corresponding bit in the MPU_CESR[MPERR] field set. Additional information about the faulting access is captured in the corresponding MPU_EDR*n* register at the same time.

Offset: MPU_BASE + 0x0010 (MPU_EAR0) Access: User read only
 MPU_BASE + 0x0018 (MPU_EAR1)
 MPU_BASE + 0x0020 (MPU_EAR2)

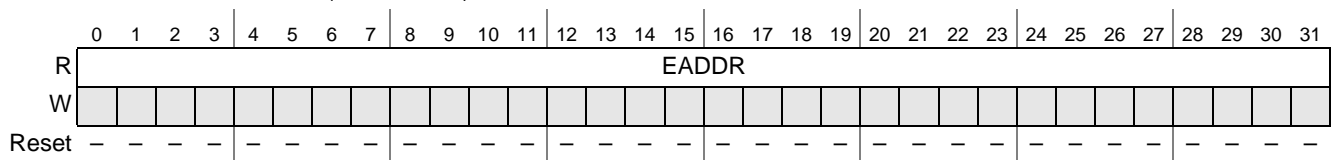


Figure 17-3. MPU Error Address Register, Slave Port *n* (MPU_EAR*n*)

Table 17-3. MPU_EAR*n* Field Descriptions

Field	Description
EADDR	<p>Error Address. This read-only field is the reference address from slave port <i>n</i> that generated the access error.</p>

17.3.2.3 MPU Error Detail Register, MPU Port 0 to 2 (MPU_EDR n)

When the MPU detects an access error on MPU port n , 32 bits of error detail are captured in this read-only register and the corresponding bit in the MPU_CESR[MPERR] field set. Information on the faulting address is captured in the corresponding MPU_EAR n register at the same time. A read of the MPU_EDR n register clears the corresponding bit in the MPU_CESR[MPERR] field.

Offset: MPU_BASE + 0x00014 (MPU_EDR0)

Access: User read only

MPU_BASE + 0x001C (MPU_EDR1)

MPU_BASE + 0x0024 (MPU_EDR2)

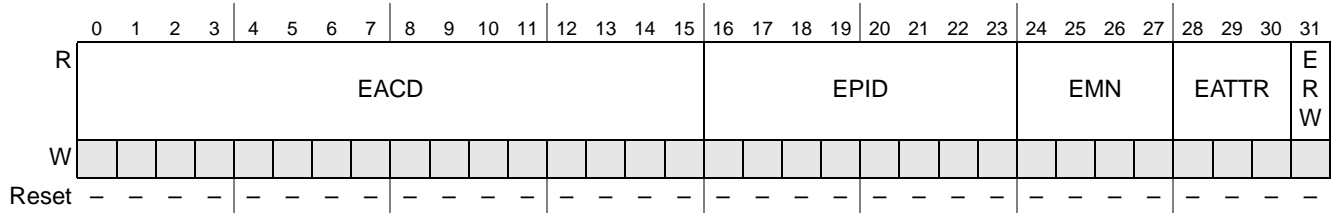


Figure 17-4. MPU Error Detail Register, Slave Port n (MPU_EDR n)

Table 17-4. MPU_EDR n Field Descriptions

Field	Description
EACD	Error Access Control Detail. This 16-bit read-only field implements one bit per region descriptor and is an indication of the region descriptor hit logically-ANDed with the access error indication. The MPU performs a reference-by-reference evaluation to determine the presence/absence of an access error. When an error is detected, the hit-qualified access control vector is captured in this field. If the MPU_EDR n register contains a captured error and the EACD field is all zeroes, this signals an access that did not hit in any region descriptor. All non-zero EACD values signal references that hit in a region descriptor(s), but failed due to a protection error as defined by the specific set bits.
EPID	Error Process Identification. This 8-bit read-only field records the process identifier of the faulting reference. The process identifier is typically driven by processor cores only; for other bus masters, this field is cleared.
EMN	Error Master Number. This 4-bit read-only field records the logical master number of the faulting reference. This field is used to determine the bus master that generated the access error.
EATTR	Error Attributes. This 3-bit read-only field records attribute information about the faulting reference. The supported encodings are defined as: 000 User mode, instruction access 001 User mode, data access 010 Supervisor mode, instruction access 011 Supervisor mode, data access All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (0b011).
ERW	Error Read/Write. This 1-bit read-only field signals the access type (read, write) of the faulting reference. 0 Read 1 Write

17.3.2.4 MPU Region Descriptor n (MPU_RGD n)

Each 128-bit (16 byte) region descriptor specifies a given memory space and the access attributes associated with that space. The descriptor definition is fundamental to the operation of the MPU.

The region descriptors are organized sequentially in the MPU’s programming model and each of the four 32-bit words are detailed in the subsequent sections.

17.3.2.4.1 MPU Region Descriptor *n*, Word 0 (MPU_RGD*n*.Word0)

The first word of the MPU region descriptor defines the 0-modulo-32 byte start address of the memory region. Writes to this word clear the region descriptor’s valid bit.

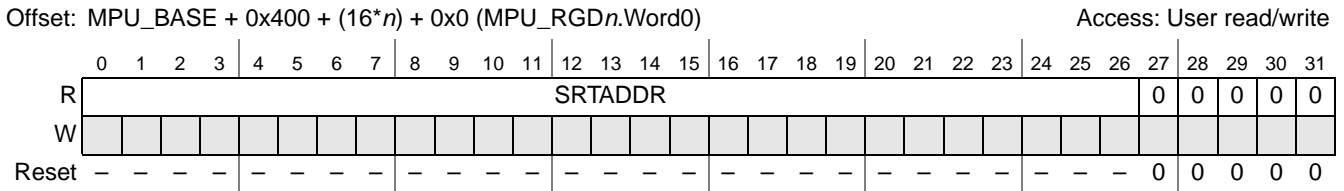


Figure 17-5. MPU Region Descriptor, Word 0 Register (MPU_RGD*n*.Word0)

Table 17-5. MPU_RGD*n*.Word0 Field Descriptions

Field	Description
SRTADDR	Start Address. This field defines the most significant bits of the 0-modulo-32 byte start address of the memory region.

17.3.2.4.2 MPU Region Descriptor *n*, Word 1 (MPU_RGD*n*.Word1)

The second word of the MPU region descriptor defines the 31-modulo-32 byte end address of the memory region. Writes to this word clear the region descriptor’s valid bit.

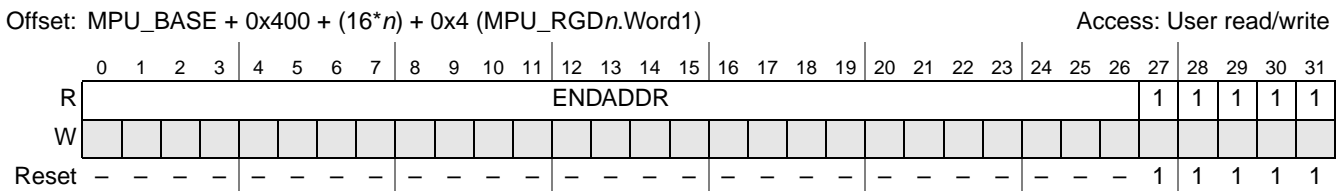


Figure 17-6. MPU Region Descriptor, Word 1 Register (MPU_RGD*n*.Word1)

Table 17-6. MPU_RGD*n*.Word1 Field Descriptions

Field	Description
ENDADDR	End Address. This field defines the most significant bits of the 31-modulo-32 byte end address of the memory region. There are no hardware checks to verify that ENDADDR ≥ SRTADDR; the software must properly load these region descriptor fields.

17.3.2.4.3 MPU Region Descriptor *n*, Word 2 (MPU_RGD*n*.Word2)

The third word of the MPU region descriptor defines the access control rights of the memory region. The access control privileges are dependent on two broad classifications of bus masters. Bus masters 0–3 are typically reserved for processor cores. The corresponding access control is a 6-bit field defining separate privilege rights for user and supervisor mode accesses as well as the optional inclusion of a process identification field within the definition. Bus masters 4–7 are typically reserved for data movement engines and their capabilities are limited to separate read and write permissions. For these fields, the bus master number refers to the logical master number defined as the AHB *hmaster*[3:0] signal.

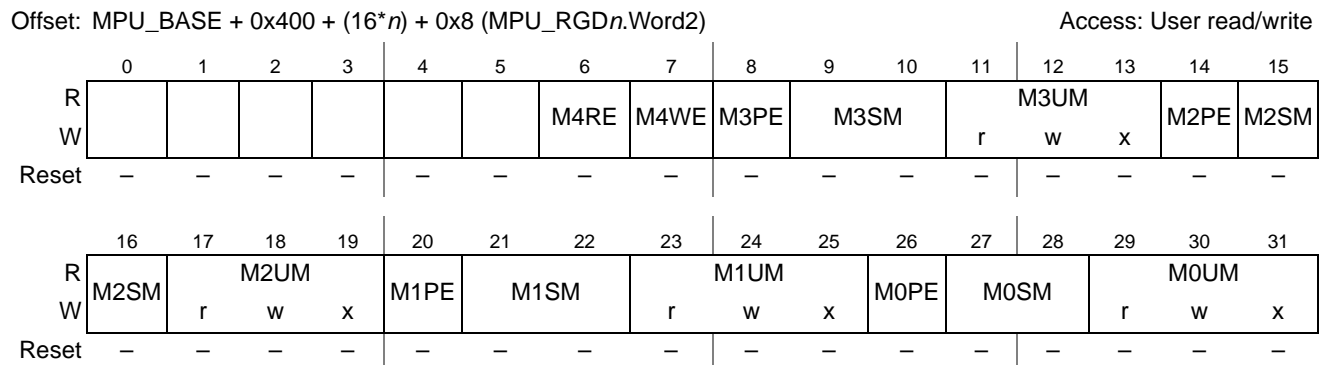
For the processor privilege rights, there are three flags associated with this function: {read, write, execute}. In this context, these flags follow the traditional definition:

- Read (r) permission refers to the ability to access the referenced memory address using an operand (data) fetch.
- Write (w) permission refers to the ability to update the referenced memory address using a store (data) instruction.
- Execute (x) permission refers to the ability to read the referenced memory address using an instruction fetch.

The evaluation logic defines the processor access type based on multiple AHB signals: read or write as specified by the `hwrite` signal and the low-order two bits of `hprot[1:0]`, which identify a data reference versus an instruction fetch and the operating mode (supervisor, user) of the requesting processor.

For non-processor data movement engines (bus masters 4–7), the evaluation logic simply uses `hwrite` to determine if the access is a read or write. The `hprot[1:0]` signal is ignored for these masters.

Writes to this word clear the region descriptor's valid bit. Because it is also expected that system software may adjust only the access controls within a region descriptor (`MPU_RGDn.Word2`) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to `MPU_RGDAACn` (alternate access control *n*) as stores to these locations do not affect the descriptor's valid bit.



Note: Refer to [Figure 17-1](#) to see the Master ID assignments.

Figure 17-7. MPU Region Descriptor, Word 2 Register (`MPU_RGDn.Word2`)

Table 17-7. `MPU_RGDn.Word2` Field Descriptions

Field	Description
bits 0–5	Reserved. Note: These bits must never be set.
M4RE	Bus Master ID 4 Read Enable. If set, this flag allows bus master ID 4 to perform read operations. If cleared, any attempted read by bus master ID 4 terminates with an access error and the read is not performed. Note: Bus Master 4 (EBI) is available for Factory Test only.

Table 17-7. MPU_RGDn.Word2 Field Descriptions (continued)

Field	Description
M4WE	Bus Master ID 4 Write Enable. If set, this flag allows bus master ID 4 to perform write operations. If cleared, any attempted write by bus master ID 4 terminates with an access error and the write is not performed. Note: Bus Master 4 (EBI) is available for Factory Test only.
M3PE	This bit can be read and written to either a 0 or 1, but the MPU will behave as if this bit was permanently tied to 0, so that the PID is not part of the region hit evaluation.
M3SM	Bus Master ID 3 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 3 when operating in supervisor mode. The M3SM field is defined as: 00 r, w, x = read, write and execute allowed 01 r, -, x = read and execute allowed, but no write 10 r, w, - = read and write allowed, but no execute 11 Same access controls as that defined by M3UM for user mode
M3UM	Bus Master ID 3 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: {r, w, x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M2PE	This bit can be read and written to either a 0 or 1, but the MPU will behave as if this bit was permanently tied to 0, so that the PID is not part of the region hit evaluation.
M2SM	Bus Master ID 2 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 2 when operating in supervisor mode. The M2SM field is defined as: 00 r, w, x = read, write and execute allowed 01 r, -, x = read and execute allowed, but no write 10 r, w, - = read and write allowed, but no execute 11 Same access controls as that defined by M2UM for user mode
M2UM	Bus Master ID 2 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 2 when operating in user mode. The M2UM field consists of three independent bits, enabling read, write, and execute permissions: {r, w, x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M1PE	Bus Master ID 1 Process Identifier Enable. If set, this flag specifies that the process identifier and mask defined in MPU_RGDn.Word3 are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
M1SM	Bus Master ID 1 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 1 when operating in supervisor mode. The M1SM field is defined as: 00 r, w, x = read, write and execute allowed 01 r, -, x = read and execute allowed, but no write 10 r, w, - = read and write allowed, but no execute 11 Same access controls as that defined by M1UM for user mode
M1UM	Bus Master ID 1 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 1 when operating in user mode. The M1UM field consists of three independent bits, enabling read, write, and execute permissions: {r, w, x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M0PE	Bus Master ID 0 Process Identifier Enable. If set, this flag specifies that the process identifier and mask defined in MPU_RGDn.Word3 are to be included in the region hit evaluation. If cleared, the region hit evaluation does not include the process identifier.

Table 17-7. MPU_RGDn.Word2 Field Descriptions (continued)

Field	Description
M0SM	Bus Master ID 0 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 0 when operating in supervisor mode. The M0SM field is defined as: 00 r, w, x = read, write and execute allowed 01 r, -, x = read and execute allowed, but no write 10 r, w, - = read and write allowed, but no execute 11 Same access controls as that defined by M0UM for user mode
M0UM	Bus Master ID 0 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write, and execute permissions: {r, w, x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

17.3.2.4.4 MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3)

The fourth word of the MPU region descriptor contains the optional process identifier and mask, plus the region descriptor's valid bit.

Because the region descriptor is a 128-bit entity, there are potential coherency issues as this structure is being updated because multiple writes are required to update the entire descriptor. Accordingly, the MPU hardware assists in the operation of the descriptor valid bit to prevent incoherent region descriptors from generating spurious access errors. In particular, it is expected that a complete update of a region descriptor is typically done with sequential writes to MPU_RGDn.Word0, then MPU_RGDn.Word1, ... and MPU_RGDn.Word3. The MPU hardware automatically clears the valid bit on any writes to words {0,1,2} of the descriptor. Writes to this word set/clear the valid bit in a normal manner.

Because it is also expected that system software may adjust the access controls within a region descriptor (MPU_RGDn.Word2) only as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation must be performed by writing to MPU_RGDAACn (alternate access control n) as stores to these locations do not affect the descriptor's valid bit.

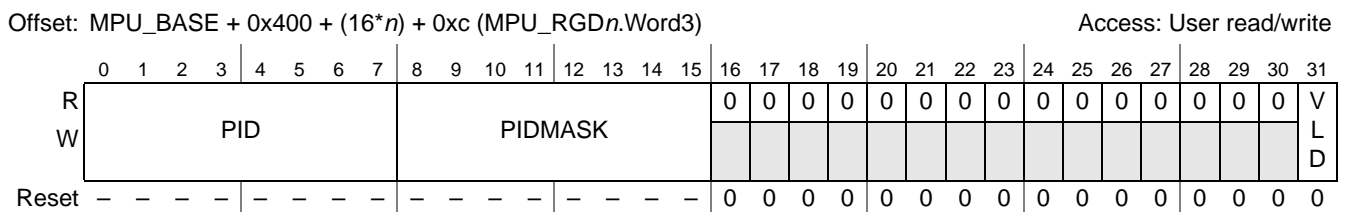


Figure 17-8. MPU Region Descriptor, Word 3 Register (MPU_RGDn.Word3)

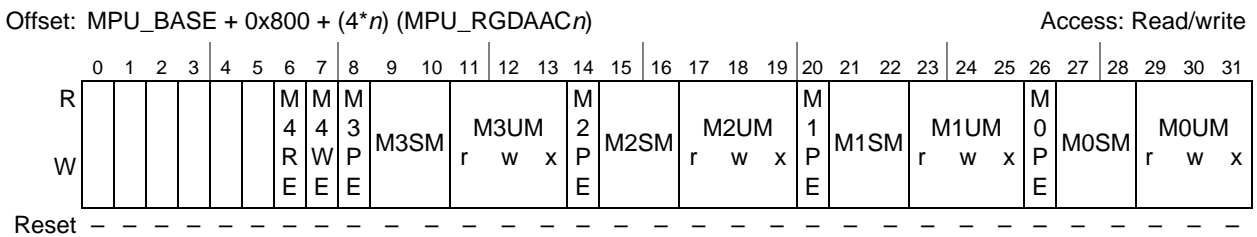
Table 17-8. MPU_RGDn.Word3 Field Descriptions

Field	Description
PID	Process Identifier. This 8-bit field specifies that the optional process identifier is to be included in the determination of whether the current access hits in the region descriptor. This field is combined with the PIDMASK and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set.
PIDMASK	Process Identifier Mask. This 8-bit field provides a masking capability so that multiple process identifiers can be included as part of the region hit determination. If a bit in the PIDMASK is set, the corresponding bit of the PID is ignored in the comparison. This field is combined with the PID and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set. For more information on the handling of the PID and PIDMASK, see Section 17.4.1.1, “Access Evaluation—Hit Determination.”
VLD	Valid. This bit signals the region descriptor is valid. Any write to MPU_RGDn.Word{0,1,2} clears this bit, but a write to MPU_RGDn.Word3 sets or clears this bit depending on bit 31 of the write operand. 0 Region descriptor is invalid 1 Region descriptor is valid

17.3.2.5 MPU Region Descriptor Alternate Access Control *n* (MPU_RGDAACn)

As noted in Section 17.3.2.4.3, “MPU Region Descriptor *n*, Word 2 (MPU_RGDn.Word2),” it is expected that because system software may adjust the access controls within a region descriptor (MPU_RGDn.Word2) only as different tasks execute, an alternate programming view of this 32-bit entity is desired. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAACn (alternate access control *n*) as stores to these locations do not affect the descriptor’s valid bit.

The memory address therefore provides an alternate location for updating MPU_RGDn.Word2.

Figure 17-9. MPU RGD Alternate Access Control *n* (MPU_RGDAACn)

Because the MPU_RGDAACn register is another memory mapping for MPU_RGDn.Word2, the field definitions shown in Table 17-9 are identical to those presented in Table 17-7.

Table 17-9. MPU_RGDAACn Field Descriptions

Field	Description
bits 0–5	Reserved. Note: These bits must never be set.
M4RE	Bus Master ID 4 Read Enable. If set, this flag allows bus master ID 4 to perform read operations. If cleared, any attempted read by bus master ID 4 terminates with an access error and the read is not performed.

Table 17-9. MPU_RGDAACn Field Descriptions (continued)

Field	Description
M4WE	Bus Master 4 Write Enable. If set, this flag allows bus master 4 to perform write operations. If cleared, any attempted write by bus master 4 terminates with an access error and the write is not performed.
M3PE	Bus Master 3 Process Identifier Enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, the region hit evaluation does not include the process identifier.
M3SM	Bus Master 3 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as: 00 r, w, x = read, write and execute allowed 01 r, -, x = read and execute allowed, but no write 10 r, w, - = read and write allowed, but no execute 11 Same access controls as that defined by M3UM for user mode
M3UM	Bus Master 3 User Mode Access Control. This 3-bit field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write, and execute permissions: {r, w, x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M2PE	Bus Master 2 Process Identifier Enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, the region hit evaluation does not include the process identifier.
M2SM	Bus Master 2 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as: 00 r, w, x = read, write and execute allowed 01 r, -, x = read and execute allowed, but no write 10 r, w, - = read and write allowed, but no execute 11 Same access controls as that defined by M2UM for user mode
M2UM	Bus Master 2 User Mode Access Control. This 3-bit field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of three independent bits, enabling read, write, and execute permissions: {r, w, x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
M1PE	Bus Master 1 Process Identifier Enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, the region hit evaluation does not include the process identifier.
M1SM	Bus Master 1 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master 1 when operating in supervisor mode. The M1SM field is defined as: 00 r, w, x = read, write and execute allowed 01 r, -, x = read and execute allowed, but no write 10 r, w, - = read and write allowed, but no execute 11 Same access controls as that defined by M1UM for user mode
M1UM	Bus Master 1 User Mode Access Control. This 3-bit field defines the access controls for bus master 1 when operating in user mode. The M1UM field consists of three independent bits, enabling read, write, and execute permissions: {r, w, x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.
MOPE	Bus Master 0 Process Identifier Enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.

Table 17-9. MPU_RGDAACn Field Descriptions (continued)

Field	Description
MOSM	Bus Master 0 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master 0 when operating in supervisor mode. The MOSM field is defined as: 00 r, w, x = read, write and execute allowed 01 r, -, x = read and execute allowed, but no write 10 r, w, - = read and write allowed, but no execute 11 Same access controls as that defined by MOUM for user mode
MOUM	Bus Master 0 User Mode Access Control. This 3-bit field defines the access controls for bus master 0 when operating in user mode. The MOUM field consists of three independent bits, enabling read, write, and execute permissions: {r, w, x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.

17.4 Functional Description

In this section, the functional operation of the MPU is detailed. In particular, subsequent sections discuss the operation of the access evaluation macro as well as the handling of error-terminated AHB bus cycles.

17.4.1 Access Evaluation Macro

As discussed, the basic operation of the MPU is performed in the access evaluation macro, a hardware structure replicated in the two-dimensional connection matrix. As shown in Figure 17-10, the access evaluation macro inputs the AHB system bus address phase signals (AHB_ap) and the contents of a region descriptor (RGDn) and performs two major functions: region hit determination (*hit_b*) and detection of an access protection violation (*error*).

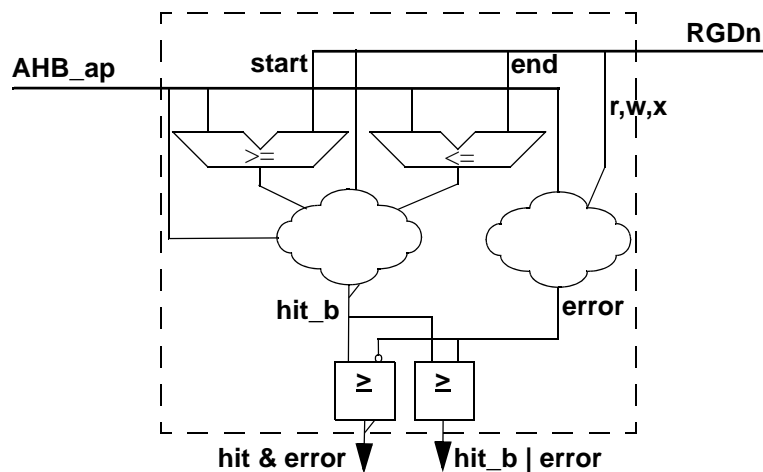


Figure 17-10. MPU Access Evaluation Macro

Figure 17-10 is not a schematic of the actual access evaluation macro, but a generalized block diagram showing the major functions included in this logic block.

17.4.1.1 Access Evaluation—Hit Determination

To determine if the current AHB reference hits in the given region, two magnitude comparators are used with the region's start and end addresses. The boolean equation for this portion of the hit determination is defined as:

```
region_hit =
    ((haddr[0:26] >= rgdn.srtaddr[0:26]) & (haddr[0:26] <= rgdn.endaddr[0:26]))
    & rgdn.vld
```

where `haddr[*]` is the current AHB reference address, `rgdn.srtaddr[*]` and `rgdn.endaddr[*]` are the start and end addresses, and `rgdn.vld` is the valid bit, all from region descriptor *n*. There are no hardware checks to verify that `rgdn.endaddr` \geq `rgdn.srtaddr`, and the software must properly load appropriate values into these fields of the region descriptor.

In addition to the algebraic comparison of the AHB reference address versus the region descriptor's start and end addresses, the optional process identifier is examined against the region descriptor's PID and PIDMASK fields. Using the `hmaster[*]` number to select the appropriate MxPE field from the region descriptor, a process identifier hit term is formed as:

```
pid_hit = ~rgdn.mxpe
    | ((current_pid[0:7] | rgdn.pidmask[0:7]) == (rgdn.pid[0:7] | rgdn.pidmask[0:7]))
```

where the `current_pid[*]` is the selected process identifier from the current bus master, and `rgdn.pid[*]` and `rgdn.pidmask[*]` are the appropriate process identifier fields from the region descriptor *n*. For AHB bus masters that do not output a process identifier, the MPU forces the `pid_hit` term to be asserted.

As shown in [Figure 17-10](#), the access evaluation macro forms the logical complement (`hit_b`) of the combined `region_hit` and `pid_hit` boolean equations.

17.4.1.2 Access Evaluation—Privilege Violation Determination

While the access evaluation macro is making the region hit determination, the logic is also evaluating if the current access is allowed by the permissions defined in the region descriptor. Using the AHB `hmaster[*]` and `hprot[1]` (supervisor/user mode) signals, a set of effective permissions (`eff_rgd[r,w,x]`) is generated from the appropriate fields in the region descriptor. The protection violation logic then evaluates the access against the effective permissions using the specification shown in [Table 17-10](#).

Table 17-10. Protection Violation Definition

Description	Inputs					Output
	<i>hwrite</i>	<i>hprot[0]</i>	<i>eff_rgd[r]</i>	<i>eff_rgd[w]</i>	<i>eff_rgd[x]</i>	Protection Violation?
inst fetch read	0	0	—	—	0	yes, no x permission
inst fetch read	0	0	—	—	1	no, access is allowed
data read	0	1	0	—	—	yes, no r permission
data read	0	1	1	—	—	no, access is allowed
data write	1	—	—	0	—	yes, no w permission
data write	1	—	—	1	—	no, access is allowed

The resulting boolean equation for the processor protection violations is:

```
cpu_protection_violation
= ~hwrite & ~hprot[0] & ~eff_rgdn[x]    // ifetch & no x
| ~hwrite & hprot[0] & ~eff_rgdn[r]    // data_read & no r
| hwrite & ~eff_rgdn[w]                // data_write & no w
```

The resulting boolean equation for the non-processor protection violations is:

```
protection_violation
= ~hwrite & ~eff_rgdn[r]                // data_read & no r
| hwrite & ~eff_rgdn[w]                // data_write & no w
```

As shown in [Figure 17-10](#), the output of the protection violation logic is the `error` signal, that is, `error = protection_violation`.

The access evaluation macro then uses the `hit_b` and `error` signals to form two outputs. The combined `(hit_b | error)` signal is used to signal the current access is not allowed and `(~hit_b & error)` is used as the input to `MPU_EDRn` (error detail register) in the event of an error.

17.4.2 Putting It All Together and AHB Error Terminations

For each AHB slave port being monitored, the MPU performs a reduction-AND of all the individual `(hit_b | error)` terms from each access evaluation macro. This expression then terminates the bus cycle with an error and reports a protection error for three conditions:

1. If the access does not hit in any region descriptor, a protection error is reported.
2. If the access hits in a single region descriptor and that region signals a protection violation, a protection error is reported.
3. If the access hits in multiple (overlapping) regions and all regions signal protection violations, then a protection error is reported.

The third condition reflects that priority is given to permission granting over access denying for overlapping regions as this approach provides more flexibility to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see [Section 17.6, “Application Information.”](#)

When the MPU causes a termination error to occur, the effect on the system depends on the bus master requesting the access. If the error was caused by a core access, a machine check is taken. If the error was caused by an eDMA access, an eDMA source or destination error occurs in the eDMA controller, which can be enabled to provide an interrupt request through the INTC. If the error was caused by a FlexRay access, a controller host interface (CHI) illegal system memory access error occurs in the FlexRay controller, which can be enabled to provide an interrupt request to the INTC.

17.5 Initialization Information

The reset state of `MPU_CESR[VLD]` disables the entire module. While the MPU is disabled, all accesses from all bus masters are allowed. This state also minimizes the power dissipation of the MPU. The power dissipation of each access evaluation macro is minimized when the associated region descriptor is marked as invalid or when `MPU_CESR[VLD] = 0`.

Typically the appropriate number of region descriptors (MPU_RGD n) are loaded at system startup, including the setting of the MPU_RGD n .Word3[VLD] bits, before MPU_CESR[VLD] is set, enabling the module. This approach allows all the loaded region descriptors to be enabled simultaneously. If a memory reference does not hit in any region descriptor, the attempted access is terminated with an error.

17.6 Application Information

In an application's system, interfacing with the MPU can generally be classified into the following activities:

1. Creation of a new memory region requires loading the appropriate region descriptor into an available register location. When a new descriptor is loaded into a RGD n , it would typically be performed using four 32-bit word writes. As discussed in [Section 17.3.2.4.4, "MPU Region Descriptor n, Word 3 \(MPU_RGD \$n\$.Word3\),"](#) the hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues associated with the multi-cycle descriptor writes. Deletion/removal of an existing memory region is performed by clearing MPU_RGD n .Word3[VLD].
2. If only the access rights for an existing region descriptor need to change, a 32-bit write to the alternate version of the access control word (MPU_RGDAAC n) would typically be performed. Writes to the region descriptor using this alternate access control location do not affect the valid bit, so there are, by definition, no coherency issues involved with the update. The access rights associated with the memory region switch instantaneously to the new value as the IPS write completes.
3. If the region's start and end addresses are to be changed, this would typically be performed by writing a minimum of three words of the region descriptor: MPU_RGD n .Word{0,1,3}, where the writes to Word0 and Word1 redefine the start and end addresses respectively and the write to Word3 re-enables the region descriptor valid bit. In many situations, all four words of the region descriptor would be rewritten.
4. Typically, references to the MPU's programming model would be restricted to supervisor mode accesses from a specific processor(s), so a region descriptor would be specifically allocated for this purpose with attempted accesses from other masters or while in user mode terminated with an error.
5. When the MPU detects an access error, the current AHB bus cycle is terminated with an error response and information on the faulting reference captured in the MPU_EAR n and MPU_EDR n registers. The error-terminated AHB bus cycle typically initiates some type of error response in the originating bus master. For example, a processor core may respond with a bus error exception, while a data movement bus master may respond with an error interrupt. In any event, the processor can retrieve the captured error address and detail information simply by reading the MPU_E{A,D}R n registers. Information on which error registers contain captured fault data is signaled by MPU_CESR[MPERR].
6. Finally, consider the use of overlapping region descriptors. Application of overlapping regions can reduce the number of descriptors required for a given set of access controls. In the overlapping memory space, the protection rights of the corresponding region descriptors are logically summed together (the boolean OR operator). In the following example of a dual-core system, there are four

bus masters: the two processors (CP0, CP1) and two DMA engines (eDMA, a traditional data movement engine transferring data between RAM and peripherals, and FlexRAY, a second engine transferring data to/from the RAM only). Consider the following region descriptor assignments:

Region Description	RGDn	CP0	CP1	eDMA	FlexRay	
CP0 Code	0	rwX	r--	--	--	Flash
CP1 Code	1	r--	rwX	--	--	
CP0 Data & Stack	3	rw-	---	--	--	RAM
CP0 -> CP1 Shared Data		r--	r--	--	--	
CP1 -> CP0 Shared Data		r--	r--	--	--	
CP0 Data & Stack	4	---	rw-	--	--	
Shared DMA Data	5	rw-	rw-	rw	rw	
MPU	6	rw-	rw-	--	--	IPS
Peripherals	7	rw-	rw-	rw	--	

Figure 17-11. Overlapping Region Descriptor Example

In this example, there are eight descriptors used to span nine regions in the three main spaces of the system memory map (flash, RAM, and IPS peripheral space). Each region indicates the specific permissions for each of the four bus masters and this definition provides an appropriate set of shared, private and executable memory spaces.

Of particular interest are the two overlapping spaces: region descriptors 2 and 3, and 3 and 4.

The space defined by RGD2 with no overlap is a private data and stack area that provides read/write access to CP0 only. The overlapping space between RGD2 and RGD3 defines a shared data space for passing data from CP0 to CP1 and the access controls are defined by the logical OR of the two region descriptors. Thus, CP0 has (rw- | r--)= (rw-) permissions, while CP1 has (--- | r--)= (r--) permission in this space. Both DMA engines are excluded from this shared processor data region. The overlapping spaces between RGD3 and RGD4 defines another shared data space, this one for passing data from CP1 to CP0. For this overlapping space, CP0 has (r-- | ---)= (r--) permission, while CP1 has (rw- | r--)= (rw-) permission. The non-overlapped space of RGD4 defines a private data and stack area for CP1 only.

The space defined by RGD5 is a shared data region, accessible by all four bus masters. Finally, the slave peripheral space mapped onto the peripheral bus is partitioned into two regions: one (RGD6) containing the MPU's programming model accessible only to the two processor cores, and the remaining peripheral region (RGD7) accessible to both processors and the traditional eDMA master.

This example is intended to show one possible application of the capabilities of the memory protection unit in a typical system.

Chapter 18

Semaphores

18.1 Introduction

In a dual processor chip, semaphores are used to let each processor know who has control of common memory. Before a core can update or read memory coherently, it has to check the semaphore to see if the other core is not already updating the memory. If the semaphore is clear, it can write common memory, but if it is set, it has to wait for the other core to finish and clear the semaphore.

The semaphores module provides the hardware support needed in multi-core systems for implementing semaphores and provide a simple mechanism to achieve lock/unlock operations via a single write access. This approach eliminates architecture-specific implementations like atomic (indivisible) read-modify-write instructions or reservation mechanisms. The result is an architecture-neutral solution that provides hardware-enforced gates as well as other useful system functions related to the gating mechanisms.

18.1.1 Block Diagram

[Figure 18-1](#) is a simplified block diagram of the semaphores that illustrates the functionality and interdependence of major blocks. In the diagram, the register blocks named gate0, gate1, ..., gate 15 include the finite state machines implementing the semaphore gates plus the interrupt notification logic.

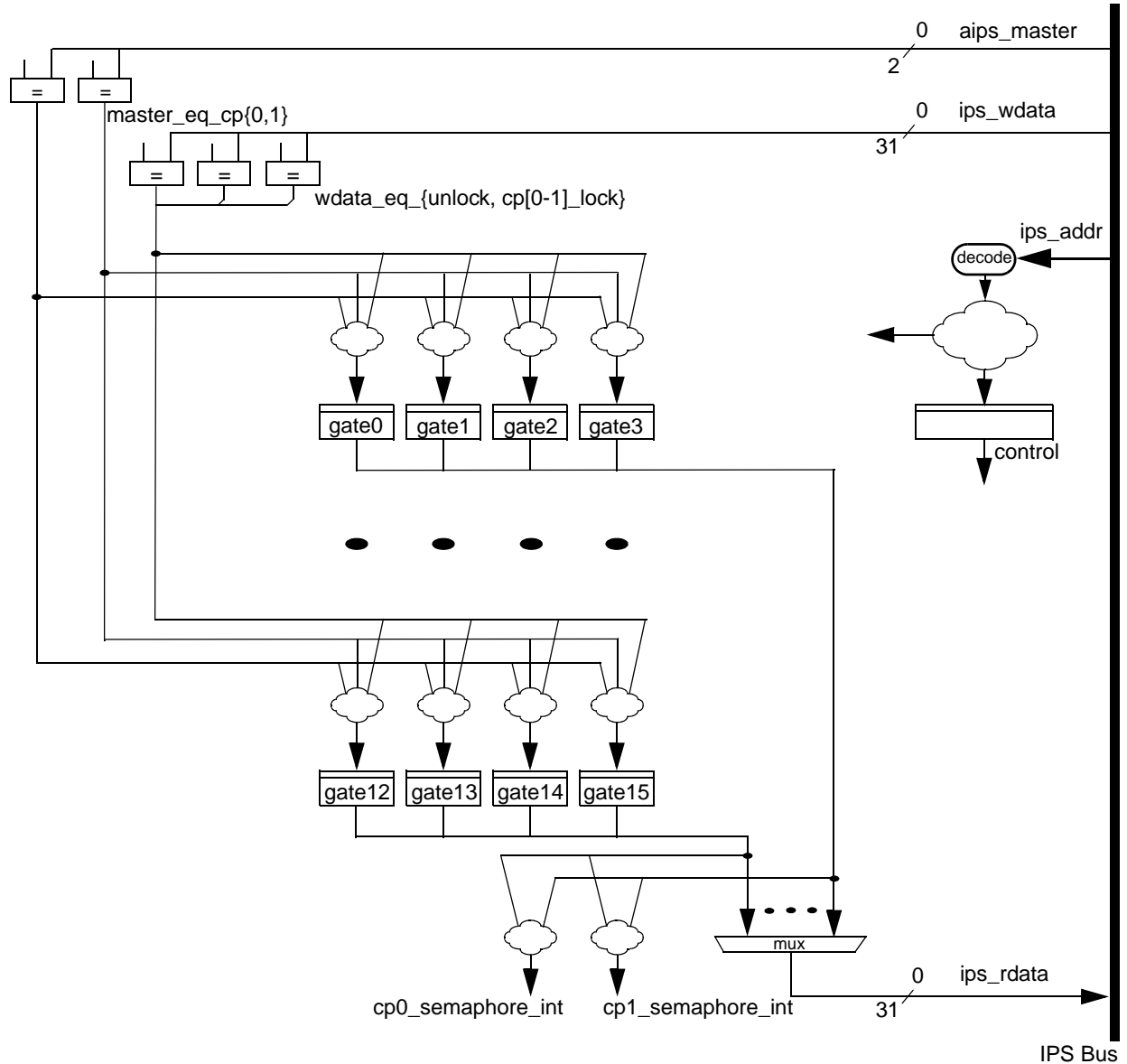


Figure 18-1. semaphores Block Diagram

18.1.2 Features

The semaphores module implements hardware-enforced semaphores as a peripheral device and has these major features:

- Support for 16 hardware-enforced gates in a dual-processor configuration
 - Each hardware gate appears as a three-state, 2-bit state machine, with all 16 gates mapped as an array of bytes
 - Three-state implementation
 - if gate = 0b00, then state = unlocked

if gate = 0b01, then state = locked by e200z1 (master ID = 0)

if gate = 0b10, then state = locked by e200z0 (master ID = 1)

- Uses the bus master ID number as a reference attribute plus the specified data patterns to validate all write operations
- After it is locked, the gate must be unlocked by a write of zeroes from the locking processor
- Optionally enabled interrupt notification after a failed lock write provides a mechanism to indicate the gate is unlocked
- Secure reset mechanisms are supported to clear the contents of individual semaphore gates or notification logic, and clear_all capability

NOTE

Semaphore gates that are locked upon entry into a low-power sleep mode will be cleared by the internal reset generated upon exiting sleep mode. Low-power stop modes have no effect on the state of the semaphore gate.

18.1.3 Modes of Operation

The semaphores module does not support any special modes of operation.

18.2 Signal Description

The semaphores module does not include any external signals.

18.3 Memory Map and Registers

This section provides a detailed description of all semaphores registers.

18.3.1 Module Memory Map

The semaphores programming model map is shown in [Table 18-1](#). The address of each register is given as an offset to the semaphore base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

Table 18-1. Semaphores Memory Map

Offset from SEMA4_BASE (0xFFF1_0000)	Register	Access	Reset Value	Section/ Page
0x0000	SEMA4_Gate00 — Semaphores gate 0	R/W	0x00	18.3.2.1/18-4
0x0001	SEMA4_Gate01 — Semaphores gate 1	R/W	0x00	18.3.2.1/18-4
0x0002	SEMA4_Gate02 — Semaphores gate 2	R/W	0x00	18.3.2.1/18-4
0x0003	SEMA4_Gate03 — Semaphores gate 3	R/W	0x00	18.3.2.1/18-4
0x0004	SEMA4_Gate04 — Semaphores gate 4	R/W	0x00	18.3.2.1/18-4
0x0005	SEMA4_Gate05 — Semaphores gate 5	R/W	0x00	18.3.2.1/18-4

Table 18-1. Semaphores Memory Map (continued)

Offset from SEMA4_BASE (0xFFF1_0000)	Register	Access	Reset Value	Section/ Page
0x0006	SEMA4_Gate06 — Semaphores gate 6	R/W	0x00	18.3.2.1/18-4
0x0007	SEMA4_Gate07 — Semaphores gate 7	R/W	0x00	18.3.2.1/18-4
0x0008	SEMA4_Gate08 — Semaphores gate 8	R/W	0x00	18.3.2.1/18-4
0x0009	SEMA4_Gate09 — Semaphores gate 9	R/W	0x00	18.3.2.1/18-4
0x000A	SEMA4_Gate10 — Semaphores gate 10	R/W	0x00	18.3.2.1/18-4
0x000B	SEMA4_Gate11 — Semaphores gate 11	R/W	0x00	18.3.2.1/18-4
0x000C	SEMA4_Gate12 — Semaphores gate 12	R/W	0x00	18.3.2.1/18-4
0x000D	SEMA4_Gate13 — Semaphores gate 13	R/W	0x00	18.3.2.1/18-4
0x000E	SEMA4_Gate14 — Semaphores gate 14	R/W	0x00	18.3.2.1/18-4
0x000F	SEMA4_Gate15 — Semaphores gate 15	R/W	0x00	18.3.2.1/18-4
0x0010–0x003F	Reserved			
00x040	SEMA4_CP0INE — Semaphores CP0 IRQ notification enable	R/W	0x0000	18.3.2.2/18-5
0x0042–0x0047	Reserved			
0x0048	SEMA4_CP1INE — Semaphores CP1 IRQ notification enable	R/W	0x0000	18.3.2.2/18-5
0x004A–0x07F	Reserved			
0x0080	SEMA4_CP0NTF — Semaphores CP0 IRQ notification	R	0x0000	18.3.2.3/18-6
0x008 2–00x087	Reserved			
0x0088	SEMA4_CP1NTF — Semaphores CP1 IRQ notification	R	0x0000	18.3.2.2/18-5
0x008A–0x00FF	Reserved			
0x0100	SEMA4_RSTGT — Semaphores reset gate	R/W	0x0000	18.3.2.4/18-6
0x0102	Reserved			
0x0104	SEMA4_RSTNTF — Semaphores reset IRQ notification	R/W	0x00000	18.3.2.5/18-8
0x0106–0x0FFF	Reserved			

18.3.2 Register Descriptions

This section lists the semaphores registers in address order and describes the registers and their bit fields.

18.3.2.1 Semaphores Gate *n* Register (SEMA4_GATE_{*n*})

Each semaphore gate is implemented in a 2-bit finite state machine, right-justified in a byte data structure. The hardware uses the bus master number in conjunction with the data patterns to validate all attempted write operations. Only processor bus masters can modify the gate registers. After it is locked, a gate must be opened (unlocked) by the locking processor core.

Multiple gate values can be read in a single access, but only a single gate at a time can be updated via a write operation. 16- and 32-bit writes to multiple gates are allowed, but the write data operand must update the state of a single gate only. A byte write data value of 0x03 is defined as no operation and does not affect the state of the corresponding gate register. Attempts to write multiple gates in a single-aligned access with a size larger than an 8-bit (byte) reference generate an error termination and do not allow any gate state changes.

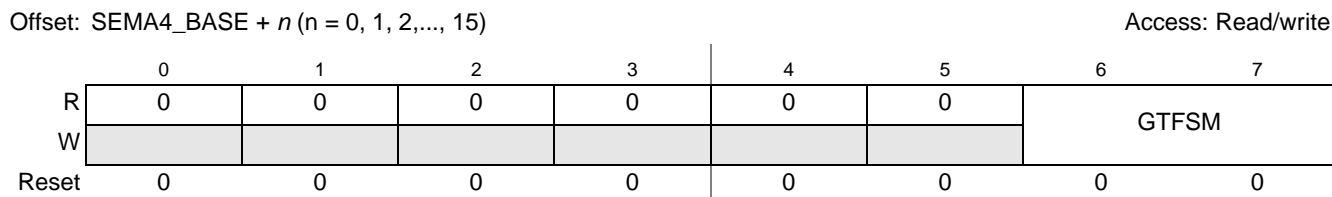


Figure 18-2. SEMA4 Gate n Register (SEMA4_GATE n)

Table 18-2. SEMA4_GATE n Field Descriptions

Field	Description
GTFSM	<p>Gate Finite State Machine. The hardware gate is maintained in a three-state implementation, defined as:</p> <p>00 The gate is unlocked (free)</p> <p>01 The gate has been locked by processor 0</p> <p>10 The gate has been locked by processor 1</p> <p>11 This state encoding is never used and therefore reserved. Attempted writes of 0x03 are treated as no operation and do not affect the gate state machine</p> <p>Note: The state of the gate reflects the last processor that locked it, which can be useful during system debug.</p>

18.3.2.2 Semaphores Processor n IRQ Notification Enable (SEMA4_CP{0,1}INE)

The application of a hardware semaphore module provides an opportunity for implementation of helpful system-level features. An example is an optional mechanism to generate a processor interrupt after a failed lock attempt. Traditional software gate functions execute a spin-wait loop in an effort to obtain and lock the referenced gate. With this module, the processor that fails in the lock attempt could continue with other tasks and allow a properly-enabled notification interrupt to return its execution to the original lock function.

The optional notification interrupt function consists of two registers for each processor: an interrupt notification enable register (SEMA4_CP n INE) and the interrupt request register (SEMA4_CP n NTF). To support implementations with more than 16 gates, these registers can be referenced with aligned 16- or 32-bit accesses. For the SEMA4_CP n INE registers, unimplemented bits read as zeroes and writes are ignored.

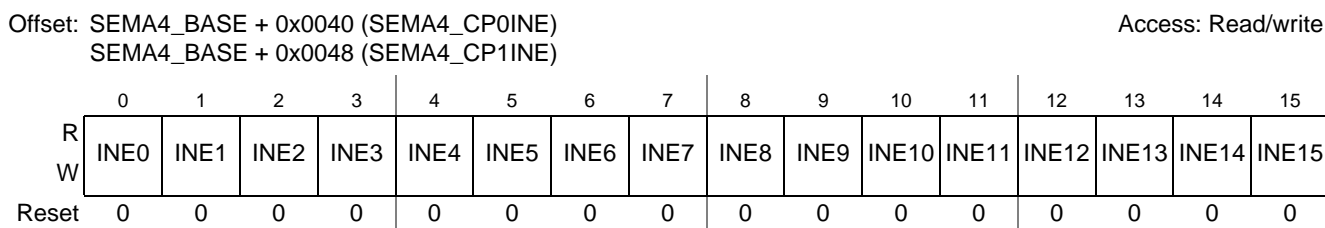


Figure 18-3. Semaphores Processor n IRQ Notification Enable (SEMA4_CP{0,1}INE)

Table 18-3. SEMA4_CP{0,1}NTF Field Descriptions

Field	Description
INEn	Interrupt Request Notification Enable n . This field is a bitmap to enable the generation of an interrupt notification from a failed attempt to lock gate n . 0 The generation of the notification interrupt is disabled. 1 The generation of the notification interrupt is enabled.

18.3.2.3 Semaphores Processor n IRQ Notification (SEMA4_CP{0,1}NTF)

The notification interrupt is generated via a unique finite state machine, one per hardware gate. This machine operates in the following manner:

- When an attempted lock fails, the FSM enters a first state where it waits until the gate is unlocked.
- After it is unlocked, the FSM enters a second state where it generates an interrupt request to the failed lock processor.
- When the failed lock processor succeeds in locking the gate, the IRQ is automatically negated and the FSM returns to the idle state. However, if the other processor locks the gate again, the FSM returns to the first state, negates the interrupt request, and waits for the gate to be unlocked again.

The notification interrupt request is implemented in a 3-bit, five-state machine, where two specific states are encoded and program-visible as SEMA4_CP0NTF[GN n] and SEMA4_CP1NTF[GN n].

Offset: SEMA4_BASE + 0x0080 (SEMA4_CP0NTF)
SEMA4_BASE + 0x0088 (SEMA4_CP1NTF)

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	GN0	GN1	GN2	GN3	GN4	GN5	GN6	GN7	GN8	GN9	GN10	GN11	GN12	GN13	GN14	GN15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-4. Semaphores Processor n IRQ Notification (SEMA4_CP{0,1}NTF)

Table 18-4. SEMA4_CP{0,1}NTF Field Descriptions

Field	Description
GNn	Gate n Notification. This read-only field is a bitmap of the interrupt request notification from a failed attempt to lock gate n . 0 No notification interrupt generated. 1 Notification interrupt generated.

18.3.2.4 Semaphores (Secure) Reset Gate n (SEMA4_RSTGT)

Although the intent of the hardware gate implementation specifies a protocol where the locking processor must unlock the gate, it is recognized that system operation may require a reset function to re-initialize the state of any gate(s) without requiring a system-level reset.

To support this special gate reset requirement, the semaphores module implements a secure reset mechanism which allows a hardware gate (or all the gates) to be initialized by following a specific dual-write access pattern. Using a technique similar to that required for the servicing of a software

watchdog timer, the secure gate reset requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the specified gate(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTGT memory location. The most significant byte (SEMA4_RSTGT[RSTGDP]) must be 0xe2; the least significant byte is a don't_care for this reference.
2. The same processor then performs a second 16-bit write to the SEMA4_RSTGT location. For this write, the upper byte (SEMA4_RSTGT[RSTGDP]) is the logical complement of the first data pattern (0x1d) and the lower byte (SEMA4_RSTGT[RSTGTN]) specifies the gate(s) to be reset. This gate field can specify a single gate be cleared or that all gates are cleared.
3. Reads of the SEMA4_RSTGT location return information on the 2-bit state machine (SEMA4_RSTGT[RSTGSM]) which implements this function, the bus master performing the reset (SEMA4_RSTGT[RSTGMS]) and the gate number(s) last cleared (SEMA4_RSTGT[RSTGTN]). Reads of the SEMA4_RSTGT register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0100 (SEMA4_RSTGT)

Access: Read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	RSTGSM		0	RSTGMS			RSTGTN							
W	RSTGDP							RSTGTN								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-5. Semaphores (Secure) Reset Gate *n* (SEMA4_RSTGT)

Table 18-5. SEMA4_RSTGT Field Descriptions

Field	Description												
RSTGSM	<p>Reset Gate Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as:</p> <p>00 Idle, waiting for the first data pattern write.</p> <p>01 Waiting for the second data pattern write.</p> <p>10 The 2-write sequence has completed. Generate the specified gate reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state.</p> <p>11 This state encoding is never used and therefore reserved.</p> <p>Reads of the SEMA4_RSTGT register return the encoded state machine value. Note the RSTGSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.</p>												
RSTGMS	<p>Reset Gate Bus Master. This 3-bit read-only field records the logical number of the bus master performing the gate reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs.</p> <table border="1" data-bbox="732 688 1045 989"> <thead> <tr> <th>Master</th> <th>Master ID</th> </tr> </thead> <tbody> <tr> <td>e200z1</td> <td>0</td> </tr> <tr> <td>e200z0</td> <td>1</td> </tr> <tr> <td>eDMA</td> <td>2</td> </tr> <tr> <td>FlexRay</td> <td>3</td> </tr> <tr> <td>EBI</td> <td>4</td> </tr> </tbody> </table>	Master	Master ID	e200z1	0	e200z0	1	eDMA	2	FlexRay	3	EBI	4
Master	Master ID												
e200z1	0												
e200z0	1												
eDMA	2												
FlexRay	3												
EBI	4												
RSTGTN	<p>Reset Gate Number. This 8-bit field specifies the specific hardware gate to be reset. This field is updated by the second write.</p> <p>If RSTGTN < 64, then reset the single gate defined by RSTGTN, else reset all the gates. The corresponding secure IRQ notification state machine(s) are also reset.</p>												
RSTGDP	<p>Reset Gate Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the gate reset mechanism. For the first write, RSTGDP = 0xe2 while the second write requires RSTGDP = 0x1d.</p>												

18.3.2.5 Semaphores (Secure) Reset IRQ Notification (SEMA4_RSTNTF)

As with the case of the secure reset function and the hardware gates, it is recognized that system operation may require a reset function to re-initialize the state of the IRQ notification logic without requiring a system-level reset.

To support this special notification reset requirement, the semaphores module implements a secure reset mechanism which allows an IRQ notification (or all the notifications) to be initialized by following a specific dual-write access pattern. When successful, the specified IRQ notification state machine(s) are reset. Using a technique similar to that required for the servicing of a software watchdog timer, the secure reset mechanism requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the IRQ notification(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTNTF memory location. The most significant byte (SEMA4_RSTNTF[RSTNDP]) must be 0x47; the least significant byte is a don't_care for this reference.

2. The same processor performs a second 16-bit write to the SEMA4_RSTNTF location. For this write, the upper byte (SEMA4_RSTNTF[RSTNDP]) is the logical complement of the first data pattern (0xb8) and the lower byte (SEMA4_RSTNTF[RSTNTN]) specifies the notification(s) to be reset. This field can specify a single notification be cleared or that all notifications are cleared.
3. Reads of the SEMA4_RSTNTF location return information on the 2-bit state machine (SEMA4_RSTNTF[RSTNSM]) that implements this function, the bus master performing the reset (SEMA4_RSTNTF[RSTNMS]) and the notification number(s) last cleared (SEMA4_RSTNTF[RSTNTN]). Reads of the SEMA4_RSTNTF register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0104 (SEMA4_RSTNTF)

Access: Read/write

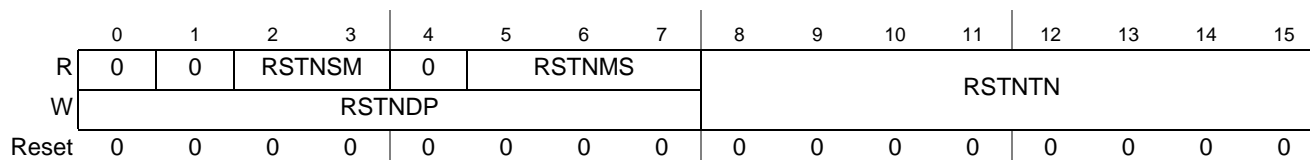


Figure 18-6. Semaphores (Secure) Reset IRQ Notification (SEMA4_RSTNTF)

Table 18-6. SEMA4_RSTGT Field Descriptions

Field	Description												
RSTNSM	<p>Reset Notification Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as:</p> <p>00 Idle, waiting for the first data pattern write.</p> <p>01 Waiting for the second data pattern write.</p> <p>10 The two-write sequence has completed. Generate the specified notification reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state.</p> <p>11 This state encoding is never used and therefore reserved.</p> <p>Reads of the SEMA4_RSTNTF register return the encoded state machine value. Note the RSTNSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.</p>												
RSTNMS	<p>Reset Notification Bus Master. This 3-bit read-only field records the logical number of the bus master performing the notification reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs.</p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>Master</th> <th>Master ID</th> </tr> </thead> <tbody> <tr> <td>e200z1</td> <td>0</td> </tr> <tr> <td>e200z0</td> <td>1</td> </tr> <tr> <td>eDMA</td> <td>2</td> </tr> <tr> <td>FlexRay</td> <td>3</td> </tr> <tr> <td>EBI</td> <td>4</td> </tr> </tbody> </table>	Master	Master ID	e200z1	0	e200z0	1	eDMA	2	FlexRay	3	EBI	4
Master	Master ID												
e200z1	0												
e200z0	1												
eDMA	2												
FlexRay	3												
EBI	4												

Table 18-6. SEMA4_RSTGT Field Descriptions

Field	Description
RSTNTN	Reset Notification Number. This 8-bit field specifies the specific IRQ notification state machine to be reset. This field is updated by the second write. If RSTNTN < 64, then reset the single IRQ notification machine defined by RSTNTN, else reset all the notifications.
RSTNDP	Reset Notification Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the notification reset mechanism. For the first write, RSTNDP = 0x47 while the second write requires RSTNDP = 0xb8.

18.4 Functional Description

Multi-processor systems require a function that can be used to safely and easily provide a locking mechanism that is then used by system software to control access to shared data structures, shared hardware resources, and etc. These gating mechanisms are used by the software to serialize (and synchronize) writes to shared data and/or resources to prevent race conditions and preserve memory coherency between processes and processors.

For example, if processor X enters a section of code where shared data values are to be updated or read coherently, it must first acquire a semaphore. This locks, or closes, a software gate. After the gate has been locked, a properly-architected software system does not allow other processes (or processors) to execute the same code segment or modify the shared data structure protected by the gate, that is, other processes/processors are locked out. Many software implementations include a spin-wait loop within the lock function until the locking of the gate is accomplished. After the lock has been obtained, processor X continues execution and updates the data values protected by the particular lock. After the updates are complete, processor X unlocks (or opens) the software gate, allowing other processes/processors access to the updated data values.

There are three important rules that must be followed for a correctly-implemented system solution:

- All writes to shared data values or shared hardware resources must be protected by a gate variable.
- After a processor locks a gate, accesses to the shared data or resources by other processes/processors must be blocked. This is enforced by software conventions.
- The processor that locks a particular gate is the only processor that can unlock, or open, that gate.

Information in the hardware gate identifying the locking processor can be useful for system-level debugging.

The Hennessy/Patterson text on computer architecture offers this description of software gating:

“One of the major requirements of a shared-memory architecture multiprocessor is being able to coordinate processes that are working on a common task. Typically, a programmer will use *lock variables* to synchronize the processes.

The difficulty for the architect of a multiprocessor is to provide a mechanism to decide which processor gets the lock and to provide the operation that locks a variable. Arbitration is easy for shared-bus multiprocessors, since the bus is the only path to memory. The processor that gets the bus locks out all the other processors from memory. If the CPU and bus provide an atomic swap operation, programmers can create locks with the proper semantics. The adjective *atomic* is key,

for it means that a processor can both read a location **and** set it to the locked value in the same bus operation, preventing any other processor from reading or writing memory.” [Hennessy/Patterson, *Computer Architecture: A Quantitative Approach*, ppg. 471-472]

The classic text continues with a description of the steps required to lock/unlock a variable using an atomic swap instruction.

“Assume that 0 means unlocked and 1 means locked. A processor first reads the lock variable to test its state. A processor keeps reading and testing until the value indicates that the lock is unlocked. The processor then races against all other processes that were similarly “spin waiting” to see who can lock the variable first. All processes use a swap instruction that reads the old value and stores a 1 into the lock variable. The single winner will see the 0, and the losers will see a 1 that was placed there by the winner. (The losers will continue to set the variable to the locked value, but that doesn’t matter.) The winning processor executes the code after the lock and then stores a 0 into the lock when it exits, starting the race all over again. Testing the old value and then setting to a new value is why the atomic swap instruction is called *test and set* in some instruction sets.” [Hennessy/Patterson, *Computer Architecture: A Quantitative Approach*, ppg. 472-473]

The sole drawback to a hardware-based semaphore module is the limited number of semaphores versus the infinite number that can be supported with PowerPC reservation instructions.

18.4.1 Semaphore Usage

Example 1: Inter-processor communication done with software interrupts and semaphores...

- The Z0 uses software interrupts to tell the Z1 that new data is available, or the Z1 does the same to tell the Z0 that there is new data available for transmission.
- Because only eight software interrupts are available, the user may need RAM locations or general-purpose registers in the SIU to refine the meaning of the software interrupt.
- Messages are passed between cores in a defined section of system RAM.
- Before a core updates a message, it must check the associated semaphore to see if the other core is in the process of updating the same message. If the RAM not being updated, then the semaphore must first be locked, then the message can be updated. A software interrupt can be sent to the other core and the semaphore can be unlocked. If the RAM is being updated, the CPU must wait for the other core to unlock the semaphore before proceeding with update.
- Using the same memory location for bidirectional communication might be difficult, so two one-way message areas might work better.
 - For example, if both cores want to update the same location, then the following sequence may occur.
 1. The Z0 locks the semaphore, updates the memory, unlocks the semaphore, and generates a software interrupt to the Z1.
 2. Before the Z1 takes the software interrupt request, it finds the semaphore to be unlocked, so it writes new data to the memory.
 3. The Z1 software interrupt ISR reads the data sent to the Z0, not the data sent from the Z0, and performs an incorrect operation.
 - Semaphores do not prevent this situation from occurring.

Example 2: Coherent read done with semaphores...

- The Z1 wants to coherently read a section of shared memory.
- The Z1 should check that the semaphore for the shared memory is not currently set.
- The Z1 should set the semaphore for the shared memory to prevent the Z0 from updating the shared memory.
- The Z1 will read the required data, then unlock the semaphore.

18.5 Initialization Information

The reset state of the semaphores module allows it to begin operation without the need for any further initialization. All the internal state machines are cleared by any reset event, allowing the module to immediately begin operation.

18.6 Application Information

In an operational multi-core system, most interactions involving the semaphores module involves reads and writes to the SEMA4_GATE n registers for implementation of the hardware-enforced software gate functions. Typical code segments for gate functions perform the following operations:

- To lock (close) a gate
 - The processor performs a byte write of `logical_processor_number + 1` to `gate[i]`
 - The processor reads back `gate[i]` and checks for a value of `logical_processor_number + 1`

If the compare indicates the expected value

then the gate is locked; proceed with the protected code segment

else

lock operation failed;

repeat process beginning with byte write to `gate[i]` in spin-wait loop, or

proceed with another execution path and wait for failed lock interrupt notification

A simple C-language example of a `gateLock` function is shown in [Example 18-1](#). This function follows the Hennessy/Patterson example.

Example 18-1. Sample GateLock Function

```
#define UNLOCK      0
#define CP0_LOCK   1
#define CP2_LOCK   2

void gateLock (n)
int  n;                /* gate number to lock */
{
    int  i;
    int  current_value;
    int  locked_value;

    i = processor_number(); /* obtain logical CPU number */

    if (i == 0)
        locked_value = CP0_LOCK;
```

```

else
    locked_value = CP1_LOCK;

/* read the current value of the gate and wait until the state == UNLOCK */
do {
    current_value = gate[n];
} while (current_value != UNLOCK);

/* the current value of the gate == UNLOCK. attempt to lock the gate for this
processor. spin-wait in this loop until gate ownership is obtained */
do {
    gate[n] = locked_value; /* write gate with processor_number + 1 */
    current_value = gate[n]; /* read gate to verify ownership was obtained */
} while (current_value != locked_value);
}

```

- To unlock (open) a gate
 - After completing the protected code segment, the locking processor performs a byte write of zeroes to gate[i], unlocking (opening) the gate

In this example, a reference to `processor_number ()` is used to retrieve this hardware configuration value. Typically, the logical processor numbers are defined by a hardwired input vector to the individual cores. The exact method for accessing the logical processor number varies by architecture. For PowerPC cores, there is a processor ID register (PIR) which is SPR 286 and contains this value. A single instruction can be used to move the contents of the PIR into a general-purpose register: `mfspr rx,286` where `rx` is the destination GPRn. Other architectures may support a specific instruction to move the contents of the logical processor number into a general-purpose register, e.g., `rdcpn rx` for a read CPU number instruction.

If the optional failed lock IRQ notification mechanisms are used, then accesses to the related registers (SEMA4_CPnINE, SEMA4_CPnNTF) are required. There is no required negation of the failed lock write notification interrupt as the request is automatically negated by the semaphores module once the gate has been successfully locked by the failing processor.

Finally, in the event a system state requires a software-controlled reset of a gate or IRQ notification register(s), accesses to the secure reset control registers (SEMA4_RSTGT, SEMA4_RSTNTF) are required. For these situations, it is recommended that the appropriate IRQ notification enable(s) (SEMA4_CPnINE) bits be disabled before initiating the secure reset 2-write sequence to avoid any race conditions involving spurious notification interrupt requests.

18.7 DMA Requests

There are no DMA requests associated with the IPS_Semaphore block.

18.8 Interrupt Requests

The semaphore interrupt requests are connected to the interrupt controller as described in [Table 8-2](#).

Chapter 19

IEEE 1149.1 Test Access Port Controller (JTAGC)

19.1 Introduction

The JTAGC provides the means to test chip functionality and connectivity and controls access to the debug features of the device, while remaining transparent to system logic when the JTAGC is not in test mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. Instructions can be executed that allow the test access port (TAP) to be shared with other modules on the MCU. All data input to and output from the JTAGC is communicated in serial format.

19.1.1 Block Diagram

A simplified block diagram of the JTAGC illustrates the functionality and interdependence of major blocks (see Figure 19-1). The JTAG port of the device consists of four inputs and one output. These pins include JTAG compliance select (JCOMP), test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS, and TCK are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface.

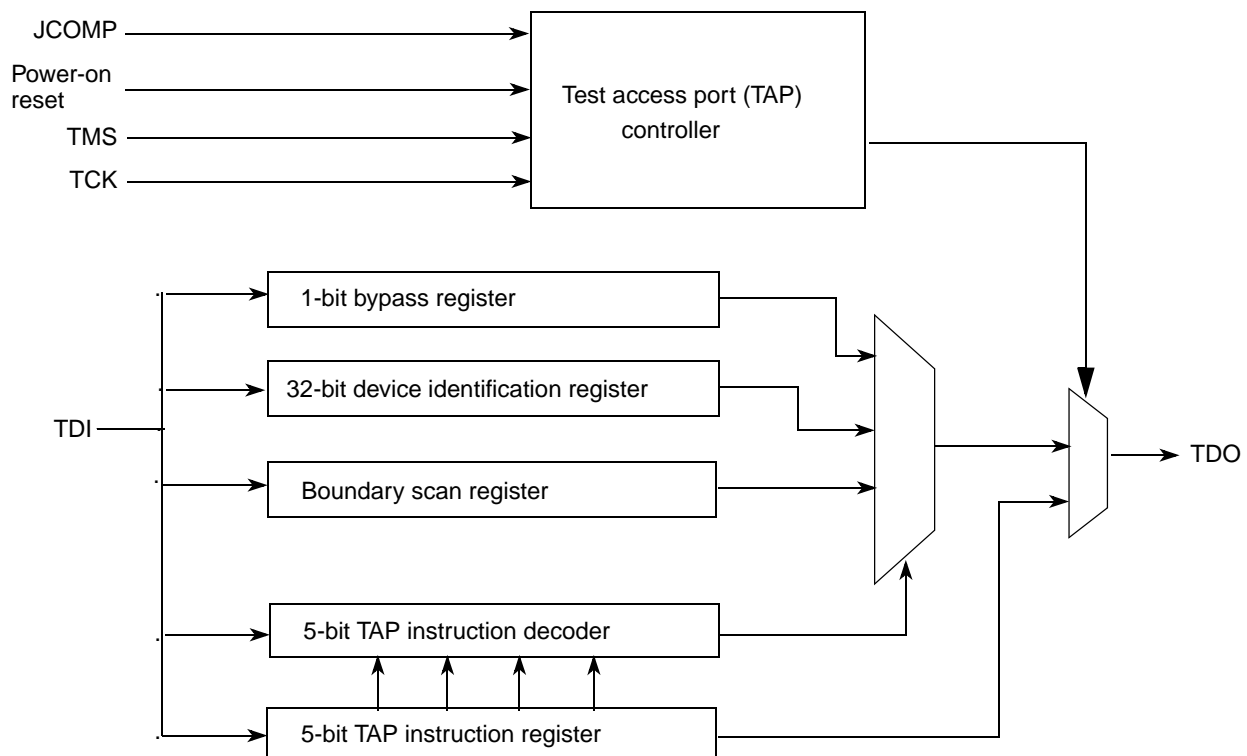


Figure 19-1. JTAGC Block Diagram

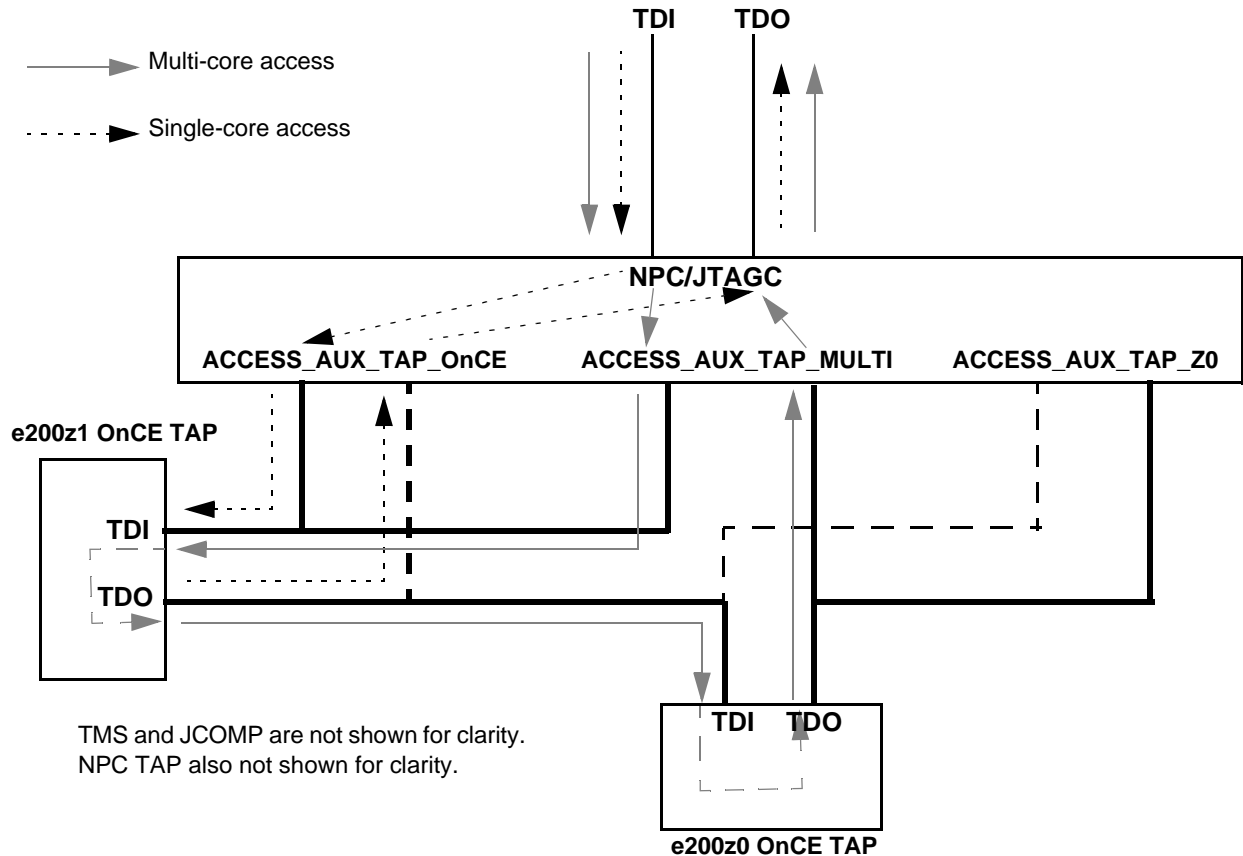


Figure 19-2. JTAG/Nexus Daisy Chain of the MPC5510 e200z1 and e200z0 Cores

19.1.2 Features

The JTAGC is compliant with the IEEE 1149.1-2001 standard and has these major features:

- IEEE 1149.1-2001 test access port (TAP) interface.
- A JCOMP input that provides the ability to share the TAP.
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions, as well as several public and private MCU specific instructions.
- Three test data registers: a bypass register, a boundary scan register, and a device identification register. The size of the boundary scan register is 276 bits.
- A TAP controller state machine that controls the operation of the data registers, instruction register, and associated circuitry.

19.1.3 Modes of Operation

The JTAGC uses JCOMP and a power-on reset indication as its primary reset signals. Several IEEE 1149.1-2001 defined test modes are supported, as well as a bypass mode.

19.1.3.1 Reset

The JTAGC is placed in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered upon the assertion of the power-on reset signal, negation of JCOMP, or through TAP controller state machine transitions controlled by TMS. Asserting power-on reset or negating JCOMP results in asynchronous entry into the reset state. While in reset, the following actions occur:

- The TAP controller is forced into the test-logic-reset state, thereby disabling the test logic and allowing normal operation of the on-chip system logic to continue unhindered.
- The instruction register is loaded with the IDCODE instruction.

In addition, execution of certain instructions can result in assertion of the internal system reset. These instructions include EXTEST, CLAMP, and HIGHZ.

19.1.3.2 IEEE 1149.1-2001 Defined Test Modes

The JTAGC supports several IEEE 1149.1-2001 defined test modes. The test mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, HIGHZ, CLAMP, SAMPLE, and SAMPLE/PRELOAD. Each instruction defines the set of data registers that may operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is enabled for serial access between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are active. The single-bit bypass register shift stage is enabled for serial access between TDI and TDO when the HIGHZ, CLAMP, or reserved instructions are active. The functionality of each test mode is explained in more detail in [Section 19.4.4, “JTAGC Instructions.”](#)

19.1.3.3 Bypass Mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC into bypass mode. While in bypass mode, the single-bit bypass shift register is used to provide a minimum-length serial path to shift data between TDI and TDO.

19.1.3.4 TAP Sharing Mode

There are three selectable auxiliary TAP controllers that share the TAP with the JTAGC. Selectable TAP controllers include the Nexus port controller (NPC), e200z1 OnCE, and e200z0. The instructions required to grant ownership of the TAP to the auxiliary TAP controllers are ACCESS_AUX_TAP_NPC, ACCESS_AUX_TAP_ONCE (for e200z1), and ACCESS_AUX_TAP_Z0. Additionally, the instruction for daisy chaining the e200z1 and e200z0 is ACCESS_AUX_TAP_MULTI (allows instructions to be clocked into both the e200z0 and e200z1 serially). Instruction opcodes for each instruction are shown in [Table 19-2](#).

When the access instruction for an auxiliary TAP is loaded, control of the JTAG pins is transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any

TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

For more information on the TAP controllers see [Chapter 20, “Nexus Development Interface \(NDI\).”](#)

19.2 External Signal Description

Refer to [Table 2-1](#) and [Section 2.7, “Detailed External Signal Descriptions,”](#) for detailed signal descriptions.

19.3 Memory Map and Registers

This section provides a detailed description of the JTAGC registers accessible through the TAP interface, including data registers and the instruction register. Individual bit-level descriptions and reset states of each register are included. These registers are not memory-mapped and can be accessed through the TAP only.

19.3.1 Instruction Register

The JTAGC uses a 5-bit instruction register as shown in [Figure 19-3](#). The instruction register allows instructions to be loaded into the module to select the test to be performed or the test data register to be accessed or both. Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can be changed in the Update-IR and Test-Logic-Reset TAP controller states only. Synchronous entry into the test-logic-reset state results in the IDCODE instruction being loaded on the falling edge of TCK. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the IDCODE instruction. During the capture-IR TAP controller state, the instruction shift register is loaded with the value 0b10101, making this value the register’s read value when the TAP controller is sequenced into the Shift-IR state.

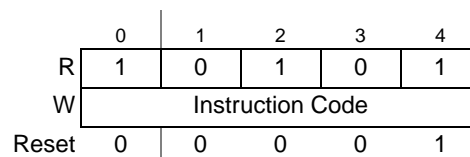


Figure 19-3. 5-Bit Instruction Register

19.3.2 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, CLAMP, HIGHZ, or reserve instructions are active. After entry into the capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

19.3.3 Device Identification Register

The device identification register, shown in Figure 19-4, allows the part revision number, design center, part identification number, and manufacturer identity code to be determined through the TAP. The device identification register is selected for serial data transfer between TDI and TDO when the IDCODE instruction is active. Entry into the capture-DR state while the device identification register is selected loads the IDCODE into the shift register to be shifted out on TDO in the Shift-DR state. No action occurs in the update-DR state.

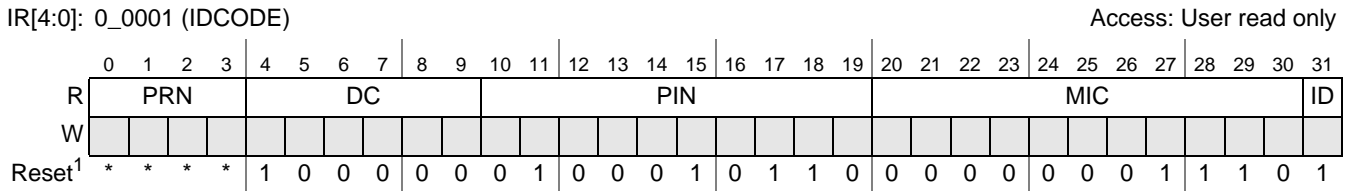


Figure 19-4. Device Identification Register

¹ PRN default value is 0x0 for the device's initial mask set and changes for each mask set revision.

Table 19-1. Device Identification Register Field Descriptions

Field	Description
PRN	Part Revision Number. Contains the revision number of the device. This field changes with each revision of the device or module.
DC	Design Center. Indicates the Freescale design center. For the MPC5510 family this value is 0x20.
PIN	Part Identification Number. Contains the part number of the device. For the MPC5510 family, this value is 0x116.
MIC	Manufacturer Identity Code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0xE.
ID	IDCODE Register ID. Identifies this register as the device identification register and not the bypass register. Always set to 1.

19.3.4 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are active. It is used to capture input pin data, force fixed values on output pins, and select a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in Section 19.4.5, “Boundary Scan.” The size of the boundary scan register is 276 bits.

19.4 Functional Description

19.4.1 JTAGC Reset Configuration

While in reset, the TAP controller is forced into the test-logic-reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. The instruction register is also loaded with the IDCODE instruction.

19.4.2 IEEE 1149.1-2001 (JTAG) Test Access Port

The JTAGC uses the IEEE 1149.1-2001 TAP for accessing registers. This port can be shared with other TAP controllers on the MCU. Ownership of the port is determined by the value of the JCOMP signal and the currently loaded instruction. For more detail on TAP sharing via JTAGC instructions refer to [Section 19.4.4.2, “ACCESS_AUX_TAP_x Instructions.”](#)

Data is shifted between TDI and TDO through the selected register starting with the least significant bit, as illustrated in [Figure 19-5](#). This applies for the instruction register, test data registers, and the bypass register.

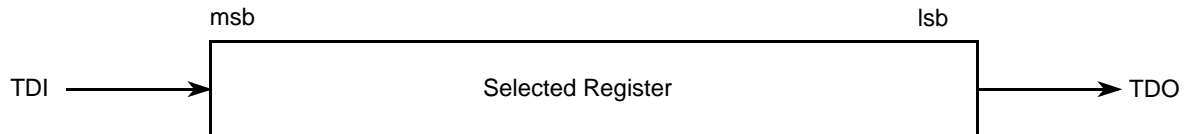
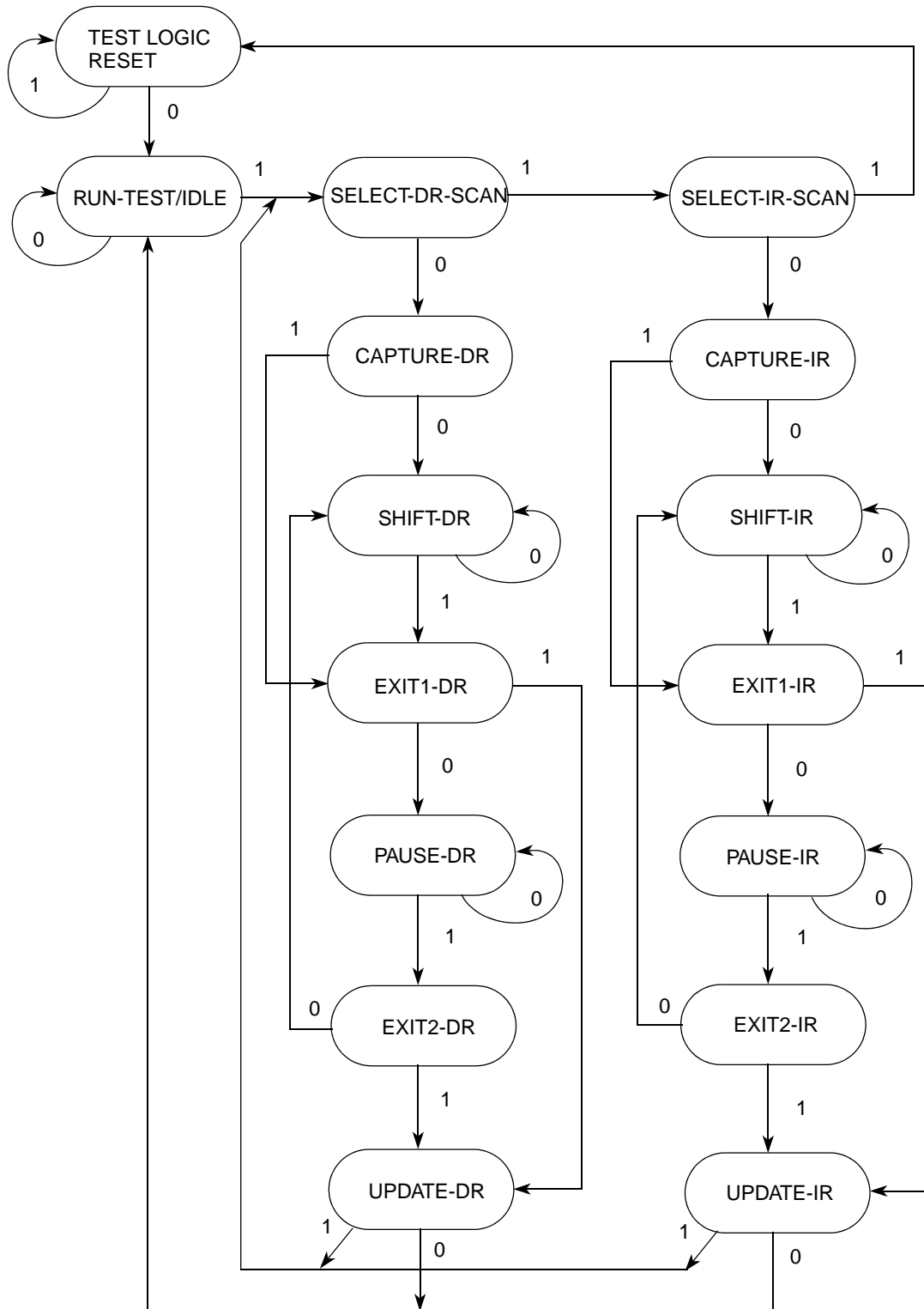


Figure 19-5. Shifting Data Through a Register

19.4.3 TAP Controller State Machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. [Figure 19-6](#) shows the machine's states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCK signal. As [Figure 19-6](#) shows, holding TMS at logic 1 while clocking TCK through a sufficient number of rising edges also causes the state machine to enter the test-logic-reset state.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 19-6. IEEE 1149.1-2001 TAP Controller Finite State Machine

19.4.3.1 Enabling the TAP Controller

The JTAGC TAP controller is enabled by setting JCOMP to a logic 1 value.

19.4.3.2 Selecting an IEEE 1149.1-2001 Register

Access to the JTAGC data registers is achieved by loading the instruction register with any of the JTAGC instructions while the JTAGC is enabled. Instructions are shifted in via the select-IR-scan path and loaded in the update-IR state. At this point, all data register access is performed via the select-DR-scan path.

The select-DR-scan path is used to read or write the register data by shifting in the data (lsb first) during the shift-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the capture-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the update-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting may be terminated after the required number of bits have been acquired.

19.4.4 JTAGC Instructions

The JTAGC implements the IEEE 1149.1-2001 defined instructions listed in [Table 19-2](#). This section gives an overview of each instruction, refer to the IEEE 1149.1-2001 standard for more details

Table 19-2. JTAG Instructions

Instruction	Code[4:0]	Instruction Summary
IDCODE	00001	Selects device identification register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
HIGHZ	01001	Selects bypass register while three-stating all output pins and asserting functional reset
CLAMP	01100	Selects bypass register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_NPC	10000	Grants the Nexus port controller (NPC) ownership of the TAP
ACCESS_AUX_TAP_ONCE	10001	Grants the Nexus e200z1 core interface ownership of the TAP
ACCESS_AUX_TAP_Z0	11001	Grants the Nexus e200z0 core interface ownership of the TAP
ACCESS_AUX_TAP_MULTI	11100	Daisy chaining the e200z1 and e200z0 cores—allows instructions to be clocked into both the e200z0 and e200z1 serially.
BYPASS	11111	Selects bypass register for data operations

Table 19-2. JTAG Instructions (continued)

Instruction	Code[4:0]	Instruction Summary
Factory Debug Reserved ¹	00101, 00110, 01010	Intended for factory debug only
Reserved ²	All Other Codes	Decoded to select bypass register

¹ Intended for factory debug, and not customer use

² Freescale reserves the right to change the decoding of reserved instruction codes in the future

19.4.4.1 BYPASS Instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test functions. While the BYPASS instruction is active, the system logic operates normally.

19.4.4.2 ACCESS_AUX_TAP_x Instructions

The ACCESS_AUX_TAP_x instructions allow the Nexus modules on the MCU to take control of the TAP. When this instruction is loaded, control of the TAP pins is transferred to the selected auxiliary TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

See [Section 19.5, “e200z0 and e200z1 OnCE Controllers,”](#) for a block diagram and e200z0 OnCE controller register descriptions.

19.4.4.3 CLAMP Instruction

CLAMP allows the state of signals driven from MCU pins to be determined from the boundary scan register while the bypass register is selected as the serial path between TDI and TDO. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. CLAMP also asserts the internal system reset for the MCU to force a predictable internal state.

19.4.4.4 EXTEST — External Test Instruction

EXTEST selects the boundary scan register as the shift path between TDI and TDO. It allows testing of off-chip circuitry and board-level interconnections by driving preloaded data contained in the boundary scan register onto the system output pins. Typically, the preloaded data is loaded into the boundary scan register using the SAMPLE/PRELOAD instruction before the selection of EXTEST. EXTEST asserts the internal system reset for the MCU to force a predictable internal state while performing external boundary scan operations.

19.4.4.5 HIGHZ Instruction

HIGHZ selects the bypass register as the shift path between TDI and TDO. While HIGHZ is active, all output drivers are placed in an inactive drive state (for example, high impedance). HIGHZ also asserts the internal system reset for the MCU to force a predictable internal state.

19.4.4.6 IDCODE Instruction

IDCODE selects the 32-bit device identification register as the shift path between TDI and TDO. This instruction allows interrogation of the MCU to determine its version number and other part identification data. IDCODE is the instruction placed into the instruction register when the JTAGC is reset.

19.4.4.7 SAMPLE Instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and immediately before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when the SAMPLE instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. There is no defined action in the update-DR state. Both the data capture and the shift operation are transparent to system operation.

19.4.4.8 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- First, the SAMPLE portion of the instruction obtains a sample of the system data and control signals present at the MCU input pins and immediately before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the shift-DR state. The data capture and the shift operation are transparent to system operation.
- Secondly, the PRELOAD portion of the instruction initializes the boundary scan register cells before selecting the EXTEST or CLAMP instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the update-DR state. The data is applied to the external output pins by the EXTEST or CLAMP instruction. System operation is not affected.

19.4.5 Boundary Scan

The boundary scan technique allows signals at component boundaries to be controlled and observed through the shift-register stage associated with each pad. Each stage is part of a larger boundary scan register cell, and cells for each pad are interconnected serially to form a shift-register chain around the border of the design. The boundary scan register consists of this shift-register chain, and is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are loaded. The shift-register chain contains a serial input and serial output, as well as clock and control signals.

19.5 e200z0 and e200z1 OnCE Controllers

The e200z0 core OnCE controller supports a complete set of Nexus 1 debug features, as well as providing access to the Nexus2+ configuration registers. A complete discussion of the e200z0 OnCE debug features is available in the *e200z0 Reference Manual*.

The following sections will describe functionality of the e200z0 OnCE controller; however, the e200z1 OnCE controller operates in the same manner as the e200z0 OnCE controller, and is fully documented in the *e200z1 Reference Manual*.

NOTE

The register select field in the e200z1 OnCE command register (OCMD[RS]) does not implement the shared nexus control register (SNC).

19.5.1 e200z0 OnCE Controller Block Diagram

Figure 19-7 is a block diagram of the e200z0 OnCE block.

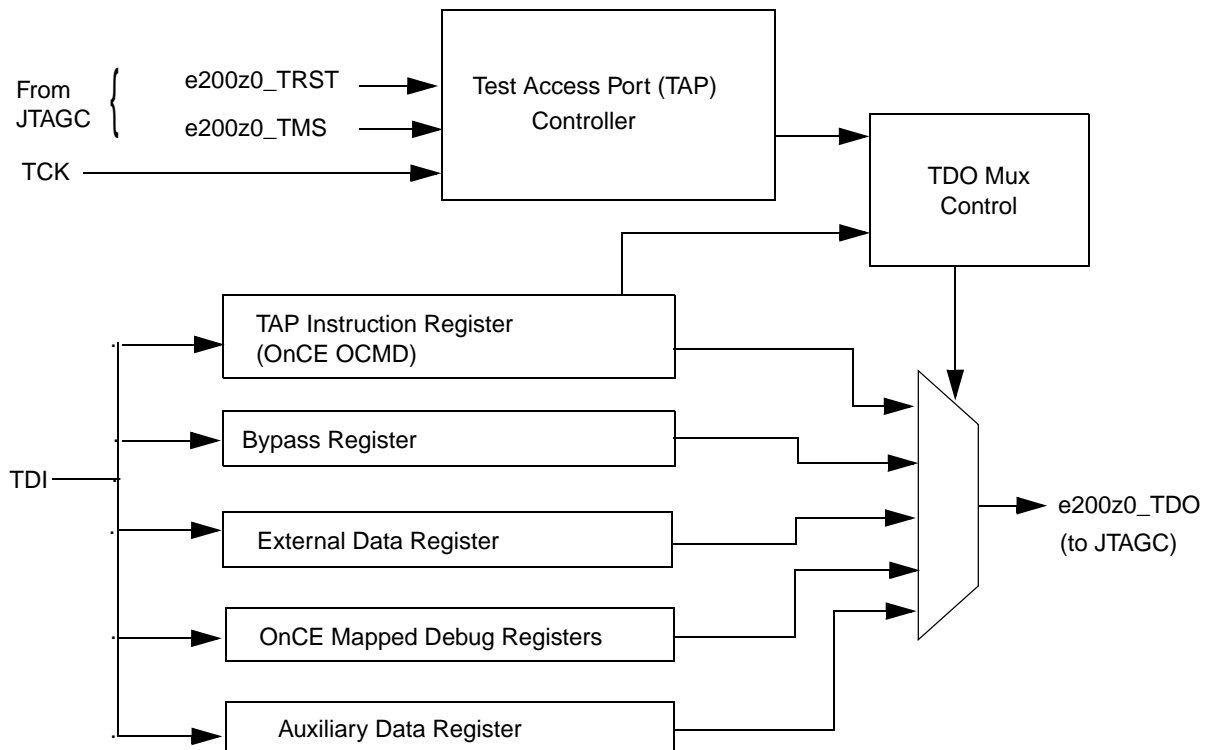


Figure 19-7. e200z0 OnCE Block Diagram

19.5.2 e200z0 OnCE Controller Functional Description

The functional description for the e200z0 OnCE controller is the same as for the JTAGC, with the differences described below.

19.5.2.1 Enabling the TAP Controller

To access the e200z0 OnCE controller, the proper JTAGC instruction needs to be loaded in the JTAGC instruction register, as discussed in [Section 19.1.3.4, “TAP Sharing Mode”](#). The e200z0 OnCE TAP controller may either be accessed independently or chained with the e200z1 OnCE TAP controller, such that the TDO output of the e200z1 TAP controller is fed into the TDI input of the e200z0 TAP controller. The chained configuration allows commands to be loaded into both core’s OnCE registers in one shift operation, so that both cores can be sent a GO command at the same time for example.

19.5.3 e200z0 OnCE Controller Register Descriptions

Most e200z0 OnCE debug registers are fully documented in the *e200z0 Reference Manual*. The MPC5510 implements a new shared nexus control register (SNC) which is defined in [Section 19.5.3.2, “OnCE Shared Nexus Control Register \(SNC\)”](#).

The SNC register is used to configure which core is being traced by the Nexus2+ block, and how the outputs of the Nexus2+ block affect each core.

The SNC register requires a new encoding in the OnCE command register’s register select field (OCMD[RS]), as defined in [Section 19.5.3.1, “OnCE Command Register \(OCMD\)”](#).

19.5.3.1 OnCE Command Register (OCMD)

The OnCE command register (OCMD) is a 10-bit shift register that receives its serial data from the TDI pin and serves as the instruction register (IR). It holds the 10-bit commands to be used as input for the e200z0 OnCE Decoder. The OCMD is shown in [Figure 19-8](#). The OCMD is updated when the TAP controller enters the update-IR state. It contains fields for controlling access to a resource, as well as controlling single-step operation and exit from OnCE mode.

Although the OCMD is updated during the update-IR TAP controller state, the corresponding resource is accessed in the DR scan sequence of the TAP controller, and as such, the update-DR state must be transitioned through in order for an access to occur. In addition, the update-DR state must also be transitioned through in order for the single-step and/or exit functionality to be performed, even though the command appears to have no data resource requirement associated with it.

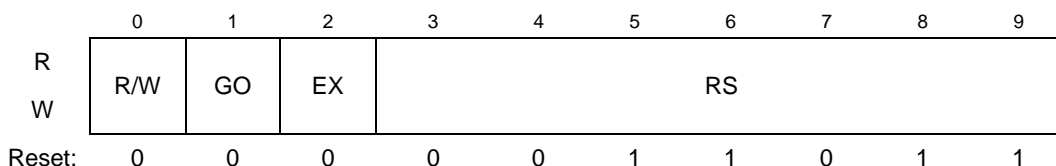


Figure 19-8. OnCE Command Register (OCMD)

Table 19-3. e200z0 OnCE Register Addressing

RS	Register Selected
000 0000 – 000 0001	Reserved
000 0010	JTAG ID (read-only)

Table 19-3. e200z0 OnCE Register Addressing (continued)

RS	Register Selected
000 0011 – 000 1111	Reserved
001 0000	CPU Scan Register (CPUSCR)
001 0001	No Register Selected (Bypass)
001 0010	OnCE Control Register (OCR)
001 0011 – 001 1111	Reserved
010 0000	Instruction Address Compare 1 (IAC1)
010 0001	Instruction Address Compare 2 (IAC2)
010 0010	Instruction Address Compare 3 (IAC3)
010 0011	Instruction Address Compare 4 (IAC4)
010 0100	Data Address Compare 1 (DAC1)
010 0101	Data Address Compare 2 (DAC2)
010 0110 – 010 1111	Reserved
011 0000	Debug Status Register (DBSR)
011 0001	Debug Control Register 0 (DBCR0)
011 0010	Debug Control Register 1 (DBCR1)
011 0011	Debug Control Register 2 (DBCR2)
011 0100 – 101 1111	Reserved (do not access)
110 1111	Shared Nexus Control Register (SNC) (only available on the e200z0 core)
111 0000 – 111 1001	General Purpose Register Selects [0:9]
111 1010 – 111 1011	Reserved
111 1100	Nexus2+ Access
111 1101	LSRL Select (factory test use only)
111 1110	Enable_OnCE
111 1111	Bypass

19.5.3.2 OnCE Shared Nexus Control Register (SNC)

This register determines which core has ownership of the Nexus2+ tracing functionality, which core's master ID is used during Nexus2+ DMA access, and if the Nexus $\overline{\text{EVTI}}$ debug request is used as a debug request to either or both cores. This register is only available on the e200z0 core.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	DBGRE		0	0	ND		0	0	NT	
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure 19-9. OnCE Shared Nexus Control Register (SNC)

Table 19-4. SNC Bit Description

Field	Description
bits 0–21	Reserved.
DBGRE	<p>\overline{EVTI} Debug Request Enable.</p> <p>00 Disabled</p> <p>x1 \overline{EVTI} is used for debug request for e200z0</p> <p>1x \overline{EVTI} is used for debug request for e200z1</p> <p>Note: The user will still need to program the Nexus2+ DC1[4:3] register in each core to enable the debug request.</p>
bits 24–25	Reserved.
ND	<p>Nexus2+ DMA Control. Determines which core’s master ID is used during Nexus2+ DMA access.</p> <p>00 DMA read/writes appear to be generated by e200z0</p> <p>01 DMA read/writes appear to be generated by e200z1</p> <p>1x Reserved (default to e200z0)</p> <p>Note: The selected master ID for Nexus2+ DMA access is set independently of which core is currently being traced.</p> <p>Note: ND does not control which core actually performs the Nexus read/write (DMA) access, it only controls the master ID that the access uses for the transfer. All Nexus read/writes are performed by the e200z0 core.</p>
bits 28–29	Reserved
NT	<p>Nexus2+ Trace Mode. Determines which core has ownership of the Nexus2+ Tracing functionality.</p> <p>00 e200z0 has ownership of Nexus2+</p> <p>01 e200z1 has ownership of Nexus2+</p> <p>1x Reserved (Default to e200z0)</p>

19.6 Initialization/Application Information

The test logic is a static logic design, and TCK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

To initialize the JTAGC module and enable access to registers, the following sequence is required:

1. Set the JCOMP signal to logic 1, thereby enabling the JTAGC TAP controller.
2. Load the appropriate instruction for the test or action to be performed.

Chapter 20

Nexus Development Interface (NDI)

20.1 Introduction

NOTE

The Power PC standard is to number the register bits according to the MSB=0 convention. However, the Nexus standard is to number the register bits according to the LSB=0 convention.

Register bits in this chapter are numbered according to the Nexus standard (LSB=0 convention).

The Nexus Development Interface (NDI) block provides real-time development support capabilities for the MPC5510 MCU in compliance with the IEEE-ISTO 5001-2003 standard. This development support is supplied for MCUs without requiring external address and data pins for internal visibility.

The NDI block is an integration of several individual Nexus blocks that are selected to provide the development support interface for MPC5510.

The NDI block interfaces to the e200z1, e200z0, and internal buses to provide development support as per the IEEE-ISTO 5001-2003 standard. The development support provided includes program trace, watchpoint messaging, ownership trace, watchpoint triggering, processor overrun control, run-time access to the MCU's internal memory map, and access to the e200z1 and e200z0 internal registers during halt, via the JTAG port.

NOTE

Because the Nexus pins are multiplexed with the EBI pins on MPC5510, several constraints apply when using the NDI along with the EBI.

To use the NDI with the EBI...

1. Run the EBI in 16-bit data port mode with data multiplexed on the LSB side.
2. Set EVT_EN low in the Nexus control register, to free up EVTI on PF0 and EVTO on PF1.
3. Configure the EBI pins. (After clearing EVT_EN, you can reclaim PF0/PF1).
4. Disconnect PF0 from the Nexus probe (probe drives EVTI high), and ensure that PFO is connected to memory R/W.

20.2 Block Diagram

Figure 20-1 shows a functional block diagram of the NDI.

A simplified block diagram of the NDI illustrates the functionality and interdependence of major blocks (see Figure 20-2) and how the individual Nexus blocks are combined to form the NDI.

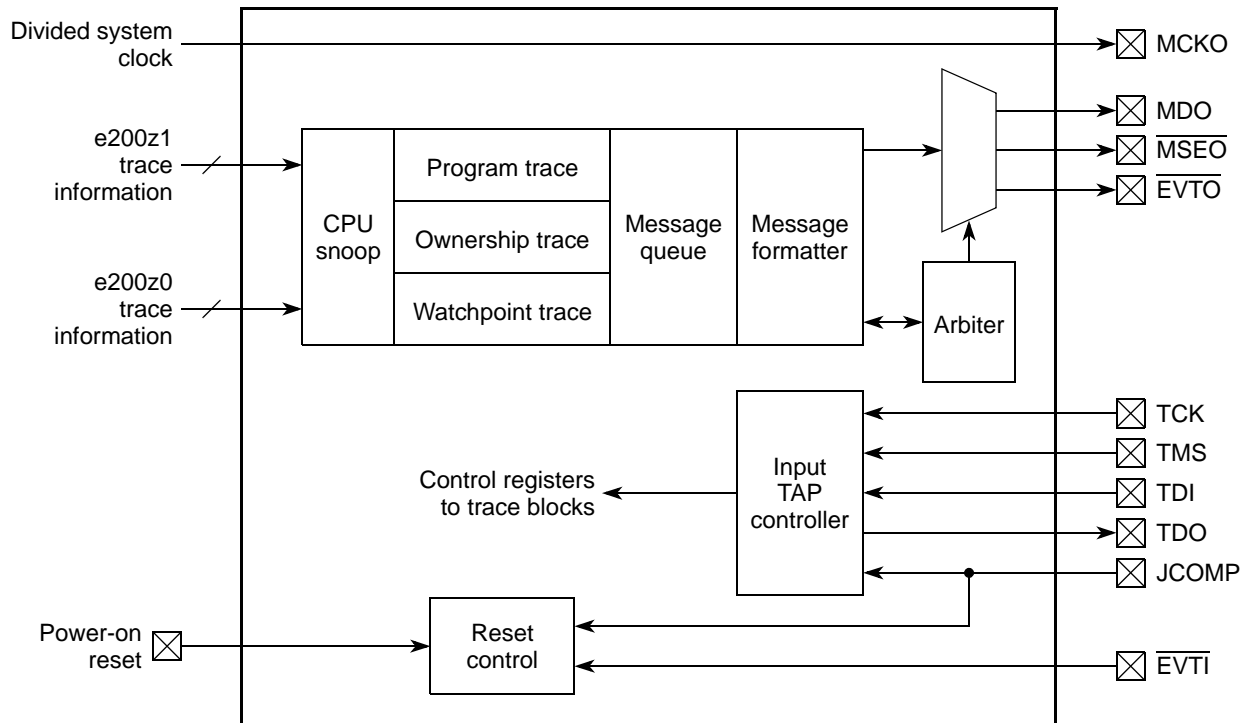


Figure 20-1. NDI Functional Block Diagram

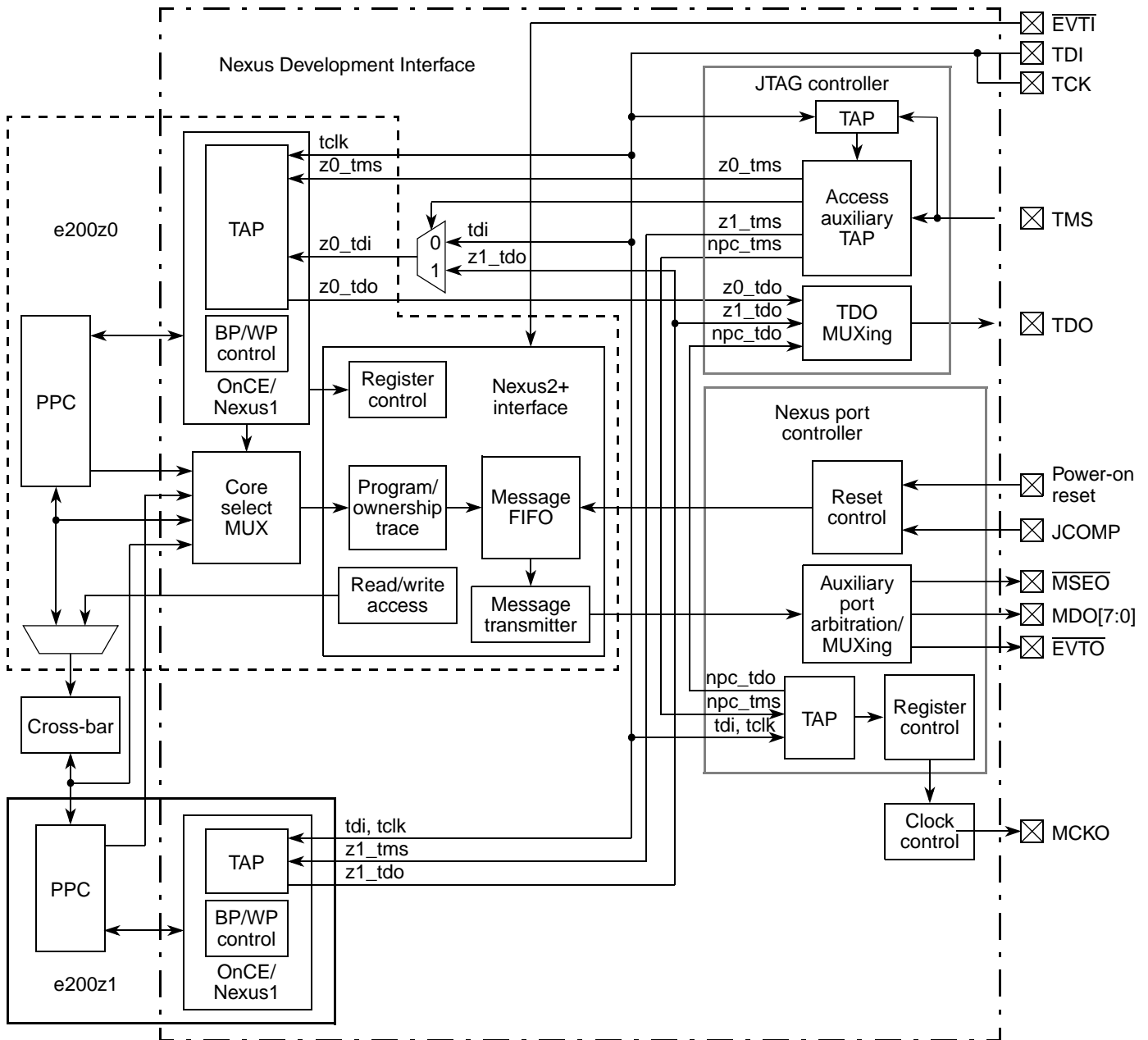


Figure 20-2. NDI Implementation Block Diagram

20.2.1 Features

The NDI module of the MPC5510 is compliant with Class 2 of the IEEE-ISTO 5001-2003 standard, with additional Class 3 and Class 4 features available. The following features are implemented:

- Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code may be traced.

- Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.
- Watchpoint messaging via the auxiliary pins.
- Watchpoint trigger enable of program trace messaging.
- Auxiliary interface for higher data input/output.
 - Configurable (min./max) message data out pins (four MDO in reduced-port mode or eight MDO in full-port mode).
 - One message start/end out pins ($\overline{\text{MSEO}}$).
 - One watchpoint event pin ($\overline{\text{EVTO}}$).
 - One event-in pin ($\overline{\text{EVTI}}$).
 - One message clock out pin (MCKO).
 - Five pin JTAG port (JCOMP, TDI, TDO, TMS, and TCK).
- Registers for program trace, ownership trace, and watchpoint trigger.
- All features controllable and configurable via the JTAG port.
- Run-time access to the on-chip memory map via the Nexus read/write access protocol. This allows for enhanced download/upload capabilities.
- All features are independently configurable and controllable via the IEEE 1149.1 I/O port.
- The NDI block reset is controlled with JCOMP, power-on reset, and the TAP state machine. All these sources are independent of system reset.
- Power-on-reset status indication via MDO[0].
- Support for internal censorship mode to prevent external access to flash memory contents when censorship is enabled.

NOTE

If the e200z1 and e200z0 cores have executed a wait instruction, then the Nexus2+ controller clocks are gated off. While both cores are in this state, it will not be possible to perform Nexus read/write operations.

20.2.2 Modes of Operation

The NDI block is in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered on the assertion of the power-on reset signal, negation of JCOMP, or through state machine transitions controlled by TMS. Assertion of JCOMP allows the NDI to move out of the reset state, and is a prerequisite to grant Nexus clients control of the TAP. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAGC controller (JTAGC) block when JCOMP is asserted.

The NPC transitions out of the reset state immediately following negation of power-on reset.

20.2.2.1 Nexus Reset

In Nexus reset mode, the following actions occur:

- Register values default back to their reset values.
- The message queues are marked as empty.
- The auxiliary output port pins are negated if the NDI controls the pads.
- The TDO output buffer is disabled if the NDI has control of the TAP.
- The TDI, TMS, and TCK inputs are ignored.
- The NDI block indicates to the MCU that it is not using the auxiliary output port. This indication can be used to three-state the output pins or use them for another function.

20.2.2.2 Full-Port Mode

In full-port mode, all available MDO pins are used to transmit messages. All trace features are enabled or can be enabled by writing the configuration registers via the JTAG port. Eight MDO pins are available in full-port mode.

20.2.2.3 Reduced-Port Mode

In reduced-port mode, a subset of the available MDO pins are used to transmit messages. All trace features are enabled or can be enabled by writing the configuration registers via the JTAG port. Four MDO pins are available. Unused MDO pins can be used as GPIO. Details on GPIO functionality configuration can be found in [Chapter 6, “System Integration Unit \(SIU\).”](#) Four MDO pins are available in reduced-port mode.

20.2.2.4 Disabled-Port Mode

In disabled-port mode, message transmission is disabled. Any debug feature that generates messages can not be used. The primary features available are class 1 features and read/write access.

20.2.2.5 Censored Mode

The NDI supports internal flash censorship mode by preventing the transmission of trace messages and Nexus access to memory-mapped resources when censorship is enabled.

20.2.2.6 Stop Mode

Stop mode logic is implemented in the Nexus port controller (NPC). When a request is made to enter stop mode, the NDI block completes monitoring of any pending bus transaction, transmits all messages already queued, and acknowledges the stop request. After the acknowledgment, the system clock input are shut off by the clock driver on the device. While the clocks are shut off, the development tool cannot access NDI registers via the JTAG port.

20.3 External Signal Description

With the exception of the JCOMP signal, all signals are shared by all the individual blocks that make up the NDI block. The Nexus port controller (NPC) block controls the signal sharing. The JCOMP signal is input to the NPC block and used to generate the Nexus reset control signal.

Refer to [Table 2-1](#) and [Section 2.7, “Detailed External Signal Descriptions,”](#) for detailed signal descriptions.

20.3.1 Nexus Signal Reset States

Table 20-1. NDI Signal Reset State

Name	Function	Nexus Reset State	Pull
$\overline{\text{EVTI}}$	Event-in pin	—	Up
$\overline{\text{EVT0}}$	Event-out pin	0b1	—
MCKO	Message clock out pin	0b0	—
MDO[3:0] or MDO[7:0]	Message data out pins	0 ¹	—
$\overline{\text{MSE0}}$	Message start/end out pins	0b11	—

¹ MDO[0] reflects the state of the internal power on reset signal until $\overline{\text{RESET}}$ is negated.

20.4 Memory Map and Registers

The NDI block contains no memory-mapped registers. Nexus registers are accessed by a development tool via the JTAG port using a client-select value and a register index. OnCE registers are accessed by loading the appropriate value in the RS field of the OnCE command register (OCMD) via the JTAG port.

20.4.1 Nexus Debug Interface Registers

[Table 20-2](#) shows the NDI registers by client select and index values. OnCE register addressing is documented in [Chapter 19, “IEEE 1149.1 Test Access Port Controller \(JTAGC\).”](#)

Table 20-2. Nexus Debug Interface Registers

Client Select	Index	Register
Client-Independent Registers		
0bxxxx	0	Device ID (DID) ¹
0bxxxx	1	Client select control (CSC) ¹
0bxxxx	127	Port configuration register (PCR) ¹
e200z0 Control/Status Registers		
0b0000	2	e200z0 development control1 (DC1)

Table 20-2. Nexus Debug Interface Registers (continued)

Client Select	Index	Register
0b0000	3	e200z0 development control2 (DC2)
0b0000	4	e200z0 development status (DS)
0b0000	7	Read/write access control/status (RWCS)
0b0000	9	Read/write access address (RWA)
0b0000	10	Read/write access data (RWD)
0b0000	11	e200z0 watchpoint trigger (PPC_WT)

¹ Implemented in NPC block. All other registers implemented in e200z0 Nexus2+ block. This register is not used on the MPC5510.

20.4.2 Register Descriptions

This section lists the NDI registers and describes the registers and their bit fields.

20.4.2.1 Nexus Device ID Register (DID)

The NPC device identification register, shown in Figure 20-3, allows the part revision number, design center, part identification number, and manufacturer identity code of the device to be determined through the auxiliary output port, and serially through TDO. This register is read-only.

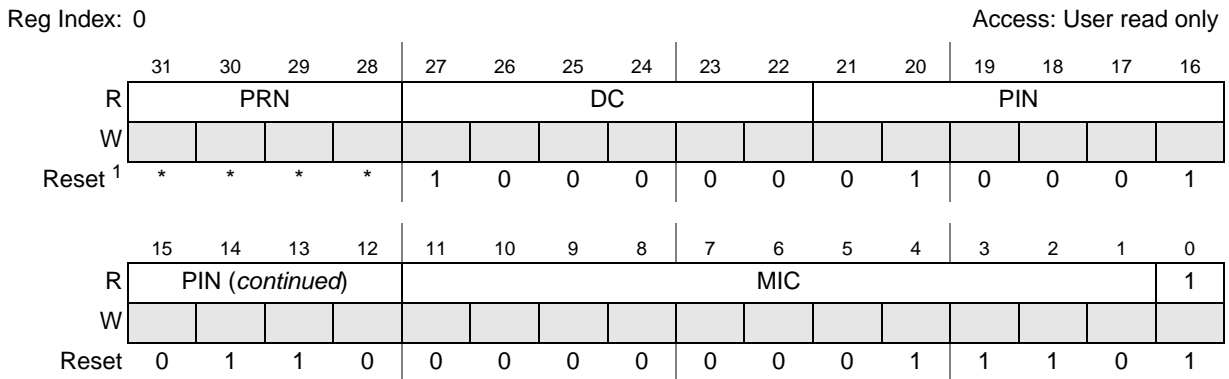


Figure 20-3. Nexus Device ID Register (DID)

¹ Part Revision Number default value is 0x0 for the device's initial mask set and changes for each mask set revision.

Table 20-3. DID Field Descriptions

Field	Description
PRN	Part Revision Number. Contains the revision number of the part. This field changes with each revision of the device or module.
DC	Design Center. Indicates the Freescale design center. This value is 0x20.

Table 20-3. DID Field Descriptions (continued)

Field	Description
PIN	Part Identification Number. Contains the part number of the device. The PIN value for the MPC5510 family is 0x0116.
MIC	Manufacturer Identity Code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0x00E.
bit 0	Fixed Per JTAG 1149.1. Always set to 1.

20.4.2.2 Port Configuration Register (PCR)

The PCR is used to select the NPC mode of operation, enable MCKO and select the MCKO frequency, and enable or disable MCKO gating. This register should be configured as soon as the NDI is enabled.

The PCR register may be rewritten by the debug tool subsequent to the enabling of the NPC for low power debug support. In this case, the debug tool may set and clear the LP_DBG_EN, SLEEP_SYNC, and STOP_SYNC bits, but must preserve the original state of the remaining bits in the register.

NOTE

The mode or clock division must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.

Reg Index: 127

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R				MCKO_DIV			EVT_EN	0	0	0	0	0	0	0	0	0
W	FPM	MCKO_GT	MCKO_EN	MCKO_DIV			EVT_EN									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R							SLEEP_SYNC	STOP_SYNC	0	0	0	0	0	0	0	
W	LP_DBG_EN						SLEEP_SYNC	STOP_SYNC								PSTAT_EN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-4. Port Configuration Register (PCR)

Table 20-4. PCR Field Descriptions

Field	Description																		
FPM	Full Port Mode. The value of the FPM bit determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 0 A subset of MDO pins are used to transmit messages. 1 All MDO pins are used to transmit messages.																		
MCKO_GT	MCKO Clock Gating Control. This bit is used to enable or disable MCKO clock gating. If clock gating is enabled, the MCKO clock is gated when the NPC is in enabled mode but not actively transmitting messages on the auxiliary output port. When clock gating is disabled, MCKO is allowed to run even if no auxiliary output port messages are being transmitted. 0 MCKO gating is disabled. 1 MCKO gating is enabled.																		
MCKO_EN	MCKO Enable. This bit enables the MCKO clock to run. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. 0 MCKO clock is driven to zero. 1 MCKO clock is enabled.																		
MCKO_DIV	MCKO Division Factor. The value of this signal determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. In this table, SYS_CLK represents the system clock frequency. <table border="1" data-bbox="597 873 1235 1314"> <thead> <tr> <th>MCKO_DIV</th> <th>MCKO Frequency</th> </tr> </thead> <tbody> <tr> <td>0b000</td> <td>SYSCLK÷1</td> </tr> <tr> <td>0b001</td> <td>SYSCLK÷2</td> </tr> <tr> <td>0b010</td> <td>Reserved</td> </tr> <tr> <td>0b011</td> <td>SYS_CLK÷4</td> </tr> <tr> <td>0b100</td> <td>Reserved</td> </tr> <tr> <td>0b101</td> <td>Reserved</td> </tr> <tr> <td>0b110</td> <td>Reserved</td> </tr> <tr> <td>0b111</td> <td>SYS_CLK÷8</td> </tr> </tbody> </table>	MCKO_DIV	MCKO Frequency	0b000	SYSCLK÷1	0b001	SYSCLK÷2	0b010	Reserved	0b011	SYS_CLK÷4	0b100	Reserved	0b101	Reserved	0b110	Reserved	0b111	SYS_CLK÷8
MCKO_DIV	MCKO Frequency																		
0b000	SYSCLK÷1																		
0b001	SYSCLK÷2																		
0b010	Reserved																		
0b011	SYS_CLK÷4																		
0b100	Reserved																		
0b101	Reserved																		
0b110	Reserved																		
0b111	SYS_CLK÷8																		
EVT_EN	$\overline{\text{EVTO}}/\overline{\text{EVTI}}$ Enable. This bit enables the $\overline{\text{EVTO}}/\overline{\text{EVTI}}$ port functions. 0 $\overline{\text{EVTO}}/\overline{\text{EVTI}}$ port disabled. 1 $\overline{\text{EVTO}}/\overline{\text{EVTI}}$ port enabled.																		
bits 24–16	Reserved.																		
LP_DBG_EN	Low Power Debug Enable. The LP_DBG_EN bit enables debug functionality to support entry and exit from low power sleep and stop modes. 0 Low power debug disabled. 1 Low power debug enabled.																		
bits 14–10	Reserved.																		

Table 20-4. PCR Field Descriptions (continued)

Field	Description
SLEEP_SYNC	Sleep Mode Synchronization. The SLEEP_SYNC bit is used to synchronize the entry into sleep mode between the device and debug tool. The device sets this bit before a pending entry into sleep mode. After reading SLEEP_SYNC as set, the debug tool then clears SLEEP_SYNC to acknowledge to the device that it may enter into sleep mode. 0 Sleep mode entry acknowledge. 1 Sleep mode entry pending.
STOP_SYNC	Stop Mode Synchronization. The STOP_SYNC bit is used to synchronize the entry into stop mode between the device and debug tool. The device sets this bit before a pending entry into stop mode. After reading STOP_SYNC as set, the debug tool then clears STOP_SYNC to acknowledge to the device that it may enter into stop mode. 0 Stop mode entry acknowledge. 1 Stop mode entry pending
bits 7–1	Reserved.
PSTAT_EN	Processor Status Mode Enable. MPC5510 does not support the PSTAT mode. Setting PSTAT_EN will drive zeros to the MDO and MSEO pins.

20.4.2.3 Development Control Register 1, 2 (DC1, DC2)

The development control registers are used to control the basic development features of the Nexus module. [Figure 20-5](#) shows development control register 1 and [Table 20-5](#) describes the register’s fields.

Nexus Reg: 0x0002

Access: User read/write

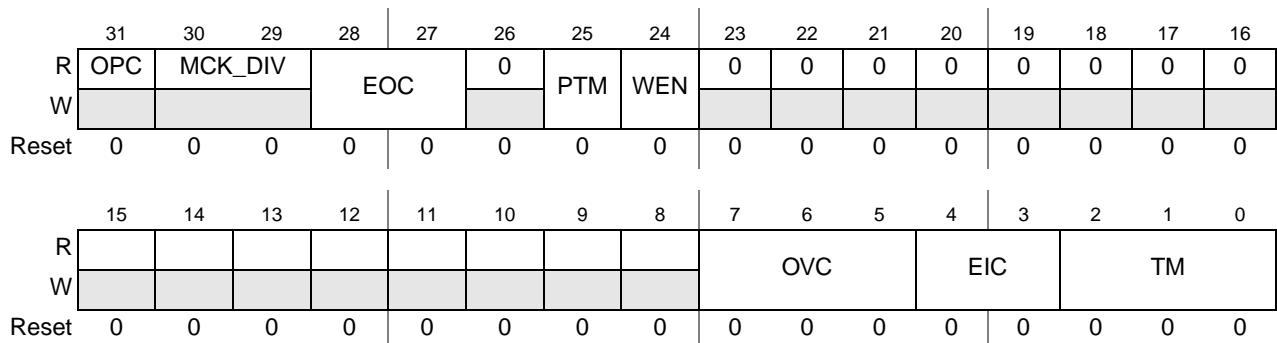


Figure 20-5. Development Control Register 1 (DC1)

Table 20-5. DC1 Field Descriptions

Field	Description
OPC ¹	Output Port Mode Control. 0 Reduced-port mode configuration (4 MDO pins). 1 Full-port mode configuration (8 MDO pins).
MCK_DIV ¹	MCKO Clock Divide Ratio (see note below). 00 MCKO is 1x processor clock frequency. 01 MCKO is 1/2x processor clock frequency. 10 MCKO is 1/4x processor clock frequency. 11 MCKO is 1/8x processor clock frequency.

Table 20-5. DC1 Field Descriptions (continued)

Field	Description
EOC	$\overline{\text{EVTO}}$ Control. 00 $\overline{\text{EVTO}}$ upon occurrence of watchpoints (configured in DC2). 01 $\overline{\text{EVTO}}$ upon entry into debug mode. 10 $\overline{\text{EVTO}}$ upon timestamping event. 11 Reserved.
bit 26	Reserved.
PTM	Program Trace Method. 0 Program trace uses traditional branch messages. 1 Program trace uses branch history messages.
WEN	Watchpoint Trace Enable. 0 Watchpoint messaging disabled. 1 Watchpoint messaging enabled.
bits 23–8	Reserved.
OVC	Overrun Control. 000 Generate overrun messages. 001–010 Reserved. 011 Delay processor for BTM / DTM / OTM overruns. 1XX Reserved.
EIC	$\overline{\text{EVTI}}$ Control. 00 $\overline{\text{EVTI}}$ is used for synchronization (program trace/ data trace). 01 $\overline{\text{EVTI}}$ is used for debug request. 1X Reserved.
TM	Trace Mode. Any or all of the TM bits may set, enabling one or more traces. 000 No trace. 1XX Program trace enabled. X1X Data trace enabled. XX1 Ownership trace enabled.

¹ The output port mode control bit (OPC) and MCKO divide bits (MCK_DIV) are shown for clarity. These functions are controlled globally by the NPC port control register (PCR). These bits are writable in the PCR but have no effect.

Development control register 2 is shown in [Figure 20-6](#) and its fields are described in [Table 20-6](#).

Nexus Reg: 0x0003

Access: User read/write

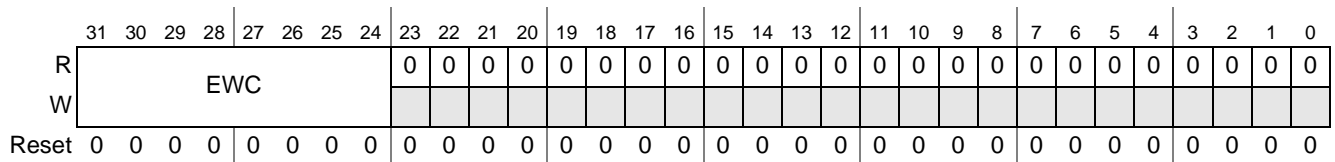


Figure 20-6. Development Control Register 2 (DC2)

Table 20-6. DC2 Field Descriptions

Field	Description
EWC	<p>$\overline{EVT0}$ Watchpoint Configuration. Any or all of the bits in EWC may be set to configure the $\overline{EVT0}$ watchpoint.</p> <p>00000000 No Watchpoints trigger $\overline{EVT0}$</p> <p>1XXXXXXX Watchpoint #0 (IAC1 from Nexus1) triggers $\overline{EVT0}$.</p> <p>X1XXXXXX Watchpoint #1 (IAC2 from Nexus1) triggers $\overline{EVT0}$.</p> <p>XX1XXXXX Watchpoint #2 (IAC3 from Nexus1) triggers $\overline{EVT0}$.</p> <p>XXX1XXXX Watchpoint #3 (IAC4 from Nexus1) triggers $\overline{EVT0}$.</p> <p>XXXX1XXX Watchpoint #4 (DAC1 from Nexus1) triggers $\overline{EVT0}$.</p> <p>XXXXX1XX Watchpoint #5 (DAC2 from Nexus1) triggers $\overline{EVT0}$.</p> <p>XXXXXX1X Watchpoint #6 (DCNT1 from Nexus1) triggers $\overline{EVT0}$.</p> <p>XXXXXXX1 Watchpoint #7 (DCNT2 from Nexus1) triggers $\overline{EVT0}$.</p>
bits 23–0	Reserved.

NOTE

The EOC bits in DC1 must be programmed to trigger $\overline{EVT0}$ on watchpoint occurrence for the EWC bits to have any effect.

20.4.2.4 Development Status Register (DS)

The development status register is used to report system debug status. When debug mode is entered or exited, or a core-defined low-power mode is entered, a debug status message is transmitted with DS. The external tool can read this register at any time.

Nexus Reg: 0x0004

Access: User read only

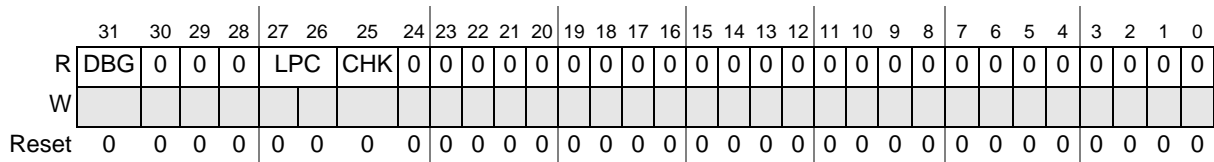


Figure 20-7. Development Status Register (DS)

Table 20-7. DS Field Descriptions

Field	Description
DBG	<p>CPU Debug Mode Status.</p> <p>0 CPU not in debug mode.</p> <p>1 CPU in debug mode.</p>
bits 30–28	Reserved.
LPC	<p>CPU Low-Power Mode Status.</p> <p>00 Normal (run) mode.</p> <p>01 CPU in halted state.</p> <p>10 CPU in stopped state.</p> <p>11 Reserved.</p>

Table 20-7. DS Field Descriptions (continued)

Field	Description
CHK	CPU Checkstop Status. 0 CPU not in checkstop state. 1 CPU in checkstop state.
bits 24–0	Reserved.

20.4.2.5 Read/Write Access Control/Status (RWCS)

The read write access control/status register provides control for read/write access. Read/write access provides DMA-like access to memory-mapped resources on the system bus while the processor is halted or during runtime. The RWCS register also provides read/write access status information as shown in Table 20-9.

Nexus Reg: 0x0007

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R												0	0	0	0	0
W	AC	RW	SZ		MAP			PR		BST						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CNT															
W	CNT													ERR	DV	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-8. Read/Write Access Control/Status Register (RWCS)

Table 20-8. RWCS Field Description

Field	Description
AC	Access Control. 0 End access. 1 Start access.
RW	Read/Write Select. 0 Read access. 1 Write access.
SZ	Word Size. 000 8-bit (byte). 001 16-bit (halfword). 010 32-bit (word). 011 64-bit (doubleword—only in burst mode). 100–111 Reserved (default to word).
MAP	MAP Select. 000 Primary memory map. 001–111 Reserved.

Table 20-8. RWCS Field Description (continued)

Field	Description
PR	Read/Write Access Priority. 00 Lowest access priority. 01 Reserved (default to lowest priority). 10 Reserved (default to lowest priority). 11 Highest access priority.
BST	Burst Control. 0 Module accesses are single bus cycle at a time. 1 Module accesses are performed as burst operation.
bits 20–16	Reserved.
CNT	Access Control Count. Number of accesses of word size SZ.
ERR	Read/Write Access Error. See Table 20-9 .
DV	Read/Write Access Data Valid. See Table 20-9 .

[Table 20-9](#) details the status bit encodings.

Table 20-9. Read/Write Access Status Bit Encoding

Read Action	Write Action	ERR	DV
Read access has not completed	Write access completed without error	0	0
Read access error has occurred	Write access error has occurred	1	0
Read access completed without error	Write access has not completed	0	1
Not allowed	Not allowed	1	1

20.4.2.6 Read/Write Access Address (RWA)

The read/write access address register provides the system bus address to be accessed when initiating a read or a write access.

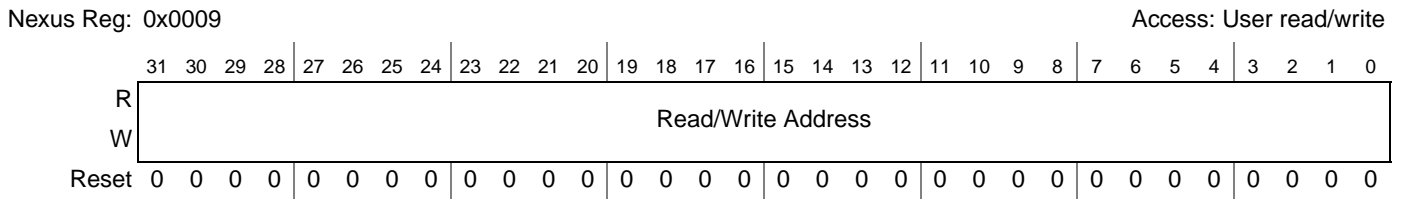


Figure 20-9. Read/Write Access Address Register (RWA)

20.4.2.7 Read/Write Access Data (RWD)

The read/write access data register provides the data to/from system bus memory-mapped locations when initiating a read or a write access.

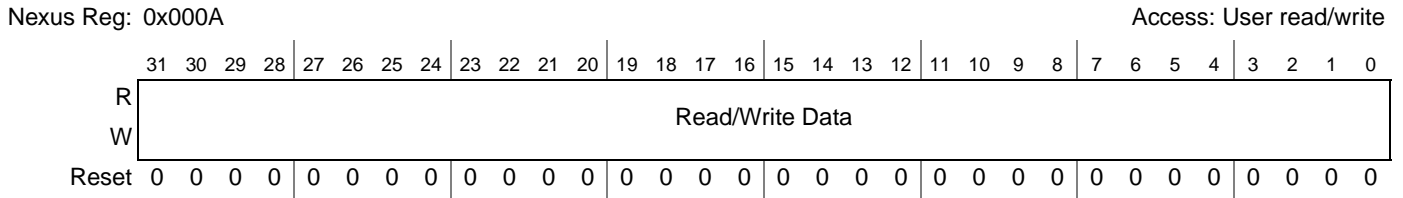


Figure 20-10. Read/Write Access Data Register (RWD)

20.4.2.8 Watchpoint Trigger Register (WT)

The watchpoint trigger register allows the watchpoints defined within the Nexus1 logic to trigger actions. These watchpoints can control program and/or data trace enable and disable. The WT bits can be used to produce an address-related window for triggering trace messages.

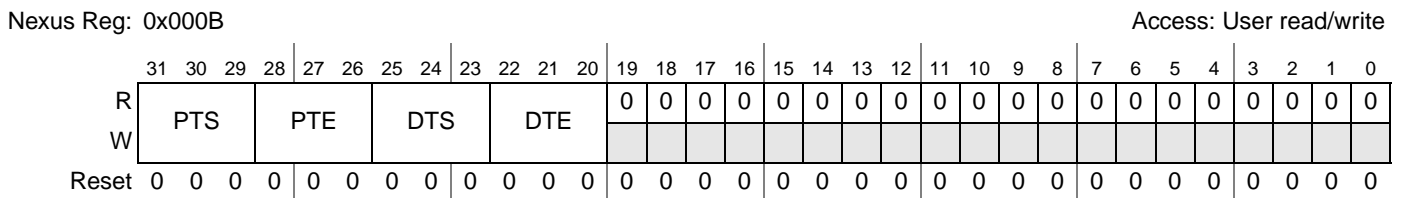


Figure 20-11. Watchpoint Trigger Register (WT)

Table 20-10 details the watchpoint trigger register fields.

Table 20-10. WT Field Descriptions

Field	Description
PTS	Program Trace Start Control. 000 Trigger disabled. 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).
PTE	Program Trace End Control. 000 Trigger disabled. 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).

Table 20-10. WT Field Descriptions (continued)

Field	Description
DTS	Data Trace Start Control. 000 Trigger disabled. 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).
DTE	Data Trace End Control. 000 Trigger disabled. 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).
bits 19–0	Reserved.

NOTE

The WT bits will only control program/data trace if the TM bits in the development control register 1 (DC1) have not already been set to enable program and data trace, respectively.

20.5 Functional Description

The NDI block is implemented by integrating the following blocks on the MPC5510:

- Nexus e200z0 development interface (OnCE and Nexus2p sub-blocks)
- Nexus port controller (NPC) Block

20.5.1 Enabling Nexus Clients for TAP Access

After the conditions have been met to bring the NDI out of the reset state, the loading of a specific instruction in the JTAG controller (JTAGC) block is required to grant the NDI ownership of the TAP. Each Nexus client has its own JTAGC instruction opcode for ownership of the TAP, granting that client the means to read/write its registers. The JTAGC instruction opcode for each Nexus client is shown in [Table 20-11](#). After the JTAGC opcode for a client has been loaded, the client is enabled by loading its NEXUS-ENABLE instruction. The NEXUS-ENABLE instruction opcode for each Nexus client is listed in [Table 20-12](#). Opcodes for all other instructions supported by Nexus clients can be found in the relevant sections of this chapter.

When the ACCESS_AUX_TAP_MULTI instruction has been loaded into the JTAGC OCMD register, the TDO output of the e200z1 TAP controller will be connected to the TDI input to the e200z0 TAP controller.

This allows debug commands to be simultaneously loaded into both TAP controllers, and is useful in getting the cores to simultaneously exit debug mode for example.

Table 20-11. JTAGC Instruction Opcodes to Enable Nexus Clients

JTAGC Instruction	Opcode	Description
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller.
ACCESS_AUX_TAP_ONCE	10001	Enables access to the e200z1 TAP controller.
ACCESS_AUX_TAP_Z0	11001	Enables access to the e200z0 TAP controller.
ACCESS_AUX_TAP_MULTI	11100	Enables access to the e200z1 and e200z0 TAP controllers chained together.

Table 20-12. Nexus Client JTAG Instructions

Instruction	Description	Opcode
NPC JTAG Instruction Opcodes		
NEXUS_ENABLE	Opcode for NPC Nexus ENABLE instruction (4-bits)	0x0
BYPASS	Opcode for the NPC BYPASS instruction (4-bits)	0xF
e200z0 OnCE JTAG Instruction Opcodes¹		
NEXUS2_ACCESS	Opcode for e200z0 OnCE Nexus ENABLE instruction (10-bits)	0x7C
BYPASS	Opcode for the e200z0 OnCE BYPASS instruction (10-bits)	0x7F
e200z1 OnCE JTAG Instruction Opcodes²		
BYPASS	Opcode for the e200z1 OnCE BYPASS instruction (10-bits)	0x7F

¹ Refer to the e200z0 reference manual for a complete list of available OnCE instructions.

² Refer to the e200z1 reference manual for a complete list of available OnCE instructions.

20.5.2 Configuring the NDI for Nexus Messaging

The NDI is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the port configuration register (PCR) located in the NPC is then required to enable the NDI and select the mode of operation. Asserting MCKO_EN in the PCR places the NDI in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting or negating the FPM bit selects full-port or reduced-port mode, respectively. When writing to the PCR, the PCR LSB must be written to a logic zero. Setting the LSB of the PCR enables factory debug mode and prevents the transmission of Nexus messages.

Table 20-13 describes the NDI configuration options.

Table 20-13. NDI Configuration Options

JCOMP Asserted	MCKO_EN bit of the Port Configuration Register	FPM bit of the Port Configuration Register	Configuration
No	X	X	Reset
Yes	0	X	Disabled

Table 20-13. NDI Configuration Options (continued)

JCOMP Asserted	MCKO_EN bit of the Port Configuration Register	FPM bit of the Port Configuration Register	Configuration
Yes	1	1	Full-port mode
Yes	1	0	Reduced-port mode

20.5.3 Switching Ownership of Nexus2+

On MPC5510, the Nexus2+ is shared by the e200z1 and e200z0 cores. Out of reset, the default ownership of the Nexus2+ belongs to the e200z0 core.

To switch the trace between cores, without a system reset, requires a software reset of the Nexus2+ registers to clear the previously programmed values. The following sequence is an example of a software reset. It is recommended that both CPUs be placed in debug mode when switching program trace.

1. Disable tracing by writing to the TM bits in DC1.
2. Switch control to the other core by writing to the NT bits in SNC.
3. Reprogram all trace related Nexus2+ registers as desired.
4. Enable tracing by writing to the TM bits in DC1 or by writing to WT.
5. Exit debug mode.

20.5.4 Programmable MCKO Frequency

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSE0}}$ and MDO pin functions. MCKO is derived from the system clock, and its frequency is determined by the value of the MCKO_DIV field in the port configuration register (PCR) located in the NPC. Possible operating frequencies include one-quarter and one-eighth system clock speed.

Table 20-14 shows the MCKO_DIV encodings. In this table, SYS_CLK represents the system clock frequency. The default value selected if a reserved encoding is programmed is $\text{SYS_CLK} \div 1$.

NOTE

On MPC5510, the pad type used for the Nexus2+ signals will not support the $\text{SYSCLK} \div 1$ or $\text{SYSCLK} \div 2$ setting, so the user must change the MCKO frequency to be not faster than $\text{SYSCLK} \div 4$.

Table 20-14. MCKO_DIV Values

MCKO_DIV	MCKO Frequency
0b000	$\text{SYSCLK} \div 1$
0b001	$\text{SYSCLK} \div 2$
0b010	Reserved
0b011	$\text{SYS_CLK} \div 4$
0b100	Reserved

Table 20-14. MCKO_DIV Values (continued)

MCKO_DIV	MCKO Frequency
0b101	Reserved
0b110	Reserved
0b111	SYS_CLK÷8

20.5.5 Nexus Messaging

Most of the messages transmitted by the NDI include an SRC field. This field is used to identify which source generated the message. Figure 20-15 shows the values used for the SRC field by the different clients on the MPC5510. These values are specific to the MPC5510. The size of the SRC field in transmitted messages is 4 bits. This value is also specific to the MPC5510. The same values are used for the client select values written to the client select control register.

Table 20-15. SRC Packet Encodings

SRC	MPC5510 Client
0b0000	e200z1
0b1000	e200z0
All other combinations	Reserved

20.5.6 $\overline{\text{EVTO}}$ Sharing

The NPC block controls sharing of the $\overline{\text{EVTO}}$ output between all Nexus clients that generate an $\overline{\text{EVTO}}$ signal. The sharing mechanism is a logical AND of all incoming $\overline{\text{EVTO}}$ signals from Nexus blocks, thereby asserting $\overline{\text{EVTO}}$ whenever any block drives its $\overline{\text{EVTO}}$. When there is no active MCKO, such as in disabled mode, the NPC drives $\overline{\text{EVTO}}$ for two system clock periods. $\overline{\text{EVTO}}$ sharing is active as long as the NDI is not in reset.

20.5.7 Nexus2+ DMA Control

The shared Nexus2+ implementation allows each core to perform DMA access independently of the core being traced. The setting of the ND bits in the OnCE shared Nexus control register (SNC) determines which CPU's master ID is used during Nexus2+ DMA access. Depending on the configuration of the MPU, different master IDs in DMA transfers may be needed to distinguish which core initiated the access to the peripheral, so that the access is not blocked by the MPU.

The DMA control can be switched between the CPUs whenever the Nexus2+ DMA is idle.

20.5.8 Debug Mode Control

On MPC5510, program breaks can be requested either by using the $\overline{\text{EVTI}}$ pin as a break request, or when a Nexus event is triggered. These break requests can affect either of the e200z1 or e200z0 cores, or can be used to request a break for both cores.

20.5.8.1 $\overline{\text{EVTI}}$ Generated Break Request

To use the $\overline{\text{EVTI}}$ pin as a debug request, the EIC field in the e200z0 Nexus2+ Development Control Register 1 (DC1) must be set to configure the $\overline{\text{EVTI}}$ input as a debug request. The Shared Nexus Control Register (SNC) $\overline{\text{EVTI}}$ Debug Request Enable field can then be used to control which core, or both, responds to the assertion of the $\overline{\text{EVTI}}$ pin.

20.5.8.2 Nexus Event-Out Generated Break Request

On MPC5510, the Nexus2+ event-out signal has been connected to the external debug event 2 input to both cores. This allows the assertion of a Nexus event-out signal to trigger a debug request to either or both cores.

To use the Nexus event-out signal as a debug request, the EOC field in the e200z0 Nexus2+ development control register 1 (DC1) must be set to enable the assertion of the event-out signal on the occurrence of a watchpoint, entry into debug mode, or upon a time-stamping event.

To enable the debug request for a given core, the DEVT2 field of the associated core's OnCE debug control register 0 (DBCR0) must be set to enable debug events when the DEVT2 input asserts.

A description of Nexus support when leaving low power sleep mode is given in [Section 5.3.4.1, “Low Power Mode Debug Support”](#).

[Figure 20-12](#) and [Figure 20-13](#) show the process flow to initialize the e200z1 as the core generating the event-out break request, and the e200z0 as the core receiving the break request. This configuration will cause the e200z0 to follow the e200z1 into debug mode. The e200z0 may be configured as the core generating the break request, and the e200z1 as the core receiving the break request, by reversing the roles of the e200z0 and e200z1 in [Figure 20-12](#) and [Figure 20-13](#).

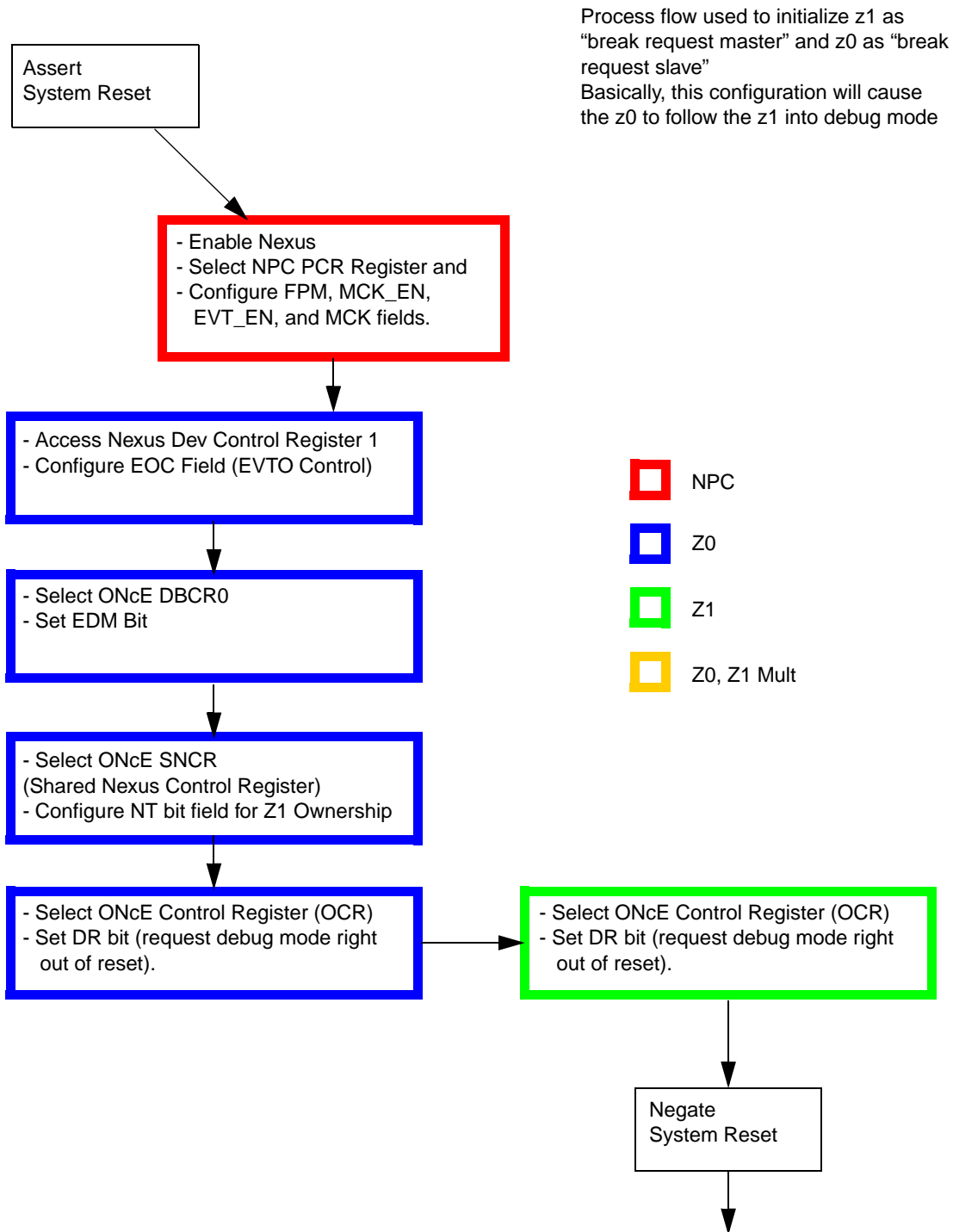


Figure 20-12. Nexus Event-Out Generated Break Request (5510) — Part 1

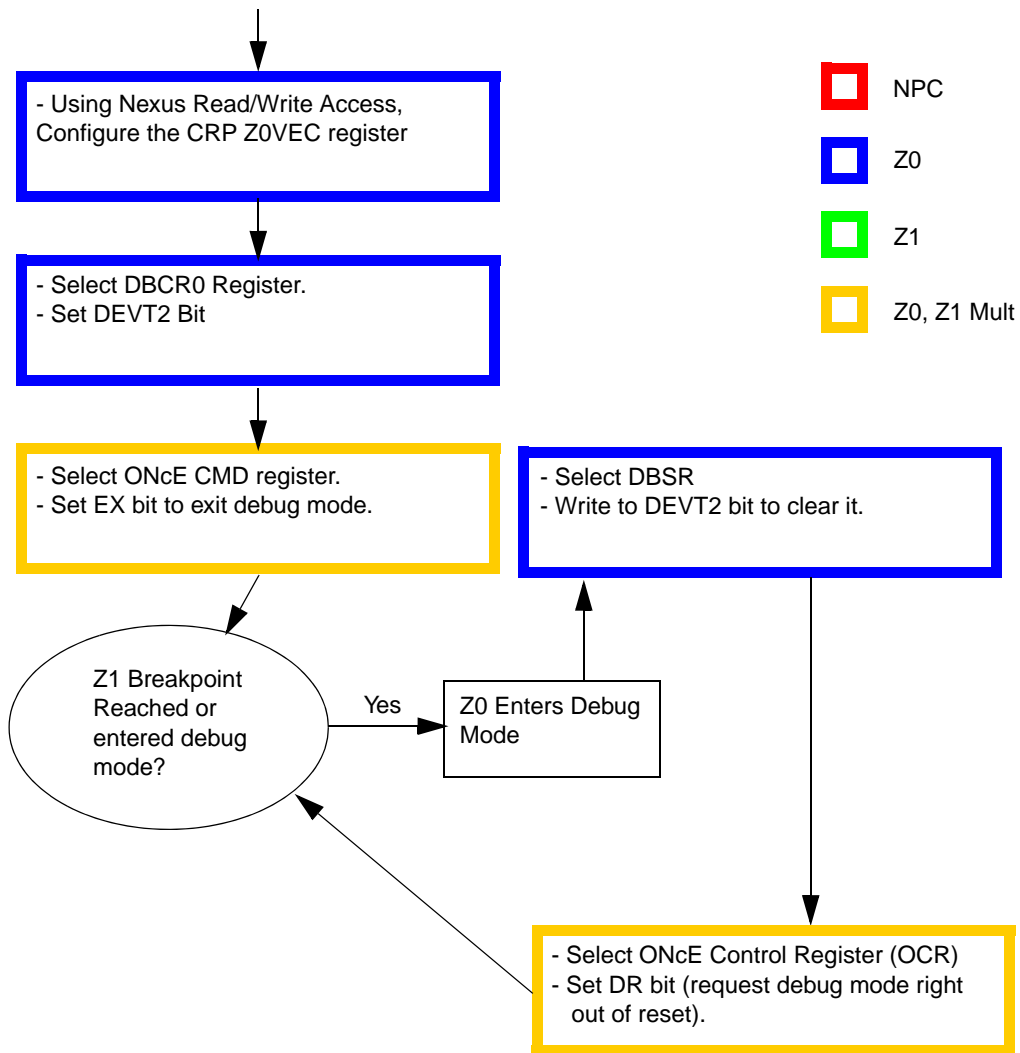


Figure 20-13. Nexus Event-Out Generated Break Request (5510) — Part 2

20.5.9 Nexus Reset Control

The JCOMP input that is used as the primary reset signal for the NPC is also used by the NPC to generate a single-bit reset signal for other Nexus blocks. The single bit reset signal functions much like the IEEE 1149.1-2001 defined $\overline{\text{TRST}}$ signal but has a default value of disabled (JCOMP is pulled low during reset). The IEEE 1149.1-2001 defines $\overline{\text{TRST}}$ to be pulled up (enabled) by default.

Chapter 21

Internal Static RAM (SRAM)

21.1 Introduction

The MPC5510 provides 80 KB of general-purpose system SRAM, that is implemented using ten 8 KB arrays. This implementation allows a configurable number of arrays to remain powered during low-power sleep modes.

21.1.1 Block Diagram

A simplified block diagram of the SRAM illustrates the functionality and interdependence of major blocks (see [Figure 21-1](#)).

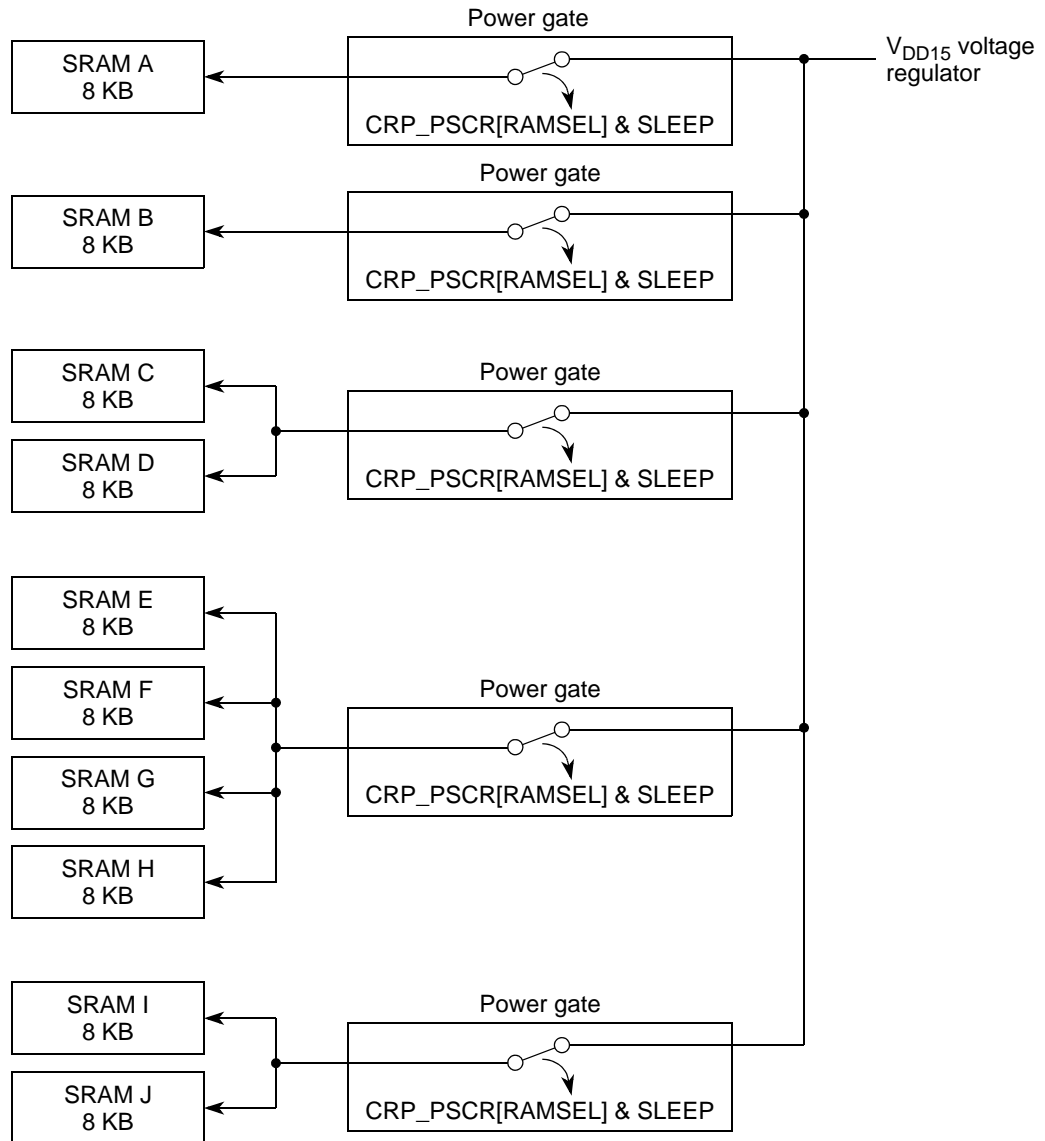


Figure 21-1. SRAM Block Diagram

21.1.2 Features

The SRAM has these major features:

- Supports read/write accesses mapped to the SRAM memory from any master
- Configurable number of 8 KB blocks powered during low-power sleep
- Byte, halfword, and word addressable
- Error correcting code (ECC) performs single bit correction, double bit detection on a 32-bit boundary

21.1.3 Modes of Operation

There are two main operating modes of SRAM: normal mode and sleep mode. These modes are briefly described in this section.

21.1.3.1 Normal (Functional) Mode

Allows for reads and writes of the SRAM memory arrays.

21.1.3.2 Sleep Mode

Preserves the contents of the portion of the memory, as defined by the CRP_PSCR[RAMSEL] register, during low-power sleep mode.

21.2 External Signal Description

There are no external signals associated with the SRAM.

21.3 Memory Map and Registers

This section provides an array memory map of the SRAM. There are no registers associated with the SRAM.

21.3.1 Array Memory Map

Table 21-1 list the addresses ranges of the SRAM whose contents are maintained during the various low-power sleep modes. All SRAM is powered during run and stop modes.

Table 21-1. SRAM Array Address Range

Address Range	Size	Description
0x4000_0000–0x4000_1FFF	8 KB	RAM array A
0x4000_2000–0x4000_3FFF	8 KB	RAM array B
0x4000_4000–0x4000_5FFF	8 KB	RAM array C
0x4000_6000–0x4000_7FFF	8 KB	RAM array D
0x4000_8000–0x4000_9FFF	8 KB	RAM array E
0x4000_A000–0x4000_BFFF	8 KB	RAM array F
0x4000_C000–0x4000_DFFF	8 KB	RAM array G
0x4000_E000–0x4000_FFFF	8 KB	RAM array H
0x4001_0000–0x4001_1FFF	8 KB	RAM array I
0x4000_2000–0x4001_3FFF	8 KB	RAM array J

21.3.2 Register Descriptions

The SRAM has no registers. The registers associated with the ECC are located in the MCM, see [Section 16.2.2.5, “ECC Registers,”](#) in [Chapter 16, “Miscellaneous Control Module \(MCM\).”](#)

21.4 Functional Description

The AMBA-AHB bus is a two stage pipelined bus that may require the SRAM controller to insert a wait state during certain types of back-to-back accesses. The SRAM controller implements a late-write-buffer to enable zero wait state write-read combinations.

To implement ECC, the RAM BIU generates a 39 bit code word based upon a 32 bit data write. During a read operation, single bit corrections are made and multiple bit errors are flagged. The ECC code that was chosen will perform single bit corrections and indicate a multiple bit error on all double-bit errors. A code word of 39 zeros and 39 ones will cause a multiple bit error. Multiple bit errors will assert an error indication with the bus cycle, as well as setting the PRNCE bit in the MCM’s ESR.

During a write operation for 8-bit and 16-bit writes, a read of 32-bit data will be checked for ECC, prior to merging in the write data. If a correction is required, it will be corrected prior to merging in the write data. Then a new ECC code word is generated and written to the RAM. If a multiple bit error occurs during the read portion of the write operation, then the write will not be performed.

CAUTION

It is essential for the ECC check bits to be initialized after power on. The write transfer must be 32 bits in size because a less than 32 bit write transfer will generate a read/modify/write operation that will check the ECC value upon the read.

21.4.1 Access Timing

The AMBA-AHB bus is a two stage pipelined bus, which makes the timing of any access dependent on the access during the previous clock. [Table 21-2](#) shows the wait states for accesses. Current is the type of access being measured during the current clock. Previous is the SRAM access during the previous clock.

Table 21-2. Wait States During SRAM Access

Current	Previous	Wait States
Read	Idle	0
	Read	0
	32-bit write	0
	8/16-bit write	1
32-bit write	Idle	0
	Read	0
	32-bit write	0
	8/16-bit write	1

Table 21-2. Wait States During SRAM Access (continued)

Current	Previous	Wait States
8/16-bit write	Idle	0
	Read	0
	32-bit write	0
	8/16-bit write	1

21.4.2 Reset Operation

A 'destructive' reset is associated with an event after which critical register or memory content can no longer be guaranteed, if a write operation occurred during the event.

'Destructive' resets are as follows.

- Power-on reset
- 1.5 V low-voltage detect
- 3.3 V low-voltage detect
- 3.3SynV low-voltage detect
- 5.0Vlv51 low-voltage detect
- 5.0Vlv5 low-voltage detect (if enabled)
- External reset
- PLL loss of clock (if enabled)
- PLL loss of lock (if enabled)

The user code must re-initialize the RAM after any of the above resets; otherwise, an ECC event might occur.

21.5 DMA Requests

There are no DMA requests associated with the system SRAM.

21.6 Interrupt Requests

There are no interrupt requests associated with the system SRAM, except for the ECC reporting through the MCM module.

21.7 Initialization/Application Information

You must initialize the ECC check bits after the device is powered on before you can use the SRAM. The write transfer must be 32 or 64 bits, on a 32-bit boundary. If not, a read/modify/write operation is generated that checks the ECC value upon the read.

NOTE

You must initialize the SRAM, even if the application does not use ECC reporting. If using a part with less than 80 KB of RAM, you must also initialize four words past the end address (or up to the entire 80 KB space).

21.7.1 Example Code

It is essential that each memory address be written to a known value before it is read. This includes reads generated from the read/modify/write operation which occurs when a write transfer of less than 32 bits or unaligned write is requested. Without writing an address to a known value first, a read from this address will most likely generate an ECC multiple error.

The following Book E assembly instructions ([Example 21-1](#)), formed as a subroutine, is an example of clearing SRAM memory space. This code clears 0x2000 bytes of memory.

Example 21-1. Clearing SRAM Memory Space

```
enable_and_invalidate_sram();
asm ("li    r14,0x2000");      // size to clear
asm ("lis   r13,0x4000");      // RAM base address
asm ("clear_mem:");
asm ("subi  r14,r14,0x20");
asm ("dcbz  r14,r13");
asm ("dcbf  r14,r13");
asm ("cmpwi r14,0x0");
asm ("beq   continue");
asm ("b     clear_mem");
```

Chapter 22

Flash Array and Control

22.1 Introduction

The primary function of the flash memory block is to serve as electrically programmable and erasable non-volatile memory. The NVM memory can be used for instruction and data storage. The block is a non-volatile solid-state silicon memory device consisting of blocks of single-transistor storage elements, an electrical means for selectively adding (programming) and removing (erasing) charge from these elements, and a means of selectively sensing (reading) the charge stored in these elements. The flash is addressable by word (32 bits) and page (128 bits).

The flash block is arranged as two functional units. The first functional unit is the flash core (FC). The FC is composed of arrayed non-volatile storage elements, sense amplifiers, row selects, column selects, charge pumps, and redundancy logic. The arrayed storage elements in the FC are sub-divided into physically separate units referred to as blocks.

The second functional unit of the flash is the memory interface (MI). The MI contains the registers and logic which control the operation of the FC. The MI is also the interface to the platform flash bus interface unit (PFBIU).

The flash core has three address spaces. The low-address space is 256 KB. The mid-address space is also 256 KB. The high-address space is 1 MB. The 256 KB of low memory will be implemented using eight 16 KB blocks and two 64 KB blocks. The mid and high memory will be implemented using ten 128 KB blocks.

[Figure 22-1](#) shows the segmentation for the flash on MPC5510.

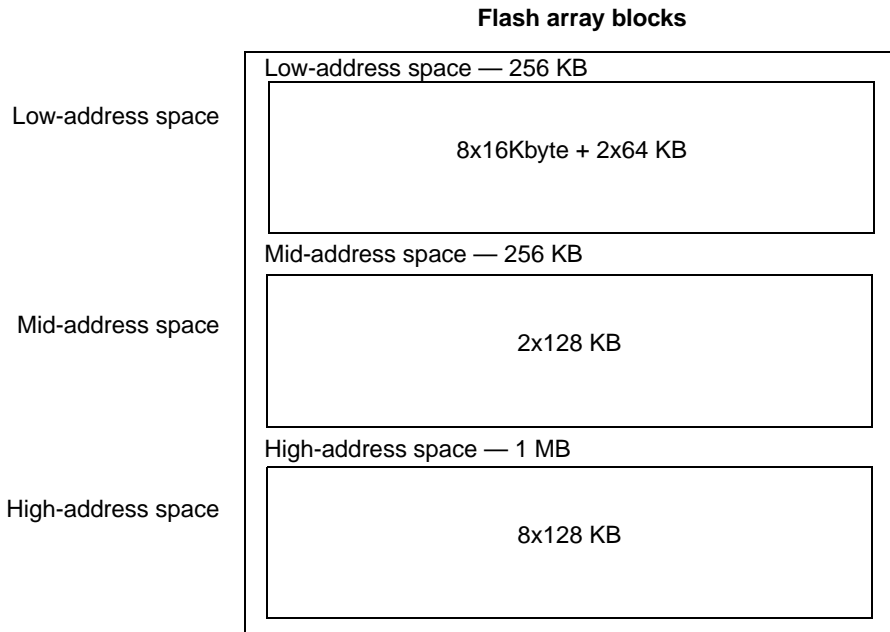
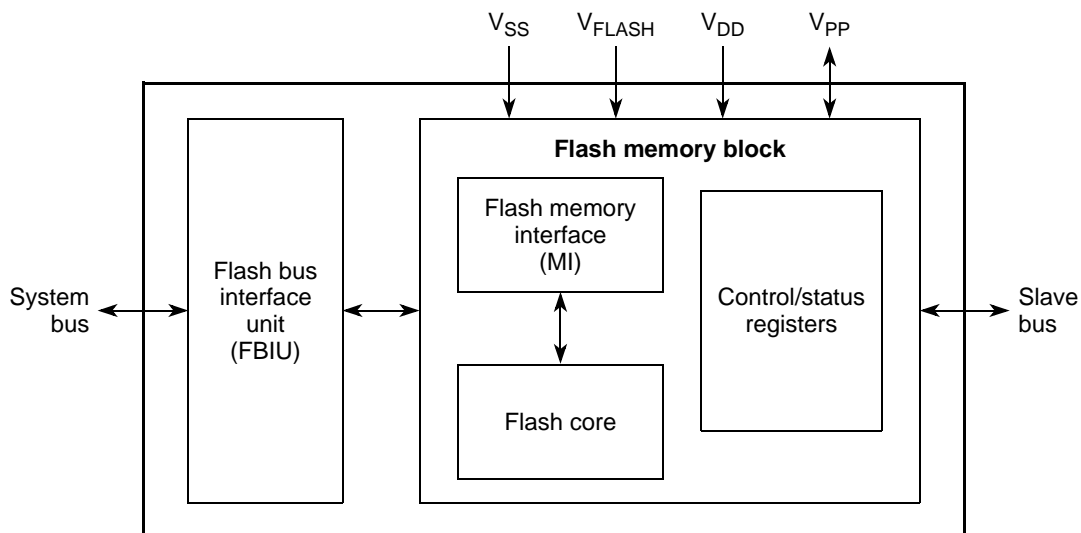


Figure 22-1. Flash Segmentation

22.2 Block Diagram

Figure 22-2 shows a block diagram of the flash memory module. The FBIU is addressed through the system bus while the flash control and status registers are addressed through the slave (peripheral) bus.



Note: V_{PP} is the only externally visible power supply that is necessary for the programming and erasing of the flash array (see Section 22.3, “External Signal Description.”)

Figure 22-2. Flash System Block Diagram

22.2.1 Features

The flash has these major features:

- Software programmable block program/erase restriction control for low-, mid-, and high-address space
- Erase of selected block(s)
- Read page size of 128 bits (4 words)
- ECC with single-bit correction, double-bit detection
- Page programming (64-bit granularity)
- Embedded hardware program and erase algorithm
- Read while write with multiple partitions
- Stop mode for low-power stand-by
- Erase suspend, program suspend and erase-suspended program
- Automotive flash, which meets automotive endurance and reliability requirements
- Shadow information stored in non-volatile shadow block
- Independent program/erase of shadow block

22.2.2 Modes of Operation

There are three main operating modes of flash: user mode, stop mode, and disable mode. These modes are briefly described in this section.

User mode is the default operating mode of the flash module. In this mode, it is possible to read and write, program and erase the flash module. In user mode, program and erase operations are initiated by the user, but controlled by an internal state machine.

Stop mode is a low-power stand-by mode in which only read and write of the MCR register space is enabled. Stop mode turns off all DC current sources within the flash module.

Disable mode turns off all DC current sources and no reads from or writes to the flash module are allowed. The FC and registers are not accessible for read and write after they are disabled.

22.3 External Signal Description

V_{pp} is the only externally visible power supply that is necessary for programming and erasing the flash array. The other flash supplies are tied to the appropriate supply pads in the package. Refer to [Table 2-1](#) and [Section 2.7, “Detailed External Signal Descriptions,”](#) and the *MPC5510 Microcontroller Family Data Sheet*.

22.4 Memory Map and Registers

This section provides a detailed description of all flash registers.

22.4.1 Module Memory Map

The flash memory map is shown in [Table 22-1](#). The addresses are given as an offset to the flash base address.

The flash register memory map is shown in [Table 22-2](#). There are no program-visible registers that physically reside inside the flash. The flash receives control and configuration information from the flash array controller to determine operating configurations. These are part of the flash array controller's configuration registers mapped into the IPS address space but are described herein. These registers should only be referenced with 32-bit accesses.

Table 22-1. Flash Memory Map

Offset from FLASH_BASE (0x0000_0000)	Use	Block	Partition
0x0000_0000	Low-address space	L0	1
0x0000_4000		L1	
0x0000_8000		L2	
0x0000_C000		L3	
0x0001_0000		L4	2
0x0001_4000		L5	
0x0001_8000		L6	
0x0001_C000		L7	
0x0002_0000		L8	3
0x0003_0000	L9		
0x0004_0000	Mid-address space	M0	4
0x0006_0000		M1	
0x0008_0000	High-address space	H0	5
0x000A_0000		H1	
0x000C_0000		H2	6
0x000E_0000		H3	
0x0010_0000		H4	7
0x0012_0000		H5	
0x0014_0000		H6	8
0x0016_0000		H7	
0x0018_0000–0xF0_FFFF		Reserved	

Table 22-1. Flash Memory Map (continued)

Offset from FLASH_BASE (0x0000_0000)	Use	Block	Partition
0x00FF_8000–0x00FF_FDD3	General use	S	All ¹
0x00FF_FDD8	Serial passcode (0xFEED_FACE_CAFE_BEEF)		
0x00FF_FDE0	Censorship control word (0x55AA_55AA)		
0x00FF_FDE4	IRC trim 8 bytes unused		
0x00FF_FDE5	IRC trim 8 bytes unused		
0x00FF_FDE6	TRIM32IRC 8 bytes		
0x00FF_FDE7	TRIMIRC 8 bytes		
0x00FF_FDE8	LML reset configuration (0x0010_0000)		
0x00FF_FDEC	General use		
0x00FF_FDF0	HBL reset configuration (0xFFFF_FFFF)		
0x00FF_FDF4	General use		
0x00FF_FDF8	SLL reset configuration (0x000F_FFFF)		
0x00FF_FDFC– 0x00FF_FFFF	General use		

¹ For read while write operations, the shadow row behaves as if it is in all partitions.

Table 22-2. Flash Configuration Register Memory Map

Offset from FLASH_REGS_BASE (0xFFFF_8000)	Register	Access	Reset Value	Section/Page
0x0000	MCR—Module configuration register	R/W ¹	0x0540_3200	22.4.2.1/22-6
0x0004	LML—Low-/Mid-address space block locking register	R/W ¹	0x001F_FFFF	22.4.2.2/22-9
0x0008	HBL—High-address space block locking register	R/W ¹	0x0FFF_FFFF	22.4.2.3/22-11
0x000C	SLL—Secondary low-/mid-address space block locking register	R/W ¹	0x001F_FFFF	22.4.2.4/22-12
0x0010	LMS—Low-/mid-address space block select register	R/W ¹	0x0000_0000	22.4.2.5/22-13
0x0014	HBS—High-address space block select register	R/W ¹	0x0000_0000	22.4.2.6/22-14
0x0018	ADR—Address register	R/W ¹	0x0000_0000	22.4.2.7/22-14
0x001C	PFCRP0—Platform flash configuration register for port 0	R/W ¹	0x0000_FF00	22.4.2.8/22-15
0x0020	PFCRP1—Platform flash configuration register for port 1	R/W ¹	0x3000_FF00	22.4.2.8/22-15
0x0028 – 0x3FFF	Reserved			

¹ Some bits are read only.

22.4.2 Register Descriptions

This section lists the flash registers in address order and describes the registers and their bit fields.

22.4.2.1 Module Configuration Register (MCR)

The MCR reset value is 0x0540_3200. This may read as 0x0540_3600 if enough time has passed to allow the DONE bit to be set.

Offset: FLASH_REGS_BASE + 0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	SFS	SIZE				0	LAS			0	0	0	MAS
W																
Reset	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE	BBEPE	EPE	PEAS	DONE	PEG	0	PRD	STOP	0	PGM	PSUS	ERS	ESUS	EHV
W	w1c	w1c														
Reset	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0

Figure 22-3. Module Configuration Register (MCR)

Table 22-3. MCR Field Descriptions

Field	Description
bits 0–2	Reserved.
SFS	Special Flash Sector. For MPC5510, this read-only bit field is 0b0 indicating no special flash sector size.
SIZE	The value of the SIZE field depends on the size of the flash module. For MPC5510, this bit field is 0b0101 indicating a 1.5 MB array size (with 1 MB in high-address space).
bit 8	Reserved.
LAS	Low-Address Space Size. The value of the LAS field corresponds to the configuration of the low-address space. For MPC5510, this bit field is 0b100 indicating 8 x 16 KB + 2 x 64 KB blocks in low-address space.
bits 12–14	Reserved.
MAS	Mid-Address Space Size. The value of the MAS corresponds to the configuration of the mid-address space. For MPC5510, this bit field is 0b0 indicating 2 x 128 KB blocks in mid-address space.
EER	ECC Event Error. EER provides information on previous reads. If a double bit detection occurred, the EER bit will be set to 1. This bit must then be cleared or a reset must occur before this bit will return to a 0 state. This bit may not be set by the user. In the event of a single bit detection and correction, this bit will not be set. If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) were correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 will have no effect 0 Reads are occurring normally 1 An ECC Error occurred during a previous read
RWE	Read While Write Event Error. Provides information on previous RWW reads. If a read while write error occurs, this bit will be set to 1. This bit must then be cleared or a reset must occur before this bit will return to a 0 state. This bit may not be written to a 1 by the user. If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) were correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 will have no effect. 0 Reads are occurring normally 1 A read while write error occurred during a previous read
18 BBEPE	Boot Block External Program Erase Status. This read-only bit reads as 1.
19 EPE	External Program Erase Status. EPE is a hardware lock that indicates that all blocks including the shadow block and excluding the boot block are enabled for program/erase. This read-only bit reads as 1.

Table 22-3. MCR Field Descriptions (continued)

Field	Description
20 PEAS	Program/Erase Access Space. Indicates which space is valid for program and erase operations, either main array space or shadow space. PEAS is read only. 0 Shadow address space is disabled for program/erase and main address space enabled 1 Shadow address space is enabled for program/erase and main address space disabled
21 DONE	State Machine Status. Indicates if the flash module is performing a high-voltage operation. DONE is set to a 1 on termination of the flash module reset and at the end of program and erase high-voltage sequences. 0 Flash is executing a high-voltage operation 1 Flash is not executing a high-voltage operation
22 PEG	Program/Erase Good. Indicates the completion status of the last flash program or erase sequence for which high-voltage operations were initiated. The value of PEG is updated automatically during the program and erase high-voltage operations. Aborting a program/erase high-voltage operation will cause PEG to be cleared, indicating the sequence failed. PEG is set to a 1 when the module is reset. PEG is read only. The value of PEG is valid only when PGM = 1 or ERS = 1 and after DONE has transitioned from 0 to 1 due to an abort or the completion of a program/erase operation. PEG is valid until PGM/ERS makes a 1 to 0 transition or EHV makes a 0 to 1 transition. The value in PEG is not valid after a 0 to 1 transition of DONE caused by PSUS or ESUS being set to logic 1. A diagram presenting PEG valid times is shown in Figure 22-4 . If PGM and ERS are both 1 when DONE makes a qualifying 0 to 1 transition the value of PEG indicates the completion status of the PGM sequence. This happens in an erase-suspended program operation. 0 Program or erase operation failed. 1 Program or erase operation successful.
23	Reserved.
24 PRD	Pipelined Reads Disabled. PRD is used to allow pipelined reads to be disabled. By default PRD is 0, which enables pipelined accesses. In systems with slower clocks (<30 MHz), the pipelined read feature can be disabled by writing this bit to a 1. This would allow single cycle clock accesses in systems with a slower clock. In systems with faster clocks (>30 MHz), accesses will be multiple cycles, and the pipelined read feature may be used to get faster throughput on successive reads (PRD = 0). 1 Pipelined reads are disabled. 0 Pipelined reads are enabled. Note: PRD must be set before setting the flash wait states to 0.
25 STOP	Stop Mode Enabled. Puts the flash into stop mode. Changing the value in STOP from a 0 to a 1 places the flash module in stop mode. A 1 to 0 transition of STOP returns the flash module to normal operation. STOP may be written only when PGM and ERS are low. When STOP = 1, only the STOP bit in the MCR can be written. In STOP mode all address spaces, registers, and register bits are deactivated except for the MCR[STOP] bit. 0 Flash is not in stop mode; the read state is active 1 Flash is in stop mode
26	Reserved.
27 PGM	Program. Used to set up flash for a program operation. A 0-to-1 transition of PGM initiates a flash program sequence. A 1-to-0 transition of PGM ends the program sequence. PGM can be set under one of the following conditions only: <ul style="list-style-type: none"> • User mode read (STOP and ERS are low). • Erase suspend¹ (ERS and ESUS are 1) with EHV low. PGM can be cleared by the user only when PSUS and EHV are low and DONE is high. PGM is cleared on reset. 0 Flash is not executing a program sequence 1 Flash is executing a program sequence

Table 22-3. MCR Field Descriptions (continued)

Field	Description
28 PSUS	<p>Program Suspend. Indicates the flash module is in program suspend or in the process of entering a suspend state. The flash module is in program suspend when PSUS = 1 and DONE = 1. PSUS can be set high only when PGM and EHV are high. A 0 to 1 transition of PSUS starts the sequence which sets DONE and places the flash in program suspend. PSUS can be cleared only when DONE and EHV are high. A 1 to 0 transition of PSUS with EHV = 1 starts the sequence which clears DONE and returns the flash module to program. The flash module cannot exit program suspend and clear DONE while EHV is low. PSUS is cleared on reset.</p> <p>0 Program sequence is not suspended 1 Program sequence is suspended</p>
29 ERS	<p>Erase. Used to set up flash for an erase operation. A 0 to 1 transition of ERS initiates an flash erase sequence. A 1 to 0 transition of ERS ends the erase sequence. ERS can be set in a normal operating mode read only (STOP and PGM are low). ERS can be cleared by the user only when ESUS and EHV are low and DONE is high. ERS is cleared on reset.</p> <p>0 Flash is not executing an erase sequence 1 Flash is executing an erase sequence</p>
30 ESUS	<p>Erase Suspend. Indicates that the flash module is in erase suspend or in the process of entering a suspend state. The flash module is in erase suspend when ESUS = 1 and DONE = 1. ESUS can be set high only when ERS and EHV are high and PGM is low. A 0 to 1 transition of ESUS starts the sequence which sets DONE and places the flash in erase suspend. ESUS can be cleared only when DONE and EHV are high and PGM is low. A 1 to 0 transition of ESUS with EHV = 1 starts the sequence which clears DONE and returns the flash module to erase mode. The flash module cannot exit erase suspend and clear DONE while EHV is low. ESUS is cleared on reset.</p> <p>0 Erase sequence is not suspended 1 Erase sequence is suspended</p>
31 EHV	<p>Enable High Voltage. Enables the flash module for a high voltage program/erase operation. EHV is cleared on reset. EHV must be set after an interlock write to start a program/erase sequence. EHV may be set, initiating a program/erase, after an interlock write under one of the following conditions:</p> <ul style="list-style-type: none"> • Erase (ERS = 1, ESUS = 0). • Program (ERS = 0, ESUS = 0, PGM = 1, PSUS = 0). • Erase-suspended program (ERS = 1, ESUS = 1, PGM = 1, PSUS = 0). <p>If a program operation is to be initiated while an erase is suspended, the user must clear EHV while in erase suspend before setting PGM.</p> <p>In normal operation, a 1 to 0 transition of EHV with DONE high, PSUS and ESUS low terminates the current program/erase high-voltage operation.</p> <p>When an operation is aborted², there is a 1 to 0 transition of EHV with DONE low and the suspend bit for the current program/erase sequence low. An abort causes the value of PEG to be cleared, indicating a failed program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort.</p> <p>A suspended operation cannot be aborted. EHV may be written during suspend. EHV must be high for the flash to exit suspend. EHV may not be written after a suspend bit is set high and before DONE has transitioned high. EHV may not be set low after the current suspend bit is set low and before DONE has transitioned low.</p> <p>0 Flash is not enabled to perform a high-voltage operation. 1 Flash is enabled to perform a high-voltage operation.</p>

¹ In an erase-suspended program, programming flash locations in blocks that were being operated on in the erase may corrupt flash core data. This should be avoided due to reliability implications.

² Aborting a high voltage operation will leave flash core addresses in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

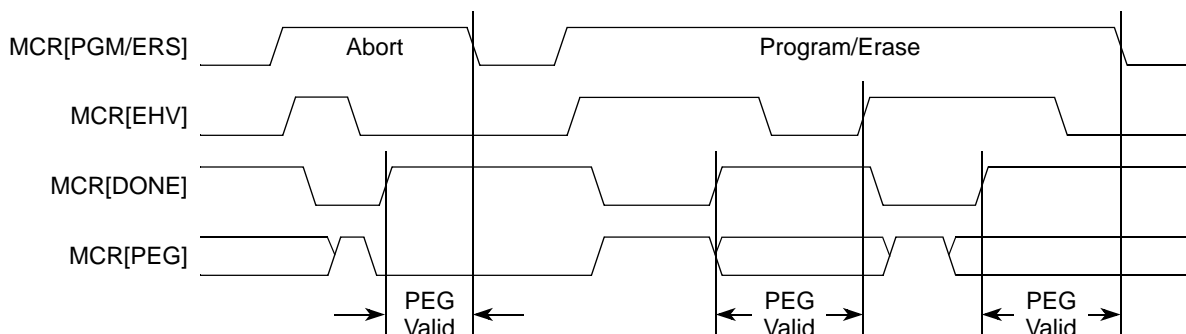


Figure 22-4. PEG Valid Times

22.4.2.1.1 MCR Simultaneous Register Writes

A number of MCR bits are protected against write when another bit or set of bits is in a specific state. These write locks are covered on a bit by bit basis in [Section 22.4.2.1, “Module Configuration Register \(MCR\).”](#) The write locks detailed in that section do not consider the effects of trying to write two or more bits simultaneously. The effects of writing bits simultaneously, which would put the flash module in an illegal state, are detailed here.

The flash does not allow the user to write bits simultaneously. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in [Table 22-4](#).

Table 22-4. MCR Bit Set/Clear Priority Levels

Priority Level	MCR Bits
1	STOP
2	ERS
3	PGM
4	EHV
5	ESUS, PSUS

If the user attempts to write two or more MCR bits simultaneously, only the bit with the highest priority level will be written. Setting two bits with the same priority level is prevented by existing write locks and will not put the flash in an illegal state.

For example, setting MCR[STOP] and MCR[PGM] simultaneously results in MCR[STOP] only being set. Attempting to clear MCR[EHV] while setting MCR[PSUS] will result in MCR[EHV] being cleared, but MCR[PSUS] will remain unaffected.

22.4.2.2 Low-/Mid-Address Space Block Locking Register

The low- and mid-address block locking register provides a means to protect blocks from being modified. These bits along with bits in the secondary locking register (SLL), determine if the block is locked from program or erase. An “OR” of LML and SLL determine the final lock status. See [Section 22.4.2.4, “Secondary Low-/Mid-Address Space Block Locking Register \(SLL\),”](#) for more information on SLL.

NOTE

If blocks are not present (due to configuration or total memory size), the LOCK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the shadow block) and register writes will have no effect.

Offset: FLASH_REGS_BASE + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	0	0	0	0	0	0	0	0	0	0	SLOCK	1	1	MLOCK[1:0]	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31			
R	1	1	1	1	1	1	LLOCK[9:0]												
W																			
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1			

Figure 22-5. Low-/Mid-Address Space Block Locking Register (LML)

Table 22-5. LML Field Descriptions

Field	Description
LME	Low- and Mid-Address Lock Enable. Enables the locking register fields (SLOCK, MLOCK, and LLOCK) to be set or cleared by register writes. This bit is a status bit only. It may not be written or cleared and the reset value is 0. To set this bit, write a password and if the password matches, the LME bit will be set to reflect the status of enabled. It is enabled until a reset operation occurs. For LME, the password 0xA1A1_1111 must be written to the LML. 0 Low- and mid-address locks are disabled, and cannot be modified 1 Low- and mid-address locks are enabled and can be written
bits 1–10	Reserved.
SLOCK	Shadow Lock. Locks the shadow row from programs and erases. The SLOCK bit is not writeable if a high-voltage operation is suspended. Upon reset, information from the shadow row is loaded into the SLOCK bit. The SLOCK bit may be written as a register. Reset will cause the bits to go back to their shadow row value. The default value of the SLOCK bit (assuming the corresponding shadow row bit is erased) would be locked. SLOCK is not writable unless LME is high. 0 Shadow row is available to receive program and erase pulses. 1 Shadow row is locked for program and erase.
bits 12–13	Reserved.

Table 22-5. LML Field Descriptions (continued)

Field	Description
MLOCK[1:0]	<p>Mid-Address Block Lock. A value of 1 in a bit of the lock register signifies that the corresponding block is locked for program and erase. A value of 0 in the lock register signifies that the corresponding block is available to receive program and erase pulses. Likewise, the lock register is not writable if a high-voltage operation is suspended.</p> <p>Upon reset, information from the shadow row is loaded into the block registers. The LOCK bits may be written as a register. Reset will cause the bits to go back to their shadow row value. The default value of the LOCK bits (assuming erased fuses) would be locked.</p> <p>If blocks are not present (due to configuration or total memory size), the LOCK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the shadow row), and register writes will have no effect.</p> <p>MLOCK is not writable unless LME is high.</p>
bits 16–21	Reserved.
LLOCK[9:0]	<p>Low-Address Block Lock. These bits have the same description and attributes as MLOCK. As an example of how the LLOCK bits are used, if a configuration has sixteen 16 KB blocks in the low-address space (MCR[LAS] = 0b011), the block residing at address FLASH_REGS_BASE + 0x0000, will correspond to LLOCK0. The next 16 KB block will correspond to LLOCK1, and so on up to LLOCK15.</p>

22.4.2.3 High-Address Space Block Locking Register (HBL)

The high-address space block locking register provides a means to protect blocks from being modified.

Offset: FLASH_REGS_BASE + 0x0008

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HBE	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	1	1	1	1	1	1	1	HBLOCK[7:0]							
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 22-6. High-Address Space Block Locking Register (HBL)

Table 22-6. HBL Field Descriptions

Field	Description
HBE	<p>High-Address Lock Enable. Enables the locking field (HLOCK) to be set or cleared by register writes. This bit is a status bit only, may not be written or cleared, and the reset value is 0. To set this bit, write a password and if the password matches, the HBE bit will be set to reflect the status of enabled. It is enabled until a reset operation occurs. For HBE, the password 0xB2B2_2222 must be written to HBL.</p> <p>0 High address locks are disabled, and cannot be modified</p> <p>1 High address locks are enabled to be written</p>

Table 22-6. HBL Field Descriptions (continued)

Field	Description
bits 1–23	Reserved.
HLOCK[7:0]	High-Address Space Block Lock. Has the same characteristics as MLOCK. See Section 22.4.2.2, “Low-/Mid-Address Space Block Locking Register,” for more information. The block numbering for high-address space starts with HLOCK[0] and continues until all blocks are accounted. HLOCK is not writable unless HBE is set. If blocks are not present (due to configuration or total memory size), the HLOCK bits will default to locked, and will not be writable.

22.4.2.4 Secondary Low-/Mid-Address Space Block Locking Register (SLL)

The SLL provides an alternative means to protect blocks from being modified. These bits along with bits in the lock register (LML), determine if the block is locked from program or erase. An OR of LML and SLL determine the final lock status. See [Section 22.4.2.2, “Low-/Mid-Address Space Block Locking Register,”](#) for more information on LML.

Offset: FLASH_REGS_BASE + 0x000C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	0	0	0	0	0	0	0	0	0	0	SSLOCK	1	1		
W																SMLOCK[1:0]
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	1	1	1	1	1	SLLOCK[9:0]									
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 22-7. Secondary Low-/Mid-Address Space Block Locking Register (SLL)

Table 22-7. SLL Field Descriptions

Field	Description
SLE	Secondary Low- and Mid-Address Lock Enable. Enables the secondary lock fields (SSLOCK, SMLOCK, and SLLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. To set this bit, write a password and if the password matches, the SLE bit will be set to reflect the status of enabled. It is enabled until a reset operation occurs. For SLE, the password 0xC3C3_3333 must be written to the SLL. 0 Secondary low- and mid-address locks are disabled and cannot be modified 1 Secondary low- and mid-address locks are enabled to be written
bits 1–10	Reserved.
SSLOCK	Secondary Shadow Lock. An alternative method that may be used to lock the shadow row from programs and erases. SSLOCK has the same description as SLOCK in Section 22.4.2.2, “Low-/Mid-Address Space Block Locking Register.” SSLOCK is not writable unless SLE is high.
bits 12–13	Reserved.

Table 22-7. SLL Field Descriptions (continued)

Field	Description
SMLOCK[1:0]	Secondary Mid-Address Block Lock. Alternative method that may be used to lock the mid-address space blocks from programs and erases. SMLOCK has the same description as MLOCK in section Section 22.4.2.2, “Low-/Mid-Address Space Block Locking Register.” SMLOCK is not writable unless SLE is set. If blocks are not present (due to configuration or total memory size), the SMLOCK bits will default to locked and will not be writable.
bits 16–21	Reserved.
SLLOCK[9:0]	Secondary Low-Address Block Lock. These bits are an alternative method that may be used to lock the low-address space blocks from programs and erases. SLLOCK has the same description as LLOCK in Section 22.4.2.2, “Low-/Mid-Address Space Block Locking Register.” SLLOCK is not writable unless SLE is high. If blocks are not present (due to configuration or total memory size), the SLLOCK bits will default to locked, and will not be writable.

22.4.2.5 Low-/Mid-Address Space Block Select Register (LMS)

The LMS provides a means to select blocks to be operated on during erase.

Offset: FLASH_REGS_BASE + 0x0010

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSEL	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	LSEL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-8. Low-/Mid-Address Space Block Select Register (LMS)

Table 22-8. LMS Field Descriptions

Field	Description
bits 0–13	Reserved.
MSEL	Mid-Address Space Block Select. Values in the selected register signify that a block is or is not selected for erase. The reset value for the select registers is 0, or unselected. The blocks must be selected (or unselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high-voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding SELECT bits will default to unselected, and will not be writable. The reset value will always be 0 and register writes will have no effect. A description of how blocks are numbered is detailed in Section 22.4.2.2, “Low-/Mid-Address Space Block Locking Register.” 0b00 Mid-address space blocks are not selected for erase 0b01 One mid-address space block is selected for erase 0b11 Two mid-address space blocks are selected for erase

Table 22-8. LMS Field Descriptions (continued)

Field	Description
16–21	Reserved.
22–31 LSEL[9:0]	Low-Address Space Block Select. Used to select blocks in the low-address space; these have the same description and attributes as the MSEL bits 0b00_0000_0000 Low-address space blocks are not selected for erase 0b00_0000_0001 One low-address space block is selected for erase 0b00_0000_0011 Two low-address space blocks are selected for erase 0b00_0000_0111 Three low-address space blocks are selected for erase ... 0b11_1111_1111 Ten low-address space blocks are selected for erase

22.4.2.6 High-Address Space Block Select Register (HBS)

The HBS provides a means to select blocks to be operated on.

Offset: FLASH_REGS_BASE + 0x0014

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	HBSSEL							
R	0	0	0	0	0	0	0	0								
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-9. High-Address Space Block Select Register (HBS)

Table 22-9. HBS Field Descriptions

Field	Description
bits 0–23	Reserved.
HBSSEL	High-Address Space Block Select. Has the same characteristics as MSEL. For more information see Section 22.4.2.5, “Low-/Mid-Address Space Block Select Register (LMS).” 0b0000_0000 High-address space blocks are not selected for erase 0b0000_0001 One high-address space block is selected for erase 0b0000_0011 Two high-address space blocks are selected for erase 0b0000_0111 Three high-address space blocks are selected for erase 0b0000_1111 Four high-address space blocks are selected for erase 0b0001_1111 Five high-address space blocks are selected for erase 0b0011_1111 Six high-address space blocks are selected for erase 0b0111_1111 Seven high-address space blocks are selected for erase 0b1111_1111 Eight high-address space blocks are selected for erase

22.4.2.7 Address Register (ADR)

The ADR provides the first failing address in the event of ECC event error (MCR[EER] set) and the address of a failure that may have occurred in a state machine operation (MCR[PEG] cleared). ECC event

errors take priority over state machine errors. This is especially valuable in the event of a RWW operation, where the read senses an ECC error and the state machine fails simultaneously. This address is always a doubleword address that selects 64 bits.

In normal operating mode, the ADR is not writable.

Offset: FLASH_REGS_BASE + 0x0018

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	ADDR[10:15]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ADDR[16:28]												0	0	0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-10. Address Register (ADR)

Table 22-10. ADR Field Descriptions

Field	Description
bits 0–9	Reserved.
ADDR	Doubleword address of first failing address in the event of an ECC error or the address of a failure occurring during state machine operation.
bits 29–31	Reserved.

22.4.2.8 Platform Flash Configuration Register for Port *n* (PFCRP_{*n*})

The PFLASH configuration register for port 0 (PFCRP0) is used to specify operation of port p0 of the PFLASH2P_H7Fb. This register also has two bits (ARB and PRI) to control arbitration between the p0/p1 ports.

The PFLASH configuration register for port 1 (PFCRP1) is used to specify operation of port p1 of the PFLASH2P_H7Fb

Flash Array and Control

Offset: FLASH_REGS_BASE + 0x001C (PFCRP0)
FLASH_REGS_BASE + 0x0020 (PFCRP1)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LBCFG				ARB ¹	PRI ¹	0	0	0	0	0	M4PFE	M3PFE	M2PFE	M1PFE	M0PFE
W																
Reset	0	0	— ²	— ²	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	APC				WWSC	RWSC			0	DPEN	0	IPEN	0	PFLIM		BFEN
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

¹ This bit is only available in PFCRP0. For PFCRP1, treat this bit as reserved.

² Reset value for port 0 is LBCFG = 0b0000, port 1 is LBCFG = 0b0011

Figure 22-11. PFLASH Configuration Register for Port *n* (PFCRP_{*n*})

Table 22-11. PFCRP0 and PFCRP1 Field Descriptions

Field	Description
LBCFG	Line Buffer Configuration. Controls the configuration of the four line buffers in the PFLASH controller. The buffers can be organized as a pool of available resources or with a fixed partition between instruction and data buffers. In all cases, when a buffer miss occurs, it is allocated to the least recently used buffer within the group and the just-fetched entry then marked as most recently used. If the flash access is for the next sequential line, the buffer is not marked as most recently used until the given address produces a buffer hit. For PFCRP0, this field is set to 0b0000 by hardware reset. For PFCRP1, this field is set to 0b0011 by hardware reset. xx00 All four buffers are available for any flash access, i.e., there is no partitioning of the buffers based on the access type xx01 Reserved xx10 The buffers are partitioned into two groups: buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. xx11 The buffers are partitioned into two groups: buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses.
ARB	Arbitration Mode. This field controls which arbitration mode is used. In both fixed priority or round-robin modes, write requests are prioritized higher than read requests, and read requests are prioritized higher than speculative prefetch requests whenever both ports issue concurrent requests. This bit is set to 1 by hardware reset. 0 Fixed-priority arbitration is used; the port specified in PRI has highest fixed priority 1 Round-robin arbitration is used Note: This bit is only available in PFCRP0. For PFCRP1, treat this bit as reserved.
PRI	Fixed Priority. Controls which port has highest fixed priority when fixed priority arbitration is selected. This field has no effect when operating in round-robin mode. This bit is cleared by hardware reset. 0 Port p0 is given highest fixed priority. 1 Port p1 is given highest fixed priority. Note: This bit is only available in PFCRP0. For PFCRP1, treat this bit as reserved.
bits 6–10	Reserved.

Table 22-11. PFCRP0 and PFCRP1 Field Descriptions (continued)

Field	Description
MnPFE n=4:0	<p>Master <i>n</i> Prefetch Enable. Used to control whether prefetching may be triggered based on the AHB hmaster attribute. For example, MOPFE will enable prefetching for accesses where hmaster[3:0] = 0b0000. Likewise, M4PFE will enable prefetching only when hmaster[3:0] == 0b0100. Note that hmaster[3] is ignored when determining which MnPFE to use for a given access. These bits are cleared by hardware reset.</p> <p>0 No prefetching may be triggered by this master 1 Prefetching may be triggered by this master</p> <p>Note: These bits refer to the master ID, not the master port number. Therefore, n=0 (e200z1), n=1 (e200z0), n=2 (eDMA), n=3 (FlexRay), and n=4 (EBI).</p>
APC	<p>Address Pipelining Control. Used to control the number of cycles between pipelined access requests. This field must be set to a value corresponding to the operating frequency of the PFLASH. The required settings are documented in the SoC specification. Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to 0b111 by hardware reset.</p> <p>000 Accesses may be pipelined back-to-back 001 Access requests require one additional hold cycle 010 Access requests require two additional hold cycles ... 110 Access requests require six additional hold cycles 111 No address pipelining</p> <p>Note: The settings for APC and RWSC should be the same.</p>
WWSC	<p>Write Wait State Control. Used to control the number of wait states to be added to the best case flash array access time for writes. This field must be set to a value corresponding to the operating frequency of the PFLASH. Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to 0b11 by hardware reset.</p> <p>00 No additional wait-states are added 01 One additional wait-state is added 10 Two additional wait-states are added 11 Three additional wait-states are added</p>
RWSC	<p>Read Wait State Control. Used to control the number of wait states to be added to the best case flash array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFLASH and the actual read access time of the PFLASH. This field is set to 0b111 by hardware reset.</p> <p>000 No additional wait states are added 001 One additional wait state is added ... 111 Seven additional wait states are added</p> <p>Note: The settings for APC and RWSC should be the same.</p>
bit 24	Reserved.
DPFEN	<p>Data Prefetch Enable. Enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset.</p> <p>0 No prefetching is triggered by a data read access 1 Prefetching may be triggered by any data read access</p>
bit 26	Reserved.
IPFEN	<p>Instruction Prefetch Enable. Enables or disables prefetching initiated by an instruction read access. This field is cleared by hardware reset.</p> <p>0 No prefetching is triggered by an instruction read access 1 Prefetching may be triggered by any instruction read access</p>
bit 28	Reserved.

Table 22-11. PFCRP0 and PFCRP1 Field Descriptions (continued)

Field	Description
PFLIM	<p>PFLASH Prefetch Limit. Controls the prefetch algorithm used by the PFLASH prefetch controller. This field defines a limit on the maximum number of sequential prefetches which will be attempted between buffer misses. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is cleared by hardware reset.</p> <p>00 No prefetching or buffering is performed</p> <p>01 The referenced line is prefetched on a buffer miss, i.e., prefetch on miss</p> <p>1x the referenced line is prefetched on a buffer miss, or the next sequential line is prefetched on a buffer hit (if not already present), i.e., prefetch on miss or hit</p>
BFEN	<p>PFLASH Line Read Buffers Enable. Enables or disables line read buffer hits. It is also used to invalidate the buffers. This bit is cleared by hardware reset.</p> <p>0 The line read buffers are disabled from satisfying read requests, and all buffer valid bits are cleared</p> <p>1 The line read buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled</p>

22.5 Functional Description

22.5.1 Flash User Mode

In user mode the flash module can be read and written (register writes and interlock writes), programmed or erased. The following sub-sections define all actions that can be performed in user mode.

22.5.2 Flash Read and Write

The default state of the flash module is read. The main and shadow address space can be read only in the read state. The module configuration register (MCR) is always available for read. The flash module enters the read state on reset. The flash module is in the read state under four sets of conditions:

- The read state is active when STOP=0 in the MCR (user mode read).
- The read state is active when PGM=1 or ERS=1 in the MCR and high-voltage operation is ongoing (read while write).

NOTE

Reads done to the partition(s) being operated on (either erased or programmed) will result in an error and the RWE bit in the MCR will be set.

- The read state is active when PGM=1 and PSUS=1 in the MCR (program suspend).
- The read state is active when ERS=1 and ESUS=1 and PGM=0 in the MCR (erase suspend).

NOTE

FC reads are done through the BIU. In many cases the BIU will do page buffering to allow sequential reads to be done with higher performance. This can create a data coherency issue that must be handled with software. Data coherency can be an issue after a program, erase, or shadow row operations.

In flash user mode, registers can be written. Array can be written to do interlock writes.

Reads attempted to invalid locations will result in indeterminate data. Invalid locations occur when addressing is done to blocks that do not exist in non 2^n array sizes.

Interlock writes attempted to invalid locations (due to blocks that do not exist in non 2^n array sizes), will result in an interlock occurring, but attempts to program or erase these blocks will not occur since they are forced to be locked.

22.5.3 Read While Write (RWW)

The flash core is divided into partitions. Partitions are always comprised of two or more blocks. Partitions are used to determine read-while-write (RWW) groupings. While a write (program or erase) is being done within a given partition, a read can be simultaneously executed to any other partition. Partitions are listed in [Table 22-1](#). Each partition in high address space comprises of two 128KB blocks. The shadow block has unique RWW restrictions described in [Section 22.5.6, “Flash Shadow Block.”](#)

The FC is also divided into blocks to implement independent erase or program protection. The shadow block exists outside the normal address space and is programmed, erased, and read independently of the other blocks. The shadow block is included to support systems that require NVM for security or system initialization information.

A software mechanism is provided to independently lock or unlock each block in high-, mid-, and low-address space against program and erase. Two hardware locks are also provided to enable/disable the FC for program/erase. See [Section 22.5.4.1, “Software Locking,”](#) for more information.

22.5.4 Flash Programming

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1. Addresses in locked/disabled blocks cannot be programmed. The user can program the values in any or all of four words within a page in a single program sequence. Word addresses are selected using bits 3:2 of the page-bound word.

Whenever a program operation occurs, ECC bits are programmed. ECC is handled on a 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed because ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word will probably result in an operation failure. It is recommended that all programming operations be from 64 bits to 128 bits, and be 64-bit aligned. The programming operation should completely fill selected ECC segments within the page.

The program operation consists of the following sequence of events:

1. Change the value in the MCR[PGM] bit from a 0 to a 1.

NOTE

Ensure the block that contains the address to be programmed is unlocked. See [Section 22.4.2.2, “Low-/Mid-Address Space Block Locking Register,”](#) [Section 22.4.2.3, “High-Address Space Block Locking Register \(HBL\),”](#) and [Section 22.4.2.4, “Secondary Low-/Mid-Address Space Block Locking Register \(SLL\),”](#) for more information.

2. Write the first address to be programmed in the flash module with the program data. This write is referred to as a program data interlock write. An interlock write may be either be an aligned word or doubleword.
3. If more than one word or doubleword is to be programmed, write each additional address in the page with data to be programmed. This is referred to as a program data write. All unwritten data words default to 0xFFFF_FFFF.
4. Write a logic 1 to the MCR[EHV] bit to start the internal program sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the MCR[PGM] bit to terminate the program sequence.

The program sequence is presented graphically in [Figure 22-12](#). The program suspend operation detailed in [Figure 22-12](#) is discussed in [Section 22.5.4.1.1](#), “Flash Program Suspend/Resume.”

The first write after a program is initiated determines the page address to be programmed. Program may be initiated with the 0 to 1 transition of the MCR[PGM] bit or by clearing the MCR[EHV] bit at the end of a previous program. This first write is referred to as an interlock write. If the program is not an erase-suspended program, the interlock write determines if the shadow or normal array space will be programmed and causes MCR[PEAS] to be set/cleared.

In the case of an erase-suspended program, the value in MCR[PEAS], is retained from the erase.

An interlock write must be performed before setting MCR[EHV]. The user may terminate a program sequence by clearing MCR[PGM] prior to setting MCR[EHV].

If multiple writes are done to the same location the data for the last write is used in programming.

While MCR[DONE] is low, MCR[EHV] is high, and MCR[PSUS] is low, the user may clear MCR[EHV], resulting in a program abort. A program abort forces the module to step 8 of the program sequence. An aborted program will result in MCR[PEG] being set low, indicating a failed operation. The data space being operated on before the abort will contain indeterminate data. The user may not abort a program sequence while in program suspend.

CAUTION

Aborting a program operation will leave the flash core addresses being programmed in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

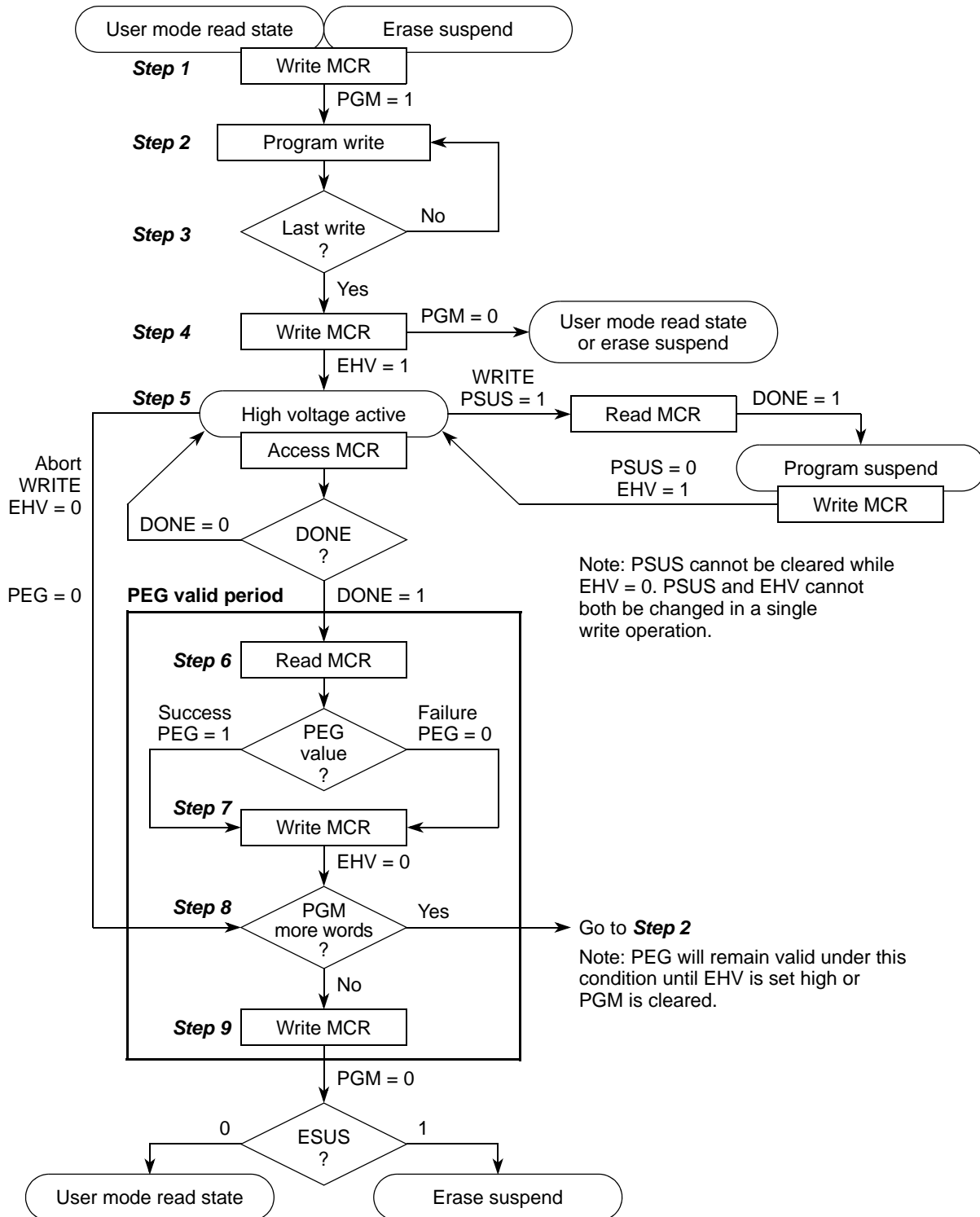


Figure 22-12. Program Sequence

22.5.4.1 Software Locking

A software mechanism is provided to independently lock/unlock each high-, mid-, and low-address space against program and erase.

Software locking is done through the LML (low-/mid-address space block locking register), SLL (secondary low-/mid-address space block locking register), or HBL (high-address space block locking register). These can be written through register writes and read through register reads.

When the program/erase operations are enabled through hardware, software locks are enforced through doing register writes.

22.5.4.1.1 Flash Program Suspend/Resume

The program sequence may be suspended to allow read access to the flash core. It is not possible to erase or program during a program suspend. Interlock writes should not be attempted during program suspend.

A program suspend can be initiated by changing the value of the MCR[PSUS] bit from a 0 to a 1. MCR[PSUS] can be set high at any time when MCR[PGM] and MCR[EHV] are high. A 0 to 1 transition of MCR[PSUS] causes the flash module to start the sequence to enter program suspend, which is a read state. The module is not suspended until MCR[DONE] = 1. At this time flash core reads may be attempted. After it is suspended, the flash core may be read only. Reads to the blocks being programmed/erased return indeterminate data.

The program sequence is resumed by writing a logic 0 to MCR[PSUS]. MCR[EHV] must be set to a 1 before clearing MCR[PSUS] to resume operation. When the operation resumes, the flash module continues the program sequence from one of a set of predefined points. This may extend the time required for the program operation.

22.5.5 Flash Erase

Erase changes the value stored in all bits of the selected blocks to logic 1. Locked or disabled blocks cannot be erased. If multiple blocks are selected for erase during an erase sequence, the blocks are erased sequentially starting with the lowest numbered block and terminating with the highest. Aborting an erase operation will leave the flash core blocks being erased in an indeterminate data state. This can be recovered by executing an erase on the affected blocks.

The erase sequence consists of the following sequence of events:

1. Change the value in the MCR[ERS] bit from 0 to a 1.
2. Select the block, or blocks, to be erased by writing 1s to the appropriate registers in LMS or HBS. If the shadow row is to be erased, this step may be skipped, and LMS and HBS are ignored. For shadow row erase, see section [Section 22.5.6, “Flash Shadow Block,”](#) for more information.

NOTE

Lock and select are independent. If a block is selected and locked, no erase will occur. See [Section 22.4.2.2, “Low-/Mid-Address Space Block Locking Register,”](#) [Section 22.4.2.3, “High-Address Space Block Locking Register \(HBL\),”](#) and [Section 22.4.2.4, “Secondary Low-/Mid-Address Space Block Locking Register \(SLL\),”](#) for more information.

3. Write to any address in flash. This is referred to as an erase interlock write.
4. Write a logic 1 to the MCR[EHV] bit to start an internal erase sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more blocks are to be erased, return to step 2.
9. Write a logic 0 to the MCR[ERS] bit to terminate the erase.

The erase sequence is presented graphically in [Figure 22-13](#). The erase suspend operation detailed in [Figure 22-13](#) is discussed in section [Section 22.5.5.1, “Flash Erase Suspend/Resume.”](#)

After setting MCR[ERS], one write (referred to as an interlock write) must be performed before MCR[EHV] can be set to a 1. Data words written during erase sequence interlock writes are ignored. The user may terminate the erase sequence by clearing MCR[ERS] before setting MCR[EHV].

An erase operation may be aborted by clearing MCR[EHV] assuming MCR[DONE] is low, MCR[EHV] is high, and MCR[ESUS] is low. An erase abort forces the module to step 8 of the erase sequence. An aborted erase will result in MCR[PEG] being set low, indicating a failed operation. The blocks being operated on before the abort contain indeterminate data. The user may not abort an erase sequence while in erase suspend.

CAUTION

Aborting an erase operation will leave the flash core blocks being erased in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

22.5.5.1 Flash Erase Suspend/Resume

The erase sequence may be suspended to allow read access to the flash core. The erase sequence may also be suspended to program (erase-suspended program) the flash core. A program started during erase suspend can be suspended. One erase suspend and one program suspend are allowed at a time during an operation. It is not possible to erase during an erase suspend, or program during a program suspend. During suspend, all reads to flash core locations targeted for program and blocks targeted for erase return indeterminate data. Programming locations in blocks targeted for erase during erase-suspended program may result in corrupted data.

An erase suspend operation is initiated by setting the MCR[ESUS] bit. MCR[ESUS] can be set to a 1 at any time when MCR[ERS] and MCR[EHV] are high and MCR[PGM] is low. A 0 to 1 transition of MCR[ESUS] causes the flash module to start the sequence which places it in erase suspend. The user must

wait until $MCR[DONE] = 1$ before the module is suspended and further actions are attempted. After it is suspended, the array may be read or a program sequence may be initiated (erase-suspended program). Before initiating a program sequence the user must first clear $MCR[EHV]$. If a program sequence is initiated, the value of the $MCR[PEAS]$ is not reset. These values are fixed at the time of the first interlock of the erase. Flash core reads from the blocks being erased while $MCR[ESUS] = 1$ return indeterminate data.

The erase operation is resumed by clearing the $MCR[ESUS]$ bit. The flash continues the erase sequence from one of a set of predefined points. This can extend the time required for the erase operation.

CAUTION

In an erase-suspended program, programming flash locations in blocks which were being operated on in the erase may corrupt flash core data.

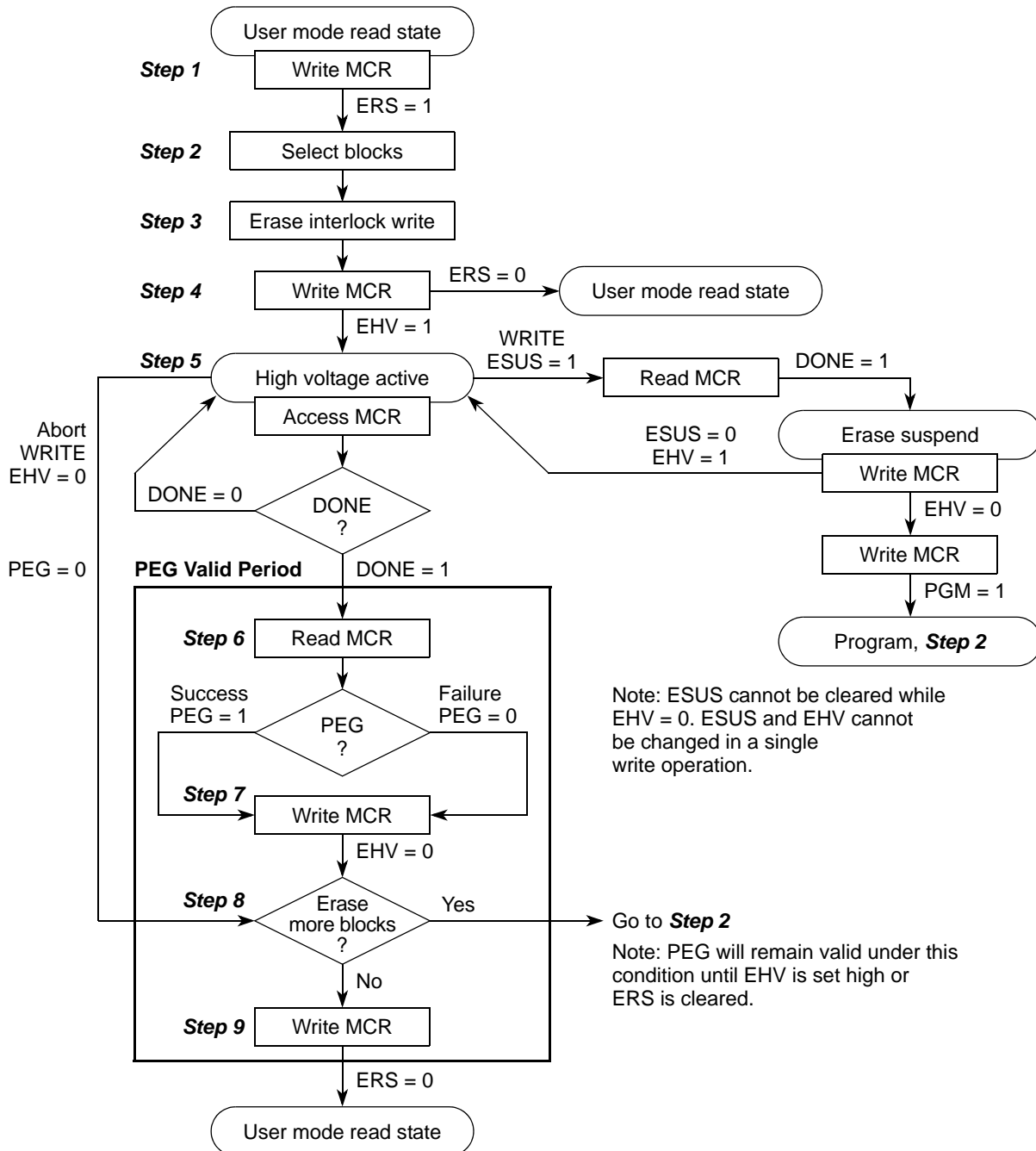


Figure 22-13. Erase Sequence

22.5.6 Flash Shadow Block

The flash shadow block is a memory-mapped block in the flash memory map. Program and erase of the shadow block are enabled when $MCR[PEAS] = 1$ only. After the user has begun an erase operation on the shadow block, the operation cannot be suspended to program the main address space and vice-versa. The user must terminate the shadow erase operation to program or erase the main address space.

NOTE

If an erase of user space is requested, and a suspend is done with attempts to erase suspend program shadow space, this attempted program will be directed to user space as dictated by the state of MCR[PEAS]. Likewise an attempted erase suspended program of user space, while the shadow space is being erased, will be directed to shadow space as dictated by the state of MCR[PEAS].

The shadow block cannot use the RWW feature. After an operation is started in the shadow block, a read cannot be done to the shadow block, or any other block. Likewise, after an operation is started in a block in low-/mid-/high-address space, a read cannot be done in the shadow block.

The shadow block contains information about how the lock registers are reset. The first and second words can be used for reset configuration words. All other words can be used for user-defined functions or other configuration words.

The shadow block may be locked/unlocked against program or erase by using the LML or SLL discussed in [Section 22.4.2, “Register Descriptions.”](#)

Programming the shadow row has similar restrictions to programming the array in terms of how ECC is calculated. See [Section 22.5.4, “Flash Programming,”](#) for more information. Only one program is allowed per 64 bit ECC segment between erases. Erase of the shadow row is done similarly as an array erase. See section [Section 22.5.5, “Flash Erase,”](#) for more information.

22.5.7 Flash Stop Mode

Stop mode is entered by setting the STOP bit in the MCR. The STOP bit cannot be written when PGM=1 or ERS=1 in the MCR. In stop mode all DC current sources in the flash module are disabled. Stop mode is exited by clearing the STOP bit.

NOTE

Exiting the stop mode requires a recovery time of t_{SRCV} .

22.5.8 Flash Reset

A reset is the highest priority operation for the flash and terminates all other operations.

The flash uses reset to initialize register and status bits to their default reset values. If the flash is executing a program or erase operation and a reset is issued, the operation will be aborted and the flash will disable the high voltage logic without damage to the high-voltage circuits. Reset aborts all operations and forces the flash into user mode ready to receive accesses.

After reset is negated, register accesses can be performed, although it should be noted that registers that require updating from shadow information, or other inputs, cannot read updated until flash exits reset.

22.6 DMA Requests

The flash has no DMA requests.

22.7 Interrupt Requests

The flash has no interrupt requests.

Chapter 23

Deserial Serial Peripheral Interface (DSPI)

23.1 Introduction

The deserial serial peripheral interface (DSPI) block provides a synchronous serial interface for communication between the MPC5510 and external devices. The DSPI supports pin-count reduction through serialization and deserialization of eMIOS channels and memory-mapped registers. The channels and register content are transmitted using a SPI-like protocol. There are up to four identical DSPI blocks: DSPI_A, DSPI_B, DSPI_C, and DSPI_D; use [Table 1-6](#) in [Chapter 1, “Overview”](#) to determine which DSPI modules are available on your chosen device.

The DSPIs have three configurations:

- Serial peripheral interface (SPI) configuration where the DSPI operates as a SPI with support for queues.
- Deserial serial interface (DSI) configuration where the DSPI serializes eMIOS200 output channels and deserializes the received data by placing it on the eMIOS200 input channels.
- Combined serial interface (CSI) configuration where the DSPI operates in both SPI and DSI configurations interleaving DSI frames with SPI frames, giving priority to SPI frames.

NOTE

The DSPI_D deserialized outputs cannot be used as eMIOS200 input channel signals, but can be read from a memory mapped register.

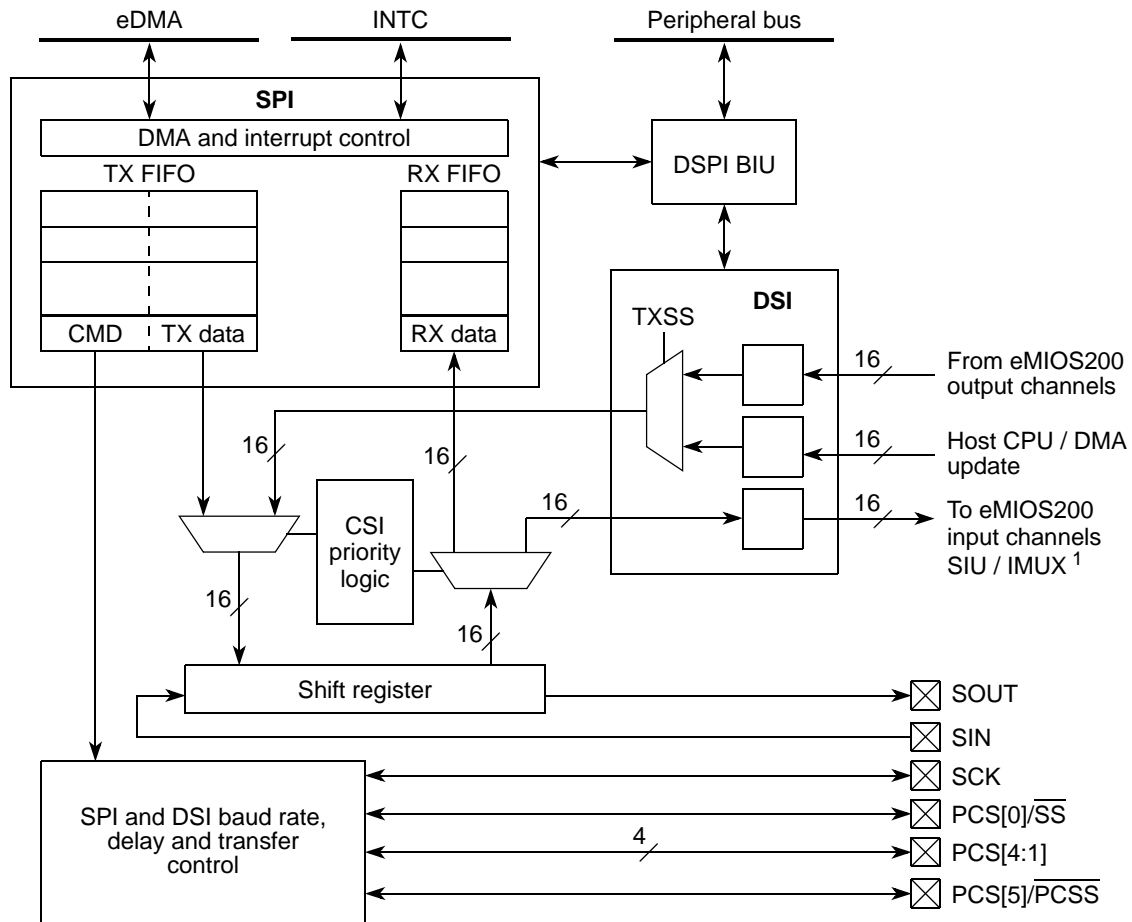
For queued operations, the SPI queues reside in system memory external to the DSPI. Data transfers between the memory and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software.

23.1.1 Block Diagram

[Figure 23-1](#) is a simplified block diagram of the DSPI that illustrates the functionality and interdependence of major blocks.

NOTE

Not all chip selects are available in the 144-pin package. For example, DSPI_B3 and DSPI_B4 are not available. Also, owing to the multiplexing of functions on the various pins, some chip selects may not be available if other functions are configured to use these pins. Please pay careful attention to your system's pin allocation requirements.



¹ Parallel outputs to the eMIOS200 are not supported by SDPI_D.

Figure 23-1. DSPI Block Diagram

23.1.2 Features

The DSPI supports these SPI features:

- Full-duplex, synchronous transfers
- Master and slave mode
- Buffered operation with separate four-entry TX and RX FIFOs
- Visibility into the TX and RX FIFOs for ease of debugging
- FIFO bypass mode for low-latency updates to SPI queues
- Programmable SPI transfer attributes on a per-frame basis:
 - Eight clock and transfer attribute registers
 - Serial clock with programmable polarity and phase
 - Programmable delays:
 - PCS to SCK delay

- SCK to PCS delay
- Delay between frames
- Programmable serial frame size of 4 to 16 bits, expandable with software control
- Continuously held chip-select capability
- Six peripheral chip selects, expandable to 64 with external demultiplexer
- Deglitching support for up to 32 peripheral chip selects with external demultiplexer
- DMA support for SPI queues residing in RAM or flash:
 - TX FIFO is not full
 - RX FIFO is not empty
- Six interrupt conditions:
 - End of queue reached
 - TX FIFO is not full
 - Transfer of current frame is complete
 - RX FIFO is not empty
 - FIFO underrun (slave-only and SPI mode, the slave is asked to transfer data when the TX FIFO is empty)
- FIFO overrun (serial frame received while RX FIFO is full)
- Modified transfer formats for communication with slower peripheral devices
- Continuous serial communications clock (SCK)
- Power-saving architectural features

The DSPIs also support these features unique to the DSI and CSI configurations:

- Sources of the serialized data:
 - eMIOS output channels
 - Memory-mapped register in the DSPI
- Destinations for the deserialized data:
 - eMIOS input channels (not supported by DSPI_D)
 - Memory-mapped register in the DSPI
- Transfer initiation conditions:
 - Continuous
 - Change in data
- Pin serialization/deserialization with interleaved SPI frames for control and diagnostics
- Debug and STOP (with STOP ack) are supported.
- The reset value of the MDIS register bit is 1 and thus the DSPI is disabled by default after reset.

23.1.3 Modes of Operation

The DSPI has four modes of operation that can be divided into two categories: block-specific modes and an MCU-specific mode. Master mode, slave mode, and module disable mode are the block-specific modes, and debug mode is the MCU-specific mode.

The block-specific modes are entered by host software writing to a register bit. The MCU-specific mode is selected by a signal external to the DSPI. The MCU-specific mode is a mode that the MCU may enter in parallel to the DSPI being in one of its block-specific modes.

23.1.3.1 Master Mode

Master mode allows the DSPI to initiate and control serial communication. In this mode the SCK, PCS, and SOUT signals are controlled by the DSPI and configured as outputs.

23.1.3.2 Slave Mode

Slave mode allows the DSPI to communicate with SPI/DSI bus masters. In this mode the DSPI responds to externally-controlled serial transfers. The DSPI cannot initiate serial transfers in slave mode.

23.1.3.3 Module Disable Mode

The module disable mode is used for MCU power management. The clock to the non-memory-mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI_MCR is set.

23.1.3.4 Debug Mode

Debug mode is used for system development and debugging. If the device enters debug mode while the DSPI_MCR[FRZ] bit is set, the DSPI halts operation on the next frame boundary. If the device enters debug mode while the FRZ bit is negated, the DSPI behavior is unaffected and remains dictated by the block-specific mode and configuration of the DSPI.

23.2 External Signal Description

Refer to [Table 2-1](#) and [Section 2.7, “Detailed External Signal Descriptions,”](#) for detailed signal descriptions.

23.3 Memory Map and Registers

This section provides a detailed description of all DSPI registers.

23.3.1 Module Memory Map

The DSPI memory map is shown in [Table 23-1](#) (the memory map is the same for each individual DSPI module). The address of each register is given as an offset to the DSPI base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

Table 23-1. DSPI Memory Map

Offset from DSPI_BASE (DSPI_A= 0xFFFF9_0000 DSPI_B= 0xFFFF9_4000 DSPI_C= 0xFFFF9_8000 DSPI_D= 0xFFFF9_C000)	Register	Access	Reset Value	Section/Page
0x0000	DSPI_MCR — DSPI module configuration register	R/W	0x0000_4001	23.3.2.1/23-5
0x0004	Reserved			
0x0008	DSPI_TCR — DSPI transfer count register	R/W	0x0000_0000	23.3.2.2/23-8
0x000C–0x0028	DSPI_CTAR0 — DSPI clock and transfer attributes register 0 – DSPI_CTAR7 — DSPI clock and transfer attributes register 7	R/W	0x8400_0000	23.3.2.3/23-9
0x002C	DSPI_SR — DSPI status register	R	0x0200_0000	23.3.2.4/23-17
0x0030	DSPI_RSER — DSPI DMA/interrupt request select and enable register	R/W	0x0000_0000	23.3.2.5/23-19
FIFO Registers				
0x0034	DSPI_PUSHR — DSPI push TX FIFO register	R/W	0x0000_0000	23.3.2.6/23-21
0x0038	DSPI_POPOP — DSPI pop RX FIFO register	R	0x0000_0000	23.3.2.7/23-23
0x003C–0x0048	DSPI_TXFR0 — DSPI transmit FIFO register 0 – DSPI_TXFR3 — DSPI transmit FIFO register 3	R	0x0000_0000	23.3.2.8/23-23
0x004C–0x0078	Reserved			
0x007C–0x0088	DSPI_RXFR0 — DSPI receive FIFO register 0 – DSPI_RXFR3 — DSPI receive FIFO register 3	R	0x0000_0000	23.3.2.9/23-24
0x008C–0x00B8	Reserved			
DSI Registers				
0x00BC	DSPI_DSICR — DSPI DSI configuration register	R/W	0x0000_0000	23.3.2.10/23-25
0x00C0	DSPI_SDR — DSPI DSI serialization data register	R	0x0000_0000	23.3.2.11/23-26
0x00C4	DSPI_ASADR — DSPI DSI alternate serialization data register	R/W	0x0000_0000	23.3.2.12/23-27
0x00C8	DSPI_COMPR — DSPI DSI transmit comparison register	R	0x0000_0000	23.3.2.13/23-28
0x00CC	DSPI_DDR — DSPI DSI deserialization data register	R	0x0000_0000	23.3.2.14/23-28

23.3.2 Register Descriptions

This section lists the DSPI registers in address order and describes the registers and their bit fields.

23.3.2.1 DSPI Module Configuration Register (DSPI_MCR)

The DSPI_MCR contains bits which configure various attributes associated with DSPI operation. The HALT and MDIS bits can be changed at any time but will take effect on the next frame boundary only.

NOTE

Only the HALT and MDIS bits in the DSPI_MCR may be changed while the DSPI is running.

Offset: DSPI_BASE + 0x0000

Access: Read/Write

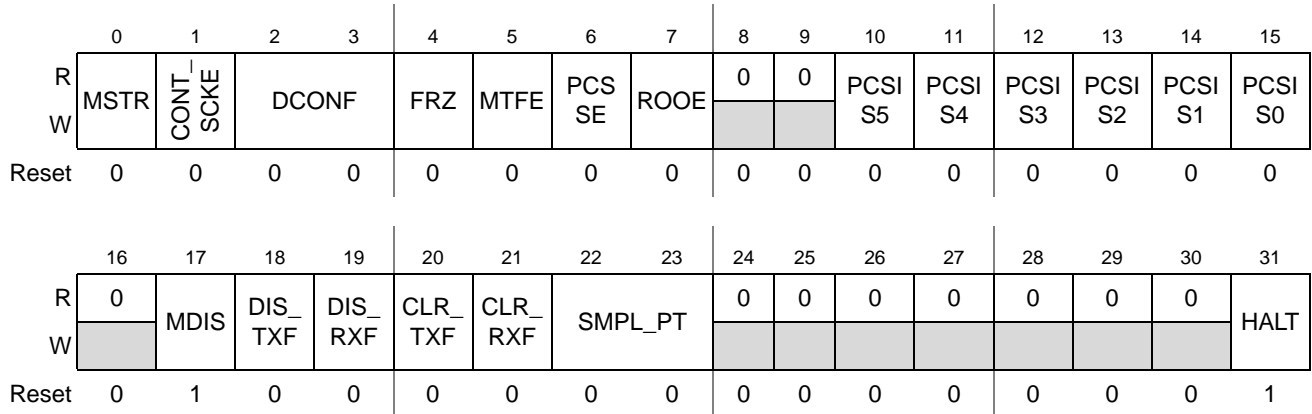


Figure 23-2. DSPI Module Configuration Register (DSPI_MCR)

Table 23-2. DSPI_MCR Field Descriptions

Field	Description										
MSTR	Master/Slave Mode Select. Configures the DSPI for either master mode or slave mode. 0 DSPI is in slave mode 1 DSPI is in master mode										
CONT_SCKE	Continuous SCK Enable. Enables the serial communication clock (SCK) to run continuously. See Section 23.4.9, “Continuous Serial Communications Clock,” for details. 0 Continuous SCK disabled 1 Continuous SCK enabled										
DCONF	DSPI Configuration. Selects between the three different configurations of the DSPI. The table below lists the DCONF values for the various configurations. <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>DCONF</th> <th>Configuration</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>SPI</td> </tr> <tr> <td>01</td> <td>DSI</td> </tr> <tr> <td>10</td> <td>CSI</td> </tr> <tr> <td>11</td> <td>Reserved</td> </tr> </tbody> </table>	DCONF	Configuration	00	SPI	01	DSI	10	CSI	11	Reserved
DCONF	Configuration										
00	SPI										
01	DSI										
10	CSI										
11	Reserved										
FRZ	Freeze. Enables the DSPI transfers to be stopped on the next frame boundary when the device enters debug mode. 0 Do not halt serial transfers 1 Halt serial transfers										
MTFE	Modified Timing Format Enable. Enables a modified transfer format to be used. See Section 23.4.8.4, “Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 1),” for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled										

Table 23-2. DSPI_MCR Field Descriptions (continued)

Field	Description
PCSSE	Peripheral Chip Select Strobe Enable. Enables the PCSx5/ $\overline{\text{PCSS}}$ to operate as an PCS strobe output signal. See Section 23.4.7.5, “Peripheral Chip Select Strobe Enable (PCSS)” , for more information. 0 PCSx5/ $\overline{\text{PCSS}}$ is used as the peripheral chip select 5 signal 1 PCSx5/ $\overline{\text{PCSS}}$ is used as an active-low PCS strobe signal
ROOE	Receive FIFO Overflow Overwrite Enable. Enables an RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer that generated the overflow is ignored or shifted in to the shift register. If the ROOE bit is asserted, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored. See Section 23.4.11.6, “Receive FIFO Overflow Flag (RFOF)” , for more information. 0 Incoming data is ignored 1 Incoming data is shifted in to the shift register
bits 8–9	Reserved.
PCISn	Peripheral Chip Select Inactive State. Determines the inactive state of the PCSn signal. PCS0/ $\overline{\text{SS}}$ must be configured as inactive high for slave mode operation. 0 The inactive state of PCSn is low 1 The inactive state of PCSn is high
bit 16	Reserved.
MDIS	Module Disable. Allows the clock to be stopped to the non-memory-mapped logic in the DSPI effectively putting the DSPI in a software controlled power-saving state. See Section 23.4.12, “Power Saving Features” , for more information. The reset value of the MDIS bit is 1. 0 Enable DSPI clocks 1 Allow external logic to disable DSPI clocks
DIS_TXF	Disable Transmit FIFO. Provides a mechanism to disable the TX FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See Section 23.4.3.3, “FIFO Disable Operation” , for details. 0 TX FIFO is enabled 1 TX FIFO is disabled
DIS_RXF	Disable Receive FIFO. Provides a mechanism to disable the RX FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See Section 23.4.3.3, “FIFO Disable Operation” , for details. 0 RX FIFO is enabled 1 RX FIFO is disabled
CLR_TXF	Clear TX FIFO. Flushes the TX FIFO. Writing a 1 to CLR_TXF clears the TX FIFO counter. The CLR_TXF bit is always read as zero. 0 Do not clear the TX FIFO counter 1 Clear the TX FIFO counter
CLR_RXF	Clear RX FIFO. Flushes the RX FIFO. Writing a 1 to CLR_RXF clears the RX counter. The CLR_RXF bit is always read as zero. 0 Do not clear the RX FIFO counter 1 Clear the RX FIFO counter

Table 23-2. DSPI_MCR Field Descriptions (continued)

Field	Description										
SMPL_PT	<p>Sample Point. Allows the host software to select when the DSPI master samples SIN in modified transfer format. Figure 23-32 shows where the master can sample the SIN pin. The table below lists the various delayed sample points.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>SMPL_PT</th> <th>Number of system clock cycles between odd-numbered edge of SCKx and sampling of SINx.</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>0</td> </tr> <tr> <td>01</td> <td>1</td> </tr> <tr> <td>10</td> <td>2</td> </tr> <tr> <td>11</td> <td>Reserved</td> </tr> </tbody> </table>	SMPL_PT	Number of system clock cycles between odd-numbered edge of SCKx and sampling of SINx.	00	0	01	1	10	2	11	Reserved
SMPL_PT	Number of system clock cycles between odd-numbered edge of SCKx and sampling of SINx.										
00	0										
01	1										
10	2										
11	Reserved										
bits 24–30	Reserved.										
HALT	<p>Halt. Provides a mechanism for software to start and stop DSPI transfers. See Section 23.4.2, “Start and Stop of DSPI Transfers,” for details on the operation of this bit.</p> <p>0 Start transfers 1 Stop transfers</p>										

23.3.2.2 DSPI Transfer Count Register (DSPI_TCR)

The DSPI_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management.

NOTE

The user must not write to the DSPI_TCR while the DSPI is running.

Offset: DSPI_BASE + 0x0008

Access: Read/Write

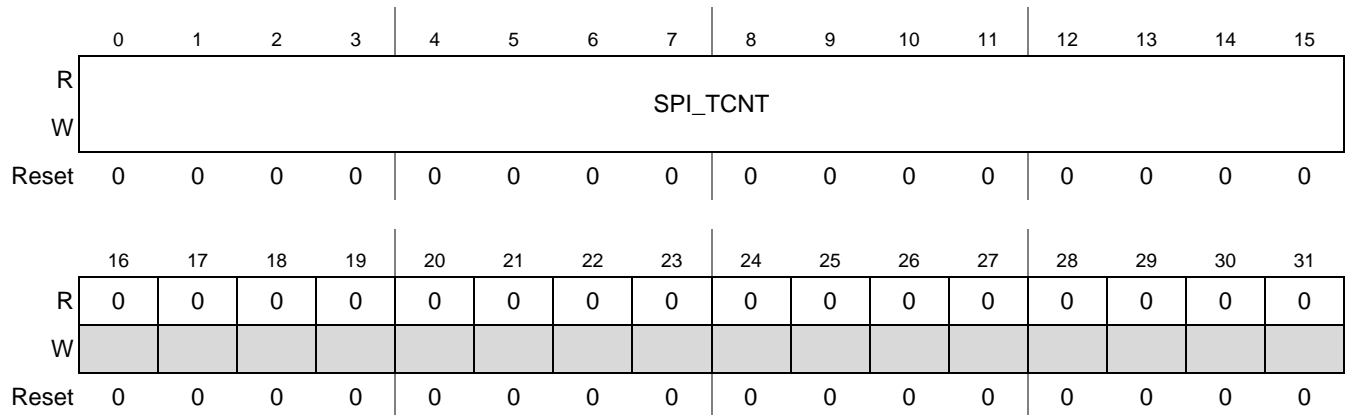


Figure 23-3. DSPI Transfer Count Register (DSPI_TCR)

Table 23-3. DSPI_TCR Field Descriptions

Field	Description
SPI_TCNT	SPI Transfer Counter. Counts the number of SPI transfers the DSPI makes. The SPI_TCNT field is incremented every time the last bit of an SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The transfer counter wraps around, incrementing the counter past 65535 resets the counter to zero.
bits 16–31	Reserved.

23.3.2.3 DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR_n)

The DSPI modules each contain eight clock and transfer attribute registers (DSPI_CTAR_n) which are used to define different transfer attribute configurations. Each DSPI_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB/LSB first

At the initiation of an SPI or DSI transfer, control logic selects the DSPI_CTAR that contains the transfer's attributes.

NOTE

The user must not write to the DSPI_CTARs while the DSPI is running.

In master mode, the DSPI_CTAR_n registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. In slave mode, a subset of the bit fields in the DSPI_CTAR0 and DSPI_CTAR1 registers are used to set the slave transfer attributes. See the individual bit descriptions for details on which bits are used in slave modes.

When the DSPI is configured as an SPI master, the CTAS field in the command portion of the TX FIFO entry selects which of the DSPI_CTAR registers is used on a per-frame basis. When the DSPI is configured as an SPI bus slave, the DSPI_CTAR0 register is used.

When the DSPI is configured as a DSI master, the DSICTAS field in the DSPI DSI configuration register (DSPI_DSICR) selects which of the DSPI_CTAR register is used. For more information on the DSPI_DSICR see [Section 23.3.2.10, “DSPI DSI Configuration Register \(DSPI_DSICR\).”](#) When the DSPI is configured as a DSI bus slave, the DSPI_CTAR1 register is used.

In CSI configuration, the transfer attributes are selected based on whether the current frame is SPI data or DSI data. SPI transfers in CSI configuration follow the protocol described for SPI configuration, and DSI transfers in CSI configuration follow the protocol described for DSI configuration. CSI configuration is only valid with master mode. See [Section 23.4.5, “Combined Serial Interface \(CSI\) Configuration,”](#) for more details.

Deserial Serial Peripheral Interface (DSPI)

Offset: DSPI_BASE +
 0x000C (DSPI_CTAR0)
 0x0010 (DSPI_CTAR1)
 0x0014 (DSPI_CTAR2)
 0x0018 (DSPI_CTAR3)
 0x001C (DSPI_CTAR4)
 0x0020 (DSPI_CTAR5)
 0x0024 (DSPI_CTAR6)
 0x0028 (DSPI_CTAR7)

Access: Read/Write

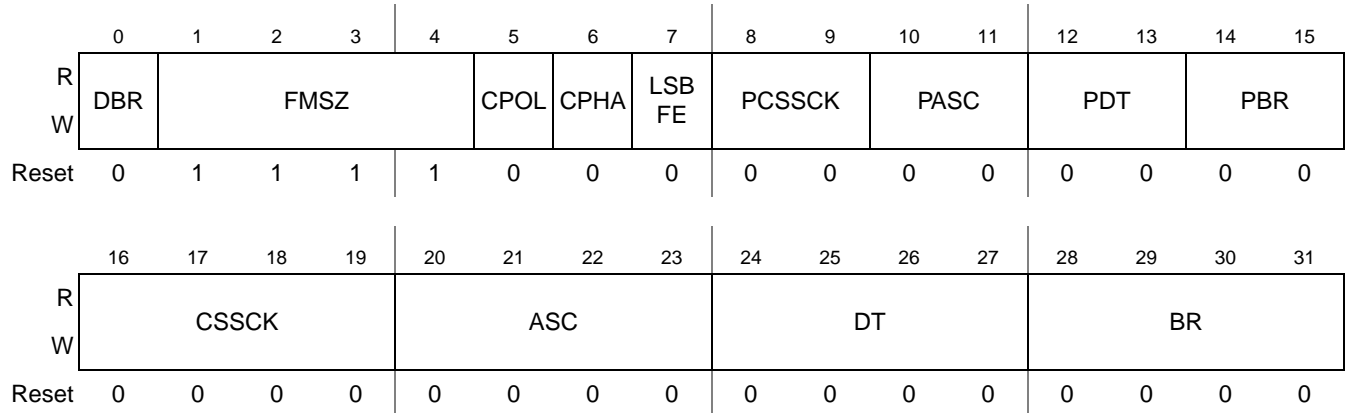


Figure 23-4. DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR n)

Table 23-4. DSPI_CTAR n Field Description

Field	Description																																								
DBR	<p>Double Baud Rate. The DBR bit doubles the effective baud rate of the serial communications clock (SCK). This field is only used in master mode. It effectively halves the baud rate division ratio supporting faster frequencies and odd division ratios for the serial communications clock (SCK). When the DBR bit is set, the duty cycle of the serial communications clock (SCK) depends on the value in the baud rate prescaler and the clock phase bit as listed below. See the BR field below and Section 23.4.7.1, "Baud Rate Generator," for details on how to compute the baud rate. If the overall baud rate is divide by two or divide by three of the system clock then the continuous SCK enable or the modified timing format enable bits must not be set.</p> <p>0 The baud rate is computed normally with a 50/50 duty cycle 1 Baud rate is doubled with the duty cycle depending on the baud rate prescaler</p> <table border="1"> <thead> <tr> <th>DBR</th> <th>CPHA</th> <th>PBR</th> <th>SCK Duty Cycle</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>any</td> <td>any</td> <td>50/50</td> </tr> <tr> <td>1</td> <td>0</td> <td>00</td> <td>50/50</td> </tr> <tr> <td>1</td> <td>0</td> <td>01</td> <td>33/66</td> </tr> <tr> <td>1</td> <td>0</td> <td>10</td> <td>40/60</td> </tr> <tr> <td>1</td> <td>0</td> <td>11</td> <td>43/57</td> </tr> <tr> <td>1</td> <td>1</td> <td>00</td> <td>50/50</td> </tr> <tr> <td>1</td> <td>1</td> <td>01</td> <td>66/33</td> </tr> <tr> <td>1</td> <td>1</td> <td>10</td> <td>60/40</td> </tr> <tr> <td>1</td> <td>1</td> <td>11</td> <td>57/43</td> </tr> </tbody> </table>	DBR	CPHA	PBR	SCK Duty Cycle	0	any	any	50/50	1	0	00	50/50	1	0	01	33/66	1	0	10	40/60	1	0	11	43/57	1	1	00	50/50	1	1	01	66/33	1	1	10	60/40	1	1	11	57/43
DBR	CPHA	PBR	SCK Duty Cycle																																						
0	any	any	50/50																																						
1	0	00	50/50																																						
1	0	01	33/66																																						
1	0	10	40/60																																						
1	0	11	43/57																																						
1	1	00	50/50																																						
1	1	01	66/33																																						
1	1	10	60/40																																						
1	1	11	57/43																																						
FMSZ	<p>FMSZ. Selects the number of bits transferred per frame. The FMSZ field is used in master mode and slave mode. The table below lists the frame sizes.</p> <table border="1"> <thead> <tr> <th>FMSZ</th> <th>Framesize</th> <th>FMSZ</th> <th>Framesize</th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>Reserved</td> <td>1000</td> <td>9</td> </tr> <tr> <td>0001</td> <td>Reserved</td> <td>1001</td> <td>10</td> </tr> <tr> <td>0010</td> <td>Reserved</td> <td>1010</td> <td>11</td> </tr> <tr> <td>0011</td> <td>4</td> <td>1011</td> <td>12</td> </tr> <tr> <td>0100</td> <td>5</td> <td>1100</td> <td>13</td> </tr> <tr> <td>0101</td> <td>6</td> <td>1101</td> <td>14</td> </tr> <tr> <td>0110</td> <td>7</td> <td>1110</td> <td>15</td> </tr> <tr> <td>0111</td> <td>8</td> <td>1111</td> <td>16</td> </tr> </tbody> </table>	FMSZ	Framesize	FMSZ	Framesize	0000	Reserved	1000	9	0001	Reserved	1001	10	0010	Reserved	1010	11	0011	4	1011	12	0100	5	1100	13	0101	6	1101	14	0110	7	1110	15	0111	8	1111	16				
FMSZ	Framesize	FMSZ	Framesize																																						
0000	Reserved	1000	9																																						
0001	Reserved	1001	10																																						
0010	Reserved	1010	11																																						
0011	4	1011	12																																						
0100	5	1100	13																																						
0101	6	1101	14																																						
0110	7	1110	15																																						
0111	8	1111	16																																						

Table 23-4. DSPI_CTAR n Field Description (continued)

Field	Description										
CPOL	<p>Clock Polarity. Selects the inactive state of the serial communications clock (SCK). This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock polarities. When the continuous selection format is selected (CONT = 1 or DCONT = 1), switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. For more information on continuous selection format, refer to Section 23.4.8.5, "Continuous Selection Format."</p> <p>0 The inactive state value of SCK is low 1 The inactive state value of SCK is high</p>										
CPHA	<p>Clock Phase. Selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock phase settings.</p> <p>0 Data is captured on the leading edge of SCK and changed on the following edge 1 Data is changed on the leading edge of SCK and captured on the following edge</p>										
LSBFE	<p>LSB First Enable. Selects if the LSB or MSB of the frame is transferred first. This bit is only used in master mode.</p> <p>0 Data is transferred MSB first 1 Data is transferred LSB first</p>										
PCSSCK	<p>PCS to SCK Delay Prescaler. Selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. This field is only used in master mode. The table below lists the prescaler values. The description for bitfield CSSCK in Table 23-4 details how to compute the PCS to SCK delay.</p> <table border="1" data-bbox="643 915 1131 1194"> <thead> <tr> <th>PCSSCK</th> <th>PCS to SCK Delay Prescaler Value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>1</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table>	PCSSCK	PCS to SCK Delay Prescaler Value	00	1	01	3	10	5	11	7
PCSSCK	PCS to SCK Delay Prescaler Value										
00	1										
01	3										
10	5										
11	7										
PASC	<p>After SCK Delay Prescaler. Selects the prescaler value for the delay between the last edge of SCK and the negation of PCSx. This field is only used in master mode. The table below lists the prescaler values. The description for bitfield ASC in Table 23-4 details how to compute the after SCK delay.</p> <table border="1" data-bbox="643 1335 1131 1614"> <thead> <tr> <th>PASC</th> <th>After SCK Delay Prescaler Value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>1</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table>	PASC	After SCK Delay Prescaler Value	00	1	01	3	10	5	11	7
PASC	After SCK Delay Prescaler Value										
00	1										
01	3										
10	5										
11	7										

Table 23-4. DSPI_CTAR n Field Description (continued)

Field	Description										
PDT	<p>Delay After Transfer Prescaler. The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is used in master mode only. The table below lists the prescaler values. The description for bitfield DT in Table 23-4 details how to compute the delay after transfer.</p> <table border="1"> <thead> <tr> <th>PDT</th> <th>Delay after Transfer Prescaler Value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>1</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table>	PDT	Delay after Transfer Prescaler Value	00	1	01	3	10	5	11	7
PDT	Delay after Transfer Prescaler Value										
00	1										
01	3										
10	5										
11	7										
PBR	<p>Baud Rate Prescaler. Selects the prescaler value for the baud rate. This field is used in master mode only. The baud rate is the frequency of the serial communications clock (SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. The baud rate prescaler values are listed in the table below. The description for Section 23.4.7.1, "Baud Rate Generator," details how to compute the baud rate.</p> <table border="1"> <thead> <tr> <th>PBR</th> <th>Baud Rate Prescaler Value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>2</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table>	PBR	Baud Rate Prescaler Value	00	2	01	3	10	5	11	7
PBR	Baud Rate Prescaler Value										
00	2										
01	3										
10	5										
11	7										

Table 23-4. DSPI_CTAR_n Field Description (continued)

Field	Description																																				
CSSCK	<p>PCS to SCK Delay Scaler. Selects the scaler value for the PCS to SCK delay. This field is used in master mode only. The PCS to SCK delay is the delay between the assertion of PCS and the first edge of the SCK. The table below lists the scaler values.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">CSSCK</th> <th style="text-align: center;">PCS to SCK Delay Scaler Value</th> <th style="text-align: center;">CSSCK</th> <th style="text-align: center;">PCS to SCK Delay Scaler Value</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">0000</td><td style="text-align: center;">2</td><td style="text-align: center;">1000</td><td style="text-align: center;">512</td></tr> <tr><td style="text-align: center;">0001</td><td style="text-align: center;">4</td><td style="text-align: center;">1001</td><td style="text-align: center;">1024</td></tr> <tr><td style="text-align: center;">0010</td><td style="text-align: center;">8</td><td style="text-align: center;">1010</td><td style="text-align: center;">2048</td></tr> <tr><td style="text-align: center;">0011</td><td style="text-align: center;">16</td><td style="text-align: center;">1011</td><td style="text-align: center;">4096</td></tr> <tr><td style="text-align: center;">0100</td><td style="text-align: center;">32</td><td style="text-align: center;">1100</td><td style="text-align: center;">8192</td></tr> <tr><td style="text-align: center;">0101</td><td style="text-align: center;">64</td><td style="text-align: center;">1101</td><td style="text-align: center;">16384</td></tr> <tr><td style="text-align: center;">0110</td><td style="text-align: center;">128</td><td style="text-align: center;">1110</td><td style="text-align: center;">32768</td></tr> <tr><td style="text-align: center;">0111</td><td style="text-align: center;">256</td><td style="text-align: center;">1111</td><td style="text-align: center;">65536</td></tr> </tbody> </table> <p>The PCS to SCK delay is a multiple of the system clock period and it is computed according to the following equation:</p> $t_{CSC} = \frac{1}{f_{SYS}} \times \text{PCSSCK Prescaler value} \times \text{CSSCK Scaler value}$ <p>Note: See Section 23.4.7.2, "PCS to SCK Delay (tCSC)," for more details.</p>	CSSCK	PCS to SCK Delay Scaler Value	CSSCK	PCS to SCK Delay Scaler Value	0000	2	1000	512	0001	4	1001	1024	0010	8	1010	2048	0011	16	1011	4096	0100	32	1100	8192	0101	64	1101	16384	0110	128	1110	32768	0111	256	1111	65536
CSSCK	PCS to SCK Delay Scaler Value	CSSCK	PCS to SCK Delay Scaler Value																																		
0000	2	1000	512																																		
0001	4	1001	1024																																		
0010	8	1010	2048																																		
0011	16	1011	4096																																		
0100	32	1100	8192																																		
0101	64	1101	16384																																		
0110	128	1110	32768																																		
0111	256	1111	65536																																		

Table 23-4. DSPI_CTAR n Field Description (continued)

Field	Description																																				
ASC	<p>After SCK Delay Scaler. Selects the scaler value for the after SCK delay. This field is used in master mode only. The after SCK delay is the delay between the last edge of SCKx and the negation of PCS. The table below lists the scaler values.</p> <table border="1"> <thead> <tr> <th>ASC</th> <th>After SCK Delay Scaler Value</th> <th>ASC</th> <th>After SCK Delay Scaler Value</th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>2</td> <td>1000</td> <td>512</td> </tr> <tr> <td>0001</td> <td>4</td> <td>1001</td> <td>1024</td> </tr> <tr> <td>0010</td> <td>8</td> <td>1010</td> <td>2048</td> </tr> <tr> <td>0011</td> <td>16</td> <td>1011</td> <td>4096</td> </tr> <tr> <td>0100</td> <td>32</td> <td>1100</td> <td>8192</td> </tr> <tr> <td>0101</td> <td>64</td> <td>1101</td> <td>16384</td> </tr> <tr> <td>0110</td> <td>128</td> <td>1110</td> <td>32768</td> </tr> <tr> <td>0111</td> <td>256</td> <td>1111</td> <td>65536</td> </tr> </tbody> </table> <p>The after SCK delay is a multiple of the system clock period, and it is computed according to the following equation:</p> $t_{ASC} = \frac{1}{f_{SYS}} \times \text{PASC Prescaler value} \times \text{ASC Scaler value}$ <p>Note: See Section 23.4.7.3, “After SCK Delay (tASC),” for more details.</p>	ASC	After SCK Delay Scaler Value	ASC	After SCK Delay Scaler Value	0000	2	1000	512	0001	4	1001	1024	0010	8	1010	2048	0011	16	1011	4096	0100	32	1100	8192	0101	64	1101	16384	0110	128	1110	32768	0111	256	1111	65536
ASC	After SCK Delay Scaler Value	ASC	After SCK Delay Scaler Value																																		
0000	2	1000	512																																		
0001	4	1001	1024																																		
0010	8	1010	2048																																		
0011	16	1011	4096																																		
0100	32	1100	8192																																		
0101	64	1101	16384																																		
0110	128	1110	32768																																		
0111	256	1111	65536																																		

Table 23-4. DSPI_CTARn Field Description (continued)

Field	Description																																				
DT	<p>Delay After Transfer Scaler. The DT field selects the delay after transfer scaler. This field is used in master mode only. The delay after transfer is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The table below lists the scaler values.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">DT</th> <th style="text-align: center;">Delay after Transfer Scaler Value</th> <th style="text-align: center;">DT</th> <th style="text-align: center;">Delay after Transfer Scaler Value</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">0000</td><td style="text-align: center;">2</td><td style="text-align: center;">1000</td><td style="text-align: center;">512</td></tr> <tr><td style="text-align: center;">0001</td><td style="text-align: center;">4</td><td style="text-align: center;">1001</td><td style="text-align: center;">1024</td></tr> <tr><td style="text-align: center;">0010</td><td style="text-align: center;">8</td><td style="text-align: center;">1010</td><td style="text-align: center;">2048</td></tr> <tr><td style="text-align: center;">0011</td><td style="text-align: center;">16</td><td style="text-align: center;">1011</td><td style="text-align: center;">4096</td></tr> <tr><td style="text-align: center;">0100</td><td style="text-align: center;">32</td><td style="text-align: center;">1100</td><td style="text-align: center;">8192</td></tr> <tr><td style="text-align: center;">0101</td><td style="text-align: center;">64</td><td style="text-align: center;">1101</td><td style="text-align: center;">16384</td></tr> <tr><td style="text-align: center;">0110</td><td style="text-align: center;">128</td><td style="text-align: center;">1110</td><td style="text-align: center;">32768</td></tr> <tr><td style="text-align: center;">0111</td><td style="text-align: center;">256</td><td style="text-align: center;">1111</td><td style="text-align: center;">65536</td></tr> </tbody> </table> <p>The delay after transfer is a multiple of the system clock period and it is computed according to the following equation:</p> $t_{DT} = \frac{1}{f_{SYS}} \times \text{PDT Prescaler value} \times \text{DT Scaler value}$ <p>Note: See Section 23.4.7.4, "Delay after Transfer (tDT)," for more details</p>	DT	Delay after Transfer Scaler Value	DT	Delay after Transfer Scaler Value	0000	2	1000	512	0001	4	1001	1024	0010	8	1010	2048	0011	16	1011	4096	0100	32	1100	8192	0101	64	1101	16384	0110	128	1110	32768	0111	256	1111	65536
DT	Delay after Transfer Scaler Value	DT	Delay after Transfer Scaler Value																																		
0000	2	1000	512																																		
0001	4	1001	1024																																		
0010	8	1010	2048																																		
0011	16	1011	4096																																		
0100	32	1100	8192																																		
0101	64	1101	16384																																		
0110	128	1110	32768																																		
0111	256	1111	65536																																		

Table 23-4. DSPI_CTAR n Field Description (continued)

Field	Description																																				
BR	<p>Baud Rate Scaler. Selects the scaler value for the baud rate. This field is used in master mode only. The pre-scaled system clock is divided by the baud rate scaler to generate the frequency of the SCK. The table below lists the baud rate scaler values.</p> <table border="1"> <thead> <tr> <th>BR</th> <th>Baud Rate Scaler Value</th> <th>BR</th> <th>Baud Rate Scaler Value</th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>2</td> <td>1000</td> <td>256</td> </tr> <tr> <td>0001</td> <td>4</td> <td>1001</td> <td>512</td> </tr> <tr> <td>0010</td> <td>6</td> <td>1010</td> <td>1024</td> </tr> <tr> <td>0011</td> <td>8</td> <td>1011</td> <td>2048</td> </tr> <tr> <td>0100</td> <td>16</td> <td>1100</td> <td>4096</td> </tr> <tr> <td>0101</td> <td>32</td> <td>1101</td> <td>8192</td> </tr> <tr> <td>0110</td> <td>64</td> <td>1110</td> <td>16384</td> </tr> <tr> <td>0111</td> <td>128</td> <td>1111</td> <td>32768</td> </tr> </tbody> </table> <p>The baud rate is computed according to the following equation:</p> $\text{SCK baud rate} = \frac{f_{\text{SYS}}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$ <p>Note: See Section 23.4.7.1, “Baud Rate Generator,” for more details.</p>	BR	Baud Rate Scaler Value	BR	Baud Rate Scaler Value	0000	2	1000	256	0001	4	1001	512	0010	6	1010	1024	0011	8	1011	2048	0100	16	1100	4096	0101	32	1101	8192	0110	64	1110	16384	0111	128	1111	32768
BR	Baud Rate Scaler Value	BR	Baud Rate Scaler Value																																		
0000	2	1000	256																																		
0001	4	1001	512																																		
0010	6	1010	1024																																		
0011	8	1011	2048																																		
0100	16	1100	4096																																		
0101	32	1101	8192																																		
0110	64	1110	16384																																		
0111	128	1111	32768																																		

23.3.2.4 DSPI Status Register (DSPI_SR)

The DSPI_SR contains status and flag bits. The bits reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear a flag bit in the DSPI_SR by writing a 1 to it. Writing a 0 to a flag bit has no effect.

NOTE

This register cannot be written in MDIS Mode, owing to the use of power saving mechanisms.

Deserial Serial Peripheral Interface (DSPI)

Offset: DSPI_BASE + 0x002C

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	0	0	RFOF	0	RDFD	0
W	w1c			w1c	w1c		w1c						w1c		w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-5. DSPI Status Register (DSPI_SR)

Table 23-5. DSPI_SR Field Descriptions

Field	Description
TCF	Transfer Complete Flag. Indicates that all bits in a frame have been shifted out. The TCF bit is set after the last incoming databit is sampled, but before the tASC delay starts. Refer to Section 23.4.8.1, “Classic SPI Transfer Format (CPHA = 0)” , for details. The TCF bit is cleared by writing 1 to it. 0 Transfer not complete 1 Transfer complete
TXRXS	TX and RX Status. Reflects the status of the DSPI. See Section 23.4.2, “Start and Stop of DSPI Transfers,” for information on what causes this bit to be negated or asserted. 0 TX and RX operations are disabled (DSPI is in stopped state) 1 TX and RX operations are enabled (DSPI is in running state)
bit 2	Reserved.
EOQF	End of Queue Flag. Indicates that transmission in progress is the last entry in a queue. The EOQF bit is set when the TX FIFO entry has the EOQ bit set in the command halfword and after the last incoming databit is sampled, but before the tASC delay starts. Refer to Section 23.4.8.1, “Classic SPI Transfer Format (CPHA = 0)” , for details. The EOQF bit is cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executing command 1 EOQ bit is set in the executing SPI command Note: EOQF does not function in slave mode.
TFUF	Transmit FIFO Underflow Flag. Indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit is cleared by writing 1 to it. 0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred
bit 5	Reserved.
7266>>>	Transmit FIFO Fill Flag. Indicates that the TX FIFO can be filled. Provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by writing 1 to it or by an acknowledgement from the eDMA controller when the TX FIFO is full. 0 TX FIFO is full 1 TX FIFO is not full

Table 23-5. DSPI_SR Field Descriptions (continued)

Field	Description
bit 7–11	Reserved.
RFOF	Receive FIFO Overflow Flag. Indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit is cleared by writing 1 to it. 0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred
bit 13	Reserved.
RFDF	Receive FIFO Drain Flag. Indicates that the RX FIFO can be drained. Provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing 1 to it or by an acknowledgement from the eDMA controller when the RX FIFO is empty. 0 RX FIFO is empty 1 RX FIFO is not empty Note: In the interrupt service routine, RFDF must be cleared only after the DSPI_POPR register is read.
bit 15	Reserved.
TXCTR	TX FIFO Counter. Indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI_PUSH register is written. The TXCTR is decremented every time an SPI command is executed and the SPI data is transferred to the shift register.
TXNXT PTR	Transmit Next Pointer. Indicates which TX FIFO entry will be transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See Section 23.4.3.4, “Transmit First-In First-Out (TX FIFO) Buffering Mechanism,” for more details.
RXCTR	RX FIFO Counter. Indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI_POPR is read. The RXCTR is incremented after the last incoming databit is sampled, but before the tASC delay starts. Refer to Section 23.4.8.1, “Classic SPI Transfer Format (CPHA = 0),” for details.
POPNXTPTR	Pop Next Pointer. Contains a pointer to the RX FIFO entry that will be returned when the DSPI_POPR is read. The POPNXTPTR is updated when the DSPI_POPR is read. See Section 23.4.3.5, “Receive First-In First-Out (RX FIFO) Buffering Mechanism,” for more details.

23.3.2.5 DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

The DSPI_RSER serves two purposes. It enables flag bits in the DSPI_SR to generate DMA requests or interrupt requests. The DSPI_RSER also selects the type of request to be generated. See the individual bit descriptions for information on the types of requests the bits support.

NOTE

The user must not write to the DSPI_RSER while the DSPI is running.

Deserial Serial Peripheral Interface (DSPI)

Offset: DSPI_BASE + 0x0030

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF_	0	0	EOQF	TFUF	0	TFFF	TFFF_	0	0	0	0	RFOF	0	RFDF	RFDF_
W	RE			_RE	_RE		_RE	DIRS					_RE		_RE	DIRS
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-6. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

Table 23-6. DSPI_RSER Field Descriptions

Field	Description
TCF_RE	Transmission Complete Request Enable. Enables TCF flag in the DSPI_SR to generate an interrupt request. 0 TCF interrupt requests are disabled 1 TCF interrupt requests are enabled
bit 1–2	Reserved.
EOQF_RE	DSPI Finished Request Enable. Enables the EOQF flag in the DSPI_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled 1 EOQF interrupt requests are enabled
TFUF_RE	Transmit FIFO Underflow Request Enable. The TFUF_RE bit enables the TFUF flag in the DSPI_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled 1 TFUF interrupt requests are enabled
bit 5	Reserved.
TFFF_RE	Transmit FIFO Fill Request Enable. Enables the TFFF flag in the DSPI_SR to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled 1 TFFF interrupt requests or DMA requests are enabled
TFFF_DIRS	Transmit FIFO Fill DMA or Interrupt Request Select. Selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI_SR is set and the TFFF_RE bit in the DSPI_RSER is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated
bits 8–11	Reserved.
RFOF_RE	Receive FIFO Overflow Request Enable. Enables the RFOF flag in the DSPI_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled 1 RFOF interrupt requests are enabled
bit 13	Reserved.

Table 23-6. DSPI_RSER Field Descriptions (continued)

Field	Description
RFDF_RE	Receive FIFO Drain Request Enable. Enables the RFDF flag in the DSPI_SR to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled 1 RFDF interrupt requests or DMA requests are enabled
RFDF_DIRS	Receive FIFO Drain DMA or Interrupt Request Select. Selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI_SR is set and the RFDF_RE bit in the DSPI_RSER is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated
bits 16–31	Reserved.

23.3.2.6 DSPI PUSH TX FIFO Register (DSPI_PUSHR)

The DSPI_PUSHR provides a means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. See [Section 23.4.3.4, “Transmit First-In First-Out \(TX FIFO\) Buffering Mechanism,”](#) for more information. Write accesses of 8- or 16-bits to the DSPI_PUSHR will transfer 32 bits to the TX FIFO.

NOTE

Only the TXDATA field is used for DSPI slaves.

Offset: DSPI_BASE + 0x0034

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
R	CONT				CTAS		EOQ	CT	0	0	0	0	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
W	CONT				CTAS		EOQ	CNT					PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		
R	TXDATA																	
W	TXDATA																	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-7. DSPI PUSH TX FIFO Register (DSPI_PUSHR)

Table 23-7. DSPI_PUSHR Field Descriptions

Field	Description																		
CONT	<p>Continuous Peripheral Chip Select Enable. Selects a continuous selection format. The bit is used in SPI master mode. The bit enables the selected PCS signals to remain asserted between transfers. See Section 23.4.8.5, "Continuous Selection Format," for more information.</p> <p>0 Return peripheral chip select signals to their inactive state between transfers 1 Keep peripheral chip select signals asserted between transfers</p>																		
CTAS	<p>Clock and Transfer Attributes Select. Selects which of the DSPI_CTARs is used to set the transfer attributes for the associated SPI frame. In SPI slave mode DSPI_CTAR0 is used. The table below shows how the CTAS values map to the DSPI_CTARs. There are eight DSPI_CTARs in this device's DSPI.</p> <p>Note: The field is used in SPI master mode only.</p> <table border="1" data-bbox="625 598 1112 1066"> <thead> <tr> <th>CTAS</th> <th>Use Clock and Transfer Attributes from</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>DSPI_CTAR0</td> </tr> <tr> <td>001</td> <td>DSPI_CTAR1</td> </tr> <tr> <td>010</td> <td>DSPI_CTAR2</td> </tr> <tr> <td>011</td> <td>DSPI_CTAR3</td> </tr> <tr> <td>100</td> <td>DSPI_CTAR4</td> </tr> <tr> <td>101</td> <td>DSPI_CTAR5</td> </tr> <tr> <td>110</td> <td>DSPI_CTAR6</td> </tr> <tr> <td>111</td> <td>DSPI_CTAR7</td> </tr> </tbody> </table>	CTAS	Use Clock and Transfer Attributes from	000	DSPI_CTAR0	001	DSPI_CTAR1	010	DSPI_CTAR2	011	DSPI_CTAR3	100	DSPI_CTAR4	101	DSPI_CTAR5	110	DSPI_CTAR6	111	DSPI_CTAR7
CTAS	Use Clock and Transfer Attributes from																		
000	DSPI_CTAR0																		
001	DSPI_CTAR1																		
010	DSPI_CTAR2																		
011	DSPI_CTAR3																		
100	DSPI_CTAR4																		
101	DSPI_CTAR5																		
110	DSPI_CTAR6																		
111	DSPI_CTAR7																		
EOQ	<p>End of Queue. Provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer, the EOQF bit in the DSPI_SR is set.</p> <p>0 The SPI data is not the last data to transfer 1 The SPI data is the last data to transfer</p> <p>Note: This bitfield is used in SPI master mode only.</p>																		
CTCNT	<p>Clear SPI_TCNT. Provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the SPI_TCNT field in the DSPI_TCR. The SPI_TCNT field is cleared before transmission of the current SPI frame begins.</p> <p>0 Do not clear SPI_TCNT field in the DSPI_TCR 1 Clear SPI_TCNT field in the DSPI_TCR</p> <p>Note: This bitfield is used in SPI master mode only.</p>																		
bits 6–7	Reserved.																		
bits 8–9	Reserved.																		
PCS n	<p>Peripheral Chip Select n. Selects which PCSx signals will be asserted for the transfer.</p> <p>0 Negate the PCSn signal 1 Assert the PCSn signal</p> <p>Note: This bitfield is used in SPI master mode only.</p>																		
TXDATA	Transmit Data. Holds SPI data to be transferred according to the associated SPI command.																		

23.3.2.7 DSPI POP RX FIFO Register (DSPI_POPR)

The DSPI_POPR provides a means to read the RX FIFO. See [Section 23.4.3.5, “Receive First-In First-Out \(RX FIFO\) Buffering Mechanism,”](#) for a description of the RX FIFO operations. Eight- or 16-bit read accesses to the DSPI_POPR will read from the RX FIFO and update the counter and pointer.

NOTE

The DSPI_POPR must not be read speculatively. For future compatibility, the TLB (MMU table) entry covering the DSPI_POPR must be configured to be guarded.

Offset: DSPI_BASE + 0x0038

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-8. DSPI POP RX FIFO Register (DSPI_POPR)

Table 23-8. DSPI_POPR Field Descriptions

Field	Description
bits 0–15	Reserved.
RXDATA	Received Data. The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (POPNEXTPTR).

23.3.2.8 DSPI Transmit FIFO Registers 0–3 (DSPI_TXFR n)

The DSPI_TXFR n registers provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI_TXFR n registers does not alter the state of the TX FIFO. The MCU uses four registers to implement the TX FIFO, that is DSPI_TXFR0–DSPI_TXFR3 are used.

Offset: DSPI_BASE +
 0x003C (DSPI_TXFR0)
 0x0040 (DSPI_TXFR1)
 0x0044 (DSPI_TXFR2)
 0x0048 (DSPI_TXFR3)

Access: Read

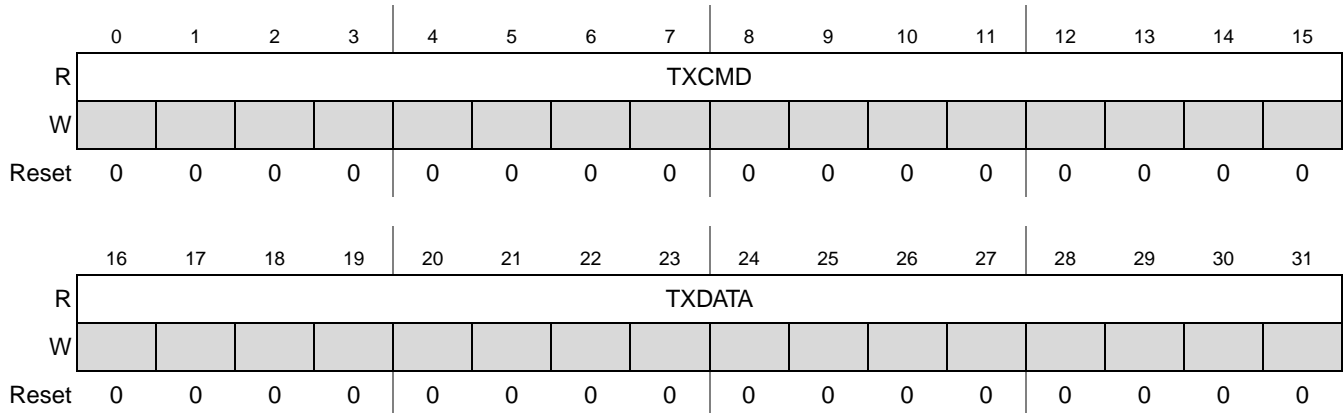


Figure 23-9. DSPI Transmit FIFO Register 0–3 (DSPI_TXFR n)

Table 23-9. DSPI_TXFR n Field Descriptions

Field	Description
TXCMD	Transmit Command. Contains the command that sets the transfer attributes for the SPI data. See Section 23.3.2.6, “DSPI PUSH TX FIFO Register (DSPI_PUSHR)” , for details on the command field.
TXDATA	Transmit Data. Contains the SPI data to be shifted out.

23.3.2.9 DSPI Receive FIFO Registers 0–3 (DSPI_RXFR n)

The DSPI_RXFR n registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI_RXFR registers are read-only. Reading the DSPI_RXFR n registers does not alter the state of the RX FIFO. The device uses four registers to implement the RX FIFO, that is DSPI_RXFR0–DSPI_RXFR3 are used.

Offset: DSPI_BASE +
 0x007C (DSPI_RXFR0)
 0x0080 (DSPI_RXFR1)
 0x0084 (DSPI_RXFR2)
 0x0088 (DSPI_RXFR3)

Access: Read

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-10. DSPI Receive FIFO Registers 0–3 (DSPI_RXFRn)

Table 23-10. DSPI_RXFRn Field Description

Field	Description
bits 0–15	Reserved.
RXDATA	Receive Data. Contains the received SPI data.

23.3.2.10 DSPI DSI Configuration Register (DSPI_DSICR)

The DSPI_DSICR selects various attributes associated with DSI and CSI configurations.

NOTE

The user must not write to the DSPI_DSICR while the DSPI is running.

Offset: DSPI_BASE + 0x00BC

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	TXSS	0	0	CID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DCO	DSICTAS			0	0	0	0	0	0	DPCS	DPCS	DPCS	DPCS	DPCS	DPCS
W	NT										5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-11. DSPI DSI Configuration Register (DSPI_DSICR)

Table 23-11. DSPI_DSICR Field Descriptions

Field	Description																		
bits 0–11	Reserved.																		
TXSS	Transmit Data Source Select. Selects the source of data to be serialized. The source can be data from host software written to the DSPI DSI alternate serialization data register (DSPI_ASDR) or parallel output pin states latched into the DSPI DSI serialization data register (DSPI_SDR). 0 Source of serialized data is the DSPI_SDR 1 Source of serialized data is the DSPI_ASDR																		
bits 13–14	Reserved.																		
CID	Change in Data Transfer Enable. Enables a change in serialization data to initiate a transfer. The bit is used in master mode in DSI and CSI configurations to control when to initiate transfers. When the CID bit is set, serialization is initiated when the current DSI data differs from the previous DSI data shifted out. The DSPI_COMPR is compared with the DSPI_SDR or DSPI_ASDR to detect a change in data. Refer to Section 23.4.4.5, “DSI Transfer Initiation Control,” for more information. 0 Change in data transfer operation disabled 1 Change in data transfer operation enabled																		
DCONT	DSI Continuous Peripheral Chip Select Enable. Enables the PCSx signals to remain asserted between transfers. The DCONT bit affects the PCS signals in DSI master mode only. See Section 23.4.8.5, “Continuous Selection Format,” for details. 0 Return peripheral chip select signals to their inactive state after transfer is complete 1 Keep peripheral chip select signals asserted after transfer is complete																		
DSICTAS	DSI Clock and Transfer Attributes Select. The DSICTAS field selects which of the DSPI_CTARs is used to provide transfer attributes in DSI configuration. The DSICTAS field is used in DSI master mode. In DSI slave mode, the DSPI_CTAR1 is always selected. The table below shows how the DSICTAS values map to the DSPI_CTARs. <table border="1" data-bbox="550 1020 1216 1493"> <thead> <tr> <th>DSICTAS</th> <th>DSI Clock and Transfer Attributes Controlled by</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>DSPI_CTAR0</td> </tr> <tr> <td>001</td> <td>DSPI_CTAR1</td> </tr> <tr> <td>010</td> <td>DSPI_CTAR2</td> </tr> <tr> <td>011</td> <td>DSPI_CTAR3</td> </tr> <tr> <td>100</td> <td>DSPI_CTAR4</td> </tr> <tr> <td>101</td> <td>DSPI_CTAR5</td> </tr> <tr> <td>110</td> <td>DSPI_CTAR6</td> </tr> <tr> <td>111</td> <td>DSPI_CTAR7</td> </tr> </tbody> </table>	DSICTAS	DSI Clock and Transfer Attributes Controlled by	000	DSPI_CTAR0	001	DSPI_CTAR1	010	DSPI_CTAR2	011	DSPI_CTAR3	100	DSPI_CTAR4	101	DSPI_CTAR5	110	DSPI_CTAR6	111	DSPI_CTAR7
DSICTAS	DSI Clock and Transfer Attributes Controlled by																		
000	DSPI_CTAR0																		
001	DSPI_CTAR1																		
010	DSPI_CTAR2																		
011	DSPI_CTAR3																		
100	DSPI_CTAR4																		
101	DSPI_CTAR5																		
110	DSPI_CTAR6																		
111	DSPI_CTAR7																		
bits 20–25	Reserved.																		
DPCS _n	DSI Peripheral Chip Select <i>n</i> . The DPCS bits select which of the PCSx signals to assert during a DSI transfer. The DPCS bits control the assertions of the PCSx signals in DSI master mode only. 0 Negate PCS _n 1 Assert PCS _n																		

23.3.2.11 DSPI DSI Serialization Data Register (DSPI_SDR)

The DSPI_SDR contains the signal states of the parallel input signals from the eMIOS. The pin states of the parallel input signals are latched into the DSPI_SDR on the rising edge of every system clock. The

DSPI_SDR is read-only. When the TXSS bit in the DSPI_DSICR is negated, the data in the DSPI_SDR is the source of the serialized data.

Offset: DSPI_BASE + 00C0

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SER_DATA [15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-12. DSPI DSI Serialization Data Register (DSPI_SDR)

Table 23-12. DSPI_SDR Field Description

Bits	Description
bits 0–15	Reserved.
SER_DATA [15:0]	Serialized Data. The SER_DATA field contains the signal states of the parallel input signals. SER_DATA [15:0] maps to DSPI serialization inputs IN[15:0].

23.3.2.12 DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

The DSPI_ASDR provides a means for host software to write the data to be serialized. When the TXSS bit in the DSPI_DSICR is set, the data in the DSPI_ASDR is the source of the serialized data. Writes to the DSPI_ASDR take effect on the next frame boundary.

Offset: DSPI_BASE + 0x00C4

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ASER_DATA [15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-13. DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

Table 23-13. DSPI_ASDR Field Description

Field	Description
bits 0–15	Reserved.
ASER_DATA [15:0]	Alternate Serialized Data. The ASER_DATA field holds the alternate data to be serialized.

23.3.2.13 DSPI DSI Transmit Comparison Register (DSPI_COMPR)

The DSPI_COMPR holds a copy of the last transmitted DSI data. The DSPI_COMPR is read-only. DSI data is transferred to this register as it is loaded into the TX shift register.

Offset: DSPI_BASE + 0x00C8

Access: Read

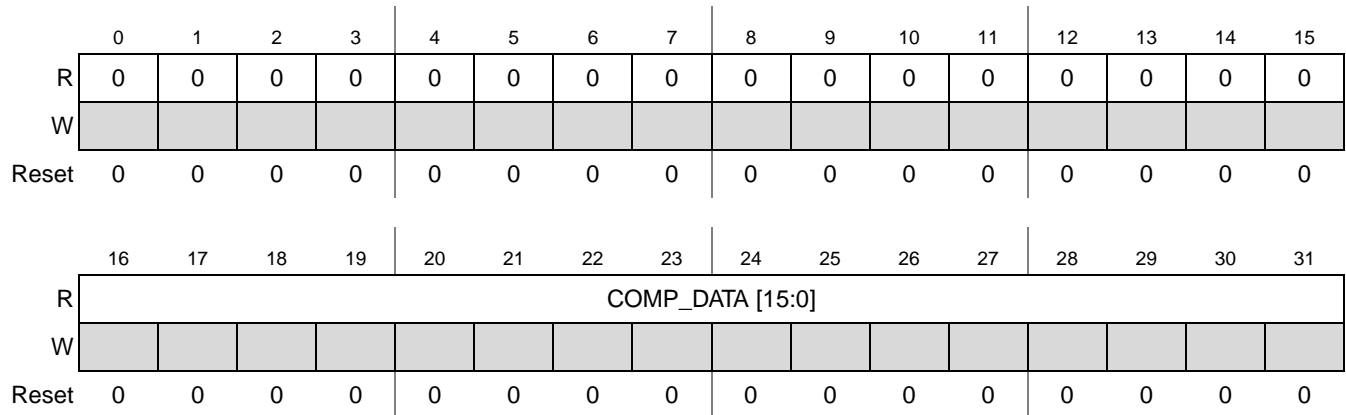


Figure 23-14. DSPI DSI Transmit Comparison Register (DSPI_COMPR)

Table 23-14. DSPI_COMPR Field Description

Field	Description
bits 0–15	Reserved.
COMP_DATA [15:0]	Compare Data. The COMP_DATA field holds the last serialized DSI data.

23.3.2.14 DSPI DSI Deserialization Data Register (DSPI_DDR)

The DSPI_DDR holds the signal states for the parallel output signals. The DSPI_DDR is read-only and is memory mapped so that host software can read the incoming DSI frames.

Offset: DSPI_BASE + 0x00CC

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DESER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-15. DSPI Deserialization Data Register (DSPI_DDR)

Table 23-15. DSPI_DDR Field Description

Field	Description
bits 0–15	Reserved.
DESER_DATA	Deserialized Data. Holds deserialized data that is presented as signal states to the parallel output signals.

23.4 Functional Description

The DSPI supports full-duplex, synchronous serial communications between the MCU and peripheral devices. The DSPI can also be used to reduce the number of pins required for I/O by serializing and deserializing up to 16 parallel input/output signals from the eMIOS. All communications are through an SPI-like protocol.

The DSPI has three configurations:

- SPI configuration in which the DSPI operates as a basic SPI or a queued SPI.
- DSI configuration in which the DSPI serializes and deserializes parallel input/output signals or bits from memory mapped registers.
- CSI configuration in which the DSPI combines the functionality of the SPI and DSI configurations.

The DCONF field in the DSPI_x_MCR register determines the DSPI configuration. See [Table 23-2](#) for the DSPI configuration values.

The DSPI_x_CTAR0–DSPI_x_CTAR7 registers hold clock and transfer attributes. The manner in which a CTAR is selected depends on the DSPI configuration (SPI, DSI, or CSI). The SPI configuration can select which CTAR to use on a frame by frame basis by setting the CTAS field in the DSPI_x_PUSHR. The DSI configuration statically selects which CTAR to use. In CSI configuration, priority logic determines if SPI data or DSI data is transferred. The type of data transferred (whether DSI or SPI) dictates which CTAR the CSI configuration will use. See [Section 23.3.2.3, “DSPI Clock and Transfer Attributes Registers 0–7 \(DSPI_CTARn\),”](#) for information on DSPI_x_CTAR fields.

23.4.1 Modes of Operation

The DSPI modules have four available distinct modes:

- Master mode
- Slave mode
- Module disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes, but debug mode is a device-specific mode.

The module-specific modes are determined by bits in the DSPIx_MCR. Debug mode is a mode that the entire device can enter in parallel with the DSPI being configured in one of its module-specific modes.

23.4.1.1 Master Mode

In master mode the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the MSTR bit in the DSPIx_MCR is set. The serial communications clock (SCK) is controlled by the master DSPI. All three DSPI configurations are valid in master mode.

In SPI configuration, master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPIx_CTARs will be used to set the transfer attributes. Transfer attribute control is on a frame by frame basis. See [Section 23.4.3, “Serial Peripheral Interface \(SPI\) Configuration,”](#) for more details.

In DSI configuration, master mode transfer attributes are controlled by the DSPIx_DSCIR. A detailed description of the DSPIx_DSCIR is located in [Section 23.3.2.10, “DSPI DSI Configuration Register \(DSPI_DSICR\).”](#) The DSISCTAS field in the DSPIx_DSICR selects which of the DSPIx_CTARs will be used to set the transfer attributes. Transfer attributes are set up during initialization and must not be changed between frames. See [Section 23.4.4, “Deserial Serial Interface \(DSI\) Configuration,”](#) for more details.

The CSI configuration is only available in master mode. In CSI configuration, the DSI data is transferred using DSI configuration transfer attributes and SPI data is transferred using the SPI configuration transfer attributes. For the bus slave to distinguish between DSI and SPI frames, the transfer attributes for the two types of frames must use different peripheral chip select signals. See [Section 23.4.5, “Combined Serial Interface \(CSI\) Configuration,”](#) for details.

23.4.1.2 Slave Mode

In slave mode the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the MSTR bit in the DSPIx_MCR is negated. The DSPI slave is selected by a bus master by asserting the slave's \overline{SS} . In slave mode, the bus master provides the SCK. All transfer attributes are controlled by the bus master but clock polarity, clock phase, and numbers of bits to transfer must still be configured in the DSPI slave for proper communications.

The SPI and DSI configurations are valid in slave mode. CSI configuration is not available in slave mode. In SPI slave mode the slave transfer attributes are set in the DSPIx_CTAR0. In DSI slave mode the slave

transfer attributes are set in the DSPIx_CTAR1. In slave mode, for both SPI and DSI configurations, data is transferred MSB first. The LSBFE field of the associated CTAR is ignored.

23.4.1.3 Module Disable Mode

The module disable mode is used for MCU power management. The clock to the non-memory-mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPIx_MCR is set. See [Section 23.4.12, “Power Saving Features,”](#) for more details on the module disable mode.

23.4.1.4 Halt Mode

When the appropriate bit in the SIU_HLT register is set, a request to enter halt mode will be made to the DSPI. The DSPI will not acknowledge the request to enter halt mode until it has reached a frame boundary. When the DSPI has reached a frame boundary it will halt all operations and indicate that it is ready to have its clocks shut off. The DSPI exits halt mode and resumes normal operation once the clocks are turned on. Serial communications or register accesses made while in halt mode are ignored even if the clocks have not been shut off yet. See [Section 23.4.12, “Power Saving Features,”](#) for more details on the halt mode.

23.4.1.5 Debug Mode

The debug mode is used for system development and debugging. If the MCU enters debug mode while the FRZ bit in the DSPIx_MCR is set, the DSPI stops all serial transfers and enters a stopped state. If the MCU enters debug mode while the FRZ bit is negated, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI. The DSPI enters debug mode when a debug request is asserted by an external controller. See [Figure 23-16](#) for a state diagram.

23.4.2 Start and Stop of DSPI Transfers

The DSPI has two operating states: stopped and running. The states are independent of DSPI configuration. The default state of the DSPI is stopped. In the stopped state no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The stopped state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. The TXRXS bit in the DSPIx_SR is negated in this state. In the running state, serial transfers take place. The TXRXS bit in the DSPIx_SR is asserted in the running state. [Figure 23-16](#) shows a state diagram of the start and stop mechanism. The transitions are described in [Table 23-16](#).

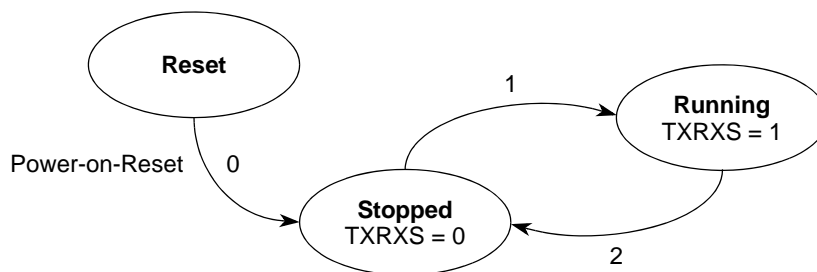


Figure 23-16. DSPI Start and Stop State Diagram

Table 23-16. State Transitions for Start and Stop of DSPI Transfers

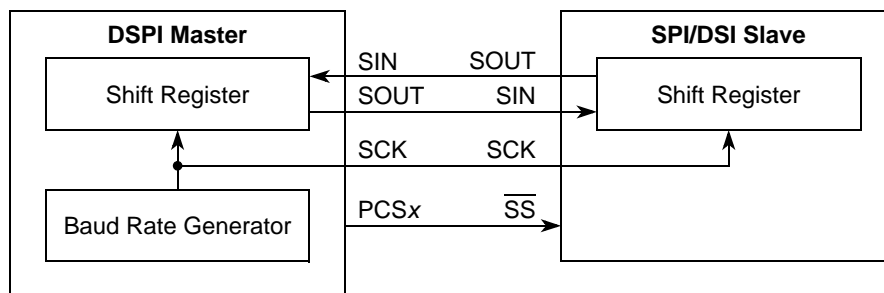
Transition #	Current State	Next State	Description
0	Reset	Stopped	Generic power-on-reset transition
1	Stopped	Running	The DSPI is started (DSPI transitions to running) when all of the following conditions are true: <ul style="list-style-type: none"> • EOQF bit is clear • Debug mode is unselected or the FRZ bit is clear • HALT bit is clear
2	Running	Stopped	The DSPI stops (transitions from running to stopped) after the current frame for any one of the following conditions: <ul style="list-style-type: none"> • EOQF bit is set • Debug mode is selected and the FRZ bit is set • HALT bit is set

State transitions from running to stopped occur on the next frame boundary if a transfer is in progress, or on the next system clock cycle if no transfers are in progress.

23.4.3 Serial Peripheral Interface (SPI) Configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI configuration when the DCONF field in the DSPIx_MCR is 0b00. The SPI frames can be from four to 16 bits long. The data to be transmitted can come from queues stored in RAM external to the DSPI. Host software or an eDMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or an eDMA controller transfers the received data from the RX FIFO to memory external to the DSPI. The FIFO buffer operations are described in [Section 23.4.3.4, “Transmit First-In First-Out \(TX FIFO\) Buffering Mechanism,”](#) and [Section 23.4.3.5, “Receive First-In First-Out \(RX FIFO\) Buffering Mechanism.”](#) The interrupt and DMA request conditions are described in [Section 23.4.11, “DMA and Interrupt Conditions.”](#)

Figure 23-17 shows an example of how a master DSPI connects to a SPI slave in SPI Configuration.

**Figure 23-17. DSPI Connections for SPI and DSI Transfers**

The SPI configuration supports two module-specific modes: master mode and slave mode. The FIFO operations are similar for the master mode and slave mode. The main difference is that in master mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO

entry. In slave mode the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field of the TX FIFO entry is ignored.

23.4.3.1 SPI Master Mode

In SPI master mode the DSPI initiates the serial transfers by controlling the serial communications clock (SCK_x) and the peripheral chip select (PCS_x) signals. The SPI command field in the executing TX FIFO entry determines which CTARs will be used to set the transfer attributes and which PCS_x signal to assert. The command field also contains various bits that help with queue management and transfer protocol. See [Section 23.3.2.6, “DSPI PUSH TX FIFO Register \(DSPI_PUSHR\),”](#) for details on the SPI command fields. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the serial out (SOUT_x) pin. In SPI master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

23.4.3.2 SPI Slave Mode

In SPI slave mode the DSPI responds to transfers initiated by an SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for successful communication with an SPI master. The SPI slave mode transfer attributes are set in the DSPI_x_CTAR0.

23.4.3.3 FIFO Disable Operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The TX and RX FIFOs are disabled separately. The TX FIFO is disabled by writing a 1 to the DIS_TXF bit in the DSPI_x_MCR. The RX FIFO is disabled by writing a 1 to the DIS_RXF bit in the DSPI_x_MCR.

The FIFO disable mechanisms are transparent to the user and to host software; transmit data and commands are written to the DSPI_x_PUSHR and received data is read from the DSPI_x_POPR. When the TX FIFO is disabled, the TFFF, TFUF, and TXCTR fields in DSPI_x_SR behave as if there is a one-entry FIFO but the contents of the DSPI_x_TXFRs and TXNXTPTR are undefined. When the RX FIFO is disabled, the RFDF, RFOF, and RXCTR fields in the DSPI_x_SR behave as if there is a one-entry FIFO but the contents of the DSPI_x_RXFRs and POPNXTPTR are undefined.

The TX and RX FIFOs must be disabled only if the application's operating mode requires the FIFO to be disabled. A FIFO must be disabled before it is accessed. Failure to disable a FIFO prior to a first FIFO access is not supported, and may result in incorrect results.

23.4.3.4 Transmit First-In First-Out (TX FIFO) Buffering Mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds four entries, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI push TX FIFO register (DSPI_x_PUSHR). For more information on DSPI_x_PUSHR, refer to [Section 23.3.2.6, “DSPI PUSH TX FIFO Register \(DSPI_PUSHR\).”](#) TX FIFO entries can be removed from the TX FIFO only by being shifted out or by flushing the TX FIFO.

The TX FIFO counter field (TXCTR) in the DSPI status register (DSPIx_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI_PUSHHR is written or SPI data is transferred into the shift register from the TX FIFO. For more information on DSPIx_SR, refer to [Section 23.3.2.4, “DSPI Status Register \(DSPI_SR\).”](#)

The TXNXTPTR field indicates which TX FIFO entry will be transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPIx_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPIx_TXFR2 contains the SPI data and command for the next transfer. The TXNXTPTR field is incremented every time SPI data is transferred from the TX FIFO to the shift register.

23.4.3.4.1 Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPIx_PUSHHR. When the TX FIFO is not full, the TX FIFO fill flag (TFFF) in the DSPIx_SR is set. The TFFF bit is cleared when the TX FIFO is full and the eDMA controller indicates that a write to DSPIx_PUSHHR is complete or alternatively by host software writing a 1 to the TFFF in the DSPIx_SR. The TFFF can generate a DMA request or an interrupt request. See [Section 23.4.11.2, “Transmit FIFO Fill Interrupt or DMA Request \(TFFF\),”](#) for details.

The DSPI ignores attempts to push data to a full TX FIFO; that is, the state of the TX FIFO is unchanged. No error condition is indicated.

23.4.3.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO counter is decremented by one. At the end of a transfer, the TCF bit in the DSPIx_SR is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a 1 to the CLR_TXF bit in DSPIx_MCR.

If an external SPI bus master initiates a transfer with a DSPI slave while the slave’s DSPI TX FIFO is empty, the transmit FIFO underflow flag (TFUF) in the slave’s DSPIx_SR is set. See [Section 23.4.11.4, “Transmit FIFO Underflow Flag \(TFUF\),”](#) for details.

23.4.3.5 Receive First-In First-Out (RX FIFO) Buffering Mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds four received SPI data frames. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data is removed (popped) from the RX FIFO by reading the DSPIx_POPR register. RX FIFO entries can only be removed from the RX FIFO by reading the DSPIx_POPR or by flushing the RX FIFO. For more information on the DSPIx_POPR, refer to [Section 23.3.2.7, “DSPI POP RX FIFO Register \(DSPI_POPR\).”](#)

The RX FIFO counter field (RXCTR) in the DSPI status register (DSPIx_SR) indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPI_POPR is read or SPI data is copied from the shift register to the RX FIFO.

The POPNXTPTR field in the DSPLx_SR points to the RX FIFO entry that is returned when the DSPLx_POPR is read. The POPNXTPTR contains the positive, 32-bit word offset from DSPLx_RXFR0. For example, POPNXTPTR equal to two means that the DSPLx_RXFR2 contains the received SPI data that will be returned when DSPLx_POPR is read. The POPNXTPTR field is incremented every time the DSPLx_POPR is read. POPNXTPTR rolls over every four frames on the MCU.

23.4.3.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time an SPI frame is transferred to the RX FIFO the RX FIFO counter is incremented by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPLx_SR is asserted indicating an overflow condition. Depending on the state of the ROOE bit in the DSPLx_MCR, the data from the transfer that generated the overflow is ignored or shifted in to the shift register. If the ROOE bit is asserted, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored.

23.4.3.5.2 Draining the RX FIFO

Host software or the eDMA can remove (pop) entries from the RX FIFO by reading the DSPLx_POPR. For more information on DSPLx_POPR, refer to [Section 23.3.2.7, “DSPI POP RX FIFO Register \(DSPI_POPR\).”](#) A read of the DSPLx_POPR decrements the RX FIFO counter by one. Attempts to pop data from an empty RX FIFO are ignored, the RX FIFO counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

When the RX FIFO is not empty, the RX FIFO drain flag (RFDF) in the DSPLx_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the eDMA controller indicates that a read from DSPLx_POPR is complete; alternatively the RFDF bit can be cleared by the host writing a 1 to it.

23.4.4 Deserial Serial Interface (DSI) Configuration

The DSI configuration supports pin-count reduction by serializing eMIOS output channels or register bits and shifting them out in an SPI-like protocol. The timing and transfer protocol is described in [Section 23.4.8, “Transfer Formats.”](#) The received serial frames are converted to a parallel form (deserialized) and placed on the eMIOS input channels or in a register. See [Section 23.4.4.6, “DSPI_A Connectivity,”](#) for the source of the serialization data for each DSPI block.

The various features of the DSI configuration are set in the DSPLx_DSICR. For more information on the DSPLx_DSICR, refer to [Section 23.3.2.10, “DSPI DSI Configuration Register \(DSPI_DSICR\).”](#) The DSPI is in DSI configuration when the DCONF field in the DSPLx_MCR is 0b01.

The DSI frames can be from four to 16 bits long.

[Figure 23-18](#) shows an example of how a master DSPI connects to a SPI slave in DSI configuration.

23.4.4.1 DSI Master Mode

In DSI master mode the DSPI initiates and controls the DSI transfers. The DSI master has two different conditions that can initiate a transfer:

- Continuous
- Change in data

The two transfer initiation conditions are described in [Section 23.4.4.5, “DSI Transfer Initiation Control.”](#) Transfer attributes are set during initialization. The DSICTAS field in the DSPLx_DSICR determines which of the DSPLx_CTARs will control the transfer attributes.

23.4.4.2 DSI Slave Mode

In DSI slave mode the DSPI responds to transfers initiated by an SPI or DSI bus master. In this mode the DSPI does not initiate DSI transfers. Certain transfer attributes such as clock polarity and phase must be set for successful communication with a DSI master. The DSI slave mode transfer attributes are set in the DSPLx_CTAR1.

If the CID bit in the DSPLx_DSICR is set and the data in the DSPLx_COMPR differs from the selected source of the serialized data, the slave DSPI will assert the $\overline{\text{MTRIG}}$ signal.

23.4.4.3 DSI Serialization

In the DSI configuration, four to 16 bits can be serialized using two different sources. The TXSS bit in the DSPLx_DSICR selects between the DSPLx_SDR and DSPLx_AS DR as the source of serialized data. See [Section 23.3.2.11, “DSPI DSI Serialization Data Register \(DSPI_SDR\),”](#) and [Section 23.3.2.12, “DSPI DSI Alternate Serialization Data Register \(DSPI_AS DR\),”](#) for more details. The DSPLx_SDR holds the latest parallel input signal values which is sampled at every rising edge of the system clock. The DSPLx_AS DR is written by host software and used as an alternate source of serialized data.

A copy of the last DSI frame shifted out of the shift register is stored in the DSPLx_COMPR. This register provides added visibility for debugging and it serves as a reference for transfer initiation control. [Figure 23-18](#) shows the DSI serialization logic. [Section 23.3.2.13, “DSPI DSI Transmit Comparison Register \(DSPI_COMPR\),”](#) contains details on the DSPLx_COMPR.

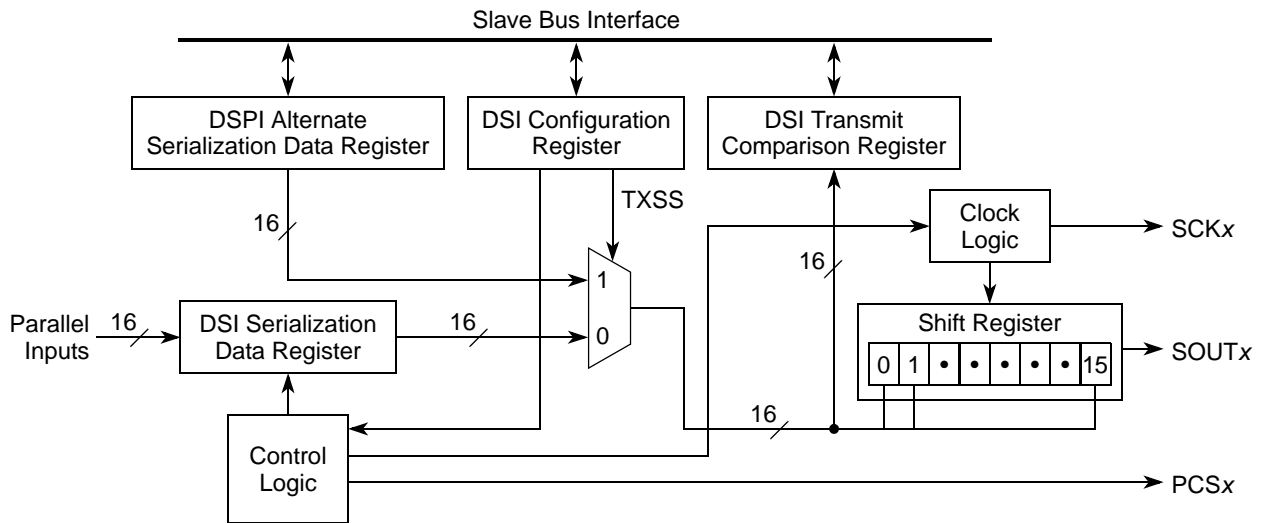
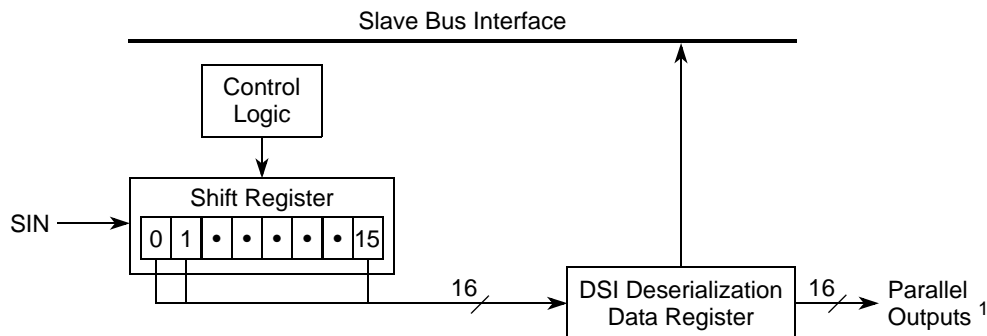


Figure 23-18. DSI Serialization Diagram

23.4.4.4 DSI Deserialization

When all bits in a DSI frame have been shifted in, the frame is copied to the DSPIx_DDR. This register presents the deserialized data as parallel output signal values. The DSPIx_DDR is memory mapped to allow host software to read the deserialized data directly. Figure 23-19 shows the DSI deserialization logic. For more information on the DSPIx_DDR, refer to Section 23.3.2.14, “DSPI DSI Deserialization Data Register (DSPI_DDR).”



¹ Parallel outputs not supported by DSPI_D.

Figure 23-19. DSI Deserialization Diagram

23.4.4.5 DSI Transfer Initiation Control

Data transfers for a master DSPI in DSI configuration are initiated by a condition. When chaining DSPIs, the master and all slaves must be configured for the transfer initiation. The transfer initiation conditions are selected by the CID bit in the DSPIx_DSICR. Table 23-17 lists the two transfer initiation conditions.

Table 23-17. DSI Data Transfer Initiation Control

DSPI _x _DSICR[CID]	Type of Transfer Initiation Control
0	Continuous
1	Change in Data

23.4.4.5.1 Continuous Control

For continuous control, the initiation of a transfer is based on the baud rate at which data is transferred between the DSPI and the external device. The baud rate is set in the DSPI_x_CTAR selected by the DSICTAS field in the DSPI_x_DSICR. A new DSI frame shifts out when the previous transfer cycle has completed and the delay after transfer (t_{DT}) has elapsed.

23.4.4.5.2 Change In Data Control

For change in data control, a transfer is initiated when the data to be serialized has changed since the transfer of the last DSI frame. A copy of the previously transferred DSI data is stored in the DSPI_x_COMPR. When the data in the DSPI_x_SDR or the DSPI_x_ASDR is different from the data in the DSPI_x_COMPR, a new DSI frame is transmitted. The TXSS bit in the DSPI_x_DSICR selects which register the DSPI_x_COMPR is compared to. The \overline{MTRIG} output signal is asserted every time a change in data is detected.

23.4.4.6 DSPI_A Connectivity

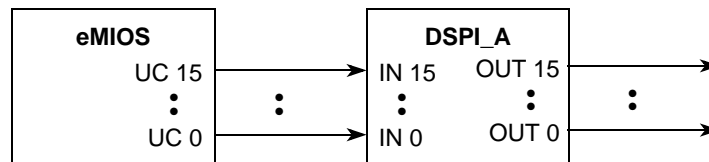


Figure 23-20. DSPI_A Connectivity

Table 23-18. DSPI_A Connectivity Table

DSPI_A Input	Connected to:	DSPI_A Output	Connected to:
0	eMIOS output channel 0	0	Set by SIU IMUX2. See Table 6-23 .
1	eMIOS output channel 1	1	Set by SIU IMUX2. See Table 6-23 .
2	eMIOS output channel 2	2	Set by SIU IMUX2. See Table 6-23 .
3	eMIOS output channel 3	3	Set by SIU IMUX2. See Table 6-23 .
4	eMIOS output channel 4	4	Set by SIU IMUX2. See Table 6-23 .
5	eMIOS output channel 5	5	Set by SIU IMUX2. See Table 6-23 .
6	eMIOS output channel 6	6	Set by SIU IMUX2. See Table 6-23 .
7	eMIOS output channel 7	7	Set by SIU IMUX2. See Table 6-23 .
8	eMIOS output channel 8	8	Set by SIU IMUX2. See Table 6-23 .

Table 23-18. DSPI_A Connectivity Table (continued)

DSPI_A Input	Connected to:	DSPI_A Output	Connected to:
9	eMIOS output channel 9	9	Set by SIU IMUX2. See Table 6-23 .
10	eMIOS output channel 10	10	Set by SIU IMUX2. See Table 6-23 .
11	eMIOS output channel 11	11	Set by SIU IMUX2. See Table 6-23 .
12	eMIOS output channel 12	12	Set by SIU IMUX2. See Table 6-23 .
13	eMIOS output channel 13	13	Set by SIU IMUX2. See Table 6-23 .
14	eMIOS output channel 14	14	Set by SIU IMUX2. See Table 6-23 .
15	eMIOS output channel 15	15	Set by SIU IMUX2. See Table 6-23 .

23.4.4.7 DSPI_B Connectivity

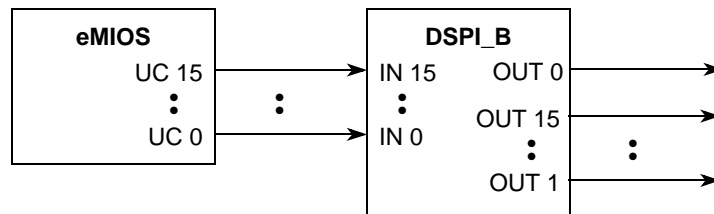


Figure 23-21. DSPI_B Connectivity

Table 23-19. DSPI_B Connectivity Table

DSPI_B Input	Connected to:	DSPI_B Output	Connected to:
0	eMIOS output channel 0	0	Set by SIU IMUX2. See Table 6-23 .
1	eMIOS output channel 1	1	Set by SIU IMUX2. See Table 6-23 .
2	eMIOS output channel 2	2	Set by SIU IMUX2. See Table 6-23 .
3	eMIOS output channel 3	3	Set by SIU IMUX2. See Table 6-23 .
4	eMIOS output channel 4	4	Set by SIU IMUX2. See Table 6-23 .
5	eMIOS output channel 5	5	Set by SIU IMUX2. See Table 6-23 .
6	eMIOS output channel 6	6	Set by SIU IMUX2. See Table 6-23 .
7	eMIOS output channel 7	7	Set by SIU IMUX2. See Table 6-23 .
8	eMIOS output channel 8	8	Set by SIU IMUX2. See Table 6-23 .
9	eMIOS output channel 9	9	Set by SIU IMUX2. See Table 6-23 .
10	eMIOS output channel 10	10	Set by SIU IMUX2. See Table 6-23 .
11	eMIOS output channel 11	11	Set by SIU IMUX2. See Table 6-23 .
12	eMIOS output channel 12	12	Set by SIU IMUX2. See Table 6-23 .
13	eMIOS output channel 13	13	Set by SIU IMUX2. See Table 6-23 .

Table 23-19. DSPI_B Connectivity Table (continued)

DSPI_B Input	Connected to:	DSPI_B Output	Connected to:
14	eMIOS output channel 14	14	Set by SIU IMUX2. See Table 6-23 .
15	eMIOS output channel 15	15	Set by SIU IMUX2. See Table 6-23 .

23.4.4.8 DSPI_C Connectivity

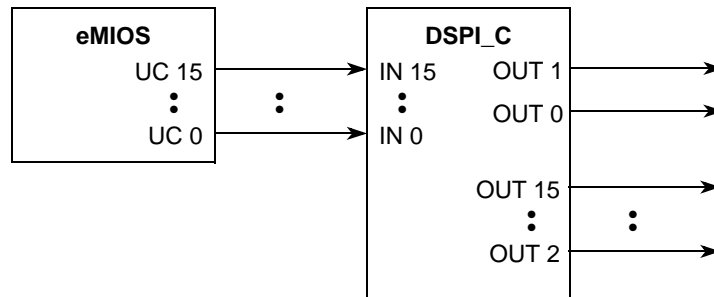


Figure 23-22. DSPI_C Connectivity

Table 23-20. DSPI_C Connectivity Table

DSPI_C Input	Connected to:	DSPI_C Output	Connected to:
0	eMIOS output channel 0	0	Set by SIU IMUX2. See Table 6-23 .
1	eMIOS output channel 1	1	Set by SIU IMUX2. See Table 6-23 .
2	eMIOS output channel 2	2	Set by SIU IMUX2. See Table 6-23 .
3	eMIOS output channel 3	3	Set by SIU IMUX2. See Table 6-23 .
4	eMIOS output channel 4	4	Set by SIU IMUX2. See Table 6-23 .
5	eMIOS output channel 5	5	Set by SIU IMUX2. See Table 6-23 .
6	eMIOS output channel 6	6	Set by SIU IMUX2. See Table 6-23 .
7	eMIOS output channel 7	7	Set by SIU IMUX2. See Table 6-23 .
8	eMIOS output channel 8	8	Set by SIU IMUX2. See Table 6-23 .
9	eMIOS output channel 9	9	Set by SIU IMUX2. See Table 6-23 .
10	eMIOS output channel 10	10	Set by SIU IMUX2. See Table 6-23 .
11	eMIOS output channel 11	11	Set by SIU IMUX2. See Table 6-23 .
12	eMIOS output channel 12	12	Set by SIU IMUX2. See Table 6-23 .
13	eMIOS output channel 13	13	Set by SIU IMUX2. See Table 6-23 .
14	eMIOS output channel 14	14	Set by SIU IMUX2. See Table 6-23 .
15	eMIOS output channel 15	15	Set by SIU IMUX2. See Table 6-23 .

23.4.4.9 DSPI_D Connectivity

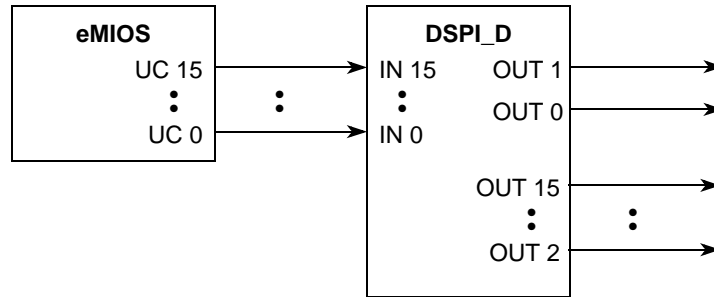


Figure 23-23. DSPI_D Connectivity

Table 23-21. DSPI_D Connectivity Table

DSPI_D Input	Connected to:
0	eMIOS output channel 0
1	eMIOS output channel 1
2	eMIOS output channel 2
3	eMIOS output channel 3
4	eMIOS output channel 4
5	eMIOS output channel 5
6	eMIOS output channel 6
7	eMIOS output channel 7
8	eMIOS output channel 8
9	eMIOS output channel 9
10	eMIOS output channel 10
11	eMIOS output channel 11
12	eMIOS output channel 12
13	eMIOS output channel 13
14	eMIOS output channel 14
15	eMIOS output channel 15

23.4.5 Combined Serial Interface (CSI) Configuration

In master mode, the CSI configuration of the DSPI is used to support SPI and DSI functions on a frame by frame basis. CSI configuration allows interleaving of DSI data frames from the parallel input signals (from the eMIOS) with SPI commands and data from the TX FIFO. The data returned from the bus slave is either used to drive the parallel output signals (to the eMIOS) or is stored in the RX FIFO. CSI configuration

allows serialized data and configuration or diagnostic data to be transferred to a slave device using only one serial link. The DSPI is in CSI configuration when the DCONF field in the DSPI_x_MCR is 0b10. [Figure 23-24](#) shows an example of how a DSPI can be used with a deserializing peripheral that supports SPI control for control and diagnostic frames.

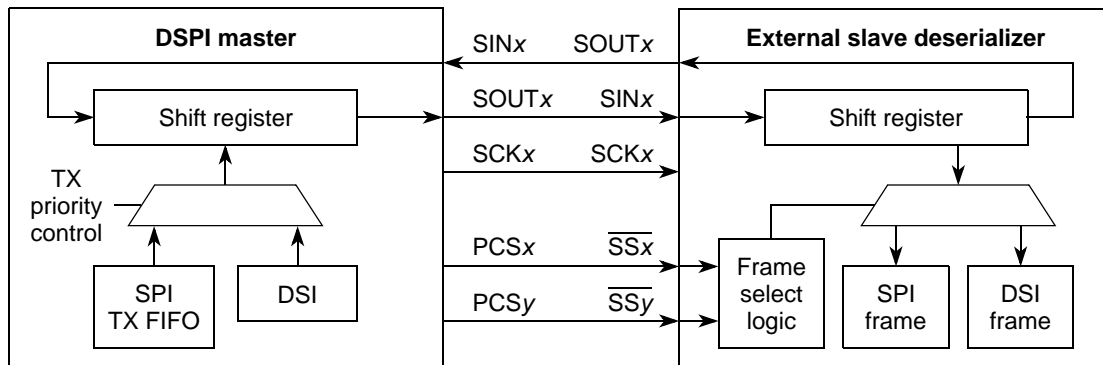


Figure 23-24. Example of System Using DSPI in CSI Configuration

In CSI configuration the DSPI transfers DSI data based on [Section 23.4.4.5, “DSI Transfer Initiation Control.”](#) When there are SPI commands in the TX FIFO, the SPI data has priority over the DSI frames. When the TX FIFO is empty, DSI transfer resumes.

Two peripheral chip select signals indicate whether DSI data or SPI data is transmitted. The user must configure the DSPI so the two CTARs associated with DSI data and SPI data assert different peripheral chip-select signals denoted in the figure as PCS_x and PCS_y. The CSI configuration is only supported in master mode.

Data returned from the external slave while a DSI frame is transferred is placed on the parallel output signals. Data returned from the external slave while an SPI frame is transferred is moved to the RX FIFO. The TX FIFO and RX FIFO are fully functional in CSI mode.

23.4.5.1 CSI Serialization

Serialization in the CSI configuration is similar to serialization in DSI configuration. The transfer attributes for SPI frames are determined by the DSPI_x_CTAR selected by the CTAS field in the SPI command halfword. The transfer attributes for the DSI frames are determined by the DSPI_x_CTAR selected by the DSICTAS field in the DSPI_x_DSICR. [Figure 23-25](#) shows the CSI serialization logic.

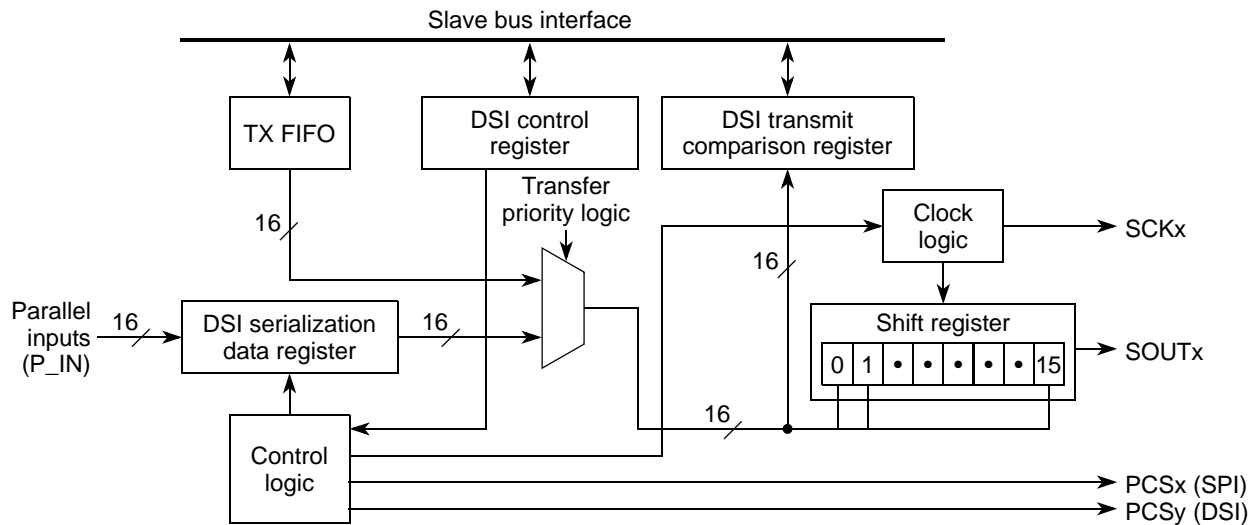


Figure 23-25. CSI Serialization Diagram

The parallel inputs signal states are latched into the DSPI_x_SDR on the rising edge of every system clock and serialized based on the transfer initiation control settings in the DSPI_x_DSICR. For more information on the DSPI_x_SDR, refer to [Section 23.3.2.11, “DSPI DSI Serialization Data Register \(DSPI_SDR\).”](#) SPI frames written to the TX FIFO have priority over DSI data from the DSPI_x_SDR and are transferred at the next frame boundary. A copy of the most recently transferred DSI frame is stored in the DSPI_x_COMPR. The transfer priority logic selects the source of the serialized data and asserts the appropriate \overline{CS} signal.

23.4.5.2 CSI Deserialization

The deserialized frames in CSI configuration go into the DSPI_x_SDR or the RX FIFO based on the transfer priority logic. When DSI frames are transferred the returned frames are deserialized and latched into the DSPI_x_DDR. When SPI frames are transferred the returned frames are deserialized and written to the RX FIFO. [Figure 23-26](#) shows the CSI deserialization logic.

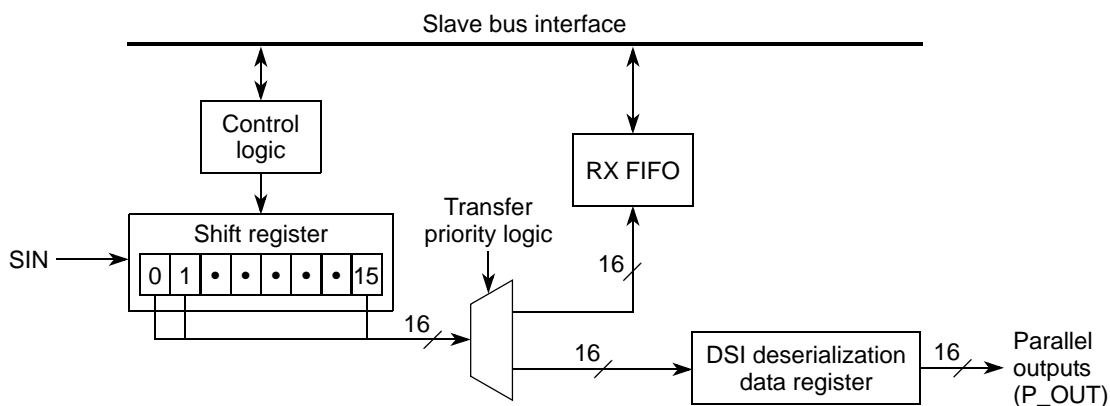


Figure 23-26. CSI Deserialization Diagram

23.4.6 Buffered SPI Operation

The DSPI can use a FIFO buffering mechanism to transmit and receive commands and data to and from external devices. The transmit FIFO buffers SPI commands and data to be transferred. The receive FIFO buffers incoming serial data. Both FIFOs are four entries deep. The TX FIFO stores 32-bit words when the DSPIs are configured for master mode. The 32-bit words are composed of 16-bit command fields and data fields up to 16 bits wide. The RX FIFOs store 16-bit words of received data from external devices. When the DSPI is configured for slave mode, the DSPI ignores the SPI command in the TX FIFO. See the DSPI block guide for a complete description of the command portion of the TX FIFO.

For queued operations, the SPI queues reside in system memory external to the DSPI. Data transfers between the memory and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software. See [Figure 23-27](#) for conceptual diagram of the queue data transfer control in the MCU.

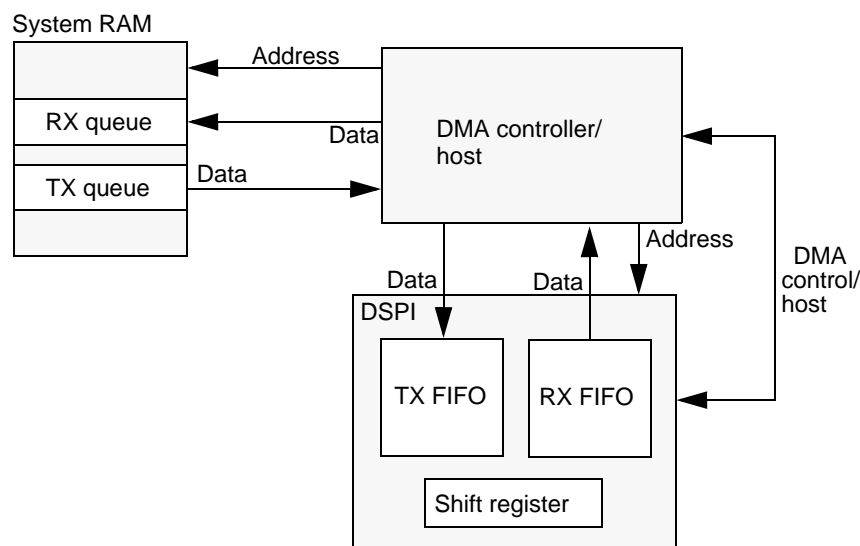


Figure 23-27. DSPI Queue Transfer Control in MPC5510

23.4.7 DSPI Baud Rate and Clock Delay Generation

The SCK_x frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option of doubling the baud rate. [Figure 23-28](#) shows conceptually how the SCK signal is generated.

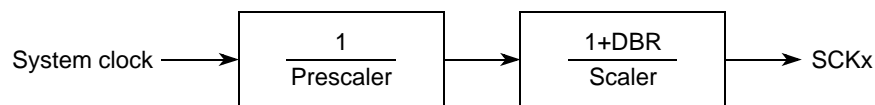


Figure 23-28. Communications Clock Prescalers and Scalers

23.4.7.1 Baud Rate Generator

The baud rate is the frequency of the serial communication clock (SCK_x). The system clock is divided by a baud rate prescaler (defined by DSPI_x_CTAR[PBR]) and baud rate scaler (defined by DSPI_x_CTAR[BR]) to produce SCK_x with the possibility of doubling the baud rate. The DBR, PBR, and BR fields in the DSPI_x_CTARs select the frequency of SCK_x using the following formula:

$$\text{SCK baud rate} = \frac{f_{\text{SYS}}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$$

Table 23-22 shows an example of a computed baud rate.

Table 23-22. Baud Rate Computation Example

f _{SYS}	PBR	Prescaler Value	BR	Scaler Value	DBR Value	Baud Rate
66 MHz	0b00	2	0b0000	2	0	16.67 Mb/s
20 MHz	0b00	2	0b0000	2	1	10 Mb/s

23.4.7.2 PCS to SCK Delay (t_{csc})

The PCS_x to SCK_x delay is the length of time from assertion of the PCS_x signal to the first SCK_x edge. See Figure 23-30 for an illustration of the PCS_x to SCK_x delay. The PCSSCK and CSSCK fields in the DSPI_x_CTAR_n registers select the PCS_x to SCK_x delay, and the relationship is expressed by the following formula:

$$t_{\text{csc}} = \frac{1}{f_{\text{SYS}}} \times \text{PCSSCK} \times \text{CSSCK}$$

Table 23-23 shows an example of the computed PCS to SCK delay.

Table 23-23. PCS to SCK Delay Computation Example

PCSSCK	Prescaler Value	CSSCK	Scaler Value	f _{SYS}	PCS to SCK Delay
0b01	3	0b0100	32	66 MHz	1.44 μs

23.4.7.3 After SCK Delay (t_{asc})

The after SCK_x delay is the length of time between the last edge of SCK_x and the negation of PCS_x. See Figure 23-30 and Figure 23-31 for illustrations of the after SCK_x delay. The PASC and ASC fields in the DSPI_x_CTAR_n registers select the after SCK delay. The relationship between these variables is given in the following formula:

$$t_{\text{asc}} = \frac{1}{f_{\text{SYS}}} \times \text{PASC} \times \text{ASC}$$

Table 23-24 shows an example of the computed after SCK delay.

Table 23-24. After SCK Delay Computation Example

PASC	Prescaler Value	ASC	Scaler Value	Fsys	After SCK Delay
0b01	3	0b0100	32	66 MHz	1.44 μ s

23.4.7.4 Delay after Transfer (t_{DT})

The delay after transfer is the length of time between negation of the PCS x signal for a frame and the assertion of the PCS x signal for the next frame. See Figure 23-30 for an illustration of the delay after transfer. The PDT and DT fields in the DSPI x _CTAR n registers select the delay after transfer. The following formula expresses the PDT/DT/delay after transfer relationship:

$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$

Table 23-25 shows an example of the computed delay after transfer.

Table 23-25. Delay after Transfer Computation Example

PDT	Prescaler Value	DT	Scaler Value	f _{SYS}	Delay after Transfer
0b01	3	0b1110	32768	66 Hz	1.47 ms

23.4.7.5 Peripheral Chip Select Strobe Enable (\overline{PCSS})

The \overline{PCSS} signal provides a delay to allow the PCS x signals to settle after transitioning thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI x _MCR, \overline{PCSS} provides a signal for an external demultiplexer to decode the PCS x [0:4] signals into as many as 32 glitch-free PCS x signals. Figure 23-29 shows the timing of the \overline{PCSS} signal relative to PCS signals.

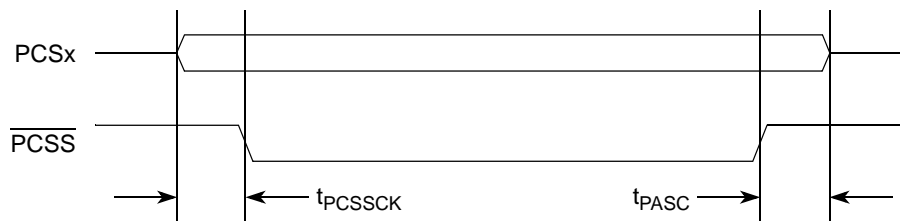


Figure 23-29. Peripheral Chip Select Strobe Timing

The delay between the assertion of the PCS x signals and the assertion of \overline{PCSS} is selected by the PCSSCK field in the DSPI x _CTAR based on the following formula:

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between \overline{PCSS} negation and PCS x negation is selected by the PASC field in the DSPI x _CTAR based on the following formula:

$$t_{\text{PASC}} = \frac{1}{f_{\text{SYS}}} \times \text{PASC}$$

Table 23-26 shows an example of the computed t_{PCSSCK} delay.

Table 23-26. Peripheral Chip Select Strobe Assert Computation Example

PCSSCK	Prescaler	f _{SYS}	Delay before Transfer
0b11	7	66 MHz	105.0 ns

Table 23-27 shows an example of the computed the t_{PASC} delay.

Table 23-27. Peripheral Chip Select Strobe Negate Computation Example

PASC	Prescaler	f _{SYS}	Delay after Transfer
0b11	7	66 MHz	105.0 ns

23.4.8 Transfer Formats

The SPI serial communication is controlled by the serial communications clock (SCK_x) signal and the PCS_x signals. The SCK_x signal provided by the master device synchronizes shifting and sampling of the data by the SIN_x and SOUT_x pins. The PCS_x signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI clock and transfer attributes registers (DSPI_x_CTAR_n) select the polarity and phase of the serial clock, SCK_x. The polarity bit selects the idle state of the SCK_x. The clock phase bit selects if the data on SOUT_x is valid before or on the first SCK_x edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI_x_CTAR₀ (SPI slave mode) or DSPI_x_CTAR₁ (DSI slave mode) select the polarity and phase of the serial clock. Even though the bus slave does not control the SCK signal, clock polarity, clock phase, and number of bits to transfer must be identical for the master device and the slave device to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI_x_MCR selects between classic SPI format and modified transfer format. The classic SPI formats are described in Section 23.4.8.1, “Classic SPI Transfer Format (CPHA = 0),” and Section 23.4.8.2, “Classic SPI Transfer Format (CPHA = 1).” The modified transfer formats are described in Section 23.4.8.3, “Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 0),” and Section 23.4.8.4, “Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 1).”

In the SPI and DSI configurations, the DSPI provides the option of keeping the PCS signals asserted between frames. See Section 23.4.8.5, “Continuous Selection Format” for details.

23.4.8.1 Classic SPI Transfer Format (CPHA = 0)

The transfer format shown in Figure 23-30 is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their SIN_x pins on the odd-numbered SCK_x edges and change the data on their SOUT_x pins on the even-numbered SCK_x edges.

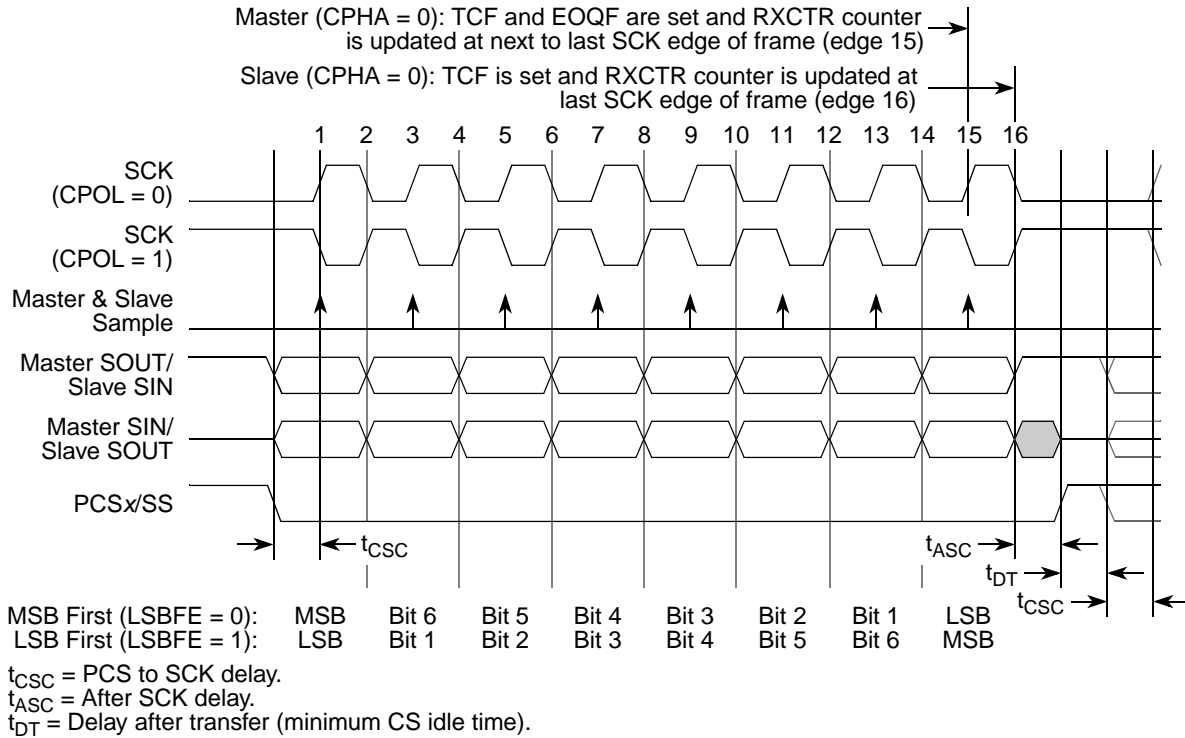


Figure 23-30. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

The master initiates the transfer by placing its first data bit on the SOUT_x pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT_x pin. After the t_{CSC} delay has elapsed, the master outputs the first edge of SCK_x. This is the edge used by the master and slave devices to sample the first input data bit on their serial data input signals. At the second edge of the SCK_x the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN_x pins on the odd-numbered clock edges and changes the data on their SOUT_x pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For the CPHA = 0 condition of the master, TCF and EOQF are set and the RXCTR counter is updated at the next to last serial clock edge of the frame (edge 15) of Figure 23-30.

For the CPHA=0 condition of the slave, TCF is set and the RXCTR counter is updated at the last serial clock edge of the frame (edge 16) of Figure 23-30.

23.4.8.2 Classic SPI Transfer Format (CPHA = 1)

This transfer format shown in Figure 23-31 is used to communicate with peripheral SPI slave devices that require the first SCK_x edge before the first data bit becomes available on the slave SOUT pin. In this format the master and slave devices change the data on their SOUT_x pins on the odd-numbered SCK_x edges and sample the data on their SIN_x pins on the even-numbered SCK_x edges.

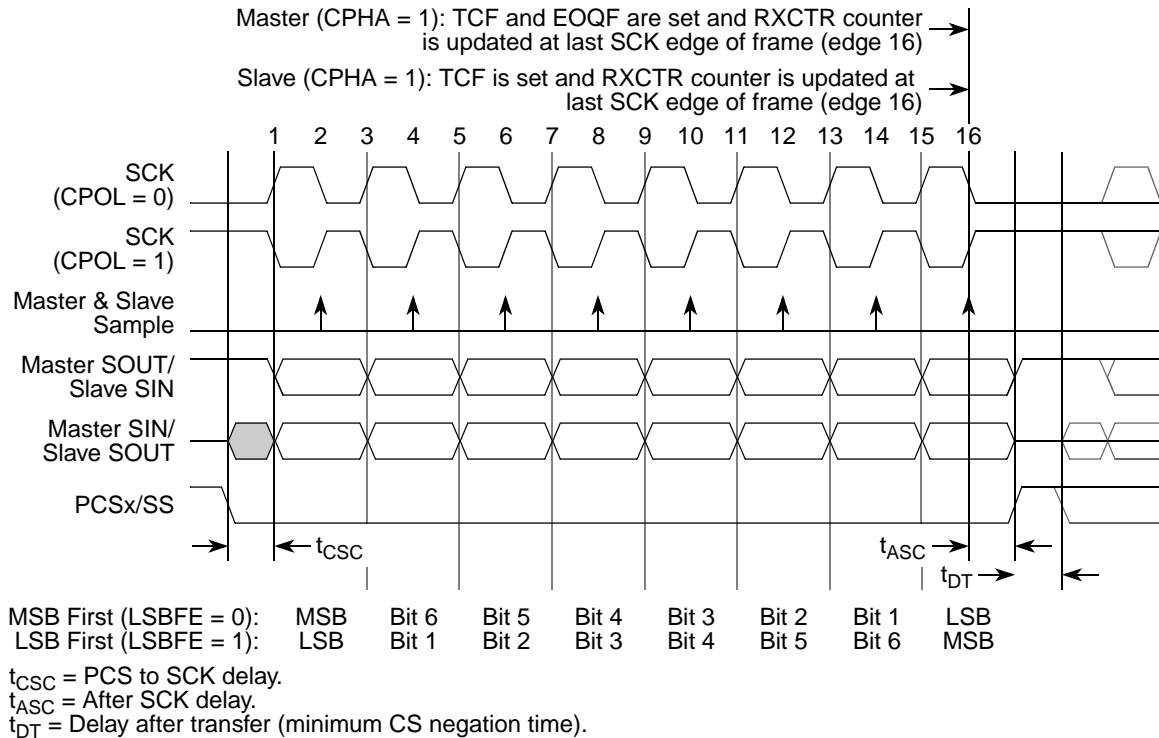


Figure 23-31. DSPI Transfer Timing Diagram (MTEF = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the PCS_x signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first SCK_x edge and at the same time places valid data on the master SOUT_x pin. The slave responds to the first SCK_x edge by placing its first data bit on its slave SOUT_x pin.

At the second edge of the SCK_x the master and slave sample their SIN_x pins. For the rest of the frame the master and the slave change the data on their SOUT_x pins on the odd-numbered clock edges and sample their SIN_x pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS_x signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For CPHA=1 the master EOQF and TCF and slave TCF are set at the last serial clock edge (edge 16) of Figure 23-31. For CPHA=1 the master and slave RXCTR counters are updated on the same clock edge.

23.4.8.3 Modified SPI/DSI Transfer Format (MTEF = 1, CPHA = 0)

In this modified transfer format both the master and the slave sample later in the SCK period than in classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

NOTE

For correct operation of the modified transfer format, the user must thoroughly analyze the SPI link timing budget.

The master and the slave place data on the SOUT_x pins at the assertion of the PCS_x signal. After the PCS_x to SCK_x delay has elapsed the first SCK_x edge is generated. The slave samples the master SOUT_x signal on every odd numbered SCK_x edge. The slave also places new data on the slave SOUT_x on every odd numbered clock edge.

The master places its second data bit on the SOUT_x line one system clock after odd numbered SCK_x edge. The point where the master samples the slave SOUT_x is selected by writing to the SMPL_PT field in the DSPI_x_MCR. Table 23-28 lists the number of system clock cycles between the active edge of SCK_x and the master sample point for different values of the SMPL_PT bit field. The master sample point can be delayed by one or two system clock cycles.

Table 23-28. Delayed Master Sample Point

SMPL_PT	Number of System Clock Cycles Between Odd-Numbered Edge of SCK and Sampling of SIN
00	0
01	1
10	2
11	Reserved

Figure 23-32 shows the modified transfer format for CPHA = 0. Only the condition where CPOL = 0 is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.

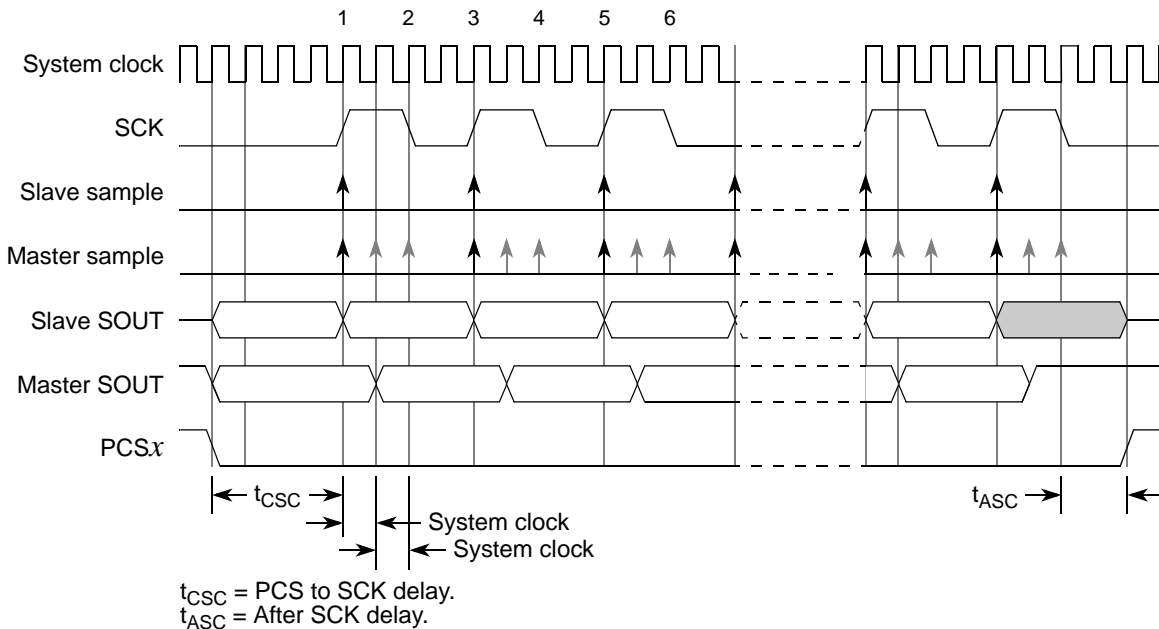


Figure 23-32. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, F_{sck} = F_{sys}/4)

23.4.8.4 Modified SPI/DSI Transfer Format (MTFE = 1, CPHA = 1)

Figure 23-33 shows the modified transfer format for CPHA = 1. Only the condition where CPOL = 0 is described. At the start of a transfer the DSPI asserts the PCS signal to the slave device. After the PCS to SCK delay has elapsed, the master and the slave put data on their SOUT pins at the first edge of SCK. The slave samples the master SOUT signal on the even numbered edges of SCK. The master samples the slave SOUT signal on the odd numbered SCK edges starting with the third SCK edge. The slave samples the last bit on the last edge of the SCK. The master samples the last slave SOUT bit one half SCK cycle after the last edge of SCK. No clock edge will be visible on the master SCK pin during the sampling of the last bit. The SCK to PCS delay must be greater or equal to half of the SCK period.

NOTE

For correct operation of the modified transfer format, the user must thoroughly analyze the SPI link timing budget.

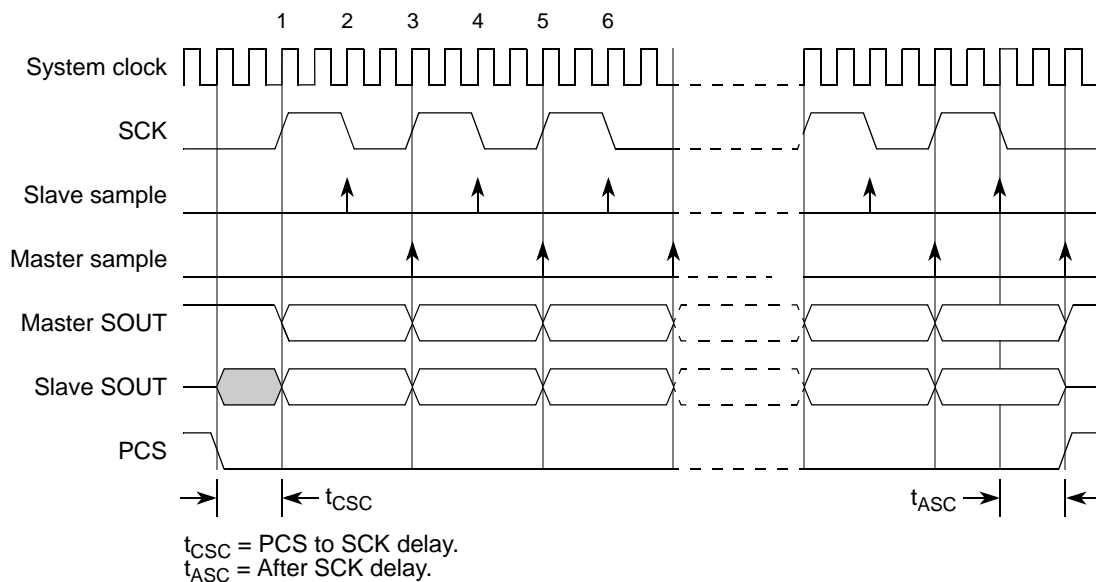


Figure 23-33. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, F_{sck} = F_{sys}/4)

23.4.8.5 Continuous Selection Format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The continuous selection format provides the flexibility to handle both cases. The continuous selection format is enabled for the SPI configuration by setting the CONT bit in the SPI command. Continuous selection is enabled for the DSI configuration by setting the DCONT bit in the DSPI_x_DSICR. The behavior of the PCS signals in the two configurations is identical so only SPI configuration will be described.

When the CONT bit = 0, the DSPI drives the asserted chip select signals to their idle states in between frames. The idle states of the chip select signals are selected by the PCSIS field in the DSPI_x_MCR.

Figure 23-34 shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 0.

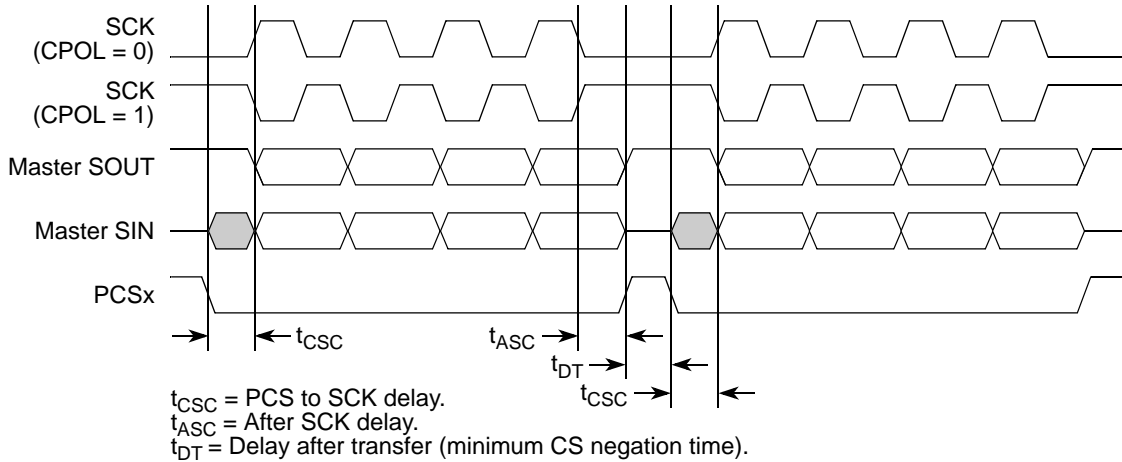


Figure 23-34. Example of Non-Continuous Format (CPHA=1, CONT=0)

When the CONT = 1 and the PCS signal for the next transfer is the same as for the current transfer, the PCS signal remains asserted for the duration of the two transfers. The delay between transfers (t_{DT}) is not inserted between the transfers. Figure 23-35 shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 1.

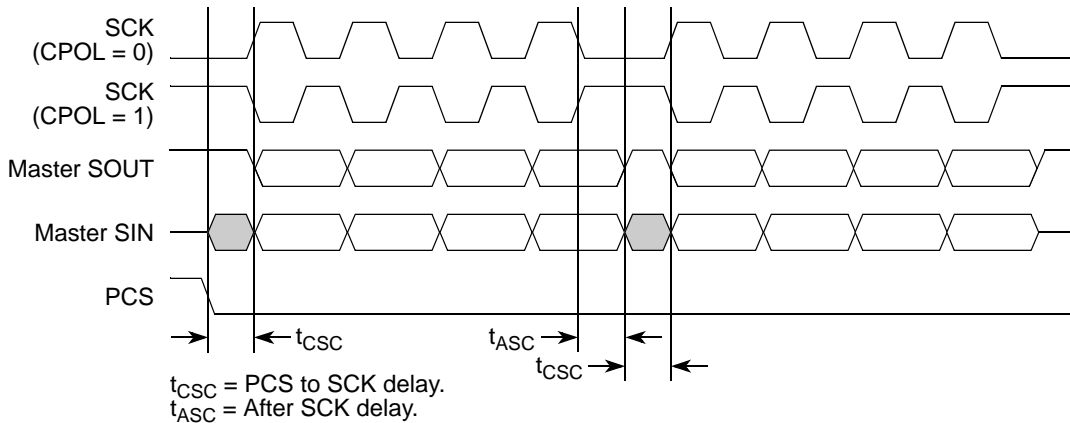


Figure 23-35. Example of Continuous Transfer (CPHA = 1, CONT = 1)

In Figure 23-35, the period length at the start of the next transfer is the sum of t_{ASC} and t_{CSC} ; i.e., it does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations, t_{ASC} and t_{CSC} must be increased if a full half-clock period is required.

Switching CTARs between frames while using continuous selection can cause errors in the transfer. The PCS signal must be negated before CTAR is switched.

When the CONT bit = 1 and the PCS signals for the next transfer are different from the present transfer, the PCS signals behave as if the CONT bit was not set.

23.4.8.6 Clock Polarity Switching Between DSPI Transfers

If it is desired to switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the assertion of the chip select for the next frame. In [Figure 23-36](#), time A shows the one clock interval. Time B is user programmable from a minimum of two system clocks. Refer to [Section 23.3.2.3, “DSPI Clock and Transfer Attributes Registers 0–7 \(DSPI_CTARn\).”](#)

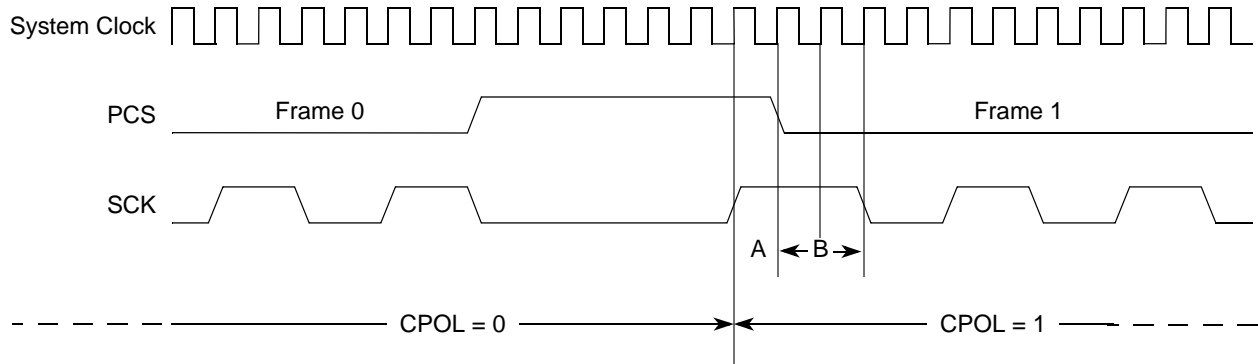


Figure 23-36. Polarity Switching Between Frames

23.4.9 Continuous Serial Communications Clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting the CONT_SCKE bit in the DSPIx_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Setting CPHA = 0 will be ignored if the CONT_SCKE bit is set. Continuous SCK is supported for modified transfer format.

Clock and transfer attributes for the continuous SCK mode are set according to the following rules:

- When the DSPI is in SPI configuration, CTAR0 shall be used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame will be used.
- When the DSPI is in DSI configuration, the CTAR specified by the DSICTAS field will be used at all times.
- When the DSPI is in CSI configuration, the CTAR selected by the DSICTAS field will be used initially. At the start of an SPI frame transfer, the CTAR specified by the CTAS value for the frame will be used. At the start of a DSI frame transfer, the CTAR specified by the DSICTAS field will be used.
- In all configurations, the currently selected CTAR will remain in use until the start of a frame with a different CTAR specified, or the continuous SCK mode is terminated.

It is recommended that the baud rate is the same for all transfers made while using the continuous SCK. Switching clock polarity between frames while using continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into module disable mode.

Enabling continuous SCK disables the PCS to SCK delay and the after SCK delay. The delay after transfer is fixed at one SCK cycle. [Figure 23-37](#) shows timing diagram for continuous SCK format with continuous selection disabled.

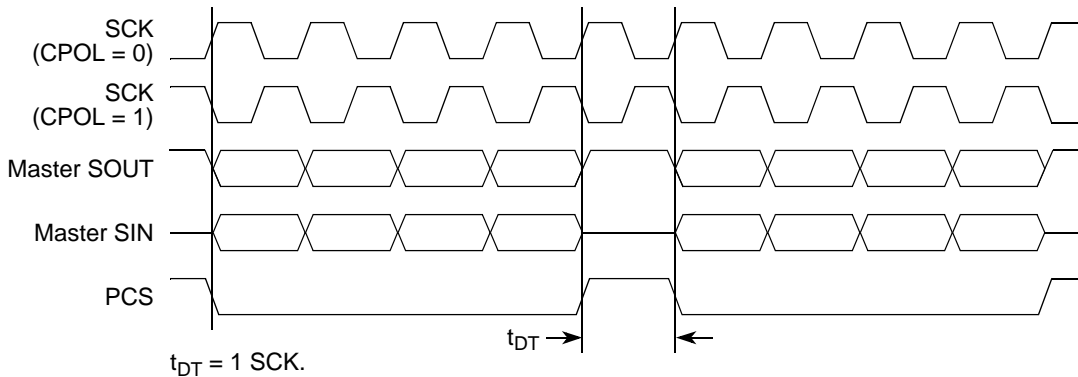


Figure 23-37. Continuous SCK Timing Diagram (CONT= 0)

If the CONT bit in the TX FIFO entry is set or the DCONT in the DSPLx_DSICR is set, PCS remains asserted between the transfers when the PCS signal for the next transfer is the same as for the current transfer. [Figure 23-38](#) shows timing diagram for continuous SCK format with continuous selection enabled.

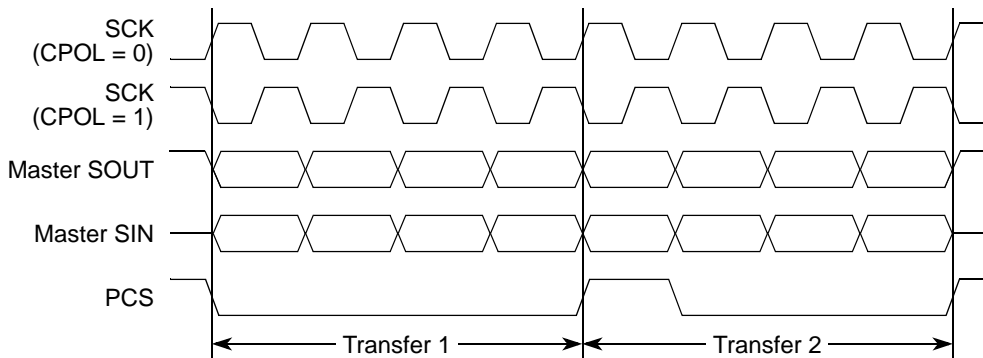


Figure 23-38. Continuous SCK Timing Diagram (CONT=1)

23.4.10 Peripheral Chip Select Expansion and Deglitching

The DSPI supports up to 64 peripheral chip select signals with the use of an external demultiplexer. Up to 32 peripheral chip select signals can be used if deglitching is desired. The $\overline{\text{PCSS}}$ signal provides the appropriate timing to enable and disable the demultiplexer for the PCS[0:4] signals.

[Figure 23-39](#) shows how an external 5-to-32 demultiplexer (decoder) can be connected to the DSPI.

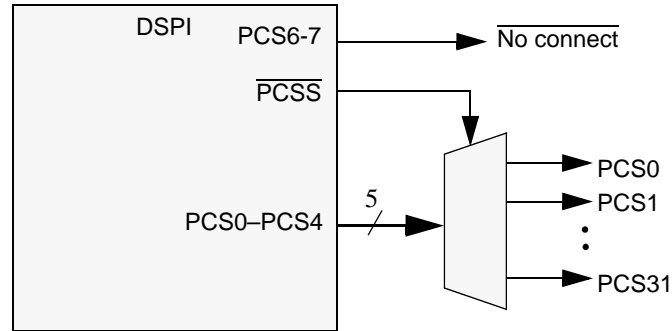


Figure 23-39. DSPI PCS Expansion and Deglitching

23.4.11 DMA and Interrupt Conditions

The DSPI has six conditions that can generate interrupt requests only and two conditions that can generate interrupt or DMA request. Table 23-29 lists the conditions. The x indicates which signals are connected.

Table 23-29. DSPI Interrupt and DMA Request Conditions

Condition	Flag	Interrupt	DMA
End of queue reached (EOQ)	EOQF	X	
TX FIFO is not full	TFFF	X	X
Transfer of current frame complete	TCF	X	
Attempt to transmit with an empty transmit FIFO	TFUF	X	
RX FIFO is not empty	RFDF	X	X
Frame received while receive FIFO is full	RFOF	X	

All request conditions are detected in the SPI configuration and in the CSI configuration. In DSI configuration only the transfer of current frame complete condition is detected. See Table 13-4 for the DSPI DMA channel assignments and Section 8.3.1, “Interrupt Source Summary Table,” for the DSPI interrupt vectors.

23.4.11.1 End of Queue Interrupt Request (EOQF)

The end of queue request indicates that the end of a transmit queue is reached. The end of queue request is generated when the EOQ bit in the executing SPI command is asserted and the EOQF_RE bit in the DSPI_x_RSER is asserted. See the EOQ bit description in Section 23.3.2.4, “DSPI Status Register (DSPI_SR).” Refer to Figure 23-30 and Figure 23-31 that illustrate when EOQF is set.

23.4.11.2 Transmit FIFO Fill Interrupt or DMA Request (TFFF)

The transmit FIFO fill request indicates that the TX FIFO is not full. The transmit FIFO fill request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the TFFF_RE bit in the DSPI_x_RSER is asserted. The TFFF_DIRS bit in the DSPI_x_RSER selects whether a DMA request or an interrupt request is generated.

23.4.11.3 Transfer Complete Interrupt Request (TCF)

The transfer complete request indicates the end of the transfer of a serial frame. The transfer complete request is generated at the end of each frame transfer when the TCF_RE bit is set in the DSPIx_RSER. See the TCF bit description in [Section 23.3.2.4, “DSPI Status Register \(DSPI_SR\).”](#) Refer to [Figure 23-30](#) and [Figure 23-31](#) that illustrate when TCF is set.

23.4.11.4 Transmit FIFO Underflow Flag (TFUF)

The Transmit FIFO Underflow Flag indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI blocks operating in slave mode and SPI configuration. The transmit underflow condition is detected when the TX FIFO of a DSPI operating as a SPI slave is empty, and a transfer is initiated from an external SPI master.

23.4.11.5 Receive FIFO Drain Interrupt or DMA Request (RFDF)

The receive FIFO drain request indicates that the RX FIFO is not empty. The receive FIFO drain request is generated when the number of entries in the RX FIFO is not zero, and the RFDF_RE bit in the DSPIx_RSER is asserted. The RFDF_DIRS bit in the DSPIx_RSER selects whether a DMA request or an interrupt request is generated.

23.4.11.6 Receive FIFO Overflow Flag (RFOF)

The receive FIFO overflow flag indicates that an overflow condition in the RX FIFO has occurred, and that data may be lost. The receive FIFO overflow flag is asserted when the RX FIFO is full, a new frame has been received in the shift register, and a transfer is initiated.

23.4.11.7 DMA Requests

The connection of the DSPI DMA request signals to the DMA channel mux is described in [Table 13-4](#).

23.4.11.8 Interrupt Requests

The DSPI interrupts on connected as described in [Table 8-2](#).

23.4.12 Power Saving Features

The DSPI supports three power-saving strategies:

- Halt mode
- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

23.4.12.1 Halt Mode

By setting the appropriate bit in the SIU_HLT register, a request is made to shut down all clocks in the DSPI. If there is no serial transfer in progress, the DSPI immediately asserts an acknowledge signal to the system, allowing the clocks to be disabled. If a serial transfer is in progress when the request is received,

the DSPI waits until it reaches a frame boundary before it asserts the acknowledge signal to the system. The status of this acknowledge signal can be determined by reading the SIU_HLTACK register.

While the clocks are shut off, the DSPI memory-mapped logic is not accessible. The states of the interrupt and DMA request signals cannot be changed while in Halt Mode.

Halt Mode is exited by negating the appropriate bit in the SIU_HLT register.

23.4.12.2 Module Disable Mode

Module disable mode is a module-specific mode that the DSPI can enter to save power. Host software can initiate the module disable mode by writing a 1 to the MDIS bit in the DSPIx_MCR.

In module disable mode, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different affect when the DSPI is in the module disable mode. Reading the RX FIFO pop register will not change the state of the RX FIFO. Likewise, writing to the TX FIFO push register will not change the state of the TX FIFO. Clearing either of the FIFOs will not have any affect in the module disable mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPIx_MCR will not have any affect in the module disable mode. In the module disable mode, all status bits and register flags in the DSPI will return the correct values when read, but writing to them will have no affect. Writing to the DSPIx_TCR during module disable mode will not have any affect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

23.4.12.3 Slave Interface Signal Gating

The DSPI's module enable signal is used to gate slave interface signals such as address, byte enable, read/write and data. This prevents toggling slave interface signals from consuming power unless the DSPI is accessed.

23.5 Initialization/Application Information

23.5.1 How to Change Queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI configuration. This section presents an example of how to change queues for the DSPI.

1. The last command word from a queue is executed. The EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag (EOQF) in the DSPIx_SR is set.
3. The setting of the EOQF flag will disable both serial transmission, and serial reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is negated to indicate the STOPPED state.
4. The eDMA will continue to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the eDMA controller.

6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the RXCNT in DSPIx_SR or by checking RFDF in the DSPIx_SR after each read operation of the DSPIx_POPR.
7. Modify DMA descriptor of TX and RX channels for new queues.
8. Flush TX FIFO by writing a 1 to the CLR_TXF bit in the DSPIx_MCR, Flush RX FIFO by writing a 1 to the CLR_RXF bit in the DSPIx_MCR.
9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to SPI_TCNT field in the DSPIx_TCR.
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

23.5.2 Baud Rate Settings

Table 23-30 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPIx_CTARs. The values calculated assume a 66 MHz system frequency.

Table 23-30. Baud Rate Values

		Baud Rate Divider Prescaler Values (DSPI_CTAR[PBR])			
		2	3	5	7
Baud Rate Scaler Values (DSPI_CTAR[BR])	2	16.67 MHz	11.11 MHz	6.67 MHz	4.76 MHz
	4	8.33 MHz	5.55 MHz	3.33 MHz	2.38 MHz
	6	5.55 MHz	3.70 MHz	2.22 MHz	1.59 MHz
	8	4.17 MHz	2.78 MHz	1.67 MHz	1.19 MHz
	16	2.08 MHz	1.37 MHz	833.3 kHz	595.3 kHz
	32	1.04 MHz	693 kHz	417 kHz	297 kHz
	64	521 kHz	347 kHz	208 kHz	149 kHz
	128	261 kHz	174 kHz	104 kHz	75 kHz
	256	130 kHz	87.7 kHz	52.1 kHz	37.2 kHz
	512	65.1 kHz	43.4 kHz	26.1 kHz	18.6 kHz
	1024	32.5 kHz	21.7 kHz	13 kHz	9.33 kHz
	2048	16.3 kHz	10.8 kHz	6.51 kHz	4.65 kHz
	4096	8.13 kHz	5.42 kHz	3.25 kHz	2.33 kHz
	8192	4.07 kHz	2.71 kHz	1.63 kHz	1.16 kHz
	16384	2.03 kHz	1.36 kHz	813 Hz	581 Hz
	32768	1.02 kHz	680 Hz	407 Hz	291 Hz

23.5.3 Delay Settings

Table 23-31 shows the values for the delay after transfer (t_{DT}) and CS to SCK delay (t_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPIx_CTARs. The values calculated assume a 66 MHz system frequency.

Table 23-31. Delay Values

		Delay Prescaler Values (DSPI_CTAR[PBR])			
		1	3	5	7
Delay Scaler Values (DSPI_CTAR[DT])	2	30.0 ns	90.0 ns	150.0 ns	210.0 ns
	4	60.0 ns	180.0 ns	300.0 ns	420.0 ns
	8	120.0 ns	360.0 ns	600.0 ns	840.0 ns
	16	240.0 ns	720.0 ns	1.2µs	1.65 µs
	32	480.0 ns	1.44 µns	2.4 µs	3.3 µs
	64	960.0 ns	2.9 µs	4.8 µs	6.8 µs
	128	2.0 µs	5.7 µs	9.6 µs	13.5 µs
	256	3.9 µs	11.6 µs	19.2 µs	26.9 µs
	512	7.7 µs	23.1 µs	38.4 µs	53.7 µs
	1024	15.3 µs	46.1 µs	76.8 µs	107.6 µs
	2048	30.8 µs	92.1 µs	153.6 µs	215.1 µs
	4096	61.6 µs	184.4 µs	307.2 µs	430.1 µs
	8192	122.9 µs	368.7 µs	614.4 µs	860.1 µs
	16384	245.7 µs	737.3 µs	1.2 ms	1.7 ms
	32768	491.6 µs	1.5 ms	2.4 ms	3.5 ms
65536	998.1 µs	2.0 ms	3.0 ms	6.9 ms	

23.5.4 Calculation of FIFO Pointer Addresses

The user has complete visibility of the TX and RX FIFO contents through the FIFO registers, and valid entries can be identified through a memory-mapped pointer and a memory-mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the transmit next pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the pop next pointer (POPNXTPTR). [Figure 23-40](#) illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO. See [Section 23.4.3.4, “Transmit First-In First-Out \(TX FIFO\) Buffering Mechanism,”](#) and [Section 23.4.3.5, “Receive First-In First-Out \(RX FIFO\) Buffering Mechanism,”](#) for details on the FIFO operation.

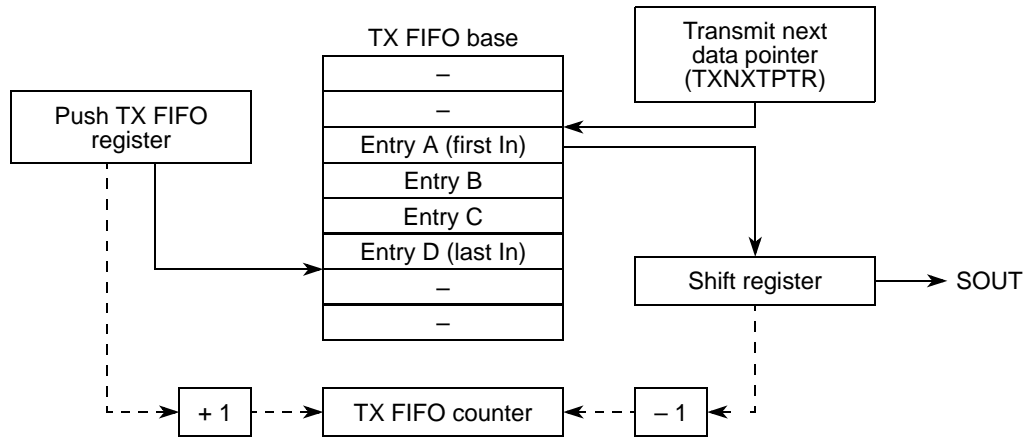


Figure 23-40. TX FIFO Pointers and Counter

23.5.4.1 Address Calculation for the First-in Entry and Last-in Entry in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

$$\text{First-in entry address} = \text{TXFIFO base} + 4 (\text{TXNXPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

$$\text{Last-in entry address} = \text{TX FIFO base} + 4 * [(\text{TXCTR} + \text{TXNXPTR} - 1) \text{ modulo TX FIFO depth}]$$

where:

TX FIFO base: base address of TX FIFO

TXCTR: TX FIFO counter

TXNXPTR: transmit next pointer

TX FIFO depth: transmit FIFO depth

23.5.4.2 Address Calculation for the First-in Entry and Last-in Entry in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

$$\text{First-in entry address} = \text{TX FIFO base} + 4 * (\text{POPNXPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

$$\text{Last-in entry address} = \text{RX FIFO base} + 4 * [(\text{RXCTR} + \text{POPNXPTR} - 1) \text{ modulo RX FIFO depth}]$$

RX FIFO base: base address of RX FIFO

RXCTR: RX FIFO counter

POPNXPTR: pop next pointer

RX FIFO depth: receive FIFO depth

Chapter 24

Enhanced Serial Communication Interface (eSCI)

24.1 Introduction

The eSCI allows asynchronous serial communications with peripheral devices and other CPUs. The eSCI has special features that allow the eSCI to operate as a LIN bus master, complying with the LIN 2.0 specification.

24.1.1 Block Diagram

A simplified block diagram of the eSCI illustrates the functionality and interdependence of major blocks (see Figure 24-1).

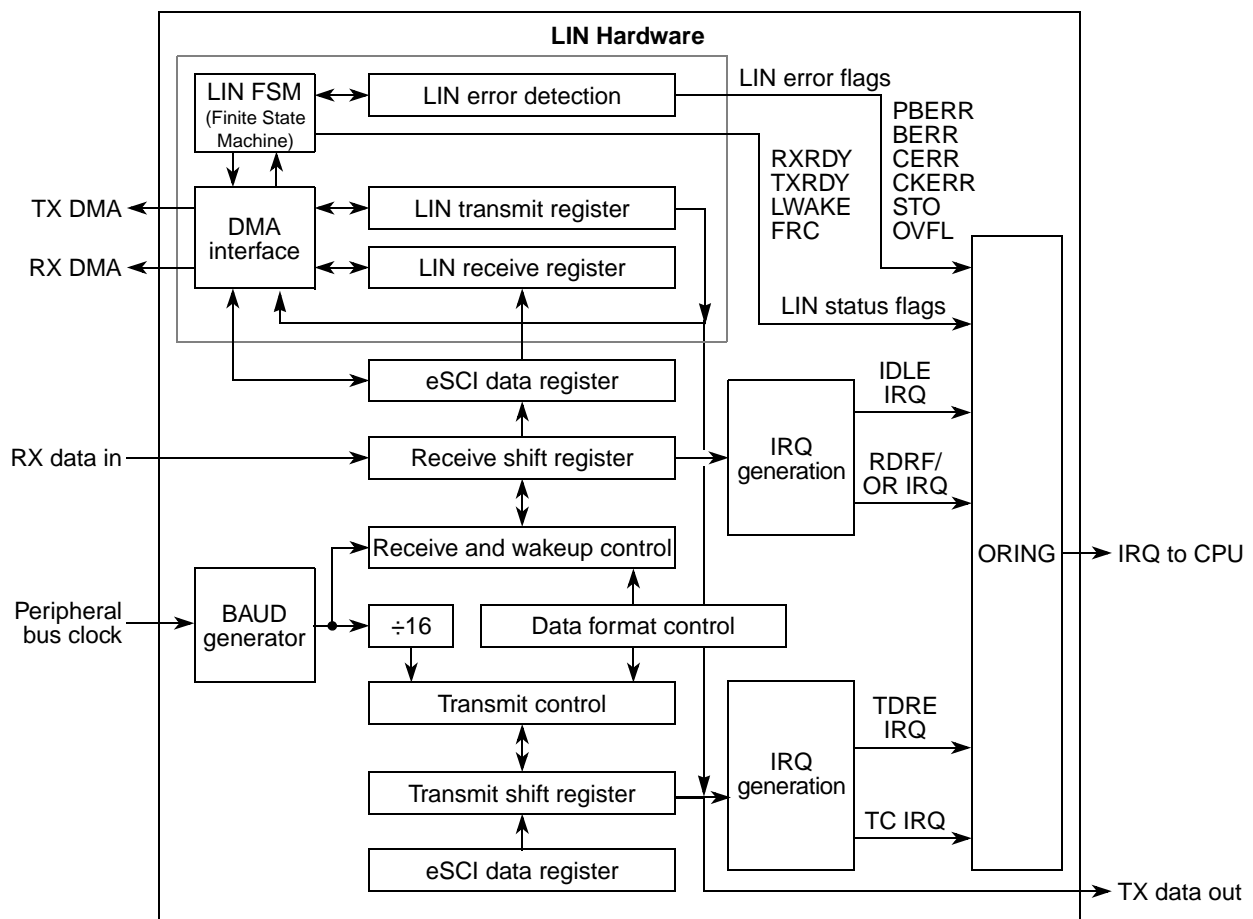


Figure 24-1. eSCI Block Diagram

24.1.2 Features

The eSCI has these major features:

- Full-duplex operation
- Standard mark/space non-return-to-zero (NRZ) format
- 13-bit baud rate selection
- Programmable 8-bit or 9-bit data format
- LIN master node support
- Configurable CRC detection for LIN
- Separately enabled transmitter and receiver
- Two receiver wakeup methods:
 - Idle line wakeup
 - Address mark wakeup
- Interrupt-driven operation with flags
- Receiver framing error detection
- Hardware parity checking
- 1/16 bit-time noise detection
- Two-channel DMA interface

24.1.3 Modes of Operation

There are two operating modes of the eSCI module: run mode and stop mode. In run mode, $eSCI_x = 0$ in the SIU_HLT register and all functional parts of the $eSCI_x$ module are running. In stop mode, $eSCI_x = 1$ in the SIU_HLT register and all clocks to the $eSCI_x$ module are disabled.

The eSCI delays the system going into stop mode, until it has completely transmitted the current TX byte, or completely received the current RX byte. In LIN mode it will complete any frames that do not require further processor intervention (e.g. transmission of a checksum byte).

24.2 External Signal Description

Each $eSCI_x$ module has two external signals: TXD_x (transmit data output of $eSCI_x$) and RXD_x (receive data input of $eSCI_x$). Refer to [Table 2-1](#) and [Section 2.7, “Detailed External Signal Descriptions,”](#) for detailed signal descriptions.

24.3 Memory Map and Registers

This section provides a detailed description of all eSCI registers.

24.3.1 Module Memory Map

The eSCI memory map is shown in [Table 24-1](#). The address of each register is given as an offset to the eSCI base address. Registers are listed in address order, identified by complete name and mnemonic, and include the type of accesses allowed.

Table 24-1. eSCI Memory Map

Offset from eSCI_BASE (eSCI_A = 0xFFFFA_0000 eSCI_B = 0xFFFFA_4000 eSCI_C = 0xFFFFA_8000 eSCI_D = 0xFFFFA_C000 eSCI_E = 0xFFFFB_0000 eSCI_F = 0xFFFFB_4000 eSCI_G = 0xFFFFB_8000 eSCI_H = 0xFFFFB_C000)	Register	Access	Reset Value	Section/Page
0x0000	ESCIx_CR1—eSCI control register 1	R/W	0x0004_0000	24.3.2.1/24-3
0x0004	ESCIx_CR2—eSCI control register 2	R/W	0xA000	24.3.2.2/24-6
000x06	ESCIx_DR—eSCI data register	R/W	0x0000	24.3.2.3/24-7
0x0008	ESCIx_SR—eSCI status register	R	0x8000_0000	24.3.2.4/24-8
0x000C	ESCIx_LCR—LIN control register	R/W	0x0000_0000	24.3.2.5/24-10
0x0010	ESCIx_LTR—LIN transmit register	R/W	0x0000_0000	24.3.2.6/24-12
0x0014	ESCIx_LRR—LIN receive register	R/W	0x0000_0000	24.3.2.7/24-15
0x0018	ESCIx_LPR—LIN cyclic redundancy check (CRC) polynomial register	R	0xC599_0000	24.3.2.8/24-15

24.3.2 Register Descriptions

This section lists the eSCI registers in address order and describes the registers and their bit fields.

24.3.2.1 eSCI Control Register 1 (ESCIx_CR1)

Offset: Base + 0x0000

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0		SBR											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LOOPS	0	RSRC	M	WAKE	ILT	PE	PT	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-2. eSCI Control Register 1 (ESCIx_CR1)

Table 24-2. ESCIx_CR1 Field Descriptions

Field	Description												
bits 0–2	Reserved.												
SBR	<p>SCI Baud Rate. Used by the counter to determine the baud rate of the eSCI. The formula for calculating the baud rate is:</p> $\text{SCI baud rate} = \frac{\text{eSCI system clock}}{16 \times \text{SBR}}$ <p>where SBR can contain a value from 1 to 8191. After reset, the baud generator is disabled until the TE bit or the RE bit is set for the first time. The baud rate generator is disabled when SBR = 0x0.</p>												
LOOPS	<p>Loop Select. Enables loop operation. In loop operation, the RXD pin is disconnected from the eSCI and the transmitter output is internally connected to the receiver input. Both the transmitter and the receiver must be enabled to use the loop function.</p> <p>0 Normal operation enabled, loop operation disabled 1 Loop operation enabled</p> <p>Note: The receiver input is determined by the RSRC bit.</p>												
bit 17	<p>Reserved.</p> <p>Note: Reserved bit 17 is writeable, but writing to this bit has no effect other than to update the value of the register.</p>												
RSRC	<p>Receiver Source. When LOOPS = 1, the RSRC bit determines the source for the receiver shift register input.</p> <p>0 Receiver input internally connected to transmitter output 1 Receiver input connected externally to transmitter</p> <p>The table below shows how LOOPS and RSRC determine the loop function of the eSCI.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>LOOPS</th> <th>RSRC</th> <th>Function</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>x</td> <td>Normal operation</td> </tr> <tr> <td>1</td> <td>0</td> <td>Loop mode with RXD input internally connected to TXD output</td> </tr> <tr> <td>1</td> <td>1</td> <td>Single-wire mode with RXD input connected to TXD</td> </tr> </tbody> </table>	LOOPS	RSRC	Function	0	x	Normal operation	1	0	Loop mode with RXD input internally connected to TXD output	1	1	Single-wire mode with RXD input connected to TXD
LOOPS	RSRC	Function											
0	x	Normal operation											
1	0	Loop mode with RXD input internally connected to TXD output											
1	1	Single-wire mode with RXD input connected to TXD											
M	<p>Data Format Mode. Determines whether data characters are 8 or 9 bits long.</p> <p>0 1 start bit, 8 data bits, 1 stop bit 1 1 start bit, 9 data bits, 1 stop bit</p>												
WAKE	<p>Wakeup Condition. Determines which condition wakes up the eSCI: a logic 1 (address mark) in the most significant bit position of a received data character or an idle condition on the RXD.</p> <p>0 Idle line wakeup 1 Address mark wakeup</p> <p>Note: This is not a wakeup out of a power-save mode, it refers solely to the receiver standby mode.</p>												
ILT	<p>Idle Line Type. Determines when the receiver starts counting logic 1s as idle character bits. The counting begins after the start bit or after the stop bit. If the count begins after the start bit, a string of logic 1s preceding the stop bit may cause false recognition of an idle character. Beginning the count after the stop bit avoids false idle character recognition, but requires properly synchronized transmissions.</p> <p>0 Idle character bit count begins after start bit 1 Idle character bit count begins after stop bit</p>												

Table 24-2. ESCIx_CR1 Field Descriptions (continued)

Field	Description
PE	Parity Enable. Enables the parity function. When enabled, the parity function inserts a parity bit in the most significant bit position of the transmitted word. During reception, the received parity bit will be verified in the most significant bit position. The received parity bit will not be masked out. 0 Parity function disabled 1 Parity function enabled
PT	Parity Type. Determines whether the eSCI generates and checks for even parity or odd parity. With even parity, an even number of 1s clears the parity bit and an odd number of 1s sets the parity bit. With odd parity, an odd number of 1s clears the parity bit and an even number of 1s sets the parity bit. 0 Even parity 1 Odd parity
TIE	Transmitter Interrupt Enable. Enables the transmit data register empty flag ESCIx_SR[TDRE] to generate interrupt requests. The interrupt is suppressed in TX DMA mode. 0 TDRE interrupt requests disabled 1 TDRE interrupt requests enabled
TCIE	Transmission Complete Interrupt Enable. Enables the transmission complete flag ESCIx_SR[TC] to generate interrupt requests. The interrupt is suppressed in TX DMA mode. 0 TC interrupt requests disabled 1 TC interrupt requests enabled
RIE	Receiver Full Interrupt Enable. Enables the receive data register full flag ESCIx_SR[RDRF] and the overrun flag ESCIx_SR[OR] to generate interrupt requests. The interrupt is suppressed in RX DMA mode. 0 RDRF and OR interrupt requests disabled 1 RDRF and OR interrupt requests enabled
ILIE	Idle Line Interrupt Enable. Enables the idle line flag ESCIx_SR[IDLE] to generate interrupt requests. 0 IDLE interrupt requests disabled 1 IDLE interrupt requests enabled
TE	Transmitter Enable. Enables the eSCI transmitter and configures the TXD pin as being controlled by the eSCI. The TE bit can be used to queue an idle preamble. 0 Transmitter disabled 1 Transmitter enabled
RE	Receiver Enable. Enables the eSCI receiver. 0 Receiver disabled 1 Receiver enabled
RWU	Receiver wakeup. Standby state. 0 Normal operation 1 RWU enables the wakeup function and inhibits further receiver interrupt requests. Normally, hardware wakes the receiver by automatically clearing RWU.
SBK	Send Break. Toggling SBK sends one break character (see the description of ESCIx_CR2[BRK13] for break character length). Toggling implies clearing the SBK bit before the break character has finished transmitting. As long as SBK is set, the transmitter continues to send complete break characters. 0 No break characters 1 Transmit break characters

24.3.2.2 eSCI Control Register 2 (ESCIx_CR2)

Offset: Base + 0x0004

Access: Read/Write

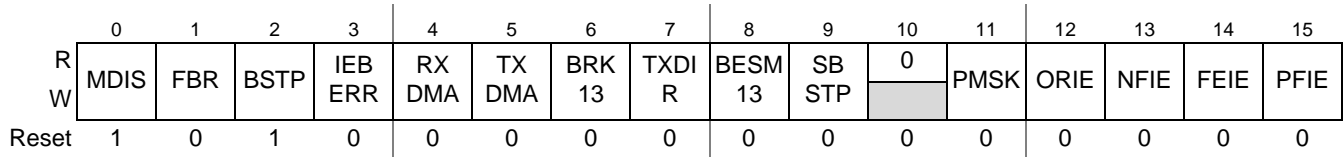


Figure 24-3. eSCI Control Register 2 (ESCIx_CR2)

Table 24-3. ESCIx_CR2 Field Description

Field	Description															
MDIS	Module Disable. The MDIS bit enables or disables the module. DMA requests are negated if the device is in module disable mode. 0 Module enabled 1 Module disabled															
FBR	Fast Bit Error Detection. Manages bit error detection on a per-bit basis. If this is not enabled, bit errors will be detected on a per-byte basis.															
BSTP	Bit Error/Physical Bus Error Stop. Causes DMA TX requests to be suppressed, as long as the bit error and physical bus error flags are not cleared. This stops further DMA writes, which would otherwise cause data bytes to be interpreted as LIN header information.															
IEBERR	Enable Bit Error Interrupt. Generates an interrupt, when a LIN bit error is detected.															
RXDMA	Activate RX DMA Channel. If this bit is enabled and the eSCI has received data, it will raise a DMA RX request.															
TXDMA	Activate TX DMA Channel. When the eSCI is able to transmit data, it will raise a DMA TX request.															
BRK13	Break Transmit Character Length. Determines whether the transmit break character is 10/11 or 13/14 bits long. The detection of a framing error is not affected by this bit. <p style="text-align: center;">Break Length:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td> <td></td> <td colspan="2" style="text-align: center;">ESCIx_CR1[M]</td> </tr> <tr> <td></td> <td></td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> </tr> <tr> <td rowspan="2" style="text-align: center;">BRK13</td> <td style="text-align: center;">0</td> <td style="text-align: center;">10</td> <td style="text-align: center;">11</td> </tr> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">13</td> <td style="text-align: center;">14</td> </tr> </table> 0 Break Character is 10 or 11 bits long 1 Break character is 13 or 14 bits long Note: LIN 2.0 now requires that a break character is always 13 bits long, so this bit must always be set to 1. The eSCI will work with BRK13=0, but it will violate LIN 2.0.			ESCIx_CR1[M]				0	1	BRK13	0	10	11	1	13	14
		ESCIx_CR1[M]														
		0	1													
BRK13	0	10	11													
	1	13	14													
TXDIR	Transmitter Pin Data Direction in Single-Wire Mode. The TXDIR bit determines whether the TXD pin is going to be used as an input or output in the Single-Wire mode of operation. 0 TXD pin to be used as an input in Single-Wire mode 1 TXD pin to be used as an output in Single-Wire mode															

Table 24-3. ESCIx_CR2 Field Description (continued)

Field	Description
BESM13	Bit Error Sample Mode, Bit 13. Determines when to sample the incoming bit to detect a bit error. (This is only relevant when FBR is set.) 0 Sample at RT clock 9 1 Sample at RT clock 13 (see Section 24.4.4.3, "Data Sampling")
SBSTP	SCI Bit Error Stop. Stops the SCI when a bit error is asserted. This allows to stop driving the LIN bus quickly after a bit error has been detected. 0 Byte is completely transmitted 1 Byte is partially transmitted
bit 10	Reserved.
PMSK	Parity Mask. The PMSK bit forces bit 7 in the Data Register to 0 on reads. This can be used to mask the parity bit in applications which use 7 data bits and 1 parity bit.
ORIE	Overrun Error Interrupt Enable. Generates an interrupt, when a frame error is detected.
NFIE	Noise Flag Interrupt Enable. Generates an interrupt, when noise flag is set.
FEIE	Frame Error Interrupt Enable. Generates an interrupt when a frame error is detected.
PFIE	Parity Flag Interrupt Enable. Generates an interrupt when parity flag is set.

24.3.2.3 eSCI Data Register (ESCIx_DR)

Offset: Base + 0x0006

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R8	T8	0	0	0	0	0	0	R7	R6	R5	R4	R3	R2	R1	R0
W									T7	T6	T5	T4	T3	T2	T1	T0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-4. eSCI Data Register (ESCIx_DR)

Table 24-4. ESCIx_DR Field Description

Field	Description
R8	Received Bit 8. R8 is the ninth data bit received when the eSCI is configured for 9-bit data format (M = 1).
T8	Transmit Bit 8. T8 is the ninth data bit transmitted when the eSCI is configured for 9-bit data format (M = 1). Note: If the value of T8 is the same as in the previous transmission, T8 does not have to be rewritten. The same value is transmitted until T8 is rewritten.
bits 2–7	Reserved.
R7–R0 T7–T0	Received Bits/Transmit Bits 7–0 for 9-bit or 8-bit Formats. Bits 7–0 from SCI communication may be read from ESCIx_DR[8–15] (provided that SCI communication was successful). Writing to ESCIx_DR [8–15] provides bits 7–0 for SCI transmission.

NOTES

ESCIx_DR should not be used in LIN mode, writes to this register are blocked in LIN mode (ESCIx_LCR[LIN] = 1).

Even if parity generation/checking is enabled via ESCIx_CR[PE], the parity bit will not be masked out.

24.3.2.4 eSCI Status Register (ESCIx_SR)

The ESCIx_SR indicates the current status. The status flags can be polled, and some can also be used to generate interrupts. All bits in ESCIx_SR except for RAF are cleared by writing 1 to them.

Offset: Base + 0x0008

Access: Read/Write to clear

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	0	0	0	BERR	0	0	0	RAF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c				w1c				
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RX RDY	TX RDY	LWAKE	STO	PB ERR	CERR	CK ERR	FRC	0	0	0	0	0	0	URE Q	OVFL
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c							w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-5. eSCI Status Register (ESCIx_SR)

Table 24-5. ESCIx_SR Field Descriptions

Field	Description
TDRE	Transmit Data Register Empty Flag. TDRE is set when the transmit shift register receives a byte from the eSCI data register. When TDRE is 1, the data register (ESCIx_DR) is empty and can receive a new value to transmit. Clear TDRE by writing 1 to it. 0 No byte transferred to transmit shift register 1 Byte transferred to transmit shift register; transmit data register empty
TC	Transmit Complete Flag. TC is set low when there is a transmission in progress or when a preamble or break character is loaded. TC is set high when the TDRE flag is set and no data, preamble, or break character is being transmitted. When TC is set, the TXD out signal becomes idle (logic 1). After the device is switched on (by clearing the MDIS bit, see Section 24.3.2.2, “eSCI Control Register 2 (ESCIx_CR2)”), a preamble is transmitted; if no byte is written to the SCI data register then the completion of the preamble can be monitored using the TC flag. Clear TC by writing 1 to it. 0 Transmission in progress 1 No transmission in progress; indicates that TXD out is idle
RDRF	Receive Data Register Full Flag. RDRF is set when the data in the receive shift register transfers to the eSCI data register. Clear RDRF by writing 1 to it. 0 Data not available in eSCI data register 1 Received data available in eSCI data register
IDLE	Idle Line Flag. IDLE is set when 10 consecutive logic 1s (if M = 0) or 11 consecutive logic 1s (if M = 1) appear on the receiver input. After the IDLE flag is cleared, a valid frame must again set the RDRF flag before an idle condition can set the IDLE flag. Clear IDLE by writing 1 to it. 0 Receiver input is either active now or has never become active because the IDLE flag was last cleared 1 Receiver input has become idle Note: When the receiver wakeup bit (RWU) is set, an idle line condition does not set the IDLE flag.

Table 24-5. ESCI_x_SR Field Descriptions (continued)

Field	Description
OR	<p>Overrun Flag. OR is set when software fails to read the eSCI data register before the receive shift register receives the next frame. The OR bit is set immediately after the stop bit has been completely received for the second frame. The data in the shift register is lost, but the data already in the eSCI data registers is not affected. Clear OR by writing 1 to it.</p> <p>0 No overrun 1 Overrun</p>
NF	<p>Noise Flag. NF is set when the eSCI detects noise on the receiver input. NF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear NF by writing 1 to it.</p> <p>0 No noise 1 Noise</p>
FE	<p>Framing Error Flag. FE is set when a logic 0 is accepted as the stop bit. FE bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear FE by writing 1 to it.</p> <p>0 No framing error 1 Framing error</p>
PF	<p>Parity error flag. PF is set when the parity enable bit, PE, is set and the parity of the received data does not match its parity bit. Clear PE by writing 1 to it.</p> <p>0 No parity error 1 Parity error</p>
bits 8–10	Reserved.
BERR	<p>Bit Error. Indicates a bit on the bus did not match the transmitted bit. If FBR = 0, checking happens after a complete byte has been transmitted and received again. If FBR = 1, checking happens bit by bit. This bit is used for LIN mode only.</p> <p>BERR is also set if an unrequested byte is received (i.e. a byte that is not part of an RX frame) that is not recognized as a wakeup flag. (Because the data on the RX line does not match the idle state that was assigned to the TX line.)</p> <p>Clear BERR by writing 1 to it.</p> <p>A bit error causes the LIN finite state machine (FSM) to reset unless ESCI_x_LCR[LDBG] is set.</p> <p>0 No bit error 1 Bit error</p>
bits 12–14	Reserved.
RAF	<p>Receiver Active Flag. RAF is set when the receiver detects a logic 0 during the RT1 time period of the start bit search. RAF is cleared when the receiver detects an idle character.</p> <p>0 No reception in progress 1 Reception in progress</p>
RXRDY	<p>Receive Data Ready. The eSCI has received LIN data. This bit is set when the ESCI_x_LCR receives a byte. Clear RXRDY by writing it with 1.</p> <p>0 No receive data ready 1 Receive data ready</p>
TXRDY	<p>Transmit Data Ready. The LIN FSM can accept another write to ESCI_x_LTR. This bit is set when the ESCI_x_LTR register becomes free. Clear TXRDY by writing it with 1.</p> <p>0 ESCI_x_LTR register is not free 1 ESCI_x_LTR register is free</p>

Table 24-5. ESCIx_SR Field Descriptions (continued)

Field	Description
LWAKE	Received LIN Wakeup Signal. A LIN slave has sent a wakeup signal on the bus. When this signal is detected, the LIN FSM will reset. If the setup of a frame had already started, it must be repeated. LWAKE will also be set if ESCI receives a LIN 2.0 wakeup signal (in which the baud rate is lower than 32K baud). See the WU bit. 0 LIN2.0 wakeup signal not received 1 LIN2.0 wakeup signal received
STO	Slave Time Out. Represents a NO_RESPONSE_ERROR. This is set if a slave does not complete a frame within the specified maximum frame length. For LIN 1.3 the following formula is used: $TFRAME_MAX = (10 \times NDATA + 44) \times 1.4$ 0 No time out detected 1 A slave did not complete a frame within the specified maximum frame length
PBERR	Physical Bus Error. No valid message can be generated on the bus. This is set if, after the start of a byte transmission, the input remains unchanged for 31 cycles. This will reset the LIN FSM. 0 No error 1 Physical bus error
CERR	CRC Error. The CRC pattern received with an extended frame was not correct. 0 No error 1 CRC error
CKERR	Checksum Error. Checksum error on a received frame. 0 No error 1 Checksum error
FRC	Frame Complete. LIN frame completely transmitted. All LIN data bytes received. 0 Frame not complete 1 Frame complete
bits 24–29	Reserved.
UREQ	Unrequested Data on LIN Bus. The UREQ bit indicates whether unrequested activity has been detected on the LIN bus. Since the eSCI is used as a master node, this is normally an error condition. The UREQ flag is not set if the activity is identified as a wakeup character. In addition, the RXRDY flag will also be set and the ESCIx_LRR register must be read before normal operations can proceed. Set when the condition is detected and cleared by writing 1 to it. 0 No unrequested data detected 1 Unrequested data detected
OVFL	ESCIx_LRR Overflow. The LIN receive register has not been read before a new data byte, CRC, or checksum byte has been received from the LIN bus. Set when the condition is detected and cleared by writing 1 to it. 0 No overflow 1 Overflow detected

24.3.2.5 LIN Control Register (ESCIx_LCR)

ESCIx_LCR can be written when there are no ongoing transmissions only.

Offset: Base + 0x000C

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LRES	0	WUD	WUD	LDBG	DSF	PRTY	LIN	RXIE	TXIE	WUIE	STIE	PBIE	CIE	CKIE	FCIE
W		WU	0	1												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	UQIE	OFIE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-6. LIN Control Register (ESCIx_LCR)

Table 24-6. ESCI_x_LCR Field Descriptions

Field	Description										
LRES	LIN Resynchronize. Causes the LIN protocol engine to return to start state. This happens automatically after bit errors, but software may force a return to start state manually via this bit. The bit first must be set then cleared, so that the protocol engine is operational again.										
WU	LIN Bus Wakeup. Generates a wakeup signal on the LIN bus. This must be set before a transmission if the bus is in sleep mode. This bit will auto-clear, so a read from this bit will always return 0. According to LIN 2.0, generating a valid wakeup character requires programming the SCI baud rate to a range of 32K baud down to 1.6K baud. Refer to ESCI _x _CR1[SBR] field description (Table 24-2).										
WUD	Wakeup Delimiter Time. Determines how long the LIN engine waits after generating a wakeup signal, before starting a new frame. The eSCI will not set ESCI _x _SR[TXRDY] before this time expires. In addition to this delimiter time, the CPU and the eSCI will require some setup time to start a new transmission and typically there is an additional bit time delay. The table below shows how the values for WUD0 and WUD1 affect the delimiter time. <table border="1" data-bbox="745 1142 1031 1392"> <thead> <tr> <th>WUD</th> <th>Bit Times</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>4</td> </tr> <tr> <td>01</td> <td>8</td> </tr> <tr> <td>10</td> <td>32</td> </tr> <tr> <td>11</td> <td>64</td> </tr> </tbody> </table>	WUD	Bit Times	00	4	01	8	10	32	11	64
WUD	Bit Times										
00	4										
01	8										
10	32										
11	64										
LDBG	LIN Debug Mode. Prevents the LIN FSM from automatically resetting, after an exception (bit error, physical bus error, wakeup flag) has been received. This is for debug purposes only.										
DSF	Double Stop Flags. When a bit error has been detected, this will add an additional stop flag to the byte in which the error occurred.										
PRTY	Activating Parity Generation. Generate the two parity bits in the LIN header.										
LIN	LIN Mode. Switch device into LIN mode. 0 LIN disabled 1 LIN enabled										
RXIE	LIN RXREG Ready Interrupt Enable. Generates an interrupt when new data is available in the LIN RXREG.										
TXIE	LIN TXREG Ready Interrupt Enable. Generates an interrupt when new data can be written to the LIN TXREG.										

Table 24-6. ESCIx_LCR Field Descriptions (continued)

Field	Description
WUIE	RX Wakeup Interrupt Enable. Generates an interrupt when a wakeup flag from a LIN slave has been received.
STIE	Slave Timeout Error Interrupt Enable. Generates an interrupt when the slave response is too slow.
PBIE	Physical Bus Error Interrupt Enable. Generates an interrupt when no valid message can be generated on the bus.
CIE	CRC Error Interrupt Enable. Generates an interrupt when a CRC error on a received extended frame is detected.
CKIE	Checksum Error Interrupt Enable. Generates an interrupt on a detected checksum error.
FCIE	Frame Complete Interrupt Enable. Generates an interrupt after complete transmission of a TX frame, or after the last byte of an RX frame is received. (The complete frame includes all header, data, CRC, and checksum bytes as applicable.)
bits 16–21	Reserved.
UQIE	Unrequested Data Interrupt Enable. Generates an interrupt when a data byte in the ESCIx_LRR register has not been read before the next data byte is received.
OFIE	Overflow Interrupt Enable. Generates an interrupt when a data byte in the ESCIx_LRR has not been read before the next data byte is received.
bits 24–31	Reserved.

24.3.2.6 LIN Transmit Register (ESCIx_LTR)

ESCIx_LTR can be written to only when TXRDY is set. The first byte written to the register selects the transmit address, the second byte determines the frame length, the third and fourth byte set various frame options and determine the timeout counter. Header parity will be automatically generated if the ESCIx_LCR[PRTY] bit is set. For TX frames, the fourth byte (bits T7–T0) is skipped, because the timeout function does not apply. All following bytes are data bytes for the frame. CRC and checksum bytes will be automatically appended when the appropriate options are selected.

When a bit error is detected, an interrupt is set and the transmission aborted. The register can be written again only after the interrupt is cleared. Afterward a new frame starts and the first byte needs to contain a header again.

It is also possible to flush the ESCIx_LTR by setting the ESCIx_LCR[LRES] bit.

NOTE

Not all values written to the ESCIx_LTR will generate valid LIN frames. The values are determined according to the LIN specification.

Offset: Base + 0x0010

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R									0	0	0	0	0	0	0	0
W	P1/ L7/ HDCHK/ T7/ D7	P0/ L6/ CSUM/ T6/ D6	ID5/ L5/ CRC/ T5/ D5	ID4/ L4/ TX/ T4/ D4	ID3/ L3/ T11/ T3/ D3	ID2/ L2/ T10/ T2/ D2	ID1/ L1/ T9/ T1/ D1	ID0/ L0/ T8/ T0/ D0								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-7. LIN Transmit Register (ESCIx_LTR)

Offset: eSCI x Base + 0x0010

Access: Write

	0	1	2	3	4	5	6	7
R								
1st Write (Table 24-7) W	P[1:0]		ID[5:0]					
2nd Write (Table 24-8) W	L[7:0]							
3rd Write (Table 24-9) W	HDCHK	CSUM	CRC	TX (RX)	T[11:8]			
4th Write (Table 24-10) W	T[7:0]							
5th Write (Table 24-11) W	D[7:0]							
Reset	0	0	0	0	0	0	0	0

Figure 24-8. LIN Transmit Register (ESCIx_LTR) Alternate Diagram

Table 24-7. ESCI_x_LTR First Byte Field Description

Field	Description															
P_n	Parity Bit n . When parity generation is enabled (ESCI _x _LCR[PRTY] = 1), the parity bits are generated automatically. Otherwise they must be provided in this field.															
ID_n^1	Header Bit n . The LIN address, for LIN 1.x standard frames the length bits must be set appropriately (see the table below), extended frames are recognized by their specific patterns. <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>ID5</th> <th>ID4</th> <th>data bytes</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>2</td> </tr> <tr> <td>0</td> <td>1</td> <td>2</td> </tr> <tr> <td>1</td> <td>0</td> <td>4</td> </tr> <tr> <td>1</td> <td>1</td> <td>8</td> </tr> </tbody> </table>	ID5	ID4	data bytes	0	0	2	0	1	2	1	0	4	1	1	8
ID5	ID4	data bytes														
0	0	2														
0	1	2														
1	0	4														
1	1	8														
bits 8–31	Reserved.															

¹ The values 3C, 3D, 3E, and 3F of the ID-field (ID0-5) indicate command and extended frames. Refer to LIN specification revision 2.0.

Table 24-8. ESCIx_LTR Second Byte Field Descriptions

Field	Description
L_n	Length Bit n . Defines the length of the frame (0 to 255 data bytes). This information is needed by the LIN state machine to insert the checksum or CRC pattern as required. LIN 1.x slaves will accept frames with 2, 4, or 8 data bytes only.
bits 8–31	Reserved.

Table 24-9. ESCIx_LTR Third Byte Field Descriptions

Field	Description
HDCHK	Header Checksum Enable. Include the header fields into the mod 256 checksum of the standard frames.
CSUM	Checksum Enable. Append a checksum byte to the end of a TX frame. Verify the checksum byte of an RX frame.
CRC	CRC Enable. Append two CRC bytes to the end of a TX frame. Verify the two CRC bytes of an RX frame are correct. If both CSUM and CRC bits are set, the LIN FSM will first append the CRC bytes, then the checksum byte, and will expect them in this order, as well. If HDCHK is set, the CRC calculation will include header and data bytes, otherwise the data bytes only. CRC bytes are not part of the LIN standard; they are normal data bytes and belong to a higher-level protocol.
TX	Transmit Direction. Indicates a TX frame; that is, the eSCI will transmit data to a slave. Otherwise, an RX frame is assumed, and the eSCI only transmits the header. The data bytes are received from the slave. 0 RX frame 1 TX frame
T_n	Timeout Bit n . Sets the counter to determine a NO_RESPONSE_ERROR, if the frame is a read access to a LIN slave. Following LIN standard rev. 1.3, the value $(10 \times N_{DATA} + 45) \times 1.4$ is recommended. For transmissions, this counter has to be set to 0. The timeout bits 7–0 will not be written on a TX frame. For TX frames, the fourth byte written to the LIN transmit register (ESCIx_LTR) is the first data byte, for RX frames it contains timeout bits 7–0. The time is specified in multiples of bit times. The timeout period starts with the transmission of the LIN break character.
bits 8–31	Reserved.

Table 24-10. ESCIx_LTR Rx Frame Fourth Byte Field Descriptions

Field	Description
T_n	Timeout Bit n . Sets the counter to determine a NO_RESPONSE_ERROR, if the frame is a read access to a LIN slave. Following LIN standard rev. 1.3, the value $(10 \times N_{DATA} + 45) \times 1.4$ is recommended. For transmissions, this counter has to be set to 0. The timeout bits 7–0 will not be written on a TX frame. For TX frames, the fourth byte written to the LIN transmit register (ESCIx_LTR) is the first data byte. For RX frames, it contains timeout bits 7–0. The time is specified in multiples of bit times. The timeout period starts with the transmission of the LIN break character.
bits 8–31	Reserved.

**Table 24-11. ESCIx_LTR Tx Frame Fourth+ Byte/
Rx Frame Fifth+ Byte Field Description**

Field	Description
D_n	Data bits for transmission.
bits 8–31	Reserved.

24.3.2.7 LIN Receive Register (ESCIx_LRR)

ESCIx_LRR can be read only when ESCIx_SR[RXRDY] is set.

NOTE

Application software must ensure that ESCIx_LRR be read before new data or checksum bytes or CRCs are received from the LIN bus.

Offset: Base + 0x0014

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-9. LIN Receive Register (ESCIx_LRR)

Table 24-12. ESCIx_LRR Field Descriptions

Field	Description
D_n	<p>Data Bit n. Provides received data bytes from RX frames. Data is valid only when the ESCIx_SR[RXRDY] flag is set. CRC and checksum information will not be available in the ESCIx_LRR unless they are treated as data. It is possible to treat CRC and checksum bytes as data by deactivating the CSUM respectively CRC control bits in the ESCIx_LTR; however, then CRC and CSUM checking has to be performed by software.</p> <p>Data bytes must be read from the ESCIx_LRR (by CPU or DMA) before any new bytes (including CRC or checksum) are received from the LIN bus otherwise the data byte is lost and OVFL is set.</p> <p>Note: The data must be collected and the LIN frame finished (including CRC and checksum if applicable) before a wakeup character can be sent.</p>
bits 8–31	Reserved.

24.3.2.8 LIN CRC Polynomial Register (ESCIx_LPR)

ESCIx_LPR n can be written when there are no ongoing transmissions.

Offset: Base + 0x0018

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0
W																
Reset	1	1	0	0	0	1	0	1	1	0	0	1	1	0	0	1
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-10. LIN CRC Polynomial Register (ESCIx_LPR)

Table 24-13. ESCIx_LPR Field Description

Field	Description
Pn	Polynomial Bit x^n . Bits P15–P0 are used to define the LIN polynomial—standard is $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$ (the polynomial used for the CAN protocol).
bits 16–31	Reserved.

24.4 Functional Description

This section provides a complete functional description of the eSCI module, detailing the operation of the design from the end-user perspective in a number of subsections.

Figure 24-11 shows the structure of the eSCI module. The eSCI allows full duplex, asynchronous, NRZ serial communication between the CPU and remote devices, including other CPUs. The eSCI transmitter and receiver operate independently, although they use the same baud rate generator. The CPU monitors the status of the eSCI, writes the data to be transmitted, and processes received data.

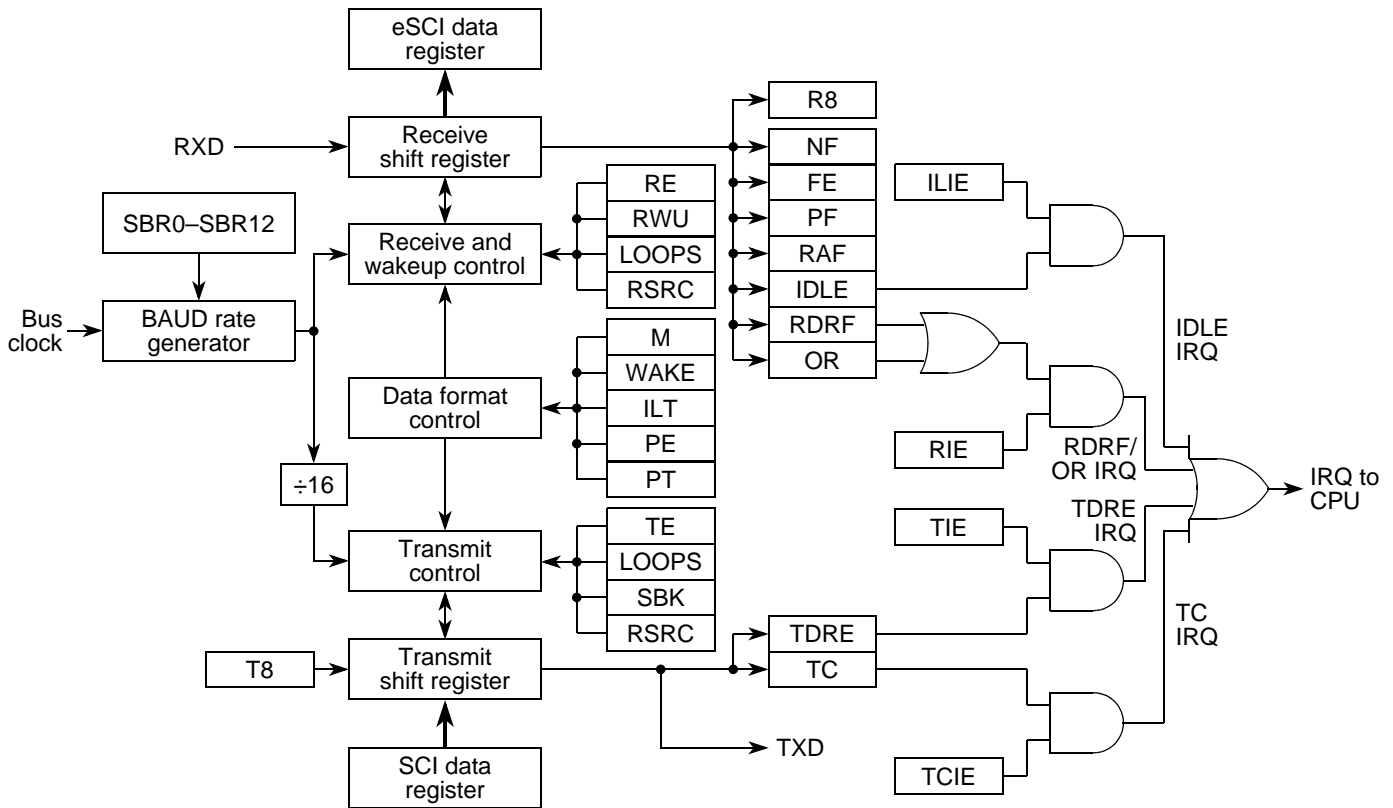


Figure 24-11. eSCI Operation Block Diagram

24.4.1 Data Format

The eSCI uses the standard NRZ mark/space data format. Each data character is contained in a frame that includes a start bit, eight or nine data bits, and a stop bit. Clearing the M bit in eSCI control register 1 configures the eSCI for 8-bit data characters. A frame with eight data bits has a total of 10 bits. Setting the M bit configures the eSCI for 9-bit data characters. A frame with nine data bits has a total of 11 bits.

When the eSCI is configured for 9-bit data characters, the ninth data bit is the T8 bit in the eSCI data register (ESCI_x_DR). It remains unchanged after transmission and can be used repeatedly without rewriting it. A frame with nine data bits has a total of 11 bits.

The two different data formats are illustrated in Figure 24-12. Table 24-14 and Table 24-15 show the number of each type of bit in 8-bit data format and 9-bit data format, respectively.

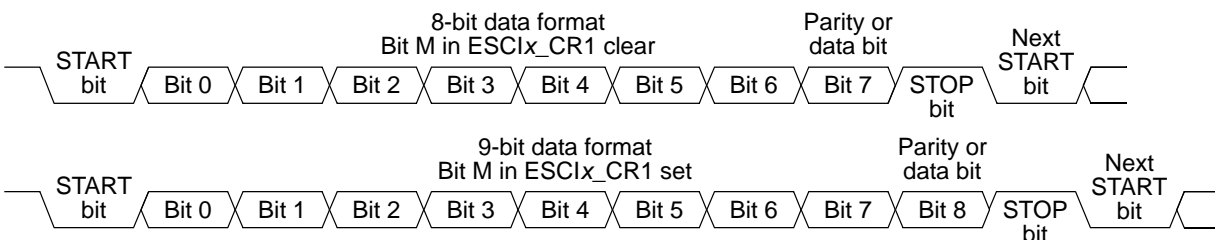


Figure 24-12. eSCI Data Formats

Table 24-14. Example of 8-bit Data Formats

Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
1	8	0	0	1
1	7	0	1	1
1	7	1 ¹	0	1

¹The address bit identifies the frame as an address character. See [Section 24.4.4.6, "Receiver Wakeup."](#)

Table 24-15. Example of 9-Bit Data Formats

Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
1	9	0	0	1
1	8	0	1	1
1	8	1 ¹	0	1

¹The address bit identifies the frame as an address character. See [Section 24.4.4.6, "Receiver Wakeup."](#)

24.4.2 Baud Rate Generation

A 13-bit modulus counter in the baud rate generator derives the baud rate for both the receiver and the transmitter. The value, 1 to 8191, written to the SBR bits determines the system clock divider. The SBR bits are in the eSCI control register 1 (ESCIx_CR1). The baud rate clock is synchronized with the system clock and drives the receiver. The baud rate clock divided by 16 drives the transmitter. The receiver has an acquisition rate of 16 samples per bit time.

Baud rate generation is subject to one source of error:

- Integer division of the system clock may not give the exact target frequency.

[Table 24-16](#) lists some examples of achieving target baud rates with a system clock frequency of 66 MHz.

$$\text{SCI baud rate} = \frac{\text{System clock}}{16 \times \text{ESCIx_CR1[SBR]}}$$

Table 24-16. Baud Rates (Example: System Clock = 66 MHz)

Value in SBR	Receiver clock (Hz)	Transmitter clock (Hz)	Target baud rate	Error (%)
0x0012	3,666,667	229,167	230,400	-0.54
0x0024	1,833,333	114,583	115,200	-0.54
0x0048	916,667	57,292	57,600	-0.54
0x006B	616,822	38,551	38,400	+0.39
0x00D7	306,977	19,186	19,200	-0.07

Table 24-16. Baud Rates (Example: System Clock = 66 MHz)

Value in SBR	Receiver clock (Hz)	Transmitter clock (Hz)	Target baud rate	Error (%)
0x011E	230,769	14,423	14,400	+0.16
0x01AE	153,488	9,593	9,600	-0.07
0x035B	76,834	4,802	4,800	+0.04
0x06B7	38,394	2399.7	2,400	-0.01
0x0D6E	19,197	1,199.8	1,200	-0.01

24.4.3 Transmitter

Figure 24-13 illustrates the features of the eSCI transmitter.

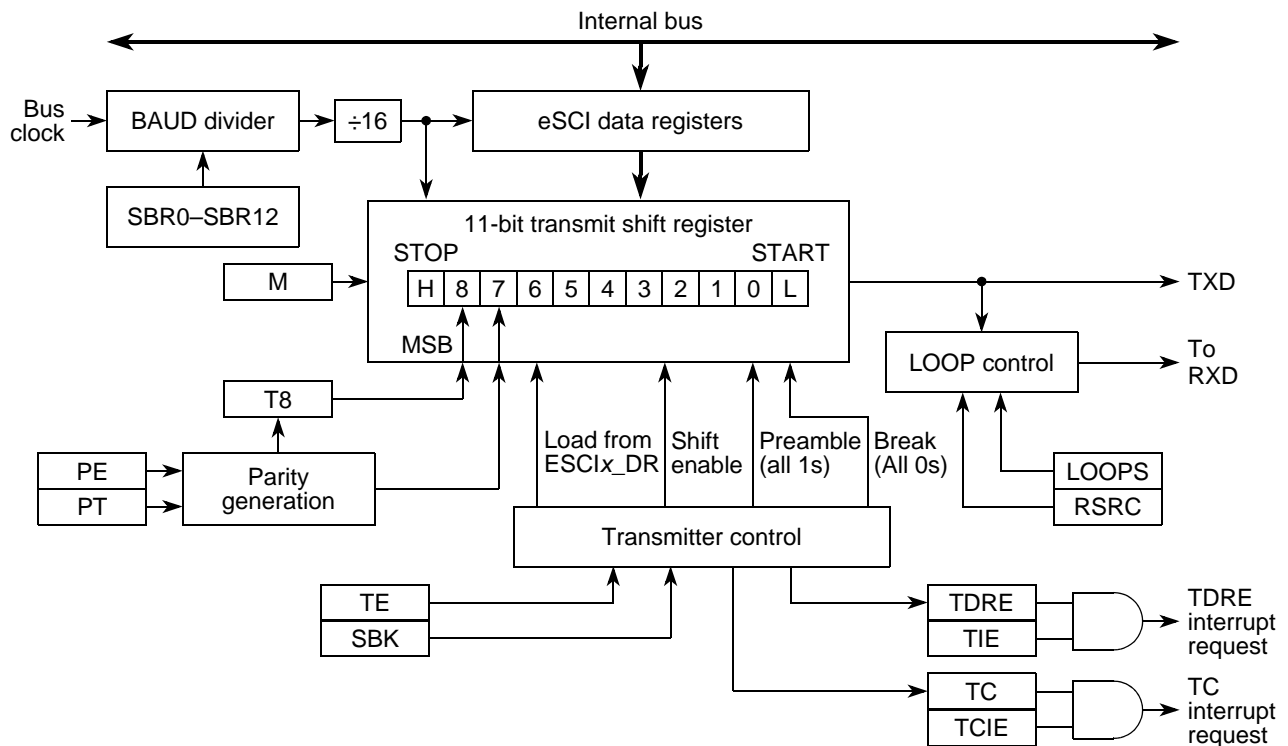


Figure 24-13. eSCI Transmitter Block Diagram

24.4.3.1 Transmitter Character Length

The eSCI transmitter can accommodate either 8-bit or 9-bit data characters. The state of the M bit in eSCI control register 1 (ESCIx_CR1) determines the length of data characters. When transmitting 9-bit data, bit T8 in the eSCI data register (ESCIx_DR) is the ninth bit (bit 8).

24.4.3.2 Character Transmission

To transmit data, the MCU writes the data bits to the eSCI data register (ESCIx_DR), which in turn are transferred to the transmit shift register. The transmit shift register then shifts a frame out through the TXD signal, after it has prefaced them with a start bit and appended them with a stop bit. The eSCI data register (ESCIx_DR) is the buffer (write-only during transmit) between the internal data bus and the transmit shift register.

The eSCI also sets a flag, the transmit data register empty flag (TDRE), every time it transfers data from the buffer (ESCIx_DR) to the transmit shift register. The transmit driver routine may respond to this flag by writing another byte to the transmitter buffer (ESCIx_DR), while the shift register remains shifting out the first byte.

To initiate an eSCI transmission:

1. Clear ESCIx_CR2[MDIS] bit, if this bit is set, to enable the eSCI_x module.
2. Configure the eSCI with single 32-bit write to ESCIx_CR1:
 - c) Write to ESCIx_CR1[SBR] to start the baud generator with a target baud rate.
 - d) Write to ESCIx_CR1 to configure word length, parity, and other configuration bits (LOOPS, RSRC, M, WAKE, ILT, PE, PT).
 - e) Write to ESCIx_CR1 to enable the transmitter, receiver, interrupts, and wakeup as required (TIE, TCIE, RIE, ILIE, TE, RE, RWU, SBK). A preamble or idle character will now be shifted out of the transmitter shift register.
3. Transmit procedure for each byte:
 - a) Poll the TDRE flag by reading the ESCIx_SR or responding to the TDRE interrupt. Remember that the TDRE bit resets to 1.
 - b) If the TDRE flag is set, write the data to be transmitted to ESCIx_DR, where the ninth bit is written to the T8 bit in ESCIx_DR if the eSCI is in 9-bit data format. A new transmission will not result until the TDRE flag has been cleared.
4. Repeat step 3 for each subsequent transmission.

NOTE

The TDRE flag is set when the shift register is loaded with the next data to be transmitted from ESCIx_DR, which occurs approximately halfway through the stop bit of the previous frame. Specifically, this transfer occurs 9/16ths of a bit time AFTER the start of the stop bit of the previous frame.

Toggling the TE bit from 0 to 1 automatically loads the transmit shift register with a preamble of 10 logic 1s (if M = 0) or 11 logic 1s (if M = 1). After the preamble shifts out, control logic transfers the data from the eSCI data register into the transmit shift register. A logic 0 start bit automatically goes into the least significant bit position of the transmit shift register. A logic 1 stop bit goes into the most significant bit position.

The eSCI hardware supports odd or even parity. When parity is enabled, the most significant bit (msb) of the data character is the parity bit.

The transmit data register empty flag, TDRE, in the eSCI status register (ESCIx_SR) becomes set when the eSCI data register transfers a byte to the transmit shift register. The TDRE flag indicates that the eSCI data register can accept new data from the internal data bus. If the transmit interrupt enable bit, TIE, in eSCI control register 1 (ESCIx_CR1) is also set, the TDRE flag generates a transmitter interrupt request.

When the transmit shift register is not transmitting a frame, the TXD output goes to the idle condition, logic 1. If at any time software clears the TE bit in eSCI control register 1 (ESCIx_CR1), the transmitter enable signal goes low and the TXD output goes idle.

If software clears TE while a transmission is in progress (ESCIx_CR1[TC] = 0), the frame in the transmit shift register continues to shift out. To avoid accidentally cutting off the last frame in a message, always wait for TDRE to go high after the last frame before clearing TE.

To separate messages with preambles with minimum idle line time, use the following sequence between messages:

1. Write the last byte of the first message to ESCIx_DR.
2. Wait for the TDRE flag to go high, indicating the transfer of the last frame to the transmit shift register.
3. Queue a preamble by clearing and then setting the TE bit.
4. Write the first byte of the second message to ESCIx_DR.

24.4.3.3 Break Characters

Setting the break bit, SBK, in eSCI control register 1 (ESCIx_CR1) loads the transmit shift register with a break character. A break character contains all logic 0s and has no start, stop, or parity bit. Break character length depends on the M bit in the eSCI control register 1 (ESCIx_CR1) and on the BRK13 bit in the eSCI control register 2 (ESCIx_CR2). As long as SBK is set, the transmitter logic continuously loads break characters into the transmit shift register. After software clears the SBK bit, the shift register finishes transmitting the last break character and then transmits at least one logic 1. The automatic logic 1 at the end of a break character guarantees the recognition of the start bit of the next frame.

NOTE

LIN 2.0 now requires that a break character is always 13 bits long, so the BRK13 bit should always be set to 1. The eSCI will work with BRK13=0, but it will violate LIN 2.0.

The eSCI recognizes a break character when a start bit is followed by eight or nine logic 0 data bits and a logic 0 where the stop bit should be. Receiving a break character has the following effects on eSCI registers:

- Sets the framing error flag, FE.
- Sets the receive data register full flag, RDRF.
- Clears the eSCI data register (ESCIx_DR).
- May set the overrun flag, OR, noise flag, NF, parity error flag, PF, or the receiver active flag, RAF. For more detail, see [Section 24.3.2.4, “eSCI Status Register \(ESCIx_SR\).”](#)

24.4.3.4 Idle Characters

An idle character contains all logic 1s and has no start, stop, or parity bit. Idle character length depends on the M bit in eSCI control register 1 (ESCIx_CR1). The preamble is a synchronizing idle character that begins the first transmission initiated after toggling the TE bit from 0 to 1.

If the TE bit is cleared during a transmission, the TXD output becomes idle after completion of the transmission in progress. Clearing and then setting the TE bit during a transmission queues an idle character to be sent after the frame currently being transmitted.

NOTE

When queuing an idle character, return the TE bit to logic 1 before the stop bit of the current frame shifts out through the TXD output. Setting the TE bit after the stop bit shifts out through the TXD output causes data previously written to the eSCI data register to be lost. Toggle the TE bit for a queued idle character while the TDRE flag is set and immediately before writing the next byte to the eSCI data register.

24.4.3.5 Fast Bit Error Detection in LIN Mode

Fast bit error detection has been designed to allow flagging of LIN bit errors while they occur, rather than flagging them after a byte transmission has completed. To use this feature, it is assumed a physical interface connects to the LIN bus as shown in [Figure 24-14](#).

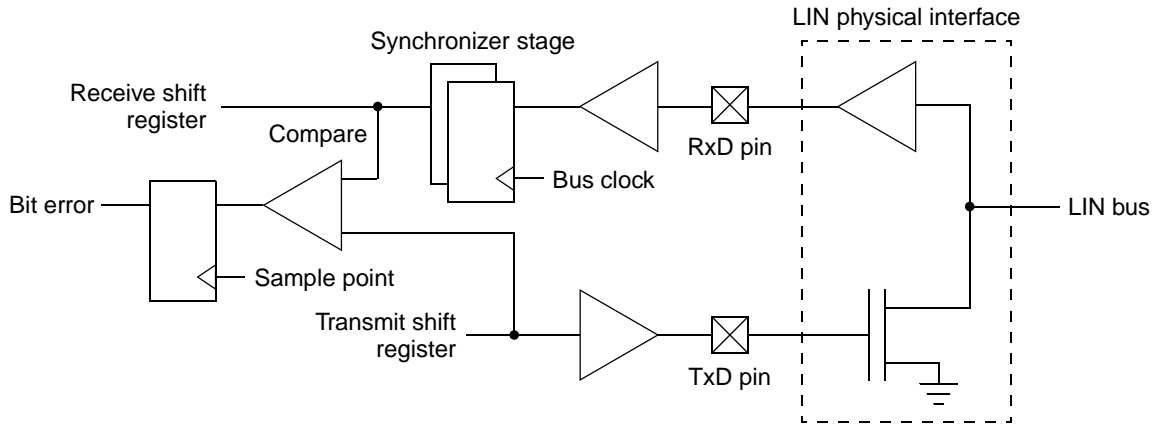


Figure 24-14. Fast Bit Error Detection on a LIN Bus

If fast bit error detection is enabled (FBR = 1), the eSCI will compare the transmitted and the received data stream when the transmitter is active (not idle). After a mismatch between the transmitted data and the received data is detected the following actions are performed:

- The LIN frame is aborted (provided LDBG=0).
- The bit error flag BERR will be set.
- If SBSTP is 0, the remainder of the byte will be transmitted normally.
- If SBSTP is 1, the remaining bits in the byte after the error bit are transmitted as 1s (idle).

To adjust to different bus loads, the sample point at which the incoming bit is compared to the one which was transmitted can be selected with the BESM13 bit (see Figure 24-15). If set, the comparison will be performed at RT clock 13, otherwise at RT clock 9 (also see Section 24.4.4.3, “Data Sampling”).

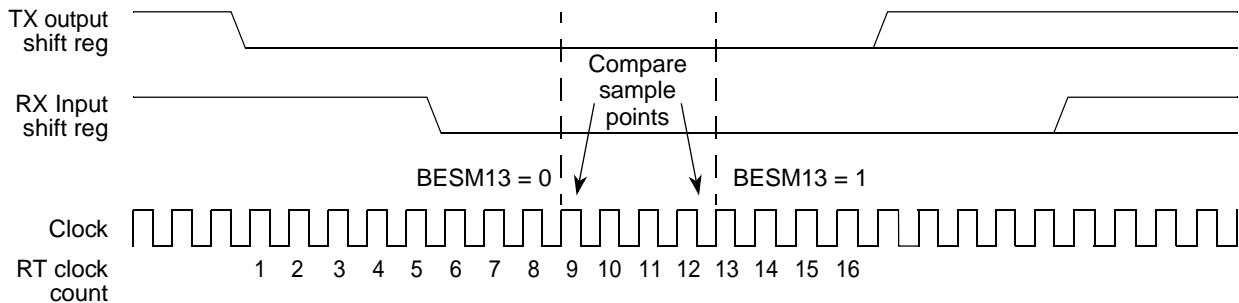


Figure 24-15. Fast Bit Error Detection Timing Diagram

24.4.4 Receiver

Figure 24-16 illustrates the eSCI receiver.

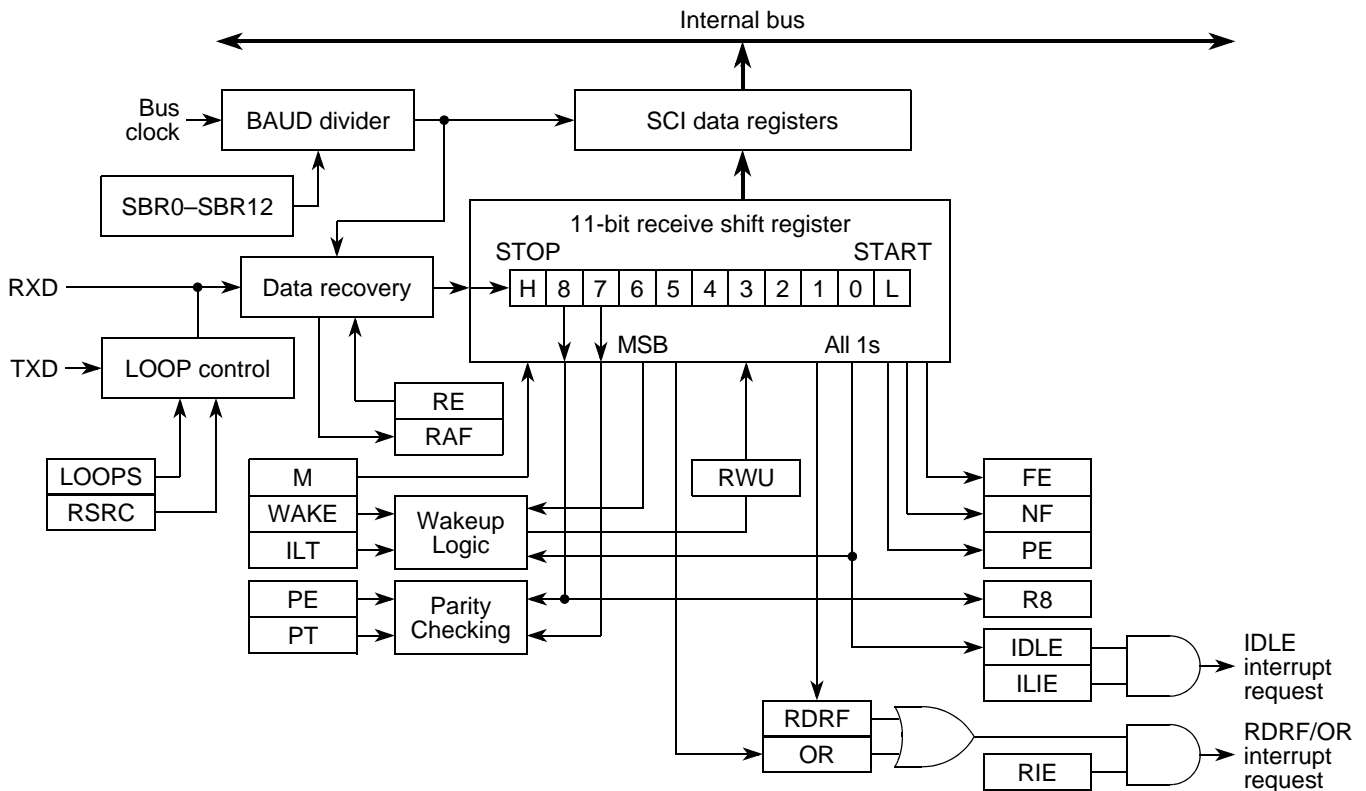


Figure 24-16. eSCI Receiver Block Diagram

24.4.4.1 Receiver Character Length

The eSCI receiver can accommodate 8-bit or 9-bit data characters. The state of the M bit in eSCI control register 1 (ESCIx_CR1) determines the length of data characters. When receiving 9-bit data, bit R8 in the eSCI data register (ESCIx_DR) is the ninth bit (bit 8).

24.4.4.2 Character Reception

During an eSCI reception, the receive shift register shifts a frame in from the RXD input signal. The eSCI data register is the buffer (read-only during receive) between the internal data bus and the receive shift register.

After a complete frame shifts into the receive shift register, the data portion of the frame transfers to the eSCI data register. The receive data register full flag, RDRF, in eSCI status register (ESCIx_SR) is then set, indicating that the received byte can be read. If the receive interrupt enable bit, RIE, in eSCI control register 1 (ESCIx_CR1) is also set, the RDRF flag generates an RDRF interrupt request.

24.4.4.3 Data Sampling

The receiver uses a sampling clock to sample the RXD input signal at the 16 times the baud-rate frequency. This sampling clock is called the RT clock. To adjust for baud rate mismatch, the RT clock (see Figure 24-17) is re-synchronized:

- After every start bit.
- After the receiver detects a data bit change from logic 1 to logic 0. This data bit change is detected when a majority of data samples return a valid logic 1 and a majority of the next data samples return a valid logic 0. Data samples are taken at RT8, RT9, and RT10, as shown in Figure 24-17.

To locate the start bit, eSCI data recovery logic performs an asynchronous search for a logic 0 preceded by three logic 1s. When the falling edge of a possible start bit occurs, the RT clock begins to count to 16.

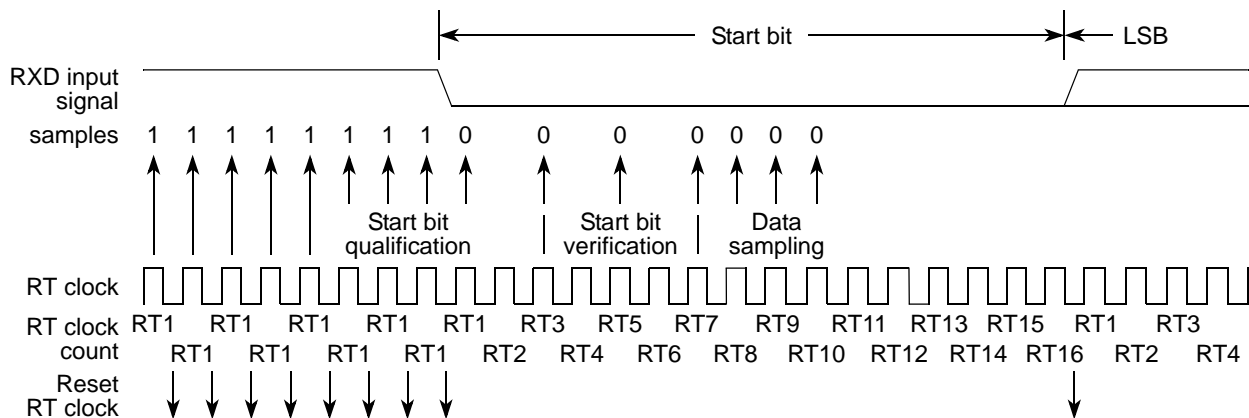


Figure 24-17. Receiver Data Sampling

To verify the start bit and to detect noise, the eSCI data recovery logic takes samples at RT3, RT5, and RT7. Table 24-17 summarizes the results of the start bit verification samples.

Table 24-17. Start Bit Verification

RT3, RT5, and RT7 Samples	Start Bit Verification	Noise Flag
000	Yes	0
001	Yes	1
010	Yes	1
011	No	0
100	Yes	1
101	No	0
110	No	0
111	No	0

If start bit verification is not successful, the RT clock is reset and a new search for a start bit begins.

To determine the value of a data bit and to detect noise, eSCI recovery logic takes samples at RT8, RT9, and RT10. [Table 24-18](#) summarizes the results of the data bit samples.

Table 24-18. Data Bit Recovery

RT8, RT9, and RT10 Samples	Data Bit Determination	Noise Flag
000	0	0
001	0	1
010	0	1
011	1	1
100	0	1
101	1	1
110	1	1
111	1	0

NOTE

The RT8, RT9, and RT10 samples do not affect start bit verification. If any or all of the RT8, RT9, and RT10 start bit samples are logic 1s following a successful start bit verification, the noise flag (NF) is set.

To verify a stop bit and to detect noise, recovery logic takes samples at RT8, RT9, and RT10. [Table 24-19](#) summarizes the results of the stop bit samples.

Table 24-19. Stop Bit Recovery

RT8, RT9, and RT10 samples	Framing error flag	Noise flag
000	1	0
001	1	1
010	1	1

Table 24-19. Stop Bit Recovery (continued)

RT8, RT9, and RT10 samples	Framing error flag	Noise flag
011	0	1
100	1	1
101	0	1
110	0	1
111	0	0

In Figure 24-18 the verification samples RT3 and RT5 determine that the first low detected was noise and not the beginning of a start bit. The RT clock is reset and the start bit search begins again. The noise flag is not set because the noise occurred before the start bit was found.

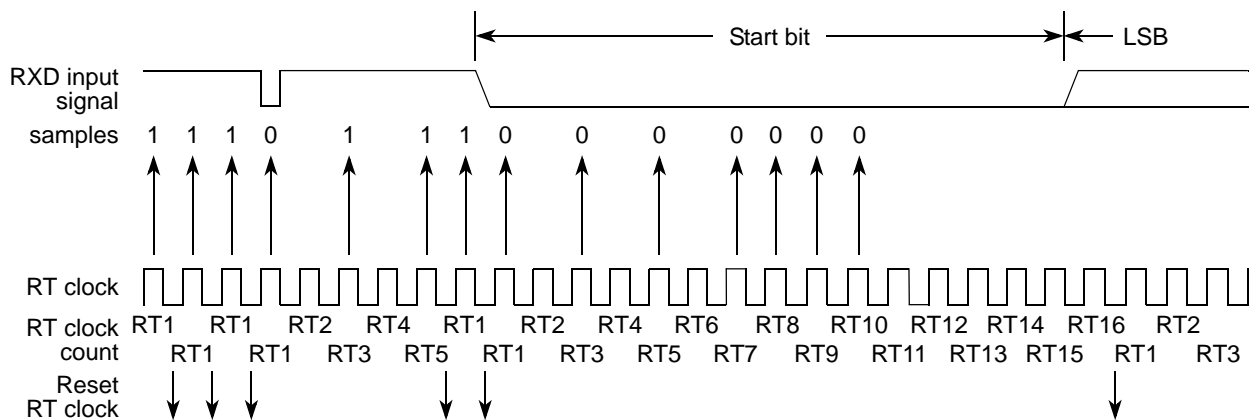


Figure 24-18. Start Bit Search Example

24.4.4.4 Framing Errors

If the data recovery logic sets the framing error flag, `ESCIx_SR[FE]`, it does not detect a logic 1 where the stop bit should be in an incoming frame. A break character also sets the FE flag because a break character has no stop bit. The FE flag is set at the same time that the RDRF flag is set.

24.4.4.5 Baud Rate Tolerance

A transmitting device may be operating at a baud rate below or above the receiver baud rate. Accumulated bit time misalignment can cause one of the three stop bit data samples (RT8, RT9, and RT10) to fall outside the actual stop bit. A noise error occurs if the RT8, RT9, and RT10 samples are not all the same logical values. A framing error occurs if the receiver clock is misaligned in such a way that the majority of the RT8, RT9, and RT10 stop bit samples are a logic zero.

As the receiver samples an incoming frame and re-synchronizes the RT clock on any valid falling edge within the frame. Re-synchronization within frames will correct a misalignment between transmitter bit times and receiver bit times.

24.4.4.5.1 Slow Data Tolerance

Figure 24-19 shows how much a slow received frame can be misaligned without causing a noise error or a framing error. The slow stop bit begins at RT8 instead of RT1 but arrives in time for the stop bit data samples at RT8, RT9, and RT10.

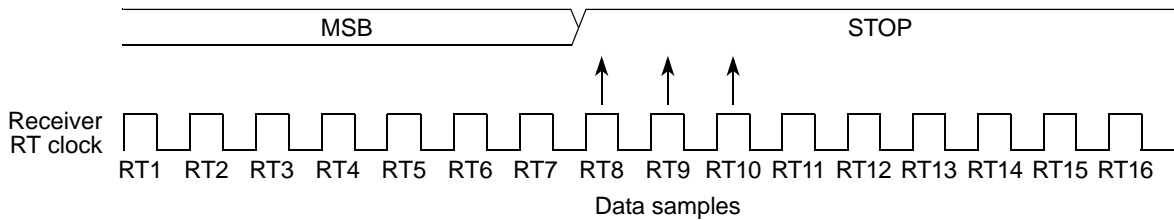


Figure 24-19. Slow Data

For an 8-bit data character, data sampling of the stop bit takes the receiver RT clock 151 clock cycles, as is shown below:

$$9 \text{ bit times} \times 16 \text{ RT cycles} + 7 \text{ RT cycles} = 151 \text{ RT cycles}$$

With the misaligned character shown in Figure 24-19, the receiver counts 151 RT cycles at the point when the count of the transmitting device is 9 bits multiplied by 16 RT cycles = 144 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a slow 8-bit data character with no errors is 4.63%, as is shown below:

$$\frac{151 - 144}{151} \times 100 = 4.63\%$$

For a 9-bit data character, data sampling of the stop bit takes the receiver 167 RT cycles, as is shown below:

$$10 \text{ bit times} \times 16 \text{ RT cycles} + 7 \text{ RT cycles} = 167 \text{ RT cycles}$$

With the misaligned character shown in Figure 24-19, the receiver counts 167 RT cycles at the point when the count of the transmitting device is 10 bit multiplied by 16 RT cycles = 160 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a slow 9-bit character with no errors is 4.19%, as is shown below:

$$\frac{167 - 160}{167} \times 100 = 4.19\%$$

24.4.4.5.2 Fast Data Tolerance

Figure 24-20 shows how much a fast received frame can be misaligned. The fast stop bit ends at RT10 instead of RT16 but remains sampled at RT8, RT9, and RT10.

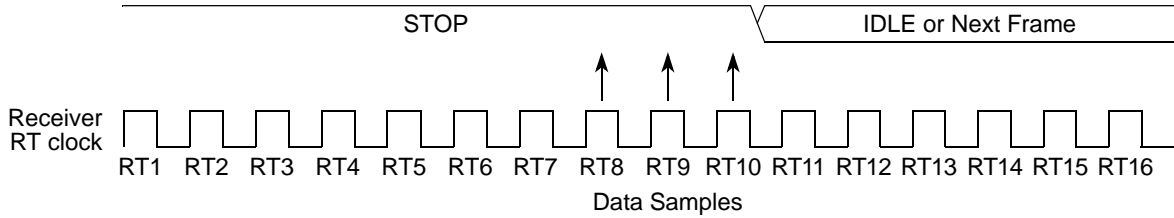


Figure 24-20. Fast Data

For an 8-bit data character, data sampling of the stop bit takes the receiver 154 RT cycles, as is shown below:

$$9 \text{ bit times} \times 16 \text{ RT cycles} + 10 \text{ RT cycles} = 154 \text{ RT cycles}$$

With the misaligned character shown in [Figure 24-20](#), the receiver counts 154 RT cycles at the point when the count of the transmitting device is 10 bit multiplied by 16 RT cycles = 160 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a fast 8-bit character with no errors is 3.40%, as is shown below:

$$\frac{160 - 154}{160} \times 100 = 3.40\%$$

For a 9-bit data character, data sampling of the stop bit takes the receiver 170 RT cycles, as shown below:

$$10 \text{ bit times} \times 16 \text{ RT cycles} + 10 \text{ RT cycles} = 170 \text{ RT cycles}$$

With the misaligned character shown in [Figure 24-20](#), the receiver counts 170 RT cycles at the point when the count of the transmitting device is 11 bit multiplied by 16 RT cycles = 176 RT cycles.

The maximum percent difference between the receiver count and the transmitter count of a fast 9-bit character with no errors is 3.40%, as is shown below:

$$\frac{176 - 170}{176} \times 100 = 3.40\%$$

24.4.4.6 Receiver Wakeup

The receiver can be put into a standby state, which enables the eSCI to ignore transmissions intended only for other receivers in multiple-receiver systems. Setting the receiver wakeup bit, `ESCIx_CR1[RWU]`, in eSCI control register 1 (`ESCIx_CR1`) puts the receiver into standby state during which receiver interrupts are disabled. The eSCI will load the received data into the `ESCIx_DR`, but it will not set the receive data register full (`RDRF`) flag.

The transmitting device can address messages to selected receivers by including addressing information (address bits) in the initial frame or frames of each message. See [Section 24.4.1, “Data Format,”](#) for an example of address bits.

The WAKE bit in eSCI control register 1 (ESCIx_CR1) determines how the eSCI is brought out of the standby state to process an incoming message. The WAKE bit enables either idle line wakeup or address mark wakeup.

24.4.4.6.1 Idle Input Line Wakeup (WAKE = 0)

Using the receiver idle input line wakeup method allows an idle condition on the RXD signal clears the ESCIx_CR1[RWU] bit and wakes up the eSCI. The initial frame or frames of every message contain addressing information. All receivers evaluate the addressing information, and receivers for which the message is addressed process the frames that follow. Any receiver for which a message is not addressed can set its RWU bit and return to the standby state. The RWU bit remains set and the receiver remains on standby until another idle character appears on the RXD signal.

Idle line wakeup requires that messages be separated by at least one idle character and that no message contains idle characters.

The idle character that wakes a receiver does not set the receiver idle bit, ESCIx_SR[IDLE], or the receive data register full flag, RDRF.

The idle line type bit, ESCIx_CR1[ILT], determines whether the receiver begins counting logic 1s as idle character bits after the start bit or after the stop bit.

24.4.4.6.2 Address Mark Wakeup (WAKE = 1)

Using the address mark wakeup method allows a logic 1 in the most significant bit (msb) position of a frame to clear the RWU bit and wakeup the eSCI. The logic 1 in the msb position marks a frame as an address frame that contains addressing information. All receivers evaluate the addressing information, and the receivers for which the message is addressed process the frames that follow. Any receiver for which a message is not addressed can set its RWU bit and return to the standby state. The RWU bit remains set and the receiver remains on standby until another address frame appears on the RXD signal.

The logic 1 msb of an address frame clears the receiver's RWU bit before the stop bit is received and sets the RDRF flag.

Address mark wakeup allows messages to contain idle characters but requires that the msb be reserved for use in address frames.

NOTE

With the WAKE bit clear, setting the RWU bit after the RXD signal has been idle can cause the receiver to wakeup immediately.

24.4.5 Single-Wire Operation

Normally, the eSCI uses two pins for transmitting and receiving. In single-wire operation, the RXD pin is disconnected from the eSCI. The eSCI uses the TXD pin for both receiving and transmitting.

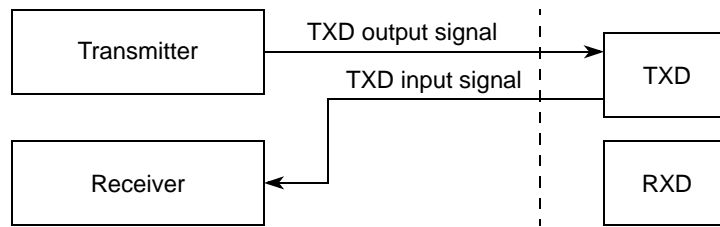


Figure 24-21. Single-Wire Operation (LOOPS = 1, RSRC = 1)

Enable single-wire operation by setting the LOOPS bit and the receiver source bit, RSRC, in eSCI control register 1 (ESCLx_CR1). Setting the LOOPS bit disables the path from the RXD signal to the receiver. Setting the RSRC bit connects the receiver input to the output of the TXD pin driver. The TXDIR bit determines whether the TXD pin is going to be used as an input (TXDIR=0) or an output (TXDIR=1) in the single-wire mode of operation.

During reception, both the transmitter and receiver must be enabled (TE = 1 and RE = 1). The SIU_PCR89[PA] and SIU_PCR91[PA] bits must be set to select the TXD function for the relevant eSCI module, and the TXD pin should be set for open drain operation (SIU_PCR $_{nn}$ [ODE] = 1). Weak pullup may optionally be enabled if the external transmitting device is also open drain. See [Section 6.3.2.13, “Pad Configuration Registers \(SIU_PCR\)”](#).

During transmission, the transmitter must be enabled (TE=1); the receiver may be enabled or disabled. If the receiver is enabled (RE=1), transmissions will be echoed back on the receiver. Set or clear open drain output enable depending on desired operation.

24.4.6 Loop Operation

In loop operation, the transmitter output goes to the receiver input. The RXD signal is disconnected from the eSCI.

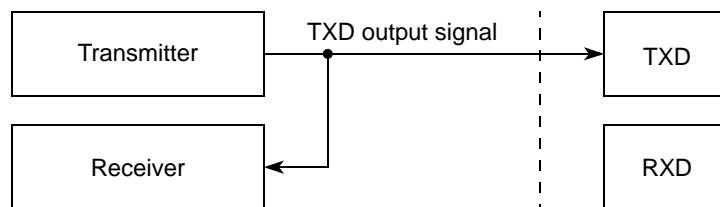


Figure 24-22. Loop Operation (LOOPS = 1, RSRC = 0)

Enable loop operation by setting the LOOPS bit and clearing the RSRC bit in eSCI control register 1 (ESCLx_CR1). Setting the LOOPS bit disables the path from the RXD signal to the receiver. Clearing the RSRC bit connects the transmitter output to the receiver input. Both the transmitter and receiver must be enabled (TE = 1 and RE = 1).

24.4.7 Disabling the eSCI

Each of the eSCI modules can be independently disabled by setting ESCLx_CR2[MDIS] = 1. Disabling the module turns off the clock to the module, although some of the module registers may be accessed by the

core via the slave bus. The MDIS bit is intended to be used when the module is not required in the application.

The module disable bit (ESCIx_CR2[MDIS]) in the eSCI control register 2 can be used to turn off the eSCI. This saves power by stopping the eSCI core from being clocked.

By default the eSCI is disabled (ESCIx_CR2[MDIS]=1).

24.4.7.1 Stop Mode

The eSCI is inactive during stop mode (SIU_HLT[ESCI_x] = 1) for reduced power consumption. To avoid corrupting data, the eSCI will prevent the system from entering Stop mode before the current operation is completed. In SCI mode the eSCI will wait until the current byte has been received or transmitted. It is possible that a received byte will nevertheless be corrupted, so the first byte in the eSCI after waking up from Stop mode could be invalid. To help prevent byte corruption, first enter doze mode, then go into stop mode.

In LIN mode the eSCI will wait at least until the current byte has been received or transmitted before entering stop mode. If the LIN FSM has some more data to receive or transmit which does not require processor access (CRC and checksum bytes or last transmit byte of a frame) the eSCI will delay stop mode until these operations are complete, too.

If a LIN frame was aborted, the DMA controller will be out of sync, and the channel needs to be restarted after leaving stop mode.

24.4.8 Interrupt Operation

24.4.8.1 Interrupt Sources

There are several interrupt sources that can generate an eSCI interrupt to the CPU. They are listed with details and descriptions in [Chapter 8, “Interrupts”](#) (specifically [Table 8-2](#)).

The eSCI originates interrupt requests only. The following sections describe how the eSCI generates a request and how the MCU acknowledges that request. The eSCI has a single interrupt line (eSCI interrupt signal, active high operation) only and all the following interrupts, when generated, are ORed together and issued through that port.

24.4.8.2 Interrupt Flags

24.4.8.2.1 TDRE Description

The transmit data register empty (TDRE) interrupt is set high by the eSCI when the transmit shift register receives data, 8 or 9 bits, from the eSCI data register, ESCIx_DR. A TDRE interrupt indicates that the transmit data register (ESCIx_DR) is empty and that a new data can be written to the ESCIx_DR for transmission. The TDRE bit is cleared by writing a 1 to the TDRE bit location in the ESCIx_SR.

24.4.8.2.2 TC Description

The transmit complete (TC) interrupt is set by the eSCI when a transmission has completed. A TC interrupt indicates that there is no transmission in progress. TC is set high when the TDRE flag is set and no data, preamble, or break character is transmitted. When TC is set, the TXD pin becomes idle (logic 1). The TC bit is cleared by writing a 1 to the TC bit location in the ESCIx_SR.

24.4.8.2.3 RDRF Description

The receive data register full (RDRF) interrupt is set when the data in the receive shift register transfers to the eSCI data register. An RDRF interrupt indicates that the received data has been transferred to the eSCI data register and that the received data can now be read by the MCU. The RDRF bit is cleared by writing a one to the RDRF bit location in the ESCIx_SR.

24.4.8.2.4 OR Description

The overrun (OR) interrupt is set when software fails to read the eSCI data register before the receive shift register receives the next frame. The newly acquired data in the shift register is lost in this case, but the data already in the eSCI data registers is not affected. The OR bit is cleared by writing a 1 to the OR bit location in the ESCIx_SR.

24.4.8.2.5 IDLE Description

The idle line (IDLE) interrupt is set when 10 consecutive logic 1s (if M = 0) or 11 consecutive logic 1s (if M = 1) appear on the receiver input. After the IDLE is cleared, a valid frame must again set the RDRF flag before an idle condition can set the IDLE flag. The IDLE bit is cleared by writing a 1 to the IDLE bit location in the ESCIx_SR.

24.4.8.2.6 PF Description

The interrupt is set when the parity of the received data is not correct. Writing a 1 clears the PF.

24.4.8.2.7 FE Description

The interrupt is set when the stop bit is read as a 0, which violates the SCI protocol. Writing a 1 clears the FE.

24.4.8.2.8 NF Description

The NF interrupt is set when the eSCI detects noise on the receiver input.

24.4.8.2.9 BERR Description

While the eSCI is in LIN mode, the bit error (BERR) flag is set when one or more bits in the last transmitted byte is not read back with the same value. The BERR flag is cleared by writing a 1 to the bit. A bit error will cause the LIN FSM to reset. Writing a 1 to the bit clears the BERR flag.

24.4.8.2.10 RXRDY Description

While in LIN mode, the receiver ready (RXRDY) flag is set when the eSCI receives a valid data byte in an RX frame. RXRDY will not be set for bytes that the receiver obtains by reading back the data which the LIN finite state machine (FSM) has sent out. Writing a 1 to the bit clears the RXRDY flag.

24.4.8.2.11 TXRDY Description

While in LIN mode, the transmitter ready (TXRDY) flag is set when the eSCI can accept a control or data byte. Writing a 1 to the bit clears the TXRDY flag.

24.4.8.2.12 LWAKE Description

The LIN wakeup (LWAKE) flag is set when the LIN hardware receives a wakeup character sent by one of the LIN slaves. This occurs only when the LIN bus is in sleep mode. Writing a 1 to the bit clears the LWAKE flag.

24.4.8.2.13 STO Description

The slave timeout (STO) flag is set during an RX frame when the LIN slave has not transmitted all requested data bytes before the specified timeout period. Writing a 1 to the bit clears the STO flag.

24.4.8.2.14 PBERR Description

If the RXD input remains stuck at a fixed value for 15 cycles after a transmission has started, the LIN hardware sets the physical bus error (PBERR) flag. Writing a 1 to the bit clears the PBERR flag.

24.4.8.2.15 CERR Description

If an RX frame has the CRC checking flag set and the two CRC bytes do not match the calculated CRC pattern, the CRC error (CERR) flag is set. Writing a 1 to the bit clears the CERR flag.

24.4.8.2.16 CKERR Description

If an RX frame has the checksum checking flag set and the last byte does not match the calculated checksum, the checksum error (CKERR) flag is set. Writing a 1 to the bit clears the CKERR flag.

24.4.8.2.17 FRC Description

The frame complete (FRC) flag is set after the last byte of a TX frame is sent out, or after the last byte of an RX frame is received. Writing a 1 to the bit clears the FRC flag.

NOTE

The last byte of a TX frame being sent or an RX frame being received indicates that the checksum comparison has taken place.

NOTE

The FRC flag is used to indicate to the CPU that the next frame can be set up. However, it might be set before the DMA controller has transferred the last byte from the eSCI to system memory. The FRC flag should not be used if the intention is to process data. Instead, the appropriate interrupt of the DMA controller should be used.

24.4.8.2.18 OVFL Description

The overflow (OVFL) flag is set when a byte is received in the ESCIx_LRR before the previous byte is read. Because the system is responsible for reading the register before the next byte arrives, this condition indicates a problem with CPU load. The OVFL flag is cleared by writing a 1 to the bit.

24.4.9 Using the LIN Hardware

The eSCI provides special support for the LIN protocol. It can be used to automate most tasks of a LIN master. In conjunction with the DMA interface it is possible to transmit entire frames (or sequences of frames) and receive data from LIN slaves without any CPU intervention. There is no special support for LIN slave mode. If required, LIN slave mode may be implemented in software.

A LIN frame consists of a break character (10 or 13 bits), a sync field, an ID field, n data fields (n could be 0) and a checksum field. The data and checksum bytes are either provided by the LIN master (TX frame) or by the LIN slave (RX frame). The header fields will always be generated by the LIN master.



Figure 24-23. Typical LIN frame

The LIN hardware is highly configurable. This configurability allows the eSCI's LIN hardware to generate frames for LIN slaves from all revisions of the LIN standard. The settings are adjusted according to the capabilities of the slave device.

To activate the LIN hardware, the LIN mode bit in the ESCIx_LCR needs to be set. Other settings, such as double stop flags after bit errors and automatic parity bit generation, are also available for use in LIN mode.

The eSCI settings must be made according to the LIN specification. The eSCI must be configured for 2-wire operation (2 wires connected to the LIN transceiver) with 8 data bytes and no parity. Normally a 13-bit break is used, but the eSCI can also be configured for 10-bit breaks as required by the application.

24.4.9.1 Features of the LIN Hardware

The eSCI's LIN hardware has several features to support different revisions of the LIN slaves. The ESCIx_LTR can be configured to include or not include header bits in the checksum on a frame by frame basis. This feature supports LIN slaves with different LIN revisions. The LIN control register allows the user to decide whether the parity bits in the ID field should be calculated automatically and whether double

stop flags should be inserted after a bit error. The BRK13 bit in ESCIx_CR2 decides whether to generate 10 or 13 bit break characters.

NOTE

LIN 2.0 now requires that a break character is always 13 bits long, so the BRK13 bit should always be set to 1. The eSCI will work with BRK13=0, but it will violate LIN 2.0.

The application software can decide to turn off the checksum generation/verification on a per frame basis and manage that function on its own. The application software can also decide to let the LIN hardware append two CRC bytes (Figure 24-24). The CRC bytes are not part of the LIN standard, but could be part of the application layer, that is they would be treated as data bytes by the LIN protocol. This can be useful when very long frames are transmitted. By default the CRC polynomial used is the same polynomial as for the CAN protocol.



Figure 24-24. LIN Frame with CRC bytes

It is possible to force a resync of the LIN FSM, with the LRES bit in the LIN control register. However, under normal circumstances, the LIN hardware will automatically abort a frame after detecting a bit error.

24.4.9.2 Generating a TX Frame

The following procedure describes how a basic TX frame is generated.

The frame is controlled via the LIN transmit register (ESCIx_LTR). Initially, the application software will need to check the TXRDY bit (either using an interrupt, the TX DMA interface, or by polling the LIN status register). If TXRDY is set, the register is writable. Before each write, TXRDY must be checked (though this step is performed automatically in DMA mode). The first write to the ESCIx_LTR must contain the LIN ID field. The next write to ESCIx_LTR specifies the length of the frame (0 to 255 Bytes). The third write to ESCIx_LTR contains the control byte (frame direction, checksum/CRC settings). Timeout bits are not included in TX frames because they refer to LIN slaves only. The three previously mentioned writes to the ESCIx_LTR specify the LIN frame data. After the LIN frame data is specified, the eSCI LIN hardware starts to generate a LIN frame.

First, the eSCI transmits a break field. The sync field is transmitted next. The third field is the ID field. After these three fields have been broadcast, the ESCIx_LTR accepts data bytes; the LIN hardware transmits these data bytes as soon as they are available and can be sent out. After the last step, the LIN hardware automatically appends the checksum field.

It is possible to set up a DMA channel to manage all the tasks required to send a TX frame (see Figure 24-25). For this operation, the TX DMA channel must be activated by setting the ESCIx_CR2[TXDMA] bit. The control information for the LIN frame (ID, message length, TX/RX type, timeout, etc.) and the data bytes are stored at an appropriate memory location. The DMA controller is then set up to transfer this block of memory to a location (the ESCIx_LTR). After transmission is complete, either the DMA controller or the LIN hardware can generate an interrupt to the CPU.

NOTE

In contrast to the standard software implementation where each byte transmission requires several interrupts, the DMA controller and eSCI manage communication, bit error and physical bus error checking, checksum, and CRC generation (checking on the RX side).

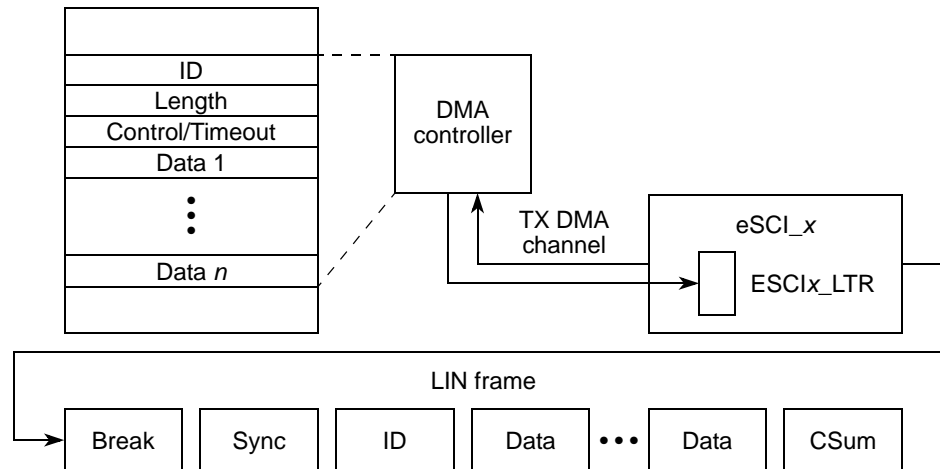


Figure 24-25. DMA Transfer of a TX Frame

24.4.9.3 Generating an RX Frame

For RX frames, the header information is provided by the LIN master. The data, CRC, and checksum bytes (as enabled) are provided by the LIN slave. The LIN master verifies CRC and checksum bytes transmitted by the slave.

For an RX frame, control information must be written to the ESCIx_LTR in the same manner as for the TX frames. Additionally the timeout bits, which define the time to complete the entire frame, must be written. Then the ESCIx_SR[RXRDY] bit must be checked (either with an interrupt, RX DMA interface, or by polling) to detect incoming data bytes. The checksum byte normally does not appear in the ESCIx_LRR, instead the LIN hardware will verify the checksum and issue an interrupt, if the checksum value is not correct.

Two DMA channels can be used when executing an RX frame: one to transfer the header/control information from a memory location to the ESCIx_LTR, and one to transfer the incoming data bytes from the ESCIx_LRR to a table in memory. See [Figure 24-26](#) for more information. After the last byte from the RX frame has been stored, the DMA controller can indicate completion to the CPU.

NOTE

It is also possible to set up a whole sequence of RX and TX frames, and generate a single event at the end of that sequence.

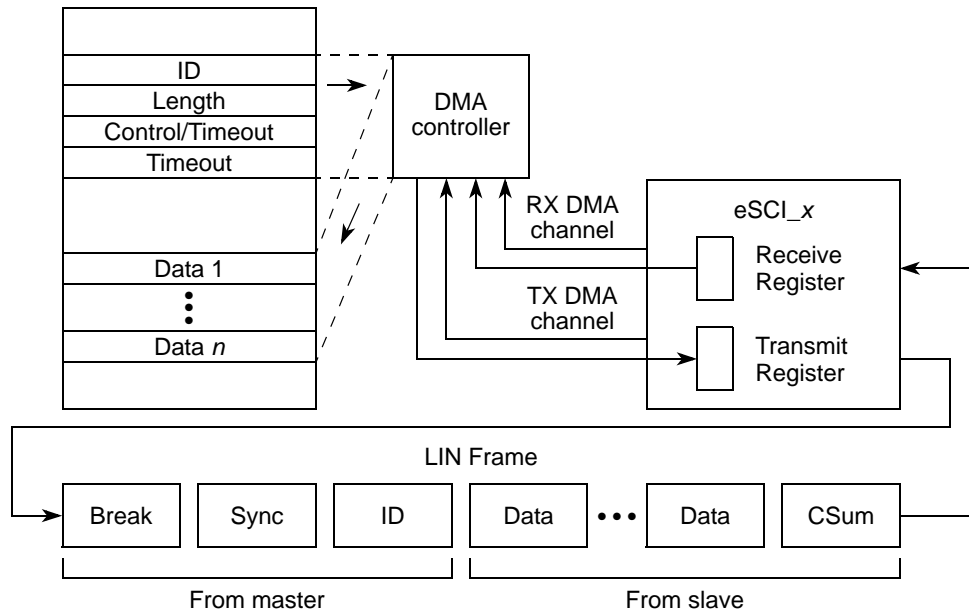


Figure 24-26. DMA Transfer of an RX Frame

24.4.9.4 LIN Error Handling

The LIN hardware can detect several error conditions of the LIN protocol. LIN hardware will receive every byte that was transmitted and compare it with the intended values. If there is a mismatch, a bit error is issued, and the LIN FSM will return to its start state.

For an RX frame the LIN hardware can detect a slave timeout error. The exact slave timeout error value can be set via the timeout bits in the ESCI_x_LTR. If the frame is not complete within the number of clock cycles specified in the register, the LIN FSM will return to its start state, and the STO interrupt is issued.

The LIN protocol supports a sleep mode. After 25,000 bus cycles of inactivity, the bus is assumed to be in sleep mode. Normally entering sleep mode can be avoided if the LIN master is regularly creating some bus activity. Otherwise the timeout state needs to be detected by the application software; for example, by setting a timer.

Both LIN masters and LIN slaves can cause the bus to exit sleep mode by sending a break signal. The LIN hardware will generate such a break, when WU bit in the LIN control register is written. After transmitting this break the LIN hardware will not send out data (that is, not raise the TXRDY flag) before the wakeup delimiter period has expired. This period can be selected by setting the WUD bits in the LIN control register.

Break signals sent by a LIN slave are received by the LIN hardware, and so indicated by setting the WAKE flag in the LIN status register.

A physical bus error (LIN bus is permanently stuck at a fixed value) will set several error flags. If the input is permanently low, the eSCI will set the framing error (FE) flag in the eSCI status register. If the RXD input remains stuck at a fixed value for 15 cycles, after a transmission has started, the LIN hardware will set the PBERR flag in the LIN status register. In addition a bit error may be generated.

24.4.9.5 LIN Setup

Because the eSCI is for general-purpose use, some of the settings are not applicable for LIN operation. The following setup applies for most applications, regardless of which kind of LIN slave is addressed:

- The module is enabled by writing the ESCIx_CR2[MDIS] bit to 0.
- Both transmitter and receiver are enabled (ESCIx_CR1[TE] = 1, ESCIx_CR1[RE] = 1).
- The data format bit ESCIx_CR1[M], is set to 0 (8 data bits), and the parity is disabled (PE = 0).
- ESCIx_CR1[TIE], ESCIx_CR1[TCIE], ESCIx_CR1[RIE] interrupt enable bits should be inactive. Instead, the LIN interrupts should be used.
- Switch eSCI to LIN mode (ESCIx_LCR[LIN] = 1).
- The LIN standard requires that the break character always be 13 bits long (ESCIx_CR2[BRK13] = 1). The eSCI will work with BRK13=0, but it will violate LIN 2.0.
- Normally, bit errors should cause the LIN FSM to reset, stop driving the bus immediately, and stop further DMA requests until the BERR flag has been cleared. Set ESCIx_LCR[LDBG] = 0, ESCIx_CR2[SBSTP] = 1, and ESCIx_CR2[BSTP] = 1 to accomplish these functions.
- Fast bit error detection provides superior error checking, so ESCIx_CR2[FBR] should be set; normally it will be used with ESCIx_CR2[BESM13] = 1.
- If available, a pulldown should be enabled on the RX input. (If the transceiver fails, the RX pin will not float).
- The error indicators NF, FE, BERR, STO, PBERR, CERR, CKERR, and OVFL should be enabled.
- Initially a wakeup character may need to be transmitted on the LIN bus, so that the LIN slaves activate.

Other settings such as baud rate, length of break character etc., depend on the LIN slaves to which the eSCI is connected.

Chapter 25

Controller Area Network (FlexCAN)

25.1 Introduction

The MPC5510 contains six controller area network (FlexCAN) blocks. Each FlexCAN module is a communication controller implementing the CAN protocol according to Bosch Specification version 2.0B and ISO Standard 11898. The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth.

The CAN protocol interface (CPI) submodule manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The message buffer management (MBM) submodule handles message buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The bus interface unit (BIU) submodule controls the access to and from the internal interface bus, to establish connection to the CPU and other blocks. Clocks, address and data buses, interrupt outputs, and test signals are accessed through the bus interface unit.

25.1.1 Block Diagram

A simplified block diagram of the FlexCAN illustrates the functionality and interdependence of major sub-blocks (see [Figure 25-1](#)).

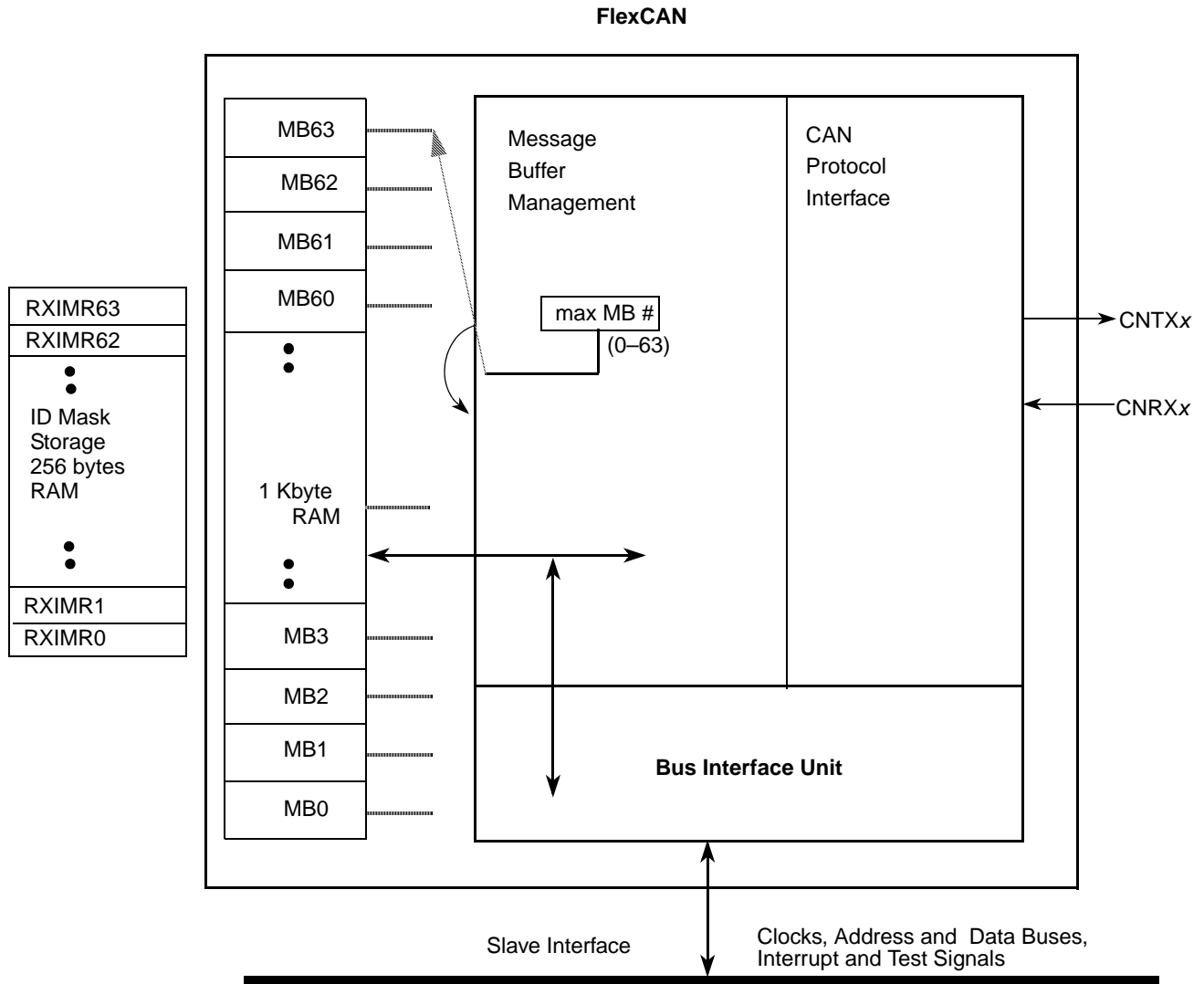


Figure 25-1. FlexCAN Block Diagram

25.1.2 Features

The FlexCAN has these major features:

- Full implementation of the CAN protocol specification, Version 2.0A/B
 - Standard data and remote frames
 - Extended data and remote frames
 - Zero to eight bytes data length
 - Programmable bit rate up to 1 Mbit/sec
 - Content-related addressing
- 64 flexible message buffers (MBs) of zero to eight bytes data length
- Each message buffer configurable as Rx or Tx, all supporting standard and extended messages

- Individual Rx mask registers per message buffer
- Includes 1056 bytes of RAM used for message buffer storage
- Includes 256 bytes of RAM used for individual Rx mask registers
- Full featured Rx FIFO with storage capacity for six frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either eight extended, 16 standard, or 32 partial (8 bits) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN version
- Programmable clock source to the CAN protocol interface, either bus clock or crystal oscillator
- Unused message buffer and Rx mask register space can be used as general-purpose RAM space
- Listen-only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number or local priority on individual Tx message buffers.
- Hardware cancellation on Tx message buffers.
- Time stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low-power modes

25.1.3 Modes of Operation

There are four main operating modes of FlexCAN: normal, freeze, listen-only, and loop-back. Two low-power modes are supported: module disable and stop. For more details, refer to [Section 25.4.8, “Modes of Operation Details.”](#)

25.1.3.1 Normal Mode

In normal mode the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN protocol functions are enabled. In the MCU, there is no distinction between user and supervisor modes.

25.1.3.2 Freeze Mode

Freeze mode is entered when the FRZ bit in the module configuration register (CANx_MCR) is asserted, while the HALT bit in CANx_MCR is set, or if debug mode is requested by either core. In freeze mode no transmission or reception of frames is done, and synchronicity to the CAN bus is lost.

25.1.3.3 Listen-Only Mode

In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN error passive mode. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.

25.1.3.4 Loop-Back Mode

The module enters this mode when the LPB bit in the control register is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Transmit and receive interrupts are generated.

25.1.3.5 Module-Disabled Mode

This low-power mode is entered when the MDIS bit in the CAN_x_MCR register is asserted. When disabled, the clocks to the CAN protocol interface and message buffer management submodules are shut down. Exit from this mode is done by negating the MDIS bit in the CAN_x_MCR register.

25.1.3.6 Stop Mode

This low-power mode is entered when stop mode is requested at MCU level. When in stop mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally. Exit from this mode happens when the stop mode request is removed or when activity is detected on the CAN bus and the self wake up mechanism is enabled.

25.2 External Signal Description

Please refer to [Table 2-1](#) and [Chapter 2, “Signal Descriptions,”](#) for a complete description of the FlexCAN signals.

25.3 Memory Map and Registers

This section provides a detailed description of all FlexCAN registers.

25.3.1 Module Memory Map

The complete memory map for an individual FlexCAN module is shown in [Table 25-1](#). Except for the base addresses, all FlexCAN modules have identical memory maps.

The Rx global mask (CAN_x_RXGMASK), Rx buffer 14 mask (CAN_x_RX14MASK) and the Rx buffer 15 mask (CAN_x_RX15MASK) registers are provided for backwards compatibility, and are not used when the BCC bit in CAN_x_MCR is asserted.

The offset address ranges 0x0060–0x047F and 0x0880–0x097F are occupied by two separate embedded memories. These two ranges are completely occupied by RAM (1 KB and 256 bytes, respectively) when FlexCAN is configured with 64 MBs. Furthermore, if the BCC bit in CAN_x_MCR is negated, then the whole Rx individual mask registers address range (0x0880–0x097F) is considered reserved space.

Table 25-1. FlexCAN Memory Map

Offset from FlexCAN_BASE (FlexCAN_A = 0xFFFC_0000 FlexCAN_B = 0xFFFC_4000 FlexCAN_C = 0xFFFC_8000 FlexCAN_D = 0xFFFC_C000 FlexCAN_E = 0xFFFD_0000 FlexCAN_F = 0xFFFD_4000)	Register	Access	Reset Value ¹	Section/Page
0x0000	CAN _x _MCR — Module Configuration	R/W	Note1	25.3.4.1/25-11
0x0004	CAN _x _CTRL — Control Register	R/W	Note1	25.3.4.2/25-15
0x0008	CAN _x _TIMER — Free-running Timer	R/W	Note1	25.3.4.3/25-18
0x000C	Reserved			
0x0010	CAN _x _RXGMASK — Rx Global Mask	R/W	Note1	25.3.4.4.1/25-19
0x0014	CAN _x _RX14MASK — Rx Buffer 14 Mask	R/W	Note1	25.3.4.4.2/25-20
0x0018	CAN _x _RX15MASK — Rx Buffer 15 Mask	R/W	Note1	25.3.4.4.3/25-20
0x001C	CAN _x _ECR — Error Counter Register	R/W	Note1	25.3.4.5/25-21
0x0020	CAN _x _ESR — Error and Status Register	R/W	Note1	25.3.4.6/25-22
0x0024	CAN _x _IMASK2 — Interrupt Masks 2	R/W	Note1	25.3.4.7/25-24
0x0028	CAN _x _IMASK1 — Interrupt Masks 1	R/W	Note1	25.3.4.8/25-25
0x002C	CAN _x _IFLAG2 — Interrupt Flags 2	R/W	Note1	25.3.4.9/25-25
0x0030	CAN _x _IFLAG1 — Interrupt Flags 1	R/W	Note1	25.3.4.10/25-26
0x0034–0x007F	Reserved			
0x0080–0x017F	MB0–MB15 — Message Buffers	R/W	Note1	25.3.2/25-6
0x0180–0x027F	MB16–MB31 — Message Buffers	R/W	Note1	
0x0280–0x047F	MB32–MB63 — Message Buffers	R/W	Note1	
0x0480–087F	Reserved			

Table 25-1. FlexCAN Memory Map (continued)

Offset from FlexCAN_BASE (FlexCAN_A = 0xFFFC_0000 FlexCAN_B = 0xFFFC_4000 FlexCAN_C = 0xFFFC_8000 FlexCAN_D = 0xFFFC_C000 FlexCAN_E = 0xFFFD_0000 FlexCAN_F = 0xFFFD_4000)	Register	Access	Reset Value ¹	Section/Page
0x0880-0x08BF	CANx_RXIMR0–CANx_RXIMR15 — Rx Individual Mask Registers	R/W	Note1	25.3.4.11/25-27
0x08C0-0x08FF	CANx_RXIMR16–CANx_RXIMR31 — Rx Individual Mask Registers	R/W	Note1	
0x0900-0x097F	CANx_RXIMR32–CANx_RXIMR63 — Rx Individual Mask Registers	R/W	Note1	

¹ Please refer to the register definition.

The FlexCAN module stores CAN messages for transmission and reception using a message buffer structure. Each MB is formed by 16 bytes mapped in memory as described in Table 25-2. The FlexCAN module can manage up to 64 message buffers. Table 25-2 shows a standard/extended message buffer (MB0) memory map, using 16 bytes (0x80–0x8F) total space.

Table 25-2. Message Buffer MB0 Memory Mapping

Address Offset	MB Field
0x80	Control and status (C/S)
0x84	Identifier field
0x88–0x8F	Data fields 0–7 (1 byte each)

NOTE

Reading the C/S word of a message buffer (the first word of each MB) will lock it, preventing it from receiving further messages until it is unlocked either by reading another MB or by reading the timer.

25.3.2 Message Buffer Structure

The message buffer structure used by the FlexCAN module is represented in Figure 25-2. Both extended and standard frames (29-bit identifier and 11-bit identifier, respectively) used in the CAN specification (version 2.0 Part B) are represented.

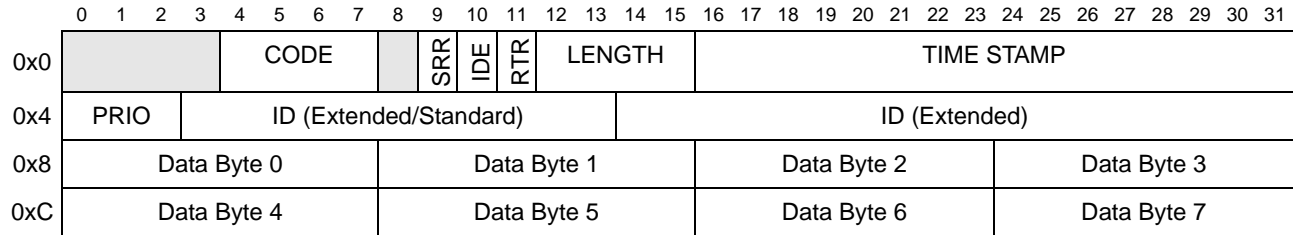


Figure 25-2. Message Buffer Structure

Table 25-3. Message Buffer Field Descriptions

Name	Description
CODE	Message Buffer Code. This 4-bit field can be accessed (read or write) by the CPU and by the FlexCAN module itself, as part of the message buffer matching and arbitration process. The encoding is shown in Table 25-4 and Table 25-5 . See Section 25.4, "Functional Description," for additional information.
SRR	Substitute Remote Request. Fixed recessive bit, used only in extended format. It must be set to '1' by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 0 Dominant is not a valid value for transmission in extended format frames 1 Recessive value is compulsory for transmission in extended format frames
IDE	ID Extended Bit. This bit identifies whether the frame format is standard or extended. 0 Frame format is standard 1 Frame format is extended
RTR	Remote Transmission Request. This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as '1' (recessive) and receives it as '0' (dominant), it is interpreted as arbitration loss. If this bit is transmitted as '0' (dominant), then if it is received as '1' (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 0 Indicates the current MB has a data frame to be transmitted 1 Indicates the current MB has a remote frame to be transmitted
LENGTH	Length of Data in Bytes. This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the MB space (see Figure 25-2). In reception, this field is written by the FlexCAN module, copied from the DLC (data length code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR = 1, the Frame to be transmitted is a remote frame and does not include the data field, regardless of the length field.
TIME STAMP	Free-Running Counter Time Stamp. This 16-bit field is a copy of the free-running timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.
PRIO	Local Priority. This 3-bit field is only used when LPRIO_EN bit is set in CANx_MCR and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See Section 25.4.2, "Arbitration Process."
ID	Frame Identifier. In standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In extended frame format, all bits are used for frame identification in both receive and transmit cases.
DATA	Data Field. Up to eight bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.

Table 25-4. Message Buffer Code for Rx Buffers

Rx Code before Rx New Frame	Description	Rx Code after Rx New Frame	Comment
0000	NOT ACTIVE: MB is not active.	—	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to Section 25.4.4, "Matching Process" for details about overrun behavior.
0110	OVERRUN: A frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB will be overwritten again, and the code will remain OVERRUN. Refer to Section 25.4.4, "Matching Process" for details about overrun behavior.
0XY1 ¹	BUSY: FlexCAN is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

¹ Note that for Tx MBs (see [Table 25-5](#)), the BUSY bit should be ignored upon read, except when AEN bit is set in the CANx_MCR.

Table 25-5. Message Buffer Code for Tx Buffers

RTR	Initial Tx Code	Code after Successful Transmission	Description
X	1000	—	INACTIVE: MB does not participate in the arbitration process.
X	1001	—	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when AEN bit in CANx_MCR is asserted. MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes and Rx MB with the same ID.

Table 25-5. Message Buffer Code for Tx Buffers (continued)

RTR	Initial Tx Code	Code after Successful Transmission	Description
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the CODE field is automatically updated to '1110' to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to '1010' to restart the process again.
0	1110	1010	the MBM as a result of match to a remote request frame. The data frame will be transmitted unconditionally once and then the code will automatically return to '1010'. The CPU can also write this code with the same effect.

25.3.3 Rx FIFO Structure

When the FEN bit is set in the CAN_x_MCR, the memory area from 0x80 to 0xFF (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. [Figure 25-3](#) shows the Rx FIFO data structure. The region 0x0–0xC contains an MB structure which is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x10–0xDF is reserved for internal use of the FIFO engine. The region 0xE0–0xFF contains an eight-entry ID table that specifies filtering criteria for accepting frames into the FIFO. [Figure 25-4](#) shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the CAN_x_MCR. Note that all elements of the table must have the same format. See [Section 25.4.6, “Rx FIFO,”](#) for more information.

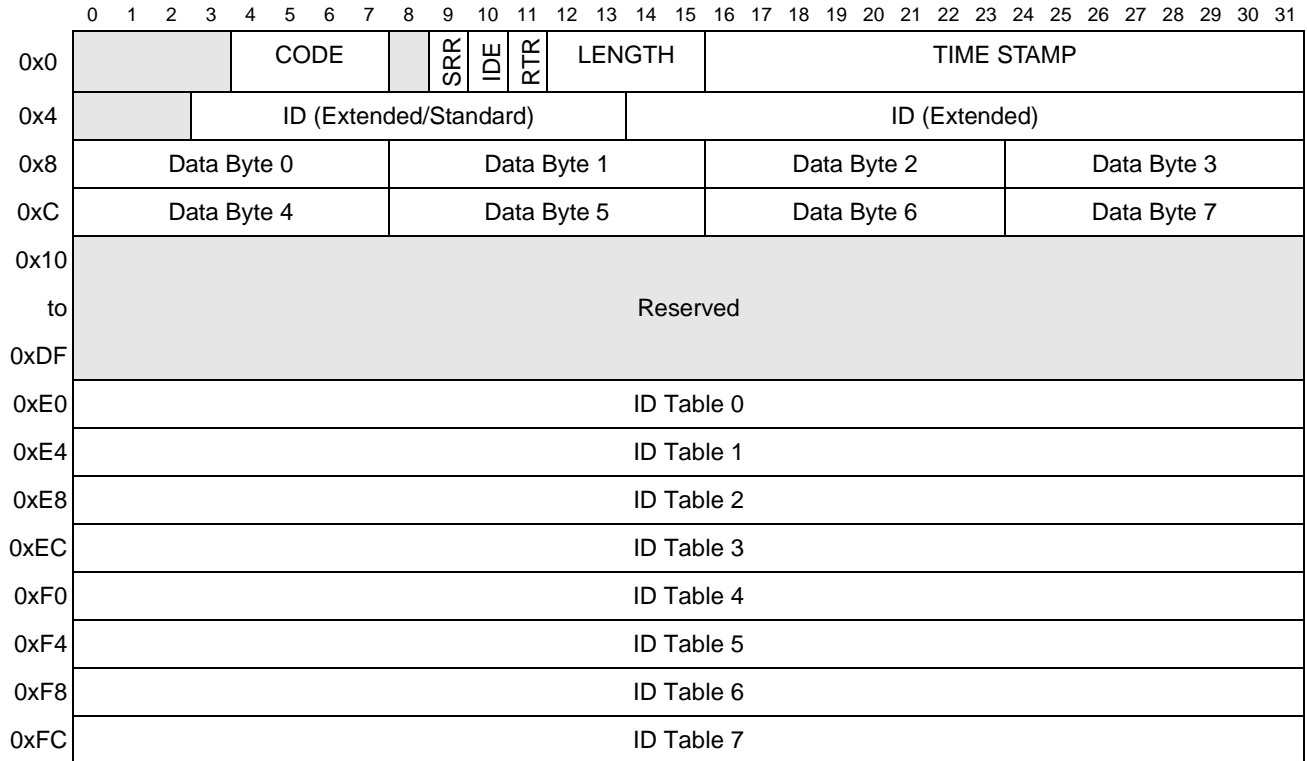


Figure 25-3. Rx FIFO Structure

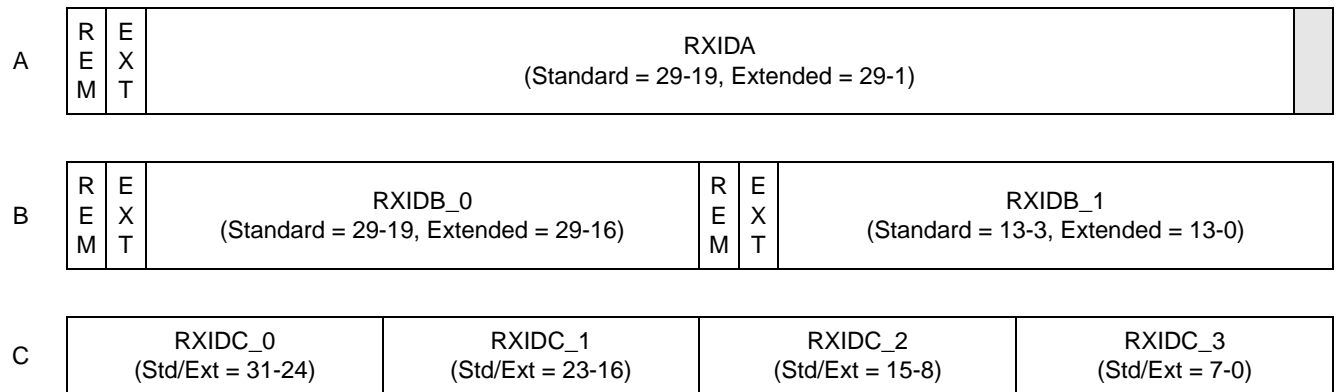


Figure 25-4. ID Table 0–7

Table 25-6. ID Table 0–7 Field Descriptions

Name	Description
REM	Remote Frame. This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID. 0 Remote frames are rejected and data frames can be accepted. 1 Remote frames can be accepted and data frames are rejected.
EXT	Extended Frame. Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 0 Extended frames are rejected and standard frames can be accepted. 1 Extended frames can be accepted and standard frames are rejected.
RXIDA	Rx Frame Identifier (Format A). Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification. In the extended frame format, all bits are used.
RXIDB_0, RXIDB_1	Rx Frame Identifier (Format B). Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (3 to 13) are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.
RXIDC_0, RXIDC_1, RXIDC_2, RXIDC_3	Rx Frame Identifier (Format C). Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

25.3.4 Register Descriptions

This section lists the FlexCAN registers in address order and describes the registers and their bit fields.

25.3.4.1 Module Configuration Register (CANx_MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in freeze mode.

Offset: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FRZ	FEN	HALT	NOT_RDY	0	SOFT_RST	FRZ_ACK	1	0	WRN_EN	LPM_ACK	0	0	SRX_DIS	BCC
W																
Reset	1	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	LPRIO_EN	AEN	0	0	IDAM		0	0	MAXMB					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 25-5. Module Configuration Register (CANx_MCR)

Table 25-7. CANx_MCR Field Descriptions

Field	Description
MDIS	Module Disable. Controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clock to the CAN protocol interface and message buffer management submodules. This is the only bit in CANx_MCR not affected by soft reset. See Section 25.4.8.2, “Module Disabled Mode,” for more information. 0 Enable the FlexCAN module. 1 Disable the FlexCAN module.
FRZ	Freeze Enable. Specifies the FlexCAN behavior when the HALT bit in the CANx_MCR is set or when debug mode is requested at MCU level. When FRZ is asserted, FlexCAN is enabled to enter freeze mode. Negation of this bit field causes FlexCAN to exit from freeze mode. 0 Not enabled to enter freeze mode. 1 Enabled to enter freeze mode.
FEN	FIFO Enable. controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0 to 7 cannot be used for normal reception and transmission because the corresponding memory region (0x80–0xFF) is used by the FIFO engine. See Section 25.3.3, “Rx FIFO Structure,” and Section 25.4.6, “Rx FIFO,” for more information. 0 FIFO not enabled. 1 FIFO enabled.
HALT	Halt FlexCAN. Assertion of this bit puts the FlexCAN module into freeze mode if FRZ is asserted. The CPU will clear it after initializing the message buffers and CANx_CTRL. If FRZ is set, no reception or transmission is performed by FlexCAN before this bit is cleared. While in freeze mode, the CPU has write access to the CANx_ECR, that is otherwise read-only. Freeze mode cannot be entered while FlexCAN is disabled. See Section 25.4.8.1, “Freeze Mode,” for more information. 0 No freeze mode request. 1 Enters freeze mode if the FRZ bit is asserted.
NOTRDY	FlexCAN Not Ready. Indicates that FlexCAN is either disabled or in freeze mode. It is negated once FlexCAN has exited these modes. 0 FlexCAN module is either in normal mode, listen-only mode or loop-back mode. 1 FlexCAN module is either disabled or freeze mode.
bit 5	Reserved.

Table 25-7. CANx_MCR Field Descriptions (continued)

Field	Description
SOFTRST	<p>Soft Reset. When asserted, FlexCAN resets its internal state machines and some of the memory-mapped registers. The following registers are affected by soft reset:</p> <ul style="list-style-type: none"> • CANx_MCR (except the MDIS bit) • CANx_TIMER • CANx_ECR • CANx_ESR • CANx_IMASK1 • CANx_IMASK2 • CANx_IFLAG1 • CANx_IFLAG2 <p>Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected:</p> <ul style="list-style-type: none"> • CANx_CTRL • CANx_RXGMASK • CANx_RX14MASK • CANx_RX15MASK • all message buffers <p>The SOFTRST bit can be asserted directly by the CPU when it writes to the CANx_MCR, but it is also asserted when global soft reset is requested at MCU level. Because soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFTRST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed.</p> <p>0 No reset request. 1 Resets values in registers indicated above.</p>
FRZACK	<p>Freeze Mode Acknowledge. Indicates that FlexCAN is in freeze mode and its prescaler is stopped. The freeze mode request cannot be granted until current transmission and reception processes have finished. Therefore the software can poll the FRZACK bit to know when FlexCAN has actually entered freeze mode. If freeze mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If freeze mode is requested while FlexCAN is disabled, then the FRZACK bit will only be set when the low-power mode is exited. See Section 25.4.8.1, “Freeze Mode,” for more information.</p> <p>0 FlexCAN not in freeze mode, prescaler running. 1 FlexCAN in freeze mode, prescaler stopped.</p>
bits 8–9	Reserved.
WRNEN	<p>Warning Interrupt Enable. When asserted, this bit enables the generation of the TWRNINT and RWRNINT flags in the error and status register. If WRNEN is negated, the TWRNINT and RWRNINT flags will always be 0, independent of the values of the error counters, and no warning interrupt will ever be generated.</p> <p>1 = TWRNINT and RWRNINT bits are set when the respective error counter transition from <96 to ≥ 96. 0 = TWRNINT and RWRNINT bits are zero, independent of the values in the error counters.</p>
LPM_SACK	<p>Low-Power Mode Acknowledge. Indicates whether FlexCAN is disabled. This cannot be performed until all current transmission and reception processes have finished, so the CPU can poll the MDISACK bit to know when FlexCAN has actually been disabled. See Section 25.4.8.2, “Module Disabled Mode,” for more information.</p> <p>0 FlexCAN not disabled. 1 FlexCAN is disabled.</p>
bits 12–13	Reserved.

Table 25-7. CANx_MCR Field Descriptions (continued)

Field	Description															
SRXDIS	This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module will not be stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal will be generated due to the frame reception. 0 Self reception enabled. 1 Self reception disabled.															
BCC	Backwards Compatibility Configuration. Provided to support backwards compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied: <ul style="list-style-type: none"> For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per MB, FlexCAN uses its previous masking scheme with CANx_RXGMASK, CANx_RX14MASK and CANx_RX15MASK. The reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, FlexCAN will not look for another matching MB. It will override this MB with the new message and set the CODE field to 0b0110 (overrun). Upon reset this bit is negated, allowing legacy software to work without modification. 0 Individual Rx masking and queue feature are disabled. 1 Individual Rx masking and queue feature are enabled.															
bits 16–17	Reserved.															
LPRIO_EN	Local Priority Enable. Provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It is used to extend the ID used during the arbitration process. With this extended ID concept, the arbitration process is done based on the full 32-bit word, but the actual transmitted ID still has 11 bits for standard frames and 29 bits for extended frames. 0 Local priority disabled. 1 Local priority enabled.															
AEN	Abort Enable. Provided for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification. 0 Abort disabled. 1 Abort enabled.															
bits 20–21	Reserved.															
IDAM	ID Acceptance Mode. Identifies the format of the elements of the Rx FIFO filter table. All elements of the table are configured at the same time by this field (they are all the same format). <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>IDAM</th> <th>Format</th> <th>Explanation</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>A</td> <td>One full ID (standard or extended) per filter element.</td> </tr> <tr> <td>01</td> <td>B</td> <td>Two full standard IDs or two partial 14-bit extended IDs per filter element.</td> </tr> <tr> <td>10</td> <td>C</td> <td>Four partial 8-bit IDs (standard or extended) per filter element.</td> </tr> <tr> <td>11</td> <td>D</td> <td>All frames rejected.</td> </tr> </tbody> </table>	IDAM	Format	Explanation	00	A	One full ID (standard or extended) per filter element.	01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.	10	C	Four partial 8-bit IDs (standard or extended) per filter element.	11	D	All frames rejected.
IDAM	Format	Explanation														
00	A	One full ID (standard or extended) per filter element.														
01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.														
10	C	Four partial 8-bit IDs (standard or extended) per filter element.														
11	D	All frames rejected.														

Table 25-7. CANx_MCR Field Descriptions (continued)

Field	Description
24–25	Reserved.
MAXMB	<p>Maximum Number Of Message Buffers. This 6-bit field defines the maximum number of message buffers of the FlexCAN module. The reset value (0x0F) is equivalent to a 16 MB configuration. This field should be changed only while the module is in freeze mode.</p> <p style="text-align: center;">Maximum MBs in use = MAXMB + 1</p> <p>Note: MAXMB has to be programmed with a value smaller or equal to the number of available message buffers, otherwise FlexCAN will not transmit or receive frames.</p>

25.3.4.2 Control Register (CANx_CTRL)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, loop back mode, listen only mode, bus off recovery behavior and interrupt enabling (bus-off, error, warning). It also determines the division factor for the clock prescaler. Most of the fields in this register should only be changed while the module is in disable mode or in freeze mode. Exceptions are the BOFF_MSK, ERR_MSK, TWRN_MSK, RWRN_MSK and BOFF_REC bits, that can be accessed at any time.

Offset: Base + 0x0004

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRESDIV								RJW	PSEG1			PSEG2			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BOFF_MSK	ERR_MSK	CLK_SRC	LPB	TWRN_MSK	RWRN_MSK	0	0	SMP	BOFF_REC	TSYN	LBUF	LOM	PROPSEG		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-6. Control Register (CANx_CTRL)

Table 25-8. CANx_CTRL Field Descriptions

Bits	Description
PRES DIV	<p>Prescaler Division Factor. Defines the ratio between the CPI clock frequency and the serial clock (SCK) frequency. The SCK period defines the time quantum of the CAN protocol. For the reset value, the SCK frequency is equal to the CPI clock frequency. The maximum value of this register is 0xFF, that gives a minimum SCK frequency equal to the CPI clock frequency divided by 256. For more information, refer to Section 25.4.7.4, "Protocol Timing."</p> $\text{S-clock frequency} = \frac{\text{CPI clock frequency}}{\text{PRES DIV} + 1}$
RJW	<p>Resync Jump Width. Defines the maximum number of time quanta¹ that a bit time can be changed by one re-synchronization. The valid programmable values are 0–3.</p> $\text{Resync Jump Width} = \text{RJW} + 1$
PSEG1	<p>Phase Segment 1. Defines the length of phase buffer segment 1 in the bit time. The valid programmable values are 0–7.</p> $\text{Phase Buffer Segment 1} = (\text{PSEG1} + 1) \times \text{Time Quanta}$
PSEG2	<p>Phase Segment 2. Defines the length of phase buffer segment 2 in the bit time. The valid programmable values are 1–7.</p> $\text{Phase Buffer Segment 2} = (\text{PSEG2} + 1) \times \text{Time Quanta}$
BOFFMSK	<p>Bus Off Mask. Provides a mask for the bus off interrupt.</p> <p>0 Bus off interrupt disabled 1 Bus off interrupt enabled</p>
ERRMSK	<p>Error Mask. Provides a mask for the error interrupt.</p> <p>0 Error interrupt disabled 1 Error interrupt enabled</p>
CLK_SRC	<p>CAN Engine Clock Source. Selects the clock source to the CAN Protocol Interface (CPI) to be either the system clock (driven by the PLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the serial clock (SCK). In order to guarantee reliable operation, this bit should only be changed while the module is disabled.</p> <p>0 = The CAN engine clock source is the oscillator clock 1 = The CAN engine clock source is the system clock</p>
LPB	<p>Loop Back. Configures FlexCAN to operate in loop-back mode. See Section 25.4.8, "Modes of Operation Details" for information about this operating mode.</p> <p>0 Loop back disabled 1 Loop back enabled</p>
TWRNMSK	<p>This bit provides a mask for the Tx Warning Interrupt associated with the TWRNINT flag in the Error and Status Register. This bit has no effect if the WRNEN bit in CANx_MCR is negated and it is read as zero when WRNEN is negated.</p> <p>1 = Tx Warning Interrupt enabled 0 = Tx Warning Interrupt disabled</p>

Table 25-8. CANx_CTRL Field Descriptions

Bits	Description
RWRNMSK	This bit provides a mask for the Rx Warning Interrupt associated with the RWRNINT flag in the Error and Status Register. This bit has no effect if the WRNEN bit in CANx_MCR is negated and it is read as zero when WRNEN is negated. 1 = Rx Warning Interrupt enabled 0 = Rx Warning Interrupt disabled
bits 22–23	Reserved.
SMP	Sampling Mode. Defines the sampling mode of each bit in the receiving messages (Rx). 0 Just one sample is used to determine the Rx bit value 1 Three samples are used to determine the value of the received bit: the regular one (sample point) and 2 preceding samples, a majority rule is used
BOFFREC	Bus Off Recovery Mode. Defines how FlexCAN recovers from bus off state. If this bit is negated, automatic recovering from bus off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from bus off is disabled and the module remains in bus off state until the bit is negated by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then bus off recovery happens as if the BOFFREC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN will re-synchronize to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFFREC bit can be re-asserted again during bus off, but it will only be effective the next time the module enters bus off. If BOFFREC was negated when the module entered bus off, asserting it during bus off will not be effective for the current bus off recovery. 0 Automatic recovering from bus off state enabled, according to CAN Spec 2.0 part B 1 Automatic recovering from bus off state disabled
TSYN	Timer Sync Mode. Enables a mechanism that resets the free-running timer each time a message is received in message buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special SYNC message (that is, global network time). If the FEN bit in CANx_MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. 0 Timer sync feature disabled 1 Timer sync feature enabled Note: There is a possibility of 4–5 ticks count skew between the different FlexCAN stations that would operate in this mode.
LBUF	Lowest Buffer Transmitted First. This bit defines the ordering mechanism for message buffer transmission. When asserted, the LPRIOR_EN bit does not affect the priority arbitration. 0 Buffer with highest priority is transmitted first 1 Lowest number buffer is transmitted first
LOM	Listen-Only Mode. Configures FlexCAN to operate in listen-only mode. In this mode, the FlexCAN module receives messages without giving any acknowledge. It is not possible to transmit any message in this mode. 0 FlexCAN module is in normal active operation, listen only mode is deactivated 1 FlexCAN module is in listen only mode operation
PROPSEG	Propagation Segment. Defines the length of the propagation segment in the bit time. The valid programmable values are 0–7. $\text{Propagation Segment Time} = (\text{PROPSEG} + 1) \times \text{Time Quanta}$ $\text{Time Quantum} = \text{one S clock period}$

¹ One time quantum is equal to the S clock period.

25.3.4.3 Free-Running Timer (CANx_TIMER)

This register represents a 16-bit free-running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

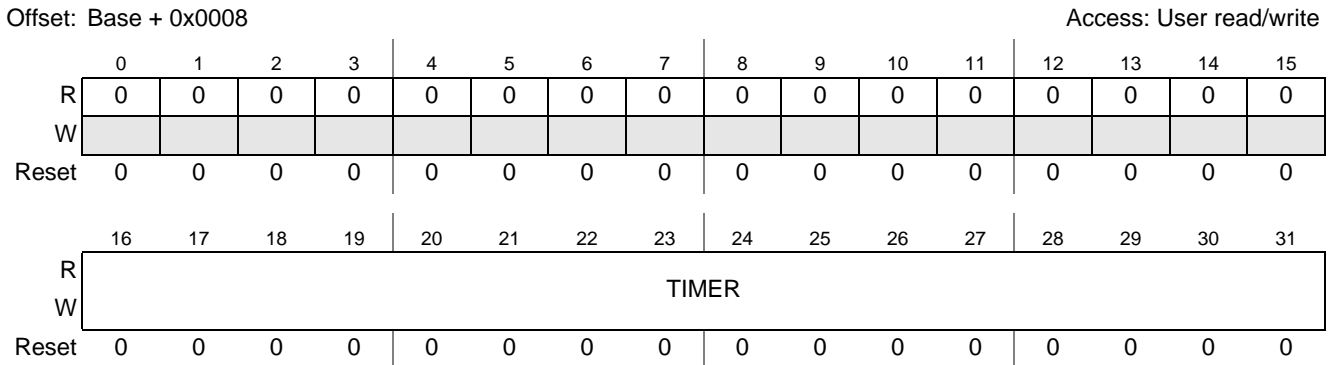


Figure 25-7. Free-Running Timer (CANx_TIMER)

25.3.4.4 Rx Mask Registers

By negating the CANx_MCR[BCC] bit, the CANx_RXGMASK, CANx_RX14MASK, and CANx_RX15MASK registers are used as acceptance masks for received frame IDs. Three masks are defined: a global mask, used for Rx buffers 0–13 and 16–63, and two extra masks dedicated for buffers 14 and 15. The meaning of each mask bit is the following:

- Mask bit = 0: the corresponding incoming ID bit is “don’t care.”
- Mask bit = 1: the corresponding ID bit is checked against the incoming ID bit, to see if a match exists.

Note that these masks are used both for standard and extended ID formats. The value of mask registers should not be changed while in normal operation. Locked frames which had matched a MB through a mask may be transferred into the MB (upon release) but may no longer match. Table 25-9 shows some examples of ID masking for standard and extended message buffers.

Table 25-9. Mask Examples for Standard/Extended Message Buffers

	Base ID ID28.....ID18	IDE	Extended ID ID17.....ID0	Match
MB2 ID	1 1 1 1 1 1 1 1 0 0 0	0		
MB3 ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB4 ID	0 0 0 0 0 0 1 1 1 1 1	0		
MB5 ID	0 0 0 0 0 0 1 1 1 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB14 ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
Rx Global Mask	1 1 1 1 1 1 1 1 1 1 0		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1	
Rx Msg in ¹	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	3
Rx Msg in ²	1 1 1 1 1 1 1 1 0 0 1	0		2
Rx Msg in ³	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0	
Rx Msg in ⁴	0 1 1 1 1 1 1 1 0 0 0	0		
Rx Msg in ⁵	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	14
Rx 14 Mask	0 1 1 1 1 1 1 1 1 1 1		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0	
Rx Msg in ⁶	1 0 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
Rx Msg in ⁷	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	14

¹ Match for Extended Format (MB3).

² Match for Standard Format. (MB2).

³ Mismatch for MB3 because of ID0.

⁴ Mismatch for MB2 because of ID28.

⁵ Mismatch for MB3 because of ID28, Match for MB14 (uses CANx_RX14MASK).

⁶ Mismatch for MB14 because of ID27 (uses CANx_RX14MASK).

⁷ Match for MB14 (uses CANx_RX14MASK).

25.3.4.4.1 Rx Global Mask (CANx_RXGMASK)

This register is provided for legacy support. On MPC5510, setting the BCC bit in CANx_MCR causes the CANx_RXGMASK Register to have no effect on the module operation.

CANx_RXGMASK is used as acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in CANx_MCR is set (FIFO enabled), the CANx_RXGMASK also applies to all elements of the ID filter table, except elements 6–7, which have individual masks.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Offset: Base + 0x0010

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 25-8. Rx Mask Register (CANx_RXGMASK)

Table 25-10. CANx_RXGMASK Field Descriptions

Field	Description
MI <i>n</i>	Mask Bits. For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 the corresponding bit in the filter is “don’t care” 1 The corresponding bit in the filter is checked against the one received

25.3.4.4.2 Rx 14 Mask (CANx_RX14MASK)

This register is provided for legacy support. On MPC5510, setting the BCC bit in CANx_MCR causes the CANx_RX14MASK Register to have no effect on the module operation.

CANx_RX14MASK is used as acceptance mask for the Identifier in Message Buffer 14. When the FEN bit in CANx_MCR is set (FIFO enabled), the CANx_RX14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x14
- Reset Value: 0xFFFF_FFFF

25.3.4.4.3 Rx 15 Mask (CANx_RX15MASK)

This register is provided for legacy support. On MPC5510, setting the BCC bit in CANx_MCR causes the CANx_RX15MASK Register to have no effect on the module operation.

When the BCC bit is negated, CANx_RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When the FEN bit in CANx_MCR is set (FIFO enabled), the CANx_RX14MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x18
- Reset Value: 0xFFFF_FFFF

25.3.4.5 Error Counter Register (CANx_ECR)

CANx_ECR has two 8-bit fields reflecting the value of two FlexCAN error counters: the transmit error counter (TXECTR field) and receive error counter (RXECTR field). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read only except in freeze mode, where they can be written by the CPU.

Writing to the CANx_ECR while in freeze mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol: transmitting, for example, an ‘error active’ or ‘error passive’ flag, delaying its transmission start time (‘error passive’), and avoiding any influence on the bus when in the bus off state. The following are the basic rules for FlexCAN bus state transitions:

- If the value of TXECTR or RXECTR increases to be greater than or equal to 128, the FLTCONF field in the CANx_ESR is updated to reflect the ‘error passive’ state.
- If the FlexCAN state is ‘error passive,’ and either TXECTR or RXECTR decrements to a value less than or equal to 127 while the other already satisfies this condition, the FLTCONF field in the CANx_ESR is updated to reflect the ‘error active’ state.
- If the value of TXECTR increases to be greater than 255, the FLTCONF field in the CANx_ESR is updated to reflect the bus off state, and an interrupt may be issued. The value of TXECTR is then reset to zero.
- If FlexCAN is in the bus off state, then TXECTR is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, TXECTR is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TXECTR. When TXECTR reaches the value of 128, the FLTCONF field in CANx_ESR is updated to be ‘error active’ and both error counters are reset to zero. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the TXECTR value.
- If during system start-up, only one node is operating, then its TXECTR increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ACKERR bit in CANx_ESR). After the transition to the ‘error passive’ state, the TXECTR does not increment anymore by acknowledge errors. Therefore the device never goes to the bus off state.

If the RXECTR increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to ‘error active’ state.

Offset: Base + 0x001C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Rx_Err_Counter								Tx_Err_Counter							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-9. Error Counter Register (CANx_ECR)

25.3.4.6 Error and Status Register (CANx_ESR)

This register reflects various error conditions, some general status of the device, and it is the source of four interrupts to the CPU. The reported error conditions (bits 16-21) are those that occurred since the last time the CPU read this register. The CPU read action clears bits 16-21. Bits 22-28 are status bits.

Most bits in this register are read only, except TWRN_INT, RWRN_INT, BOFF_INT, and ERR_INT, which are interrupt flags that can be cleared by writing 1 to them (writing 0 has no effect).

NOTE

A read clears BIT1ERR, BIT0ERR, ACKERR, CRCERR, FRMERR, and STFERR, therefore these bits must not be read speculatively.

Offset: Base + 0x0020

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TWRN_INT	RWRN_INT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BIT1_ERR	BIT0_ERR	ACK_ERR	CRC_ERR	FRM_ERR	STF_ERR	TX_WRN	RX_WRN	IDLE	TXRX	FLT_CONF	0	BOFF_INT	ERR_INT	0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-10. Error and Status Register (CANx_ESR)

Table 25-11. CANx_ESR Field Descriptions

Field	Description
bits 0–13	Reserved.
TWRNINT	If the WRNEN bit in CANx_MCR is asserted, the TWRNINT bit is set when the TXWRN flag transitions from 0 to 1, meaning that the Tx error counter reached 96. If the corresponding mask bit in the Control Register (TWRNMSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing to 1. Writing 0 has no effect. 0 No such occurrence 1 TXECTR ≥ 96
RWRNINT	If the WRNEN bit in CANx_MCR is asserted, the RWRNINT bit is set when the RXWRN flag transitions from 0 to 1, meaning that the Rx error counter reached 96. If the corresponding mask bit in the Control Register (RWRNMSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing to 1. Writing 0 has no effect. 0 No such occurrence 1 RXECTR ≥ 96
BIT1ERR	Bit 1 Error. Indicates when an inconsistency occurs between the transmitted and the received message in a bit. A read clears BIT1ERR. 0 No such occurrence 1 At least one bit sent as recessive is received as dominant Note: This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.
BIT0ERR	Bit 0 Error. Indicates when an inconsistency occurs between the transmitted and the received message in a bit. A read clears BIT0ERR. 0 No such occurrence 1 At least one bit sent as dominant is received as recessive
ACKERR	Acknowledge Error. Indicates that an acknowledge error has been detected by the transmitter node; that is, a dominant bit has not been detected during the ACK SLOT. A read clears ACKERR. 0 No such occurrence 1 An ACK error occurred since last read of this register
CRCERR	Cyclic Redundancy Code Error. Indicates that a CRC error has been detected by the receiver node; that is, the calculated CRC is different from the received. A read clears CRCERR. 0 No such occurrence 1 A CRC error occurred since last read of this register.
FRMERR	Form Error. Indicates that a form error has been detected by the receiver node; that is, a fixed-form bit field contains at least one illegal bit. A read clears FRMERR. 0 No such occurrence 1 A form error occurred since last read of this register
STFERR	Stuffing Error. Indicates that a stuffing error has been detected. A read clears STFERR. 0 No such occurrence. 1 A stuffing error occurred since last read of this register.
TXWRN	Tx Error Counter. This status bit indicates that repetitive errors are occurring during message transmission. 0 No such occurrence 1 TXECTR ≥ 96
RXWRN	Rx Error Counter. This status bit indicates when repetitive errors are occurring during messages reception. 0 No such occurrence 1 RXECTR ≥ 96

Table 25-11. CANx_ESR Field Descriptions (continued)

Field	Description
IDLE	CAN Bus IDLE State. This status bit indicates when CAN bus is in IDLE state. 0 No such occurrence 1 CAN bus is now IDLE
TXRX	Current FlexCAN Status (Transmitting/Receiving). This status bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 0 FlexCAN is receiving a message (IDLE = 0) 1 FlexCAN is transmitting a message (IDLE = 0)
FLTCONF	Fault Confinement State. This status bit indicates the confinement state of the FlexCAN module. If the LOM bit in the CANx_CTRL is asserted, the FLTCONF field will indicate "Error Passive". Since the CANx_CTRL is not affected by soft reset, the FLTCONF field will not be affected by soft reset if the LOM bit is asserted. 00 Error active 01 Error passive 1X Bus off
bit 28	Reserved.
BOFFINT	Bus Off Interrupt. This status bit is set when FlexCAN is in the bus off state. If CANx_CTRL[BOFFMSK] is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence 1 FlexCAN module is in 'Bus Off' state
ERRINT	Error Interrupt. This status bit indicates that at least one of the error bits (bits 16-21) is set. If CANx_CTRL[ERRMSK] is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence 1 Indicates setting of any error bit in the CANx_CANx_ESR
bit 31	Reserved.

25.3.4.7 Interrupt Masks 2 Register (CANx_IMASK2)

This register allows any number of a range of 32 Message Buffer Interrupts to be enabled or disabled. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (that is, when the corresponding CANx_IFLAG2 bit is set).

Offset: Base + 0x0024

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	63M	62M	61M	60M	59M	58M	57M	56M	55M	54M	53M	52M	51M	50M	49M	48M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	47M	46M	45M	44M	43M	42M	41M	40M	39M	38M	37M	36M	35M	34M	33M	32M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-11. Interrupt Masks 2 Register (CANx_IMASK2)

Table 25-12. CANx_IMASK2 Field Descriptions

Field	Description
BUF n M	<p>Message Buffer n Mask. Enables or disables the respective FlexCAN message buffer (MB63 to MB32) Interrupt.</p> <p>0 The corresponding buffer Interrupt is disabled 1 The corresponding buffer Interrupt is enabled</p> <p>Note: Setting or clearing a bit in the CANx_IMASK2 register can assert or negate an interrupt request, respectively.</p>

25.3.4.8 Interrupt Masks 1 Register (CANx_IMASK1)

This register allows to enable or disable any number of a range of 32 Message Buffer Interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e., when the corresponding CANx_IFLAG1 bit is set).

Offset: Base + 0x0028

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	31M	30M	29M	28M	27M	26M	25M	24M	23M	22M	21M	20M	19M	18M	17M	16M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	15M	14M	13M	12M	11M	10M	9M	8M	7M	6M	5M	4M	3M	2M	1M	0M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-12. Interrupt Masks 1 Register (CANx_IMASK1)

Table 25-13. CANx_IMASK1 Field Descriptions

Field	Description
BUF n M	<p>Message Buffer n Mask. Enables or disables the respective FlexCAN message buffer (MB31 to MB0) Interrupt.</p> <p>0 The corresponding buffer Interrupt is disabled 1 The corresponding buffer Interrupt is enabled</p> <p>Note: Setting or clearing a bit in the CANx_IMASK1 register can assert or negate an interrupt request, respectively.</p>

25.3.4.9 Interrupt Flags 2 Register (CANx_IFLAG2)

This register defines the flags for 32 Message Buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding CANx_IFLAG2 bit. If the corresponding

CANx_IMASK2 bit is set, an interrupt will be generated. The interrupt flag must be cleared by writing it to '1'. Writing '0' has no effect.

When the AEN bit in the CANx_MCR is set (abort enabled), while the CANx_IFLAG2 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

Offset: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	63I	62I	61I	60I	59I	58I	57I	56I	55I	54I	53I	52I	51I	50I	49I	48I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	47I	46I	45I	44I	43I	42I	41I	40I	39I	38I	37I	36I	35I	34I	33I	32I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-13. Interrupt Flag 2 Register (CANx_IFLAG2)

Table 25-14. CANx_IFLAG2 Field Descriptions

Field	Description
BUF <i>n</i>	Message Buffer <i>n</i> Interrupt. Each bit represents the respective FlexCAN message buffer (MB63–MB32) interrupt. Write 1 to clear. 0 No such occurrence 1 The corresponding buffer has successfully completed transmission or reception.

25.3.4.10 Interrupt Flags 1 Register (CANx_IFLAG1)

This register defines the flags for 32 message buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding CANx_IFLAG1 bit. If the corresponding CANx_IMASK1 bit is set, an interrupt will be generated. The interrupt flag must be cleared by writing it to '1'. Writing '0' has no effect.

When the AEN bit in the CANx_MCR is set (Abort enabled), while the CANx_IFLAG1 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

When the FEN bit in the CANx_MCR is set (FIFO enabled), the function of the eight least significant interrupt flags (BUF7I - BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Offset: Base + 0x0030

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	31I	30I	29I	28I	27I	26I	25I	24I	23I	22I	21I	20I	19I	18I	17I	16I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF	BUF
W	15I	14I	13I	12I	11I	10I	9I	8I	7I	6I	5I	4I	3I	2I	1I	0I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-14. Interrupt Flag 1 Register (CANx_IFLAG1)

Table 25-15. CANx_IFLAG1 Field Descriptions

Field	Description
BUF31I– BUF8I	Message Buffer <i>n</i> Interrupt. Each bit represents the respective FlexCAN message buffer (MB31 to MB8) interrupt. Write 1 to clear. 0 No such occurrence 1 The corresponding buffer has successfully completed transmission or reception.
BUF7I	Buffer MB7 Interrupt or “FIFO Overflow” If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full). 0 No such occurrence 1 MB7 completed transmission/reception or FIFO overflow
BUF6I	Buffer MB6 Interrupt or “FIFO Warning” If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 4 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 0 No such occurrence 1 MB6 completed transmission/reception or FIFO almost full
BUF5I	Buffer MB5 Interrupt or “Frames Available in FIFO” If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO. 0 No such occurrence 1 MB5 completed transmission/reception or frames available in the FIFO
BUF4I– BUF0I	Buffer MB <i>n</i> Interrupt or “Reserved” If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations. 0 No such occurrence 1 Corresponding MB completed transmission/reception

25.3.4.11 Rx Individual Mask Registers (CANx_RXIMR0–CANx_RXIMR63)

These registers are used as acceptance masks for ID filtering in Rx MBs and the FIFO. If the FIFO is not enabled, one mask register is provided for each available message buffer, providing ID masking capability

on a per message buffer basis. When the FIFO is enabled (FEN bit in CANx_MCR is set), the first eight mask registers apply to the eight elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

The individual Rx mask registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in freeze mode. Out of freeze mode, write accesses are blocked and read accesses will return “all zeros”. Furthermore, if the BCC bit in the register is negated, any read or write operation to these registers results in access error.

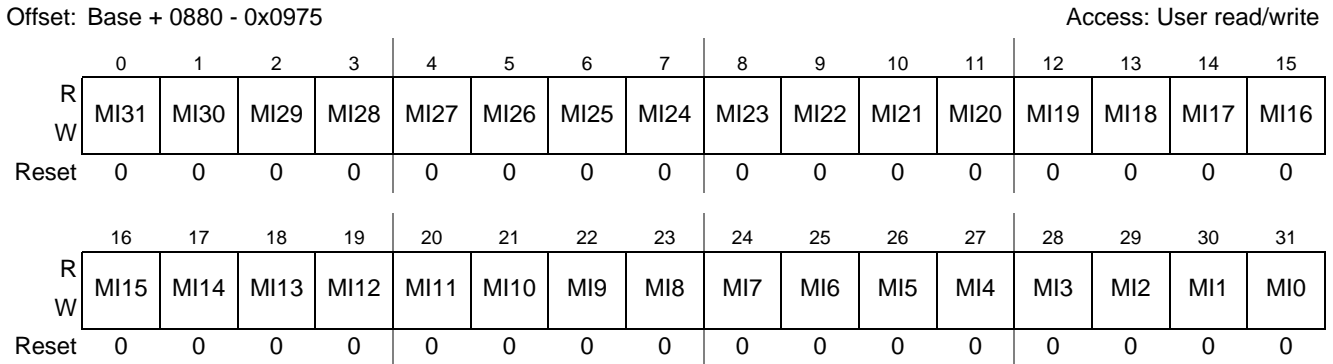


Figure 25-15. Rx Individual Mask Registers (CANx_RXIMR0-CANx_RXIMR63)

Table 25-16. CANx_RXIMR0-CANx_RXIMR63 Field Descriptions

Field	Description
MI31–M0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care” 1 The corresponding bit in the filter is checked against the one received

25.4 Functional Description

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of up to 64 message buffers (MB) that store configuration and control data, time stamp, message ID and data (see [Section 25.3.2, “Message Buffer Structure”](#)). The memory corresponding to the first eight MBs can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (up to eight extended IDs or 16 standard IDs or 32 8-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB ordering.

Before proceeding with the functional description, an important concept must be explained. A message buffer is said to be “active” at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a ‘0000’ code is inactive (refer to [Table 25-4](#)). Similarly, a Tx MB with a ‘1000’ or ‘1001’ code is also inactive (refer to [Table 25-5](#)). An MB not programmed with ‘0000’, ‘1000’ or ‘1001’ will be temporarily deactivated (will not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see [Section 25.4.5.2, “Message Buffer Deactivation”](#)).

25.4.1 Transmit Process

If the MB is active (transmission pending), write an ABORT code (‘1001’) to the code field of the control and status word to request an abortion of the transmission, then read back the code field and the IFLAG1/2 register to check if the transmission was aborted (see [Section 25.4.5.1, “Transmission Abort Mechanism”](#)). If backwards compatibility is desired (AEN in CANx_MCR negated), just write ‘1000’ to the Code field to inactivate the MB but then the pending frame may be transmitted without notification (see [Section 25.4.5.2, “Message Buffer Deactivation”](#)).

- Write the ID word.
- Write the data bytes.
- Write the length, control and code fields of the control and status word to activate the MB.

Once the MB is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the free-running timer is written into the time stamp field, the code field in the control and status word is updated, a status flag is set in the interrupt flag register and an interrupt is generated if allowed by the corresponding interrupt mask register bit. The new code field after transmission depends on the code that was used to activate the MB in step four (see [Table 25-4](#) and [Table 25-5](#) in [Section 25.3.2, “Message Buffer Structure”](#)). When the abort feature is enabled (AEN in CANx_MCR is asserted), after the Interrupt Flag is asserted for a MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to update it until the interrupt flag be negated by CPU. It means that the CPU must clear the corresponding CANx_IFLAG before starting to prepare this MB for a new transmission or reception.

25.4.2 Arbitration Process

This process selects which will be the next MB to be transmitted. All MBs programmed as transmit buffers will be scanned to find the lowest ID¹ or the lowest MB number or the highest priority, depending on the LBUF and LPRIO_EN bits on the control register. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished

1. If LBUF is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

- When MBM is in idle or bus off state and the CPU writes to the C/S word of any MB
- Upon leaving freeze mode

When LBUF is asserted, the LPRIO_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO_EN are both negated, the MB with the lowest ID is transmitted first but. If LBUF is negated and LPRIO_EN is asserted, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define which MB should be transmitted first, therefore MBs with PRIO = 000 have higher priority. If two or more MBs have the same priority, the regular ID will determine the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB will be transmitted first.

Once the highest priority MB is selected, it is transferred to a temporary storage space called serial message buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called “move-out” and after it is done, write access to the corresponding MB is blocked (if the AEN bit in CANx_MCR is asserted). The write access is released in the following events:

- After the MB is transmitted
- FlexCAN enters in HALT or BUS OFF
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits up to eight data bytes, even if the data length code (DLC) value is bigger.

25.4.3 Receive Process

The CPU prepares a message buffer for frame reception by executing the following steps:

- If the MB has a pending transmission, write an ABORT code ('1001') to the code field of the control and status word to request an abortion of the transmission, then read back the code field and the CANx_IFLAG1/2 register to check if the transmission was aborted (see [Section 25.4.5.1, “Transmission Abort Mechanism”](#)). If backwards compatibility is desired (AEN in CANx_MCR negated), just write '1000' to the code field to inactivate the MB, but then the pending frame may be transmitted without notification (see [Section 25.4.5.2, “Message Buffer Deactivation”](#)). If the MB already programmed as a receiver, just write '0000' to the code field of the control and status word to keep the MB inactive.
- Write the ID word.
- Write '0100' to the code field of the control and status word to activate the MB

After the MB is activated in the third step, it will be able to receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

- The value of the free-running timer is written into the time stamp field
- The received ID, data (8 bytes at most), and length fields are stored
- The code field in the control and status word is updated (see [Table 25-4](#) and [Table 25-5](#) in [Section 25.3.2, “Message Buffer Structure”](#))
- A status flag is set in the interrupt flag register and an interrupt is generated if allowed by the corresponding interrupt mask register bit

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

- Read the control and status word (mandatory – activates an internal lock for this buffer)
- Read the ID field (optional – needed only if a mask was used)
- Read the data field
- Read the free-running timer (optional – releases the internal lock)

Upon reading the control and status word, if the BUSY bit is set in the code field, then the CPU should defer the access to the MB until this bit is negated. Reading the free-running timer is not mandatory. If not executed, the MB remains locked, unless the CPU reads the C/S word of another MB. Only a single MB is locked at a time. The only mandatory CPU read operation is the one on the control and status word to assure data coherency (see [Section 25.4.5, “Data Coherence”](#)).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the CANx_IFLAG registers and not by the code field of that MB. Polling the code field does not work because after a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the code field will not return to EMPTY. It will remain FULL, as explained in [Table 25-4](#). If the CPU tries to workaroud this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary: *never do polling by reading directly the C/S word of the MBs. Instead, read the CANx_IFLAG registers.*

The received ID field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking. Note also that FlexCAN does receive frames transmitted by itself if there exists an Rx matching MB, provided the SRX_DIS bit in the CANx_MCR is not asserted. If SRX_DIS is asserted, FlexCAN will not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal will be generated due to the frame reception.

To be able to receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during freeze mode (see [Section 25.4.6, “Rx FIFO”](#)). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

- Read the control and status word (optional – needed only if a mask was used for IDE and RTR bits)
- Read the ID field (optional – needed only if a mask was used)
- Read the data field
- Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

25.4.4 Matching Process

The matching process is an algorithm executed by the MBM that scans the MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first and then, if a match is not found within the FIFO table, the other MBs are scanned. In the event that the FIFO is full, the matching algorithm will always look for a matching MB outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary MB called serial message buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular MBs, the contents of the SMB will be transferred to the FIFO or to the matched MB during the 6th bit of the end-of-frame field of the CAN protocol. This operation is called move-in. If any protocol error (CRC, ACK, etc.) is detected, then the move-in operation does not happen.

For the regular mailbox MBs, an MB is said to be free to receive a new frame if the following conditions are satisfied:

- The MB is not locked (see [Section 25.4.5.3, “Message Buffer Lock Mechanism”](#))
- The code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not free to receive the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it can not find one that is free, then it will overwrite the last matching MB (unless it is locked) and set the code field to OVERRUN (refer to [Table 25-4](#) and [Table 25-5](#)). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see [Section 25.4.5.3, “Message Buffer Lock Mechanism”](#)).

Suppose, for example, that the FIFO is disabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm will find the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm will find MB number 2 again, but it is not free to receive, so it will keep looking and find MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are free to receive, so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the MBs. By programming more than one MB with the same ID, received messages will be queued into the MBs. The CPU can examine the time stamp field of the MBs to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the BCC bit in CANx_MCR is negated, the matching algorithm stops at the first MB with a matching ID that it finds, whether this MB is free or not. As a result, the message queueing feature does not work if the BCC bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. Please refer to [Section 25.3.4.11, “Rx Individual Mask Registers \(CANx_RXIMR0–CANx_RXIMR63\)”](#). During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is “don’t care”. Please note that the individual mask registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the BCC bit is asserted and while the module is in freeze mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, CANx_RX14MASK, and CANx_RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the BCC bit in the CANx_MCR Register is negated.

25.4.5 Data Coherence

To maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in [Section 25.4.1, “Transmit Process,”](#) and [Section 25.4.3, “Receive Process.”](#) Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

25.4.5.1 Transmission Abort Mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. To maintain backwards compatibility, the abort mechanism must be explicitly enabled by asserting the AEN bit in the CANx_MCR.

In order to abort a transmission, the CPU must write a specific abort code (1001) to the code field of the control and status word. When the abort mechanism is enabled, the active MBs configured as transmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that was already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into freeze mode

If none of the conditions above are reached, the MB is transmitted correctly, the interrupt flag is set in the CANx_IFLAG register and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. In the other hand, if one of the above conditions is reached, the frame is not transmitted, therefore the abort code is written into the code field, the interrupt flag is set in the CANx_IFLAG and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked, therefore the MB is updated and no interrupt flag is set. In this way the CPU needs to read the abort code to make sure the active MB was deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the MB is deactivated and not aborted, because the transmission did not start yet. One MB is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

- CPU writes 1001 into the code field of the C/S word
- CPU reads the CODE field and compares it to the value that was written
- If the CODE field that was read is different from the value that was written, the CPU must read the corresponding CANx_IFLAG to check if the frame was transmitted or it is being currently transmitted. If the corresponding CANx_IFLAG is set, the frame was transmitted. If the corresponding CANx_IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the MB was aborted (CODE=1001) or it was transmitted (CODE=1000).

25.4.5.2 Message Buffer Deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the control and status word of active MBs out of freeze mode. Any CPU write access to the control and status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the control and status word of active MBs when not in freeze mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was free to receive.
- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the code field is not updated. In order to avoid this situation, the abort procedures described in [Section 25.4.5.1, “Transmission Abort Mechanism,”](#) should be used.

25.4.5.3 Message Buffer Lock Mechanism

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an “active not empty” Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and thus it sets an internal lock flag for that MB. The lock is released when the CPU reads the free-running timer (global unlock operation), or when it reads the Control and Status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

NOTE

The locking mechanism only applies to Rx MBs which have a code different than INACTIVE (‘0000’) or EMPTY¹ (‘0100’). Also, Tx MBs can not be locked.

¹.In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. This behavior will be honoured when the BCC bit is negated.

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the control and status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no free to receive MBs, so it decides to override MB number 5. However, this MB is locked, so the new message can not be written there. It will remain in the SMB waiting for the MB to be unlocked, and only then will be written to the MB. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there will be no indication of lost messages either in the code field of the MB or in the error and status register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the code field is asserted. If the CPU reads the control and status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

NOTE

If the BUSY bit is asserted or if the MB is empty, then reading the control and status word does not lock the MB.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the MB.

25.4.6 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the CANx_MCR. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80-0xFF) is now reserved for use of the FIFO engine (see [Section 25.3.3, “Rx FIFO Structure”](#)). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a Message Buffer structure at the beginning of the memory.

The FIFO can store up to six frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80 with the next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. A warning interrupt is also generated when 4 frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of 8

32-bit registers that can be configured to one of the following formats (see also [Section 25.3.3, “Rx FIFO Structure”](#)):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

NOTE

A chosen format is applied to all 8 registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight individual mask registers (CANx_RXIMR0 - CANx_RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIM8, continue to affect the regular MBs, starting from MB8. If the BCC bit is negated, then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by CANx_RX14MASK, element 7 is affected by CANx_RX15MASK and the other elements (0 to 5) are affected by CANx_RXGMASK.

25.4.7 CAN Protocol Related Features

25.4.7.1 Remote Frames

A remote frame is a special kind of frame. The user can program a MB to be a request remote frame by writing the MB as transmit with the RTR bit set to 1. After the remote request frame is transmitted successfully, the MB becomes a receive message buffer, with the same ID as before.

When a remote request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the CODE field ‘1010’. If there is a matching ID, then this MB frame will be transmitted. Note that if the matching MB has the RTR bit set, then FlexCAN will transmit a remote frame as a response.

A received remote request frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a remote request frame was received and matched a MB, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

If the Rx FIFO is enabled (bit FEN set in CANx_MCR), FlexCAN will not generate an automatic response for remote request frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it will be stored in the FIFO and presented to the CPU. Note that for filtering formats A and B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID).

25.4.7.2 Overload Frames

FlexCAN will transmit overload frames due to detection of these conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of intermission
- Detection of a dominant bit at the 7th bit (last) of end of frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of error frame delimiter or overload frame delimiter

25.4.7.3 Time Stamp

The value of the free-running timer is sampled at the beginning of the identifier field on the CAN bus, and is stored at the end of move in the TIME STAMP field, providing network behavior with respect to time.

Note that the free-running timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in [Section 25.3.4.2, “Control Register \(CANx_CTRL\).”](#)

25.4.7.4 Protocol Timing

The clock source to the CAN protocol interface (CPI) can be either the system clock or a direct feed from the oscillator pin EXTAL. The clock source is selected by the CLK_SRC bit in the CANx_CTRL. The clock is fed to the prescaler to generate the serial clock (SCK).

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The CANx_CTRL has various fields used to control bit timing parameters: PRES DIV, PROPSEG, PSEG1, PSEG2 and RJW. See [Section 25.3.4.2, “Control Register \(CANx_CTRL\).”](#)

The PRES DIV field controls a prescaler that generates the serial clock (SCK), whose period defines the ‘time quantum’ used to compose the CAN waveform. A time quantum is the atomic unit of time handled by FlexCAN.

$$f_{Tq} = \frac{f_{CANCLK}}{\text{Prescaler Value}}$$

A bit time is subdivided into three segments¹ (reference [Figure 25-16](#) and [Table 25-17](#)):

- SYNCSEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time segment 1: This segment includes the propagation segment and the phase segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CANx_CTRL register so that their sum (plus 2) is in the range of 4 to 16 time quanta.
- Time segment 2: This segment represents the phase segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CANx_CTRL register (plus 1) to be 2 to 8 time quanta long.

$$\text{Bit Rate} = \frac{f_{Tq}}{(\text{Number of Time Quanta})}$$

1. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

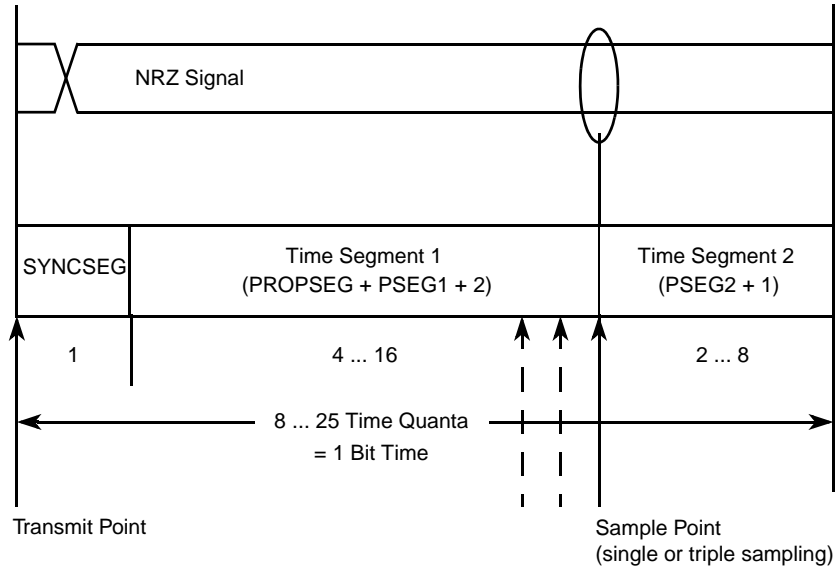


Figure 25-16. Segments within the Bit Time

Table 25-17. Time Segment Syntax

Syntax	Description
SYNCSEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node in receive mode samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 25-18 gives an overview of the CAN compliant segment settings and the related parameter values.

NOTE

It is the user’s responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an information processing time (IPT) of 2, which is the value implemented in the FlexCAN module.

Table 25-18. CAN Standard Compliant Bit Time Segment Settings

Time Segment 1	Time Segment 2	Resynchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4

Table 25-18. CAN Standard Compliant Bit Time Segment Settings (continued)

Time Segment 1	Time Segment 2	Resynchronization Jump Width
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

25.4.7.5 Arbitration and Matching Timing

During normal transmission or reception of frames, the arbitration, match, move in and move out processes are executed during certain time windows inside the CAN frame, as shown in Figure 25-17. When doing matching and arbitration, FlexCAN needs to scan the whole message buffer memory during the available time slot. In order to have sufficient time to do that, the following restrictions must be observed:

- A valid CAN bit timing must be programmed, as indicated in Figure 25-17.
- The system clock frequency cannot be smaller than the oscillator clock frequency, i.e. the PLL cannot be programmed to divide down the oscillator clock.
- There must be a minimum ratio of 16 between the system clock frequency and the CAN bit rate.

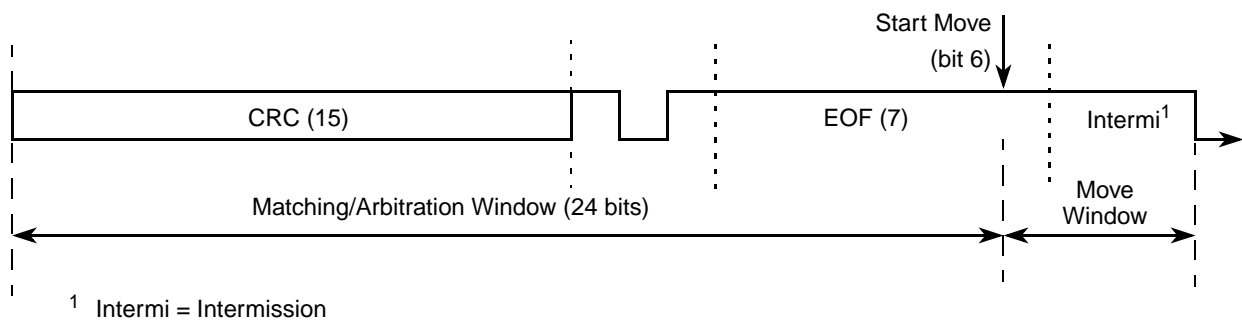


Figure 25-17. Arbitration, Match and Move Time Windows

25.4.8 Modes of Operation Details

25.4.8.1 Freeze Mode

This mode is entered by asserting the HALT bit in the CAN_x_MCR or when the MCU is put into debug mode. In both cases it is also necessary that the FRZ bit is asserted in the CAN_x_MCR. When freeze mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either intermission, passive error, bus off or idle state
- Waits for all internal activities like move in or move out to finish
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the CAN_x_ECR, which is read-only in other modes
- Sets the NOTRDY and FRZACK bits in CAN_x_MCR

After requesting freeze mode, the user must wait for the FRZACK bit to be asserted in CAN_x_MCR before executing any other action, otherwise FlexCAN can operate in an unpredictable way. In freeze mode, all memory mapped registers are accessible.

Exiting freeze mode is done in one of these ways:

- CPU negates the FRZ bit in the CAN_x_MCR
- The MCU exits debug mode and/or the HALT bit is negated

After it is out of freeze mode, FlexCAN tries to resynchronize to the CAN bus by waiting for 11 consecutive recessive bits.

25.4.8.2 Module Disabled Mode

This low-power mode is entered when the CAN_x_MCR[MDIS] bit is asserted. If the module is disabled during freeze mode, it shuts down the clocks to the CPI and MBM submodules, sets the CAN_x_MCR[MDISACK] bit and negates the CAN_x_MCR[FRZACK] bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either idle or bus off state, or else waits for the third bit of intermission and then checks it to be recessive
- Waits for all internal activities like move in or move out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM submodules
- Sets the NOTRDY and MDISACK bits in CAN_x_MCR

The bus interface unit continues to operate, enabling the CPU to access memory mapped registers except the free-running timer, the CAN_x_ECR and the message buffers, which cannot be accessed when the module is disabled. Exiting from this mode is done by negating the CAN_x_MCR[MDIS] bit, which will resume the clocks and negate the CAN_x_MCR[MDISACK] bit.

25.4.8.3 Stop Mode

This is a system low-power mode in which all MCU clocks are stopped for maximum power savings. If FlexCAN receives the global stop mode request during freeze mode, it sets the LPM_ACK bit, negates the FRZ_ACK bit and then sends a stop acknowledge signal to the CPU, in order to shut down the clocks globally. If stop mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in idle or bus off state, or waits for the third bit of intermission and checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in, and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Sets the NOT_RDY and LPM_ACK bits in CAN_x_MCR
- Sends a stop acknowledge signal to the CPU, so that it can shut down the clocks globally

Exiting stop mode is done by the CPU resuming the clocks and removing the stop mode request.

25.4.9 Interrupts

The FlexCAN module interrupts are ORed together at the chip level as described in [Table 8-2](#) in [Chapter 8](#), “[Interrupts](#).”

There is an interrupt source for each MB from MB0 to MB15. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the CAN_x_IFLAG2 or CAN_x_IFLAG1 registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to 1.

A combined interrupt for each of two MB groups, MB16–MB31 and MB32–MB63, is also generated by an OR of all the interrupt sources from the associated MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case the CPU must read the CAN_x_IFLAG2 and CAN_x_IFLAG1 registers to determine which MB caused the interrupt.

The other two interrupt sources (bus off/transmit warning/receive warning and error) generate interrupts like the MB interrupt sources, and can be read from CAN_x_ESR. The bus off/transmit warning/receive warning and error interrupt mask bits are located in the CAN_x_CTRL.

25.4.10 Bus Interface

The CPU access to FlexCAN registers are subject to the following rules:

- Read and write access to unimplemented or reserved address space results in access error. Any access to unimplemented MB Rx individual mask register locations results in access error. Any access to the Rx individual mask register space when the BCC bit in CAN_x_MCR is negated results in access error.
- For a FlexCAN configuration that uses less than the total number of MBs and MAXMB is set accordingly, the remaining MB and Rx mask register spaces can be used as general-purpose RAM space. Note that the Rx individual mask registers can only be accessed in freeze mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 64 MBs and MAXMB is programmed with zero. The maximum number of MBs in this case becomes one. The MB memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one MB. This leaves us with the available space from 0x0090 to 0x047F. The available memory in the mask registers space would be from 0x0884 to 0x097F. Byte, word, and long word accesses are allowed to the unused MB space.

NOTE

Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

25.5 Initialization and Application Information

This section provides instructions for initializing the FlexCAN module.

25.5.1 FlexCAN Initialization Sequence

The FlexCAN module can be reset in three ways:

- MCU-level hard reset, which resets all memory-mapped registers asynchronously
- MCU-level soft reset, which resets some of the memory-mapped registers synchronously (refer to [Table 25-7](#) to see what registers are affected by soft reset)
- SOFTRST bit in CAN_x_MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFTRST bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed.

After the module is enabled (CAN_x_MCR[MDIS] bit negated), FlexCAN must be put into freeze mode before doing any configuration. In freeze mode, FlexCAN is un-synchronized to the CAN bus, the HALT and FRZ bits in CAN_x_MCR are set, the internal state machines are disabled and the FRZACK and NOTRDY bits in the CAN_x_MCR are set. The CNTX pin is in recessive state and FlexCAN does not initiate frame transmission nor receives any frames from the CAN bus. Note that the message buffer contents are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization, it is required that FlexCAN is put into freeze mode (see [Section 25.4.8.1, “Freeze Mode](#)). The following is a generic initialization sequence applicable for the FlexCAN module:

- Initialize the CAN_x_MCR
 - Enable the individual filtering per MB and reception queue features by setting the BCC bit
 - Enable the warning interrupts by setting the WRN_EN bit
 - If required, disable frame self reception by setting the SRX_DIS bit
 - Enable the FIFO by setting the FEN bit
 - Enable the abort mechanism by setting the AEN bit
 - Enable the local priority feature by setting the LPRIO_EN bit
- Initialize CAN_x_CTRL.
 - Determine bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW.
 - Determine the bit rate by programming the PRES DIV field.
 - Determine internal arbitration mode (LBUF bit).
- Initialize message buffers.
 - The control and status word of all message buffers must be initialized
 - If FIFO was enabled, the 8-entry ID table must be initialized
 - Other entries in each message buffer should be initialized as required
- Initialize the Rx individual mask registers
- Set required interrupt mask bits in the IMASK registers (for all MB interrupts), in CAN_x_CTRL (for bus off and error interrupts) and in CAN_x_MCR for wake-up interrupt
- Negate the HALT bit in CAN_x_MCR

Starting with this last event, FlexCAN attempts to synchronize with the CAN bus.

Chapter 26

Enhanced Modular I/O Subsystem (eMIOS200)

26.1 Introduction

The eMIOS200 provides functionality to generate or measure time events. The eMIOS200 uses timer channels that are a reduced version of the unified channel (UC) module used on MPC555x devices. Each channel provides a subset of the functionality available in the unified channel, at a resolution of 16 bits, and provides a consistent user interface with previous eMIOS implementations.

26.1.1 Block Diagram

[Figure 26-1](#) shows the eMIOS200 block diagram.

Channels 0 through 15 use channel type 1, channels 16 through 22 use channel type 2, and channel 23 uses channel type 3 (see [Section 26.1.4](#), “Channel Types”).

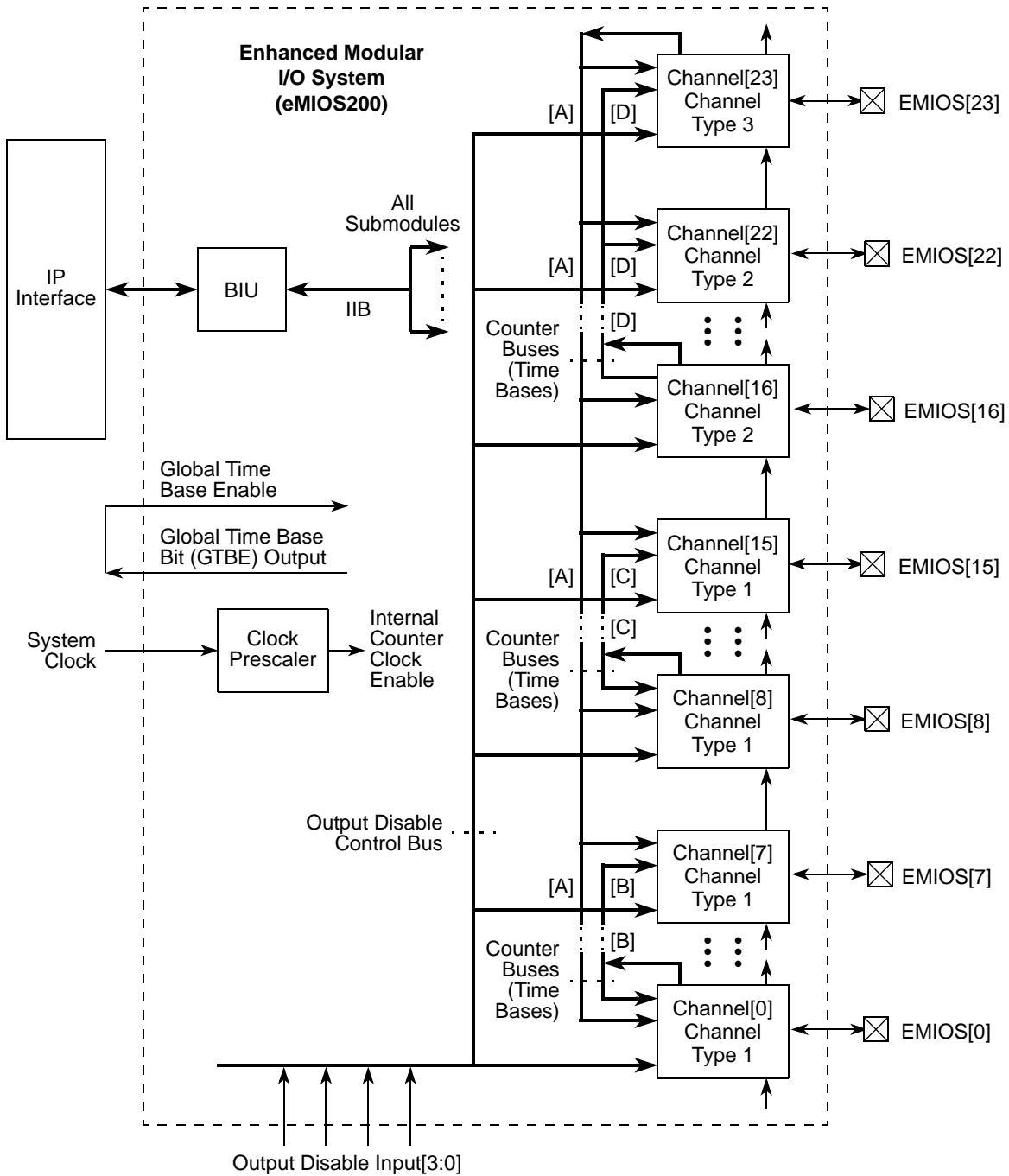


Figure 26-1. eMIOS200 Block Diagram

26.1.2 Features

- 24 channels implemented using three channel types.
- Channels features:
 - 16-bit registers for captured/match values

- 16-bit internal counter
- Internal prescaler
- Selectable time base
- Can generate its own time base
- Four 16-bit-wide counter buses
 - Counter bus A can be driven by unified channel 23
 - Counter bus B, C, and D are driven by unified channels 0, 8, and 16, respectively
 - Counter bus A can be shared among all unified channels. UCs 0 to 7, 8 to 15, and 16 to 23 can share counter buses B, C, and D, respectively
- One global prescaler
- The output signal from the module configuration register's global time base enable bit (EMIOS_MCR[GTBE]) is wrapped back into the global timebase enable input so that the timebase of each channel can be started simultaneously.
- Shared time bases through the counter buses
- Shadow FLAG register
- State of eMIOS200 can be frozen for debug purposes
- Debug and stop modes are supported.

26.1.3 Modes of Operation

There are four main operating modes of eMIOS200: run mode, module disable mode, debug mode, and stop mode. These modes are briefly described in this section.

Run mode is the normal operation mode and is described in [Section 26.5, “Functional Description.”](#)

Module disable mode is used for MCU power management. The clock to the non-memory-mapped logic in the eMIOS200 is stopped while in module disable mode. Module disable mode is entered when MDIS=1 in the EMIOS_MCR. Individual disabling of the channels is not supported.

Debug mode is individually programmed for each channel. When entering this mode, the unified channel registers' contents are frozen, but remain available for read and write access through the IP interface.

Stop mode is also used for MCU power management. In stop mode, eMIOS=1 in SIU_HLT register and all clocks to the eMIOS200 module are disabled.

26.1.4 Channel Types

The 24 16-bit timer channels available on the MPC5510 are implemented using three different channel configurations. The available modes of operation for each channel type are listed in [Table 26-1](#).

Table 26-1. Unified Channel (UC) Modes

Acronym	Mode (Section/Page)	Channels		
		0–15 Type 1 (PWM)	16–22 Type 2 (Input Capture/ Output Compare)	23 Type 3 (Counter)
GPIO	General purpose input/output (26.5.1.1.1/26-18)	✓	✓	✓
SAIC	Single action input capture (26.5.1.1.2/26-19)	✓	✓	✓
SAOC	Single action output compare (26.5.1.1.3/26-19)	✓	✓	✓
IPWM	Input pulse width measurement (26.5.1.1.4/26-20)	✓		
IPM	Input period measurement (26.5.1.1.5/26-22)	✓		
DAOC	Double action output compare (26.5.1.1.6/26-24)	✓		
MCB	Modulus counter buffered (26.5.1.1.7/26-25)	✓		✓
OPWFMB	Output pulse width and frequency modulation buffered (26.5.1.1.8/26-28)	✓		
OPWMCB	Center aligned output pulse width modulation with dead time insertion buffered (26.5.1.1.9/26-33)	✓		
OPWMB	Output pulse width modulation buffered (26.5.1.1.10/26-38)	✓		

26.2 External Signal Description

Refer to [Table 2-1](#) and [Section 2.7](#), “Detailed External Signal Descriptions,” for detailed signal descriptions.

Each channel has one external signal, eMIOS[*n*]. Through the pad configuration register (SIU_PCR[PA]), you can choose to have a pin’s function be the eMIOS channel in either or both places as described in [Table 26-5](#).

The output disable input [3:0] is provided to implement the output disable feature. They are connected to emios_flag_out signals according to [Section 26.2.2](#), “Output Disable Input — eMIOS200 Output Disable Input Signal.”

26.2.1 eMIOS[*n*]

eMIOS[*n*] are the eMIOS channel pins. When used as input, an eMIOS[*n*] signal is available to be read by the MCU through the EMIOS_CSR[*n*UCIN]. When used as output, eMIOS[*n*] signal is configured in the unified channel status and control register (EMIOS_CSR[*n*]).

NOTE

All eMIOS channels support both input and output functions. When the eMIOS function is the primary function of a pin, then both the input and output functions are supported. When the eMIOS function is not the primary function of the pin, then only the output functions are supported.

26.2.2 Output Disable Input — eMIOS200 Output Disable Input Signal

Output disable inputs are connected as defined in [Table 26-2](#).

Table 26-2. ODIS Input Signals

eMIOS200 channel	Output Disable Input Signal
emios_flag_out[17]	Output disable input[3]
emios_flag_out[16]	Output disable input[2]
emios_flag_out[9]	Output disable input[1]
emios_flag_out[8]	Output disable input[0]

26.3 Memory Map and Registers

This section provides a detailed description of all eMIOS200 registers.

26.3.1 Module Memory Map

The eMIOS200 memory map is shown in [Table 26-3](#). The address of each register is given as an offset to the eMIOS200 base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 26-3. eMIOS200 Memory Map

Offset from EMIOS_BASE (0xFFFE_4000)	Register	Access ¹	Reset Value	Section/Page
Global Registers				
0x0000	EMIOS_MCR — Module Configuration Register	R/W	0x0000_0000	26.4.1/26-6
0x0004	EMIOS_GFR — Global FLAG Register	R	0x0000_0000	26.4.2/26-8
0x0008	EMIOS_OUDR — Output Update Disable Register	R/W	0x0000_0000	26.4.3/26-8
0x000C	EMIOSUCDIS — Stop (Disable) Channel Register	R/W	0x0000_0000	26.4.4/26-9
0x0010–0x001F	Reserved			
Unified Channel 0 Registers				
0x0020	EMIOS_CADR[0] — Channel A Data Register	R/W	0x0000_0000	26.4.5/26-9
0x0024	EMIOS_CBDR[0] — Channel B Data Register	R/W	0x0000_0000	26.4.6/26-10
0x0028	EMIOS_CCNTR[0] — Channel Counter Register	R	0x0000_0000	26.4.7/26-11
0x002C	EMIOS_CCR[0] — Channel Control Register	R/W	0x0000_0000	26.4.8/26-11
0x0030	EMIOS_CSR[0] — Channel Status Register	R	0x0000_0000	26.4.9/26-16
0x0038–0x003F	Reserved			
Unified Channel 1 Registers				

Table 26-3. eMIOS200 Memory Map (continued)

Offset from EMIOS_BASE (0xFFFE_4000)	Register	Access ¹	Reset Value	Section/Page
0x0040	EMIOS_CADR[1] — A Register	R/W	0x0000_0000	26.4.5/26-9
0x0044	EMIOS_CBDR[1] — B Register	R/W	0x0000_0000	26.4.6/26-10
0x0048	EMIOS_CCNTR[1] — Counter Register	R	0x0000_0000	26.4.7/26-11
0x004C	EMIOS_CCR[1] — Control Register	R/W	0x0000_0000	26.4.8/26-11
0x0050	EMIOS_CSR[1] — Status Register	R	0x0000_0000	26.4.9/26-16
0x0058–0x005F	Reserved			
Unified Channel 2–23 Registers				
0x0060–0x0031F	Same as Channel 0 and Channel 1 Registers (e.g. EMIOS_CADR[2], EMIOS_CBDR[2], etc)	—	—	—

¹ Note that R/W registers may contain some read-only or write-only bits.

26.4 Register Descriptions

This section lists the eMIOS200 registers in address order and describes the registers and their bit fields.

26.4.1 eMIOS200 Module Configuration Register (EMIOS_MCR)

The EMIOS_MCR contains global control bits for the eMIOS200 block.

Offset: EMIOS_BASE + 0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R				GTBE	0	GPREN	0	0	0	0	0	0	0	0	0	0
W		MDIS	FRZ													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPRES[0:7]								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-2. eMIOS200 Module Configuration Register (EMIOS_MCR)

Table 26-4. EMIOS_MCR Field Descriptions

Field	Description																				
bit 0	Reserved. Note: Writing to this bit will update the register value, and reading it will return the last value written, but the bit has no other effect.																				
MDIS	Module Disable Bit. Puts the eMIOS200 in low-power mode. The MDIS bit is used to stop the clock of the block, except the access to registers EMIOS_MCR, EMIOS_OUDR, and EMIOSUCDIS. 0 Clock is running 1 Enter low-power mode																				
FRZ	Freeze Bit. Enable the eMIOS200 to freeze the registers of the unified channels when debug mode is requested at MCU level. Each unified channel must have FREN bit set in order to enter freeze mode. While in freeze mode, the eMIOS200 continues to operate to allow the MCU access to the unified channel registers. The unified channel will remain frozen until the FRZ bit is written to 0 or the MCU exits debug mode or the unified channel FREN bit is cleared. 0 Exit freeze mode 1 Stops unified channel operation when in debug mode and the FREN bit is set in the EMIOS_CCR[n] register																				
GTBE	Global Time Base Enable Bit. The GTBE bit is used to export a global time base enable from the module and provide a method to start time bases of several blocks simultaneously. 0 Global time base enable out signal negated 1 Global time base enable out signal asserted Note: The global time base enable input pin controls the internal counters. When asserted, internal counters are enabled. When negated, internal counters are disabled.																				
bit 4	Reserved.																				
GPREN	Global Prescaler Enable Bit. The GPREN bit enables the prescaler counter. 0 Prescaler disabled (no clock) and prescaler counter is cleared 1 Prescaler enabled																				
bits 6–15	Reserved.																				
GPRE	Global Prescaler Bits. The GPRE bits select the clock divider value for the global prescaler. <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>GPRE</th> <th>Divide Ratio</th> </tr> </thead> <tbody> <tr> <td>00000000</td> <td>1</td> </tr> <tr> <td>00000001</td> <td>2</td> </tr> <tr> <td>00000010</td> <td>3</td> </tr> <tr> <td>00000011</td> <td>4</td> </tr> <tr> <td>.</td> <td>.</td> </tr> <tr> <td>.</td> <td>.</td> </tr> <tr> <td>.</td> <td>.</td> </tr> <tr> <td>11111110</td> <td>255</td> </tr> <tr> <td>11111111</td> <td>256</td> </tr> </tbody> </table>	GPRE	Divide Ratio	00000000	1	00000001	2	00000010	3	00000011	4	11111110	255	11111111	256
GPRE	Divide Ratio																				
00000000	1																				
00000001	2																				
00000010	3																				
00000011	4																				
.	.																				
.	.																				
.	.																				
11111110	255																				
11111111	256																				

26.4.2 eMIOS200 Global FLAG Register (EMIOS_GFR)

Offset: EMIOS_BASE + 0x0004

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	F23	F22	F21	F20	F19	F18	F17	F16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-3. eMIOS200 Global FLAG Register (EMIOS_GFR)

Table 26-5. EMIOS_GFR Field Descriptions

Field	Description
8–31 F[23:0]	FLAG Bits 23–0. The EMIOS_GFR is a read-only register that groups the FLAG bits from all channels. This organization improves interrupt handling on simpler devices. These bits are mirrors of the FLAG bits of each channel register (EMIOS_CSR).

26.4.3 eMIOS200 Output Update Disable (EMIOS_OUDR)

Offset: EMIOS_BASE + 0x0008

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	OU23	OU22	OU21	OU20	OU19	OU18	OU17	OU16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OU15	OU14	OU13	OU12	OU11	OU10	OU9	OU8	OU7	OU6	OU5	OU4	OU3	OU2	OU1	OU0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-4. eMIOS200 Output Update Disable Register (EMIOS_OUDR)

Table 26-6. EMIOS_OUDR Field Descriptions

Field	Description
OU[23:0]	Channel [n] Output Update Disable Bits. When running MCB mode or an output mode, values are written to registers A2 and B2. OU[n] bits are used to disable transfers from registers A2 to A1 and B2 to B1. Each bit controls one channel. 0 Transfer enabled. Depending on the operation mode, transfer may occur immediately or in the next period. Unless stated otherwise, transfer occurs immediately. 1 Transfers disabled

26.4.4 eMIOS200 Disable Channel (EMIOSUCDIS)

Offset: EMIOS_BASE + 0x000C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	UCDIS[23:16]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	UCDIS[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-5. eMIOS200 Enable Channel Register (EMIOSUCDIS)

Table 26-7. EMIOSUCDIS Field Descriptions

Field	Description
UCDIS[23:0]	Enable Channel [n] Bit. The UCDIS[n] bit is used to disable each of the unified channels by stopping its respective clock. 0 UC [n] enabled 1 UC [n] disabled

26.4.5 eMIOS200 A Register (EMIOS_CADR[n])

Offset: UC[n] base address + 0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	A															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-6. eMIOS200 A Register (EMIOS_CADR[n])

Depending on the mode of operation, internal registers A1 or A2, used for matches and captures, can be assigned to address EMIOS_CADR[n]. A1 and A2 are cleared by reset. [Table 26-8](#) summarizes the EMIOS_CADR[n] writing and reading accesses for all operation modes. For more information see [Section 26.5.1.1, “Unified Channel Modes of Operation.”](#)

26.4.6 eMIOS200 B Register (EMIOS_CBDR[n])

Offset: UC[n] base address + 0x0004

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	B															
W	B															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-7. eMIOS200 B Register (EMIOS_CBDR[n])

Depending on the mode of operation, internal registers B1 or B2 can be assigned to address EMIOS_CBDR[n]. Both B1 and B2 are cleared by reset. Table 26-8 summarizes the EMIOS_CBDR[n] writing and reading accesses for all operation modes. For more information see section Section 26.5.1.1, “Unified Channel Modes of Operation.”

Depending on the channel configuration, it may have EMIOS_CBDR register or not. This means that if at least one mode that requires the register is implemented, then the register is present, otherwise it is absent.

Table 26-8. EMIOS_CADR[n] and EMIOS_CBDR[n] Values Assignment

Operation Mode	Register Access				
	Write	Read	Write	Read	Alternate Read
GPIO	A1, A2	A1	B1, B2	B1	—
SAIC ¹	—	A2	B2	B2	—
SAOC ¹	A2	A1	B2	B2	—
IPWM	—	A2	—	B1	—
IPM	—	A2	—	B1	—
DAOC	A2	A1	B2	B1	—
MCB ¹	A2	A1	B2	B2	—
OPWFMB	A2	A1	B2	B1	—
OPWMCB	A2	A1	B2	B1	—
OPWMB	A2	A1	B2	B1	—

¹ In these modes, the register EMIOS_CBDR[n] is not used, but B2 can be accessed.

26.4.7 eMIOS200 Counter Register (EMIOS_CCNTR[n])

Offset: UC[n] base address + 0x0008

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	C															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

¹ In GPIO mode or freeze action, this register is writable.

Figure 26-8. eMIOS200 Counter Register (EMIOS_CCNTR[n])

The EMIOS_CCNTR[n] register contains the value of the internal counter. When GPIO mode is selected or the channel is frozen, the EMIOS_CCNTR[n] register is read/write. For all other modes, the EMIOS_CCNTR[n] is a read-only register. When entering some operation modes, this register is automatically cleared (refer to [Section 26.5.1.1, “Unified Channel Modes of Operation,”](#) for details).

Depending on the channel configuration it may have an internal counter or not. It means that if at least one mode that requires the counter is implemented, then the counter is present, otherwise it is absent.

26.4.8 eMIOS200 Control Register (EMIOS_CCR[n])

Offset: UC[n] base address + 0x000C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FREN	ODIS	ODISSL		UCPRE		UCPREN	DMA	0	IF			FCK	FEN	0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	BSL		EDSEL	EDPOL	MODE[0:6]						
W			FORCMA	FORCMB												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-9. eMIOS200 Control Register (EMIOS_CCR[n])

The control register gathers bits reflecting the status of the unified channel input/output signals and the overflow condition of the internal counter, as well as several read/write control bits.

Table 26-9. EMIOS_CCR[n] Field Descriptions

Field	Description										
FREN	Freeze Enable Bit. The FREN bit, if set and validated by FRZ bit in EMIOS_MCR register, freezes all registers' values when in debug mode, allowing the MCU to perform debug functions. 0 Normal operation 1 Freeze unified channel registers' values										
ODIS	Output Disable Bit. The ODIS bit allows disabling the output pin when running any of the output modes with the exception of GPIO mode. 0 The output pin operates normally 1 If the selected output disable input signal is asserted, the output pin goes to EDPOL for OPWFMB and OPWMB modes and to the complement of EDPOL for other modes, but the unified channel continues to operate normally, i.e., it continues to produce FLAG and matches. When the selected output disable input signal is negated, the output pin operates normally										
ODISSL	Output Disable Select Bits. The ODISSL bits select one of the four output disable input signals. <table border="1" data-bbox="550 682 1255 932"> <thead> <tr> <th>ODISSL</th> <th>Input Signal</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>Output disable input 0</td> </tr> <tr> <td>01</td> <td>Output disable input 1</td> </tr> <tr> <td>10</td> <td>Output disable input 2</td> </tr> <tr> <td>11</td> <td>Output disable input 3</td> </tr> </tbody> </table>	ODISSL	Input Signal	00	Output disable input 0	01	Output disable input 1	10	Output disable input 2	11	Output disable input 3
ODISSL	Input Signal										
00	Output disable input 0										
01	Output disable input 1										
10	Output disable input 2										
11	Output disable input 3										
UCPRE	Prescaler Bits. The UCPRE bits select the clock divider value for the internal prescaler of unified channel. <table border="1" data-bbox="570 1022 1239 1272"> <thead> <tr> <th>UCPRE</th> <th>Divide Ratio</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>1</td> </tr> <tr> <td>01</td> <td>2</td> </tr> <tr> <td>10</td> <td>3</td> </tr> <tr> <td>11</td> <td>4</td> </tr> </tbody> </table>	UCPRE	Divide Ratio	00	1	01	2	10	3	11	4
UCPRE	Divide Ratio										
00	1										
01	2										
10	3										
11	4										
UCPREN	Prescaler Enable Bit. The UCPREN bit enables the prescaler counter. 0 Prescaler disabled (no clock) and prescaler counter is loaded with UCPRE value 1 Prescaler enabled										
DMA	Direct Memory Access Bit. The DMA bit selects if the FLAG generation will be used as an interrupt or as a DMA request. 0 FLAG assigned to interrupt request 1 FLAG assigned to DMA request										

Table 26-9. EMIOS_CCR[n] Field Descriptions (continued)

Field	Description														
IF	<p>Input Filter Bits. The IF bits control the programmable input filter, selecting the minimum input pulse width that can pass through the filter. For output modes, these bits have no meaning.</p> <table border="1"> <thead> <tr> <th>IF¹</th> <th>Minimum Input Pulse Width [FLT_CLK Periods]</th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>Bypassed²</td> </tr> <tr> <td>0001</td> <td>02</td> </tr> <tr> <td>0010</td> <td>04</td> </tr> <tr> <td>0100</td> <td>08</td> </tr> <tr> <td>1000</td> <td>16</td> </tr> <tr> <td>All others</td> <td>Reserved</td> </tr> </tbody> </table> <p>¹ Filter latency is three clock edges. ² The input signal is synchronized before arriving to the digital filter.</p>	IF ¹	Minimum Input Pulse Width [FLT_CLK Periods]	0000	Bypassed ²	0001	02	0010	04	0100	08	1000	16	All others	Reserved
IF ¹	Minimum Input Pulse Width [FLT_CLK Periods]														
0000	Bypassed ²														
0001	02														
0010	04														
0100	08														
1000	16														
All others	Reserved														
FCK	<p>Filter Clock Select Bit. The FCK bit selects the clock source for the programmable input filter.</p> <p>0 Prescaled clock 1 Main clock</p>														
FEN	<p>FLAG Enable Bit. The FEN bit allows the unified channel FLAG bit to generate an interrupt signal or a DMA request signal (the type of signal to be generated is defined by the DMA bit).</p> <p>0 Disable (FLAG does not generate an interrupt or DMA request) 1 Enable (FLAG will generate an interrupt or DMA request)</p>														
FORCMA	<p>Force Match A Bit. For output modes, the FORCMA bit is equivalent to a successful comparison on comparator A (except that the FLAG bit is not set). This bit is cleared by reset and is always read as 0. This bit is valid for every output operation mode which uses comparator A, otherwise it has no effect.</p> <p>0 Has no effect 1 Force a match at comparator A</p> <p>For input modes, the FORCMA bit is not used and writing to it has no effect.</p>														
FORCMB	<p>Force Match B Bit. For output modes, the FORCMB bit is equivalent to a successful comparison on comparator B (except that the FLAG bit is not set). This bit is cleared by reset and is always read as 0. This bit is valid for every output operation mode which uses comparator B, otherwise it has no effect.</p> <p>0 Has no effect 1 Force a match at comparator B</p> <p>For input modes, the FORCMB bit is not used and writing to it has no effect.</p>														

Table 26-9. EMIOS_CCR[n] Field Descriptions (continued)

Field	Description										
BSL	<p>Bus Select Bits. The BSL bits are used to select either one of the counter buses or the internal counter to be used by the unified channel.</p> <table border="1"> <thead> <tr> <th>BSL</th> <th>Selected Bus</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>All channels: counter bus[A]</td> </tr> <tr> <td>01</td> <td>Channels 0 to 7: counter bus[B] Channels 8 to 15: counter bus[C] Channels 16 to 23: counter bus[D]</td> </tr> <tr> <td>10</td> <td>Reserved</td> </tr> <tr> <td>11</td> <td>All channels: internal counter</td> </tr> </tbody> </table>	BSL	Selected Bus	00	All channels: counter bus[A]	01	Channels 0 to 7: counter bus[B] Channels 8 to 15: counter bus[C] Channels 16 to 23: counter bus[D]	10	Reserved	11	All channels: internal counter
BSL	Selected Bus										
00	All channels: counter bus[A]										
01	Channels 0 to 7: counter bus[B] Channels 8 to 15: counter bus[C] Channels 16 to 23: counter bus[D]										
10	Reserved										
11	All channels: internal counter										
EDSEL	<p>Edge Selection Bit. For input modes, the EDSEL bit selects if the internal counter is triggered by both edges of a pulse or by a single edge only as defined by the EDPOL bit. When not shown in the mode of operation description, this bit has no effect.</p> <p>0 Single edge triggering defined by the EDPOL bit 1 Both edges triggering</p> <p>For GPIO in mode, the EDSEL bit selects if a FLAG can be generated.</p> <p>0 A FLAG is generated as defined by the EDPOL bit 1 No FLAG is generated</p> <p>For SAOC mode, the EDSEL bit selects the behavior of the output flip-flop at each match.</p> <p>0 The EDPOL value is transferred to the output flip-flop 1 The output flip-flop is toggled</p>										
EDPOL	<p>Edge Polarity Bit. For input modes, the EDPOL bit asserts which edge triggers either the internal counter or an input capture or a FLAG. When not shown in the mode of operation description, this bit has no effect.</p> <p>0 Trigger on a falling edge 1 Trigger on a rising edge</p> <p>For output modes, the EDPOL bit is used to select the logic level on the output pin.</p> <p>0 A match on comparator A clears the output flip-flop, while a match on comparator B sets it 1 A match on comparator A sets the output flip-flop, while a match on comparator B clears it</p>										
MODE	<p>Mode Selection Bits. The MODE bits select the mode of operation of the unified channel, as shown in Table 26-10. Refer to Table 26-1 for more information on the different modes.</p> <p>Note: If a reserved value is written to MODE the results are unpredictable.</p>										

Table 26-10. MODE Bits

MODE	Mode of Operation
000_0000	GPIO (input)
000_0001	GPIO (output)
000_0010	SAIC
000_0011	SAOC
000_0100	IPWM
000_0101	IPM

Table 26-10. MODE Bits (continued)

MODE	Mode of Operation
000_0110	DAOC (flag set on the second match)
000_0111	DAOC (flag set on both match)
000_1000– 100_1111	Reserved
101_0000	MCB (up counter, internal clock)
101_0001	MCB (up counter, external clock)
101_0010	Reserved
101_0011	Reserved
101_0100	MCB (up/down counter, flag on A1 match, internal clock)
101_0101	MCB (up/down counter, flag on A1 match, external clock)
101_0110	MCB (up/down counter, flag on A1 match or cycle boundary, internal clock)
101_0111	MCB (up/down counter, flag on A1 match or cycle boundary, external clock)
101_1000	OPWFMB (flag on B1 match)
101_1001	Reserved
101_1010	OPWFMB (flag on A1 or B1 matches)
101_1011	Reserved
101_1100	OPWMCB (flag in trailing edge, trail edge dead-time)
101_1101	OPWMCB (flag in trailing edge, lead edge dead-time)
101_1110	OPWMCB (flag in both edges, trail edge dead-time)
101_1111	OPWMCB (flag in both edges, lead edge dead-time)
110_0000	OPWMB (flag on B1 match)
110_0001	Reserved
110_0010	OPWMB (flag on A1 or B1 matches)
110_0011– 111_1111	Reserved

26.4.9 eMIOS200 Status Register (EMIOS_CSR[n])

Offset: UC[n] base address + 0x0010

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OVR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVFL	0	0	0	0	0	0	0	0	0	0	0	0	UCIN	UCOUT	FLAG
W	w1c															w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-10. eMIOS200 Status Register (EMIOS_CSR[n])

Table 26-11. EMIOS_CSR[n] Field Descriptions

Field	Description
OVR	Overflow Bit. The OVR bit indicates that FLAG generation occurred when the FLAG bit was already set. This bit can be cleared by clearing the FLAG bit or by software writing a 1. 0 Overrun has not occurred 1 Overrun has occurred
OVFL	Overflow Bit. The OVFL bit indicates that an overflow has occurred in the internal counter. This bit must be cleared by software writing a 1. 0 An overflow has not occurred 1 An overflow has occurred
UCIN	Unified Channel Input Pin Bit. The UCIN bit reflects the input pin state after being filtered and synchronized.
UCOUT	Unified Channel Output. The UCOUT bit reflects the output pin state.
FLAG	FLAG Bit. The FLAG bit is set when an input capture or a match event in the comparators occurred. This bit must be cleared by software writing a 1. 0 FLAG cleared 1 FLAG set event has occurred Note: emios_flag_out reflects the FLAG bit value. When DMA bit is set, the FLAG bit can be cleared by the DMA controller.

26.5 Functional Description

The three types of channels of the eMIOS200 can operate in the modes as listed in [Table 26-1](#).

The eMIOS200 provides independently operating unified channels (UC) that can be configured and accessed by a host MCU. Up to four time bases can be shared by the channels through four counter buses and each unified channel can generate its own time base.

The eMIOS200 block is reset at positive edge of the clock (synchronous reset). All registers are cleared on reset.

26.5.1 Unified Channel (UC)

[Figure 26-11](#) shows the unified channel block diagram. Each unified channel consists of:

- Counter bus selector, which selects the time base to be used by the channel for all timing functions
- A programmable clock prescaler
- Two double buffered data registers, A and B, that allow up to two input capture and/or output compare events to occur before software intervention is needed
- Two comparators (equal only), A and B, which compare the selected counter bus with the value in the data registers
- Internal counter, which can be used as a local time base or to count input events
- Programmable input filter, which ensures that only valid pin transitions are received by channel
- Programmable input edge detector, which detects the rising, falling or either edges
- An output flip-flop, which holds the logic level to be applied to the output pin
- eMIOS200 status and control register
- An output disable input selector, which selects the output disable input signal that will be used as output disable

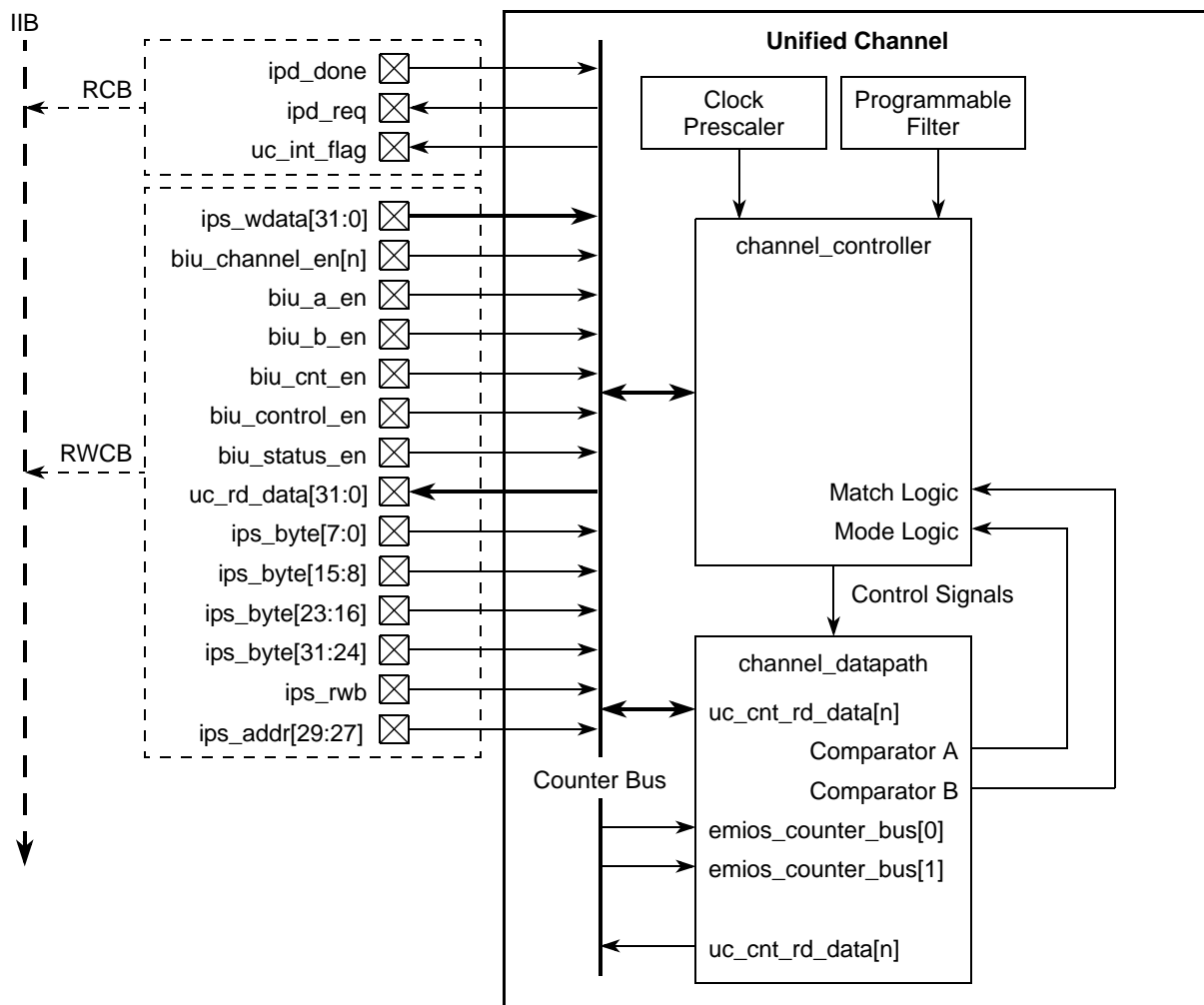


Figure 26-11. Unified Channel Block Diagram

The [Figure 26-12](#) shows the unified channel control block diagram.

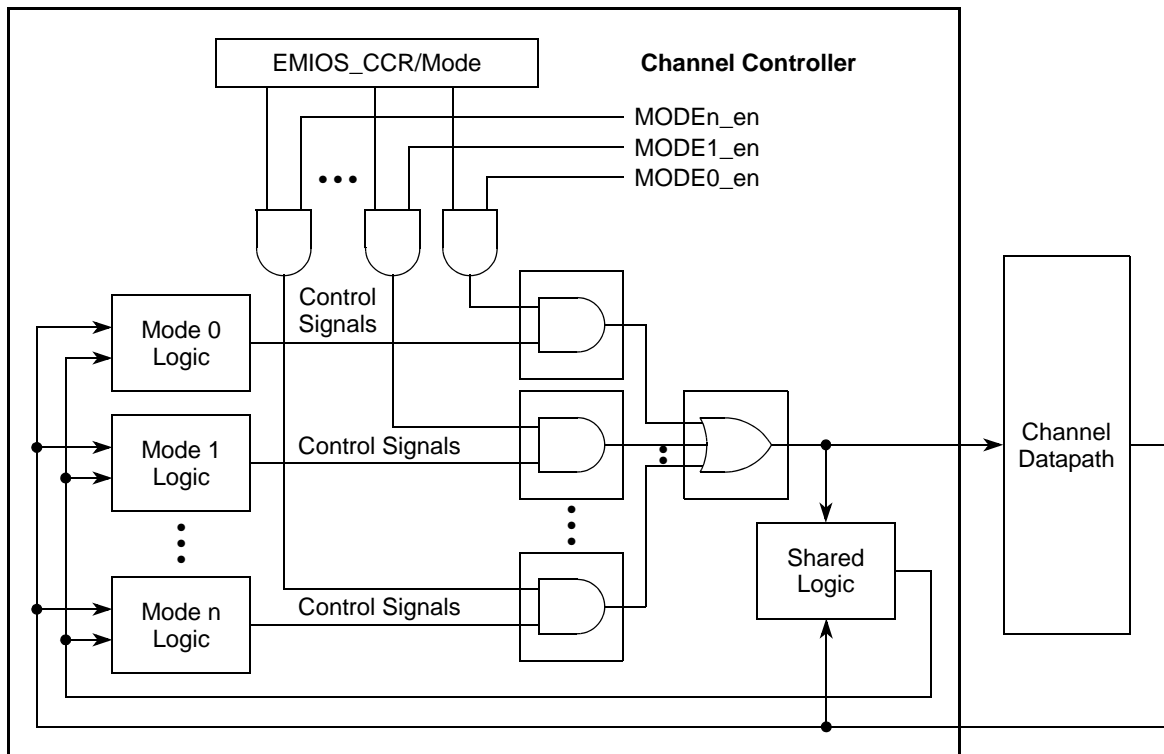


Figure 26-12. Unified Channel Control Block Diagram

26.5.1.1 Unified Channel Modes of Operation

The mode of operation of the unified channel is determined by the mode select bits MODE in the EMIOS_CCR[n] register (see [Table 26-10](#) for details).

When entering an output mode (except for GPIO mode), the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[n] register.

As the internal counter EMIOS_CCNTR[n] continues to run in all modes (except for GPIO mode), it is possible to use this as a time base if the resource is not used in the current mode.

26.5.1.1.1 General-Purpose Input/Output (GPIO) Mode

In GPIO mode, all input capture and output compare functions of the unified channel are disabled, the internal counter (EMIOS_CCNTR[n] register) is cleared and disabled. All control bits remain accessible. In order to prepare the unified channel for a new operation mode, writing to registers EMIOS_CADR[n] or EMIOS_CBDR[n] stores the same value in registers A1/A2 or B1/B2, respectively.

The MODE[6] bit selects between input (MODE[6] = 0) and output (MODE[6] = 1) modes.

When changing MODE, the application software must go to GPIO mode first to reset the unified channel's internal functions properly. Failure to do this could lead to invalid and unexpected output compare or input capture results or the FLAGS being set incorrectly.

In GPIO input mode, the FLAG generation is determined according to EDPOL and EDSEL bits and the input pin status can be determined by reading the UCIN bit.

In GPIO output mode, the unified channel is used as a single output port pin and the value of the EDPOL bit is permanently transferred to the output flip-flop.

26.5.1.1.2 Single Action Input Capture (SAIC) Mode

In SAIC mode, when a triggering event occurs on the input pin, the value on the selected time base is captured into register A2. At the same time, the FLAG bit is set to indicate that an input capture has occurred. Register EMIOS_CADR[n] returns the value of register A2

The input capture is triggered by a rising, falling or either edges in the input pin, as configured by EDPOL and EDSEL bits in EMIOS_CCR[n] register.

Figure 26-13 shows how the unified channel can be used for input capture.

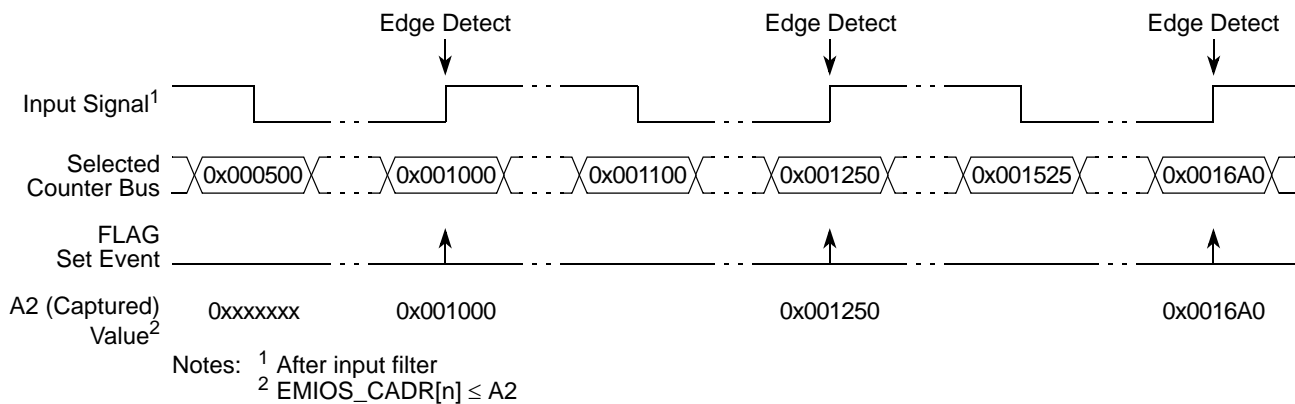


Figure 26-13. Single Action Input Capture Example

26.5.1.1.3 Single Action Output Compare (SAOC) Mode

In SAOC mode, a match value is loaded in register A2 and then transferred to register A1 to be compared with the selected time base. When a match occurs, the EDSEL bit selects if the output flip-flop is toggled or if the value in EDPOL is transferred to it. At the same time, the FLAG bit is set to indicate that the output compare match has occurred. Writing to register EMIOS_CADR[n] stores the value in register A2 and reading to register EMIOS_CADR[n] returns the value of register A1.

An output compare match can be simulated in software by setting the FORCMA bit in EMIOS_CCR[n] register. In this case, the FLAG bit is not set.

Figure 26-14 and Figure 26-15 show how the unified channel can be used to perform a single output compare with EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively.

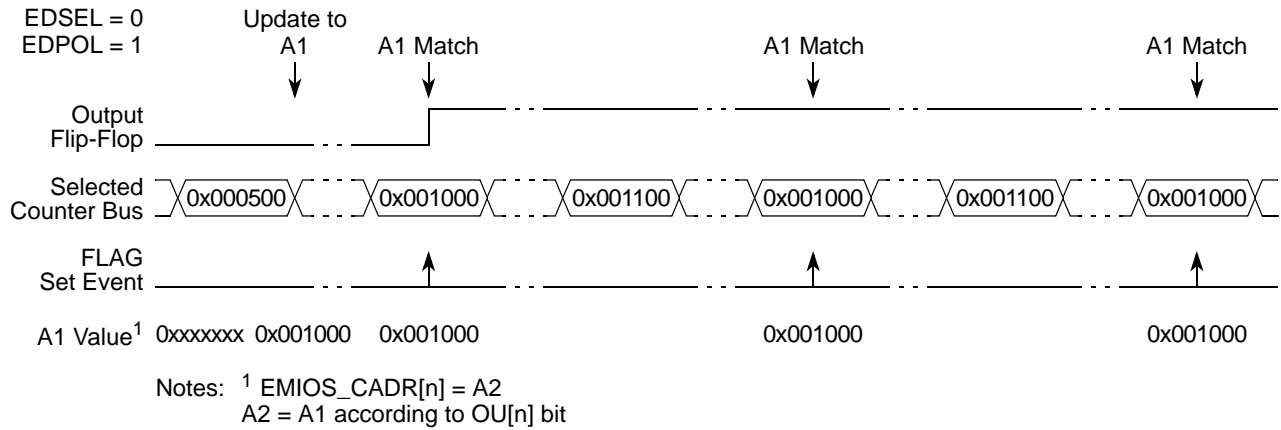


Figure 26-14. SAOC Example — EDPOL value being transferred to the output flip-flop

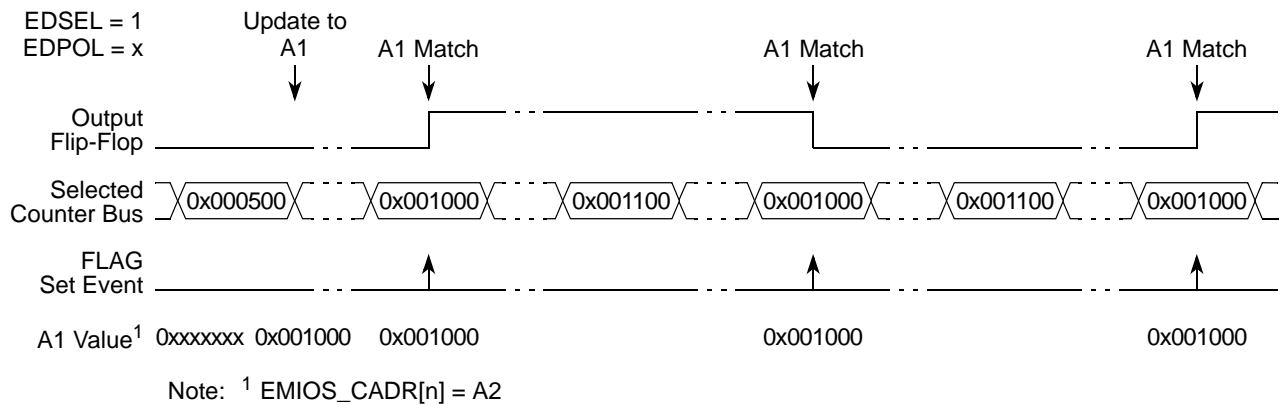


Figure 26-15. SAOC Example — Toggling the Output Flip-Flop

26.5.1.1.4 Input Pulse-Width Measurement (IPWM) Mode

The IPWM mode allows the measurement of the width of a positive or negative pulse by capturing the leading edge on register B1 and the trailing edge on register A2. Successive captures are done on consecutive edges of opposite polarity. The leading edge sensitivity (i.e., pulse polarity) is selected by EDPOL bit in the EMIOS_CCR[n] register. Registers EMIOS_CADR[n] and EMIOS_CBDR[n] return the values in register A2 and B1, respectively.

The capture function of register A2 remains disabled until the first leading edge triggers the first input capture on register B2. When this leading edge is detected, the count value of the selected time base is latched into register B2; the FLAG bit is not set. When the trailing edge is detected, the count value of the selected time base is latched into register A2 and, at the same time, the FLAG bit is set and the content of register B2 is transferred to register B1 and to register A1.

If subsequent input capture events occur while the corresponding FLAG bit is set, registers A2, B1, and A1 will be updated with the latest captured values and the FLAG will remain set. Registers EMIOS_CADR[n] and EMIOS_CBDR[n] return the value in registers A2 and B1, respectively.

In order to guarantee coherent access, reading EMIOS_CADR[n] forces B1 to be updated with the content of register A1. At the same time transfers between B2 and B1 are disabled until the next read of

EMIOS_CBDR[n] register. Reading EMIOS_CBDR[n] register forces B1 be updated with A1 register content and re-enables transfers from B2 to B1, to take effect at the next trailing edge capture. Transfers from B2 to A1 are not blocked at any time.

The input pulse width is calculated by subtracting the value in B1 from A2.

Figure 26-16 shows how the unified channel can be used for input pulse-width measurement.

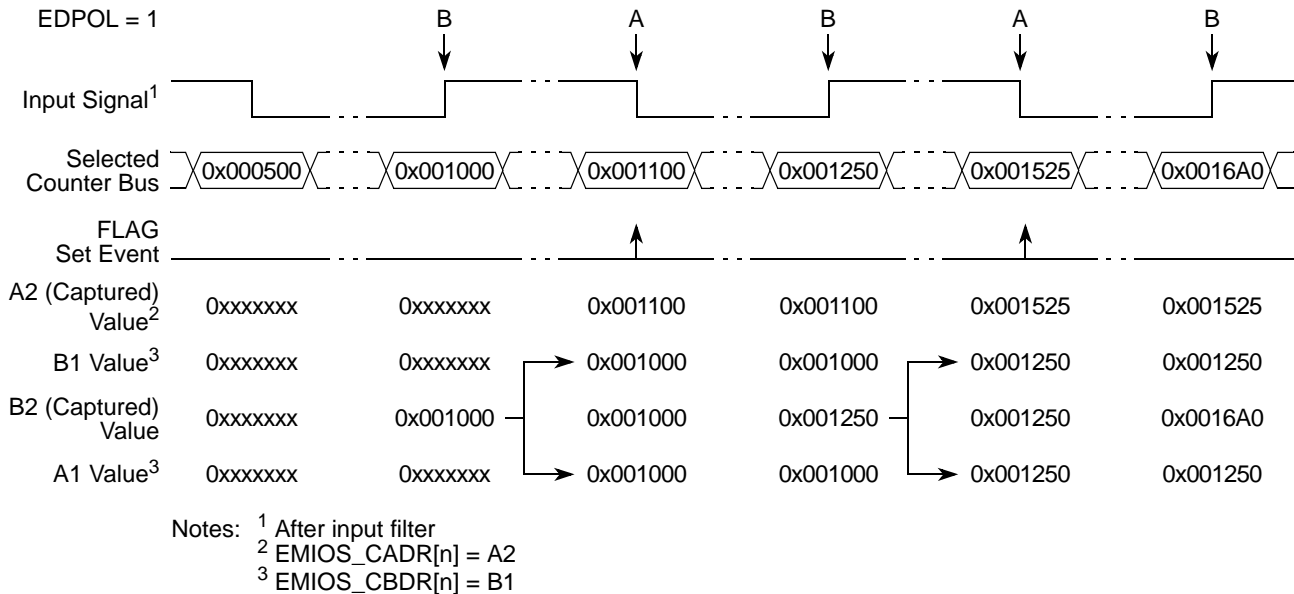


Figure 26-16. Input Pulse-Width Measurement Example

Figure 26-17 shows the A1 and B1 updates when EMIOS_CADR[n] and EMIOS_CBDR[n] register reads occur. The A1 register has always coherent data related to A2 register. When EMIOS_CADR[n] read is performed, the B1 register is loaded with the A1 register content. This guarantees that the data in register B1 always has the coherent data related to the last EMIOS_CADR[n] read. The B1 register updates remain locked until EMIOS_CBDR[n] read occurs. If EMIOS_CADR[n] read is performed, B1 is updated with A1 register content even if the B1 update is locked by a previous EMIOS_CADR[n] read operation.

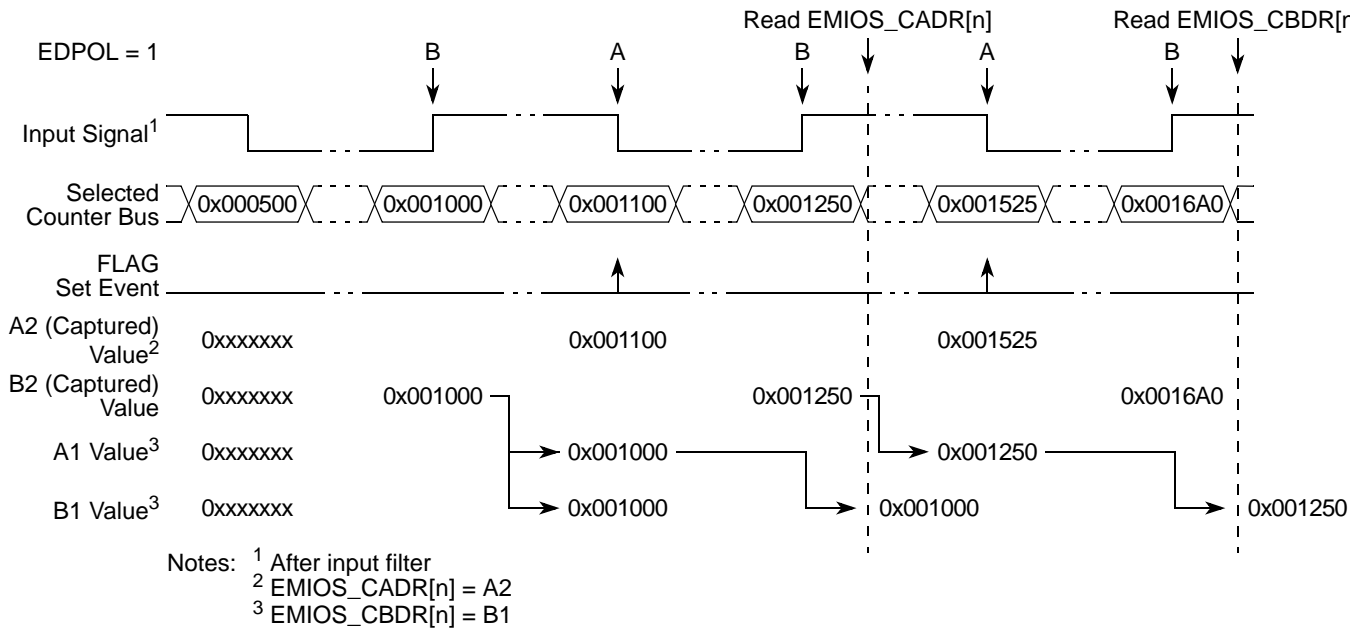


Figure 26-17. B1 and A1 Updates at EMIOS_CADR[n] and EMIOS_CBDR[n] Reads

Reading EMIOS_CADR[n] followed by EMIOS_CBDR[n] always provides coherent data. If no coherent data is required, the sequence of reads should be inverted, therefore EMIOS_CBDR[n] should be read prior to EMIOS_CADR[n] register. Even in this case B1 register updates will be blocked after EMIOS_CADR[n] read, therefore a second EMIOS_CBDR[n] is required to release the B1 register updates.

26.5.1.1.5 Input Period Measurement (IPM) Mode

The IPM mode allows the measurement of the period of an input signal by capturing two consecutive rising edges or two consecutive falling edges. Successive input captures are done on consecutive edges of the same polarity. The edge polarity is defined by the EDPOL bit in the EMIOS_CCR[n] register.

When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set and the values in registers B1 are meaningless. On the second and subsequent captures, the FLAG line is set and data in register B2 is transferred to register B1.

When the second edge of the same polarity is detected, the counter bus value is latched into registers A2 and B2, and the data previously held in register B2 is transferred to data register B1 and to register A1. The FLAG bit is set to indicate that the start and end points of a complete period have been captured. This sequence of events is repeated for each subsequent capture. Registers EMIOS_CADR[n] and EMIOS_CBDR[n] return the values in register A2 and B1, respectively.

To allow coherent data, reading EMIOS_CADR[n] forces A1 content be transferred to B1 register and disables transfers between B2 and B1. These transfers are disabled until the next read of the EMIOS_CBDR[n] register. Reading EMIOS_CBDR[n] register forces A1 content to be transferred to B1 and re-enables transfers from B2 to B1, to take effect at the next edge capture.

The input pulse period is calculated by subtracting the value in B1 from A2.

Figure 26-18 shows how the unified channel can be used for input period measurement.

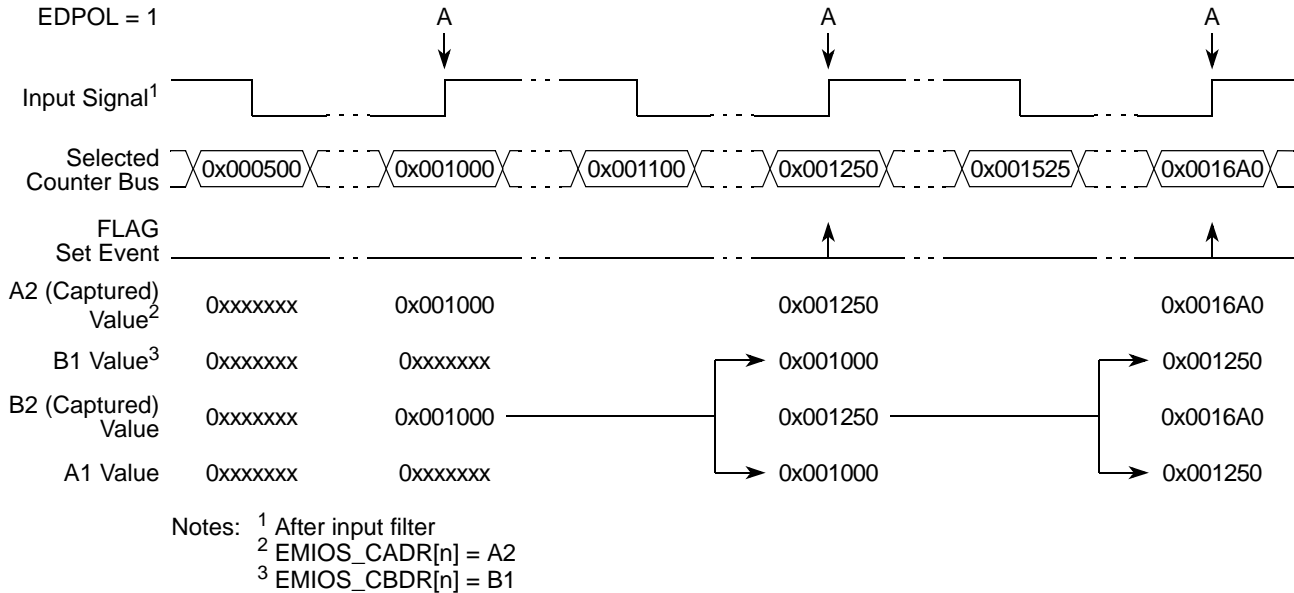


Figure 26-18. Input Period Measurement Example

Figure 26-19 describes the A1 and B1 register updates when EMIOS_CADR[n] and EMIOS_CBDR[n] read operations are performed. When EMIOS_CADR[n] read occurs the content of A1 is transferred to B1 thus providing coherent data in A2 and B1 registers. Transfers from B2 to B1 are then blocked until EMIOS_CBDR[n] is read. After EMIOS_CBDR[n] is read, register A1 content is transferred to register B1 and the transfers from B2 to B1 are re-enabled to occur at the transfer edges, which is the leading edge in the Figure 26-19 example.

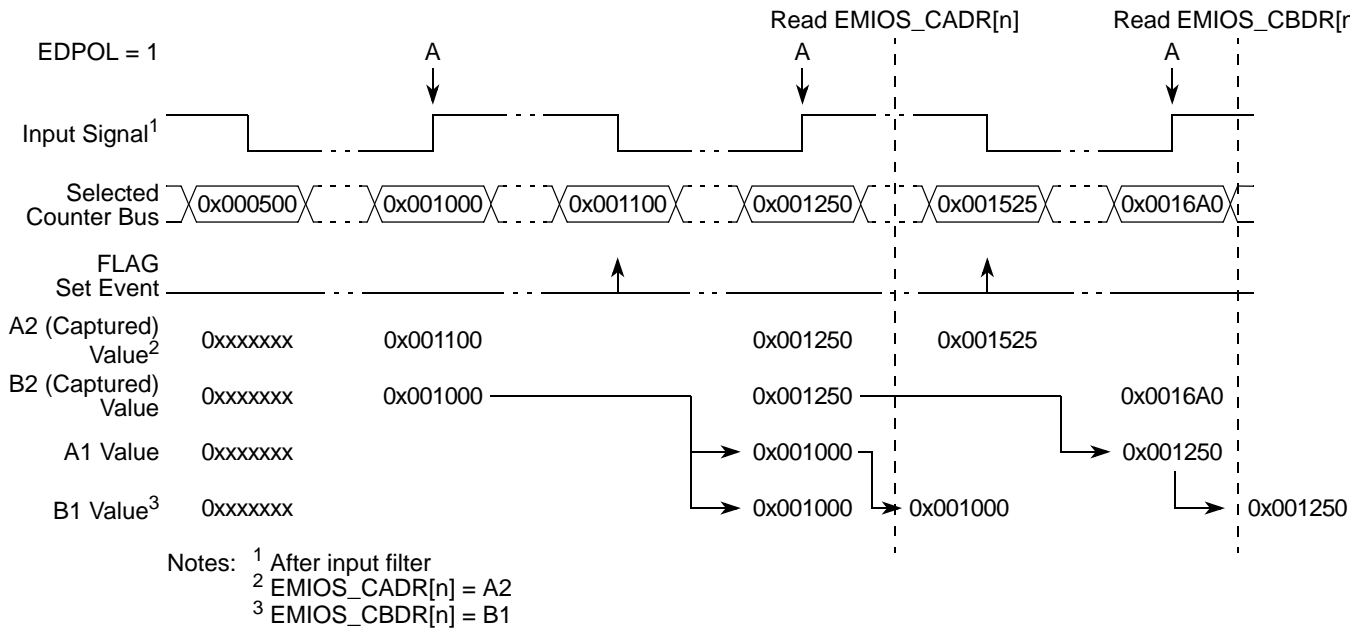


Figure 26-19. A1 and B1 Updates at EMIOS_CADR[n] and EMIOS_CBDR[n] Reads

26.5.1.1.6 Double Action Output Compare (DAOC) Mode

In the DAOC mode the leading and trailing edges of the variable pulse-width output are generated by matches occurring on comparators A and B, respectively.

When the DAOC mode is first selected (coming from GPIO mode) both comparators are disabled. Comparators A and B are enabled by updating registers A1 and B1 respectively and remain enabled until a match occurs on that comparator, when it is disabled again. In order to update registers A1 and B1, a write to A2 and B2 must occur and the OUDIS[n] bit must be cleared.

The output flip-flop is set to the value of EDPOL when a match occurs on comparator A and to the complement of EDPOL when a match occurs on comparator B.

MODE[6] controls if the FLAG is set on both matches or on the second match only (see Table 26-10 for details).

If subsequent enabled output compares occur on registers A1 and B1, pulses will continue to be generated, regardless of the state of the FLAG bit.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a comparison event in comparator A or B, respectively. The FLAG bit is not affected by these forced operations.

NOTE

If registers A1 and B1 are loaded with the same value, the unified channel behaves as if a single match on comparator B had occurred, i.e., the output pin will be set to the complement of EDPOL bit and the FLAG bit is set.

Figure 26-20 and Figure 26-21 show how the unified channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.

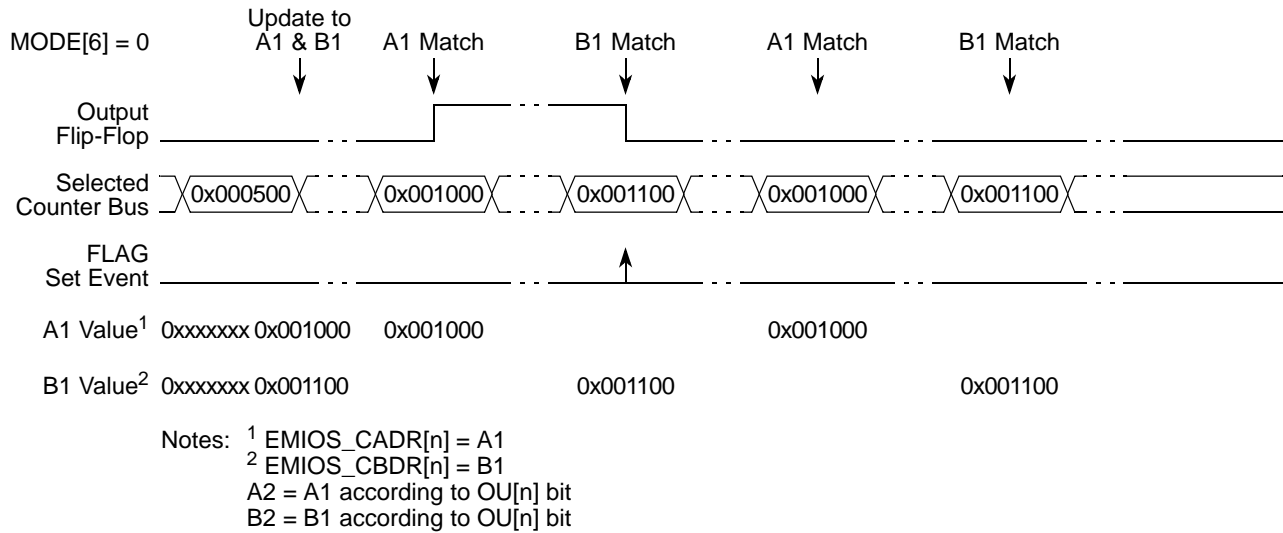


Figure 26-20. Double Action Output Compare with FLAG Set on the Second Match

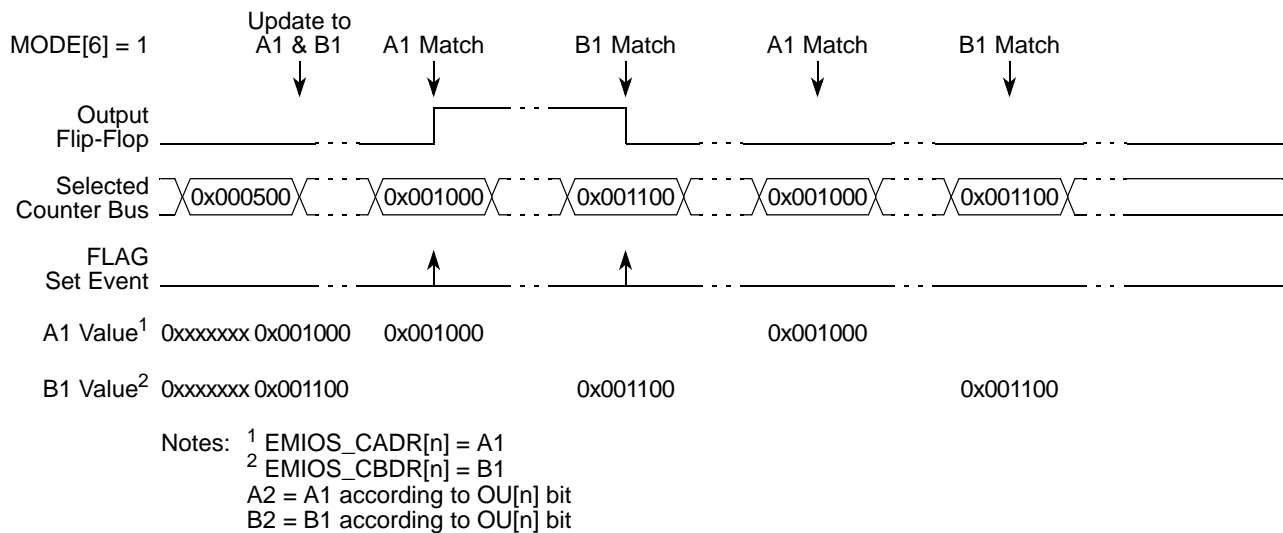


Figure 26-21. Double Action Output Compare with FLAG Set on Both Matches

26.5.1.1.7 Modulus Counter Buffered (MCB) Mode

The MCB mode provides a time base that can be shared with other channels through the internal counter buses. Register A1 is double buffered thus allowing smooth transitions between cycles when changing A2 register value. A1 register is updated at the cycle boundary, which is defined as when the internal counter reaches the value one. The internal counter values are within a range from one up to register A1 value in MCB mode. The internal counter must not reach 0x0 as consequence of a rollover. To avoid this the user must start MCB only if the value stored at internal counter is fewer than the value that EMIOS_CADR register stores.

MODE[6] bit selects the internal clock source if set to zero or external, if set to one. When the external clock is selected the input channel pin is used as the channel clock source. The active edge of this clock is defined by EDPOL and EDSEL bits in the EMIOS_CCR channel register.

When entering in MCB mode, if the up counter is selected by MODE[4]=0, the internal counter starts counting from its current value to up direction until A1 match occurs. On the next system clock cycle after the A1 match occurs, the internal counter is set to one. If up/down counter is selected by setting MODE[4]=1, the counter changes direction at the A1 match and counts down until it reaches the value one. After it has reached one, it is set to count in up direction again. Register B1 is set to one at mode entering and cannot be changed while this mode is selected. B1 register is used to generate a match to set the internal counter in up-count direction if up/down mode is selected.

The MCB mode counts between one and A1 register value. Only values greater than 0x1 are allowed to be written at A1 register. Loading values other than those leads to unpredictable results. The counter cycle period is equal to A1 value in up counter mode. If in up/down counter mode the period is defined by the expression: $(2*A1)-2$.

Figure 26-22 describes the counter cycle for several A1 values. Register A1 is loaded with A2 register value at the cycle boundary. Any value written to A2 register within cycle (n) will be updated to A1 at the next cycle boundary and therefore will be used on cycle (n+1). The cycle boundary between cycle (n) and cycle (n+1) is defined as the first system clock cycle of cycle (n+1). The flags are generated as soon as A1 match had occurred.

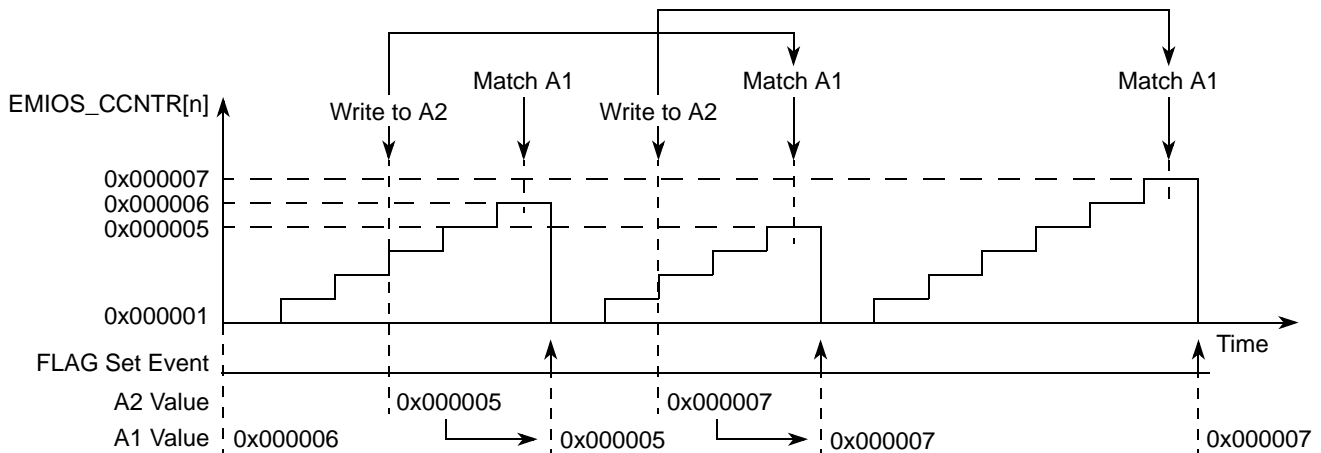


Figure 26-22. Modulus Counter Buffered (MCB) Up Count Mode

Figure 26-23 describes the MCB in up/down counter mode. A1 register is updated at the cycle boundary. If A2 is written in cycle (n), this new value will be used in cycle (n+1) for A1 match.

Flags are generated at A1 match only if MODE[5] is 0. If MODE[5] is set to 1 flags are also generated at the cycle boundary.

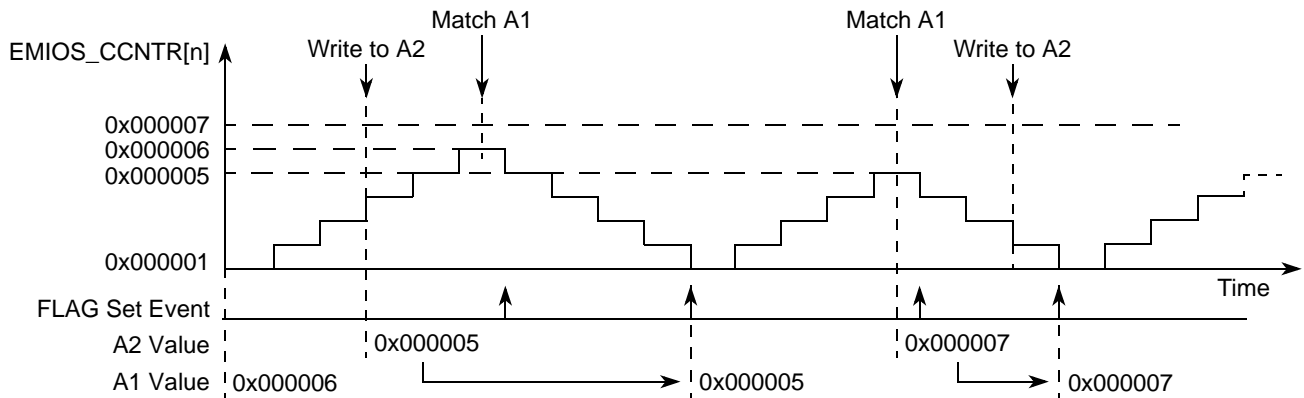


Figure 26-23. Modulus Counter Buffered (MCB) Up/Down Mode

Figure 26-24 describes the A1 register update process in up counter mode. The A1 load signal is generated based on the detection of the internal counter reaching one and has the duration of one system clock cycle. During the load pulse A1 still holds its previous value. It is updated at the second system clock cycle only.

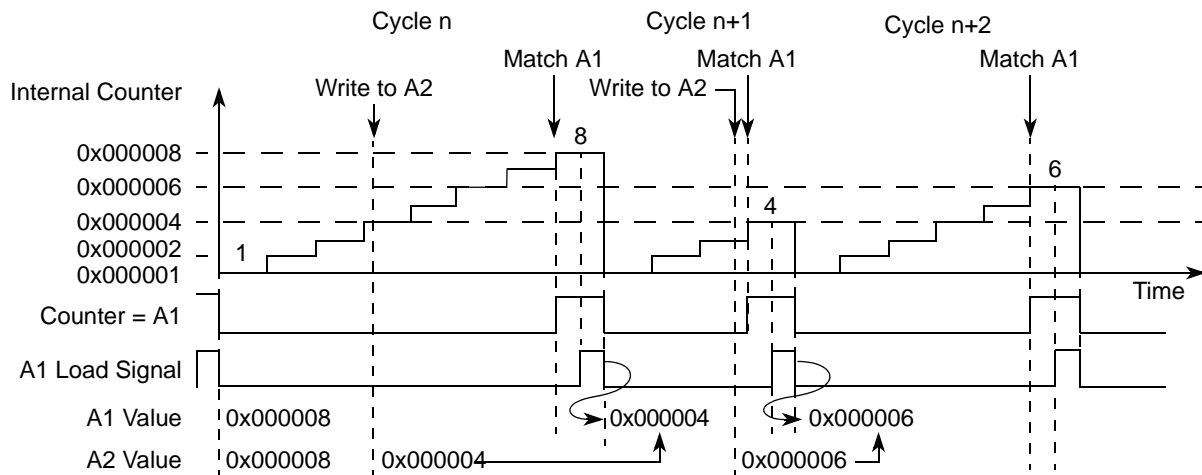


Figure 26-24. MCB Mode A1 Register Update in Up Counter Mode

Figure 26-25 describes the A1 register update in up/down counter mode. Note that A2 can be written at any time within cycle (n) in order to be used in cycle (n+1). Thus A1 receives this new value at the next cycle boundary. The update disable bits OUDIS[n] can be used to disable the update of A1 register.

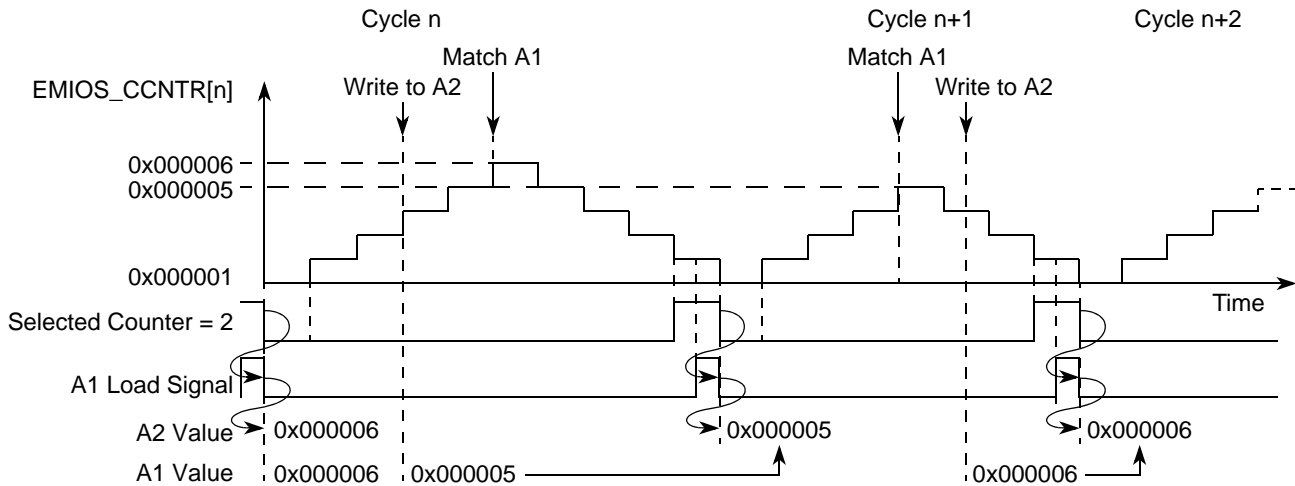


Figure 26-25. MCB Mode A1 Register Update in Up/Down Counter Mode

26.5.1.1.8 Pulse-Width and Frequency Modulation Buffered (OPWFMB) Mode

This mode provides waveforms with variable duty cycle and frequency. The internal channel counter is automatically selected as the time base when this mode is selected. A1 register indicates the duty cycle and B1 register the frequency. Both A1 and B1 registers are double buffered to allow smooth signal generation when changing the registers values. It supports 0% and 100% duty cycles.

To provide smooth and consistent channel operation this mode differs substantially from the OPWFM mode. The main differences reside in the A1 and B1 registers update, on the delay from the A1 match to the output pin transition and on the range of the internal counter values which starts from 1 up to B1 register value. The internal counter must not reach 0x0 as consequence of a rollover. To avoid this, the user must start OPWFMB only if the value stored at internal counter is fewer than the value that EMIOS_CBDR register stores.

When a match on comparator A occurs the output register is set to the value of EDPOL. When a match on comparator B occurs the output register is set to the complement of EDPOL. B1 match also causes the internal counter to transition to 1, thus restarting the counter cycle.

Only values greater than 0x1 are allowed to be written to B1 register. Loading values other than those leads to unpredictable results.

Figure 26-26 describes the operation of the OPWFMB mode regarding output pin transitions and A1/B1 registers match events. The output pin transition occurs when the A1 or B1 match signal is deasserted, which is indicated by the A1 match negedge detection signal. If register A1 is set to 0x000004, the output pin transitions four counter periods after the cycle has started, plus one system clock cycle. In the example shown in Figure 26-26 the internal counter prescaler is set to two.

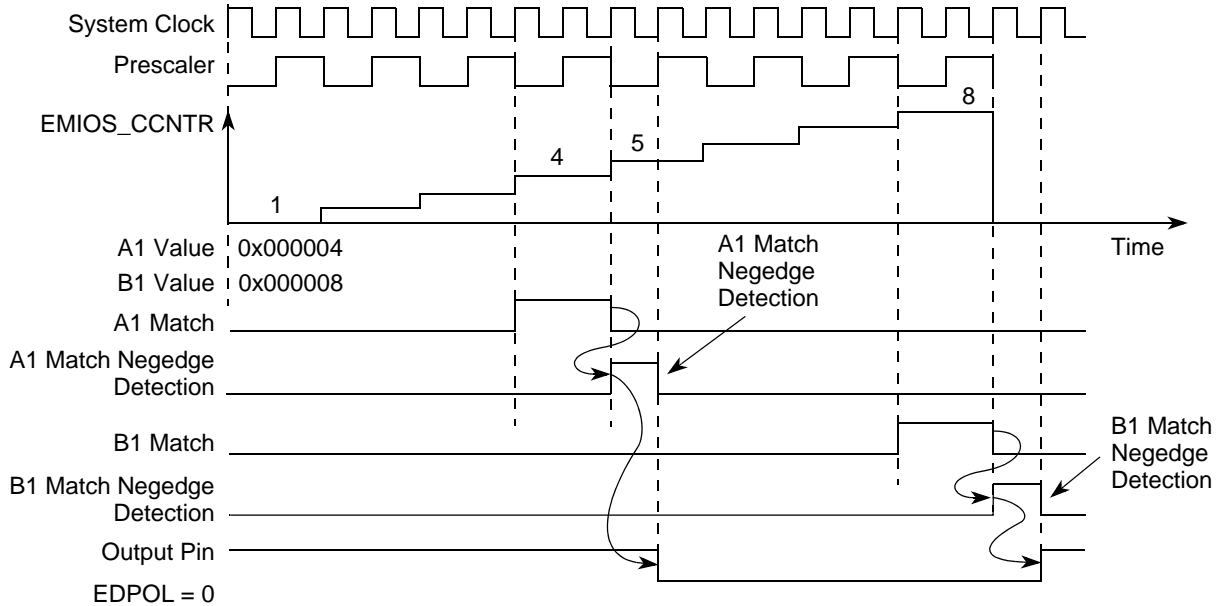


Figure 26-26. OPWFMB A1 and B1 Match to Output Register Delay

Figure 26-27 describes the generated output signal if A1 is set to zero. Because the counter does not reach zero in this mode, the channel internal logic infers a match as if A1=1 with the difference that in this case, the posedge of the match signal is used to trigger the output pin transition instead of the negedge used when A1=1. A1 posedge match signal from cycle (n+1) occurs at the same time as B1 negedge match signal from cycle (n). This allows to use the A1 posedge match to mask the B1 negedge match when they occur at the same time. The result is that no transition occurs on the output flip-flop and a 0% duty cycle is generated.

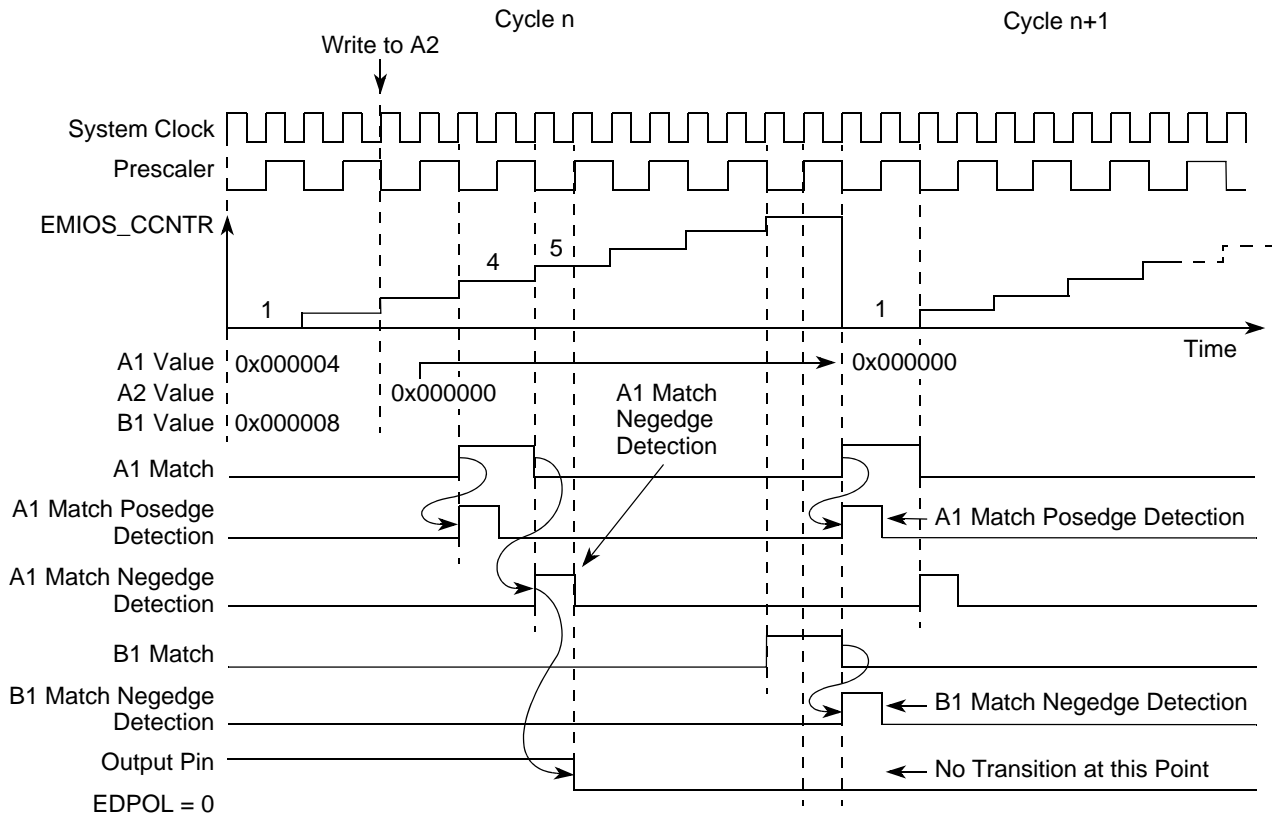


Figure 26-27. OPWFMB Mode with A1 = 0 (0% duty cycle)

Figure 26-28 describes the timing for the A1 and B1 registers load. The A1 and B1 load use the same signal which is generated based on the selected counter reaching the value one, or $EMIOS_CCNTR[n] = 1$. This event is defined as the cycle boundary. The load signal pulse has the duration of one system clock cycle and occurs at the first system clock period of every counter cycle. If A2 and B2 are written within cycle (n), their values are loaded into A1 and B1, respectively, at the first clock of cycle (n+1) and the new values are used for matches at cycle (n+1). The update disable bits $OU\text{DIS}[n]$ can be used to control the update of these registers, thus allowing to delay the A1 and B1 registers update for synchronization purposes.

In Figure 26-28 it is assumed that the channel and global prescalers are set to one, meaning that the channel internal counter transition at every system clock cycle. FLAGS can be generated only on B1 matches when $MODE[5]$ is cleared, or on either A1 or B1 matches when $MODE[5]$ is set. Because B1 FLAG occurs at the cycle boundary, this flag can be used to indicate that A2 or B2 data written on cycle (n) were loaded to A1 or B1, respectively, thus generating matches in cycle (n+1).

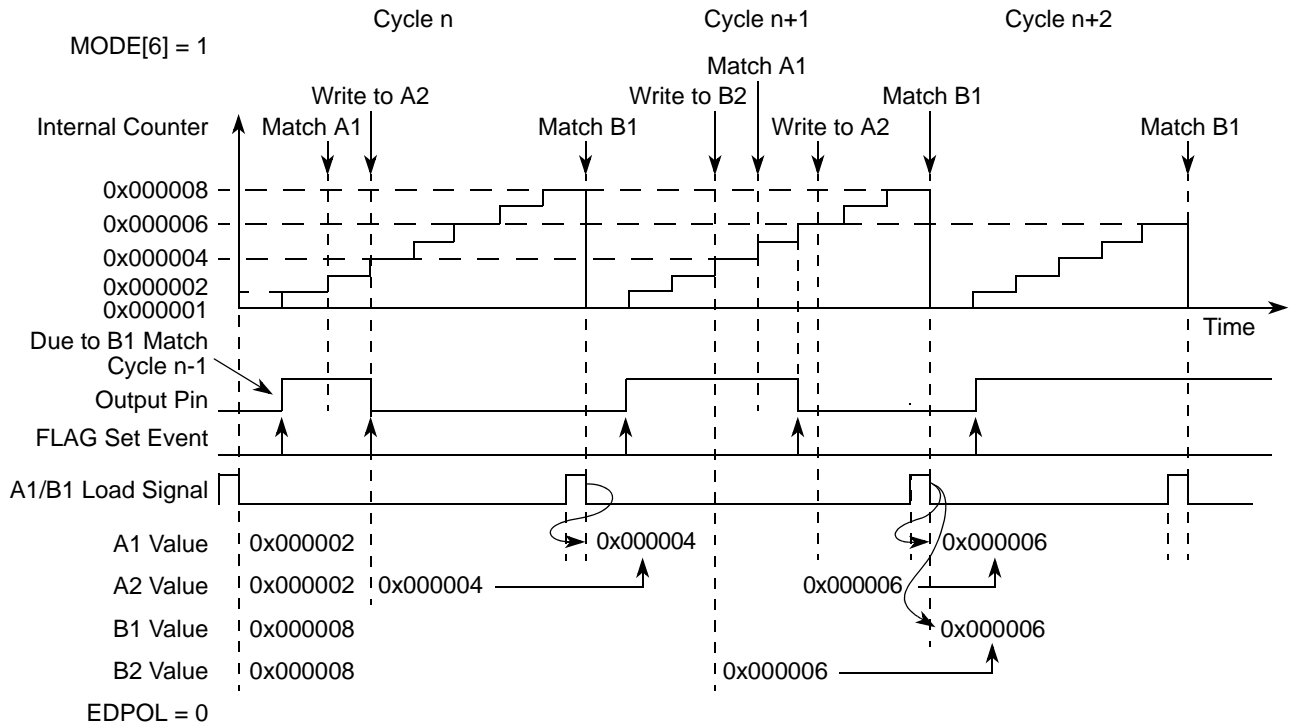


Figure 26-28. OPWFMB A1 and B1 Registers Update and Flags

Figure 26-29 describes the operation of the output disable feature in OPWFMB mode. The output disable forces the channel output flip-flop to EDPOL bit value. This functionality targets applications that use active high signals and a high to low transition at A1 match. In this case EDPOL should be set to 0.

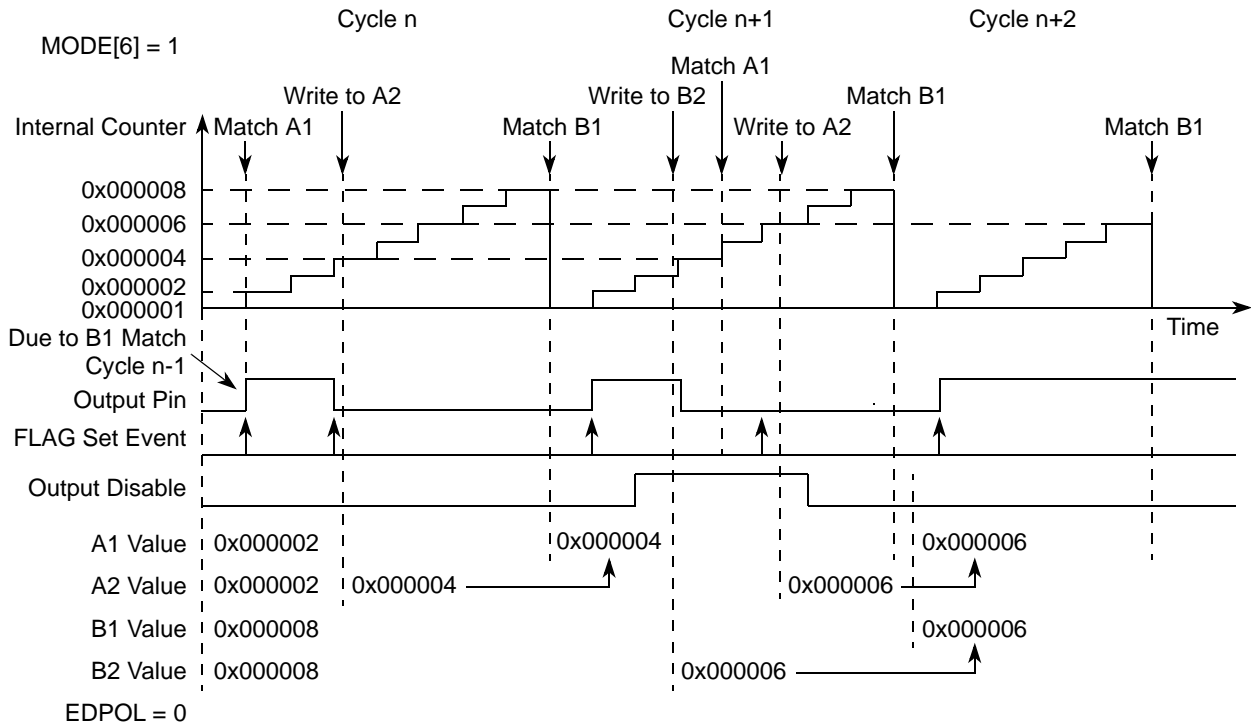


Figure 26-29. OPWFMB Mode with Active Output Disable

The output disable has a synchronous operation, meaning that the assertion of the output disable input pin causes the channel output flip-flop to transition to EDPOL at the next system clock cycle. If the output disable input is deasserted the output pin transition at the following A1 or B1 match.

In [Figure 26-29](#) it is assumed that the output disable input is enabled and selected for the channel. Refer to [Section 26.4.8, “eMIOS200 Control Register \(EMIOS_CCR\[n\]\),”](#) for a description of how the ODIS and ODISSL bits enable and select the output disable inputs.

The FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on comparators A or B respectively. Similar to a B1 match FORCMB sets the internal counter to 0x000001. The FLAG bit is not set by the FORCMA or FORCMB bits being asserted.

[Figure 26-30](#) describes the generation of 100% and 0% duty cycle signals. It is assumed EDPOL=0 and the resultant prescaler value is 1. Initially, A1=0x000008 and B1=0x000008. In this case, the B1 match has precedence over the A1 match, thus the output flip-flop is set to the complement of EDPOL bit. This cycle corresponds to a 100% duty cycle signal. The same output signal can be generated for any A1 value greater or equal to B1.

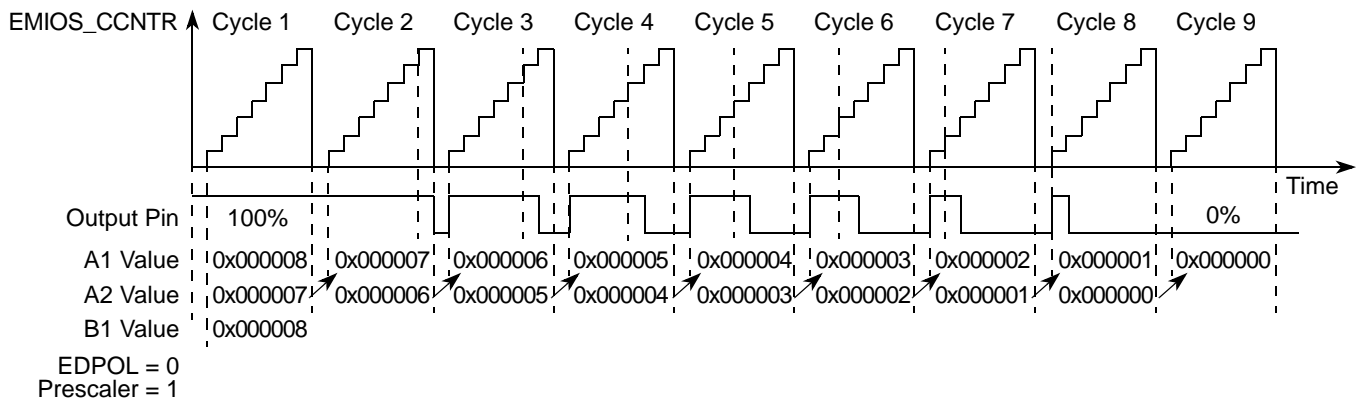


Figure 26-30. OPWFMB Mode from 100% to 0% Duty Cycle

A 0% duty cycle signal is generated if $A1=0$ as shown in cycle 9 in [Figure 26-30](#). In this case, the $B1=0x000008$ match from cycle 8 occurs at the same time as the $A1=0x000000$ match from cycle 9. Refer to [Figure 26-27](#) for a description of the A1 and B1 match generation. In this case, the A1 match has precedence over the B1 match and the output signal transitions to EDPOL.

26.5.1.1.9 Center-Aligned Output PWM Buffered with Dead-Time (OPWMCB) Mode

This operation mode generates a center-aligned PWM with dead-time insertion to the leading or trailing edge. A1 and B1 registers are double buffered to allow smooth output signal generation when changing A2 or B2 registers values.

The selected counter bus must be running in up/down counter mode, as shown in [Figure 26-23](#). The time base selected for a channel configured to OPWMCB mode should be a channel configured to MCB mode. BSL bits select the time base. The time base must start at $0x000001$ and upward not prior to OPWMCB mode is active. Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base. Register B1 contains the dead-time value and is compared against the internal counter. For a leading edge dead-time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead-time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. Mode[6] bit selects between trailing and leading dead-time insertion, respectively.

NOTE

The internal prescaler of the OPWMCB channel must be set to the same value of the MCB channel prescaler. These prescalers must also be synchronized. In this case, A1 and B1 registers represent the same timing scale for duty cycle and dead-time insertion.

[Figure 26-31](#) describes the load of A1 and B1 registers, which occurs when the selected counter bus reaches the value one. This counter value defines the cycle boundary. Values written to A2 or B2 within cycle (n) are loaded into A1 or B1 registers, respectively, and used to generate matches in cycle (n+1).

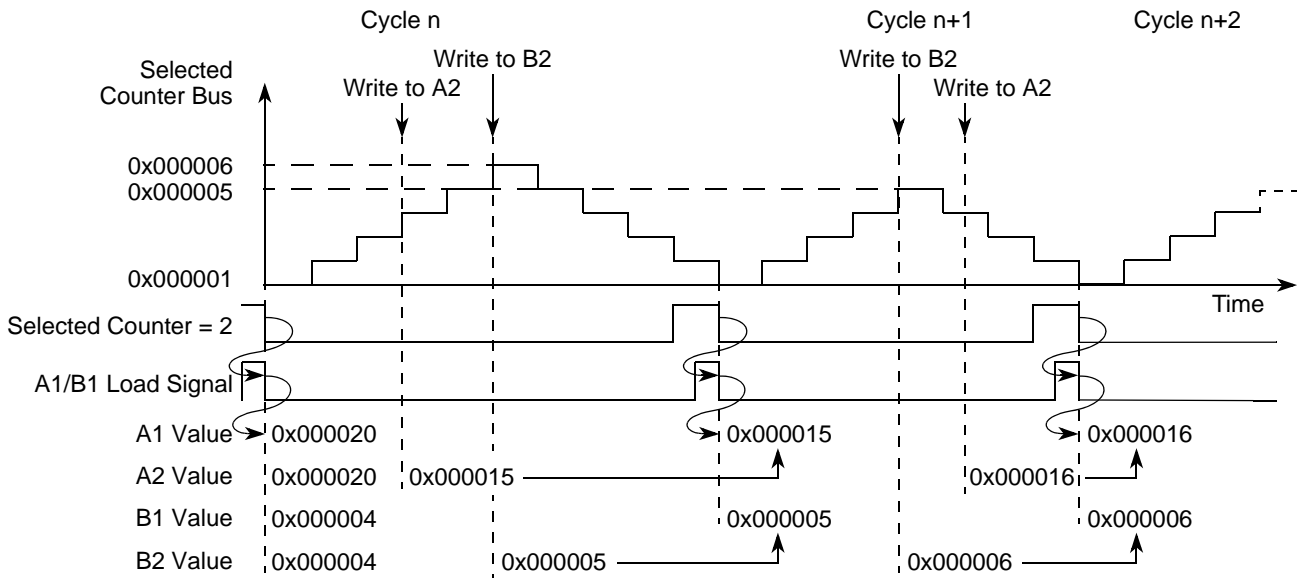


Figure 26-31. OPWMCB A1 and B1 Registers Load

The OUDIS[n] bit can be used to disable the A1 and B1 updates, thus allowing to synchronize the load on these registers with the load of A1 or B1 registers in others channels. Using the update disable bit, A1 and B1 registers can be updated at the same counter cycle, allowing both registers to change at the same time.

In this mode A1 matches always sets the internal counter to 0x000001. When operating with leading edge dead time insertion the first A1 match sets the internal counter to 0x000001. When a match occurs between register B1 and the internal time base, the output flip-flop is set to the value of the EDPOL bit. In the following match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously. The internal counter should not reach 0x0 as consequence of a rollover. To avoid this, the user must not write a value greater than twice the difference between external count up limit and EMIOS_CADR value to the EMIOS_CBDR register.

Figure 26-32 shows two cycles of a center-aligned PWM signal. Both A1 and B1 register values are changing within the same cycle, which allows to vary at the same time the duty cycle and dead-time values.

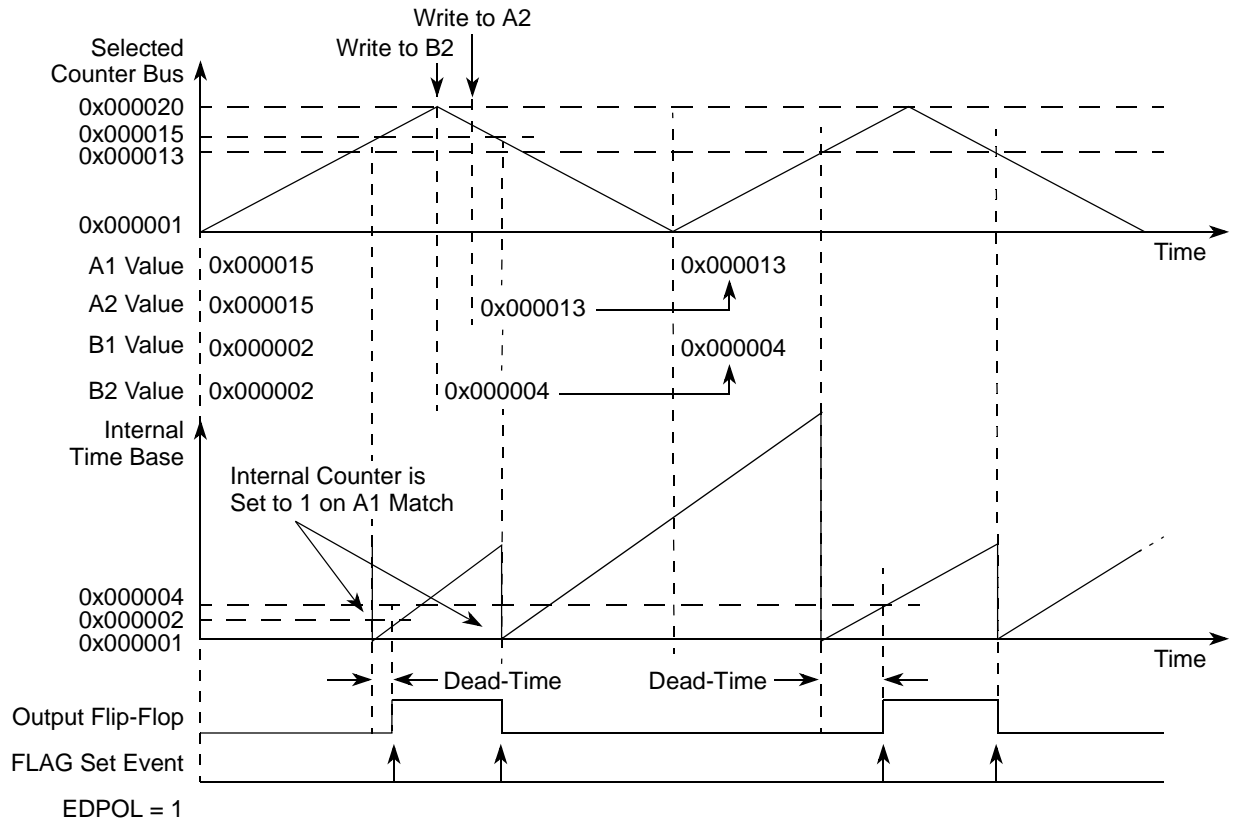


Figure 26-32. Output PWMCB with Lead Dead-Time Insertion

When operating with trailing edge dead-time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit and sets the internal counter to 0x000001. In the second match between register A1 and the selected time base, the internal counter is set to 0x000001 and B1 matches are enabled. When the match between register B1 and the selected time base occurs, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

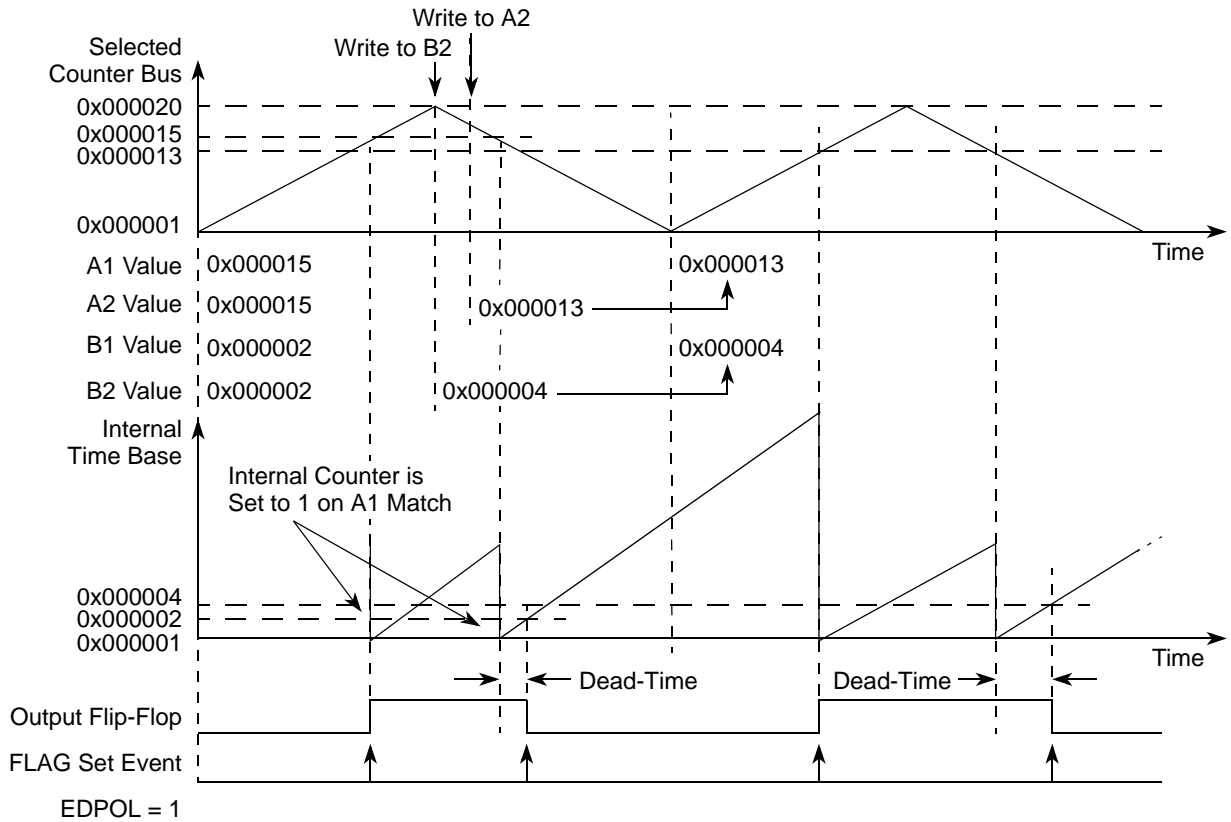


Figure 26-33. Output PWMCB with Trail Dead-Time Insertion

FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or in both edges when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated regardless of the state of the FLAG bit.

NOTE

In OPWMCB mode FORCMA and FORCMB do not have the same behavior as a regular match. Instead they force the output flip-flop to constant value, which depends on the selected dead-time insertion mode, lead or trail and the value of the EDPOL bit.

FORCMA has different behaviors depending upon the selected dead time insertion mode, lead or trail. In lead dead-time insertion FORCMA force a transition in the output flip-flop to the opposite of EDPOL. In trail dead-time insertion the output flip-flop is forced to the value of EDPOL bit.

If FORCMB bit is set, the output flip-flop value depends upon the selected dead-time insertion mode. In lead dead time insertion FORCMB forces the output flip-flop to transition to EDPOL bit value. In trail dead-time insertion the output flip-flop is forced to the opposite of EDPOL bit value.

NOTE

FORCMA bit set does not set the internal time-base to 0x000001 as a regular A1 match.

The FLAG bit is not set either in case of a FORCMA or FORCMB or even if both forces are issued at the same time.

NOTE

FORCMA and FORCMB have the same behavior even in freeze or normal mode regarding the output pin transition.

When FORCMA is issued along with FORCMB, the output flip-flop is set to the opposite of EDPOL bit value. This is equivalent of saying that FORCMA has precedence over FORCMB when lead dead-time insertion is selected and FORCMB has precedence over FORCMA when trail dead-time insertion is selected.

Duty cycle from 0% to 100% can be generated by setting appropriate values to A1 and B1 registers relatively to the period of the external time base. Setting A1=1 generates a 100% duty cycle waveform. If A1 is greater than the maximum value of the selected counter bus period, then a 0% duty cycle is produced. Assuming EDPOL is set to one and OPWMCB mode with trail dead-time insertion, 100% duty cycle signals can be generated if B1 occurs at or after the cycle boundary (external counter = 1).

Only values different than 0x0 are allowed to be written to A1 register. If 0x0 is loaded to A1 the results are unpredictable.

NOTE

A special case occurs when A1 is set to (external counter bus period)/2, which is the maximum value of the external counter. In this case, the output flip-flop is constantly set to the EDPOL bit value.

The internal channel logic prevents matches from one cycle to propagate to the next cycle. In trail dead-time insertion B1 match from cycle (n) could eventually cross the cycle boundary and occur in cycle (n+1). In this case B1 match is masked out and does not cause the output flip-flop to transition. Therefore matches in cycle(n+1) are not affected by the late B1 matches from cycle(n).

Figure 26-34 shows a 100% duty cycle output signal generated by setting A1=4 and B1=3. In this case the trailing edge is positioned at the boundary of cycle n+1, which is actually considered to belong to cycle n+2 and therefore does not cause the output flip-flop to transition.

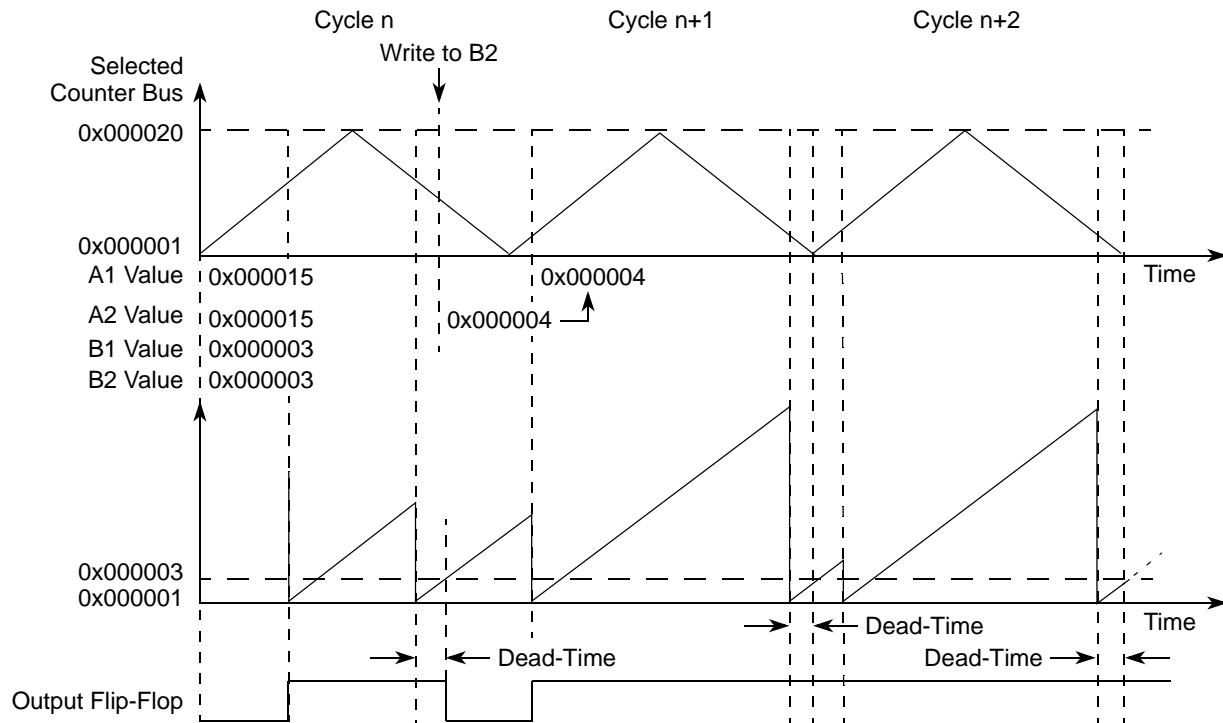


Figure 26-34. OPWMCB with 100% Duty Cycle (A1=4 and B1=3)

The output disable input, if enabled, causes the output flip-flop to transition to EDPOL inverted. This feature allows the channel to force an output pin to a safety state from the application stand point. The internal channel matches continue to occur even in this case, thus generating flags. As soon as the output disable is deasserted, the channel output pin is again controlled by the A1 and B1 matches. This process is synchronous, meaning that the output channel pin transitions on system clock edges only.

It is important to notice that, as in OPWMB and OPWFMB modes, the match signal used to set or clear the channel output flip-flop is generated on the deassertion of the channel combinational comparator output signal which compares the selected time base with A1 or B1 register values. Refer to Figure 26-26 which describes the delay from matches to output flip-flop transition in OPWFMB mode. The operation of OPWMCB mode is similar to OPWFMB regarding matches and output pin transition.

26.5.1.1.10 Pulse-Width Modulation Buffered (OPWMB) Mode

OPWMB mode is used to generate pulses with programmable leading- and trailing-edge placement. An external counter must be selected from one of the counter buses. The A1 register value defines the first edge and B1 defines the second edge. The output signal polarity is defined by the EDPOL bit. If EDPOL is zero, a negative edge occurs when A1 matches the selected counter bus; and a positive edge occurs when B1 matches the selected counter bus.

The A1 and B1 registers are double buffered and updated from A2 and B2, respectively, at the cycle boundary. The load operation is similar to the OPWFMB mode. Refer to Figure 26-28 for more information about A1 and B1 registers' update.

FLAG can be generated at B1 matches, when MODE[5] is cleared, or on either A1 or B1 matches when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A1 or B1. The FLAG bit is not set by the FORCMA and FORCMB operations.

Some rules applicable to the OPWMB mode include:

- B1 matches have precedence over A1 matches if they occur at the same time within the same counter cycle
- A1=0 match from cycle(n) has precedence over B1 match from cycle(n-1)
- A1 matches are masked out if they occur after B1 match within the same cycle
- Any value written to A2 or B2 on cycle(n) is loaded to A1 and B1 registers at the following cycle boundary (assuming OUDIS[n] is not asserted). The new values will be used for A1 and B1 matches in cycle(n+1)

Figure 26-35 describes the operation of the OPWMB mode regarding A1 and B1 matches and the transition of the channel output pin. In this example EDPOL is set to zero.

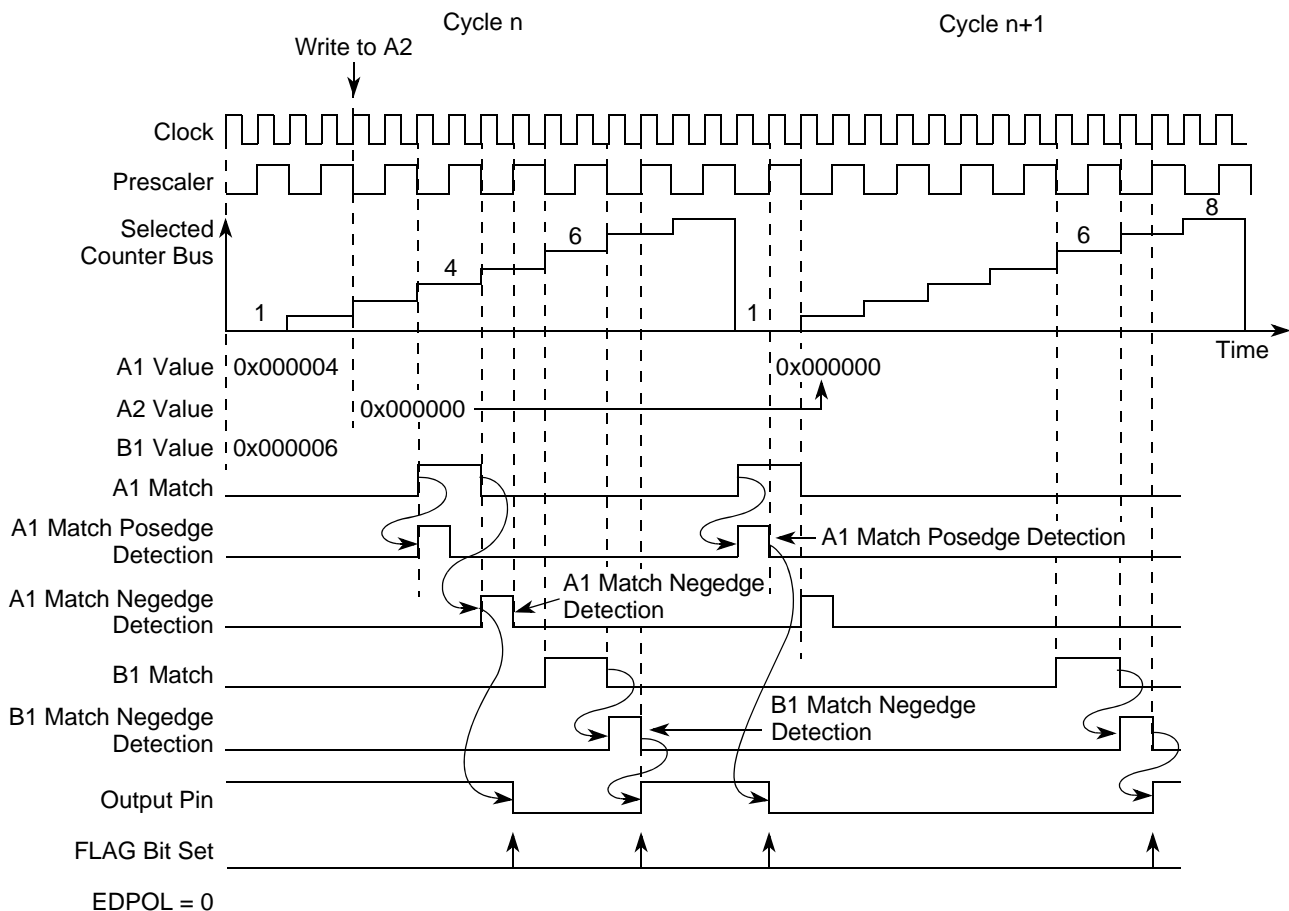


Figure 26-35. OPWMB Mode Matches and Flags

The output pin transitions are based on the negedges of the A1 and B1 match signals. Figure 26-35 shows in cycle(n+1) the value of the A1 register being set to zero. In this case, the match posedge is used instead of the negedge to transition the output flip-flop.

Figure 26-36 describes the channel operation for 0% duty cycle. Note that the A1 match posedge signal occurs at the same time as the B1=8 negedge signal. In this case A1 match has precedence over B1 match, causing the output pin to remain at EDPOL bit value, thus generating a 0% duty cycle signal.

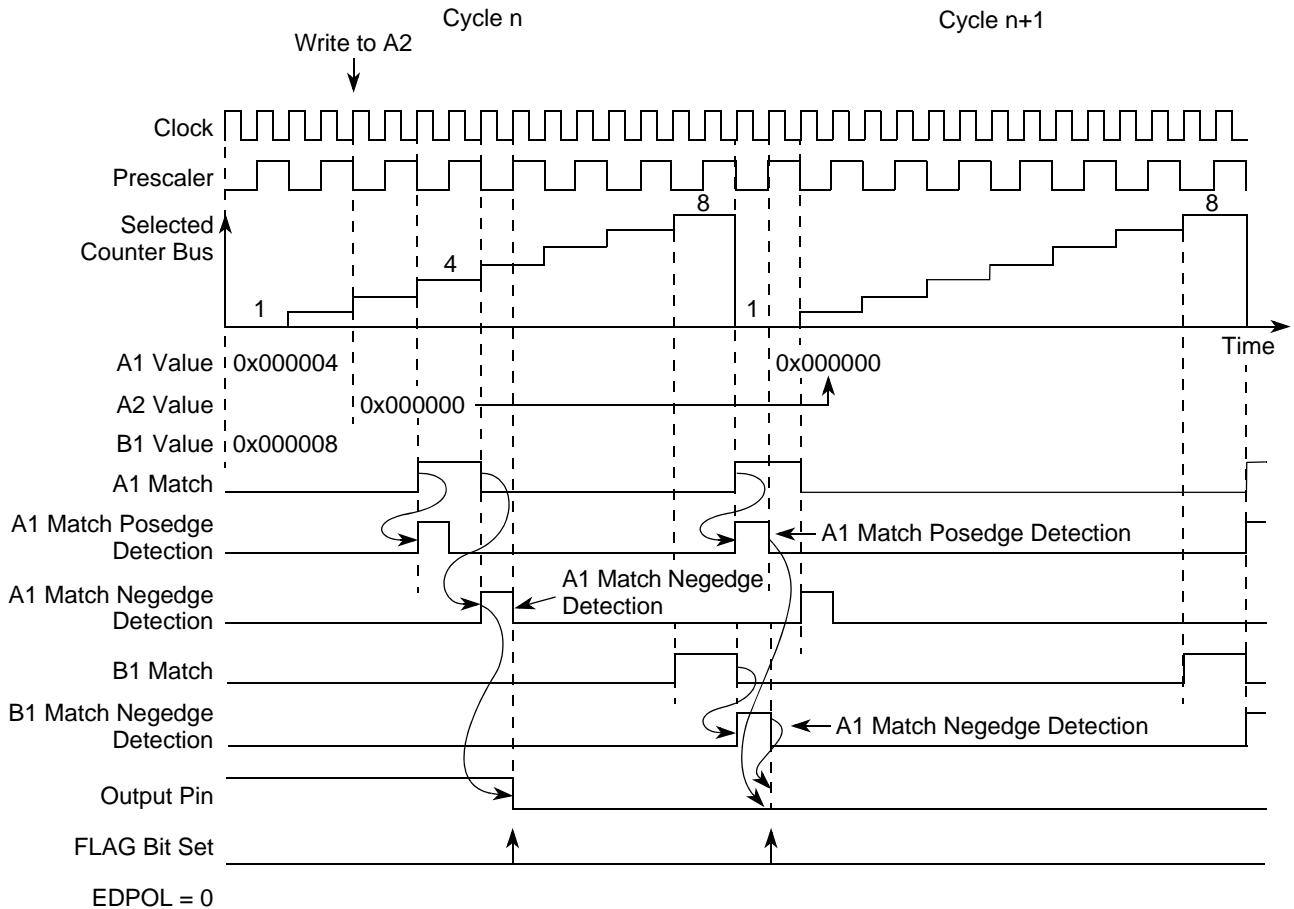


Figure 26-36. OPWMB Mode with 0% Duty Cycle

Figure 26-37 describes the operation of the OPWMB mode with the output disable signal asserted. The output disable forces a transition in the output pin to the EDPOL bit value. After deassertion, the output disable allows the output pin to transition at the following A1 or B1 match. The output disable does not modify the flag bit behavior. There is one system clock delay between the assertion of the output disable signal and the transition of the output pin to EDPOL.

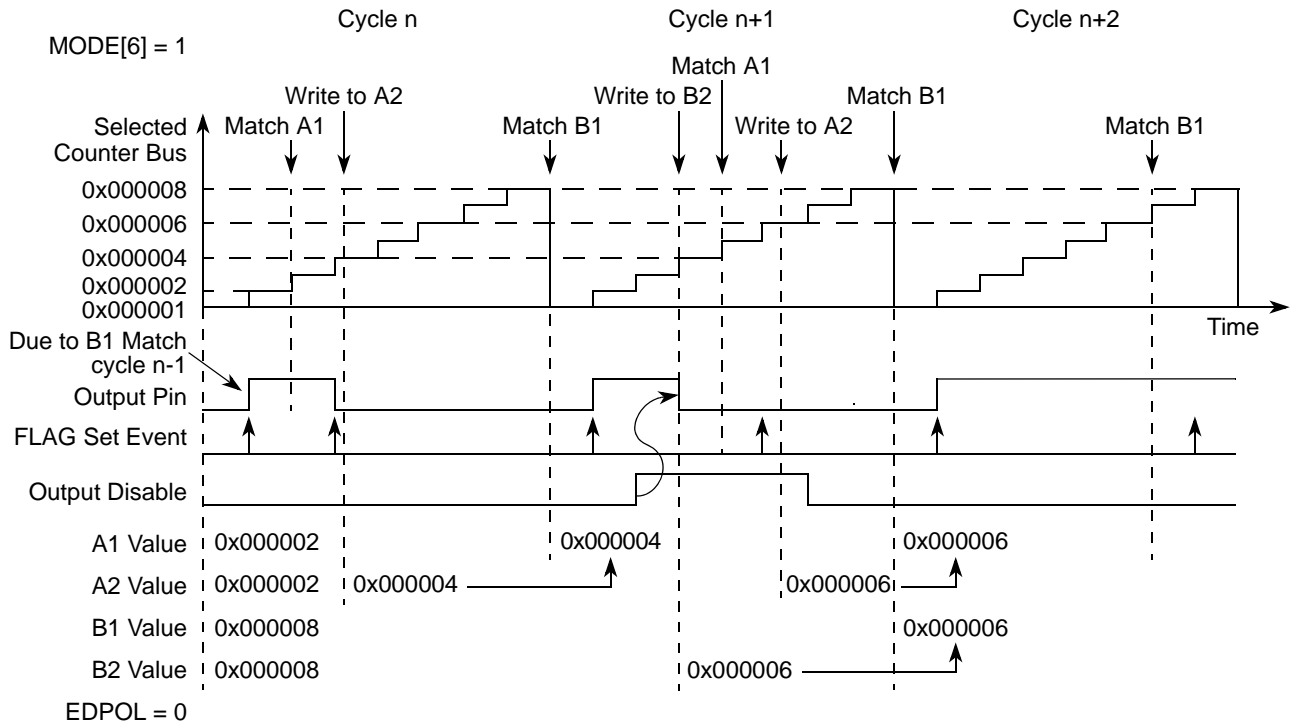


Figure 26-37. OPWMB Mode with Active Output Disable

Figure 26-38 shows a waveform changing from 100% to 0% duty cycle. In this case, EDPOL is zero. In this example, B1 is programmed to the same value as the period of the external selected time base.

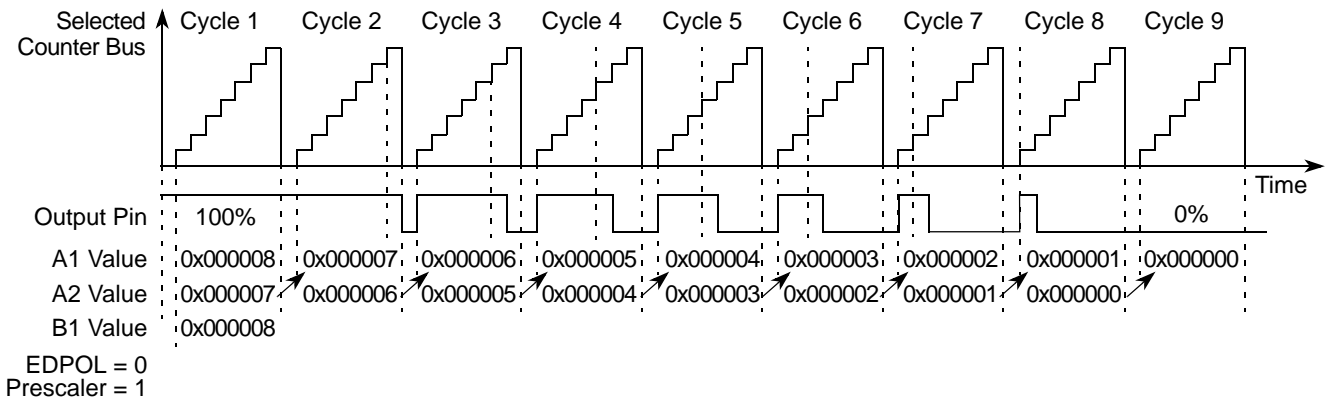


Figure 26-38. OPWMB Mode from 100% to 0% Duty Cycle

In Figure 26-38, if B1 is set to a value lower than 0x000008, it is not possible to achieve 0% duty cycle by changing only A1 register value. Because B1 matches have precedence over A1 matches the output pin transitions to the opposite of EDPOL bit at B1 match. If B1 is set to 0x000009, for instance, B1 match does not occur, thus a 0% duty cycle signal is generated.

26.5.1.2 Input Programmable Filter (IPF)

The IPF ensures that only valid input pin transitions are received by the unified channel edge detector. A block diagram of the IPF is shown in [Figure 26-39](#).

The IPF is a 5-bit programmable up counter that is incremented by the selected clock source, according to bits IF in EMIOS_CCR[n] register.

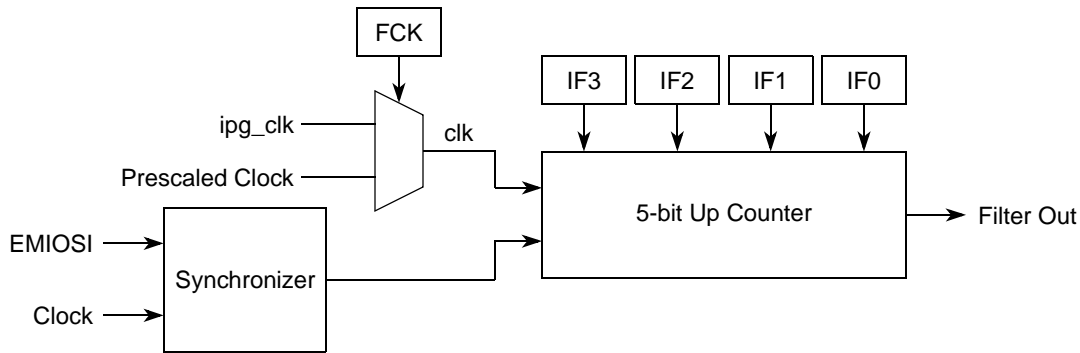


Figure 26-39. Input Programmable Filter Submodule Diagram

The input signal is synchronized by system clock. When a state change occurs in this signal, the 5-bit counter starts counting up. As long as the new state is stable on the pin, the counter remains incrementing. If a counter overflow occurs, the new pin value is validated. In this case, it is transmitted as a pulse edge to the edge detector. If the opposite edge appears on the pin before validation (overflow), the counter is reset. At the next pin transition, the counter starts counting again. Any pulse that is shorter than a full range of the masked counter is regarded as a glitch and it is not passed on to the edge detector. [Figure 26-40](#) shows a timing diagram of the input filter.

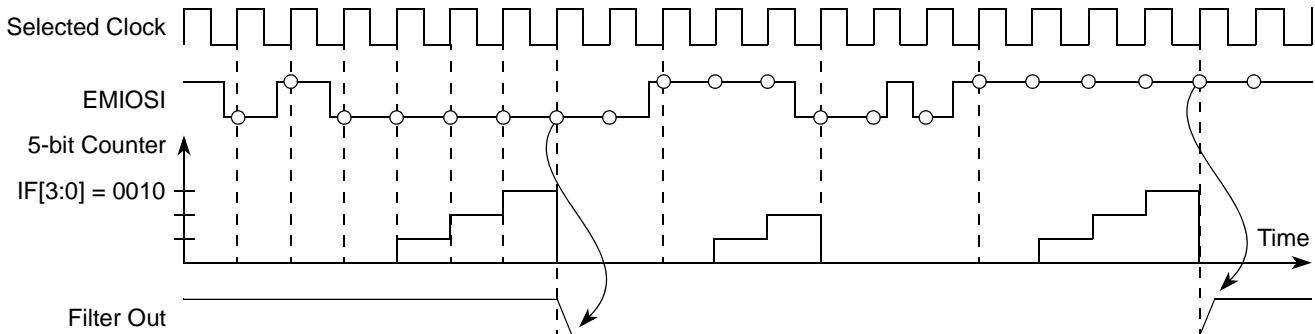


Figure 26-40. Input Programmable Filter Example

26.5.1.3 Clock Prescaler (CP)

The CP divides the GCP output signal to generate a clock enable for the internal counter of the unified channels. It is a programmable 2-bit down counter. The GCP output signal is prescaled by the value defined in the UCPRE bits in EMIOS_CCR[n] register. The output is clocked every time the counter reaches zero. Counting is enabled by setting the UCPREN bit in the EMIOS_CCR[n]. The counter can be stopped at any time by clearing this bit, thereby stopping the internal counter in the unified channel.

26.5.1.4 Effect of Freeze on the Unified Channel

When in debug mode, if the FRZ bit in the EMIOS_MCR register and the FREN bit in the EMIOS_CCR[n] are both set, the internal counter and unified channel capture and compare functions are halted. The unified channel is frozen in its current state.

During freeze, all registers are accessible. When the unified channel is operating in an output mode, the force match functions remain available, allowing the software to force the output to the desired level.

During input modes, any input events that may occur while the channel is frozen are ignored.

When exiting debug mode or when the freeze enable bit is cleared (FRZ in the EMIOS_MCR or FREN in the EMIOS_CCR[n] register), the channel actions resume but may be inconsistent until the channel enters GPIO mode again.

26.5.2 IP Bus Interface Unit (BIU)

The BIU provides the interface between the internal interface bus (IIB) and the peripheral bus, allowing communication among all submodules and this IP interface.

The BIU allows 8-, 16-, and 32-bit access. They are performed over a 32-bit data bus in a single cycle clock.

26.5.2.1 Effect of Freeze on the BIU

When the FRZ bit in the EMIOS_MCR register is set and the module is in debug mode, the operation of BIU is not affected.

26.5.3 Global Clock Prescaler Submodule (GCP)

The GCP divides the system clock to generate a clock for the CPs of the unified channels. It is a programmable 8-bit up counter. The main clock signal is prescaled by the value defined in the GPRE bits in EMIOS_MCR. The output is clocked every time the counter overflows. Counting is enabled by setting the GPREN bit in the EMIOS_MCR. The counter can be stopped at any time by clearing this bit, thereby stopping the internal counter in all the unified channels.

26.5.3.1 Effect of Freeze on the GCP

When the FRZ bit in the EMIOS_MCR register is set and the module is in debug mode, the operation of GCP submodule is not affected, i.e., there is no freeze function in this submodule.

26.6 Reset

The eMIOS200 is reset by the global asynchronous system reset signal.

The MDIS bit in the EMIOS_MCR register and the UCDIS bits in the EMIOSUCDIS registers are cleared during reset.

On resetting the eMIOS200 all unified channels enter GPIO input mode.

26.7 Interrupts

The eMIOS200 can generate one interrupt per channel. An interrupt request is generated according to the configuration of the channel and input events or matches. See [Section 8.3.1, “Interrupt Source Summary Table,”](#) for details on the eMIOS200 interrupt vectors.

26.8 DMA Requests

The connection of the eMIOS200 DMA request signals to the DMA channel mux is described in [Section 13.5.2, “Enabling and Configuring Sources.”](#)

26.9 Initialization/Application Information

On resetting the eMIOS200 all unified channels enter GPIO input mode.

26.9.1 Considerations

Before changing an operating mode, the unified channel must be programmed to GPIO mode and EMIOS_CADR[n] and EMIOS_CBDR[n] registers must be updated with the correct values for the next operating mode. Then the EMIOS_CCR[n] register can be written with the new operating mode. If a unified channel is changed from one mode to another without performing this procedure, the first operation cycle of the selected time base can be random, i.e., matches can occur in random time if the contents of EMIOS_CADR[n] or EMIOS_CBDR[n] were not updated with the correct value before the time base matches the previous contents of EMIOS_CADR[n] or EMIOS_CBDR[n].

When interrupts are enabled, the software must clear the FLAG bits before exiting the interrupt service routine.

26.9.2 Application Information

Correlated output signals can be generated by all output operation modes. Bits OUDIS[n] can be used to control the update of these output signals.

In order to guarantee the internal counters of correlated channels are incremented in the same clock cycle, the internal prescalers must be set before enabling the global prescaler. If the internal prescalers are set after enabling the global prescaler, the internal counters may increment in the same ratio but at a different clock cycle.

It is recommended to drive output disable input signals with the emios_flag_out signals of some unified channels running in SAIC mode. When an output disable condition happens, the software interrupt routine must service the output channels before servicing the channels running SAIC. This procedure avoids glitches in the output pins.

26.9.3 Coherent Accesses

For IPWM and IPM modes, it is recommended that the software wait for a new FLAG set event before reading EMIOS_CADR[n] and EMIOS_CBDR[n] registers to get a new measurement. The FLAG indicates that new data has been captured and it is the only way to assure data coherency.

The FLAG set event can be detected by polling the FLAG bit or by enabling the interrupt or DMA request generation.

Reading the EMIOS_CADR[n] register again in the same period of the last read of EMIOS_CBDR[n] register may lead to incoherent results. This will occur if the last read of EMIOS_CBDR[n] register occurred after a disabled B2 to B1 transfer.

Chapter 27

Inter-Integrated Circuit Bus Controller Module (I²C)

27.1 Introduction

The inter-integrated circuit (I²C™) bus is a two-wire bidirectional serial bus that provides a simple and efficient method of data exchange between devices. It minimizes the external connections to devices and does not require an external address decoder.

This bus is suitable for applications requiring occasional communication over a short distance between a number of devices. It also provides flexibility, allowing additional devices to be connected to the bus for further expansion and system development.

The interface is designed to operate up to 100 kbps with maximum bus loading and timing. The device is capable of operating at higher baud rates, up to a maximum of module clock/20, with reduced bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF.

27.1.1 Block Diagram

A simplified block diagram of the I²C illustrates the functionality and interdependence of major blocks (see [Figure 27-1](#)).

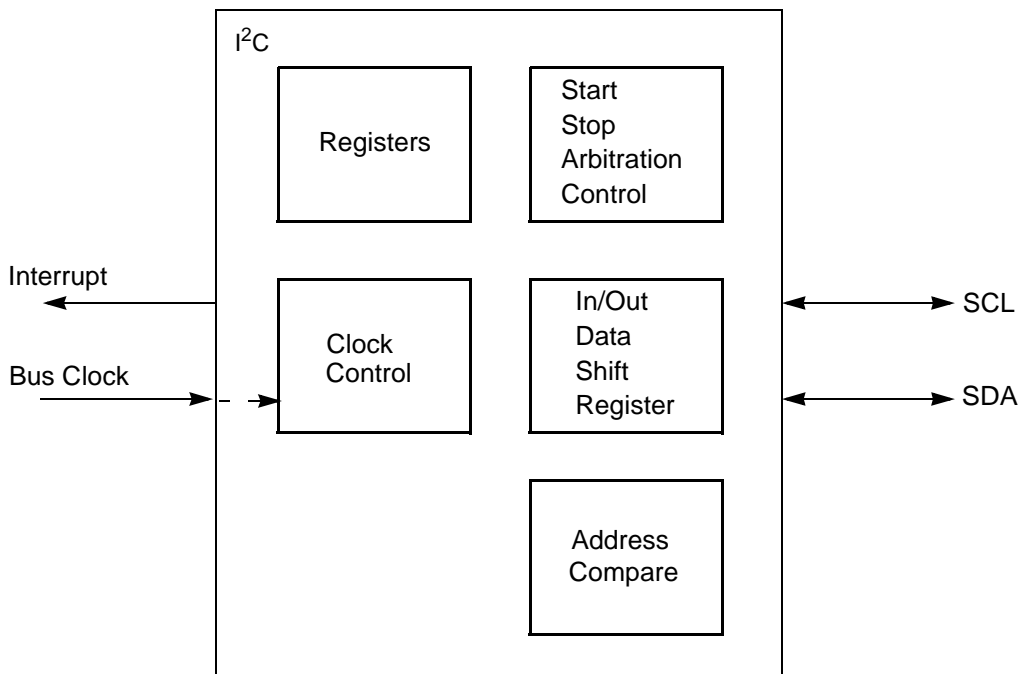


Figure 27-1. I²C Block Diagram

27.1.2 DMA Interface

A simple DMA interface is implemented so that the I²C can request data transfers with minimal support from the CPU. DMA mode is enabled by setting bit 1 in the control register.

The DMA interface is only valid when the I²C module is configured for master mode and the DMA channel mux has selected the I²C DMA request signals to be inputs to a DMA channel.

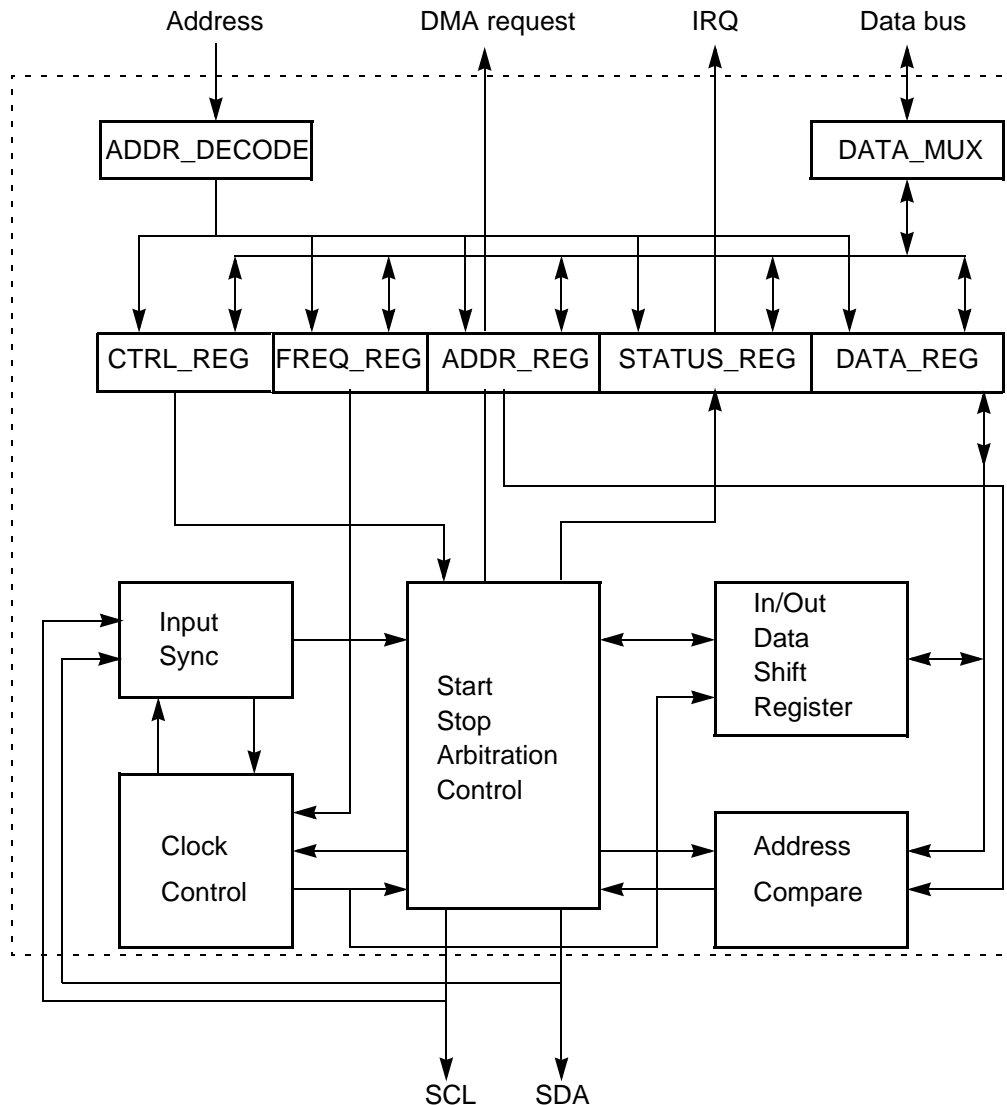


Figure 27-2. I²C Module DMA Interface Block Diagram

27.1.3 Features

The I²C has these major features:

- Compatible with I²C bus standard
- Multi-master operation
- Software programmable for one of 256 serial clock frequencies
- Software selectable acknowledge bit
- Interrupt driven byte-by-byte data transfer
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection

- Repeated start signal generation
- Acknowledge bit generation/detection
- Bus busy detection
- Basic DMA interface

Features currently not supported:

- No support for general call address
- Not compliant to ten-bit addressing

27.1.4 Modes of Operation

There are two operating modes of the I²C module: run mode and stop mode. In run mode, I²C_A = 0 in the SIU_HLT register and all functional parts of the I²C module are running. In stop mode, I²C_A = 1 in the SIU_HLT register and all clocks to the I²C module are disabled.

27.2 External Signal Description

Refer to [Table 2-1](#) and [Section 2.7, “Detailed External Signal Descriptions,”](#) for detailed signal descriptions.

27.3 Memory Map and Registers

This section provides a detailed description of all I²C registers.

27.3.1 Module Memory Map

[Table 27-1](#) shows the I²C memory map. The address of each register is given as an offset to the I²C base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed. There are no MPC5510-specific register definitions for the I²C module.

Table 27-1. I²C Memory Map

Offset from I ² C_BASE (0xFFFF8_8000)	Register	Access	Reset Value	Section/Page
0x0000	IBAD — I ² C bus address register	R/W	0x0000	27.3.2.1/27-5
0x0001	IBFD — I ² C bus frequency divider register	R/W	0x0000	27.3.2.2/27-5
0x0002	IBCR — I ² C bus control register	R/W	0x0080	27.3.2.3/27-8
0x0003	IBSR — I ² C bus status register	R/W	0x0080	27.3.2.4/27-9
0x0004	IBDR — I ² C bus data I/O register	R/W	0x0000	27.3.2.5/27-10
0x0005	IBIC — I ² C bus interrupt config register	R/W	0x0000	27.3.2.6/27-11
0x0006–0x3FFF	Reserved	R	0x0000	N/A

27.3.2 Register Descriptions

This section lists the I²C registers in address order and describes the registers and their bit fields.

27.3.2.1 I²C Bus Address Register (IBAD)

This register contains the address the I²C bus will respond to when addressed as a slave; it is not the address sent on the bus during the address transfer.

Offset: 0x00000

Access: Read/write any time

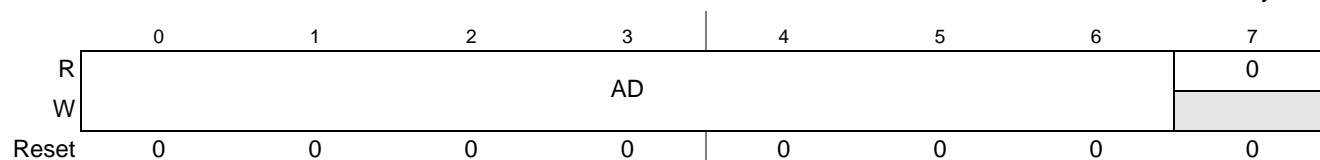


Figure 27-3. I²C Bus Address Register (IBAD)

Table 27-2. IBAD Field Descriptions

Field	Description
0–6 AD	Slave Address. Specific slave address to be used by the I ² C bus module. Note: The default mode of I ² C bus is slave mode for an address match on the bus.
7	Reserved, must be cleared; will always read 0.

27.3.2.2 I²C Bus Frequency Divider Register (IBFD)

Offset: 0x0001

Access: Read/write any time



Figure 27-4. I²C Bus Frequency Divider Register (IBFD)

Table 27-3. IBFD Field Descriptions

Field	Description
MULT	<p>I²C Multiplier Factor. The MULT bits define the multiplier factor mul. This factor is used along with the SCL divider to generate the I²C baud rate. The multiplier factor mul as defined by the MULT bits is provided below.</p> <p>00 mul = 1 01 mul = 2 10 mul = 4 11 Reserved</p>
ICR	<p>I²C Bus Clock Rate. The ICR bits are used to prescale the bus clock for bit rate selection. These bits and the MULT bits are used to determine the I²C baud rate, the SDA hold time, the SCL Start hold time and the SCL Stop hold time. Table 27-4 provides the SCL divider and hold values for corresponding values of the ICR.</p> <p>The SCL divider multiplied by multiplier factor mul is used to generate I²C baud rate.</p> $\text{I}^2\text{C baud rate} = \text{bus speed (Hz)} / (\text{mul} * \text{SCL divider}) \quad \text{Eqn. 27-1}$ <p>SDA hold time is the delay from the falling edge of SDA (I²C data) to the changing of SDA (I²C data).</p> $\text{SDA hold time} = \text{bus period (s)} * \text{mul} * \text{SDA hold value} \quad \text{Eqn. 27-2}$ <p>SCL Start hold time is the delay from the falling edge of SDA (I²C data) while SCL is high (Start condition) to the falling edge of SCL (I²C clock).</p> $\text{SCL Start hold time} = \text{bus period (s)} * \text{mul} * \text{SCL Start hold value} \quad \text{Eqn. 27-3}$ <p>SCL Stop hold time is the delay from the rising edge of SCL (I²C clock) to the rising edge of SDA (I²C data) while SCL is high (Stop condition).</p> $\text{SCL Stop hold time} = \text{bus period (s)} * \text{mul} * \text{SCL Stop hold value} \quad \text{Eqn. 27-4}$

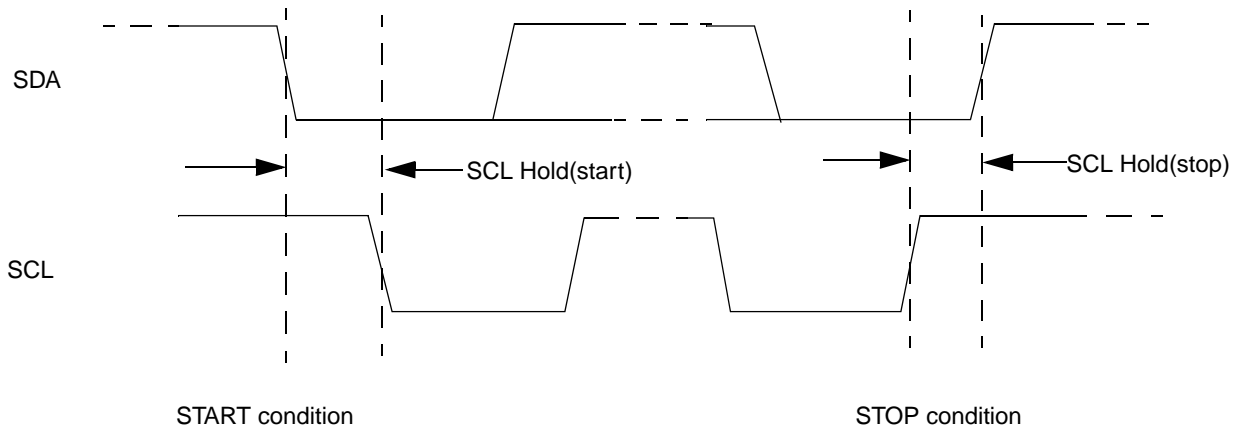


Figure 27-5. SCL Divider and SDA Hold

Table 27-4. I²C Divider and Hold Values

ICR (hex)	SCL Divider	SDA Hold Value	SCL Hold (Start) Value	SDA Hold (Stop) Value
00	20	7	6	11
01	22	7	7	12
02	24	8	8	13
03	26	8	9	14
04	28	9	10	15
05	30	9	11	16
06	34	10	13	18
07	40	10	16	21
08	28	7	10	15
09	32	7	12	17
0A	36	9	14	19
0B	40	9	16	21
0C	44	11	18	23
0D	48	11	20	25
0E	56	13	24	29
0F	68	13	30	35
10	48	9	18	25
11	56	9	22	29
12	64	13	26	33
13	72	13	30	37
14	80	17	34	41
15	88	17	38	45
16	104	21	46	53
17	128	21	58	65
18	80	9	38	41
19	96	9	46	49
1A	112	17	54	57
1B	128	17	62	65
1C	144	25	70	73
1D	160	25	78	81
1E	192	33	94	97
1F	240	33	118	121

ICR (hex)	SCL Divider	SDA Hold Value	SCL Hold (Start) Value	SCL Hold (Stop) Value
20	160	17	78	81
21	192	17	94	97
22	224	33	110	113
23	256	33	126	129
24	288	49	142	145
25	320	49	158	161
26	384	65	190	193
27	480	65	238	241
28	320	33	158	161
29	384	33	190	193
2A	448	65	222	225
2B	512	65	254	257
2C	576	97	286	289
2D	640	97	318	321
2E	768	129	382	385
2F	960	129	478	481
30	640	65	318	321
31	768	65	382	385
32	896	129	446	449
33	1024	129	510	513
34	1152	193	574	577
35	1280	193	638	641
36	1536	257	766	769
37	1920	257	958	961
38	1280	129	638	641
39	1536	129	766	769
3A	1792	257	894	897
3B	2048	257	1022	1025
3C	2304	385	1150	1153
3D	2560	385	1278	1281
3E	3072	513	1534	1537
3F	3840	513	1918	1921

27.3.2.3 I²C Bus Control Register (IBCR)

Offset: 0x0002

Access: Read/write any time

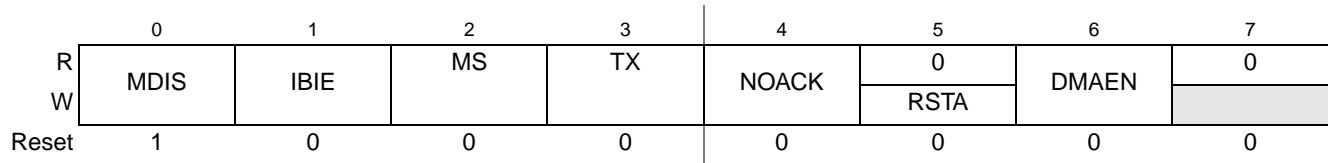


Figure 27-6. I²C Bus Control Register (IBCR)

Table 27-5. IBCR Field Descriptions

Field	Description
MDIS	<p>Module Disable. This bit controls the software reset of the entire I²C bus module.</p> <p>0 The I²C bus module is enabled. This bit must be cleared before any other IBCR bits have any effect.</p> <p>1 The module is reset and disabled. This is the power-on reset situation. When high, the interface is held in reset, but registers can be accessed.</p> <p>Note: If the I²C bus module is enabled in the middle of a byte transfer, the interface behaves as follows: slave mode ignores the current transfer on the bus and starts operating when a subsequent start condition is detected. Master mode will not be aware that the bus is busy; therefore, if a start cycle is initiated, then the current bus cycle may become corrupt. This ultimately results in the current bus master or the I²C bus module losing arbitration, after which, bus operation returns to normal.</p>
IBIE	<p>I-Bus Interrupt Enable.</p> <p>0 Interrupts from the I²C bus module are disabled. This does not clear any currently pending interrupt condition.</p> <p>1 Interrupts from the I²C bus module are enabled. An I²C bus interrupt occurs provided the IBIF bit in the status register is also set.</p>
MS	<p>Master/Slave Mode Select. This bit is cleared on reset. When this bit is changed from 0 to 1, a START signal is generated on the bus and the master mode is selected. When this bit is changed from 1 to 0, a STOP signal is generated and the operation mode changes from master to slave. A STOP signal should be generated if only the IBIF flag is set. MS is cleared without generating a STOP signal when the master loses arbitration.</p> <p>0 Slave mode.</p> <p>1 Master mode.</p>
TX	<p>Transmit/Receive Mode Select. This bit selects the direction of master and slave transfers. When addressed as a slave this bit must be set by software according to the SRW bit in the status register. In master mode this bit must be set according to the type of transfer required. Therefore, for address cycles, this bit is always high.</p> <p>0 Receive.</p> <p>1 Transmit.</p>
NOACK	<p>Data Acknowledge Disable. This bit specifies the value driven onto SDA during data acknowledge cycles for both master and slave receivers. The I²C module will always acknowledge address matches, provided it is enabled, regardless of the value of NOACK. Values written to this bit are used only when the I²C Bus is a receiver, not a transmitter.</p> <p>0 An acknowledge signal will be sent out to the bus at the 9th clock bit after receiving one byte of data.</p> <p>1 No acknowledge signal response is sent (i.e., acknowledge bit = 1).</p>
RSTA	<p>Repeat Start. Writing a 1 to this bit generates a repeated START condition on the bus, provided it is the current bus master. This bit is always read as a low. Attempting a repeated start at the wrong time, if the bus is owned by another master, results in loss of arbitration.</p> <p>0 No effect.</p> <p>1 Generate repeat start cycle.</p>

Table 27-5. IBCR Field Descriptions (continued)

Field	Description
DMAEN	DMA enable. When this bit is set, the DMA TX and RX lines are asserted when the I ² C module requires data to be read or written to the data register. No transfer done interrupts are generated when this bit is set; however, an interrupt is generated if loss of arbitration or addressed as slave conditions occur. The DMA mode is valid only when the I ² C module is configured as a master and the DMA transfer still requires CPU intervention at the start and the end of each frame of data. See the DMA application information section for more details. 0 Disable the DMA TX/RX request signals. 1 Enable the DMA TX/RX request signals.
bit 7	Reserved. Writes have no functional effect but will change the value when reading the register.

27.3.2.4 I²C Bus Status Register (IBSR)

Offset: 0x0003

Access: Read-only any time¹

	0	1	2	3	4	5	6	7
R	TCF	IAAS	IBB	IBAL	0	SRW	IBIF	RXAK
W							w1c	
Reset	1	0	0	0	0	0	0	0

Figure 27-7. I²C Bus Status Register (IBSR)¹ With the exception of IBIF and IBAL, which are software clearable.

Table 27-6. IBSR Field Descriptions

Field	Description
TCF	Transfer Complete. While one byte of data is transferred, this bit is cleared. It is set by the falling edge of the ninth clock of a byte transfer. This bit is valid only during or immediately following a transfer to the I ² C module or from the I ² C module. 0 Transfer in progress. 1 Transfer complete.
IAAS	Addressed as a Slave. When its own specific address (I-bus address register) is matched with the calling address, this bit is set. The CPU is interrupted provided the IBIE is set. Then the CPU must check the SRW bit and set its Tx/Rx mode accordingly. Writing to the I-bus control register clears this bit. 0 Not addressed. 1 Addressed as a slave.
IBB	Bus Busy. This bit indicates the status of the bus. When a START signal is detected, the IBB is set. If a STOP signal is detected, IBB is cleared and the bus enters idle state. 0 Bus is idle. 1 Bus is busy.
IBAL	Arbitration Lost. The arbitration lost bit (IBAL) is set by hardware when the arbitration procedure is lost. Arbitration is lost in the following circumstances: <ul style="list-style-type: none"> • SDA is sampled low when the master drives a high during an address or data transmit cycle. • SDA is sampled low when the master drives a high during the acknowledge bit of a data receive cycle. • A start cycle is attempted when the bus is busy. • A repeated start cycle is requested in slave mode. • A stop condition is detected when the master did not request it. This bit must be cleared by software, by writing a one to it. A write of zero has no effect.

Table 27-6. IBSR Field Descriptions (continued)

Field	Description
bit 4	Reserved for future use. A read will return 0; must be written as 0.
SRW	Slave Read/Write. When IAAS is set, this bit indicates the value of the R/W command bit of the calling address sent from the master. This bit is valid only when the I-bus is in slave mode, a complete address transfer has occurred with an address match and no other transfers have been initiated. By programming this bit, the CPU can select slave transmit/receive mode according to the command of the master. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.
IBIF	I-Bus Interrupt Flag. The IBIF bit is set when one of the following conditions occurs: <ul style="list-style-type: none"> • Arbitration lost (IBAL bit set) • Byte transfer complete (TCF bit set and DMAEN bit not set) • Addressed as slave (IAAS bit set) • NoAck from slave (MS & TX bits set) • I²C bus going idle (IBB high-low transition and enabled by BIIE) A processor interrupt request will be caused if the IBIE bit is set. This bit must be cleared by software, by writing a 1 to it. A write of 0 has no effect on this bit. In DMA mode (DMAEN set), a byte transfer complete condition will not trigger the setting of IBIF. All other conditions apply.
RXAK	Received Acknowledge. This is the value of SDA during the acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates an acknowledge signal has been received after the completion of 8 bits data transmission on the bus. If RXAK is high, it means no acknowledge signal is detected at the 9th clock. 0 Acknowledge received. 1 No acknowledge received.

27.3.2.5 I²C Bus Data I/O Register (IBDR)

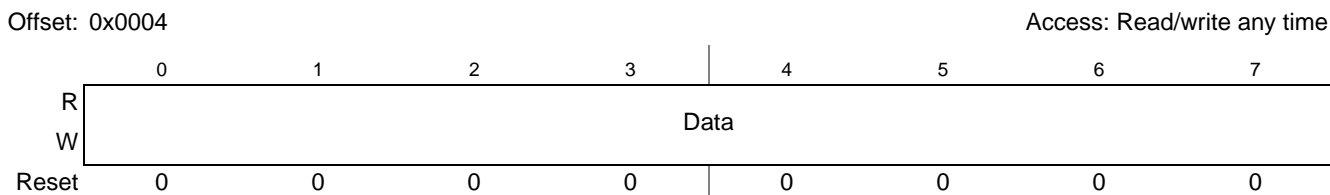


Figure 27-8. I²C Bus Data I/O Register (IBDR)

In master transmit mode, when data is written to IBDR, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates next byte data receiving. In slave mode, the same functions are available after an address match has occurred. The TX bit in the IBCR must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For instance, if the I²C is configured for master transmit but a master receive is desired, then reading the IBDR will not initiate the receive.

Reading the IBDR will return the last byte received while the I²C is configured in either master receive or slave receive modes. The IBDR does not reflect every byte that is transmitted on the I²C bus, nor can software verify that a byte has been written to the IBDR correctly by reading it back.

In master-transmit mode, the first byte of data written to IBDR following assertion of MS is used for the address transfer and should comprise the calling address (in position D0–D6) concatenated with the required R/W bit (in position D7).

27.3.2.6 I²C Bus Interrupt Config Register (IBIC)

Offset: 0x0005

Access: Read/write any time

	0	1	2	3	4	5	6	7
R	BIIE	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

Figure 27-9. I²C Bus Interrupt Config Register (IBIC)

Table 27-7. IBIC Field Descriptions

Field	Description
BIIE	Bus Idle Interrupt Enable Bit. This config bit can be used to enable the generation of an interrupt after the I ² C bus becomes idle. After this bit is set, an IBB high-low transition sets the IBIF bit. This feature can be used to signal to the CPU the completion of a STOP on the I ² C bus. 0 Bus idle interrupts disabled. 1 Bus idle interrupts enabled.
bits 1–7	Reserved for future use. A read will return 0; must be written as 0.

27.4 Functional Description

27.4.1 I-Bus Protocol

The I²C bus system uses a serial data line (SDA) and a serial clock line (SCL) for data transfer. All devices connected to it must have open-drain or open-collector outputs. A logical AND function is exercised on both lines with external pullup resistors. The value of these resistors is system dependent.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer, and STOP signal. They are described briefly in the following sections and illustrated in [Figure 27-10](#).

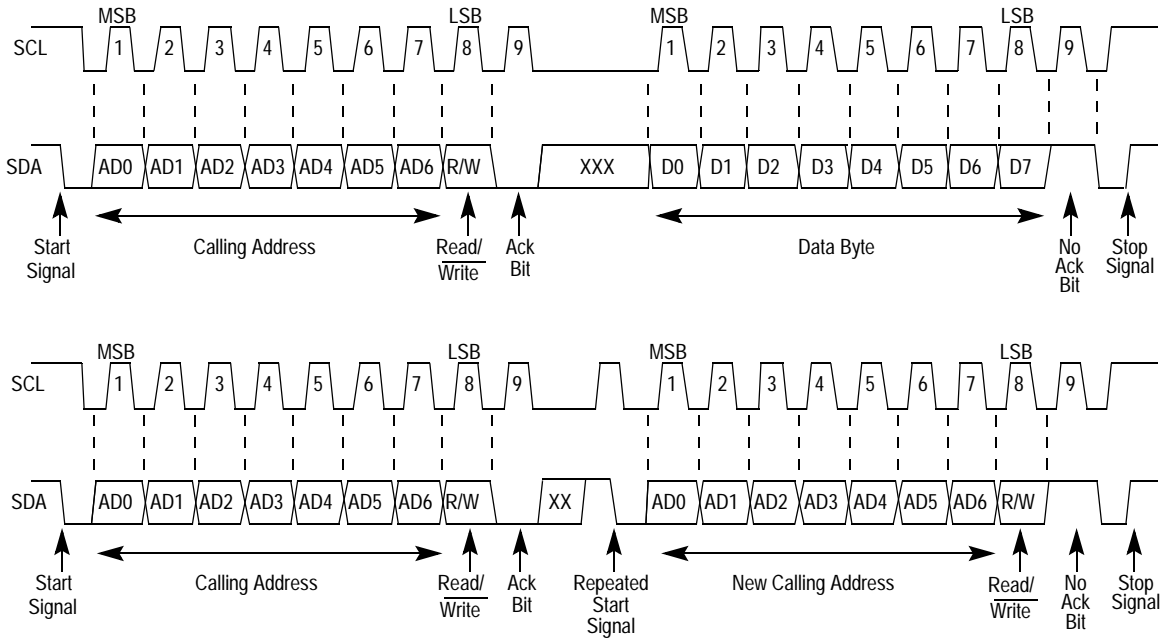


Figure 27-10. I²C Bus Transmission Signals

27.4.1.1 START Signal

When the bus is free, i.e. no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in Figure 27-10, a START signal is a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

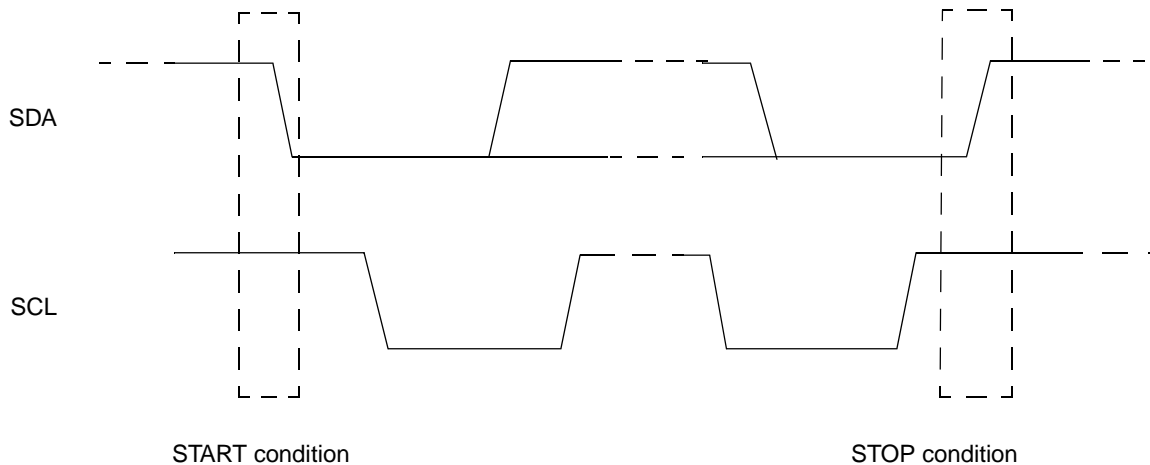


Figure 27-11. Start and Stop conditions

27.4.1.2 Slave Address Transmission

The first byte of data transfer immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a R/W bit. The R/W bit tells the slave the desired direction of data transfer.

1 = Read transfer—the slave transmits data to the master

0 = Write transfer—the master transmits data to the slave

Only the slave with a calling address that matches the one transmitted by the master will respond by sending back an acknowledge bit. This is done by pulling the SDA low at the ninth clock (see [Figure 27-10](#)).

No two slaves in the system may have the same address. If the I²C bus is master, it must not transmit an address that is equal to its own slave address. The I²C bus cannot be master and slave at the same time. However, if arbitration is lost during an address cycle the I²C bus will revert to slave mode and operate correctly, even if it is being addressed by another master.

27.4.1.3 Data Transfer

After successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/W bit sent by the calling master.

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device.

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high (see [Figure 27-10](#)). There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte must be followed by an acknowledge bit, which is signalled from the receiving device by pulling the SDA low at the ninth clock. Therefore, one complete data byte transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop signal to abort the data transfer or a start signal (repeated start) to commence a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means end of data to the slave, so the slave releases the SDA line for the master to generate a STOP or START signal.

27.4.1.4 STOP Signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL is at logical 1 (see [Figure 27-10](#)).

The master can generate a STOP even if the slave has generated an acknowledge, at which point the slave must release the bus.

27.4.1.5 Repeated START Signal

As shown in [Figure 27-10](#), a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

27.4.1.6 Arbitration Procedure

The Inter-IC bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus simultaneously, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest among the masters. The relative priority of the contending masters is determined by a data arbitration procedure. A bus master loses arbitration if it transmits logic “1” while another master transmits logic 0. The losing masters immediately switch to slave receive mode and stop driving the SDA output. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate loss of arbitration.

27.4.1.7 Clock Synchronization

Because wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock remains within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see [Figure 27-12](#)). When all engaged devices have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

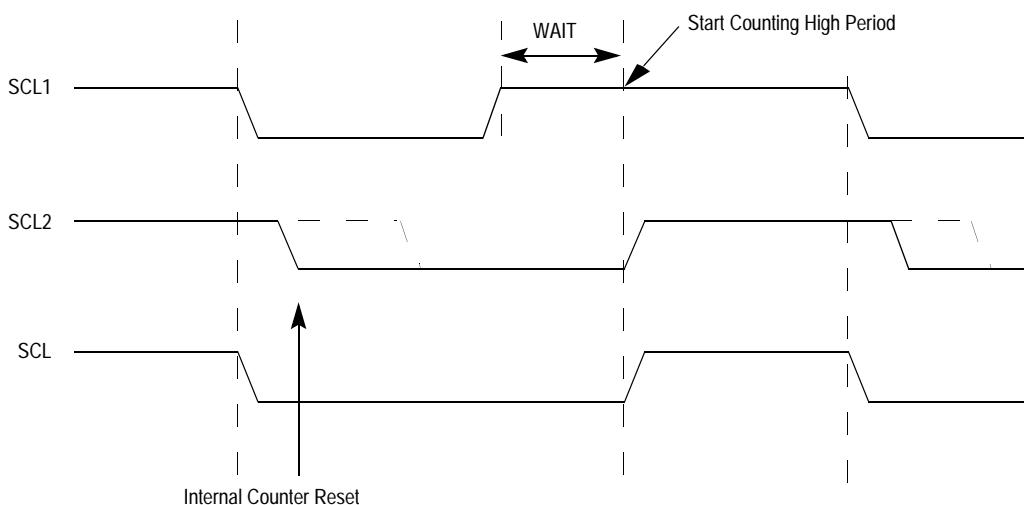


Figure 27-12. I²C Bus Clock Synchronization

27.4.1.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait state until the slave releases the SCL line.

27.4.1.9 Clock Stretching

The clock synchronization mechanism can be used by slaves to slow the bit rate of a transfer. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period then the resulting SCL bus signal low period is stretched.

27.4.2 Interrupts

27.4.2.1 General

The I²C uses one interrupt vector only.

Table 27-8. Interrupt Summary

Interrupt	Offset	Vector	Priority	Source	Description
I ² C Interrupt	—	—	—	IBAL, TCF, IAAS, IBB bits in IBSR register	When any IBAL, TCF, or IAAS bits are set an interrupt may be caused based on arbitration lost, transfer complete or address detect conditions. If enabled by BIIE, the deassertion of IBB can also cause an interrupt, indicating that the bus is idle.

27.4.2.2 Interrupt Description

There are five types of internal interrupts in the I²C. The interrupt service routine can determine the interrupt type by reading the status register.

I²C Interrupt can be generated on

- Arbitration lost condition (IBAL bit set)
- Byte transfer condition (TCF bit set and DMAEN bit not set)
- Address detect condition (IAAS bit set)
- No acknowledge from slave received when expected
- Bus going idle (IBB bit not set)

The I²C interrupt is enabled by the IBIE bit in the I²C control register. It must be cleared by writing 1 to the IBIF bit in the interrupt service routine. The bus going idle interrupt needs to be additionally enabled by the BIIE bit in the IBIC register.

27.5 Initialization/Application Information

27.5.1 I²C Programming Examples

27.5.1.1 Initialization Sequence

Reset will put the I²C bus control register to its default state. Before the interface can be used to transfer serial data, an initialization procedure must be carried out, as follows:

1. Update the frequency divider register (IBFD) and select the required division ratio to obtain SCL frequency from system clock.
2. Update the I²C bus address register (IBAD) to define its slave address.
3. Clear the MDIS bit of the I²C bus control register (IBCR) to enable the I²C interface system.
4. Modify the bits of the IBCR to select master/slave mode, transmit/receive mode and interrupt enable or not. Optionally modify the bits of the I²C bus interrupt config register (IBIC) to further refine the interrupt behavior.
5. Configure the SDA and SCL pads. (The SIU Pad Configuration registers must be configured to select the appropriate I²C function. Also, the open drain feature of the pad must be enabled by setting the ODE bit in the appropriate SIU pad configuration register.)

27.5.1.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the master transmitter mode. If the device is connected to a multi-master bus system, the state of the I²C bus busy bit (IBB) must be tested to check if the serial bus is free.

If the bus is free (IBB=0), the start condition and the first byte (the slave address) can be sent. The data written to the data register comprises the slave calling address and the LSB, which is set to indicate the direction of transfer required from the slave.

The bus free time (i.e., the time between a STOP condition and the following START condition) is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period, it may be necessary to wait until the I²C is busy after writing the calling address to the IBDR before proceeding with the following instructions. This is illustrated in the following example.

An example of the sequence of events which generates the START signal and transmits the first byte of data (slave address) is shown below:

```
while (bit 2, IBSR ==1)    // wait in loop for IBB flag to clear
bit3 and bit 2, IBCR = 1  // set transmit and master mode, i.e. generate start condition
IBDR = calling_address    // send the calling address to the data register
while (bit 2, IBSR ==0)   // wait in loop for IBB flag to be set
```

27.5.1.3 Post-Transfer Software Response

Transmission or reception of a byte will set the data transferring bit (TCF) to 1, which indicates one byte communication is finished. The I²C Bus interrupt bit (IBIF) is set also; an interrupt will be generated if the

interrupt function is enabled during initialization by setting the IBIE bit. The IBIF (interrupt flag) can be cleared by writing 1 (in the interrupt service routine, if interrupts are used).

The TCF bit will be cleared to indicate data transfer in progress by reading the IBDR data register in receive mode or writing the IBDR in transmit mode. The TCF bit must not be used as a data transfer complete flag because the flag timing depends on a number of factors including the I²C bus frequency. This bit may not conclusively provide an indication of a transfer complete situation. Transfer complete situations must be detected using the IBIF flag

Software may service the I²C I/O in the main program by monitoring the IBIF bit if the interrupt function is disabled. Polling should monitor the IBIF bit rather than the TCF bit because their operation is different when arbitration is lost.

When an interrupt occurs at the end of the address cycle, the master will always be in transmit mode, i.e. the address is transmitted. If master receive mode is required, indicated by R/W bit in IBDR, then the TX bit should be toggled at this stage.

During slave mode address cycles (IAAS=1) the SRW bit in the status register is read to determine the direction of the subsequent transfer and the TX bit is programmed accordingly. For slave mode data cycles (IAAS=0) the SRW bit is not valid. The TX bit in the control register should be read to determine the direction of the current transfer.

The following is an example software sequence for master transmitter in the interrupt routine.

```
clear bit 6, IBSR           // Clear the IBIF flag
if (bit 2, IBCR ==0)
    slave_mode()           // run slave mode routine
if (bit 3, IBCR ==0)
    receive_mode()        // run receive_mode routine
if (bit 7, IBSR == 1)     // if NO ACK
    end();                // end transmission
else
    IBDR = data_to_transmit // transmit next byte of data
```

27.5.1.4 Generation of STOP

A data transfer ends with a STOP signal generated by the master device. A master transmitter can simply generate a STOP signal after all the data has been transmitted. The following example shows how a stop condition is generated by a master transmitter.

```
if (tx_count == 0) or      // check to see if all data bytes have been transmitted
    (bit 7, IBSR == 1) {   // or if no ACK generated
    clear bit 2, IBCR      // generate stop condition
}
else {
    IBDR = data_to_transmit // write byte of data to DATA register
    tx_count --            // decrement counter
}                          // return from interrupt
```

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data. This can be done by setting the transmit acknowledge bit (TXAK) before reading the second last byte of data. Before reading the last byte of data, a STOP signal must first be generated. The following example shows how a STOP signal is generated by a master receiver.


```

rx_count --                // decrease the rx counter
if (rx_count ==1)         // 2nd last byte to be read ?
    bit 4, IBCR = 1        // disable ACK
if (rx_count == 0)        // last byte to be read ?
    bit 6, IBCR = 0        // generate stop signal
else
data_received = IBDR      // read RX data and store

```

27.5.1.5 Generation of Repeated START

At the end of data transfer, if the master wants to remain communicating on the bus, it can generate another START signal followed by another slave address without first generating a STOP signal. A program example is as shown.

```

bit 5, IBCR = 1           // generate another start ( restart)
IBDR == calling_address   // transmit the calling address

```

27.5.1.6 Slave Mode

In the slave interrupt service routine, the module addressed as slave bit (IAAS) must be tested to check if a calling of its own address has been received. If IAAS is set, software sets the transmit/receive mode select bit (TX bit of IBCR) according to the R/W command bit (SRW). Writing to the IBCR clears IAAS automatically. IAAS is read as set when it is from the interrupt at the end of the address cycle where an address match occurred. Interrupts resulting from subsequent data transfers will have IAAS cleared. A data transfer may be initiated by writing information to IBDR for slave transmits or dummy reading from IBDR in slave receive mode. The slave drives SCL low in-between byte transfers SCL is released when the IBDR is accessed in the required mode.

In slave transmitter routine, the received acknowledge bit (RXAK) must be tested before transmitting the next byte of data. Setting RXAK means an end of data signal from the master receiver, after which it must be switched from transmitter mode to receiver mode by software. A dummy read then releases the SCL line so the master can generate a STOP signal.

27.5.1.7 Arbitration Lost

If several masters try to engage the bus simultaneously, one master wins and the others lose arbitration. The devices that lost arbitration are immediately switched to slave receive mode by the hardware. Their data output to the SDA line is stopped, but SCL remains generated until the end of the byte during which arbitration was lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with IBAL=1 and MS=0. If one master attempts to start transmission while the bus is being engaged by another master, the hardware inhibits the transmission, switches the MS bit from 1 to 0 without generating a STOP condition, generates an interrupt to CPU, and sets the IBAL to indicate that the attempt to engage the bus is failed. When considering these cases, the slave service routine should test the IBAL first and the software should clear the IBAL bit if it is set.

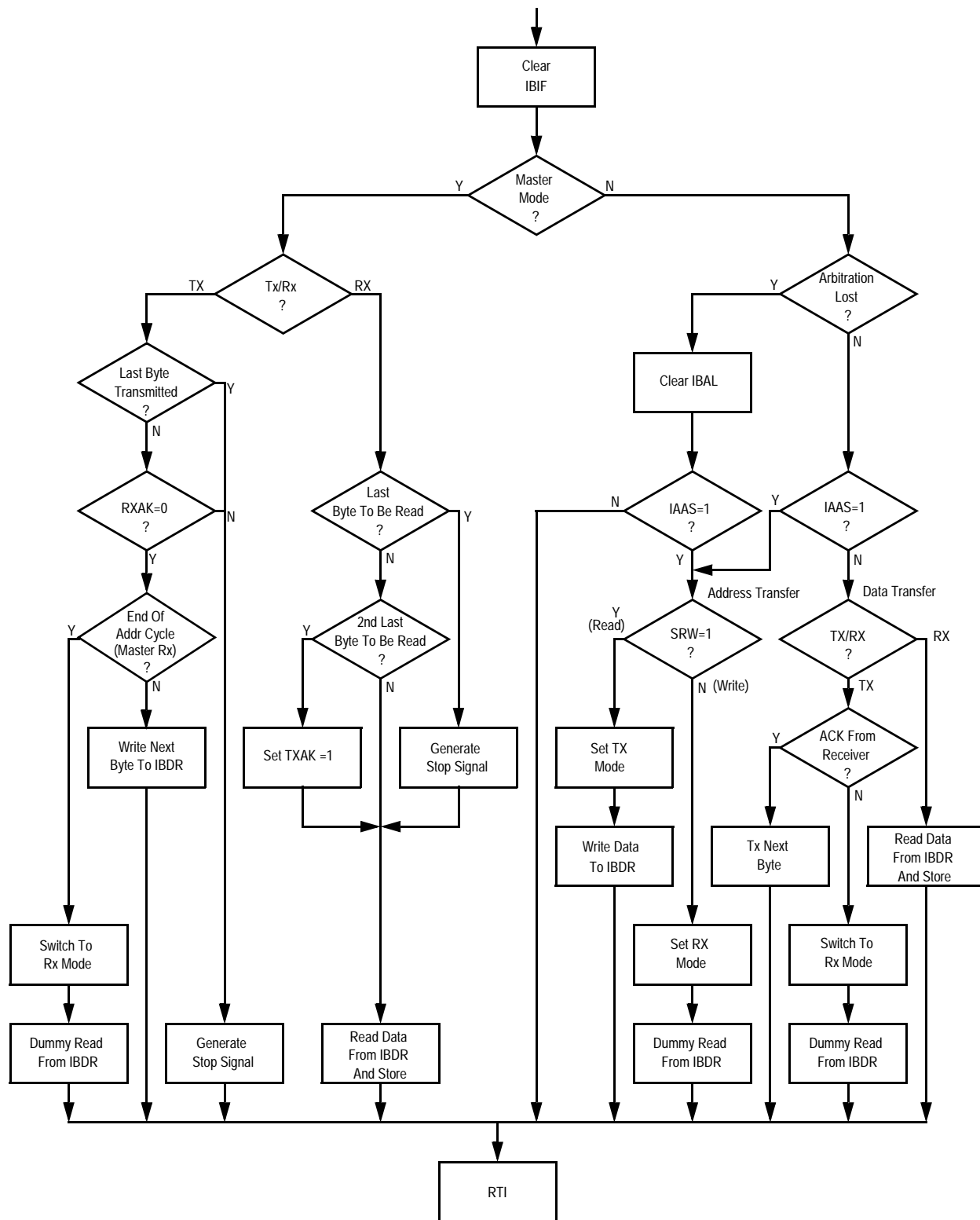


Figure 27-13. Flowchart of Typical I²C Interrupt Routine

27.5.2 DMA Application Information

The DMA interface on the I²C is not completely autonomous and requires intervention from the CPU to start and to terminate the frame transfer. DMA mode is valid for master-transmit and master-receive modes only. Software must ensure that the DMA enable bit in the control register is not set when the I²C module is configured in master mode.

The DMA controller must transfer only one byte of data per Tx/Rx request. This is because there is no FIFO on the I²C block.

The CPU should also keep the I²C interrupt enabled during a DMA transfer to detect the arbitration lost condition and take action to recover from this situation. The DMAEN bit in the IBCR register works as a disable for the transfer complete interrupt. This means that during normal transfers (no errors) there will always be either an interrupt or a request to the DMA controller, dependant on the setting of the DMAEN bit. All error conditions will trigger an interrupt and require CPU intervention. The address match condition will not occur in DMA mode as the I²C should never be configured for slave operation.

The following sections detail how to set up a DMA transfer and what intervention is required from the CPU. It is assumed that the system DMA controller is capable of generating an interrupt after a certain number of DMA transfers have taken place.

27.5.2.1 DMA Mode, Master Transmit

[Figure 27-14](#) details exactly the operation for using a DMA controller to transmit n data bytes to a slave. The first byte (the slave calling address) is always transmitted by the CPU. All subsequent data bytes (apart from the last data byte) can be transferred by the DMA controller. The last data byte must be transferred by the CPU.

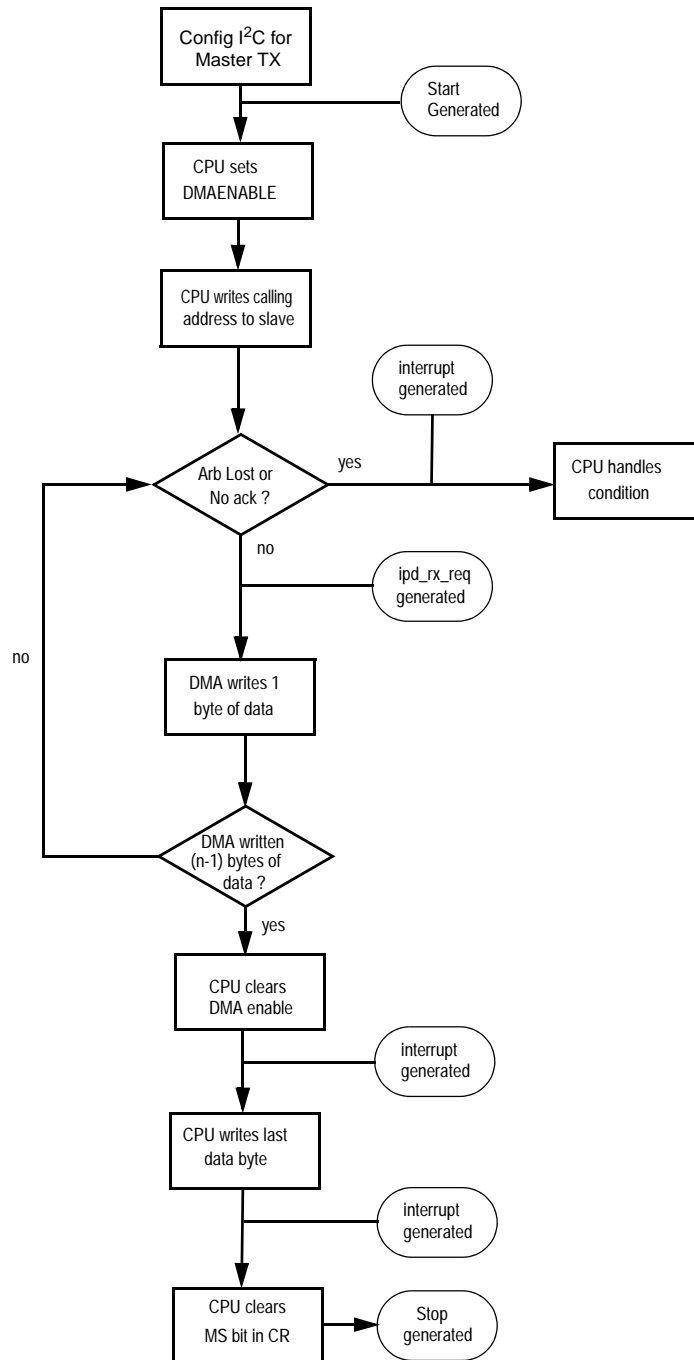


Figure 27-14. Flowchart of DMA Mode Master Transmit

27.5.2.2 DMA Mode, Master RX

Figure 27-15 details the exact operation for using a DMA controller to receive n data bytes from a slave. The first byte (the slave calling address) is always transmitted by the CPU. All subsequent data bytes (apart from the two last data bytes) can be read by the DMA controller. The last two data bytes must be transferred by the CPU.

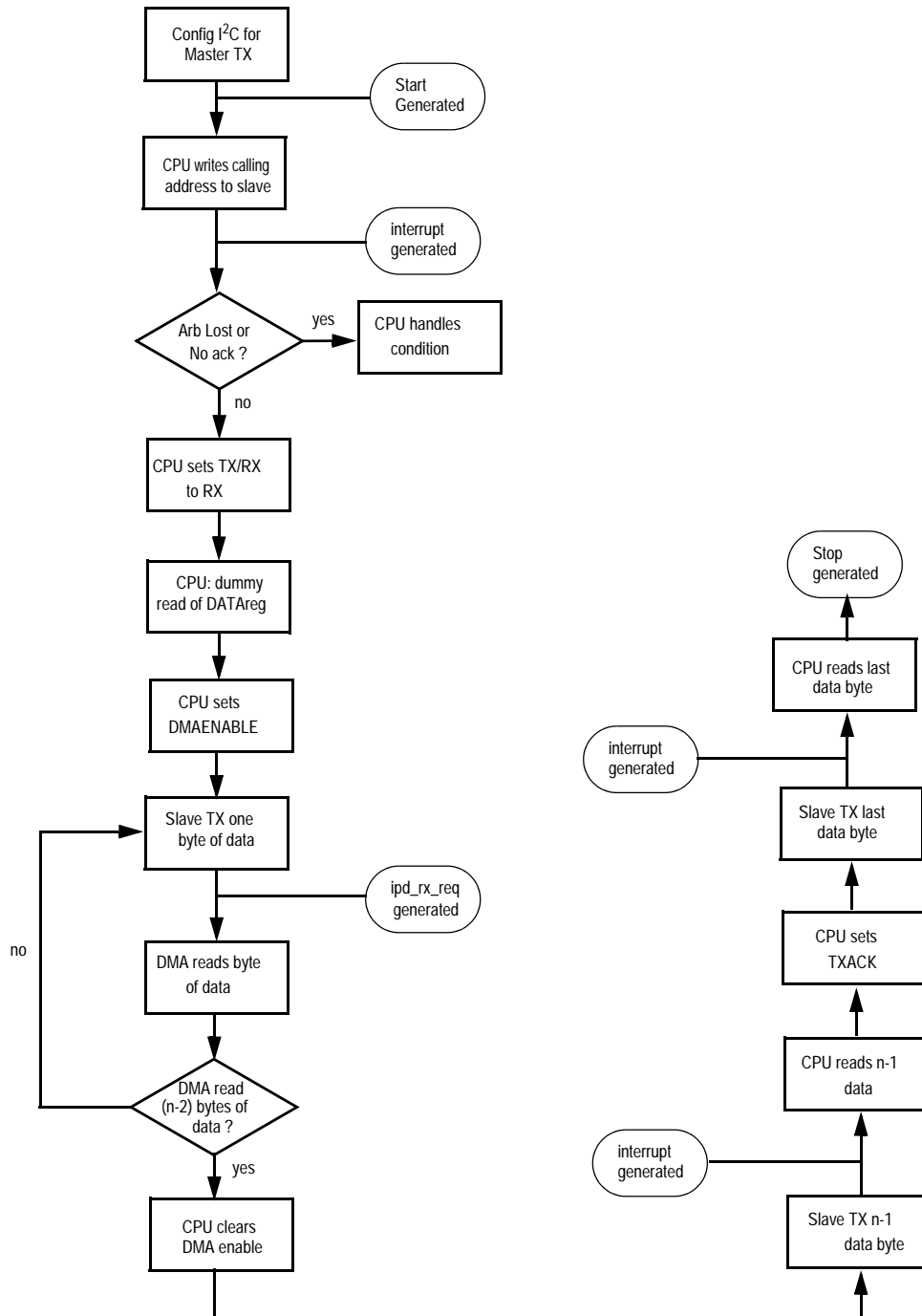


Figure 27-15. Flowchart of DMA Mode Master Receive

27.5.2.3 Exiting DMA Mode, System Requirement Considerations

As described above, the final transfers of both Tx and Rx transfers need to be managed via interrupt by the CPU. To change from DMA to interrupt driven transfers in the I²C module, disable the DMAEN bit in the IBCR register. The trigger to exit the DMA mode is that the programmed DMA transfer control descriptor (TCD) has completed all its transfers to/from the I²C module.

After the last DMA write (TX mode) to the I²C the module immediately starts the next I²C-bus transfer. The same is true for RX mode. After the DMA read from the IBDR register the module initiates the next I²C-bus transfer. This results in two possible scenarios in the DMA mode exiting scheme.

1. Fast reaction

The DMAEN bit is cleared before the next I²C-bus transfer completes. In this case the module will raise an interrupt request to the CPU which can be serviced normally.

2. Slow reaction

The DMAEN bit is cleared after the next I²C-bus transfer has already completed. In this case the module will not raise an interrupt request to the CPU. Instead the TCF bit can be read to determine that the transfer completed and the module is ready for further transfer.

27.5.2.3.1 Fast vs. Slow Reaction

The reaction time T_R for the system to disable DMAEN after the last DMA controller access to the I²C is the time required for one byte transfer over the I²C. In a fast reaction the disabling has to occur before the ninth bit of the data transfer, which is the ACK bit. So the time available is eight times the SCL period.

$$T_R = 8 \times T_{SCL} \quad \text{Eqn. 27-5}$$

In fast mode, with 400 kbit/s, T_{SCL} is 2.5 μ s, so T_R is 20 μ s.

Depending on the system and DMA controller there are different possibilities for the deassertion of DMAEN. Three options are:

1. CPU intervention via interrupt

The DMA controller is programmed to signal an interrupt to the CPU which is then responsible for the deassertion of DMAEN. This scheme is supported by most systems but can result in a slow reaction time if higher priority interrupts interfere. Therefore, the interrupt handling routine can become complicated as it has to check which of the two scenarios happened (check TCF bit) and act accordingly. In case of slow reaction you can force an interrupt for the I²C in the interrupt controller to have the further transfer handled by the normal I²C interrupt routine. The use of nested interrupts can cause problems in this scenario, if the DMA interrupt stalls between the deassertion and the DMAEN bit and the checking of the TCF bit.

2. DMA channel linking (if supported)

The transfer control descriptor in the DMA controller that performs the data transfer is linked to another channel that does a write to the I²C IBCR register to disable the DMAEN bit. This is probably the fastest system solution, but it uses two DMA channels. On the system level, no higher priority DMA requests must occur between the two linked TCDs because those can result in slow reaction.

3. DMA scatter/gather process (if supported)

The transfer control descriptor in the DMA controller that performs the data transfer has the scatter-gather feature activated. This feature initiates a reload of another TCD from system RAM after the completion of the first TCD. The new TCD will have its start bit already set and immediately start the required write to the I²C IBCR register to disable the DMAEN bit. This TCD also has scatter-gather activated and is programmed to reload the initial TCD upon completion,

bringing the system back into a ready-for-I²C-transfer state. The advantage over the two other solutions is that this does not require CPU intervention or a second DMA channel. This comes at the cost of 64 bytes RAM (two TCDs), some system bus transfer overhead, and a little increase in application code complexity. On the system level, no higher priority DMA requests must occur during the scatter-gather process because those can result in a slow reaction.

Example latencies for a 32 MHz system with a full speed 32-bit AHB bus and an I²C connected via half speed IPI bus:

- Accessing the I²C from the DMA controller via IPI bus typically requires four cycles (consecutive accesses to the I²C could be faster):

$$4 \times T_{\text{IPI}} = 4 / 16 \text{ MHz} = 250 \text{ ns} \quad \text{Eqn. 27-6}$$

- Reloading a new TCD (8 × 32 bit) via AHB to the DMA controller (scatter/gather process):

$$8 \times T_{\text{AHB}} = 8 / 32 \text{ MHz} = 250 \text{ ns} \quad \text{Eqn. 27-7}$$

With the DMA scatter-gather process, the required IBCR access can be done in 0.5 μs, leaving a large margin of 19.5 μs for additional system delays. The slow reaction case can be prevented in this way. The system user must decide which usage model suits his overall requirements best.

Chapter 28

Periodic Interrupt Timer and Real Time Interrupt (PIT_RTI)

28.1 Introduction

The PIT_RTI is an array of timers that can be used to initiate interrupts and trigger DMA channels. It also provides a dedicated real-time interrupt (RTI) timer, which runs on a separate clock and can be used for system wakeup.

28.1.1 Block Diagram

A simplified block diagram of the PIT_RTI illustrates the functionality and interdependence of major blocks (see [Figure 28-1](#)).

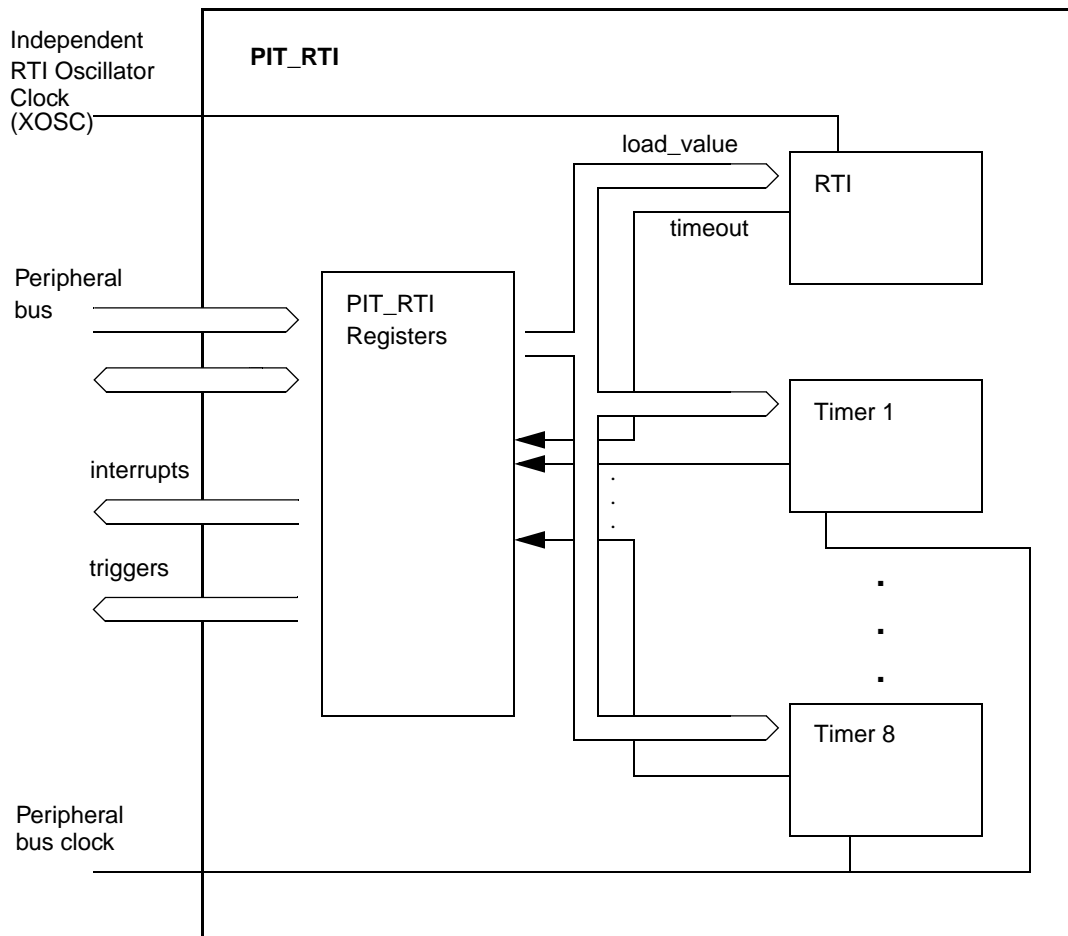


Figure 28-1. PIT_RTI Block Diagram

28.1.2 Features

The PIT_RTI has these major features:

- One 32-bit RTI timer to wakeup the CPU in wait mode
- Eight additional 32-bit timers generating DMA trigger pulses
- Timers can be configured to generate interrupts instead of triggers
- Timers 7 and 8 can be the source of the eQADC trigger inputs via SIU configuration
- All interrupts are maskable and can be initiated even when the bus clock is switched off
- Power saving with a separate input clock for the RTI timer (all other timers share one common core clock)
- Independent timeout periods for each timer

Table 28-1. Timer Features

Timer	Interrupt	DMA Trigger	eQADC Trigger
0 (RTI)	X		
1	X	X	
2	X	X	
3	X	X	
4	X	X	
5	X	X	
6	X	X	
7	X	X	X
8	X	X	X

28.1.3 Modes of Operation

There are two main operating modes of PIT_RTI: run mode and stop mode. In run mode, PIT=0 in the SIU_HLT register and all functional parts of the PIT_RTI module are running. In stop mode, PIT=1 in the SIU_HLT register and all clocks to the PIT_RTI module are disabled except the XOSC.

28.2 Signal Description

28.2.1 External Signal Description

The PIT_RTI module has no external signals.

28.3 Memory Map and Registers

This section provides a detailed description of all PIT_RTI registers.

28.3.1 Module Memory Map

The PIT_RTI memory map is shown in [Table 28-2](#). The address of each register is given as an offset to the PIT_RTI base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 28-2. PIT_RTI Memory Map

Offset from PIT_RTI_BASE (0xFFFE_0000)	Register	Access	Reset Value	Section/Page
0x0000	TLVAL0 — PIT RTI load value register	R/W	0x0000_0000	28.3.2.2/28-5
0x0004	TLVAL1 — PIT timer load value register 1	R/W	0x0000_0000	
0x0008	TLVAL2 — PIT timer load value register 2	R/W	0x0000_0000	
0x000C	TLVAL3 — PIT timer load value register 3	R/W	0x0000_0000	
0x0010	TLVAL4 — PIT timer load value register 4	R/W	0x0000_0000	
0x0014	TLVAL5 — PIT timer load value register 5	R/W	0x0000_0000	
0x0018	TLVAL6 — PIT timer load value register 6	R/W	0x0000_0000	
0x001C	TLVAL7 — PIT timer load value register 7	R/W	0x0000_0000	
0x0020	TLVAL8 — PIT timer load value register 8	R/W	0x0000_0000	
0x0024–0x007F	Reserved			
0x0080	TVAL0 — PIT current RTI value	R	0x0000_0000	28.3.2.2/28-5
0x0084	TVAL1 — PIT current timer value 1	R	0x0000_0000	
0x0088	TVAL2 — PIT current timer value 2	R	0x0000_0000	
0x008C	TVAL3 — PIT current timer value 3	R	0x0000_0000	
0x0090	TVAL4 — PIT current timer value 4	R	0x0000_0000	
0x0094	TVAL5 — PIT current timer value 5	R	0x0000_0000	
0x0098	TVAL6 — PIT current timer value 6	R	0x0000_0000	
0x009C	TVAL7 — PIT current timer value 7	R	0x0000_0000	
0x00A0	TVAL8 — PIT current timer value 8	R	0x0000_0000	
0x00A4–0x00FF	Reserved			
0x0100	PITFLG — PIT interrupt flags register	R/W	0x0000_0000	28.3.2.3/28-6
0x0104	PITINTEN — PIT interrupt enable register	R/W	0x0000_0000	28.3.2.4/28-6
0x0108	PITINTSEL — PIT interrupt/DMA select register	R/W	0x0000_0001	28.3.2.5/28-7
0x010C	PITEN — PIT timer enable register	R/W	0x0000_0000	28.3.2.6/28-8
0x0110	PITCTRL — PIT control register	R/W	0x0100_0000	28.3.2.7/28-8
0x0114–0x01FC	Reserved			

28.3.2 Register Descriptions

This section lists the PIT_RTI registers in address order and describes the registers and their bit fields.

NOTE

The RTI registers should be set when the RTI clock is running only.

28.3.2.1 PIT RTI / Timer Load Value Register (TLVAL0–TLVAL8)

These registers select the timeout period for the timer interrupts. In the case of the RTI, it will take several cycles until this value is synchronized into the RTI clock domain. For all other timers the value change is visible immediately.

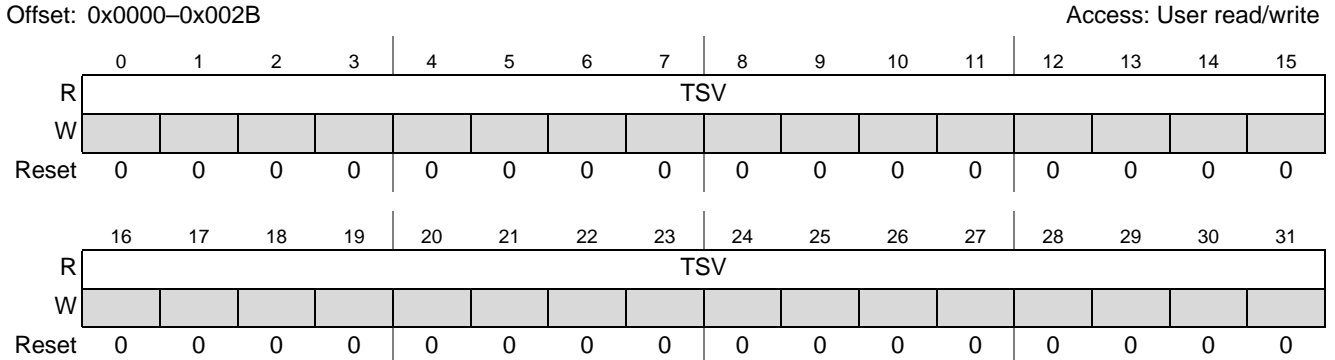


Figure 28-2. PIT Timer Load Value Register (TLVAL0–TLVAL8)

Table 28-3. TLVAL0–TLVAL8 Field Descriptions

Field	Description
TSV	<p>Time Start Value Bits. These bits set the timer start value. The timer will count down until it reaches 0, then it will generate an interrupt and load this register value again. Writing a new value to this register will not restart the timer, instead the value will be loaded after the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see Figure 28-7).</p> <p>Note: For the RTI, the timer must not be set to a value lower than 32 cycles, otherwise interrupts may be lost, as it takes several cycles to clear the RTI interrupt. For the other timers, this limit does not apply, however there will be practical limits because the processor will require several cycles to service an interrupt.</p>

28.3.2.2 PIT Current RTI / Timer Values (TVAL0–TVAL8)

These registers indicate the current timer position. In the case of the RTI, this will show a value which is several cycles old, since it originates from a potentially different clock domain.

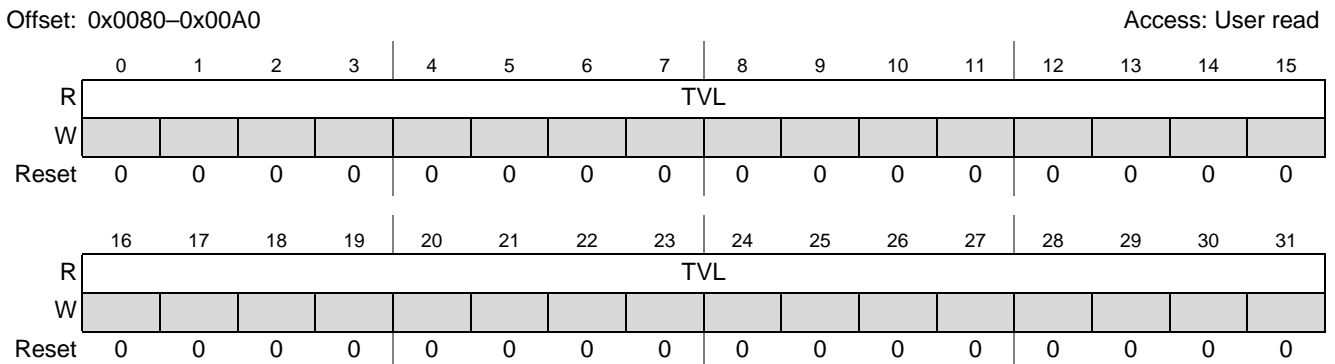


Figure 28-3. PIT Current Timer Values (TVAL0–8)

Table 28-4. TVAL0–8 Field Descriptions

Field	Description
TVL	Current Timer Value. These bits represent the current timer value. Note that the timer uses a downcounter. NOTE: The timer values will be frozen in debug mode.

28.3.2.3 Interrupt Flags Register (PITFLG)

This register holds the PIT interrupt flags. Timer 0 is the special timer RTI, which can be used to wake up the device.

Offset: 0x0100

Access: User read/write
(write to clear)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	TIF8	TIF7	TIF6	TIF5	TIF4	TIF3	TIF2	TIF1	RTIF
W								w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-4. PIT Interrupt Flags Register (PITFLG)

Table 28-5. PITFLG Field Descriptions

Field	Description
bits 0–22	Reserved.
TIF n	Real Time Interrupt Flags for Timer 1–8. TIF n is set to 1 at the end of the timer period. This flag can be cleared by writing a 1 only. Writing a 0 has no effect. If enabled (TIE x = 1 and ISEL x = 1), TIF n causes an interrupt request. 0 Time-out has not yet occurred 1 Time-out has occurred
RTIF	Real-Time Interrupt Flag. RTIF is set to 1 at the end of the RTI period. This flag can be cleared by writing a 1 only. Writing a 0 has no effect. If enabled (RTIE = 1), RTIF causes an interrupt request. 0 RTI time-out has not yet occurred 1 RTI time-out has occurred

28.3.2.4 PIT Interrupt Enable Register (PITINTEN)

This register enables PIT interrupts.

Offset: 0x0104

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	TIE8	TIE7	TIE6	TIE5	TIE4	TIE3	TIE2	TIE1	RTIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-5. PIT Interrupt Enable Register (PITINTEN)

Table 28-6. PITINTEN Field Descriptions

Field	Description
bits 0–22	Reserved.
TIE _n	Timer Interrupt Enable Bit. 0 Interrupt requests from Timer x are disabled 1 Interrupt will be requested whenever TIF _x is set When an interrupt is pending (TIF/RTIF set), enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TIF/RTIF flag must be cleared first.
RTIE	Real Time Interrupt Enable Bit. 0 Interrupt requests from RTI are disabled 1 Interrupt will be requested whenever RTIF is set

28.3.2.5 PIT Interrupt/DMA Select Registers (PITINTSEL)

This register decides whether a channel generates an interrupt or is used for DMA triggering.

Offset: 0x0108

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	ISEL8	ISEL7	ISEL6	ISEL5	ISEL4	ISEL3	ISEL2	ISEL1	1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 28-6. PIT Interrupt/DMA Select Registers (PITINTSEL)

Table 28-7. PITINTSEL Field Descriptions

Field	Description
0–22	Reserved.
23–30 ISEL _n	Interrupt Selector. 0 The timer will trigger a DMA channel 1 The timer will generate an interrupt if enabled
31	Reserved.

28.3.2.6 PIT Timer Enable Register (PITEN)

This register enables the PIT timers.

Offset: 0x010C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PEN8	PEN7	PEN6	PEN5	PEN4	PEN3	PEN2	PEN1	PEN0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-7. PIT Timer Enable Register (PITEN)

Table 28-8. PITEN Field Descriptions

Field	Description
bits 0–22	Reserved.
PEN _n	Timer Enable Bit. 0 Timer will be disabled 1 Timer will be active

28.3.2.7 PIT Control Register (PITCTRL)

This register controls whether the clock for the timers 1–8 is enabled. The RTI timer (timer 0) runs on a separate clock (XOSC) that is controlled by the CRP and PLL.

Offset: 0x0110

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0		MDIS	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-8. PIT Control Registers (PITCTRL)

Table 28-9. PITCTRL Field Descriptions

Field	Description
bits 0–6	Reserved. Note: Bit 6 is a reserved bit, but can be read and written. Writing to this bit will update the value, and reading it will return the last value written, but this bit has no other effect.
MDIS	Module Disable. This is used to disable timers 1–8. The RTI (timer 0) is not affected by this bit. The module should be enabled before any setup is done. 0 Clock for timers 1–8 is enabled 1 Clock for timers 1–8 is disabled (default)
bits 8–31	Reserved.

28.4 Functional Description

This section gives detailed information about the internal operation of the module. The PIT block has nine timers: one RTI timer and eight additional timers for general-purpose use (e.g. DMA triggering, eQADC triggering).

28.4.1 Timer / RTI

The timers generate triggers at periodic intervals, when enabled. They load their start values, as specified in their TLVAL registers, then count down until they reach 0. This creates a trigger, then they load their respective start value again. Each time a timer reaches 0, it will generate a trigger pulse and set the interrupt flag.

All interrupts can be enabled or masked (by setting the TIE/RTIE bits in the PITINTEN register and selecting interrupts in the PITINTSEL register). In the case of the RTI, because clearing the interrupt crosses clock domains, a minimum value of 32 must be maintained.

If desired, the current counter value of the timer can be read via the TVAL registers. The value of the RTI counter can be delayed considerably, as it is synchronized to the bus clock from the RTI clock domain.

The counter period can be restarted by first disabling and then enabling the timer with the PITEN register (see [Figure 28-9](#)).

The counter period of a running timer can be modified by first disabling the timer, setting a new load value, and then enabling the timer again (see [Figure 28-10](#)).

It is also possible to change the counter period without restarting the timer by writing the TLVAL register with the new load value. This value will be loaded after the next trigger event (see [Figure 28-11](#)).

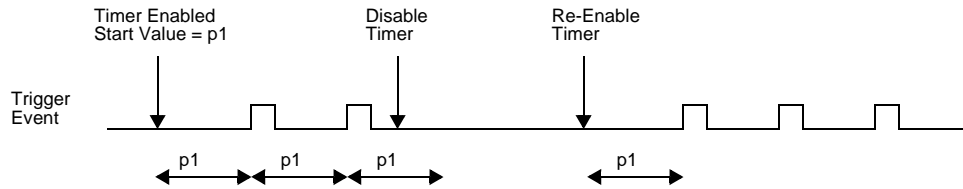


Figure 28-9. Stopping and Starting a Timer

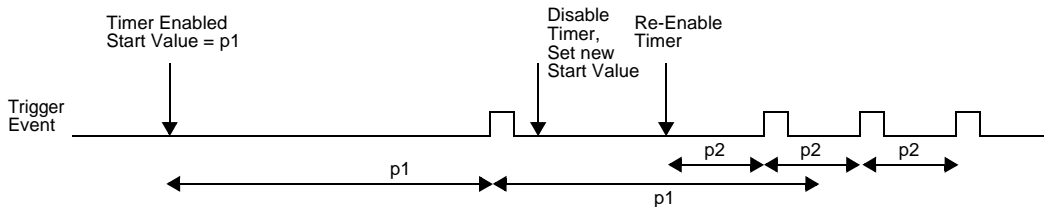


Figure 28-10. Modifying Running Timer Period

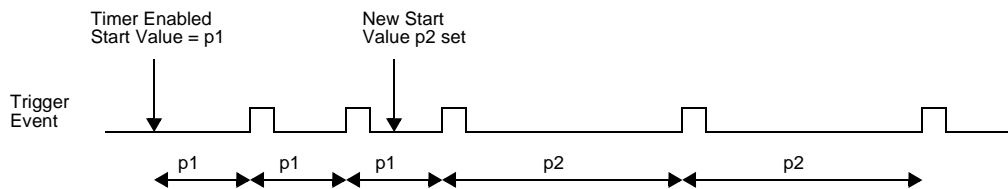


Figure 28-11. Dynamically Setting a New Load Value

28.4.2 Debug Mode

In debug mode the timers will be frozen—this is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (e.g. the timer values), and then continue the operation.

28.4.3 Interrupts

The interrupts generated by the PIT are listed in [Table 28-10](#). Refer to the MCU specification for related vector addresses and priorities.

Table 28-10. PIT Interrupt Vectors

Interrupt Source	Local Enable
Real-time interrupt	RTIE
Timer interrupts	TIE8:1

28.4.3.1 Real-Time Interrupt

The PIT_RTI generates a real-time interrupt when the selected interrupt time period elapses. The RTI interrupt is disabled locally by setting the RTIE bit to 0. The real time interrupt flag (RTIF) is set to 1 when a timeout occurs, and is cleared to 0 by writing a 1 to the RTIF bit.

28.4.3.2 Timer Interrupts

The PIT can also generate timer interrupts on all eight timer channels when the timer's selected interrupt period elapses. Timer interrupts are disabled locally by setting the TIE bits to 0. The timer interrupt flags (TIF) are set to 1 when a timeout occurs on the associated timer and are cleared to 0 by writing a 1 to that TIF bit. To activate a timer interrupt it must also be switched from trigger mode into interrupt mode, using the PITINTSEL register.

The timer interrupts are general-purpose interrupts.

28.5 Initialization and Application Information

28.5.1 Example Configuration

In the example configuration:

- The PIT clock has a frequency of 50 MHz
- The RTI clock has a frequency of 10 MHz
- The RTI timer shall be set up to create a wakeup interrupt every 500 ms
- Timer 1 shall create an interrupt every 5.12 ms
- Timer 8 shall create a trigger event every 30 ms.

First the PIT module needs to be activated by writing a 0 to the MDIS bit in the PITCTRL register.

The 50 MHz clock frequency equates to a clock period of 20 ns and the 10 MHz frequency equates to a clock period of 100 ns. Therefore the RTI timer needs to trigger every $500 \text{ ms}/100 \text{ ns} = 5000000$ cycles, timer 1 needs to trigger every $5.12 \text{ ms}/20 \text{ ns} = 256000$ cycles, and timer 8 needs to trigger every $30 \text{ ms}/20 \text{ ns} = 1500000$ cycles. The value for the TVAL register trigger would be calculated as (period / clock period) - 1.

This means that TVAL0 will be written with 004C4B3F hex, TVAL1 with 0x0003_E7FF, and TVAL8 with 0x0016_E35F.

To generate the interrupt, the interrupt line must be enabled by writing a 1 to the RTIE bit in the PITINTEN register. There is no need to modify PITINTSEL because the RTI timer is always used for interrupts and never for trigger events. To start the RTI, PEN0 in the PIT timer enable register 0 (PITEN0) is set.

The interrupt for timer 1 is enabled by setting TIE1 in the PITINTEN register and the interrupt/DMA selector ISEL1 (in PITINTSEL) is set to 1. The timer is started by writing a 1 to bit PEN1 in the PITEN register.

Timer 8 will be used for triggering only. Timer 8 is started by writing a 1 to bit PEN8 in the PITEN register.

It is also possible to set up all timers and start them simultaneously by writing to the PITEN register. However the RTI still cannot start in synchronization because it is running on a separate clock.

The following example code matches the described setup:

```
// turn on PIT
PIT_REG_P->pit_CTRL = 0x00;

// RTI
CRG_REG_P->crg_CTL   |= 1<<2; // Set RTI bit in CLKSEL
PIT_REG_P->pit_TLVAL0 = 0x004C4B3F; // setup RTI for 5000000 cycles
PIT_REG_P->pit_INTEN = 0x00000001; // let RTI generate interrupts
// writing INTSEL is unnecessary
PIT_REG_P->pit_EN   |= 1<<0; // start RTI

// Timer 1
PIT_REG_P->pit_TLVAL1 = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_REG_P->pit_INTEN |= 1<<1; // enable Timer 1 interrupts
PIT_REG_P->pit_INTSEL |= 1<<1; // select Timer 1 for interrupts
PIT_REG_P->pit_EN   |= 1<<1; // start timer 1

// Timer 8
PIT_REG_P->pit_TLVAL8 = 0x0016E35F; // setup timer 8 for 1500000 cycles
// timer 8 can't generate interrupts -> no settings needed for trigger
PIT_REG_P->pit_EN   |= 1<<8; // start timer 8
```

Chapter 29

External Bus Interface (EBI)

29.1 Introduction

The EBI manages the transfer of information between the internal buses and the memories or peripherals in the external address space. The EBI includes a memory controller that generates interface signals to support a variety of external memories. This includes single data rate (SDR) burst mode flash, SRAM, and asynchronous memories. It supports up to four regions (via chip selects), each with its own programmed attributes.

29.1.1 Block Diagram

A simplified block diagram of the EBI illustrates the functionality and interdependence of major blocks (see [Figure 29-1](#)).

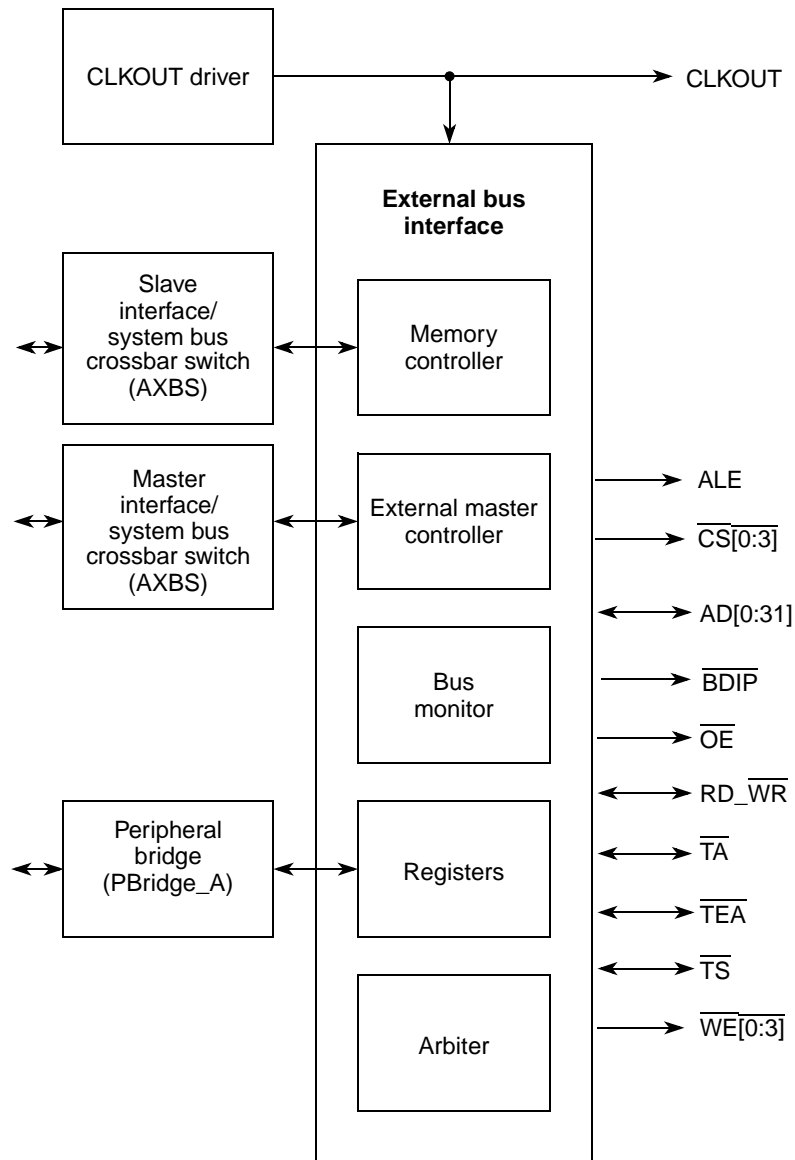


Figure 29-1. EBI Block Diagram

29.1.2 Features

The EBI has these major features:

- Multiplexed 32-bit address/data bus (single master and external master)
 - In the 208-pin package, there can be up to 24 address bits with 32-bit data and four chip selects. In the 144-pin and 176-pin packages, there are 24 address bits with only 16 bits of data and four chip selects
- Memory controller with support for various memory types:
 - Synchronous burst SDR flash and SRAM
 - Asynchronous/legacy flash and SRAM

- Burst support (wrapped only)

NOTE

Because the MPC5510 has no cache, the core does not generate any burst accesses; therefore the only burst accesses possible to the EBI are from the DMA.

- Bus monitor
- Port size configuration per chip select (16 or 32 bits)
- Configurable wait states
- Configurable internal or external transfer acknowledge (\overline{TA}) per chip select
- Four chip-select ($\overline{CS}[0:3]$) signals
- Four write/byte enable ($\overline{WE}[0:3]$) signals
- Configurable bus speed modes (1/2 or 1/4 of system clock frequency) - up to 25 MHz maximum bus frequency
- Stop and module-disable modes for power savings
- Optional automatic CLKOUT gating to save power and reduce EMI
- Misaligned access support (for chip-select accesses only)

29.1.3 Modes of Operation

The mode of the EBI is determined by the MDIS, EXTM, and AD_MUX bits in the EBI_MCR. See [Section 29.3.2.3, “EBI Module Configuration Register \(EBI_MCR\),”](#) for details. Configurable bus speed modes and debug mode are modes that the MCU can enter, in parallel to the EBI being configured in one of its module-specific modes.

29.1.3.1 Single Master Mode

In single-master mode, the EBI responds to internal requests matching one of its regions, but ignores all externally-initiated bus requests. The MCU is the only master allowed to initiate transactions on the external bus in this mode; therefore, it acts as a parked master and does not have to arbitrate for the bus before starting each cycle. Single-master mode is entered when EXTM=0 and MDIS=0 in the EBI_MCR.

29.1.3.2 External Master Mode

External-master mode is supported for factory test use only.

29.1.3.3 Module Disable Mode

The module disable mode is used for MCU power management. The clock to the non-memory-mapped logic in the EBI can be stopped while in module disable mode. Requests (other than to memory-mapped logic) must not be made to the EBI while it is in module disable mode, even if the clocks have not yet been shut off. In this case, the behavior is undefined. Module disable mode is entered when MDIS = 1 in the EBI_MCR.

29.1.3.4 Configurable Bus Speed Modes

In configurable bus speed modes, the external CLKOUT frequency is divided down from the internal system clock. The EBI behavior remains dictated by the mode of the EBI, except that the EBI drives and samples signals at the scaled CLKOUT rather than the internal system clock. This mode is selected by writing the external clock control register in the system integration module (SIU_ECCR). The configurable bus speed modes support 1/2 or 1/4 speed modes, meaning that the external CLKOUT frequency is scaled down (by 2 or 4) compared with that of the internal system clock, which is unchanged.

NOTE

Nothing prevents the user from configuring the SIU register to run the EBI in 1/2 or 1/4 of system clock frequency; however, the user must ensure that the maximum operating frequency of the EBI (per the electrical specification) is not exceeded, or unreliable behavior may result. In the common case of running the system clock at 66 MHz, this means that only 1/4 speed mode is supported, because at 1/2 speed the maximum EBI bus frequency (25 MHz) would be exceeded.

29.1.3.5 16-Bit Data Bus Mode

The EBI has an internal 32-bit data bus, but the EBI supports a 16-bit data bus mode for MCUs that have only 16 data bus signals pinned out, or for systems where the use of a different multiplexed function (e.g. GPIO) is desired on 16 of the 32 data pins. In this mode, AD[16:31] are the only data signals used by the EBI by default, though the user can change this to use AD[0:15] instead by modifying the D16_31 bit in the EBI_MCR.

For EBI-mastered accesses, the operation in 16-bit data bus mode (DBM=1, PS=x) is similar to a chip-select access to a 16-bit port in 32-bit data bus mode (DBM=0, PS=1), except for the case of a non-chip-select access of exactly 32-bit size.

EBI-mastered non-chip-select accesses of exactly 32-bit size are supported via a two (16-bit) beat burst for both reads and writes. See [Section 29.4.2.10, “Non-Chip-Select Burst in 16-bit Data Bus Mode,”](#) for more details. Non-chip-select transfers of non-32-bit size are supported in standard non-burst fashion.

16-bit data bus mode is entered when DBM=1 in the EBI_MCR. On MPC5510, the default value of the DBM bit out of reset is 0. Thus the EBI operates in 32-bit data bus mode by default.

29.1.3.6 Multiplexed Address on Data Bus Mode

This mode covers several cases aimed at reducing pin count on MCU and external components. In this mode, the AD pins will drive (for internal master cycles) the address value on the first clock of the cycle (while \overline{TS} is asserted). The AD pins will also be used to sample the incoming address on the first clock of a cycle for external master accesses. The address latch enable (ALE) is valid when the address is presented on the AD pins and the falling edge of ALE may be used to capture the valid address.

The memory controller supports per-chip-select selection of multiplexing address/data through the BRx[AD_MUX] bit.

Address on data bus multiplexing also supports the 16-bit data bus mode ($MCR[DBM]=1$) and 16-bit memories ($ORx[PS]=1$). The user can select which 16 data signals are used ($AD[0:15]$ or $AD[16:31]$) by writing the $D16_31$ bit in the EBI_MCR . For either setting of $D16_31$, the 16 least significant bits (LSBs) of external address ($ADDR[16:31]$) are driven onto the selected 16 AD pins. If additional address lines are required to interface to the memory, then non-muxed address pins are required to complete the address space ($ADDR[8:15]$ are available as non-muxed address pins).

See [Section 29.4.2.11, “Address Data Multiplexing,”](#) for more details.

29.1.3.7 Debug Mode

When the MCU is in debug mode, the EBI behavior is unaffected and remains dictated by the mode of the EBI.

29.1.3.8 Stop Mode

The EBI supports a stop mode mechanism used for MCU power management. When a request is made to enter stop mode (controlled in SIU_HLT register outside EBI), the EBI block completes any pending bus transactions and acknowledges the stop request. After the acknowledgement, the system clock input may be shut off by the clock driver on the MCU. While the clocks are shut off, the EBI is not accessible. While in stop mode, accesses to the EBI from the internal master will terminate with transfer error (internally, no external \overline{TEA} assertion).

29.2 Signal Description

29.2.1 External Signal Description

29.2.1.1 \overline{BDIP} — Burst Data in Progress

\overline{BDIP} is asserted to indicate that the EBI is requesting another data beat following the current one.

\overline{BDIP} is driven by the EBI on all EBI-mastered external burst cycles, but is only sampled by burst mode memories that have a corresponding pin.

29.2.1.2 $ADDR[8:15]$ — Address Lines 8-15

The $ADDR[8:15]$ signals specify the physical address of the bus transaction.

29.2.1.3 $CLKOUT$ — Clockout

$CLKOUT$ is a general-purpose clock output signal to connect to the clock input of SDR external memories and in some cases to the input clock of another MCU in multi-master configurations.

29.2.1.4 $\overline{CS}[0:3]$ — Chip Selects 0-3

\overline{CS}_x is asserted by the master to indicate that this transaction is targeted for a particular memory bank on the Primary external bus.

\overline{CS} is driven in the same clock as the assertion of \overline{TS} and valid address, and is kept valid until the cycle is terminated

29.2.1.5 AD[0:31] — Multiplexed Address and Data Lines 0-31

The AD[0:31] signals contain the address and data for the current transaction.

29.2.1.6 ALE — Address Latch Enable

The address latch enable (ALE) is valid when the address is presented on the AD pins and the falling edge of ALE may be used to capture the valid address. ALE is asserted high during the low phase of CLKOUT when \overline{TS} is asserted.

29.2.1.7 \overline{OE} — Output Enable

\overline{OE} is used to indicate when an external memory is permitted to drive back read data. External memories must have their data output buffers off when \overline{OE} is negated. \overline{OE} is only asserted for chip-select accesses.

For read cycles, \overline{OE} is asserted one clock after \overline{TS} assertion and held until the termination of the transfer. For write cycles, \overline{OE} is negated throughout the cycle.

29.2.1.8 RD_ \overline{WR} — Read / Write

RD_ \overline{WR} indicates whether the current transaction is a read access or a write access.

RD_ \overline{WR} is driven in the same clock as the assertion of \overline{TS} and valid address, and is kept valid until the cycle is terminated.

29.2.1.9 \overline{TA} — Transfer Acknowledge

\overline{TA} is asserted to indicate that the slave has received the data (and completed the access) for a write cycle, or returned data for a read cycle. If the transaction is a burst read, \overline{TA} is asserted for each one of the transaction beats. For write transactions, \overline{TA} is only asserted once at access completion, even if more than one write data beat is transferred.

29.2.1.10 \overline{TEA} — Transfer Error Acknowledge

\overline{TEA} is asserted by an external device to indicate that an error condition has occurred during the bus cycle.

29.2.1.11 \overline{TS} — Transfer Start

\overline{TS} is asserted by the current bus owner to indicate the start of a transaction on the external bus

\overline{TS} is only asserted for the first clock cycle of the transaction, and is negated in the successive clock cycles until the end of the transaction.

29.2.1.12 $\overline{\text{WE}}[0:3]$ — Write Enables 0-3

Write enables are used to enable program operations to a particular memory. $\overline{\text{WE}}[0:3]$ are only asserted for chip-select accesses.

For chip-select accesses to a 16-bit port, only $\overline{\text{WE}}[0:1]$ are used by the EBI, regardless of which half of the DATA bus is selected via the D16_31 bit in the EBI_MCR.

29.2.2 Signal Function and Direction by Mode

The EBI operating mode is configured using two fields in the EBI master control register (EBI_MCR): EXT_M and MDIS. Their settings determine which EBI signals are valid and the I/O direction. When a signal is configured for non-EBI function in the EBI_MCR, the EBI always negates the signal if the EBI controls the corresponding pad (determined by SIU configuration). Table 29-1 lists the function and direction of the external signals in each of the EBI modes of operation. The clock signals are not included because they are output only (from the FMPLL module) and are not affected by EBI modes. See Section 29.3.2.3, “EBI Module Configuration Register (EBI_MCR),” for details on the EXT_M and MDIS bits.

Table 29-1. Signal Function (*f*) According to EBI Mode Settings

Signal Name	Modes		
	Module Disable <i>f</i> EXT _M = <i>n</i> , MDIS = 1	Single Master <i>f</i> I/O Direction EXT _M = 0, MDIS = 0	External Master <i>f</i> I/O Direction EXT _M = 1, MDIS = 0
$\overline{\text{BDIP}}$	non-EBI function	Burst data in progress (Output) ¹	Not supported
$\overline{\text{CS}}[0:3]$	non-EBI function	Chip selects (Output) ¹	
AD[0:31]	non-EBI function	Multiplexed Address and Data bus (I/O)	
ADDR[8:15]	non-EBI function	Non-multiplexed Address (O)	
$\overline{\text{OE}}$	non-EBI function	Output enable (Output)	
RD_ $\overline{\text{WR}}$	non-EBI function	Read_Write (Output)	
$\overline{\text{TA}}$	non-EBI function	Transfer acknowledge (I/O)	
$\overline{\text{TEA}}$	non-EBI function	Transfer error acknowledge (I/O)	
$\overline{\text{TS}}$	non-EBI function	Transfer start (Output)	
ALE	non-EBI function	Address latch enable (Output)	
$\overline{\text{WE}}[0:3]$	non-EBI function	Write/Byte enables (Output) ¹	

- ¹ Although external master accesses can drive these pins, the EBI tri-states the pins and does not sample them for input.

29.2.3 Signal Pad Configuration by Mode

Depending on the mode of operation, many external signals must have their pads configured to operate as push/pull signals for correct system operation. This configuration is done in the system integration unit (SIU) module.

The open drain mode of the pads configuration module is not used for any EBI signals.

Table 29-2 shows how each EBI signal must have its pad configured prior to operating in each of the EBI modes. See Section 29.3.2.3, “EBI Module Configuration Register (EBI_MCR)” for details on the EXTM and MDIS bits.

Table 29-2. Required EBI Pad Configuration by Mode

Signal Name	Module Disable Mode ¹ (EXTM = X, MDIS = 1)	Single Master Mode (EXTM = 0, MDIS = 0)	External Master Mode (EXTM = 1, MDIS = 0)
$\overline{\text{BDIP}}$	X	Push/Pull	Not supported
$\overline{\text{CS}}[0:3]$	X	Push/Pull	
AD[0:31]	X	Push/Pull, Three-stateable	
ADDR[8:15]	X	Push/Pull	
$\overline{\text{OE}}$	X	Push/Pull	
RD $\overline{\text{WR}}$	X	Push/Pull	
$\overline{\text{TA}}$	X	Push/Pull, Three-stateable	
$\overline{\text{TEA}}$	X	Push/Pull, Three-stateable	
$\overline{\text{TS}}$	X	Push/Pull	
ALE	X	Push/Pull	
$\overline{\text{WE}}[0:3]$	X	Push/Pull	

¹ X indicates the pad configuration is a don't care because the signal is not used by the EBI in this mode.

29.3 Memory Map and Registers

This section provides a detailed description of all EBI registers.

29.3.1 Module Memory Map

The EBI memory map is shown in Table 29-3. The address of each register is given as an offset to the EBI base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 29-3. EBI Memory Map

Offset from EBI_BASE (0xFFFF_4000)	Register	Access	Reset Value	Section/Page
0x0000	EBI_MCR — EBI module configuration register	R/W ¹	0x0000_0804	29.3.2.3/29-10
0x0004	Reserved			
0x0008	EBI_TESR — EBI transfer error status register	R/W ¹	0x0000_0000	29.3.2.4/29-11
0x000c	EBI_BMCR — EBI bus monitor control register	R/W ¹	0x0000_FF80	29.3.2.5/29-12
0x0010	EBI_BR0 — EBI base register bank 0	R/W ¹	0x2000_0042	29.3.2.6/29-13
0x0014	EBI_OR0 — EBI option register bank 0	R/W ¹	0xE000_0000	29.3.2.7/29-15
0x0018	EBI_BR1 — EBI base register bank 1	R/W ¹	0x2000_0042	29.3.2.6/29-13
0x001C	EBI_OR1 — EBI option register bank 1	R/W ¹	0xE000_0000	29.3.2.7/29-15
0x0020	EBI_BR2 — EBI base register bank 2	R/W ¹	0x2000_0042	29.3.2.6/29-13
0x0024	EBI_OR2 — EBI option register bank2	R/W ¹	0xE000_0000	29.3.2.7/29-15
0x0028	EBI_BR3 — EBI base register bank 3	R/W ¹	0x2000_0042	29.3.2.6/29-13
0x002C	EBI_OR3 — EBI option register bank 3	R/W ¹	0xE000_0000	29.3.2.7/29-15
0x0020–0x005C	Reserved			

¹ All bits may not be writeable. See register description.

29.3.2 Register Descriptions

This section lists some special considerations for EBI registers and then the EBI registers in address order, describing the registers and their bit fields.

29.3.2.1 Writing EBI Registers While a Transaction is in Progress

Other than the exceptions noted below, EBI registers must not be written while a transaction to the EBI (from internal master) is in progress (or within 2 CLKOUT cycles after a transaction has just completed, to allow internal state machines to go IDLE). In such cases, the behavior is undefined.

Exceptions that can be written while an EBI transaction is in progress are the following:

- All bits in EBI_TESR

If code in external memory needs to write EBI registers, this must be done in a way that avoids modifying EBI registers while external accesses are being performed, such as the following method:

- Copy the code that is doing the register writes (plus a return branch) to internal SRAM
- Branch to internal SRAM to run this code, ending with a branch back to external flash

29.3.2.2 Separate Input Clock for Registers

The EBI registers are accessed with a clock signal separate from the clock used by the rest of the EBI. In module disable mode, the clock used by the non-register portion of the EBI is disabled to reduce power

consumption. The clock signal dedicated to the registers, however, allows access to the registers even while the EBI is in the module disable mode. Flag bits in the EBI transfer error status register (EBI_TESR), however, are set and cleared with the clock used by the non-register portion of the EBI. Consequently, in module disable mode, the EBI_TESR does not have a clock signal and is therefore not writable.

29.3.2.3 EBI Module Configuration Register (EBI_MCR)

The EBI_MCR contains bits that configure various attributes associated with EBI operation.

Offset: 0x0000 Access: User read/write

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ACG E	EXT M	EARB	0	1	0	0	0	0	MDIS	0	0	0	D16_ 31	AD_ MUX	DBM
W																
Reset	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0

Figure 29-2. EBI Module Configuration Register (EBI_MCR)

Table 29-4. EBI_MCR Field Descriptions

Field	Description
bits 0–15	Reserved.
ACGE	Automatic CLKOUT Gating Enable. Enables the EBI feature of turning off CLKOUT (holding it high) during idle periods in-between external bus accesses. 0 Automatic CLKOUT gating is disabled. 1 Automatic CLKOUT gating is enabled.
EXTM	External Master Mode. 0 External master mode is inactive (single master mode). 1 External master mode is active. Note: EXTM=1 is not supported on MPC5510.
EARB	External Arbitration. When EXTM = 0, the EARB bit is a don't care, and is treated as 0. 0 Internal arbitration is used. 1 External arbitration is used.
bits 19–24	Reserved. Note: Reserved bits 19-20 are writeable, but writing to these bits have no effect other than to update the value of the register. For future compatibility, this bit should be written to zero.
MDIS	Module Disable Mode. Allows the clock to be stopped to the non-memory mapped logic in the EBI, effectively putting the EBI in a software controlled power-saving state. See Section 29.1.3.3, “Module Disable Mode,” for more information. No external bus accesses can be performed when the EBI is in module disable mode (MDIS = 1). 0 Module disable mode is inactive. 1 Module disable mode is active.
bits 26–28	Reserved.

Table 29-4. EBI_MCR Field Descriptions (continued)

Field	Description
D16_31	Data Bus 16_31 Select. The D16_31 bit controls whether the EBI uses the AD[0:15] or AD[16:31] signals, when in 16-bit data bus mode (DBM=1) or for chip-select accesses to a 16-bit port (PS=1). For systems using A/D muxing with a 16-bit port, it is recommended to set D16_31 to 1. 0 AD[0:15] signals are used for 16-bit port accesses 1 AD[16:31] signals are used for 16-bit port accesses
AD_MUX	Address on Data Bus Multiplexing Mode. The AD_MUX bit controls whether non-chip-select accesses have the address driven on the data bus in the address phase of a cycle. 0 Only data on data pins for non-CS accesses. 1 Address on data multiplexing mode is used for non-CS accesses.
DBM	Data Bus Mode. Controls whether the EBI is in 32-bit or 16-bit data bus mode. On MPC5510, the default value of DBM is 0. 0 32-bit data bus mode is used. 1 16-bit data bus mode is used.

29.3.2.4 EBI Transfer Error Status Register (EBI_TESR)

The EBI_TESR contains a bit for each type of transfer error on the external bus. A bit set to logic 1 indicates what type of transfer error occurred since the last time the bits were cleared. Each bit can be cleared by reset or by writing a 1 to it. Writing a 0 has no effect.

This register is not writable in module disable mode due to the use of power saving clock modes.

Offset: 0x0008

Access: User read/write to clear

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TEAF	BMTF
W															w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-3. EBI Transfer Error Status Register (EBI_TESR)

Table 29-5. EBI_TESR Field Descriptions

Field	Description
bits 0–29	Reserved.

Table 29-5. EBI_TESR Field Descriptions (continued)

Field	Description
TEAF	Transfer Error Acknowledge Flag. Set if the cycle was terminated by an externally generated \overline{TEA} signal. 0 No error. 1 External \overline{TEA} occurred. This bit can be cleared by writing a 1 to it.
BMTF	Bus Monitor Timeout Flag. Set if the cycle was terminated by a bus monitor timeout. 0 No error. 1 Bus monitor timeout occurred. This bit can be cleared by writing a 1 to it.

29.3.2.5 EBI Bus Monitor Control Register (EBI_BMCR)

The EBI_BMCR controls the timeout period of the bus monitor and whether it is enabled or disabled.

Offset: 0x000C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BMT								BME	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0

Figure 29-4. EBI Bus Monitor Control Register (EBI_BMCR)

Table 29-6. EBI_BMCR Field Descriptions

Field	Description
bits 0–15	Reserved.
BMT	Bus Monitor Timing. Defines the timeout period, in eight external bus clock resolution, for the bus monitor. See Section 29.4.1.4, “Bus Monitor,” for more details on bus monitor operation. $\text{Timeout Period} = \frac{2 + (8 \times \text{BMT})}{\text{External Bus Clock Frequency}}$
BME	Bus Monitor Enable. Controls whether the bus monitor is enabled for internal to external bus cycles. The BME bit is ignored (treated as 0) for chip-select accesses with internal \overline{TA} (SETA=0). 0 Disable bus monitor. 1 Enable bus monitor (for external \overline{TA} accesses only).
bits 25–31	Reserved.

29.3.2.6 EBI Base Registers 0–3 (EBI_BR n)

The EBI_BR n are used to define the base address and other attributes for the corresponding chip select.

Offset: 0x0010 (EBI_BR0)
 0x0018 (EBI_BR1)
 0x0020 (EBI_BR2)
 0x0028 (EBI_BR3)

Access: User read/write

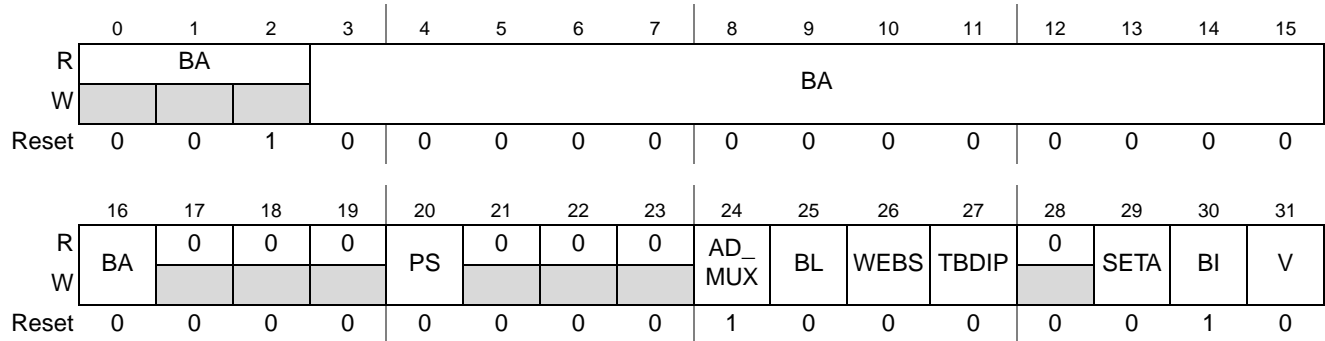


Figure 29-5. EBI Base Registers 0–3 (EBI_BR n)

Table 29-7. EBI_BR n Field Descriptions

Field	Description
BA	Base Address. Compared to the corresponding unmasked address signals among ADDR[0:16] of the internal address bus to determine if a memory bank controlled by the memory controller is being accessed by an internal bus master. Note: The upper 3 bits of the BA field, EBI_BR n [0:2] are tied to a fixed value of 001. These bits can be read but not written. They are ignored by the EBI during the chip-select address comparison
bits 17–19	Reserved.
PS	Port Size. Determines the data bus width of transactions to this chip select bank. 0 32-bit port. 1 16-bit port. Note: If EBI_MCR[DBM] is set for 16-bit data bus mode, the PS bit value is ignored and is always treated as a 1 (16-bit port).
bits 21–23	Reserved.
AD_MUX	Address on Data Bus Multiplexing. The AD_MUX bit controls whether accesses for this chip select have the address driven on the data bus in the address phase of a cycle. On MPC5510, the default value of AD_MUX is 1. 0 Address on data multiplexing mode is disabled for this chip select. 1 Address on data multiplexing mode is enabled for this chip select.

Table 29-7. EBI_BRn Field Descriptions (continued)

Field	Description										
BL	<p>Burst Length. This bit is writable, but has no functional effect on MPC5510. Because the MPC5510 uses a 32-bit data bus width (as opposed to 64-bit), all burst transfers use a 4-word length, regardless of the BL bit value. The number of beats in a burst is automatically determined by the EBI to be 4 or 8 according to the port size (PS bit) so that the burst fetches 4 32-bit words.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Burst Length¹</th> <th>PS</th> <th># Beats in Burst²</th> </tr> </thead> <tbody> <tr> <td rowspan="2">X³</td> <td rowspan="2">4-word⁴</td> <td>0 (32-bit)</td> <td>4</td> </tr> <tr> <td>1 (16-bit)</td> <td>8</td> </tr> </tbody> </table> <p>¹ Total amount of data fetched in a burst transfer. ² Number of external data beats used in external burst transfer. The size of each beat is determined by PS value. ³ An 8-word burst length is only supported for SoC's using 64-bit data bus width. Because the MPC5510 uses a 32-bit data bus width, the value of the BL bit is a don't care, and all burst transfers use a 4-word length. ⁴ A word always refers to 32-bits of data, regardless of PS.</p> <p>Note: The EBI does NOT support a 2-word external burst length. This means that neither a 4-beat burst to a 16-bit external memory (nor a 2-beat burst to 32-bit external memory) are supported.</p>	Value	Burst Length ¹	PS	# Beats in Burst ²	X ³	4-word ⁴	0 (32-bit)	4	1 (16-bit)	8
Value	Burst Length ¹	PS	# Beats in Burst ²								
X ³	4-word ⁴	0 (32-bit)	4								
		1 (16-bit)	8								
WEBS	<p>Write Enable/Byte Select. Controls the functionality of the $\overline{WE}[0:3]$ signals.</p> <p>0 The $\overline{WE}[0:3]$ signals function as write enable. 1 The $\overline{WE}[0:3]$ signals function as byte enable.</p>										
TBDIP	<p>Toggle Burst Data in Progress. Determines how long the \overline{BDIP} signal is asserted for each data beat in a burst cycle. See Section 29.4.2.5.1, "TBDIP Effect on Burst Transfer," for details.</p> <p>0 Assert \overline{BDIP} throughout the burst cycle, regardless of wait state configuration. 1 Assert \overline{BDIP} (BSCY + 1) external bus cycles only before expecting subsequent burst data beats.</p>										
bit 28	Reserved.										
SETA	<p>Select External Transfer Acknowledge. The SETA bit controls whether accesses for this chip select will terminate (end transfer without error) based on externally asserted \overline{TA} or internally asserted \overline{TA}. SETA should only be set when the BI bit is 1 as well, since burst accesses with SETA=1 are not supported. Setting SETA=1 causes the BI bit to be ignored (treated as 1, burst inhibited).</p> <p>0 Transfer acknowledge (\overline{TA}) is an output from the EBI, data phase will be terminated by the EBI. 1 Transfer acknowledge (\overline{TA}) is an input to the EBI, data phase will be terminated by an external device.</p>										
BI	<p>Burst Inhibit. Determines whether or not burst read accesses are allowed for this chip-select bank. The BI bit is ignored (treated as 1) for chip-select accesses with external TA (SETA=1).</p> <p>0 Enable burst accesses for this bank. 1 Disable burst accesses for this bank. This is the default value out of reset (or when SETA=1).</p>										
V	<p>Valid Bit. Indicates that the contents of this base register and option register pair are valid. The appropriate \overline{CS} signal does not assert unless the corresponding V-bit is set.</p> <p>0 This bank is not valid. 1 This bank is valid.</p>										

29.3.2.7 EBI Option Registers 0–3 (EBI_OR n)

The EBI_OR n registers are used to define the address mask and other attributes for the corresponding chip select.

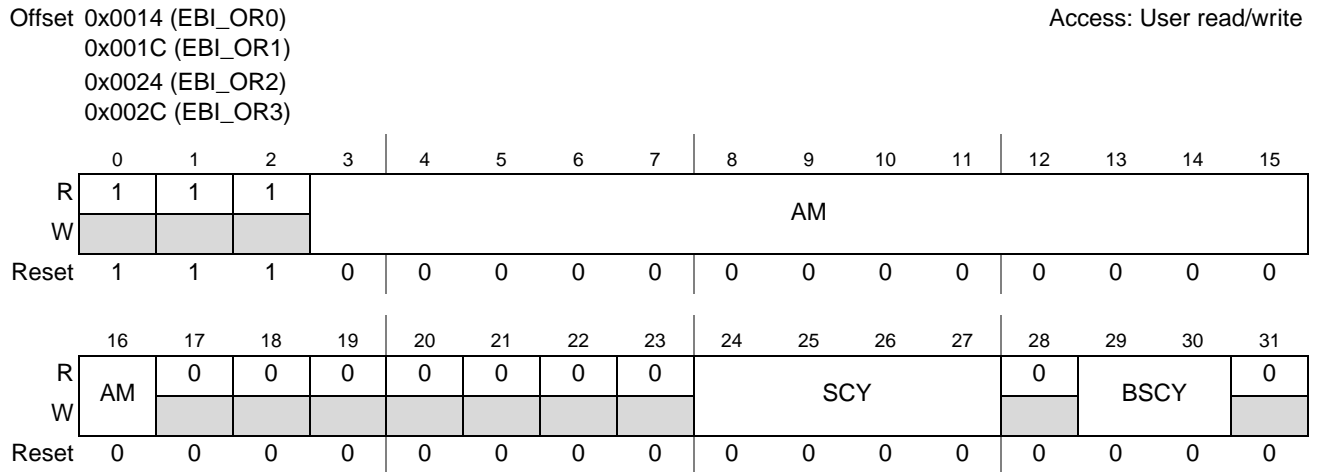


Figure 29-6. EBI Option Registers 0–3 (EBI_OR n)

Table 29-8. EBI_OR n

Field	Description
AM	Address Mask. Allows masking of any corresponding bits in the associated base register. Masking the address independently allows external devices of different size address ranges to be used. Any clear bit masks the corresponding address bit. Any set bit causes the corresponding address bit to be used in comparison with the address pins. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. This field can be read or written at any time. Note: The upper three bits of the address mask (AM) field, EBI_ORx[0:2] are tied to a fixed value of 0b111. These bits reset to their fixed value.
bits 17–23	Reserved.
SCY	Cycle Length in Clocks. Represents the number of wait states (external bus cycles) inserted after the address phase in the single cycle case, or in the first beat of a burst, when the memory controller handles the external memory access. Values range from 0 to 15. This is the main parameter for determining the length of the cycle. These bits are ignored when SETA=1. The total cycle length for the first beat (including the \overline{TS} cycle): (2 + SCY) external clock cycles See Section 29.5.3.1, “Example Wait State Calculation” .

Table 29-8. EBI_OR n (continued)

Field	Description
bit 28	Reserved.
BSCY	<p>Burst Beats Length in Clocks. This field determines the number of wait states (external bus cycles) inserted in all burst beats except the first, when the memory controller starts handling the external memory access and thus is using SCY[0:3] to determine the length of the first beat. These bits are ignored when SETA=1</p> <ul style="list-style-type: none"> Total memory access length for each beat: $(1 + \text{BSCY}) \text{ External Clock Cycles}$ Total cycle length (including the $\overline{\text{TS}}$ cycle): $(2 + \text{SCY}) + [(\text{Number of Beats} - 1) \times (\text{BSCY} + 1)]$ <p>Note: The number of beats (4, 8, 16) is determined by BL and PS bits in the base register.</p> <p>00 0-clock cycle wait states (1 clock per data beat) 01 1-clock cycle wait states (2 clocks per data beat) 10 2-clock cycle wait states (3 clocks per data beat) 11 3-clock cycle wait states (4 clocks per data beat)</p>

29.4 Functional Description

29.4.1 External Bus Interface Features

29.4.1.1 Multiplexed 32-bit Address/Data Bus (Single Master)

This is the default mode of operation for MPC5510. See [Section 29.1.3.6, “Multiplexed Address on Data Bus Mode.”](#) A 16-bit data bus mode is available via the DBM bit in EBI_MCR. See [Section 29.1.3.5, “16-Bit Data Bus Mode.”](#)

29.4.1.2 Memory Controller with Support for Various Memory Types

The EBI contains a memory controller that supports a variety of memory types, including synchronous burst mode flash and external SRAM, and asynchronous/legacy flash and external SRAM with a compatible interface.

Each $\overline{\text{CS}}$ bank is configured via its own pair of base and option registers. Each time an internal to external bus cycle access is requested, the internal address is compared with the base address of each valid base register (with 17 bits having mask). See [Figure 29-7](#). If a match is found, the attributes defined for this bank in its BR and OR are used to control the memory access. If a match is found in more than one bank, the lowest bank matched handles the memory access. For example, bank 0 is selected over bank 1.

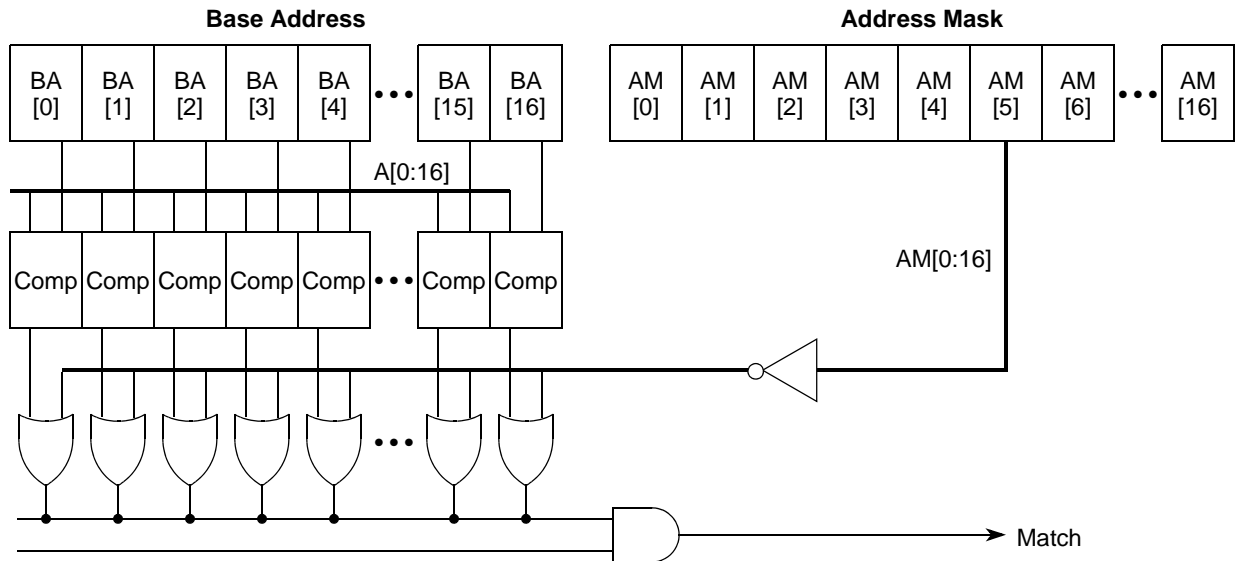


Figure 29-7. Bank Base Address and Match Structure

When a match is found on one of the chip select banks, all its attributes (from the appropriate base and option registers) are selected for the functional operation of the external memory access, such as:

- Number of wait states for a single-memory access, and for any beat in a burst access
- Burst enable
- Port size for the external accessed device

See [Section 29.3.2.6, “EBI Base Registers 0–3 \(EBI_BRn\),”](#) and [Section 29.3.2.7, “EBI Option Registers 0–3 \(EBI_ORn\),”](#) for a full description of all chip select attributes.

When no match is found on any of the chip select banks, the default transfer attributes shown in [Table 29-9](#) are used.

Table 29-9. Default Attributes for Non-Chip Select Transfers

CS Attribute	Default Value	Comment
PS	0	32-bit port size
BL	0	Burst length is don't care because burst is disabled
WEBS	0	Write enables
TBDIP	0	Don't care since burst is disabled
BI	1	Burst inhibited
SCY	0	Don't care since external \overline{TA} is used
BSCY	0	Don't care since external \overline{TA} is used
AD_MUX	0	Address on data multiplexing
SETA	1	Select external \overline{TA} to terminate access

29.4.1.3 Burst Support (Wrapped Only)

The EBI supports burst read accesses of external burstable memory. To enable bursts to a particular memory region, clear the burst inhibit (BI) bit in the appropriate base register. All external bursts use a 4-word burst length on MPC5510. See [Section 29.4.2.5, “Burst Transfer,”](#) for more details.

In 16-bit data bus mode (EBI_MCR[DBM]=1), a special 2-beat burst case is supported for reads and writes for 32-bit non-chip select accesses only. This allows 32-bit coherent accesses to another MCU. See [Section 29.4.2.10, “Non-Chip-Select Burst in 16-bit Data Bus Mode.”](#)

Bursting of accesses that are not controlled by the chip selects is not supported for any other case besides the special case of 32-bit accesses in 16-bit data bus mode.

Burst writes are not supported for any other case besides the special case of 32-bit non-chip select writes in 16-bit data bus mode. Internal requests to write more than 32 bits (such as a cache line) externally are broken up into separate 32-bit or 16-bit external transactions according to the port size. See [Section 29.4.2.6, “Small Accesses \(Small Port Size and Short Burst Length\)”](#) for more detail on these cases.

29.4.1.4 Bus Monitor

When enabled (via the BME bit in the EBI_BMCR), the bus monitor detects when no $\overline{\text{TA}}$ assertion is received within a maximum timeout period for non-chip select accesses (that is, accesses that use external $\overline{\text{TA}}$). The timeout for the bus monitor is specified by the BMT field in the EBI_BMCR. Each time a timeout error occurs, the BMTF bit is set in the EBI_TESR. The timeout period is measured in external bus (CLKOUT) cycles. Thus the effective real-time period is multiplied (by 2, 3, etc) when a slower-speed mode is used, even though the BMT field itself is unchanged.

29.4.1.5 Port Size Configuration Per Chip Select (16 or 32 Bits)

The EBI supports memories with data widths of 16 or 32 bits. The port size for a particular chip select is configured by writing the PS bit in the corresponding base register.

29.4.1.6 Configurable Wait States

From 0 to 15 wait states can be programmed for any cycle that the memory controller generates, via the SCY bits in the appropriate option register. From zero to three wait states between burst beats can be programmed using the BSCY bits in the appropriate option register.

29.4.1.7 Configurable Internal or External $\overline{\text{TA}}$ per chip select

Each chip select can be configured (via the SETA bit) to have $\overline{\text{TA}}$ driven internally (by the EBI), or externally (by an external device). See [Section 29.3.2.6, “EBI Base Registers 0–3 \(EBI_BRn\),”](#) for more details on SETA bit usage.

29.4.1.8 Four Chip Select ($\overline{\text{CS}}[0:3]$) Signals

The EBI contains four chip select signals, controlling four independent memory banks. See [Section 29.4.1.2, “Memory Controller with Support for Various Memory Types,”](#) for more details on chip select bank configuration.

29.4.1.9 Four Write/Byte Enable ($\overline{\text{WE}}$) Signals

The functionality of the $\overline{\text{WE}}[0:3]$ signals depends on the value of the WEBS bit in the corresponding base register. Setting WEBS to 1 configures these pins as byte enable ($\overline{\text{BE}}[0:3]$), but resetting it to 0 configures them as write enable ($\overline{\text{WE}}[0:3]$). $\overline{\text{WE}}[0:3]$ are asserted during write accesses only, while $\overline{\text{BE}}[0:3]$ are asserted for both read and write accesses. The timing of the $\overline{\text{WE}}[0:3]$ signals remains the same in either case.

The upper write/byte enable ($\overline{\text{WE}}0$) indicates that the upper eight bits of the data bus (AD[0:7]) contain valid data during a write/read cycle. The upper middle write/byte enable ($\overline{\text{WE}}1$) indicates that the upper middle eight bits of the data bus (AD[8:15]) contain valid data during a write/read cycle. The lower middle write/byte enable ($\overline{\text{WE}}2$) indicates that the lower middle eight bits of the data bus (AD[16:23]) contain valid data during a write/read cycle. The lower write/byte enable ($\overline{\text{WE}}3$) indicates that the lower eight bits of the data bus (AD[24:31]) contain valid data during a write/read cycle.

NOTE

The exception to the preceding $\overline{\text{WE}}$ description is that for 16-bit port transfers (DBM=1 or PS=1), only the $\overline{\text{WE}}[0:1]$ signals are used, regardless of whether AD[0:15] or AD[16:31] are selected (via the D16_31 bit in the EBI_MCR). This means if AD[16:31] are selected, $\overline{\text{WE}}0$ indicates that AD[16:23] contains valid data, and $\overline{\text{WE}}1$ indicates that AD[24:31] contains valid data.

The write/byte enable lines affected in a transaction for a 32-bit port (PS = 0) and a 16-bit port (PS = 1) are shown in [Table 29-10](#). Only big endian byte ordering is supported by the EBI.

Table 29-10. Write/Byte Enable Signals Function¹

(X indicates that valid data is transferred on these bits)

Transfer Size	Address		32-Bit Port Size				16-Bit Port Size ²			
	A30	A31	$\overline{\text{WE}}0$	$\overline{\text{WE}}1$	$\overline{\text{WE}}2$	$\overline{\text{WE}}3$	$\overline{\text{WE}}0$	$\overline{\text{WE}}1$	$\overline{\text{WE}}2$	$\overline{\text{WE}}3$
Byte	0	0	X	—	—	—	X	—	—	—
	0	1	—	X	—	—	—	X	—	—
	1	0	—	—	X	—	X	—	—	—
	1	1	—	—	—	X	—	X	—	—
16-bit	0	0	X	X	—	—	X	X	—	—
	1	0	—	—	X	X	X	X	—	—
32-bit	0	0	X	X	X	X	X ³	X ²	—	—
Burst	0	0	X	X	X	X	X	X	—	—

- ¹ This table applies to aligned internal master transfers only. In the case of a misaligned internal master transfer that is split into multiple aligned external transfers, not all write enables X'd in the table will necessarily assert. See [Section 29.4.1.13, “Misaligned Access Support.”](#)
- ² Also applies when DBM=1 for 16-bit data bus mode.
- ³ This case consists of two 16-bit external transactions, but for both transactions the $\overline{WE}[0:1]$ signals are the only \overline{WE} signals affected.

29.4.1.10 Configurable Bus Speed Clock Modes

The EBI supports configurable bus speed clock modes. Refer to [Section 29.1.3.4, “Configurable Bus Speed Modes,”](#) for more details on this feature.

29.4.1.11 Stop and Module Disable Modes for Power Savings

See [Section 29.1.3, “Modes of Operation,”](#) for a description of the power saving modes.

29.4.1.12 Optional Automatic CLKOUT Gating

The EBI has the ability to hold the external CLKOUT pin high when the EBI's internal master state machine is idle and no requests are pending. The EBI outputs a signal to the pads logic in the MCU to disable CLKOUT. This feature is disabled out of reset, and can be enabled or disabled by the ACGE bit in the EBI_MCR.

29.4.1.13 Misaligned Access Support

The EBI has limited misaligned access support. Misaligned non-burst chip-select transfers from internal masters are supported. The EBI aligns the accesses when it sends them out to the external bus (splitting them into multiple aligned accesses if necessary), so that external devices are not required to support misaligned accesses. Burst accesses (internal master) must be 32-bit aligned.

29.4.1.13.1 Misaligned Access Support (32-bit)

[Table 29-11](#) shows all the misaligned access cases supported by the EBI (using a 32-bit implementation), as seen on the internal master bus. All other misaligned cases are not supported. If an unsupported misaligned access to the EBI is attempted (such as non-chip-select or burst misaligned access), the EBI errors the access on the internal bus and does not start the access (nor assert \overline{TEA}) externally.

Table 29-11. Misalignment Cases Supported by a 32-bit EBI (internal bus)

# ¹	Program Size and byte offset	Address [30:31] ^{2,3}	Data Bus Byte Strokes ⁴	H SIZE ⁵	H UNALIGN ⁶
1	Half @0x1	01	0110	10	1
4 —	Half @0x3 (2 AHB transfers)	11 z00	0001 1000	01 ⁷ 00	1 0
8	Word @0x1 (2 AHB transfers)	01	0111 1000	10 00	1 0

Table 29-11. Misalignment Cases Supported by a 32-bit EBI (internal bus) (continued)

# ¹	Program Size and byte offset	Address [30:31] ^{2,3}	Data Bus Byte Strokes ⁴	HSIZE ⁵	HUNALIGN ⁶
9	Word @0x2 (2 AHB transfers)	10	0011 1100	10 01	1 0
10 11	Word @0x3 (2 AHB transfers)	11 z00	0001 1110	10 ⁷ 10	1 1

¹ Misaligned case number. Only transfers where HUNALIGN=1 are numbered as misaligned cases. The missing case numbers cannot occur on a 32-bit implementation.

² Address on internal master AHB bus, not necessarily address on external ADDR pins.

³ Address Z is incremented by one 32-bit word compared to previous access on the bus.

⁴ Internal byte strobe signals on AHB bus. Shown with big-endian byte ordering in this table, even though internal master AHB bus uses little-endian byte ordering (EBI flips order internally).

⁵ Internal signal on AHB bus; 00=8 bits, 01=16 bits, 10=32 bits. HSIZE is driven according to the smallest aligned container that contains all the requested bytes. This results in extra EBI external transfers in some cases.

⁶ Internal signal on AHB bus that indicates that this transfer is misaligned (when 1).

⁷ For this case, the EBI internally treats HSIZE as 00 (1-byte access).

Table 29-12 shows which external transfers are generated by the EBI for the misaligned access cases in Table 29-11, for each port size.

The number of external transfers for each internal AHB master request is determined by the HSIZE value for that request relative to the port size. For example, a halfword write to 0x3 (misaligned case #4) with 16-bit port size results in four external 16-bit transfers because the transfer granularity of 32 bits. For cases where two or more external transfers are required for one internal transfer request, these external accesses are considered part of a small access set, as described in Section 29.4.2.6, “Small Accesses (Small Port Size and Short Burst Length).”

Because all transfers are aligned on the external bus, normal timing diagrams and protocol apply.

Table 29-12. Misalignment Cases Supported by a 32-bit EBI (external bus)

# ¹	PS ²	Program Size and byte offset	ADDR[30:31] ^{3,4}	$\overline{WE}[0:3]$ ⁵
1	0	Half @0x1	00	1001
	1		00 10	1011 0111
4	0	Half @0x3 (2 AHB transfers)	11 ⁶	1110
—			z00	0111
4	1		11 ⁶	1011
—			z00	0111

Table 29-12. Misalignment Cases Supported by a 32-bit EBI (external bus) (continued)

# ¹	PS ²	Program Size and byte offset	ADDR[30:31] ^{3,4}	$\overline{WE}[0:3]$ ⁵
8	0	Word @0x1 (2 AHB transfers)	00	1000
—			z00	0111
8	1		00	1011
—			10	0011
			z00	0111
9	0	Word @0x2 (2 AHB transfers)	00	1100
—			z00	0011
9	1		10	0011
—			z00	0011
10	0	Word @0x3 (2 AHB transfers)	11 ⁶	1110
11			z00	0001
10	1		11 ⁶	1011
11			z00 z10	0011 0111

¹ Misaligned case number, from Table 29-11.

² Port size; 0=32 bits, 1=16 bits.

³ External ADDR pins, not necessarily the address on internal master AHB bus.

⁴ For address with Z — address bit 29 will increment to next word. For all other cases, address bit 29 will be unchanged.

⁵ External \overline{WE} pins. Note that these pins have negative polarity, opposite of the internal byte strobes in Table 29-11.

⁶ Treated as 1-byte access.

29.4.2 External Bus Operations

The following sections provide a functional description of the external bus, the bus cycles provided for data transfer operations, bus arbitration, and error conditions.

29.4.2.1 External Clocking

The CLKOUT signal sets the frequency of operation for the bus interface directly. Internally, the MCU uses a phase-locked loop (PLL) circuit to generate a master clock for all of the MCU circuitry (including the EBI) that is phase-locked to the CLKOUT signal. In general, all signals for the EBI are specified with respect to the rising-edge of the CLKOUT signal, and they are guaranteed to be sampled as inputs or changed as outputs with respect to that edge.

29.4.2.2 Reset

Upon detection of internal reset, the EBI immediately terminates all transactions.

29.4.2.3 Basic Transfer Protocol

The basic transfer protocol defines the sequence of actions that must occur on the external bus to perform a complete bus transaction. A simplified scheme of the basic transfer protocol is shown in [Figure 29-8](#).

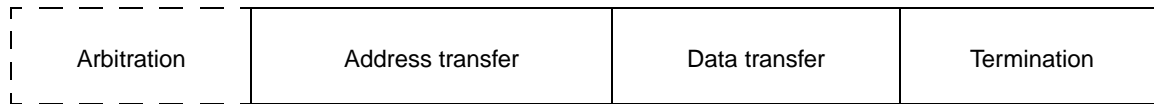


Figure 29-8. Basic Transfer Protocol

During the arbitration phase, ownership is requested and granted. This phase is not needed in single-master mode because the EBI is the permanent bus owner in this mode.

The address transfer phase specifies the address for the transaction and the transfer attributes that describe the transaction. The signals related to the address transfer phase are \overline{TS} , ADDR, $\overline{CS}[0:3]$, RD_ \overline{WR} , and \overline{BDIP} . The address and its related signals (with the exception of \overline{TS} , \overline{BDIP}) are driven on the bus with the assertion of the \overline{TS} signal, and kept valid until the bus master receives \overline{TA} asserted (the EBI holds them one cycle beyond \overline{TA} for writes and external \overline{TA} accesses). Note that for writes with internal \overline{TA} , RD_ \overline{WR} is not held one cycle past \overline{TA} .

The data transfer phase performs the transfer of data, from master to slave (in write cycles) or from slave to master (on read cycles), if any is to be transferred. The data phase may transfer a single beat of data (1-4 bytes) for non-burst operations or a 2-beat (special EBI_MCR[DBM]=1 case only), 4-beat, 8-beat, or 16-beat burst of data (2 or 4 bytes per beat depending on port size) when burst is enabled. On a write cycle, the master must not drive write data until after the address transfer phase is complete. This is to avoid electrical contentions when switching between drivers. The master must start driving write data one cycle after the address transfer cycle. The master can stop driving the data bus as soon as it samples the \overline{TA} line asserted on the rising edge of CLKOUT. To facilitate asynchronous write support, the EBI keeps driving valid write data on the data bus until 1 clock after the rising edge where RD_ \overline{WR} (and \overline{WE} for chip-select accesses) are negated. See [Figure 29-14](#) for an example of write timing. On a read cycle, the master accepts the data bus contents as valid on the rising edge of the CLKOUT in which the \overline{TA} signal is sampled asserted. See [Figure 29-10](#) for an example of read timing.

The termination phase is where the cycle is terminated by the assertion of either \overline{TA} (normal termination) or \overline{TEA} (termination with error). Termination is discussed in detail in [Section 29.4.2.9, “Termination Signals Protocol.”](#)

29.4.2.4 Single Beat Transfer

The flow and timing diagrams in this section assume that the EBI is configured in single master mode.

29.4.2.4.1 Single Beat Read Flow

The handshakes for a single beat read cycle are illustrated in the following flow and timing diagrams.

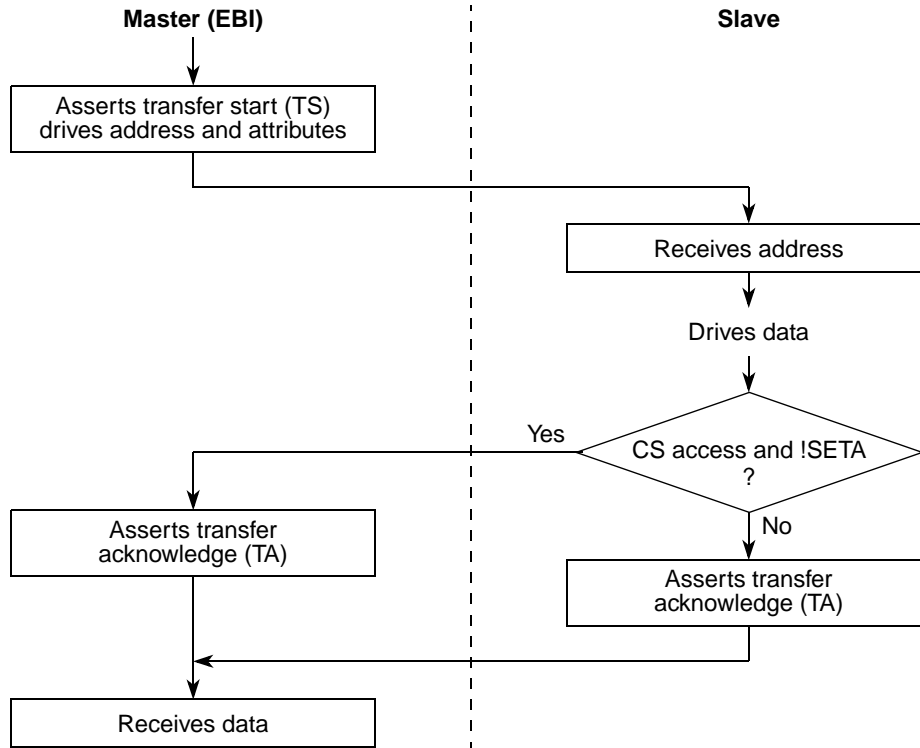


Figure 29-9. Basic Flow Diagram of a Single Beat Read Cycle

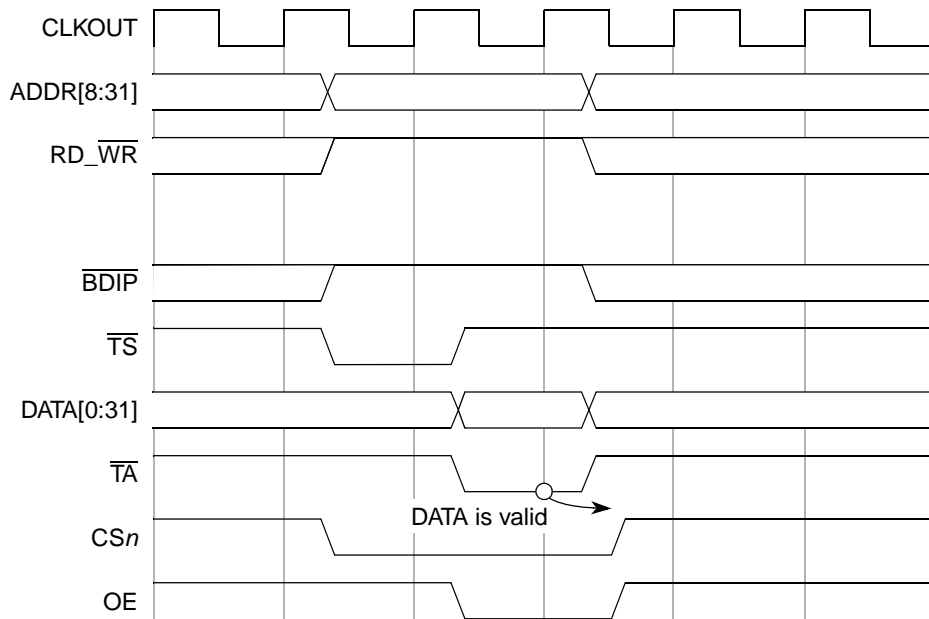


Figure 29-10. Single Beat 32-Bit Read Cycle, CS Access, Zero Wait States

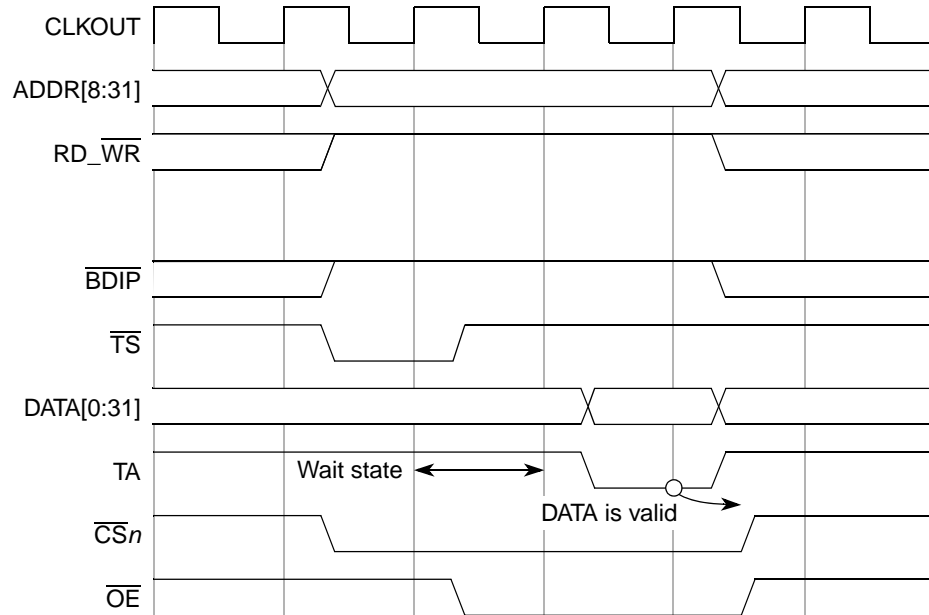
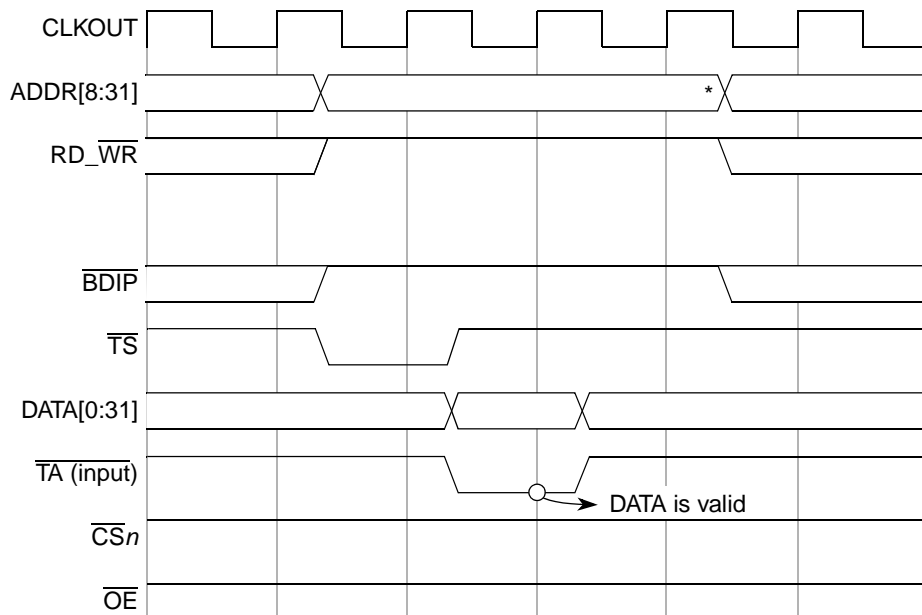


Figure 29-11. Single Beat 32-Bit Read Cycle, \overline{CS} Access, One Wait State



* The EBI drives address and control signals an extra cycle because it uses a latched version of the external \overline{TA} (1 cycle delayed) to terminate the cycle.

Figure 29-12. Single Beat 32-Bit Read Cycle, Non- \overline{CS} Access, Zero Wait States

29.4.2.4.2 Single Beat Write Flow

The handshakes for a single beat write cycle are illustrated in the following flow and timing diagrams.

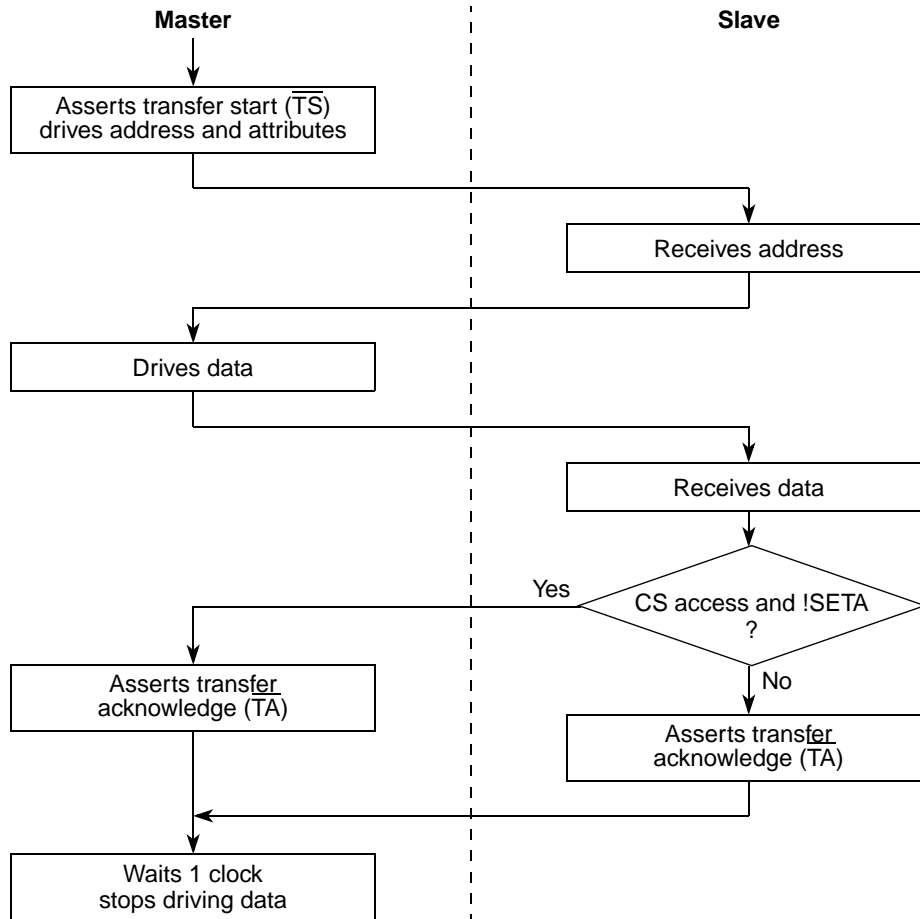


Figure 29-13. Basic Flow Diagram of a Single Beat Write Cycle

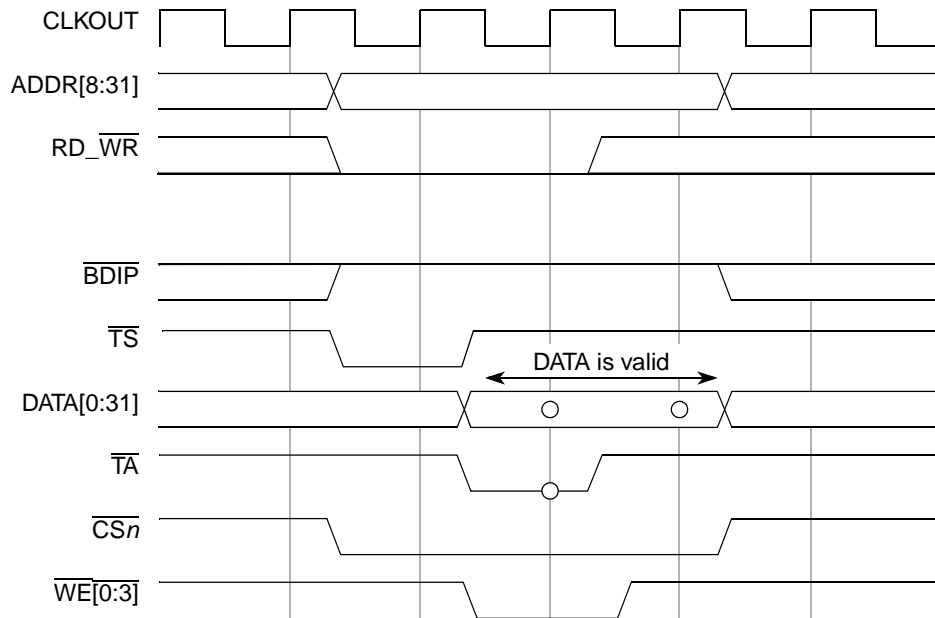


Figure 29-14. Single Beat 32-Bit Write Cycle, $\bar{C}S$ Access, Zero Wait States

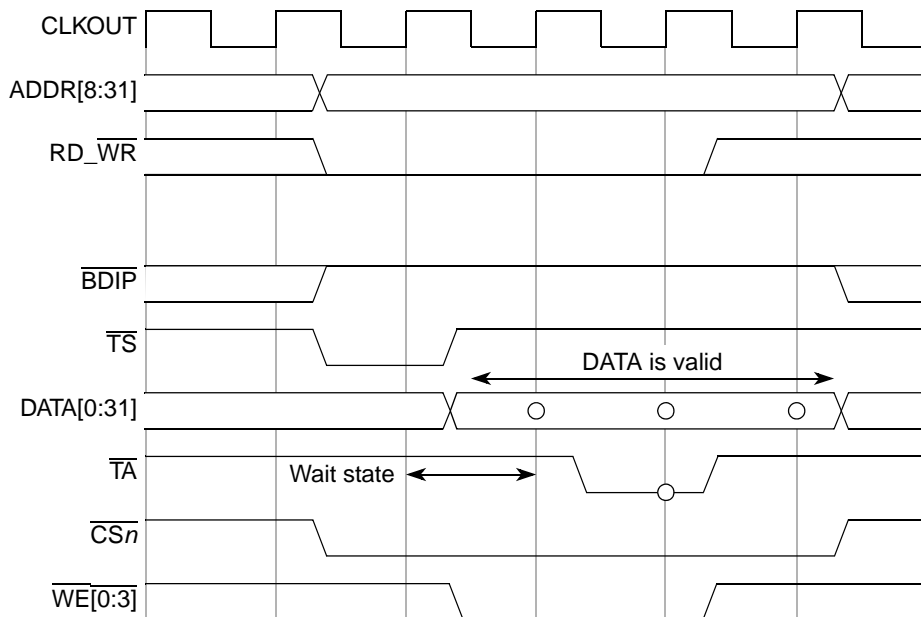
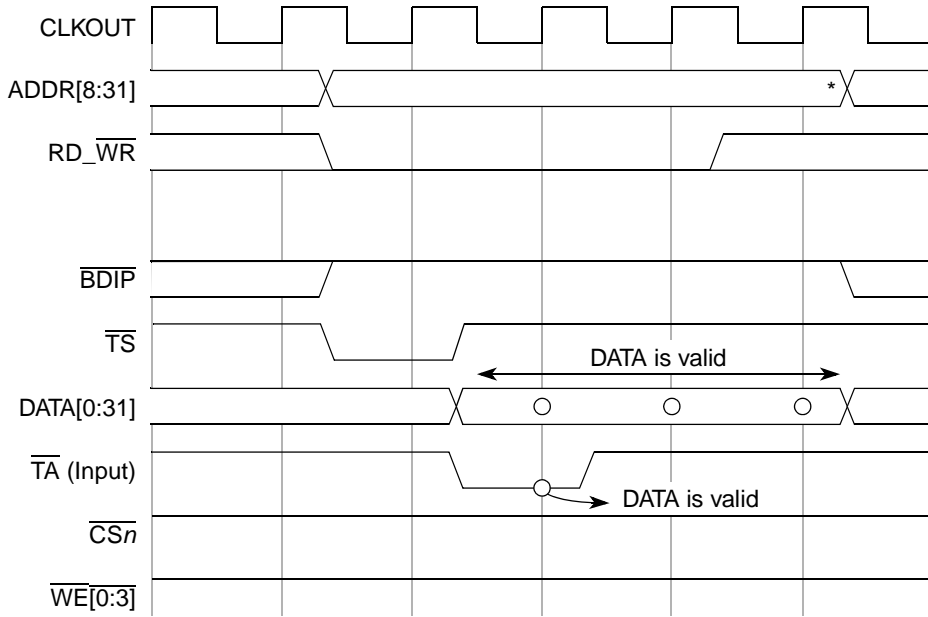


Figure 29-15. Single Beat 32-Bit Write Cycle, $\bar{C}S$ Access, One Wait State



* The EBI drives address and control signals an extra cycle because it uses a latched version of the external TA (1 cycle delayed) to terminate the cycle.

Figure 29-16. Single Beat 32-bit Write Cycle, Non- \overline{CS} Access, Zero Wait States

29.4.2.4.3 Back-to-Back Accesses

Due to internal bus protocol, one dead cycle is necessary between back-to-back external bus accesses that are not part of a set of small accesses (see Section 29.4.2.6, “Small Accesses (Small Port Size and Short Burst Length),” for small access timing). Besides this dead cycle, in most cases, back-to-back accesses on the external bus do not cause any change in the timing from that shown in the previous diagrams, and the two transactions are independent of each other. The only exceptions to this are:

- Back-to-back accesses where the first access ends with an externally-driven \overline{TA} or \overline{TEA} . In these cases, an extra cycle is required between the end of the first access and the \overline{TS} assertion of the second access. See Section 29.4.2.9, “Termination Signals Protocol,” for more details.

NOTE

In some cases, \overline{CS} remains asserted during this dead cycle, such as the cases of back-to-back writes or read-after-write to the same chip-select. See Figure 29-20 and Figure 29-21. The following diagrams show a few examples of back-to-back accesses on the external bus.

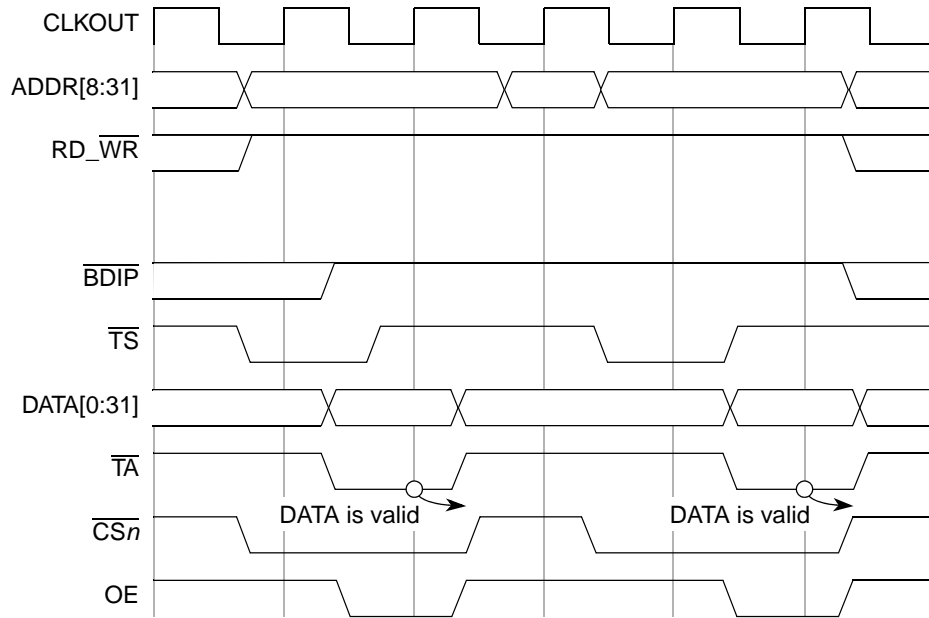


Figure 29-17. Back-to-Back 32-Bit Reads to the Same \overline{CS} Bank

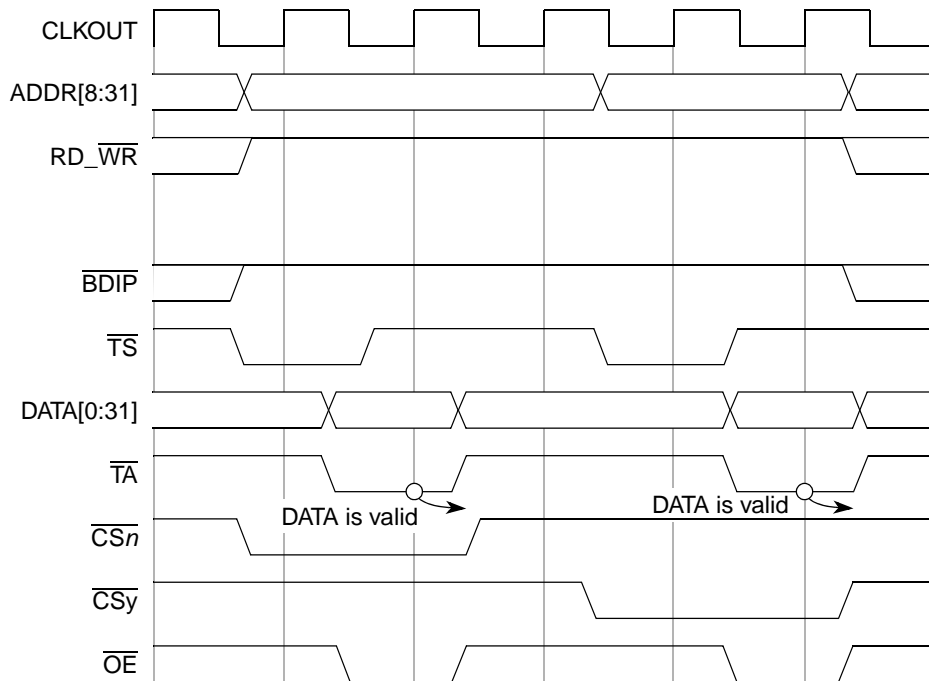


Figure 29-18. Back-to-Back 32-Bit Reads to Different \overline{CS} Banks

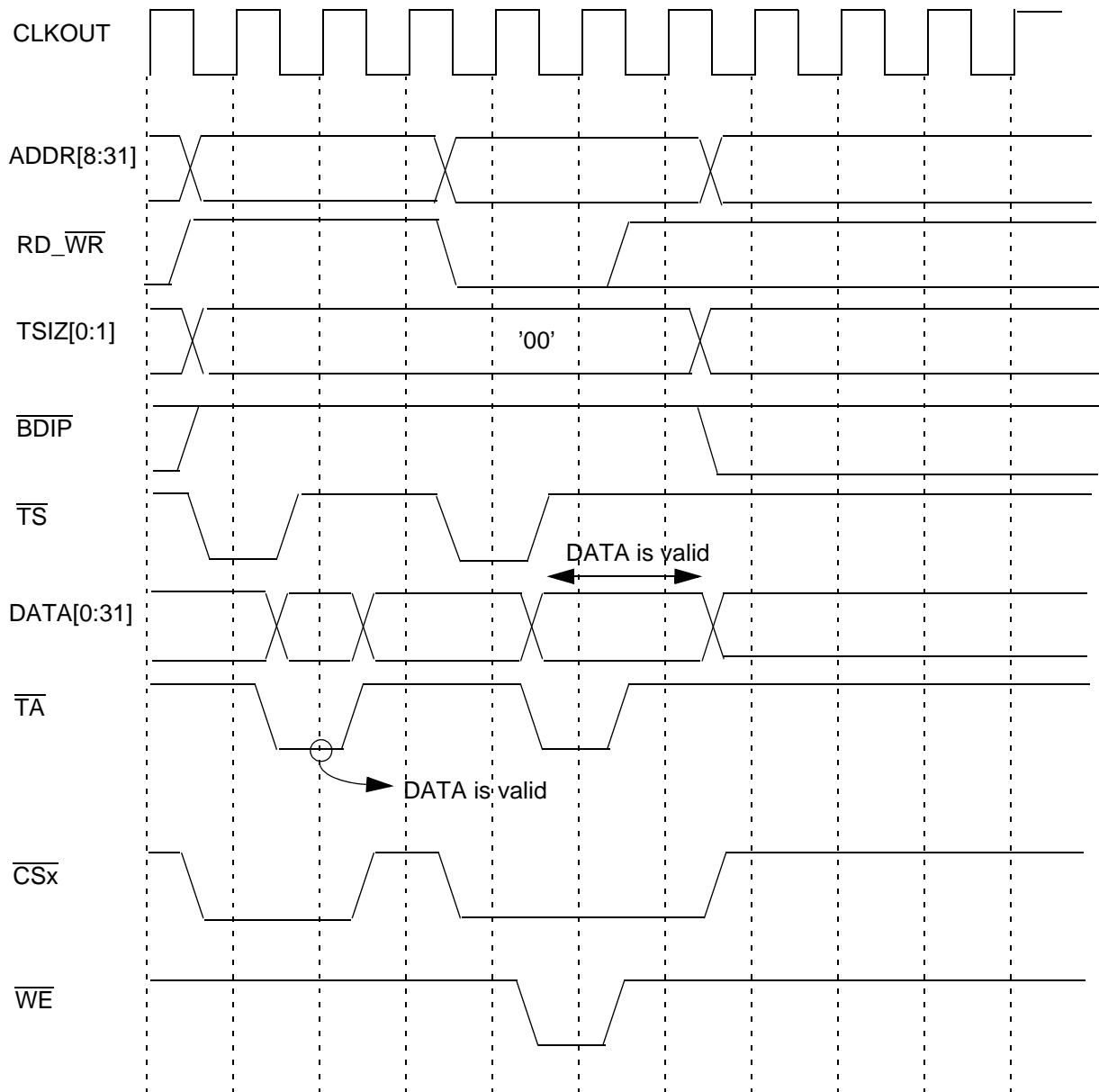


Figure 29-19. Write After Read to the Same CS Bank

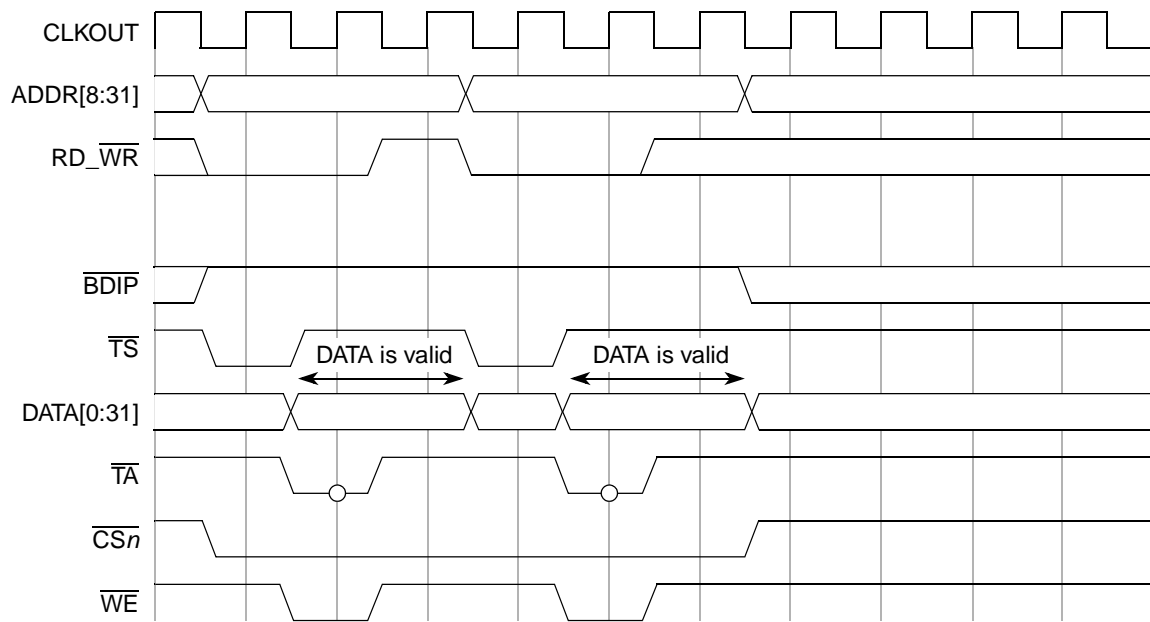


Figure 29-20. Back-to-Back 32-Bit Writes to the Same \overline{CSn} Bank

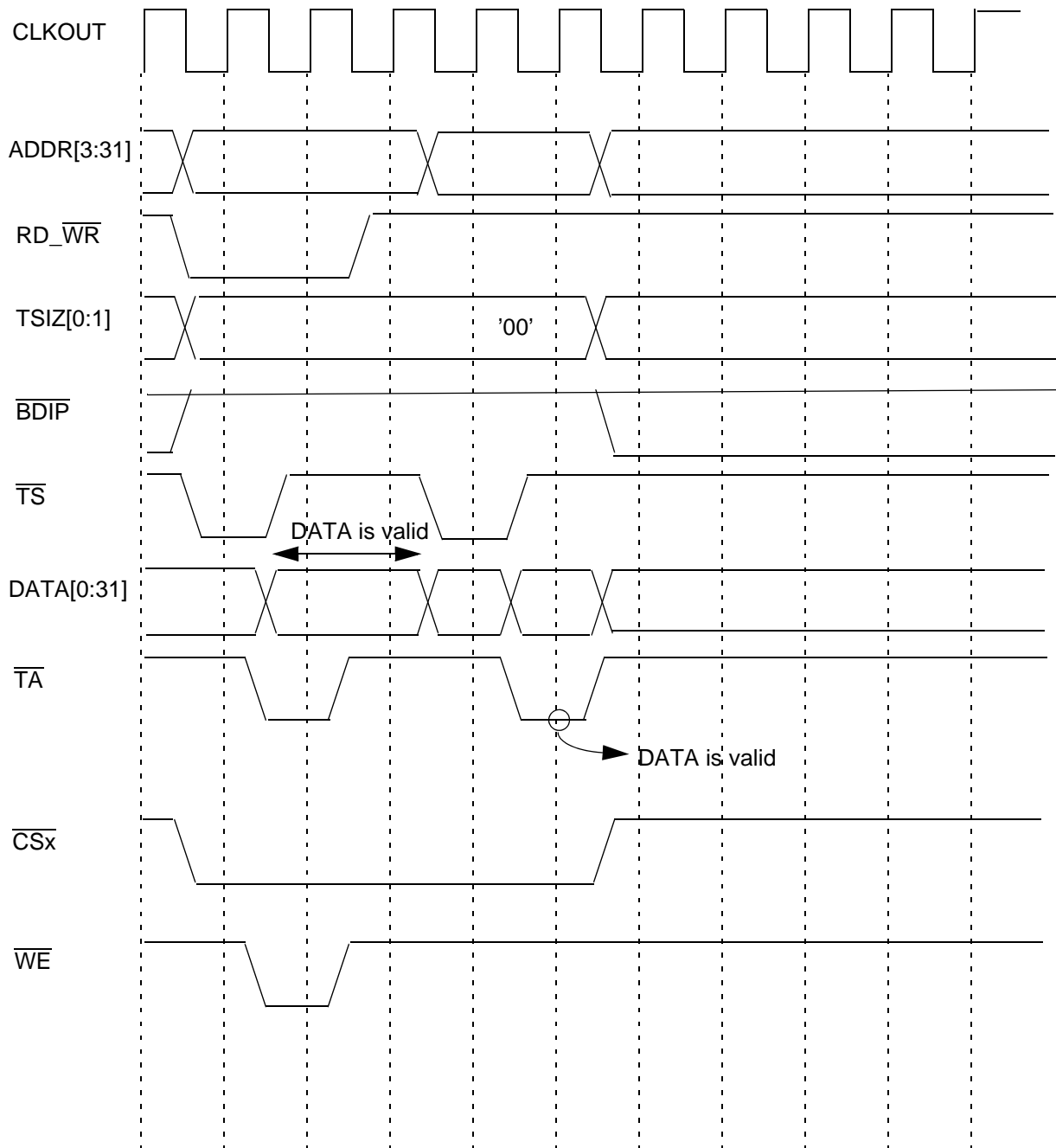


Figure 29-21. Read After Write to the Same \overline{CS} Bank

29.4.2.5 Burst Transfer

The EBI supports wrapping 16-byte critical-doubleword-first burst transfers. Bursting is supported only for internally-requested 16-byte read accesses to external devices that use the chip selects¹. Accesses from an external master or to devices operating without a chip select are always single beat. If an internal request

to the EBI indicates a size of less than 16 bytes, the request is fulfilled by running one or more single-beat external transfers, not by an external burst transfer.

A 4-word wrapping burst reads four 32-bit words by supplying a starting address that points to one of the words (word aligned) and requiring the memory device to sequentially drive each word on the data bus. The selected slave device must internally increment ADDR[28:29] (also ADDR30 in the case of a 16-bit port size device) of the supplied address for each transfer, until the address reaches a 4-word boundary, and then wrap the address to the beginning of the 4-word boundary. The address and transfer attributes supplied by the EBI remain stable during the transfers, and the EBI terminates each beat transfer by asserting $\overline{\text{TA}}$ (if SETA=0, by the EBI, or externally generated when SETA=0). The EBI requires that addresses be aligned to a doubleword boundary on all burst cycles.

Table 29-13 shows the burst order of beats returned for a 4-word burst to a 32-bit port.

Table 29-13. Wrap Bursts Order

Burst Starting Address ADDR[28:29]	Burst Order (Assuming 32-Bit Port Size)
00	word0 → word1 → word2 → word3
01	word1 → word2 → word3 → word0
10	word2 → word3 → word0 → word1
11	word3 → word0 → word1 → word2

The general case of burst transfers assumes that the external memory has a 32-bit port size. The EBI can also burst from 16-bit port size memories, taking twice as many external beats to fetch the data as compared to a 32-bit port with the same burst length.

During burst cycles, the $\overline{\text{BDIP}}$ (burst data in progress) signal is used to indicate the duration of the burst data. During the data phase of a burst read cycle, the EBI receives data from the addressed slave. If the EBI needs more than one data, it asserts the $\overline{\text{BDIP}}$ signal. Upon receiving the data prior to the last data, the EBI negates $\overline{\text{BDIP}}$. Thus, the slave stops driving new data after it receives the negation of $\overline{\text{BDIP}}$ on the rising edge of the clock. Some slave devices have their burst length and timing configurable internally and thus may not support connecting to a $\overline{\text{BDIP}}$ pin. In this case, $\overline{\text{BDIP}}$ is driven by the EBI normally, but the output is ignored by the memory and the burst data behavior is determined by the internal configuration of the EBI and slave device. When the TBDIP bit is set in the appropriate base register, the timing for $\overline{\text{BDIP}}$ is altered. See Section 29.4.2.5.1, “TBDIP Effect on Burst Transfer,” for this timing.

Because burst writes are not supported by the EBI¹, the EBI negates $\overline{\text{BDIP}}$ during write cycles.

1. Except for the special case of a 32-bit non-chip select access in 16-bit data bus mode. See Section 29.4.2.10, “Non-Chip-Select Burst in 16-bit Data Bus Mode.”

1. Except for the special case of a 32-bit non-chip select access in 16-bit data bus mode. See Section 29.4.2.10, “Non-Chip-Select Burst in 16-bit Data Bus Mode.”

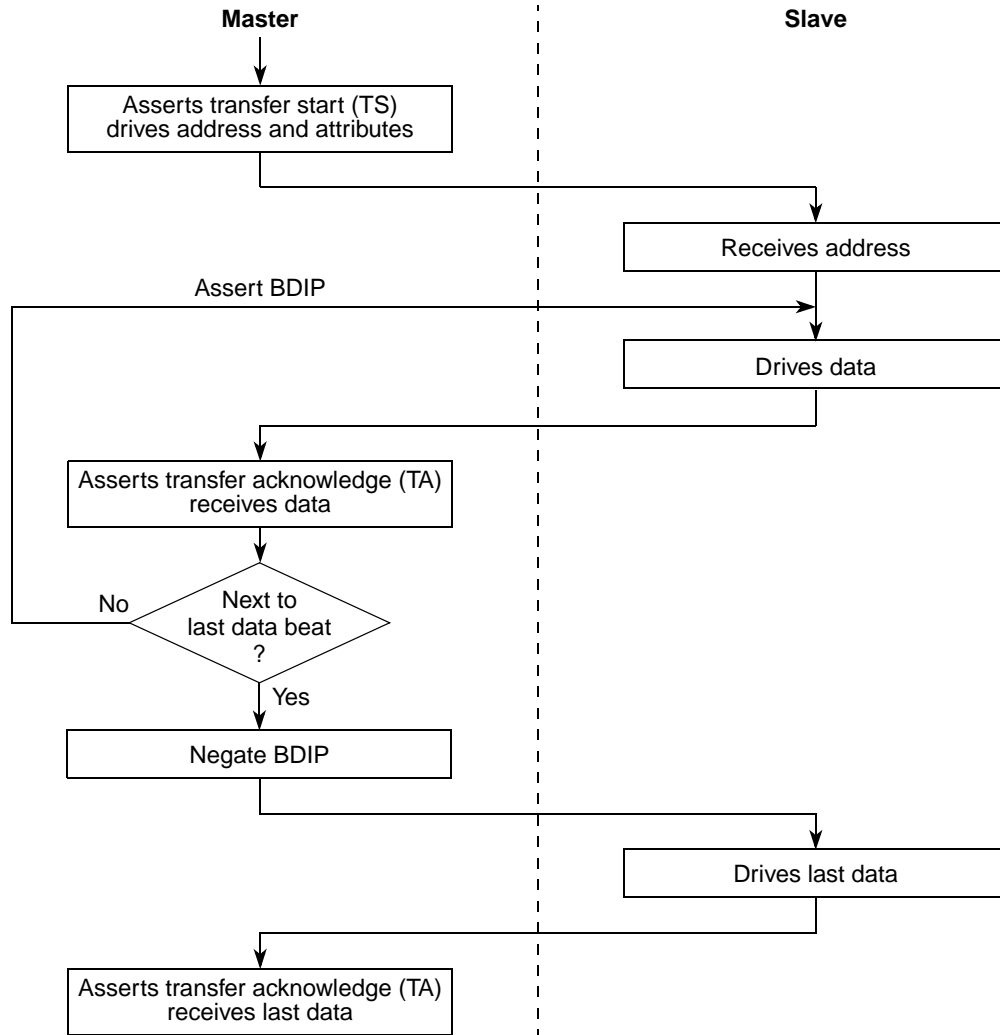


Figure 29-22. Basic Flow Diagram of a Burst Read Cycle

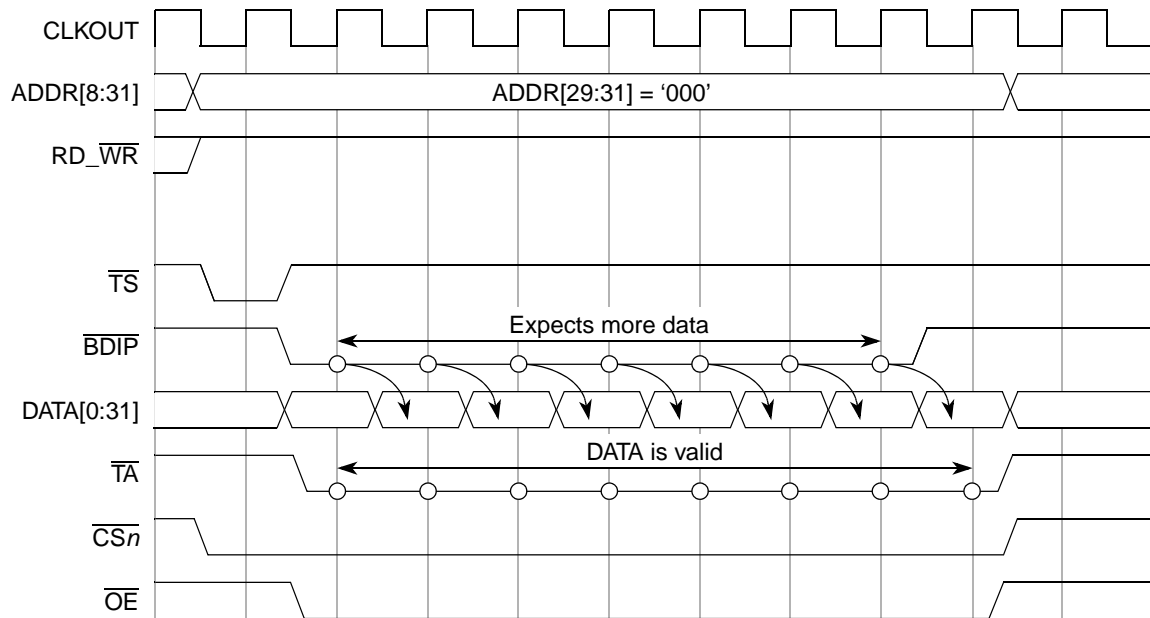


Figure 29-23. Burst 32-Bit Read Cycle, Zero Wait States

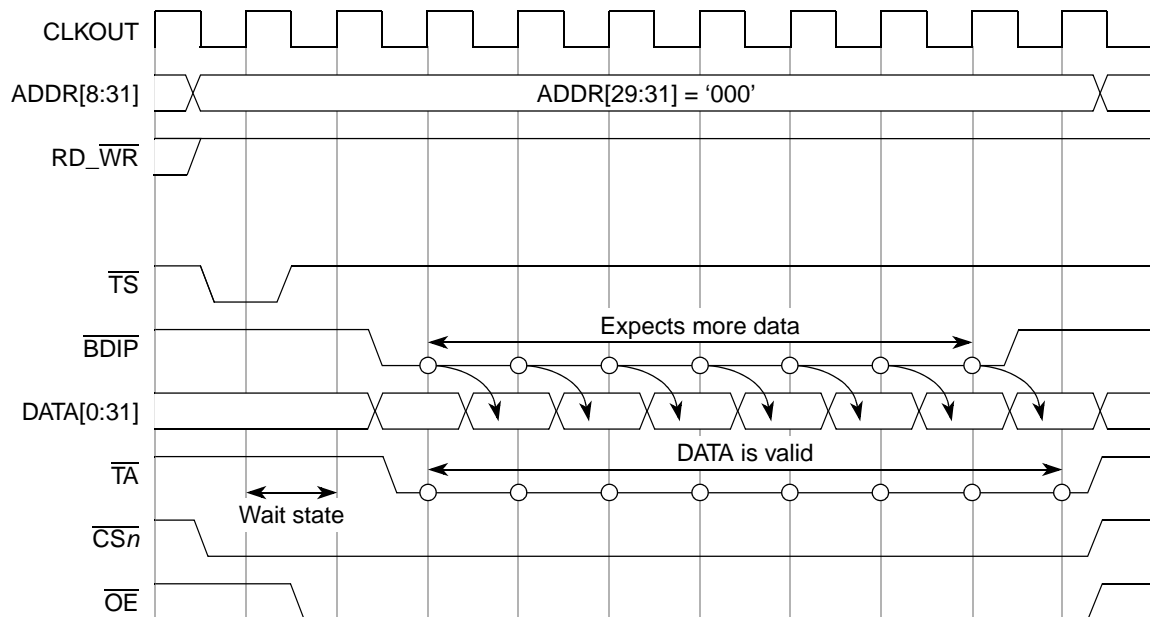


Figure 29-24. Burst 32-Bit Read Cycle, One Initial Wait State

29.4.2.5.1 TBDIP Effect on Burst Transfer

Some memories require different timing on the $\overline{\text{BDIP}}$ signal than the default to run burst cycles. Using the default value of $\text{TBDIP} = 0$ in the appropriate EBI base register results in $\overline{\text{BDIP}}$ being asserted ($\text{SCY}+1$) cycles after the address transfer phase, and being held asserted throughout the cycle regardless of the wait states between beats (BSCY). Figure 29-25 shows an example of the $\text{TBDIP} = 0$ timing for a 4-beat burst with $\text{BSCY} = 1$.

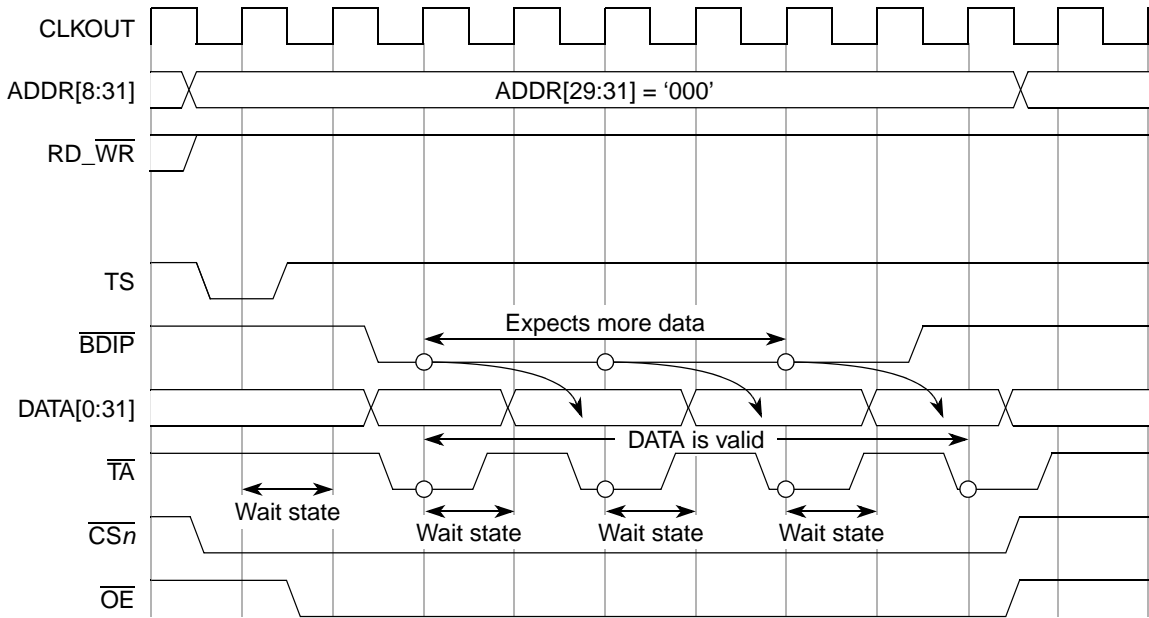


Figure 29-25. Burst 32-bit Read Cycle, One Wait State Between Beats, TBDIP = 0

When using TBDIP = 1, the $\bar{B}DIP$ behavior changes to toggle between every beat when BSCY is a non-zero value. Figure 29-26 shows an example of the TBDIP = 1 timing for the same four-beat burst shown in Figure 29-25.

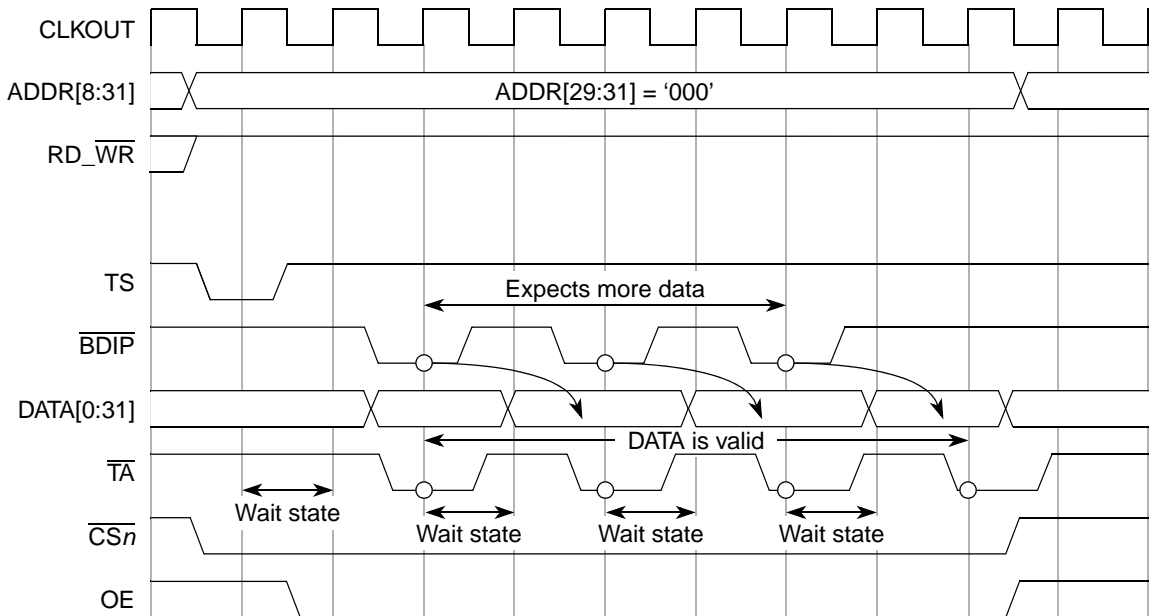


Figure 29-26. Burst 32-bit Read Cycle, One Wait State Between Beats, TBDIP = 1

29.4.2.6 Small Accesses (Small Port Size and Short Burst Length)

In this context, a small access refers to an access whose burst length and port size are such that the number of bytes requested by the internal master cannot all be fetched (or written) in one external transaction. This

is the case when the base register's burst length bit (EBI_BRn[BL]) and port size bit (EBI_BRn[PS]) are set such that one of two situations occur:

- Burst accesses are inhibited and the number of bytes requested by the master is greater than the port size (16 or 32 bit) can accommodate in a single access.
- Burst accesses are enabled and the number of bytes requested by the master is greater than the selected burst length (4 words or 8 words).

If this is the case, the EBI initiates multiple transactions until all the requested data is transferred. It should be noted that all the transactions initiated to complete the data transfer are considered as an atomic transaction, so the EBI does not allow other unrelated master accesses to intervene between the transfers.

Table 29-14 shows all the combinations of burst length, port size, and requested byte count that cause the EBI to run multiple external transactions to fulfill the request.

Table 29-14. Small Access Cases

Byte Count Requested by Internal Master ¹	Burst Length	Port Size	# External Accesses to Fulfill Request
Non-Burstable Chip-Select Banks (BI=1) or Non-Chip-Select Access			
4	1 beat	16-bit	2/1 ²
8	1 beat	32-bit	2
8	1 beat	16-bit	4
16	1 beat	32-bit	4
16	1 beat	16-bit	8
32	1 beat	32-bit	N/A
32	1 beat	16-bit	N/A
Burstable Chip-Select Banks (BI=0)			
32	4 words	16-bit (8 beats), 32-bit (4 beats)	N/A

¹ The MPC5510 bus masters do not generate any 32-byte requests so these cases cannot occur.

² In 32-bit data bus mode (DBM=0 in EBI_MCR), two accesses are performed. In 16-bit data bus mode (DBM=1), one 2-beat burst access is performed and this is not considered a small access case. See [Section 29.4.2.10, "Non-Chip-Select Burst in 16-bit Data Bus Mode"](#) for this special DBM=1 case.

In most cases, the timing for small accesses is the same as for normal single-beat and burst accesses, except that multiple back-to-back external transfers are executed for each internal request. These transfers have no additional dead cycles in-between that are not present for back-to-back stand-alone transfers except for the case of writes with an internal request size greater than 64 bits, discussed in [Section 29.4.2.6.2, "Small Access Example #2: 32-byte Write with External TA."](#)

The following sections show a few examples of small accesses. The timing for the remaining cases in [Table 29-14](#) can be extrapolated from these and the other timing diagrams in this document.

29.4.2.6.1 Small Access Example #1: 32-Bit Write to 16-Bit Port

Figure 29-27 shows an example of a 32-bit write to a 16-bit port, requiring two 16-bit external transactions.

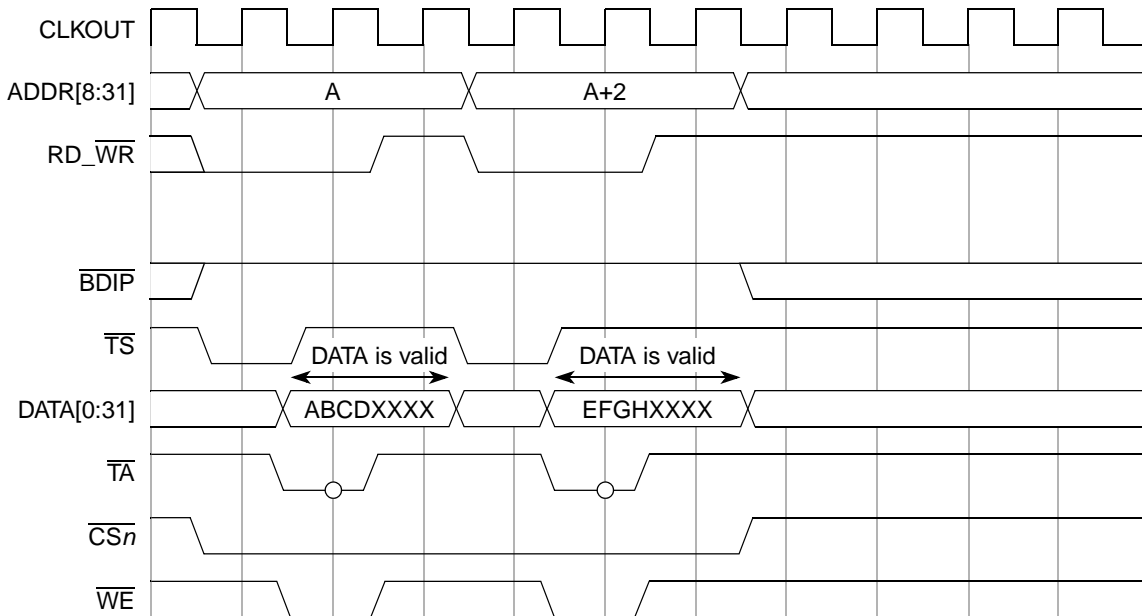
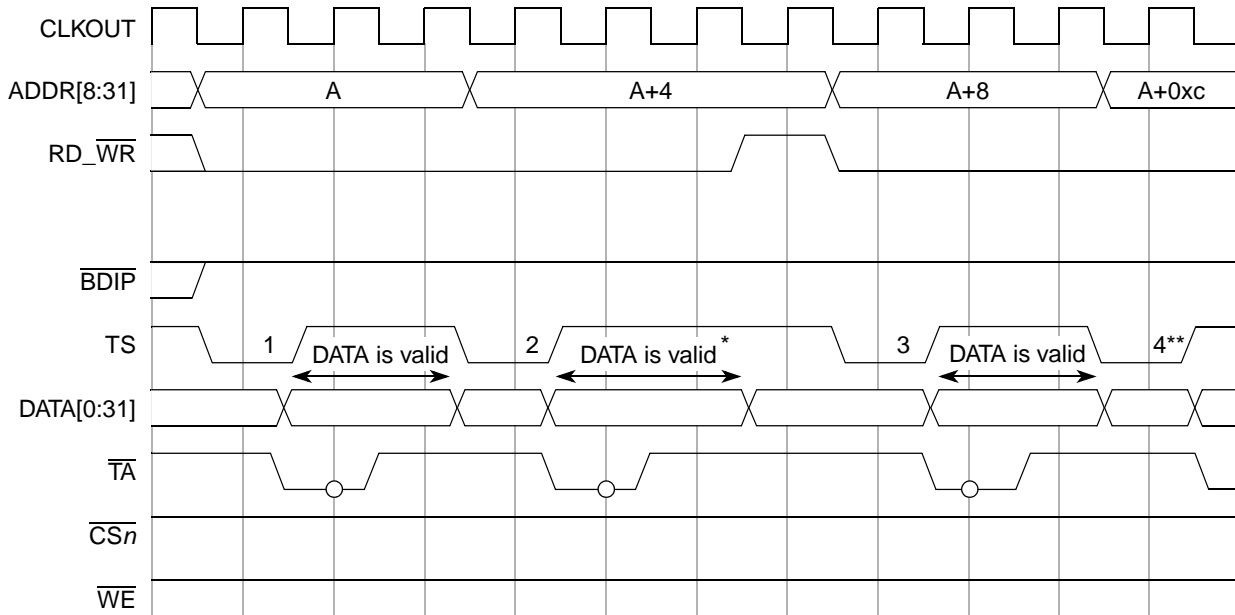


Figure 29-27. Single Beat 32-Bit Write Cycle, 16-bit Port Size, Basic Timing

29.4.2.6.2 Small Access Example #2: 32-byte Write with External TA

Figure 29-28 shows an example of a 32-byte write to a non-chip select device, such as an external master, using external TA, requiring eight 32-bit external transactions. Due to the use of external TA, RD_WB does not toggle between the accesses unless that access is the end of a 64-bit boundary. In this case, an extra cycle is required between TA and the next TS in order to get the next 64-bits of write data internally and RD_WB negates during this extra cycle.



* This extra cycle is required after accesses 2, 4, and 6 in order to get the next 64-bits of internal write data.

** Four more external accesses (not shown) are required to complete the internal 32-byte request. The timing of these is the same as accesses 1-4 shown in this diagram.

Figure 29-28. 32-Byte Write Cycle with External \overline{TA} , Basic Timing

29.4.2.7 Size, Alignment, and Packaging on Transfers

Table 29-15 shows the allowed sizes that an internal or external master can request from the EBI. The behavior of the EBI for request sizes not shown below is undefined. No error signal is asserted for these erroneous cases.

Table 29-15. Transaction Sizes Supported by EBI

No. Bytes (Internal Master)	No. Bytes (External Master)
1	1
2	2
4	4
3 ¹	
8	
16	

¹ Some misaligned access cases may result in 3-byte writes. These cases are treated as power-of-2 sized requests by the EBI, using $\overline{WE}[0:3]$ to make sure only the appropriate 3 bytes get written.

Even though misaligned non-burst transfers from internal masters are supported, the EBI naturally aligns the accesses when it sends them out to the external bus, splitting them into multiple aligned accesses if necessary. See Section 29.4.1.13, “Misaligned Access Support,” for these cases.

Natural alignment for the EBI means:

- Byte access can have any address.
- 16-bit access, address bit 31 must be 0.
- 32-bit access, address bits 30–31 must be 0.
- For burst accesses of any size, address bits 29–31 must be 0.

The EBI requires that the portion of the data bus used for a transfer to/from a particular port size be fixed. A 32-bit port must reside on data bus bits 0–31, and a 16-bit port must reside on bits 0–15.

In the following figures and tables the following convention is adopted:

- The most significant byte of a 32-bit operand is OP0, the least significant byte is OP3.
- The two bytes of a 16-bit operand are OP0 (most significant) and OP1, or OP2 (most significant) and OP3, depending on the address of the access.
- The single byte of a byte-length operand is OP0, OP1, OP2, or OP3, depending on the address of the access.

The convention can be seen in [Figure 29-29](#).

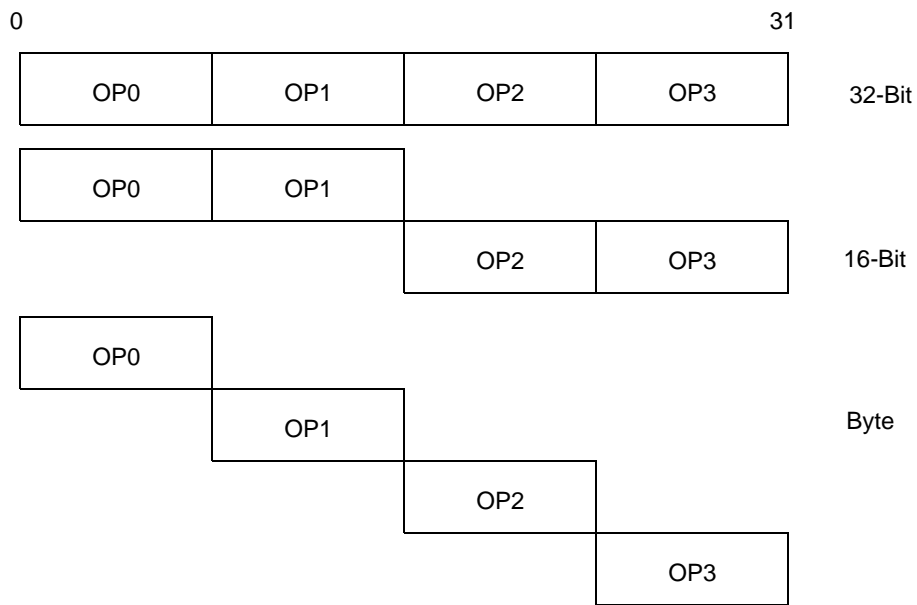


Figure 29-29. Internal Operand Representation

[Figure 29-30](#) shows the device connections on the AD[0:31] bus.

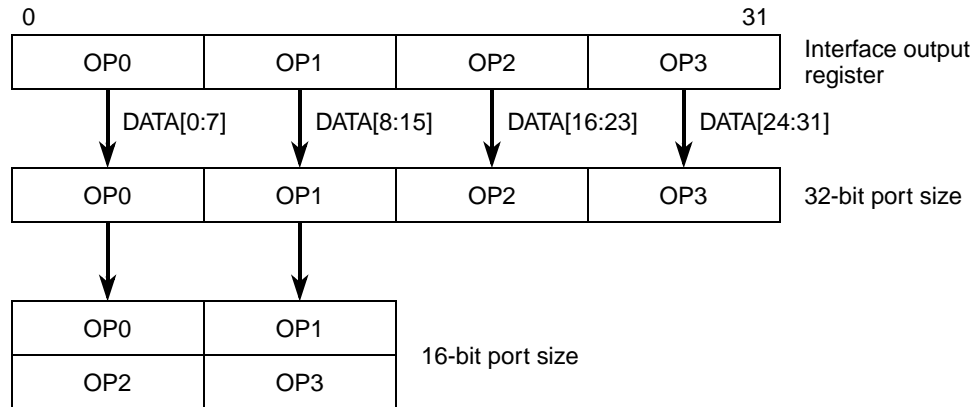


Figure 29-30. Interface to Different Port Size Devices

Table 29-16 lists the bytes required on the data bus for read cycles. The bytes indicated as ‘—’ are not required during that read cycle.

Table 29-16. Data Bus Requirements for Read Cycles

Transfer Size	TSIZ[0:1] ¹	Address		32-Bit Port Size				16-Bit Port Size ²	
		A30	A31	D0:D7	D8:D15	D16:D23	D24:D31	D0:D7	D8:D15
Byte	01	0	0	OP0	—	—	—	OP0	—
	01	0	1	—	OP1	—	—	—	OP1
	01	1	0	—	—	OP2	—	OP2	—
	01	1	1	—	—	—	OP3	—	OP3
16-bit	10	0	0	OP0	OP1	—	—	OP0	OP1
	10	1	0	—	—	OP2	OP3	OP2	OP3
32-bit	00	0	0	OP0	OP1	OP2	OP3	OP0/OP2 ³	OP1/OP3

¹ TSIZ is not enabled on the MPC5510.

² Also applies when DBM=1 for 16-bit data bus mode.

³ This case consists of two 16-bit external transactions, the first fetching OP0 and OP1, the second fetching OP2 and OP3.

Table 29-17 lists the patterns of the data transfer for write cycles when accesses are initiated by the MCU. A dash indicates bytes that are not driven during that write cycle.

Table 29-17. Data Bus Contents for Write Cycles

Transfer Size	TSIZ[0:1] ¹	Address		32-Bit Port Size				16-Bit Port Size ²	
		A30	A31	D0:D7	D8:D15	D16:D23	D24:D31	D0:D7	D8:D15
Byte	01	0	0	OP0	—	—	—	OP0	—
	01	0	1	OP1	OP1	—	—	—	OP1
	01	1	0	OP2	—	OP2	—	OP2	—
	01	1	1	OP3	OP3	—	OP3	—	OP3
16-bit	10	0	0	OP0	OP1	—	—	OP0	OP1
	10	1	0	OP2	OP3	OP2	OP3	OP2	OP3
32-bit	00	0	0	OP0	OP1	OP2	OP3	OP0/OP2 ³	OP1/OP3

¹ TSIZ is not enabled on the MPC5510.

² Also applies when DBM=1 for 16-bit data bus mode.

³ This case consists of two 16-bit external transactions, the first writing OP0 and OP1, the second writing OP2 and OP3.

29.4.2.8 Arbitration

The MPC5510 does not support arbitration.

29.4.2.9 Termination Signals Protocol

The termination signals protocol was defined to avoid electrical contention on lines that can be driven by various sources. To do that, a slave must not drive signals associated with the data transfer until the address phase is completed and it recognizes the address as its own. The slave must disconnect from signals immediately after it acknowledges the cycle and not later than the termination of the next address phase cycle.

For EBI-mastered non-chip select accesses, the EBI requires assertion of \overline{TA} from an external device to signal that the bus cycle is complete. The EBI uses a latched version of \overline{TA} (1 cycle delayed) for these accesses to help make timing at high frequencies. This results in the EBI driving the address and control signals 1 cycle longer than required, as seen in Figure 29-31. However, the data (AD) does not need to be held 1 cycle longer by the slave, because the EBI latches AD every cycle during non-chip select accesses. During these accesses, the EBI does not drive the \overline{TA} signal, leaving it up to an external device (or weak internal pullup) to drive \overline{TA} .

For EBI-mastered chip-select accesses, when the SETA bit is 0, the EBI drives \overline{TA} the entire cycle, asserting according to internal wait state counters to terminate the cycle. When the SETA bit is 1, the EBI samples the \overline{TA} for the entire cycle. During idle periods on the external bus, the EBI drives \overline{TA} negated as long as it is granted the bus; when it no longer owns the bus, it lets go of \overline{TA} . When an external master does a transaction to internal address space, the EBI only drives \overline{TA} for the cycle it asserts \overline{TA} to return data and for 1 cycle afterwards to ensure fast negation.

If no device responds by asserting \overline{TA} within the programmed timeout period (BMT in EBI_BMCR) after the EBI initiates the bus cycle, the internal bus monitor (if enabled) asserts \overline{TEA} to terminate the cycle. An external device may also drive \overline{TEA} when it detects an error on an external transaction. \overline{TEA} assertion

causes the cycle to terminate and the processor to enter exception processing for the error condition. To properly control termination of a bus cycle for a bus error with external circuitry, \overline{TEA} must be asserted at the same time or before (external) \overline{TA} is asserted. \overline{TEA} must be negated before the second rising edge after it was sampled asserted to avoid the detection of an error for the following bus cycle initiated. \overline{TEA} is driven only by the EBI during the cycle where the EBI is asserting \overline{TEA} and the cycle immediately following this assertion (for fast negation). During all other cycles, the EBI relies on a weak internal pull-up to hold \overline{TEA} negated. This allows an external device to assert \overline{TEA} when it needs to indicate an error. External devices must follow the same protocol as the EBI, only driving \overline{TEA} during the assertion cycle and 1 cycle afterwards for negation.

NOTE

In the case where an external master asserts \overline{TEA} to timeout a transaction to an internal address on this MCU, the EBI has no way to terminate the transfer internally. Therefore, any subsequent \overline{TS} assertions by the external master are ignored by the EBI until the original transfer has completed internally and the EBI has returned to an idle state. The expectation is that the internal slaves will always respond with valid data or an error indication within a reasonable period of time to avoid hanging the system.

When \overline{TEA} is asserted from an external source, the EBI uses a latched version of \overline{TEA} (1 cycle delayed) to help make timing at high frequencies. This means that for any accesses where the EBI drives \overline{TA} (chip select accesses with $SETA=0$ and external master accesses to EBI), a \overline{TEA} assertion that occurs 1 cycle before or during the last \overline{TA} of the access could be ignored by the EBI, since it will have completed the access internally before it detects the latched \overline{TEA} assertion. This means that non-burst chip select accesses with no wait states ($SCY = 0$) cannot be reliably terminated by external \overline{TEA} . If external error termination is required for such a device, the EBI must be configured for $SCY \geq 1$.

NOTE

For the cases discussed above where \overline{TEA} could be ignored, this is not guaranteed. For some small access cases, which always use chip select and internally-driven \overline{TA} , a \overline{TEA} that occurs 1 cycle before or during the \overline{TA} cycle or for $SCY = 0$ may lead to terminating the cycle with error. However, proper error termination is not guaranteed for these cases, so \overline{TEA} must always be asserted at least 2 cycles before an internally-driven \overline{TA} cycle for proper error termination.

External \overline{TEA} assertion that occurs during the same cycle that \overline{TS} is asserted by the EBI is always treated as an error (terminating the access) regardless of SCY .

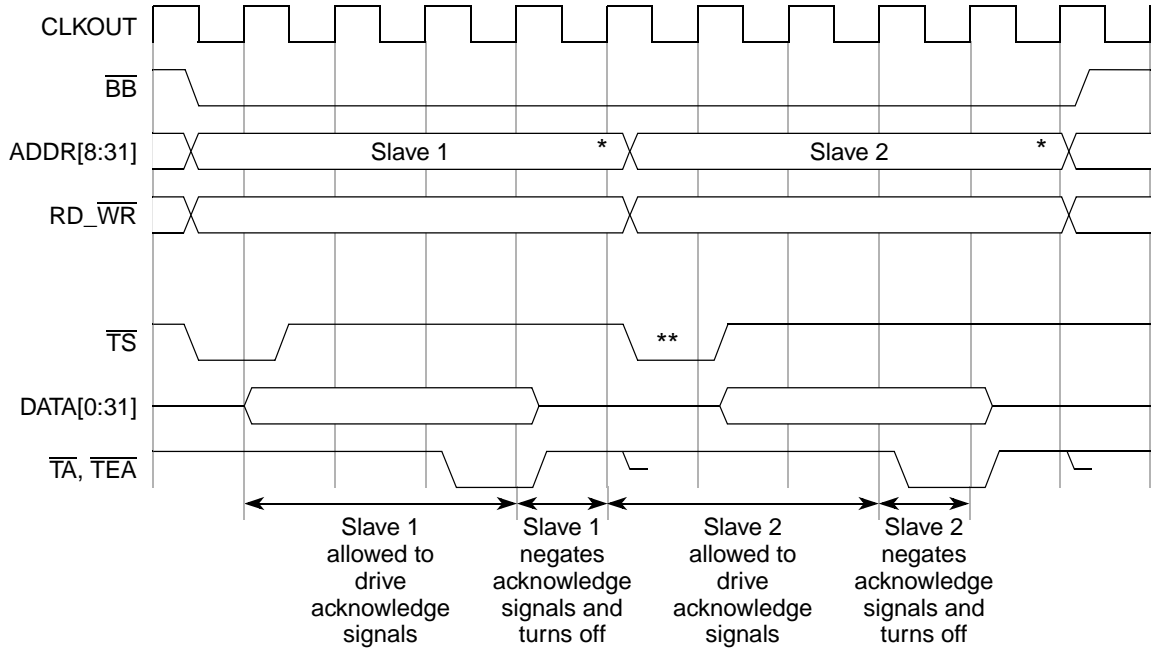
Table 29-18 summarizes how the EBI recognizes the termination signals provided from an external device.

Table 29-18. Termination Signals Protocol

TEA^1	TA^1	Action
Negated	Negated	No termination
Asserted	X	Transfer error termination
Negated	Asserted	Normal transfer termination

¹ Latched version (1 cycle delayed) used for externally driven \overline{TEA} and \overline{TA} .

Figure 29-31 shows an example of the termination signals protocol for back-to-back reads to two different slave devices that properly take turns driving the termination signals. This assumes a system using slave devices that drive termination signals.



* The EBI drives address and control signals an extra cycle because it uses a latched version of \overline{TA} (1 cycle delayed) to terminate the cycle. An external master is not required to do this.
 ** This is the earliest that the EBI can start another transfer when continuing a set of small accesses. For all other cases, an extra cycle is needed before the EBI can start another \overline{TS} .

Figure 29-31. Termination Signals Protocol Timing Diagram

29.4.2.10 Non-Chip-Select Burst in 16-bit Data Bus Mode

The timing diagrams in this section apply to the special case of a non-chip select 32-bit access in 16-bit data bus mode. They specify the behavior for the EBI-master and EBI-slave, as the external master is expected to be another MCU with this EBI.

For this case, a special 2-beat burst protocol is used for reads and writes, so the EBI-slave can internally generate one 32-bit read or write access (thus 32-bit coherent), as opposed to two separate 16-bit accesses.

Figure 29-32 shows a 32-bit non-chip-select read in 16-bit data bus mode.

Figure 29-33 shows a 32-bit non-chip-select write in 16-bit data bus mode.

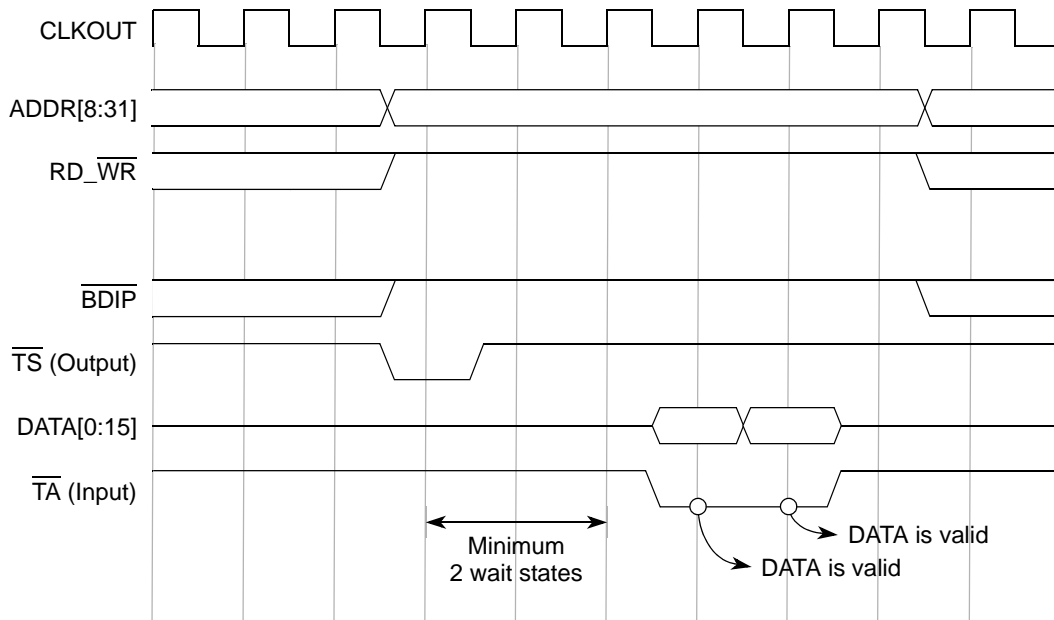


Figure 29-32. 32-bit Non-Chip-Select Read with DBM=1

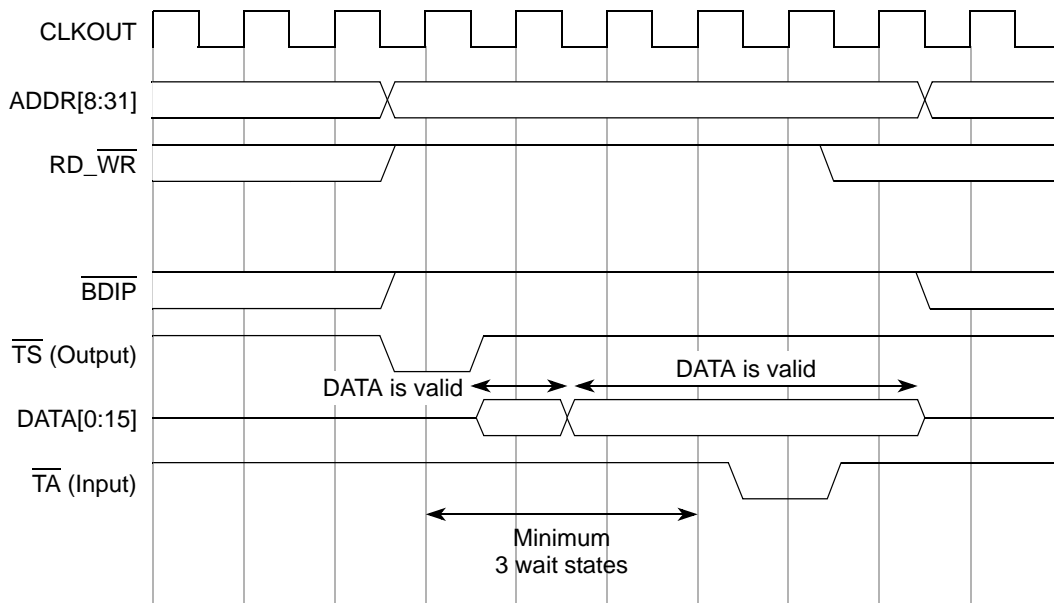


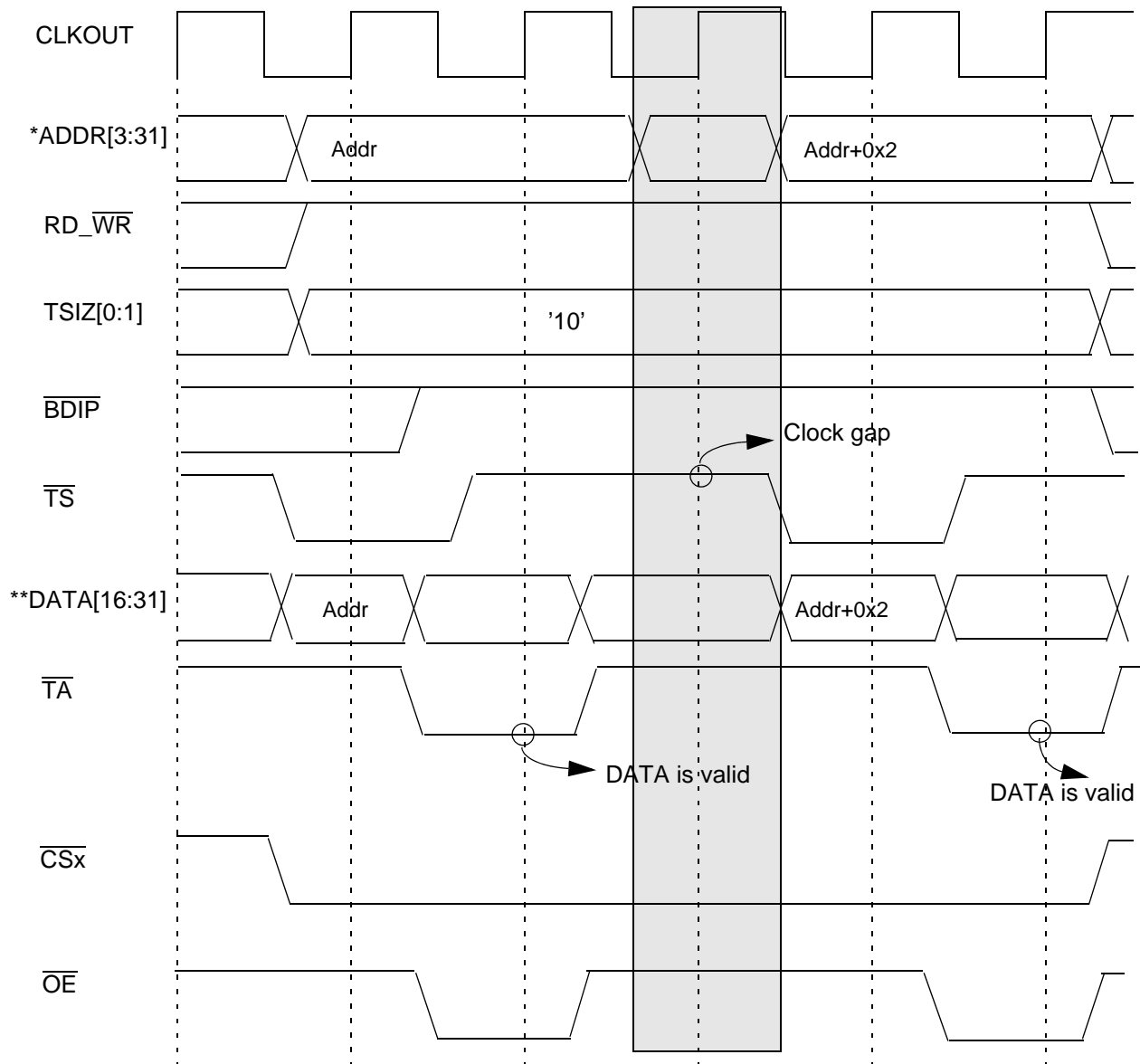
Figure 29-33. 32-bit Non-Chip-Select Write with DBM=1

29.4.2.11 Address Data Multiplexing

Address/data multiplexing enables the design of a system with reduced pin count. In such system, multiplexed address/data functions (on AD pins) are used, instead of having separate address and data pins. Compared to the normal EBI specification (e.g. 24 address pins+32 data pins), only 32 data pins are required. Compared to a 16-bit bus implementation, only 24 pins are required (e.g. ADDR[8:15] + ADDR[16:31]/DATA[16:31]).

When performing a small access read, as described in [Section 29.4.2.6, “Small Accesses \(Small Port Size and Short Burst Length\)”](#), with A/D multiplexing enabled for this access, the EBI inserts an idle clock cycle with \overline{OE} negated and \overline{CS} asserted, to allow for the memory to three-state the bus prior to the EBI driving the address on the next clock. This clock gap already exists (for other reasons) for non-small-access transfers, so no additional clock gap is inserted for those cases. See [Figure 29-34](#) for an example of a small access read with A/D multiplexing enabled.

In general, timing diagrams in A/D multiplexing mode are similar to other diagrams in this document, excepting behavior of the ADDR and DATA busses, which can be seen in [Figure 29-34](#).



* While the EBI drives all of ADDR[3:31] to valid address, typically ADDR[3:15] (or less) only are used in the system, as DATA[16:31] (or DATA[0:15]) are used for address and data on an external muxed device.

** Or DATA[0:15], based on D16_31 bit in EBI_MCR.

Figure 29-34. Small Access (32-Bit Read to 16-Bit Port) on Address/Data Multiplexed Bus

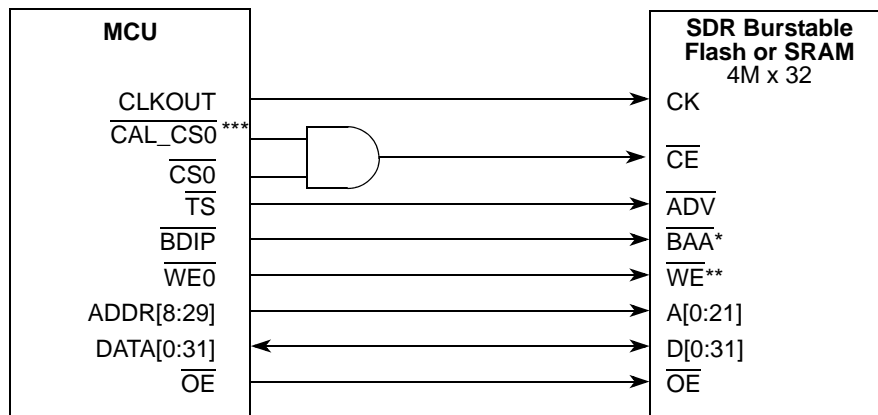
29.5 Initialization/Application Information

29.5.1 Booting from External Memory (for Factory Test only)

The EBI block does not support booting directly to external memory (i.e. fetching the first instruction after reset externally). The MCU uses an internal boot assist module, which executes after each reset. The BAM code performs basic configuration of the EBI block, allowing for external boot if desired. Refer to [Chapter 32, “Boot Assist Module \(BAM\)”](#) for information about the boot modes supported by the MCU.

29.5.2 Running with Single Data Rate (SDR) Burst Memories

This includes flash and external SRAM memories with a compatible burst interface. $\overline{\text{BDIP}}$ is required for some SDR memories only. [Figure 29-32](#) shows a block diagram of an MCU connected to a 32-bit SDR burst memory.



* May or may not be connected, depending on the memory used.

** Flash memories typically use one WE signal as shown, RAMs use 2 or 4 (16-bit or 32-bit).

*** Not available on all devices, refer to the Signals chapter

Figure 29-35. MCU Connected to SDR Burst Memory

Refer to [Figure 29-23](#) for an example of the timing of a typical burst read operation to an SDR burst memory. Refer to [Figure 29-14](#) for an example of the timing of a typical single write operation to SDR memory.

29.5.3 Running with Asynchronous Memories

The EBI also supports asynchronous memories. In this case, the CLKOUT, $\overline{\text{TS}}$, and $\overline{\text{BDIP}}$ pins are not used by the memory and bursting is not supported. However, the EBI still drives these outputs, and always drives and latches all signals at positive edge CLKOUT (i.e., there is no asynchronous mode for the EBI). The data timing is controlled by setting the SCY bits in the appropriate option register to the proper number of wait states to work with the access time of the asynchronous memory, exactly as done for a synchronous memory.

29.5.3.1 Example Wait State Calculation

This example applies to any chip-select memory, synchronous or asynchronous.

For example, if there is a memory with 50 ns access time, and the external bus is running at 66 MHz (CLKOUT period: 15.2 ns). Assume the input data spec for the MCU is 4 ns.

number of wait states = (access time) / (CLKOUT period) + (0 or 1) (depending on set-up time)

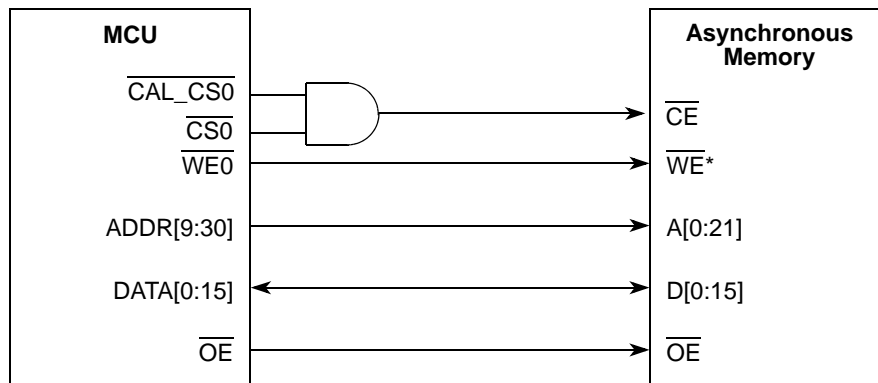
$50/15.2 = 3$ with 4.4 ns remaining (at least three wait states are needed, now check set-up time)

$15.2 - 4.4 = 10.8$ ns (this is the achieved input data set-up time)

Because actual input setup (10.8 ns) is greater than the input setup spec (4.0 ns), three wait states is sufficient. If the actual input setup was less than 4.0 ns, four wait states would be used instead.

29.5.3.2 Timing and Connections for Asynchronous Memories

The connections to an asynchronous memory are the same as for a synchronous memory, except that the CLKOUT, TS, and BDIP signals are not used. Figure 29-36 shows a block diagram of an MCU connected to an asynchronous memory.



* Flash memories typically use one \overline{WE} signal as shown, RAMs use two or four (16-bit or 32-bit).

Figure 29-36. MCU Connected to Asynchronous Memory

Figure 29-37 shows a timing diagram of a read operation to a 16-bit asynchronous memory using three wait states. Figure 29-38 shows a timing diagram of a write operation to a 16-bit asynchronous memory using three wait states.

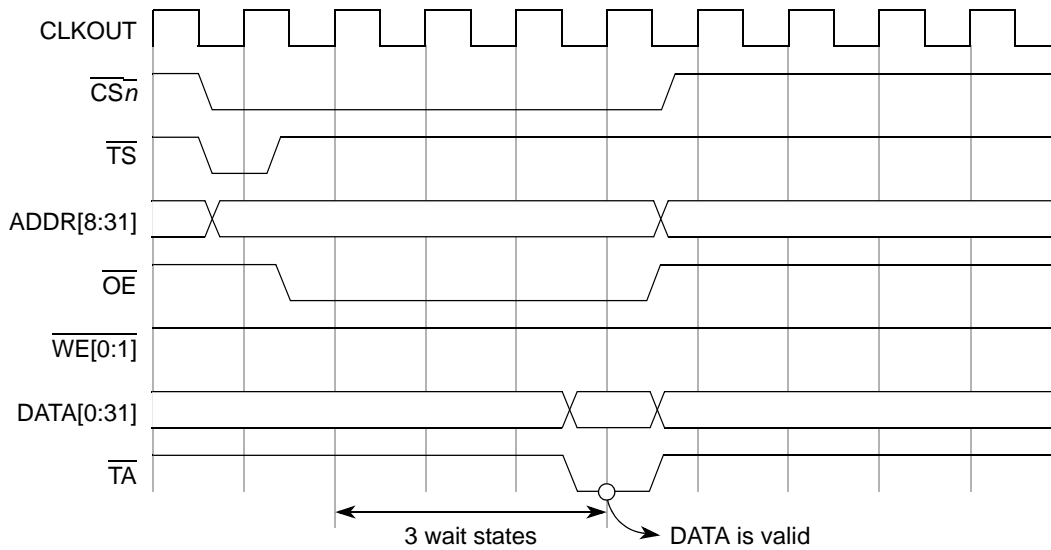


Figure 29-37. Read Operation to Asynchronous Memory, Three Initial Wait States

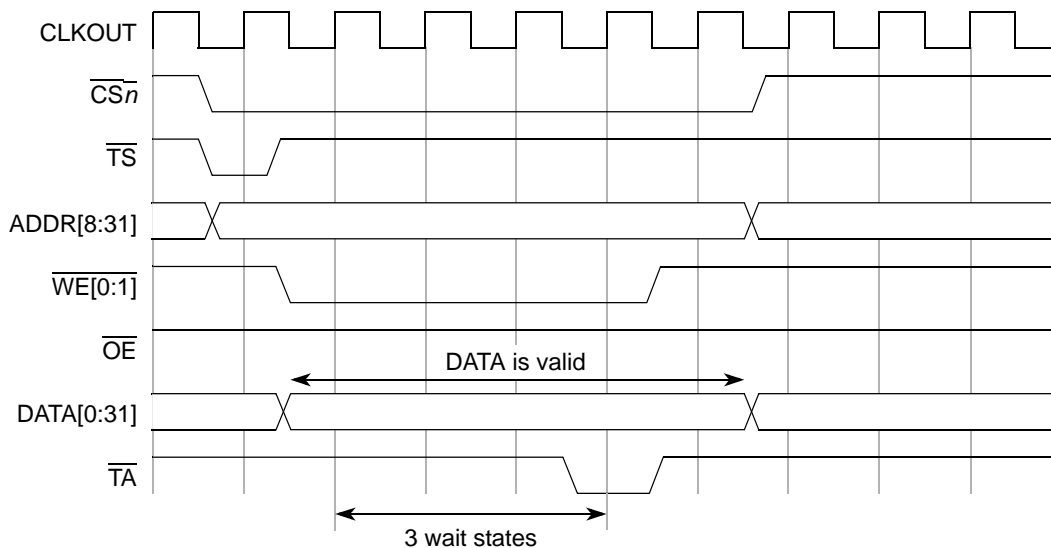
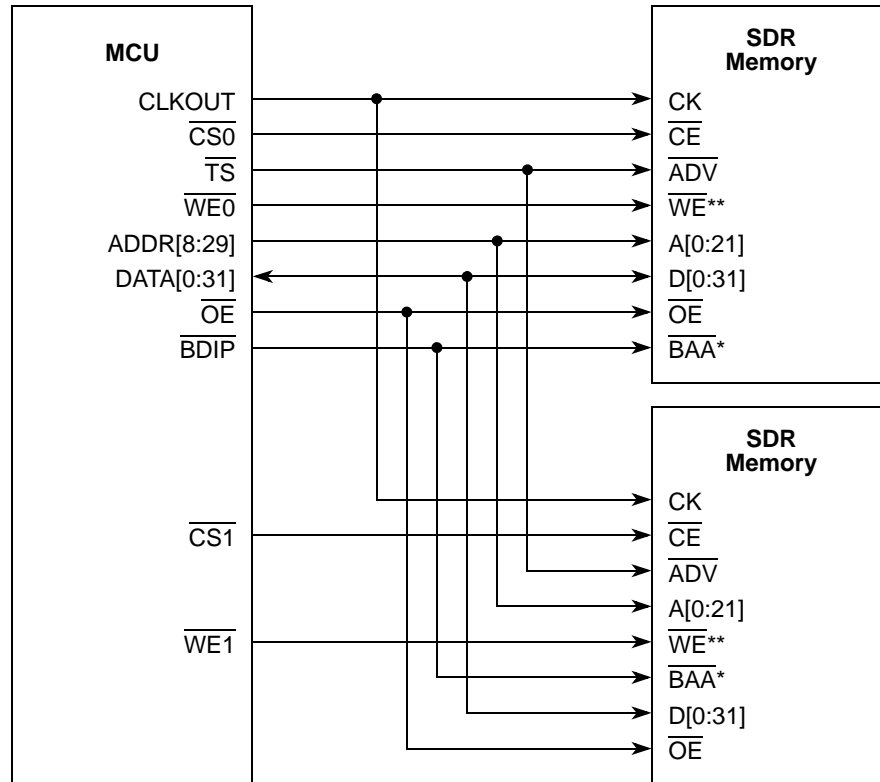


Figure 29-38. Write Operation to Asynchronous Memory, Three Initial Wait States

29.5.4 Connecting an MCU to Multiple Memories

The MCU can be connected to more than one memory at a time.

Figure 29-39 shows an example of two memories connected to one MCU.



- * May or may not be connected, depending on the memory used.
 ** Flash memories typically use one WE signal as shown, RAMs use two or four (16-bit or 32-bit).

Figure 29-39. MCU Connected to Multiple Memories

29.5.5 Dual-MCU Operation with Reduced Pinout MCUs

Some MCUs with this EBI may not have all the pins described in this document pinned out for a particular package. Some of the most common pins to be removed are AD[16:31], arbitration pins (\overline{BB} , \overline{BG} , \overline{BR}), and TSIZ[0:1]. This section describes how to configure dual-MCU systems for each of these scenarios. More than one section may apply if the applicable pins are not present on one or both MCUs.

NOTE

The MPC5510 does not have TSIZ[0:1] or arbitration pins (\overline{BB} , \overline{BR} , \overline{BG}).

29.5.5.1 Connecting 16-Bit MCU to 32-Bit MCU (Master/Master or Master/Slave)

This scenario is straightforward. Connect AD[0:15] between both MCUs, and configure both for 16-bit data bus mode operation (DBM=1 in EBI_MCR). 32-bit external memories are not supported in this scenario.

29.5.5.2 No Transfer Acknowledge (\overline{TA}) Pin

If an MCU has no \overline{TA} pin available, this restricts the MCU to chip-select accesses only. Non-chip-select accesses have no way for the EBI to know which cycle to latch the data. The EBI has no built-in protection

to prevent non-chip-select accesses in this scenario; the user must set up chip selects and external memories correctly to ensure all external accesses fall in a valid chip select region.

29.5.5.3 No Transfer Error ($\overline{\text{TEA}}$) Pin

If an MCU has no $\overline{\text{TEA}}$ pin available, this eliminates the feature of terminating an access with $\overline{\text{TEA}}$. This means if an access times out in the EBI bus monitor, the EBI (master) will still terminate the access early, but there will be no external visibility of this termination, so the slave device might drive data much later, when a subsequent access is already underway. Therefore, the EBI bus monitor must be disabled when no $\overline{\text{TEA}}$ pin exists.

Chapter 30 FlexRay Communication Controller (FLEXRAY)

30.1 Introduction

30.1.1 Reference

The following documents are referenced.

- *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*
- *FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A*

30.1.2 Glossary

This section provides a list of terms used in the description of the FlexRay block.

Table 30-1. List of Terms (Sheet 1 of 2)

Term	Definition
BCU	Buffer Control Unit. Handles message buffer access.
BMIF	Bus Master Interface. Provides master access to FlexRay memory block.
CC	Communication Controller
CDC	Clock Domain Crosser
CHI	Controller Host Interface
Cycle length in μ T	The actual length of a cycle in μ T for the ideal controller (+/- 0 ppm)
EBI	External Bus Interface
FRM	FlexRay Memory. Memory to store message buffer payload, header, and status, and to store synchronization frame related tables.
FSS	Frame Start Sequence
HIF	Host Interface. Provides host access to FlexRay block.
Host	The FlexRay CC host MCU
LUT	Look Up Table. Stores message buffer header index value.
MB	Message Buffer
MBIDX	Message Buffer Index: the position of a header field entry within the header area. If the header area is accessed as an array, this is the same as the array index of the entry.
MBNum	Message Buffer Number: Position of message buffer configuration registers within the register map. For example, Message Buffer Number 5 corresponds to the MBCCS5 register.
MCU	Microcontroller Unit
μ T	Microtick
MT	Macrotick
MTS	Media Access Test Symbol
NIT	Network Idle Time

Table 30-1. List of Terms (Sheet 2 of 2)

Term	Definition
PE	Protocol Engine
POC	Protocol Operation Control. Each state of the POC is denoted by <i>POC:state</i>
Rx	Reception
SEQ	Sequencer Engine
TCU	Time Control Unit
Tx	Transmission

30.1.3 Color Coding

Throughout this chapter types of items are highlighted through the use of an italicized color font.

FlexRay protocol parameters, constants and variables are highlighted with *blue italics*. An example is the parameter *gdActionPointOffset*.

FlexRay protocol states are highlighted in *green italics*. An example is the state *POC:normal active*.

30.1.4 Overview

The FlexRay block is a FlexRay communication controller that implements the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

The FlexRay block has three main components:

- Controller host interface (CHI)
- Protocol engine (PE)
- Clock domain crossing unit (CDC)

A block diagram of the FlexRay block with its surrounding modules is given in [Figure 30-1](#).

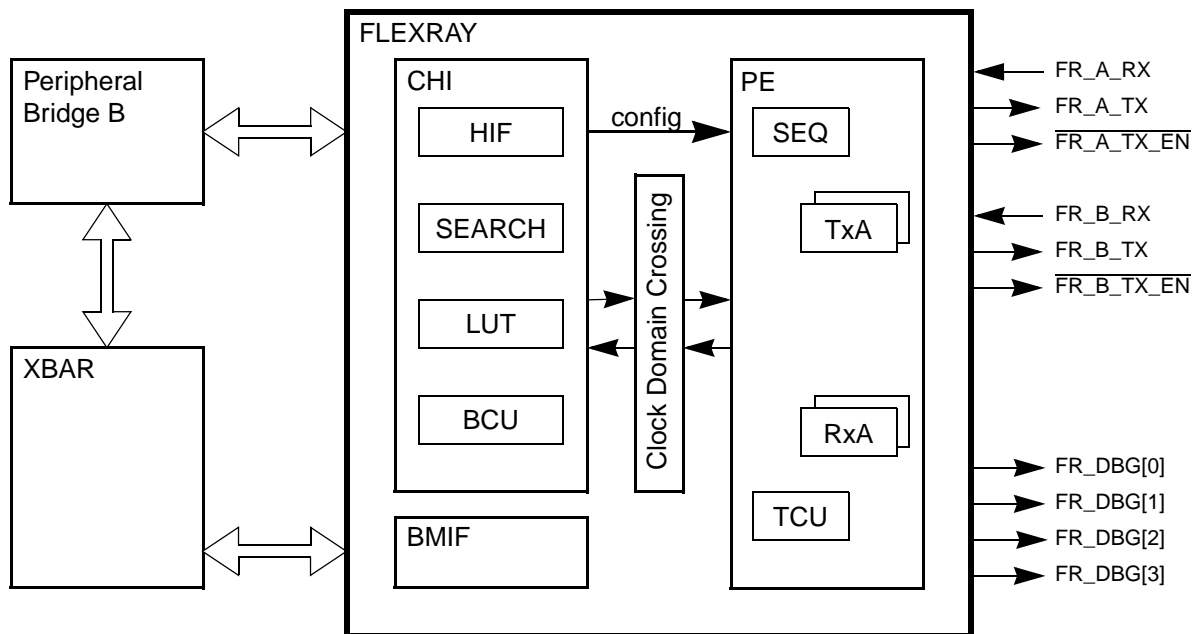


Figure 30-1. FLEXRAY Block Diagram

The protocol engine has two transmitter units TxA and TxB and two receiver units RxA and RxB for sending and receiving frames through the two FlexRay channels. The time control unit (TCU) is responsible for maintaining global clock synchronization to the FlexRay network. The overall activity of the PE is controlled by the sequencer engine (SEQ).

The controller host interface provides host access to the module's configuration, control, and status registers, as well as to the message buffer configuration, control, and status registers. The message buffers themselves, which contain the frame header and payload data received or to be transmitted, and the slot status information, are stored in the FlexRay Memory (FRM).

The clock domain crossing unit implements signal crossing from the CHI clock domain to the PE clock domain and vice versa, to allow for asynchronous PE and CHI clock domains.

The FlexRay block stores the frame header and payload data of frames received or of frames to be transmitted in the FRM. The application accesses the FRM to retrieve and provide the frames to be processed by the FlexRay block. In addition to the frame header and payload data, the FlexRay block stores the synchronization frame related tables in the FRM for application processing.

The FlexRay Memory is located in the system memory of the MCU. The FlexRay block has access to the FRM via its bus master interface (BMIF). The host provides the start address of the FRM window within the system memory by programming the [System Memory Base Address High Register \(SYMBADHR\)](#) and [System Memory Base Address Low Register \(SYMBADLR\)](#). All FRM related offsets are stored in offset registers. The physical address pointer into the FRM window of the MCU system memory is calculated using the offset values the FlexRay memory base address.

NOTE

The FlexRay block does not provide a memory protection scheme for the FlexRay Memory.

30.1.5 Features

The FlexRay block provides the following features:

- *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* compliant protocol implementation
- *FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A* compliant bus driver interface
- Single channel support
 - FlexRay Port A can be configured to be connected either to physical FlexRay channel A or physical FlexRay channel B.
- FlexRay bus data rates of 10 Mbit/s, 8 Mbit/s, 5 Mbit/s, and 2.5 Mbit/s supported
- Internal oscillator or internal PLL¹ clocking of the protocol engine
- 64 configurable message buffers with
 - Individual frame ID filtering
 - Individual channel ID filtering
 - Individual cycle counter filtering
- Message buffer header, status and payload data stored in dedicated FlexRay memory
 - Allows for flexible and efficient message buffer implementation
 - Consistent data access ensured by means of buffer locking scheme
 - Application can lock multiple buffers at the same time
- Size of message buffer payload data section configurable from 0 up to 254 bytes
- Two independent message buffer segments with configurable size of payload data section
 - Each segment can contain message buffers assigned to the static segment and message buffers assigned to the dynamic segment at the same time
- Zero padding for transmit message buffers in static segment
 - Applied when the frame payload length exceeds the size of the message buffer data section
- Transmit message buffers configurable with state/event semantics
- Message buffers can be configured as
 - Receive message buffer
 - Single buffered transmit message buffer
 - Double buffered transmit message buffer (combines two single buffered message buffer)
- Individual message buffer reconfiguration supported
 - Means provided to safely disable individual message buffers
 - Disabled message buffers can be reconfigured

1. Due to the tight timing requirements and overall system requirements of FlexRAY systems, usage of the PLL as the clock source has not been fully evaluated. It is recommended to use a 40 MHz crystal for the clock source.

- Two independent receive FIFOs
 - One receive FIFO per channel
 - Up to 255 entries for each FIFO
 - Global frame ID filtering, based on both value/mask filters and range filters
 - Global channel ID filtering
 - Global message ID filtering for the dynamic segment
- Four configurable slot error counters
- Four dedicated slot status indicators
 - Used to observe slots without using receive message buffers
- Measured value indicators for the clock synchronization
 - Internal synchronization frame ID and synchronization frame measurement tables can be copied into the FlexRay Memory
- Fractional macroticks are supported for clock correction
- Maskable interrupt sources provided via individual and combined interrupt lines
- One absolute timer
- One timer that can be configured to absolute or relative

30.1.6 Modes of Operation

This section describes the basic operational power modes of the FlexRay block.

30.1.6.1 Disabled Mode

This is the mode the FlexRay block enters during hard reset. The FlexRay block indicates that it is in the disabled mode by negating the module enable bit MEN in the [Module Configuration Register \(MCR\)](#).

No communication is performed on the FlexRay bus.

All registers with the write access conditions *Any Time* and *Disabled Mode* can be accessed for writing as stated in [Section 30.5.2, “Register Descriptions”](#).

The application configures the FlexRay block by accessing the configuration bits and fields in the [Module Configuration Register \(MCR\)](#).

30.1.6.1.1 Leave Disabled Mode

The FlexRay block leaves the disabled mode and enters the normal mode, when the application writes 1 to the module enable bit MEN in the [Module Configuration Register \(MCR\)](#)

NOTE

When the FlexRay block was enabled, it cannot be disabled the later on.

30.1.6.2 Normal Mode

In this mode the FlexRay block is fully functional. The FlexRay block indicates that it is in normal mode by asserting the module enable bit MEN in the [Module Configuration Register \(MCR\)](#).

30.1.6.2.1 Enter Normal Mode

This mode is entered when the application requests the FlexRay block to leave the disabled mode. If the normal mode was entered by leaving the disabled mode, the application has to perform the protocol initialization described in [30.7.1.2, “Protocol Initialization”](#) to achieve full FlexRay functionality.

Depending on the values of the SCM, CHA, and CHB bits in the [Module Configuration Register \(MCR\)](#), the corresponding FlexRay bus driver ports are enabled and driven.

30.2 External Signal Description

This section lists and describes the FlexRay block signals, connected to external pins. These signals are summarized in [Table 30-2](#) and described in detail in [Section 30.2.1, “Detailed Signal Descriptions.”](#)

NOTE

The off-chip signals FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ are available on each package option. The availability of the other off-chip signals depends on the package option.

Table 30-2. External Signal Properties

Name	Direction	Active	Reset	Function
FR_A_RX	Input	—	—	Receive Data Channel A
FR_A_TX	Output	—	1	Transmit Data Channel A
$\overline{\text{FR_A_TX_EN}}$	Output	Low	1	Transmit Enable Channel A
FR_B_RX	Input	—	—	Receive Data Channel B
FR_B_TX	Output	—	1	Transmit Data Channel B
$\overline{\text{FR_B_TX_EN}}$	Output	Low	1	Transmit Enable Channel B
FR_DBG[0]	Output	—	0	Debug Strobe Signal 0
FR_DBG[1]	Output	—	0	Debug Strobe Signal 1
FR_DBG[2]	Output	—	0	Debug Strobe Signal 2
FR_DBG[3]	Output	—	0	Debug Strobe Signal 3

30.2.1 Detailed Signal Descriptions

This section provides a detailed description of the FlexRay block signals, connected to external pins.

30.2.1.1 FR_A_RX — Receive Data Channel A

The FR_A_RX signal carries the receive data for channel A from the corresponding FlexRay bus driver.

30.2.1.2 FR_A_TX — Transmit Data Channel A

The FR_A_TX signal carries the transmit data for channel A to the corresponding FlexRay bus driver.

30.2.1.3 $\overline{\text{FR_A_TX_EN}}$ — Transmit Enable Channel A

The $\overline{\text{FR_A_TX_EN}}$ signal indicates to the FlexRay bus driver that the FlexRay block is attempting to transmit data on channel A.

30.2.1.4 FR_B_RX — Receive Data Channel B

The FR_B_RX signal carries the receive data for channel B from the corresponding FlexRay bus driver.

30.2.1.5 FR_B_TX — Transmit Data Channel B

The FR_B_TX signal carries the transmit data for channel B to the corresponding FlexRay bus driver.

30.2.1.6 $\overline{\text{FR_B_TX_EN}}$ — Transmit Enable Channel B

The $\overline{\text{FR_B_TX_EN}}$ signal indicates to the FlexRay bus driver that the FlexRay block is attempting to transmit data on channel B.

30.2.1.7 FR_DBG[3], FR_DBG[2], FR_DBG[1], FR_DBG[0] — Strobe Signals

These signals provide the selected debug strobe signals. For details on the debug strobe signal selection refer to [Section 30.6.16, “Strobe Signal Support.”](#)

30.3 Controller Host Interface Clocking

The clock for the CHI is derived from the system bus clock and has the same phase and frequency. Because the FlexRay protocol requires data delivery at fixed points in time, the memory read cycles from the FRM must be finished after a fixed amount of time. To ensure this, a minimum frequency f_{chi} of the CHI clock is required, which is given in [Equation 30-1](#).

$$f_{\text{chi}} \geq 32\text{MHz} \quad \text{Eqn. 30-1}$$

Additional requirements for the minimum frequency of the CHI clock result from the number of message buffer. The requirement is provided in [Section 30.7.3, “Number of Usable Message Buffers.”](#)

30.4 Protocol Engine Clocking

The clock for the protocol engine can be generated by two sources. The first source is the internal crystal oscillator and the second source is an internal PLL¹. The clock source to be used is selected by the clock source select bit CLKSEL in the [Module Configuration Register \(MCR\)](#).

1. Due to the tight timing requirements and overall system requirements of FlexRAY systems, usage of the PLL as the clock source has not been fully evaluated. It is recommended to use a 40 MHz crystal for the clock source.

30.4.1 Oscillator Clocking

If the protocol engine is clocked by the internal crystal oscillator, an 40 MHz crystal or 40 MHz CMOS compatible clock must be connected to the oscillator pins. The crystal or clock must fulfil the requirements given by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

The PLL predivider has to be configured for divide-by-2 operation.

30.4.2 PLL Clocking

If the protocol engine is clocked by the internal PLL¹, the frequency of the PE clock source is system clock / 2. The system clock frequency must be 80 MHz.

30.5 Memory Map and Register Description

The FlexRay block occupies 768 bytes of address space starting at the FlexRay block's base address defined by the memory map of the MCU.

30.5.1 Memory Map

The complete memory map of the FlexRay block is shown in [Table 30-3](#). The addresses presented here are the offsets relative to the FlexRay block base address, which is defined by the MCU address map.

Table 30-3. FlexRay Memory Map (Sheet 1 of 4)

Offset from FLEXRAY_BASE (0xFFFD_8000)	Register	Access
Module Configuration and Control		
0x0000	Module Version Register (MVR)	R
0x0002	Module Configuration Register (MCR)	R/W
0x0004	System Memory Base Address High Register (SYMBADHR)	R/W
0x0006	System Memory Base Address Low Register (SYMBADLR)	R/W
0x0008	Strobe Signal Control Register (STBSCR)	R/W
0x000A	Reserved	R
0x000C	Message Buffer Data Size Register (MBDSR)	R/W
0x000E	Message Buffer Segment Size and Utilization Register (MBSSUTR)	R/W
Test Registers		
0x0010	Reserved	R
0x0012	Reserved	R
Interrupt and Error Handling		
0x0014	Protocol Operation Control Register (POCR)	R/W
0x0016	Global Interrupt Flag and Enable Register (GIFER)	R/W
0x0018	Protocol Interrupt Flag Register 0 (PIFR0)	R/W
0x001A	Protocol Interrupt Flag Register 1 (PIFR1)	R/W

1. Due to the tight timing requirements and overall system requirements of FlexRAY systems, usage of the PLL as the clock source has not been fully evaluated. It is recommended to use a 40 MHz crystal for the clock source.

Table 30-3. FlexRay Memory Map (Sheet 2 of 4)

Offset from FLEXRAY_BASE (0xFFFFD_8000)	Register	Access
0x001C	Protocol Interrupt Enable Register 0 (PIER0)	R/W
0x001E	Protocol Interrupt Enable Register 1 (PIER1)	R/W
0x0020	CHI Error Flag Register (CHIERFR)	R/W
0x0022	Message Buffer Interrupt Vector Register (MBIVEC)	R
0x0024	Channel A Status Error Counter Register (CASERCR)	R
0x0026	Channel B Status Error Counter Register (CBSERCR)	R
Protocol Status		
0x0028	Protocol Status Register 0 (PSR0)	R
0x002A	Protocol Status Register 1 (PSR1)	R
0x002C	Protocol Status Register 2 (PSR2)	R
0x002E	Protocol Status Register 3 (PSR3)	R/W
0x0030	Macrotick Counter Register (MTCTR)	R
0x0032	Cycle Counter Register (CYCTR)	R
0x0034	Slot Counter Channel A Register (SLCTAR)	R
0x0036	Slot Counter Channel B Register (SLCTBR)	R
0x0038	Rate Correction Value Register (RTCORVR)	R
0x003A	Offset Correction Value Register (OFCORVR)	R
0x003C	Combined Interrupt Flag Register (CIFRR)	R
0x003E	System Memory Access Time-Out Register (SYMATOR)	R/W
Sync Frame Counter and Tables		
0x0040	Sync Frame Counter Register (SFCNTR)	R
0x0042	Sync Frame Table Offset Register (SFTOR)	R/W
0x0044	Sync Frame Table Configuration, Control, Status Register (SFTCCSR)	R/W
Sync Frame Filter		
0x0046	Sync Frame ID Rejection Filter Register (SFIDRFR)	R/W
0x0048	Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)	R/W
0x004A	Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)	R/W
Network Management Vector		
0x004C	Network Management Vector Register 0 (NMVR0)	R
0x004E	Network Management Vector Register 1 (NMVR1)	R
0x0050	Network Management Vector Register 2 (NMVR2)	R
0x0052	Network Management Vector Register 3 (NMVR3)	R
0x0054	Network Management Vector Register 4 (NMVR4)	R
0x0056	Network Management Vector Register 5 (NMVR5)	R
0x0058	Network Management Vector Length Register (NMVLR)	R/W
Timer Configuration		
0x005A	Timer Configuration and Control Register (TICCR)	R/W
0x005C	Timer 1 Cycle Set Register (T1CYSR)	R/W
0x005E	Timer 1 Macrotick Offset Register (T1MTOR)	R/W

Table 30-3. FlexRay Memory Map (Sheet 3 of 4)

Offset from FLEXRAY_BASE (0xFFFFD_8000)	Register	Access
0x0060	Timer 2 Configuration Register 0 (TI2CR0)	R/W
0x0062	Timer 2 Configuration Register 1 (TI2CR1)	R/W
Slot Status Configuration		
0x0064	Slot Status Selection Register (SSSR)	R/W
0x0066	Slot Status Counter Condition Register (SSCCR)	R/W
Slot Status		
0x0068	Slot Status Register 0 (SSR0)	R
0x006A	Slot Status Register 1 (SSR1)	R
0x006C	Slot Status Register 2 (SSR2)	R
0x006E	Slot Status Register 3 (SSR3)	R
0x0070	Slot Status Register 4 (SSR4)	R
0x0072	Slot Status Register 5 (SSR5)	R
0x0074	Slot Status Register 6 (SSR6)	R
0x0076	Slot Status Register 7 (SSR7)	R
0x0078	Slot Status Counter Register 0 (SSCR0)	R
0x007A	Slot Status Counter Register 1 (SSCR1)	R
0x007C	Slot Status Counter Register 2 (SSCR2)	R
0x007E	Slot Status Counter Register 3 (SSCR3)	R
MTS Generation		
0x0080	MTS A Configuration Register (MTSACFR)	R/W
0x0082	MTS B Configuration Register (MTSBCFR)	R/W
Shadow Buffer Configuration		
0x0084	Receive Shadow Buffer Index Register (RSBIR)	R/W
Receive FIFO — Configuration		
0x0086	Receive FIFO Selection Register (RFSR)	R/W
0x0088	Receive FIFO Start Index Register (RFSIR)	R/W
0x008A	Receive FIFO Depth and Size Register (RFDSR)	R/W
Receive FIFO - Status		
0x008C	Receive FIFO A Read Index Register (RFARIR)	R
0x008E	Receive FIFO B Read Index Register (RFBIRIR)	R
Receive FIFO - Filter		
0x0090	Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)	R/W
0x0092	Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)	R/W
0x0094	Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)	R/W
0x0096	Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)	R/W
0x0098	Receive FIFO Range Filter Configuration Register (RFRFCFR)	R/W
0x009A	Receive FIFO Range Filter Control Register (RFRFCTR)	R/W
Dynamic Segment Status		
0x009C	Last Dynamic Transmit Slot Channel A Register (LDTXSLAR)	R

Table 30-3. FlexRay Memory Map (Sheet 4 of 4)

Offset from FLEXRAY_BASE (0xFFFFD_8000)	Register	Access
0x009E	Last Dynamic Transmit Slot Channel B Register (LDTXSLBR)	R
Protocol Configuration		
0x00A0	Protocol Configuration Register 0 (PCR0)	R/W
...	...	–
0x00DC	Protocol Configuration Register 30 (PCR30)	R/W
0x00DE	Reserved	R
...		
0x00FE		
Message Buffers Configuration, Control, Status		
0x0100	Message Buffer Configuration, Control, Status Register 0 (MBCCSR0)	R/W
0x0102	Message Buffer Cycle Counter Filter Register 0 (MBCCFR0)	R/W
0x0104	Message Buffer Frame ID Register 0 (MBFIDR0)	R/W
0x0106	Message Buffer Index Register 0 (MBIDXR0)	R/W
...
0x02F8	Message Buffer Configuration, Control, Status Register 63 (MBCCSR63)	R/W
0x02FA	Message Buffer Cycle Counter Filter Register 63 (MBCCFR63)	R/W
0x02FC	Message Buffer Frame ID Register 63 (MBFIDR63)	R/W
0x02FE	Message Buffer Index Register 63 (MBIDXR63)	R/W

30.5.2 Register Descriptions

This section provides detailed descriptions of all registers in ascending address order, presented as 16-bit wide entities

Table 30-4 provides a key for the register figures and register tables.

Table 30-4. Register Access Conventions

Convention	Description
	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
R*	Reserved bit or field, will not be changed. Application must not write any value different from the reset value.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
Register Field Types	
rwm	A read/write bit that may be modified by a hardware in some fashion other than by a reset.
w1c	Write one to clear. A flag bit that can be read, is cleared by writing a one, writing 0 has no effect.
Reset Value	
0	Resets to zero.
1	Resets to one.
–	Not defined after reset and not affected by reset.

30.5.2.1 Register Reset

All registers except the [Message Buffer Cycle Counter Filter Registers \(MBCCFRn\)](#), [Message Buffer Frame ID Registers \(MBFIDRn\)](#), and [Message Buffer Index Registers \(MBIDXRn\)](#) are reset to their reset value on system reset. The registers mentioned above are located in physical memory blocks and, thus, they are not affected by reset. For some register fields, additional reset conditions exist. These additional reset conditions are mentioned in the detailed description of the register. The additional reset conditions are explained in [Table 30-5](#).

Table 30-5. Additional Register Reset Conditions

Condition	Description
Protocol RUN Command	The register field is reset when the application writes to RUN command "0101" to the POCCMD field in the Protocol Operation Control Register (POCR) .
Message Buffer Disable	The register field is reset when the application has disabled the message buffer. This happens when the application writes 1 to the message buffer disable trigger bit MBCCSRn.EDT while the message buffer is enabled (MBCCSn.EDS = 1) and the FlexRay block grants the disable to the application by clearing the MBCCSRn.EDS bit.

30.5.2.2 Register Write Access

This section describes the write access restriction terms that apply to all registers.

30.5.2.2.1 Register Write Access Restriction

For each register bit and register field, the write access conditions are specified in the detailed register description. A description of the write access conditions is given in [Table 30-6](#). If, for a specific register bit or field, none of the given write access conditions is fulfilled, any write attempt to this register bit or field is ignored without any notification. The values of the bits or fields are not changed. The condition term [A or B] indicates that the register or field can be written to if at least one of the conditions is fulfilled.

Table 30-6. Register Write Access Restrictions

Condition	Indication	Description
Any Time	—	No write access restriction.
Disabled Mode	MCR.MEN = 0	Write access only when the FlexRay block is in Disabled Mode.
Normal Mode	MCR.MEN = 1	Write access only when the FlexRay block is in Normal Mode.
<i>POC:config</i>	PSR0.PROTSTATE = <i>POC:config</i>	Write access only when the Protocol is in the <i>POC:config</i> state.
MB_DIS	MBCCSRn.EDS = 0	Write access only when the related Message Buffer is disabled.
MB_LCK	MBCCSRn.LCKS = 1	Write access only when the related Message Buffer is locked.

30.5.2.2.2 Register Write Access Requirements

For some of the registers, a 16-bit wide write access is required to ensure correct operation. This write access requirement is stated in the detailed register description for each register affected

30.5.2.2.3 Internal Register Access

The following memory-mapped registers are used to access multiple internal registers.

- Strobe Signal Control Register (STBSCR)
- Slot Status Selection Register (SSSR)
- Slot Status Counter Condition Register (SSCCR)
- Receive Shadow Buffer Index Register (RSBIR)

Each of these memory-mapped registers provides a SEL field and a WMD bit. The SEL field is used to select the internal register. The WMD bit controls the write mode. If the WMD bit is set to 0 during the write access, all fields of the internal register are updated. If the WMD bit set to 1, only the SEL field is changed. All other fields of the internal register remain unchanged. This allows for reading back the values of the selected internal register in a subsequent read access.

30.5.2.3 Module Version Register (MVR)

Base + 0x0000

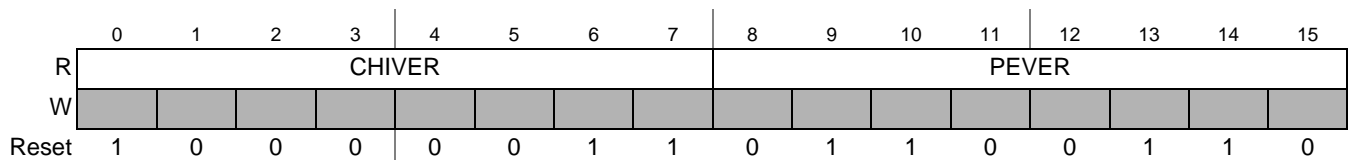


Figure 30-2. Module Version Register (MVR)

This register provides the FlexRay block version number. The module version number is derived from the CHI version number and the PE version number.

Table 30-7. MVR Field Descriptions

Field	Description
CHIVER	CHI Version Number. This field provides the version number of the controller host interface.
PEVER	PE Version Number. This field provides the version number of the protocol engine.

30.5.2.4 Module Configuration Register (MCR)

Base + 0x0002

Write: MEN, SCM, CHB, CHA, CLKSEL, BITRATE: Disabled Mode
SFFE: Disabled Mode or *POC:config*

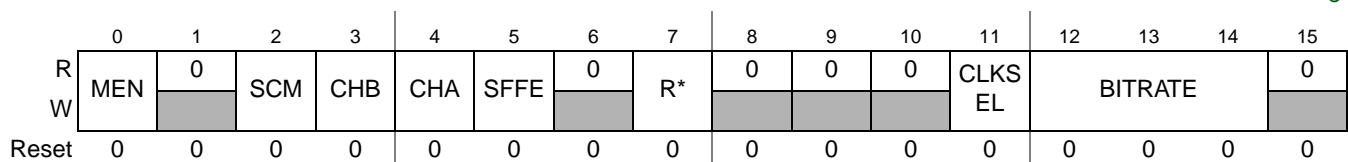


Figure 30-3. Module Configuration Register (MCR)

This register defines the global configuration of the FlexRay block.

Table 30-8. MCR Field Descriptions

Field	Description
MEN	Module Enable. This bit indicates whether or not the FlexRay block is in the disabled mode. The application requests the FlexRay block to leave the disabled mode by writing 1 to this bit. Before leaving the disabled mode, the application must configure the SCM, CHB, CHA, TMODE, CLKSEL, BITRATE values. For details see Section 30.1.6, “Modes of Operation” . 0 Write: ignored, FlexRay block disable not possible Read: FlexRay block disabled 1 Write: enable FlexRay block Read: FlexRay block enabled Note: If the FlexRay block is enabled it can not be disabled.
SCM	Single Channel Device Mode. This control bit defines the channel device mode of the FlexRay block as described in Section 30.6.10, “Channel Device Modes” . 0 FlexRay block works in dual channel device mode 1 FlexRay block works in single channel device mode
CHB CHA	Channel Enable. protocol related parameter: <i>pChannels</i> The semantic of these control bits depends on the channel device mode controlled by the SCM bit and is given Table 30-9 .
SFFE	Synchronization Frame Filter Enable. This bit controls the filtering for received synchronization frames. For details see Section 30.6.15, “Sync Frame Filtering” . 0 Synchronization frame filtering disabled 1 Synchronization frame filtering enabled
R*	Reserved. This bit is reserved. It is read as 0. Application must not write 1 to this bit.
CLKSEL	Protocol Engine Clock Source Select . This bit is used to select the clock source for the protocol engine. 0 PE clock source is generated by on-chip crystal oscillator. 1 PE clock source is generated by on-chip PLL ¹ .
BITRATE	FlexRay Bus Bit Rate. This bit field defines the bit rate of the flexray channels according to Table 30-10 .

¹ Due to the tight timing requirements and overall system requirements of FlexRAY systems, usage of the PLL as the clock source has not been fully evaluated. It is recommended to use a 40 MHz crystal for the clock source.

Table 30-9. FlexRay Channel Selection (Sheet 1 of 2)

SCM	CHB	CHA	Description
Dual Channel Device Modes			
0	0	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by FlexRay block ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by FlexRay block PE channel 0 idle PE channel 1 idle
	0	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ driven by FlexRay block ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by FlexRay block PE channel 0 active PE channel 1 idle
	1	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by FlexRay block ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ driven by FlexRay block PE channel 0 idle PE channel 1 active
	1	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ driven by FlexRay block ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ driven by FlexRay block PE channel 0 active PE channel 1 active

Table 30-9. FlexRay Channel Selection (Sheet 2 of 2)

SCM	CHB	CHA	Description
Single Channel Device Mode			
1	0	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by FlexRay block ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by FlexRay block PE channel 0 idle PE channel 1 idle
	0	1	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ driven by FlexRay block ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by FlexRay block PE channel 0 active PE channel 1 idle
	1	0	ports FR_A_RX, FR_A_TX, and $\overline{\text{FR_A_TX_EN}}$ driven by FlexRay block ports FR_B_RX, FR_B_TX, and $\overline{\text{FR_A_TX_EN}}$ not driven by FlexRay block PE channel 0 active, uses cCrclnit[B] (see Figure 30-134) PE channel 1 idle
	1	1	reserved

Table 30-10. FlexRay Channel Bit Rate Selection

MCR[BITRATE]	FlexRay Channel Bit Rate [Mbit/s]
000	10.0
001	5.0
010	2.5
011	8.0
100	reserved
101	reserved
110	reserved
111	reserved

30.5.2.5 System Memory Base Address High Register (SYMBADHR) and System Memory Base Address Low Register (SYMBADLR)

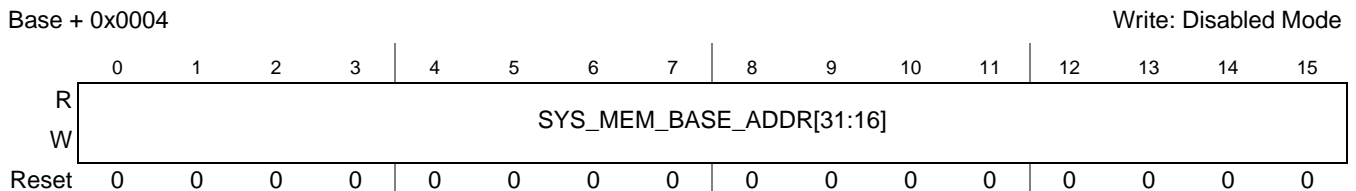


Figure 30-4. System Memory Base Address High Register (SYMBADHR)

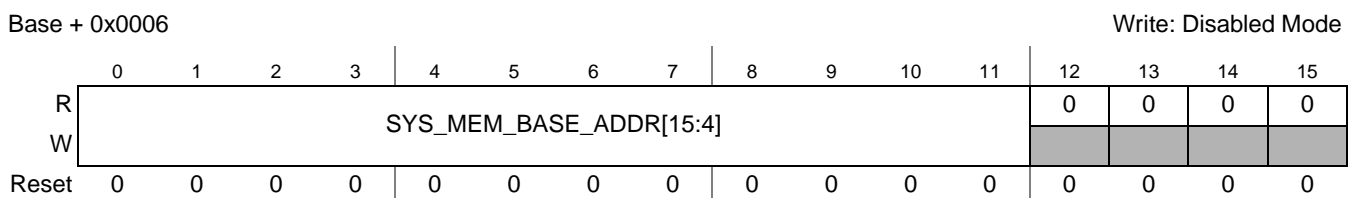


Figure 30-5. System Memory Base Address Low Register (SYMBADLR)

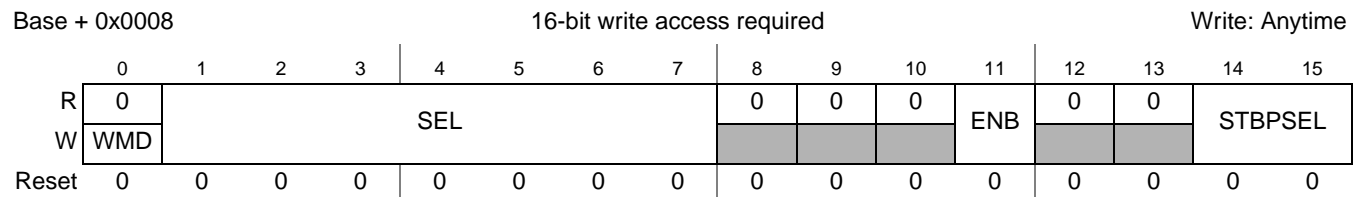
NOTE

The system memory base address must be set before the FlexRay block is enabled.

The system memory base address registers define the base address of the FRM within the system memory. The base address is used by the BMIF to calculate the physical memory address for system memory accesses.

Table 30-11. SYMBADHR and SYMBADLR Field Descriptions

Field	Description
SYMBADHR SYMBADLR	This base address will be added to all system memory offset values stored in registers or calculated in the FlexRay block before the FlexRay block accesses the system memory via its bus master interface. The system memory base address must be aligned to an 16-byte boundary.

30.5.2.6 Strobe Signal Control Register (STBSCR)**Figure 30-6. Strobe Signal Control Register (STBSCR)**

This register is used to assign the individual protocol timing related strobe signals given in [Table 30-13](#) to the external strobe ports. Each strobe signal can be assigned to at most one strobe port. Each write access to registers overwrites the previously written ENB and STBPSEL values for the signal indicated by SEL. If more than one strobe signal is assigned to one strobe port, the current values of the strobe signals are combined with a binary OR and presented at the strobe port. If no strobe signal is assigned to a strobe port, the strobe port carries logic 0. For more detailed and timing information refer to [Section 30.6.16, “Strobe Signal Support”](#).

NOTE

In single channel device mode, channel B related strobe signals are undefined and should not be assigned to the strobe ports.

Table 30-12. STBSCR Field Descriptions (Sheet 1 of 2)

Field	Description
WMD	Write Mode. This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	Strobe Signal Select. This control field selects one of the strobe signals given in Table 30-13 to be enabled or disabled and assigned to one of the four strobe ports given in Table 30-13 .

Table 30-12. STBSCR Field Descriptions (Sheet 2 of 2)

Field	Description
ENB	Strobe Signal Enable. The control bit is used to enable and to disable the strobe signal selected by STBSSEL. 0 Strobe signal is disabled and not assigned to any strobe port. 1 Strobe signal is enabled and assigned to the strobe port selected by STBPSEL.
STBPSEL	Strobe Port Select. This field selects the strobe port that the strobe signal selected by the SEL is assigned to. All strobe signals that are enabled and assigned to the same strobe port are combined with a binary OR operation. 00 assign selected signal to FR_DBG[0] 01 assign selected signal to FR_DBG[1] 10 assign selected signal to FR_DBG[2] 11 assign selected signal to FR_DBG[3]

Table 30-13. Strobe Signal Mapping (Sheet 1 of 3)

SEL		Description	Channel	Type	Offset ¹	Reference
dec	hex					
0	0x00	poc_startup_state[0] (for coding see PSR0[4])	-	value	0	MT start
1	0x01	poc_startup_state[1] (for coding see PSR0[5])				
2	0x02	poc_startup_state[2] (for coding see PSR0[6])				
3	0x03	poc_startup_state[3] (for coding see PSR0[7])				
4	0x04	poc_state[0] (for coding see PSR0[8])				
5	0x05	poc_state[1] (for coding see PSR0[9])				
6	0x06	poc_state[2] (for coding see PSR0[10])				
7	0x07	channel idle indicator	A	level	+5	FR_A_RX
8	0x08		B			FR_B_RX
9	0x09	receive data after glitch filtering	A	value	+4	FR_A_RX
10	0x0A		B			FR_B_RX
11	0x0B	synchronization edge strobe	A	pulse	+4	FR_A_RX
12	0x0C		B			FR_B_RX
13	0x0D	header received	A	pulse	+4	FR_A_RX
14	0x0E		B			FR_B_RX
15	0x0F	wakeup symbol decoded	A	pulse	+5	FR_A_RX
16	0x10		B			FR_B_RX
17	0x11	MTS or CAS symbol decoded	A	pulse	+4	FR_A_RX
18	0x12		B			FR_B_RX
19	0x13	frame decoded	A	pulse	+4	FR_A_RX
20	0x14		B			FR_B_RX
21	0x15	channel idle detected	A	pulse	+4	FR_A_RX
22	0x16		B			FR_B_RX
23	0x17	start of communication element detected	A	pulse	+4	FR_A_RX
24	0x18		B			FR_B_RX
25	0x19	potential frame start channel	A	pulse	+4	FR_A_RX
26	0x1A		B			FR_B_RX
27	0x1B	wakeup collision detected	A	pulse	+5	FR_A_RX
28	0x1C		B			FR_B_RX
29	0x1D	content error detected	A	level	+4	FR_A_RX
30	0x1E		B			FR_B_RX

Table 30-13. Strobe Signal Mapping (Sheet 2 of 3)

SEL		Description	Channel	Type	Offset ¹	Reference
dec	hex					
31	0x1F	syntax error detected	A	pulse	+4	FR_A_RX
32	0x20		B			FR_B_RX
33	0x21	start transmission of wakeup pattern	A	pulse	-1	FR_A_TX
34	0x22		B			FR_B_TX
35	0x23	start transmission of MTS or CAS symbol	A	pulse	-1	FR_A_TX
36	0x24		B			FR_B_TX
37	0x25	start of transmission	A	pulse	-1	FR_A_TX
38	0x26		B			FR_B_TX
39	0x27	end of transmission	A	pulse	-1	FR_A_TX
40	0x28		B			FR_B_TX
41	0x29	static segment indicator	-	level	0	MT start
42	0x2A	dynamic segment indicator	-	level	0	MT start
43	0x2B	symbol window indicator	-	level	0	MT start
44	0x2C	NIT indicator	-	level	0	MT start
45	0x2D	action point	-	pulse	-1	FR_A_TX
46	0x2E	sync calculation complete ²	-	pulse	-	-
47	0x2F	start of offset correction	-	pulse	-2	MT start
48	0x30	cycle count[0]	-	value	-2	MT start
49	0x31	cycle count[1]				
50	0x32	cycle count[2]				
51	0x33	cycle count[3]				
52	0x34	cycle count[4]				
53	0x35	cycle count[5]				
54	0x36	slot count[0]	A	value	0	MT start
55	0x37	slot count[1]				
56	0x38	slot count[2]				
57	0x39	slot count[3]				
58	0x3A	slot count[4]				
59	0x3B	slot count[5]				
60	0x3C	slot count[6]				
61	0x3D	slot count[7]				
62	0x3E	slot count[8]				
63	0x3F	slot count[9]				
64	0x40	slot count[10]	B	value	0	MT start
65	0x41	slot count[0]				
66	0x42	slot count[1]				
67	0x43	slot count[2]				
68	0x44	slot count[3]				
69	0x45	slot count[4]				
70	0x46	slot count[5]				
71	0x47	slot count[6]				
72	0x48	slot count[7]				
73	0x49	slot count[8]				
74	0x4A	slot count[9]				
75	0x4B	slot count[10]	-	pulse	0	MT start
76	0x4C	cycle start				

Table 30-13. Strobe Signal Mapping (Sheet 3 of 3)

SEL		Description	Channel	Type	Offset ¹	Reference
dec	hex					
77	0x4D	slot start	A	pulse	0	MT start
78	0x4E		B			
79	0x4F	minislot start	-	pulse	0	MT start
80	0x50	arm	-	value	+1	MT start
81	0x51	mt	-	value	+1	MT start

¹ Given in PE clock cycles

² Indicates internal PE event not directly related to FlexRay bus timing

30.5.2.7 Message Buffer Data Size Register (MBDSR)

Base + 0x000C

Write: *POC:config*

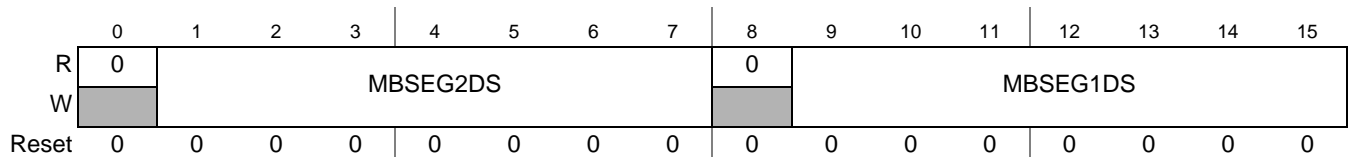


Figure 30-7. Message Buffer Data Size Register (MBDSR)

This register defines the size of the message buffer data section for the two message buffer segments in a number of two-byte entities.

The FlexRay block provides two independent segments for the individual message buffers. All individual message buffers within one segment have to have the same size for the message buffer data section. This size can be different for the two message buffer segments.

Table 30-14. MBDSR Field Descriptions

Field	Description
MBSEG2DS	Message Buffer Segment 2 Data Size. The field defines the size of the message buffer data section in two-byte entities for message buffers within the <i>second</i> message buffer segment.
MBSEG1DS	Message Buffer Segment 1 Data Size. The field defines the size of the message buffer data section in two-byte entities for message buffers within the <i>first</i> message buffer segment.

30.5.2.8 Message Buffer Segment Size and Utilization Register (MBSSUTR)

Base + 0x000E

Write: *POC:config*

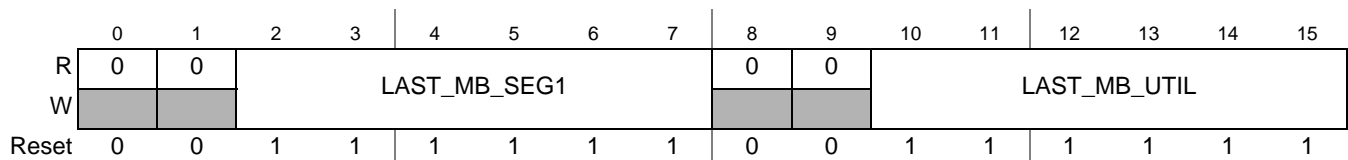


Figure 30-8. Message Buffer Segment Size and Utilization Register (MBSSUTR)

This register is used to define the last individual message buffer that belongs to the first message buffer segment and the number of the last used individual message buffer.

Table 30-15. MBSSUTR Field Descriptions

Field	Description
LAST_MB_SEG1	<p>Last Message Buffer In Segment 1. This field defines the message buffer number of the last individual message buffer that is assigned to the <i>first</i> message buffer segment. The individual message buffers in the <i>first</i> segment correspond to the message buffer control registers MBCCSRn, MBCCFRn, MBFIDRn, MBIDXRn with $n \leq \text{LAST_MB_SEG1}$. The first message buffer segment contains $\text{LAST_MB_SEG1}+1$ individual message buffers.</p> <p>Note: The <i>first</i> message buffer segment contains <i>at least</i> one individual message buffer.</p> <p>The individual message buffers in the <i>second</i> message buffer segment correspond to the message buffer control registers MBCCSRn, MBCCFRn, MBFIDRn, MBIDXRn with $\text{LAST_MB_SEG1} < n < 64$.</p> <p>Note: If $\text{LAST_MB_SEG1} = 63$ all individual message buffers belong to the <i>first</i> message buffer segment and the <i>second</i> message buffer segment is empty.</p>
LAST_MB_UTIL	<p>Last Message Buffer Utilized. This field defines the message buffer number of last utilized individual message buffer. The message buffer search engine examines all individual message buffer with a message buffer number $n \leq \text{LAST_MB_UTIL}$.</p> <p>Note: If $\text{LAST_MB_UTIL} = \text{LAST_MB_SEG1}$ all individual message buffers belong to the <i>first</i> message buffer segment and the <i>second</i> message buffer segment is empty.</p>

30.5.2.9 Protocol Operation Control Register (POCR)

Base + 0x0014

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EOC_AP		ERC_AP		BSY	0	0	0	POCCMD			
W	WME								WMC							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-9. Protocol Operation Control Register (POCR)

The application uses this register to issue

- protocol control commands
- external clock correction commands

Protocol control commands are issued by writing to the POCCMD field. For more information on protocol control commands, see [Section 30.7.4, “Protocol Control Command Execution”](#).

External clock correction commands are issued by writing to the EOC_AP and ERC_AP fields. For more information on external clock correction, refer to [Section 30.6.11, “External Clock Synchronization”](#).

Table 30-16. POCR Field Descriptions (Sheet 1 of 2)

Field	Description
WME	<p>Write Mode External Correction. This bit controls the write mode of the EOC_AP and ERC_AP fields.</p> <p>0 Write to EOC_AP and ERC_AP fields on register write.</p> <p>1 No write to EOC_AP and ERC_AP fields on register write.</p>
EOC_AP	<p>External Offset Correction Application. This field is used to trigger the application of the external offset correction value defined in the Protocol Configuration Register 29 (PCR29).</p> <p>00 do not apply external offset correction value</p> <p>01 reserved</p> <p>10 subtract external offset correction value</p> <p>11 add external offset correction value</p>

Table 30-16. POCCR Field Descriptions (Sheet 2 of 2)

Field	Description
ERC_AP	External Rate Correction Application. This field is used to trigger application of the external rate correction value defined in the Protocol Configuration Register 21 (PCR21) 00 do not apply external rate correction value 01 reserved 10 subtract external rate correction value 11 add external rate correction value
BSY	Protocol Control Command Write Busy. This status bit indicates the acceptance of the protocol control command issued by the application via the POCCMD field. The FlexRay block sets this status bit when the application has issued a protocol control command via the POCCMD field. The FlexRay block clears this status bit when protocol control command was accepted by the PE. When the application issues a protocol control command while the BSY bit is asserted, the FlexRay block ignores this command, sets the protocol command ignored error flag PCMI_EF in the CHI Error Flag Register (CHIERFR) , and will not change the value of the POCCMD field. 0 Command write idle, command accepted and ready to receive new protocol command. 1 Command write busy, command not yet accepted, not ready to receive new protocol command.
WMC	Write Mode Command. This bit controls the write mode of the POCCMD field. 0 Write to POCCMD field on register write. 1 Do not write to POCCMD field on register write.
POCCMD	Protocol Control Command. The application writes to this field to issue a protocol control command to the PE. The FlexRay block sends the protocol command to the PE immediately. While the transfer is running, the BSY bit is set. 0000 ALLOW_COLDSTART — Immediately activate capability of node to cold start cluster. 0001 ALL_SLOTS — Delayed ¹ transition to the all slots transmission mode. 0010 CONFIG — Immediately transition to the <i>POC:config</i> state. 0011 FREEZE — Immediately transition to the <i>POC:halt</i> state. 0100 READY, CONFIG_COMPLETE — Immediately transition to the <i>POC:ready</i> state. 0101 RUN — Immediately transition to the <i>POC:startup start</i> state. 0110 DEFAULT_CONFIG — Immediately transition to the <i>POC:default config</i> state. 0111 HALT — Delayed transition to the <i>POC:halt</i> state 1000 WAKEUP — Immediately initiate the wakeup procedure. 1001 reserved 1010 reserved 1011 reserved 1100 RESET ² — Immediately reset the Protocol Engine. 1101 reserved 1110 reserved 1111 reserved

¹ Delayed means on completion of current communication cycle.

² Additional to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*

After sending the RESET command, it is mandatory to execute the command sequence described in [Section 30.7.5, “Protocol Reset Command”](#) immediately, to reach the DEFAULT CONFIG state correctly.

30.5.2.10 Global Interrupt Flag and Enable Register (GIFER)

Base + 0x0016

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIF	PRIF	CHIF	WUP IF	FNEB IF	FNEA IF	RBIF	TBIF	MIE	PRIE	CHIE	WUP IE	FNEB IE	FNEA IE	RBIE	TBIE
W				w1c	w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-10. Global Interrupt Flag and Enable Register (GIFER)

This register provides the means to control some of the interrupt request lines and provides the corresponding interrupt flags. The interrupt flags MIF, PRIF, CHIF, RBIF, and TBIF are the outcome of a binary OR of the related individual interrupt flags and interrupt enables. The generation scheme for these flags is depicted in [Figure 30-143](#). For more details on interrupt generation, see [Section 30.6.19, “Interrupt Support](#). These flags are cleared automatically when all of the corresponding interrupt flags or interrupt enables in the related interrupt flag and enable registers are cleared by the application.

Table 30-17. GIFER Field Descriptions (Sheet 1 of 3)

Field	Description
MIF	Module Interrupt Flag. This flag is set if at least one of the other interrupt flags in this register is asserted and the related interrupt enable is asserted, too. The FlexRay block generates the module interrupt request if MIE is asserted. 0 No interrupt flag is asserted or no interrupt enable is set 1 At least one of the other interrupt flags in this register is asserted and the related interrupt bit is asserted, too
PRIF	Protocol Interrupt Flag. This flag is set if at least one of the individual protocol interrupt flags in the Protocol Interrupt Flag Register 0 (PIFR0) and Protocol Interrupt Flag Register 1 (PIFR1) is asserted and the related interrupt enable flag is asserted, too. The FlexRay block generates the combined protocol interrupt request if the PRIE flag is asserted. 0 All individual protocol interrupt flags are equal to 0 or no interrupt enable bit is set. 1 At least one of the individual protocol interrupt flags and the related interrupt enable is equal to 1.
CHIF	CHI Interrupt Flag. This flag is set if at least one of the individual CHI error flags in the CHI Error Flag Register (CHIERFR) is asserted and the chi error interrupt enable GIFER.CHIE is asserted. The FlexRay block generates the combined CHI error interrupt if the CHIE flag is asserted, too. 0 All CHI error flags are equal to 0 or the chi error interrupt is disabled 1 At least one CHI error flag is asserted and chi error interrupt is enabled
WUPIF	Wakeup Interrupt Flag. This flag is set when the FlexRay block has received a wakeup symbol on the FlexRay bus. The application can determine on which channel the wakeup symbol was received by reading the related wakeup flags WUB and WUA in the Protocol Status Register 3 (PSR3) . The FlexRay block generates the wakeup interrupt request if the WUPIE flag is asserted. 0 No wakeup condition or interrupt disabled 1 Wakeup symbol received on FlexRay bus and interrupt enabled
FNEBIF	Receive FIFO channel B Not Empty Interrupt Flag. This flag is set when the receive FIFO for channel B is not empty. If the application writes 1 to this bit, the FlexRay block updates the FIFO status, increments or wraps the FIFO read index in the Receive FIFO B Read Index Register (RFBRIR) and clears the interrupt flag if the FIFO B is now empty. If the FIFO is still not empty, the FlexRay block sets this flag again. The FlexRay block generates the Receive FIFO B Not empty interrupt if the FNEBIE flag is asserted. 0 Receive FIFO B is empty or interrupt is disabled 1 Receive FIFO B is not empty and interrupt enabled

Table 30-17. GIFER Field Descriptions (Sheet 2 of 3)

Field	Description
FNEAIF	Receive FIFO channel A Not Empty Interrupt Flag. This flag is set when the receive FIFO for channel A is not empty. If the application writes 1 to this bit, the FlexRay block updates the FIFO status, increments or wraps the FIFO read index in the Receive FIFO A Read Index Register (RFARIR) and clears the interrupt flag if the FIFO A is now empty. If the FIFO is still not empty, the FlexRay block sets this flag again. The FlexRay block generates the Receive FIFO A Not empty interrupt if the FNEAIE flag is asserted. 0 Receive FIFO A is empty or interrupt is disabled 1 Receive FIFO A is not empty and interrupt enabled
RBIF	Receive Message Buffer Interrupt Flag. This flag is set if for at least one of the individual receive message buffers (MBCCSn.MTD = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding Message Buffer Configuration, Control, Status Registers (MBCCSRn) are asserted. The application can not clear this RBIF flag directly. This flag is cleared by the FlexRay block when all of the interrupt flags MBIF of the individual receive message buffers are cleared by the application or if the application has cleared the interrupt enables bit MBIE. 0 None of the individual receive message buffers has the MBIF and MBIE flag asserted. 1 At least one individual receive message buffer has the MBIF and MBIE flag asserted.
TBIF	Transmit Buffer Interrupt Flag. This flag is set if for at least one of the individual single or double transmit message buffers (MBCCSn.MTD = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding Message Buffer Configuration, Control, Status Registers (MBCCSRn) are equal to 1. The application can not clear this TBIF flag directly. This flag is cleared by the FlexRay block when either all of the individual interrupt flags MBIF of the individual transmit message buffers are cleared by the application or the host has cleared the interrupt enables bit MBIE. 0 None of the individual transmit message buffers has the MBIF and MBIE flag asserted. 1 At least one individual transmit message buffer has the MBIF and MBIE flag asserted.
MIE	Module Interrupt Enable. This flag controls if the module interrupt line is asserted when the MIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
PRIE	Protocol Interrupt Enable. This flag controls if the protocol interrupt line is asserted when the PRIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
CHIE	CHI Interrupt Enable. This flag controls if the CHI interrupt line is asserted when the CHIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
WUPIE	Wakeup Interrupt Enable. This flag controls if the wakeup interrupt line is asserted when the WUPIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
FNEBIE	Receive FIFO Channel B Not Empty Interrupt Enable. This flag controls if the receive FIFO B interrupt line is asserted when the FNEBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
FNEAIE	Receive FIFO Channel A Not Empty Interrupt Enable. This flag controls if the receive FIFO A interrupt line is asserted when the FNEAIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

Table 30-17. GIFER Field Descriptions (Sheet 3 of 3)

Field	Description
RBIE	Receive Buffer Interrupt Enable. This flag controls if the receive buffer interrupt line is asserted when the RBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
TBIE	Transmit Interrupt Enable. This flag controls if the transmit buffer interrupt line is asserted when the TBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

30.5.2.11 Protocol Interrupt Flag Register 0 (PIFR0)

Base + 0x0018

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FATL_IF	INTL_IF	ILCF_IF	CSA_IF	MRC_IF	MOC_IF	CCL_IF	MXS_IF	MTX_IF	LTXB_IF	LTXA_IF	TBVB_IF	TBVA_IF	TI2_IF	TI1_IF	CYS_IF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-11. Protocol Interrupt Flag Register 0 (PIFR0)

The register holds one set of the protocol-related individual interrupt flags.

Table 30-18. PIFR0 Field Descriptions (Sheet 1 of 3)

Field	Description
FATL_IF	Fatal Protocol Error Interrupt Flag. This flag is set when the protocol engine has detected a fatal protocol error. In this case, the protocol engine goes into the <i>POC:halt</i> state immediately. The fatal protocol errors are: 1) <i>pLatestTx</i> violation, as described in the MAC process of the FlexRay protocol 2) transmission across slot boundary violation, as described in the FSP process of the FlexRay protocol 0 No such event. 1 Fatal protocol error detected.
INTL_IF	Internal Protocol Error Interrupt Flag. This flag is set when the protocol engine has detected an internal protocol error. In this case, the protocol engine goes into the <i>POC:halt</i> state immediately. An internal protocol error occurs when the protocol engine has not finished a calculation and a new calculation is requested. This can be caused by a hardware error. 0 No such event. 1 Internal protocol error detected.
ILCF_IF	Illegal Protocol Configuration Interrupt Flag. This flag is set when the protocol engine has detected an illegal protocol configuration parameter setting. In this case, the protocol engine goes into the <i>POC:halt</i> state immediately. The protocol engine checks the <i>listen_timeout</i> value programmed into the Protocol Configuration Register 14 (PCR14) and Protocol Configuration Register 15 (PCR15) when the CONFIG_COMPLETE command was sent by the application via the Protocol Operation Control Register (POCR) . If the value of <i>listen_timeout</i> is equal to zero, the protocol configuration setting is considered as illegal. 0 No such event. 1 Illegal protocol configuration detected.

Table 30-18. PIFR0 Field Descriptions (Sheet 2 of 3)

Field	Description
CSA_IF	Cold Start Abort Interrupt Flag. This flag is set when the configured number of allowed cold start attempts is reached and none of these attempts was successful. The number of allowed cold start attempts is configured by the <code>coldstart_attempts</code> field in the Protocol Configuration Register 3 (PCR3) . 0 No such event. 1 Cold start aborted and no more coldstart attempts allowed.
MRC_IF	Missing Rate Correction Interrupt Flag. This flag is set when an insufficient number of measurements is available for rate correction at the end of the communication cycle. 0 No such event 1 Insufficient number of measurements for rate correction detected
MOC_IF	Missing Offset Correction Interrupt Flag. This flag is set when an insufficient number of measurements is available for offset correction. This is related to the <code>MISSING_TERM</code> event in the CSP process for offset correction in the FlexRay protocol. 0 No such event. 1 Insufficient number of measurements for offset correction detected.
CCL_IF	Clock Correction Limit Reached Interrupt Flag. This flag is set when the internal calculated offset or rate calculation values have reached or exceeded its configured thresholds as given by the <code>offset_coorection_out</code> field in the Protocol Configuration Register 9 (PCR9) and the <code>rate_correction_out</code> field in the Protocol Configuration Register 14 (PCR14) . 0 No such event. 1 Offset or rate correction limit reached.
MXS_IF	Max Sync Frames Detected Interrupt Flag. This flag is set when the number of synchronization frames detected in the current communication cycle exceeds the value of the <code>node_sync_max</code> field in the Protocol Configuration Register 30 (PCR30) . 0 No such event. 1 More than <code>node_sync_max</code> sync frames detected. Note: Only synchronization frames that have passed the synchronization frame acceptance and rejection filters are taken into account.
MTX_IF	Media Access Test Symbol Received Interrupt Flag. This flag is set when the MTS symbol was received on channel A or channel B. 0 No such event. 1 MTS symbol received.
LTXB_IF	pLatestTx Violation on Channel B Interrupt Flag. This flag is set when the frame transmission on channel B in the dynamic segment exceeds the dynamic segment boundary. This is related to the pLatestTx violation, as described in the MAC process of the FlexRay protocol. 0 No such event. 1 pLatestTx violation occurred on channel B.
LTXA_IF	pLatestTx Violation on Channel A Interrupt Flag. This flag is set when the frame transmission on channel A in the dynamic segment exceeds the dynamic segment boundary. This is related to the pLatestTx violation as described in the MAC process of the FlexRay protocol. 0 No such event. 1 pLatestTx violation occurred on channel A.
TBVB_IF	Transmission Across Boundary on Channel B Interrupt Flag. This flag is set when the frame transmission on channel B crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel B.

Table 30-18. PIFR0 Field Descriptions (Sheet 3 of 3)

Field	Description
TBVA_IF	Transmission across boundary on channel A Interrupt Flag. This flag is set when the frame transmission on channel A crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel A.
T12_IF	Timer 2 Expired Interrupt Flag. This flag is set whenever timer 2 expires. 0 No such event. 1 Timer 2 has reached its time limit.
T11_IF	Timer 1 Expired Interrupt Flag. This flag is set whenever timer 1 expires. 0 No such event 1 Timer 1 has reached its time limit
CYS_IF	Cycle Start Interrupt Flag. This flag is set when a communication cycle starts. 0 No such event 1 Communication cycle started.

30.5.2.12 Protocol Interrupt Flag Register 1 (PIFR1)

Base + 0x001A

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IF	IPC_IF	PECF_IF	PSC_IF	SSI3_IF	SSI2_IF	SSI1_IF	SSI0_IF	0	0	EVT_IF	ODT_IF	0	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-12. Protocol Interrupt Flag Register 1 (PIFR1)

The register holds one set of the protocol-related individual interrupt flags.

Table 30-19. PIFR1 Field Descriptions (Sheet 1 of 2)

Field	Description
EMC_IF	Error Mode Changed Interrupt Flag. This flag is set when the value of the ERRMODE bit field in the Protocol Status Register 0 (PSR0) is changed by the FlexRay block. 0 No such event. 1 ERRMODE field changed.
IPC_IF	Illegal Protocol Control Command Interrupt Flag. This flag is set when the PE tries to execute a protocol control command, which was issued via the POCCMD field of the Protocol Operation Control Register (POCR) , and detects that this protocol control command is not allowed in the current protocol state. In this case the command is not executed. For more details, see Section 30.7.4, "Protocol Control Command Execution" . 0 No such event. 1 Illegal protocol control command detected.
PECF_IF	Protocol Engine Communication Failure Interrupt Flag. This flag is set if the FlexRay block has detected a communication failure between the protocol engine and the controller host interface 0 No such event. 1 Protocol Engine Communication Failure detected.
PSC_IF	Protocol State Changed Interrupt Flag. This flag is set when the protocol state in the PROTSTATE field in the Protocol Status Register 0 (PSR0) has changed. 0 No such event. 1 Protocol state changed.

Table 30-19. PIFR1 Field Descriptions (Sheet 2 of 2)

Field	Description
SSI[3:0]_IF	Slot Status Counter Incremented Interrupt Flag. Each of these flags is set when the SLOTSTATUSCNT field in the corresponding Slot Status Counter Registers (SSCR0–SSCR3) is incremented. 0 No such event. 1 The corresponding slot status counter has incremented.
EVT_IF	Even Cycle Table Written Interrupt Flag. This flag is set if the FlexRay block has written the sync frame measurement / ID tables into the FlexRay Memory for the even cycle. 0 No such event. 1 Sync frame measurement table written
ODT_IF	Odd Cycle Table Written Interrupt Flag. This flag is set if the FlexRay block has written the sync frame measurement / ID tables into the FlexRay Memory for the odd cycle. 0 No such event. 1 Sync frame measurement table written

30.5.2.13 Protocol Interrupt Enable Register 0 (PIER0)

Base + 0x001C

Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FATL	INTL	ILCF	CSA	MRC	MOC	CCL	MXS	MTX	LTXB	LTXA	TBVB	TBVA	TI2	TI1	CYS
W	_IE	_IE	_IE	_IE	_IE	_IE	_IE	_IE	_IE	_IE	_IE	_IE	_IE	_IE	_IE	_IE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-13. Protocol Interrupt Enable Register 0 (PIER0)

This register defines whether or not the individual interrupt flags defined in the [Protocol Interrupt Flag Register 0 \(PIFR0\)](#) can generate a protocol interrupt request.

Table 30-20. PIER0 Field Descriptions

Field	Description
FATL_IE	Fatal Protocol Error Interrupt Enable. This bit controls FATL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
INTL_IE	Internal Protocol Error Interrupt Enable. This bit controls INTL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
ILCF_IE	Illegal Protocol Configuration Interrupt Enable. This bit controls ILCF_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
CSA_IE	Cold Start Abort Interrupt Enable. This bit controls CSA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MRC_IE	Missing Rate Correction Interrupt Enable. This bit controls MRC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MOC_IE	Missing Offset Correction Interrupt Enable. This bit controls MOC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

Table 30-20. PIER0 Field Descriptions (continued)

Field	Description
CCL_IE	Clock Correction Limit Reached Interrupt Enable. This bit controls CCL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MXS_IE	Max Sync Frames Detected Interrupt Enable. This bit controls MXS_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MTX_IE	Media Access Test Symbol Received Interrupt Enable. This bit controls MTX_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
LTXB_IE	<i>pLatestTx</i> Violation on Channel B Interrupt Enable. This bit controls LTXB_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
LTXA_IE	<i>pLatestTx</i> Violation on Channel A Interrupt Enable. This bit controls LTXA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TBVB_IE	Transmission across boundary on channel B Interrupt Enable. This bit controls TBVB_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TBVA_IE	Transmission across boundary on channel A Interrupt Enable. This bit controls TBVA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
T12_IE	Timer 2 Expired Interrupt Enable. This bit controls T11_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
T11_IE	Timer 1 Expired Interrupt Enable. This bit controls T11_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
CYS_IE	Cycle Start Interrupt Enable. This bit controls CYC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

30.5.2.14 Protocol Interrupt Enable Register 1 (PIER1)

Base + 0x001E

Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC	IPC	PECF	PSC	SSI3	SSI2	SSI1	SSI0	0	0	EVT	ODT	0	0	0	0
W	_IE	_IE	_IE	_IE	_IE	_IE	_IE	_IE			_IE	_IE				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-14. Protocol Interrupt Enable Register 1 (PIER1)

This register defines whether or not the individual interrupt flags defined in [Protocol Interrupt Flag Register 1 \(PIFR1\)](#) can generate a protocol interrupt request.

Table 30-21. PIER1 Field Descriptions

Field	Description
EMC_IE	Error Mode Changed Interrupt Enable. This bit controls EMC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
IPC_IE	Illegal Protocol Control Command Interrupt Enable. This bit controls IPC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
PECF_IE	Protocol Engine Communication Failure Interrupt Enable. This bit controls PECF_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
PSC_IE	Protocol State Changed Interrupt Enable. This bit controls PSC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
SSI[3:0]_IE	Slot Status Counter Incremented Interrupt Enable. This bit controls SSI[3:0]_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
EVT_IE	Even Cycle Table Written Interrupt Enable. This bit controls EVT_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
ODT_IE	Odd Cycle Table Written Interrupt Enable. This bit controls ODT_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

30.5.2.15 CHI Error Flag Register (CHIERFR)

Base + 0x0020

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRLB _EF	FRLA _EF	PCMI _EF	FOVB _EF	FOVA _EF	MBS _EF	MBU _EF	LCK _EF	DBL _EF	SBCF _EF	FID _EF	DPL _EF	SPL _EF	NML _EF	NMF _EF	ILSA _EF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-15. CHI Error Flag Register (CHIERFR)

This register holds the CHI related error flags. The interrupt generation for each of these error flags is controlled by the CHI interrupt enable bit CHIE in the [Global Interrupt Flag and Enable Register \(GIFER\)](#).

Table 30-22. CHIERR Field Descriptions (Sheet 1 of 2)

Field	Description
FRLB_EF	Frame Lost Channel B Error Flag. This flag is set if a complete frame was received on channel B but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such event 1 Frame lost on channel B detected
FRLA_EF	Frame Lost Channel A Error Flag. This flag is set if a complete frame was received on channel A but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such error 1 Frame lost on channel A detected
PCMI_EF	Protocol Command Ignored Error Flag. This flag is set if the application has issued a POC command by writing to the POCCMD field in the Protocol Operation Control Register (POCR) while the BSY flag is equal to 1. In this case the command is ignored by the FlexRay block and is lost. 0 No such error 1 POC command ignored
FOVB_EF	Receive FIFO Overrun Channel B Error Flag. This flag is set when an overrun of the Receive FIFO for channel B occurred. This error occurs if a semantically valid frame was received on channel B and matches the all criteria to be appended to the FIFO for channel B but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 Receive FIFO overrun on channel B has been detected
FOVA_EF	Receive FIFO Overrun Channel A Error Flag. This flag is set when an overrun of the Receive FIFO for channel A occurred. This error occurs if a semantically valid frame was received on channel A and matches the all criteria to be appended to the FIFO for channel A but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 Receive FIFO overrun on channel B has been detected
MSB_EF	Message Buffer Search Error Flag. This flag is set if the message buffer search engine is still running while the next search cycle must be started due to the FlexRay protocol timing. In this case, not all message buffers are considered while searching. 0 No such event 1 Search engine active while search start appears
MBU_EF	Message Buffer Utilization Error Flag. This flag is asserted if the application writes to a message buffer control field that is beyond the number of utilized message buffers programmed in the Message Buffer Segment Size and Utilization Register (MBSSUTR) . If the application writes to a MBCCSRn register with n > LAST_MB_UTIL, the FlexRay block ignores the write attempt and asserts the message buffer utilization error flag MBU_EF in the CHI Error Flag Register (CHIERR) . 0 No such event 1 Non-utilized message buffer enabled
LCK_EF	Lock Error Flag. This flag is set if the application tries to lock a message buffer that is already locked by the FlexRay block due to internal operations. In that case, the FlexRay block does not grant the lock to the application. The application must issue the lock request again. 0 No such error 1 Lock error detected
DBL_EF	Double Transmit Message Buffer Lock Error Flag. This flag is set if the application tries to lock the transmit side of a double transmit message buffer. In this case, the FlexRay block does not grant the lock to the transmit side of a double transmit message buffer. 0 No such event 1 Double transmit buffer lock error occurred

Table 30-22. CHIERFR Field Descriptions (Sheet 2 of 2)

Field	Description
SBCF_EF	System Bus Communication Failure Error Flag. This flag is set if the FlexRay block was not able to transmit or receive data via the system bus in time. In the case of writing, data is lost; in the case of reading, the transmission onto the FlexRay bus is stopped for the current slot and resumed in the next slot. 0 No such event 1 System bus communication failure occurred
FID_EF	Frame ID Error Flag. This flag is set if the frame ID stored in the message buffer header area differs from the frame ID stored in the message buffer control register. 0 No such error occurred 1 Frame ID error occurred
DPL_EF	Dynamic Payload Length Error Flag. This flag is set if the payload length written into the message buffer header field of a single or double transmit message buffer assigned to the dynamic segment is greater than the maximum payload length for the dynamic segment as it is configured in the corresponding protocol configuration register field max_payload_length_dynamic in the Protocol Configuration Register 24 (PCR24) . 0 No such error occurred 1 Dynamic payload length error occurred
SPL_EF	Static Payload Length Error Flag. This flag is set if the payload length written into the message buffer header field of a single or double transmit message buffer assigned to the static segment is different from the payload length for the static segment as it is configured in the corresponding protocol configuration register field payload_length_static in the Protocol Configuration Register 19 (PCR19) . 0 No such error occurred 1 Static payload length error occurred
NML_EF	Network Management Length Error Flag. This flag is set if the payload length written into the header structure of a receive message buffer assigned to the static segment is less than the configured length of the Network Management Vector as configured in the Network Management Vector Length Register (NMVLR) . In this case the received part of the Network Management Vector will be used to update the Network Management Vector. 0 No such error occurred 1 Network management length error occurred
NMF_EF	Network Management Frame Error Flag. This flag is set if a received message in the static segment with a Preamble Indicator flag PP asserted has its Null Frame indicator flag NF asserted as well. In this case, the Global Network Management Registers (see Network Management Vector Registers (NMVR0–NMVR5)) are not updated. 0 No such error occurred 1 Network management frame error occurred
ILSA_EF	Illegal System Memory Access Error Flag. This flag is set if the external system memory subsystem has detected and indicated an illegal system memory access from the FlexRay block. The exact meaning of an illegal system memory access is defined by the current implementation of the memory subsystem. 0 No such event. 1 Illegal system memory access occurred.

30.5.2.16 Message Buffer Interrupt Vector Register (MBIVEC)

Base + 0x0022

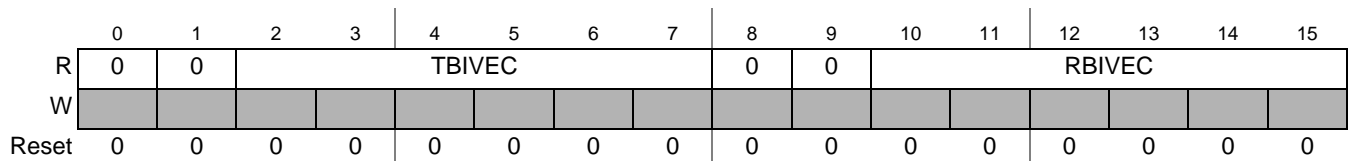


Figure 30-16. Message Buffer Interrupt Vector Register (MBIVEC)

This register indicates the lowest numbered receive message buffer and the lowest numbered transmit message buffer that have their interrupt status flag MBIF and interrupt enable MBIE bits asserted. This means that message buffers with lower message buffer numbers have higher priority.

Table 30-23. MBIVEC Field Descriptions

Field	Description
TBIVEC	Transmit Buffer Interrupt Vector. This field provides the number of the lowest numbered enabled transmit message buffer that has its interrupt status flag MBIF and its interrupt enable bit MBIE set. If there is no transmit message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.
RBIVEC	Receive Buffer Interrupt Vector. This field provides the message buffer number of the lowest numbered receive message buffer which has its interrupt flag MBIF and its interrupt enable bit MBIE asserted. If there is no receive message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.

30.5.2.17 Channel A Status Error Counter Register (CASERCR)

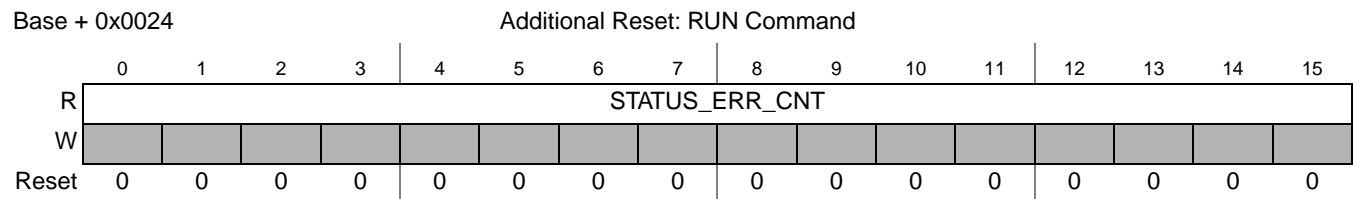


Figure 30-17. Channel A Status Error Counter Register (CASERCR)

This register provides the channel status error counter for channel A. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, and *vSS!TxConflict*. The FlexRay block increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring, see [Section 30.6.18, “Slot Status Monitoring”](#).

Table 30-24. CASERCR Field Descriptions

Field	Description
STATUS_ERR_CNT	Channel Status Error Counter. This field provides the current value channel status error counter. The counter value is updated within the first macrotick of the following slot or segment.

30.5.2.18 Channel B Status Error Counter Register (CBSERCR)

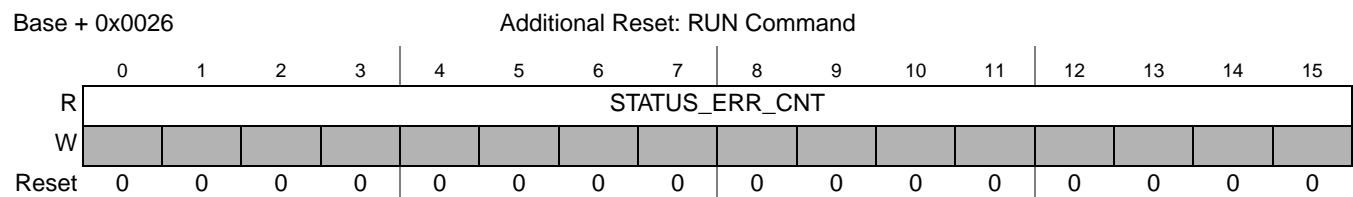


Figure 30-18. Channel B Status Error Counter Register (CBSERCR)

This register provides the channel status error counter for channel B. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, and *vSS!TxConflict*. The FlexRay block increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring see [Section 30.6.18, “Slot Status Monitoring”](#).

Table 30-25. CBSERCR Field Descriptions

Field	Description
STATUS_ERR_CNT	Channel Status Error Counter. This field provides the current channel status error count. The counter value is updated within the first macrotick of the following slot or segment.

30.5.2.19 Protocol Status Register 0 (PSR0)

Base + 0x0028

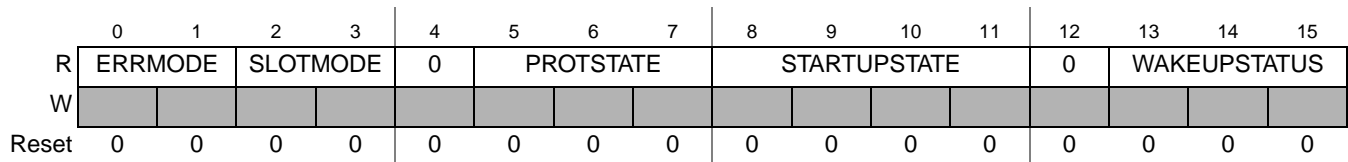


Figure 30-19. Protocol Status Register 0 (PSR0)

This register provides information about the current protocol status.

Table 30-26. PSR0 Field Descriptions (Sheet 1 of 2)

Field	Description
ERRMODE	Error Mode. Protocol related variable: <i>vPOC!ErrorMode</i> . This field indicates the error mode of the protocol. 00 ACTIVE 01 PASSIVE 10 COMM_HALT 11 reserved
SLOTMODE	Slot Mode. Protocol related variable: <i>vPOC!SlotMode</i> . This field indicates the slot mode of the protocol. 00 SINGLE 01 ALL_PENDING 10 ALL 11 reserved
PROTSTATE	Protocol State. Protocol related variable: <i>vPOC!State</i> . This field indicates the state of the protocol. 000 <i>POC:default config</i> 001 <i>POC:config</i> 010 <i>POC:wakeup</i> 011 <i>POC:ready</i> 100 <i>POC:normal passive</i> 101 <i>POC:normal active</i> 110 <i>POC:halt</i> 111 <i>POC:startup</i>

Table 30-26. PSR0 Field Descriptions (Sheet 2 of 2)

Field	Description
STARTUP STATE	Startup State. Protocol related variable: <i>vPOC!StartupState</i> . This field indicates the current sub-state of the startup procedure. 0000 reserved 0001 reserved 0010 <i>POC:coldstart collision resolution</i> 0011 <i>POC:coldstart listen</i> 0100 <i>POC:integration consistency check</i> 0101 <i>POC:integrationi listen</i> 0110 reserved 0111 <i>POC:initialize schedule</i> 1000 reserved 1001 reserved 1010 <i>POC:coldstart consistency check</i> 1011 reserved 1100 reserved 1101 <i>POC:integration coldstart check</i> 1110 <i>POC:coldstart gap</i> 1111 <i>POC:coldstart join</i>
WAKEUP STATUS	Wakeup Status. Protocol related variable: <i>vPOC!WakeupStatus</i> . This field provides the outcome of the execution of the wakeup mechanism. 000 UNDEFINED 001 RECEIVED_HEADER 010 RECEIVED_WUP 011 COLLISION_HEADER 100 COLLISION_WUP 101 COLLISION_UNKNOWN 110 TRANSMITTED 111 reserved

30.5.2.20 Protocol Status Register 1 (PSR1)

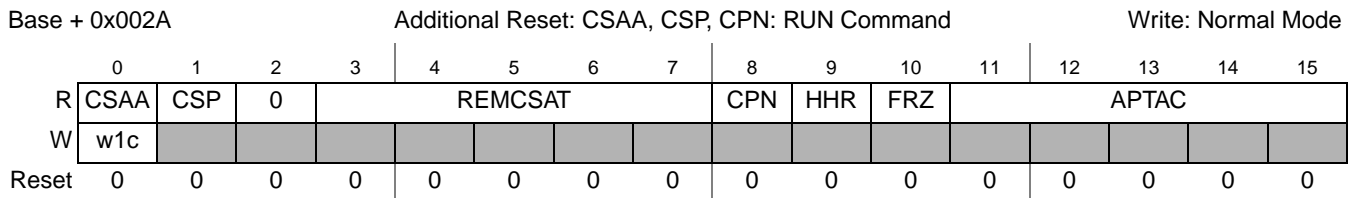


Figure 30-20. Protocol Status Register 1 (PSR1)

Table 30-27. PSR1 Field Descriptions

Field	Description
CSAA	Cold Start Attempt Aborted Flag. Protocol related event: 'set coldstart abort indicator in CHI' This flag is set when the FlexRay block has aborted a cold start attempt. 0 No such event 1 Cold start attempt aborted
CSP	Leading Cold Start Path. This status bit is set when the FlexRay block has reached the <i>POC:normal active</i> state via the leading cold start path. This indicates that this node has started the network 0 No such event 1 <i>POC:normal active</i> reached from <i>POC:startup</i> state via leading cold start path
REMCSAT	Remaining Cold Start Attempts. Protocol related variable: <i>vRemainingColdstartAttempts</i> This field provides the number of remaining cold start attempts that the FlexRay block will execute.
CPN	Leading Cold Start Path Noise. Protocol related variable: <i>vPOC!ColdstartNoise</i> This status bit is set if the FlexRay block has reached the <i>POC:normal active</i> state via the leading cold start path under noise conditions. This indicates there was some activity on the FlexRay bus while the FlexRay block was starting up the cluster. 0 No such event 1 <i>POC:normal active</i> state was reached from <i>POC:startup</i> state via noisy leading cold start path
HHR	Host Halt Request Pending. Protocol related variable: <i>vPOC!CHIHaltRequest</i> This status bit is set when FlexRay block receives the HALT command from the application via the Protocol Operation Control Register (POCR) . The FlexRay block clears this status bit after a hard reset condition or when the protocol is in the <i>POC:default config</i> state. 0 No such event 1 HALT command received
FRZ	Freeze Occurred. Protocol related variable: <i>vPOC!Freeze</i> This status bit is set when the FlexRay block has reached the <i>POC:halt</i> state due to the host FREEZE command or due to an internal error condition requiring immediate halt. The FlexRay block clears this status bit after a hard reset condition or when the protocol is in the <i>POC:default config</i> state. 0 No such event 1 Immediate halt due to FREEZE or internal error condition
APTAC	Allow Passive to Active Counter. Protocol related variable: <i>vPOC!vAllowPassivetoActive</i> This field provides the number of consecutive even/odd communication cycle pairs that have passed with valid rate and offset correction terms, but the protocol is still in the <i>POC:normal passive</i> state due to an application configured delay to enter <i>POC:normal active</i> state. This delay is defined by the allow_passive_to_active field in the Protocol Configuration Register 12 (PCR12) .

30.5.2.21 Protocol Status Register 2 (PSR2)

Base + 0x002C				Additional Reset: RUN Command												
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NBVB	NSEB	STCB	SBVB	SSEB	MTB	NBVA	NSEA	STCA	SBVA	SSEA	MTA	CLKCORRFALCNT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-21. Protocol Status Register 2 (PSR2)

This register provides a snapshot of status information about the Network Idle Time NIT, the Symbol Window and the clock synchronization. The NIT related status bits NBVB, NSEB, NBVA, and NSEA are updated by the FlexRay block after the end of the NIT and before the end of the first slot of the next

communication cycle. The Symbol Window related status bits STCB, SBVB, SSEB, MTB, STCA, SBVA, SSEB, and MTA are updated by the FlexRay block after the end of the symbol window and before the end of the current communication cycle. If no symbol window is configured, the symbol window related status bits remain in their reset state. The clock synchronization related CLKCORRFAILCNT is updated by the FlexRay block after the end of the static segment and before the end of the current communication cycle.

Table 30-28. PSR2 Field Descriptions (Sheet 1 of 2)

Field	Description
NBVB	NIT Boundary Violation on Channel B. Protocol related variable: vSS!BViolation for NIT on channel B This status bit is set when there was some media activity on the FlexRay bus channel B at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEB	NIT Syntax Error on Channel B. Protocol related variable: vSS!SyntaxError for NIT on channel B This status bit is set when a syntax error was detected during NIT on channel B. 0 No such event 1 Syntax error detected
STCB	Symbol Window Transmit Conflict on Channel B. Protocol related variable: vSS!TxConflict for symbol window on channel B This status bit is set if there was a transmission conflict during the symbol window on channel B. 0 No such event 1 Transmission conflict detected
SBVB	Symbol Window Boundary Violation on Channel B. Protocol related variable: vSS!BViolation for symbol window on channel B This status bit is set if there was some media activity on the FlexRay bus channel B at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEB	Symbol Window Syntax Error on Channel B. Protocol related variable: vSS!SyntaxError for symbol window on channel B This status bit is set when a syntax error was detected during the symbol window on channel B. 0 No such event 1 Syntax error detected
MTB	Media Access Test Symbol MTS Received on Channel B. protocol related variable: vSS!ValidMTS for Symbol Window on channel B This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel B. 0 No such event 1 MTS symbol received
NBVA	NIT Boundary Violation on Channel A. Protocol related variable: vSS!BViolation for NIT on channel A This status bit is set when there was some media activity on the FlexRay bus channel A at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEA	NIT Syntax Error on Channel A. Protocol related variable: vSS!SyntaxError for NIT on channel A This status bit is set when a syntax error was detected during NIT on channel A. 0 No such event 1 Syntax error detected
STCA	Symbol Window Transmit Conflict on Channel A. Protocol related variable: vSS!TxConflict for symbol window on channel A This status bit is set if there was a transmission conflicts during the symbol window on channel A. 0 No such event 1 Transmission conflict detected

Table 30-28. PSR2 Field Descriptions (Sheet 2 of 2)

Field	Description
SBVA	Symbol Window Boundary Violation on Channel A. Protocol related variable: <i>vSS!BViolation</i> for symbol window on channel A This status bit is set if there was some media activity on the FlexRay bus channel A at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEA	Symbol Window Syntax Error on Channel A. Protocol related variable: <i>vSS!SyntaxError</i> for symbol window on channel A This status bit is set when a syntax error was detected during the symbol window on channel A. 0 No such event 1 Syntax error detected
MTA	Media Access Test Symbol MTS Received on Channel A . Protocol related variable: <i>vSS!ValidMTS</i> for symbol window on channel A This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel A. 1 MTS symbol received 0 No such event
CLKCORR-FAILCNT	Clock Correction Failed Counter. Protocol related variable: <i>vClockCorrectionFailed</i> This field provides the number of consecutive even/odd communication cycle pairs that have passed without clock synchronization having performed an offset or a rate correction due to lack of synchronization frames. It is not incremented when it has reached the configured value of either <i>max_without_clock_correction_fatal</i> or <i>max_without_clock_correction_passive</i> as defined in the Protocol Configuration Register 8 (PCR8) . The FlexRay block resets this counter on a hard reset condition, when the protocol enters the <i>POC:normal active</i> state, or when both the rate and offset correction terms have been calculated successfully.

30.5.2.22 Protocol Status Register 3 (PSR3)

Base + 0x002E				Additional Reset: RUN Command								Write: Normal Mode				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WUB	ABVB	AACB	ACEB	ASEB	AVFB	0	0	WUA	ABVA	AACA	ACEA	ASEA	AVFA
W			w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-22. Protocol Status Register 3 (PSR3)

This register provides aggregated channel status information as an accrued status of channel activity for all communication slots, regardless of whether they are assigned for transmission or subscribed for reception. It provides accrued information for the symbol window, the NIT, and the wakeup status.

Table 30-29. PSR3 Field Descriptions (Sheet 1 of 2)

Field	Description
WUB	Wakeup Symbol Received on Channel B. This flag is set when a wakeup symbol was received on channel B. 0 No wakeup symbol received 1 Wakeup symbol received
ABVB	Aggregated Boundary Violation on Channel B. This flag is set when a boundary violation has been detected on channel B. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected

Table 30-29. PSR3 Field Descriptions (Sheet 2 of 2)

Field	Description
AACB	Aggregated Additional Communication on Channel B. This flag is set when at least one valid frame was received on channel B in a slot that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEB	Aggregated Content Error on Channel B. This flag is set when a content error has been detected on channel B. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEB	Aggregated Syntax Error on Channel B. This flag is set when a syntax error has been detected on channel B. Syntax errors are detected in the communication slots, the symbol window and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFB	Aggregated Valid Frame on Channel B. This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel B. 1 At least one syntactically valid frame received 0 No syntactically valid frames received
WUA	Wakeup Symbol Received on Channel A. This flag is set when a wakeup symbol was received on channel A. 0 No wakeup symbol received 1 Wakeup symbol received
ABVA	Aggregated Boundary Violation on Channel A. This flag is set when a boundary violation has been detected on channel A. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACA	Aggregated Additional Communication on Channel A. This flag is set when a valid frame was received in a slot on channel A that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEA	Aggregated Content Error on Channel A. This flag is set when a content error has been detected on channel A. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEA	Aggregated Syntax Error on Channel A. This flag is set when a syntax error has been detected on channel A. Syntax errors are detected in the communication slots, the symbol window, and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFA	Aggregated Valid Frame on Channel A. This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel A. 0 No syntactically valid frames received 1 At least one syntactically valid frame received

30.5.2.23 Macrotick Counter Register (MTCTR)

Base + 0x0030

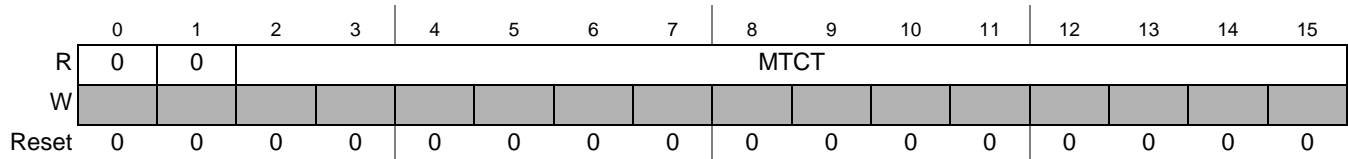


Figure 30-23. Macrotick Counter Register (MTCTR)

This register provides the macrotick count of the current communication cycle.

Table 30-30. MTCTR Field Descriptions

Field	Description
MTCT	Macrotick Counter. Protocol related variable: vMacrotick This field provides the macrotick count of the current communication cycle.

30.5.2.24 Cycle Counter Register (CYCTR)

Base + 0x0032

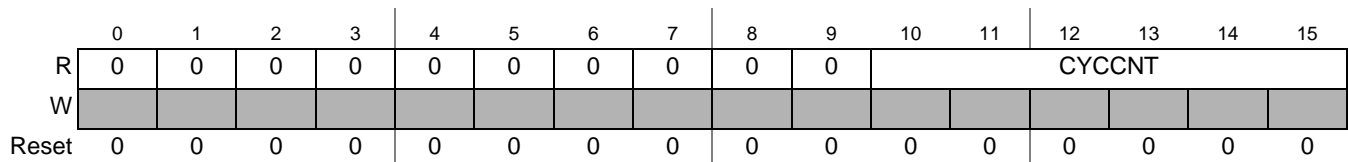


Figure 30-24. Cycle Counter Register (CYCTR)

This register provides the number of the current communication cycle.

Table 30-31. CYCTR Field Descriptions

Field	Description
CYCCNT	Cycle Counter. Protocol related variable: vCycleCounter This field provides the number of the current communication cycle. If the counter reaches the maximum value of 63, the counter wraps and starts from zero again.

30.5.2.25 Slot Counter Channel A Register (SLTCTAR)

Base + 0x0034

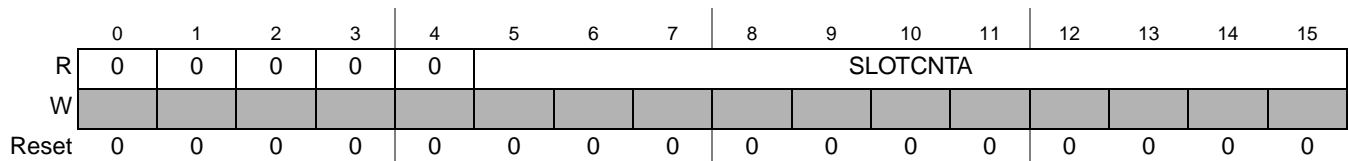


Figure 30-25. Slot Counter Channel A Register (SLTCTAR)

This register provides the number of the current slot in the current communication cycle for channel A.

Table 30-32. SLTCTAR Field Descriptions

Field	Description
SLOTCNTA	Slot Counter Value for Channel A. Protocol related variable: <i>vSlotCounter</i> for channel A This field provides the number of the current slot in the current communication cycle.

30.5.2.26 Slot Counter Channel B Register (SLTCTBR)

Base + 0x0036

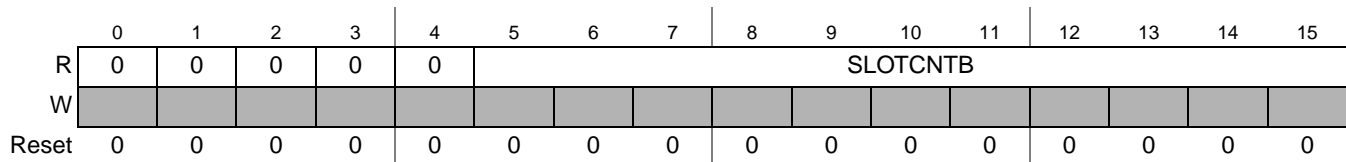


Figure 30-26. Slot Counter Channel B Register (SLTCTBR)

This register provides the number of the current slot in the current communication cycle for channel B.

Table 30-33. SLTCTBR Field Descriptions

Field	Description
SLOTCNTA	Slot Counter Value for Channel B. Protocol related variable: <i>vSlotCounter</i> for channel B This field provides the number of the current slot in the current communication cycle.

30.5.2.27 Rate Correction Value Register (RTCORVR)

Base + 0x0038

Additional Reset: RUN Command

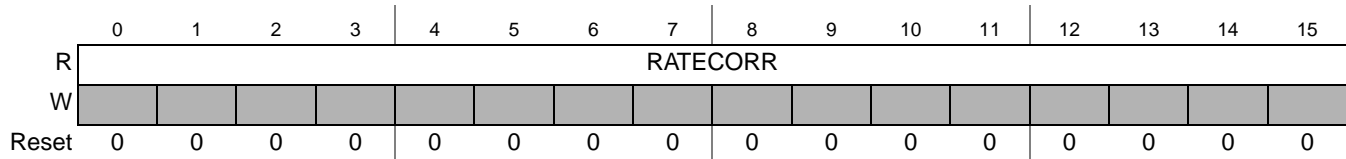


Figure 30-27. Rate Correction Value Register (RTCORVR)

This register provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The FlexRay block updates this register during the NIT of each odd numbered communication cycle.

Table 30-34. RTCORVR Field Descriptions

Field	Description
RATECORR	Rate Correction Value. Protocol related variable: <i>vRateCorrection</i> (before value limitation and external rate correction) This field provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external rate correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by <i>rate_correction_out</i> in the Protocol Configuration Register 13 (PCR13) , the clock correction reached limit interrupt flag <i>CCL_IF</i> is set in the Protocol Interrupt Flag Register 0 (PIFR0) . Note: If the FlexRay block was not able to calculate a new rate correction term due to a lack of synchronization frames, the RATECORR value is not updated.

30.5.2.28 Offset Correction Value Register (OFCORVR)

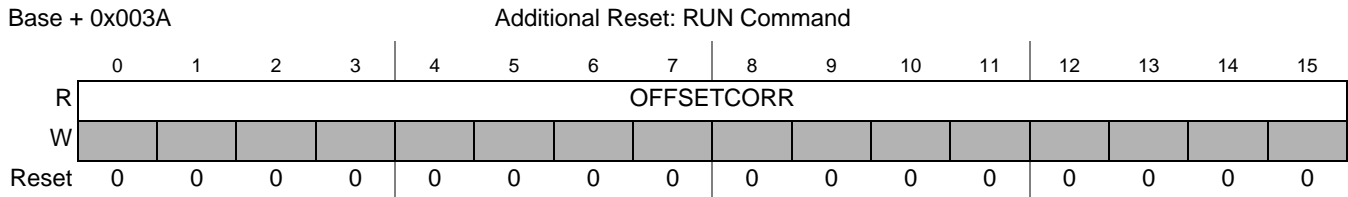


Figure 30-28. Offset Correction Value Register (OFCORVR)

This register provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The FlexRay block updates this register during the NIT.

Table 30-35. OFCORVR Field Descriptions

Field	Description
OFFSET-CORR	<p>Offset Correction Value. Protocol related variable: <i>vOffsetCorrection</i> (before value limitation and external offset correction)</p> <p>This field provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external offset correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by <code>offset_correction_out</code> field in the Protocol Configuration Register 29 (PCR29), the clock correction reached limit interrupt flag <code>CCL_IF</code> is set in the Protocol Interrupt Flag Register 0 (PIFR0).</p> <p>Note: If the FlexRay block was not able to calculate an new offset correction term due to a lack of synchronization frames, the OFFSETCORR value is not updated.</p>

30.5.2.29 Combined Interrupt Flag Register (CIFRR)

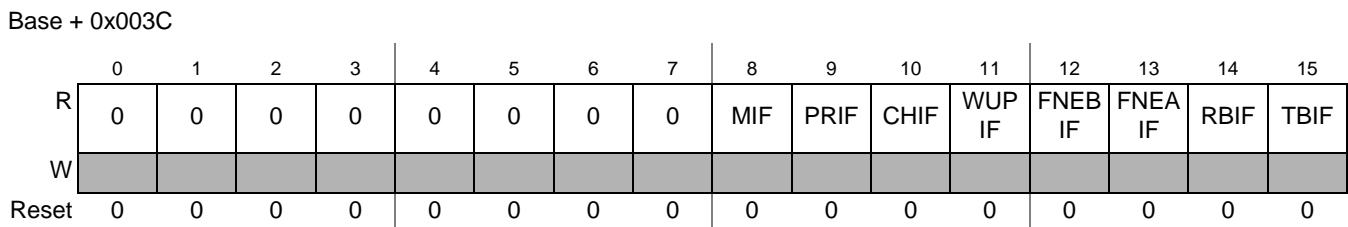


Figure 30-29. Combined Interrupt Flag Register (CIFRR)

This register provides five combined interrupt flags and a copy of three individual interrupt flags. The combined interrupt flags are the result of a binary OR of the values of other interrupt flags regardless of the state of the interrupt enable bits. The generation scheme for the combined interrupt flags is depicted in [Figure 30-144](#). The individual interrupt flags WUPIF, FNEBIF, and FNEAIF are copies of corresponding flags in the [Global Interrupt Flag and Enable Register \(GIFER\)](#) and are provided here to simplify the application interrupt flag check. To clear the individual interrupt flags, the application must use the [Global Interrupt Flag and Enable Register \(GIFER\)](#).

NOTE

The meanings of the combined status bits MIF, PRIF, CHIF, RBIF, and TBIF are different from those mentioned in the [Global Interrupt Flag and Enable Register \(GIFER\)](#).

Table 30-36. CIFRR Field Descriptions

Field	Description
MIF	Module Interrupt Flag. This flag is set if there is at least one interrupt source that has its interrupt flag asserted. 0 No interrupt source has its interrupt flag asserted 1 At least one interrupt source has its interrupt flag asserted
PRIF	Protocol Interrupt Flag. This flag is set if at least one of the individual protocol interrupt flags in the Protocol Interrupt Flag Register 0 (PIFR0) or Protocol Interrupt Flag Register 1 (PIFR1) is equal to 1. 0 All individual protocol interrupt flags are equal to 0 1 At least one of the individual protocol interrupt flags is equal to 1
CHIF	CHI Interrupt Flag. This flag is set if at least one of the individual CHI error flags in the CHI Error Flag Register (CHIERFR) is equal to 1. 0 All CHI error flags are equal to 0 1 At least one CHI error flag is equal to 1
WUPIF	Wakeup Interrupt Flag. Provides the same value as GIFER[WUPIF]
FNEBIF	Receive FIFO Channel B Not Empty Interrupt Flag. Provides the same value as GIFER[FNEBI]
FNEAIF	Receive FIFO Channel A Not Empty Interrupt Flag. Provides the same value as GIFER[FNEAIF]
RBIF	Receive Message Buffer Interrupt Flag. This flag is set if for at least one of the individual receive message buffers (MBCCSRn[MTD] = 0) the interrupt flag MBIF in the corresponding Message Buffer Configuration, Control, Status Registers (MBCCSRn) is equal to 1. 0 None of the individual receive message buffers has the MBIF flag asserted. 1 At least one individual receive message buffers has the MBIF flag asserted.
TBIF	Transmit Message Buffer Interrupt Flag. This flag is set if for at least one of the individual single or double transmit message buffers (MBCCSRn[MTD] = 1) the interrupt flag MBIF in the corresponding Message Buffer Configuration, Control, Status Registers (MBCCSRn) is equal to 1. 0 None of the individual transmit message buffers has the MBIF flag asserted. 1 At least one individual transmit message buffers has the MBIF flag asserted.

30.5.2.30 System Memory Access Time-Out Register (SYMATOR)

Base + 0x003E

Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	TIMEOUT				
W	[Shaded]											[Shaded]				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 30-30. System Memory Access Time-Out Register (SYMATOR)

Table 30-37. SYMATOR Field Descriptions

Field	Description
TIMEOUT	Time Out. This value defines the maximum number of wait states on the system memory bus interface. This value must never exceeded in order to ensure no data are lost even under internal worst case conditions. If the number of wait states is greater than the TIMEOUT value, but is less than twice the TIMEOUT value, and internal worst case conditions occur, than data might be lost. If data are lost, the System Bus Communication Failure Error Flag SBCF_EF is set in the CHI Error Flag Register (CHIERFR) . If the number of wait states is greater than twice the TIMEOUT value, data will be lost, and the System Bus Communication Failure Error Flag SBCF_EF is set in the CHI Error Flag Register (CHIERFR) .

30.5.2.31 Sync Frame Counter Register (SFCNTR)

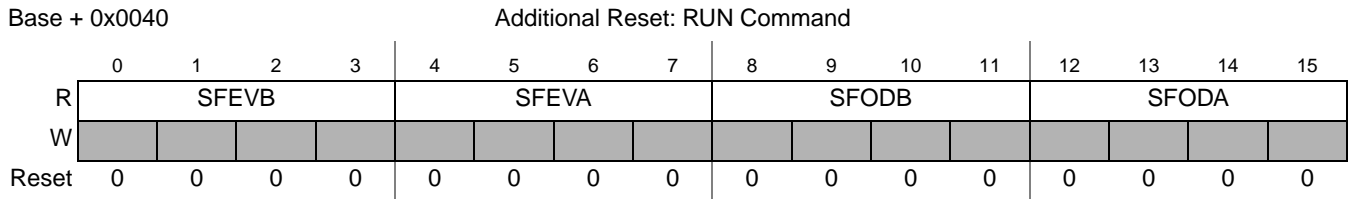


Figure 30-31. Sync Frame Counter Register (SFCNTR)

This register provides the number of synchronization frames that are used for clock synchronization in the last even and in the last odd numbered communication cycle. This register is updated after the start of the NIT and before 10 MT after offset correction start.

NOTE

If the application has locked the even synchronization table at the end of the static segment of an even communication cycle, the FlexRay block will not update the fields SFEVB and SFEVA.

If the application has locked the odd synchronization table at the end of the static segment of an odd communication cycle, the FlexRay block will not update the values SFODB and SFODA.

Table 30-38. SFCNTR Field Descriptions

Field	Description
SFEVB	Sync Frames Channel B, Even Cycle. Protocol related variable: size of (<i>vsSyncIdListB</i> for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFEVA	Sync Frames Channel A, Even Cycle. Protocol related variable: size of (<i>vsSyncIdListA</i> for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODB	Sync Frames Channel B, Odd Cycle. Protocol related variable: size of (<i>vsSyncIdListB</i> for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODA	Sync Frames Channel A, Odd Cycle. Protocol related variable: size of (<i>vsSyncIdListA</i> for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.

30.5.2.32 Sync Frame Table Offset Register (SFTOR)

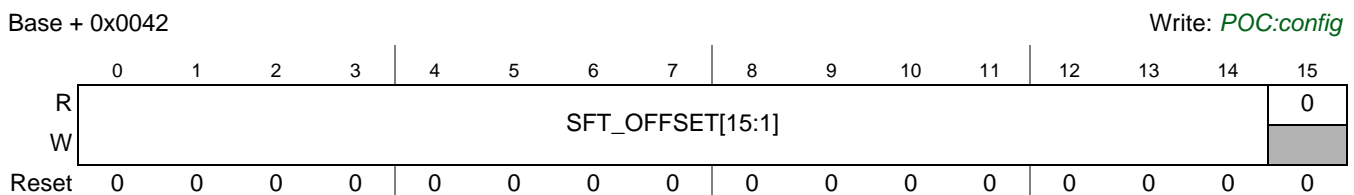


Figure 30-32. Sync Frame Table Offset Register (SFTOR)

This register defines the Flexray Memory related offset for sync frame tables. For more details, see [Section 30.6.12, “Sync Frame ID and Sync Frame Deviation Tables”](#).

Table 30-39. SFTOR Field Description

Field	Description
SFTOR	Sync Frame Table Offset. The offset of the Sync Frame Tables in the Flexray Memory. This offset is required to be 16-bit aligned. Thus STF_OFFSET[0] is always 0.

30.5.2.33 Sync Frame Table Configuration, Control, Status Register (SFTCCSR)

Base + 0x0044

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	CYCNUM				ELKS	OLKS	EVAL	OVAL	0	0	SDV	SID		
W	ELKT	OLKT											OPT	EN	EN	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-33. Sync Frame Table Configuration, Control, Status Register (SFTCCSR)

This register provides configuration, control, and status information related to the generation and access of the clock sync ID tables and clock sync measurement tables. For a detailed description, see [Section 30.6.12, “Sync Frame ID and Sync Frame Deviation Tables”](#).

Table 30-40. SFTCCSR Field Descriptions (Sheet 1 of 2)

Field	Description
ELKT	Even Cycle Tables Lock/Unlock Trigger. This trigger bit is used to lock and unlock the even cycle tables. 0 No effect 1 Triggers lock/unlock of the even cycle tables.
OLKT	Odd Cycle Tables Lock/Unlock Trigger. This trigger bit is used to lock and unlock the odd cycle tables. 0 No effect 1 Triggers lock/unlock of the odd cycle tables.
CYCNUM	Cycle Number. This field provides the number of the cycle in which the currently locked table was recorded. If none or both tables are locked, this value is related to the even cycle table.
ELKS	Even Cycle Tables Lock Status. This status bit indicates whether the application has locked the even cycle tables. 0 Application has not locked the even cycle tables. 1 Application has locked the even cycle tables.
OLKS	Odd Cycle Tables Lock Status. This status bit indicates whether the application has locked the odd cycle tables. 0 Application has not locked the odd cycle tables. 1 Application has locked the odd cycle tables.
EVAL	Even Cycle Tables Valid. This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the even cycle are valid. The FlexRay block clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).

Table 30-40. SFTCCSR Field Descriptions (Sheet 2 of 2)

Field	Description
OVAL	Odd Cycle Tables Valid. This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the odd cycle are valid. The FlexRay block clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).
OPT	One Pair Trigger. This trigger bit controls whether the FlexRay block writes continuously or only one pair of Sync Frame Tables into the FRM. If this trigger is set to 1 while SDVEN or SIDEN is set to 1, the FlexRay block writes only one pair of the enabled Sync Frame Tables corresponding to the next even-odd-cycle pair into the FRM. In this case, the FlexRay block clears the SDVEN or SIDEN bits immediately. If this trigger is set to 0 while SDVEN or SIDEN is set to 1, the FlexRay block writes continuously the enabled Sync Frame Tables into the FRM. 0 Write continuously pairs of enabled Sync Frame Tables into FRM. 1 Write only one pair of enabled Sync Frame Tables into FRM.
SDVEN	Sync Frame Deviation Table Enable. This bit controls the generation of the Sync Frame Deviation Tables. The application must set this bit to request the FlexRay block to write the Sync Frame Deviation Tables into the FRM. 0 Do not write Sync Frame Deviation Tables 1 Write Sync Frame Deviation Tables into FRM Note: If SDVEN is set to 1, then SIDEN must also be set to 1.
SIDEN	Sync Frame ID Table Enable. This bit controls the generation of the Sync Frame ID Tables. The application must set this bit to 1 to request the FlexRay block to write the Sync Frame ID Tables into the FRM. 0 Do not write Sync Frame ID Tables 1 Write Sync Frame ID Tables into FRM

30.5.2.34 Sync Frame ID Rejection Filter Register (SFIDRFR)

Base + 0x0046				16-bit write access required								Write: Normal Mode				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	SYNFRID									
W	[Shaded]						[Shaded]									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-34. Sync Frame ID Rejection Filter Register (SFIDRFR)

This register defines the Sync Frame Rejection Filter ID. The application must update this register outside of the static segment. If the application updates this register in the static segment, it can appear that the FlexRay block accepts the sync frame in the current cycle.

Table 30-41. SFIDRFR Field Descriptions

Field	Description
SYNFRID	Sync Frame Rejection ID. This field defines the frame ID of a frame that must not be used for clock synchronization. For details see Section 30.6.15.2, "Sync Frame Rejection Filtering" .

30.5.2.35 Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)

Base + 0x0048

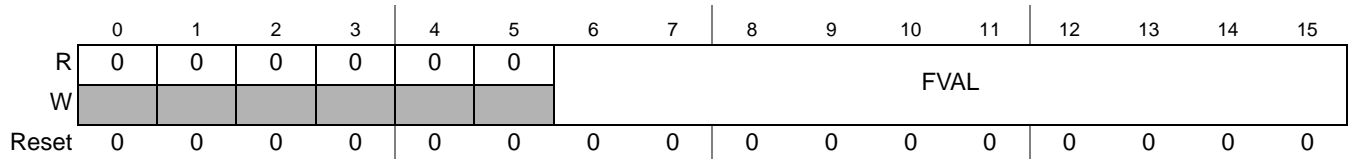
Write: *POC:config*

Figure 30-35. Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)

This register defines the sync frame acceptance filter value. For details on filtering, see [Section 30.6.15](#), “Sync Frame Filtering”.

Table 30-42. SFIDAFVR Field Descriptions

Field	Description
FVAL	Filter Value. This field defines the value for the sync frame acceptance filtering.

30.5.2.36 Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)

Base + 0x004A

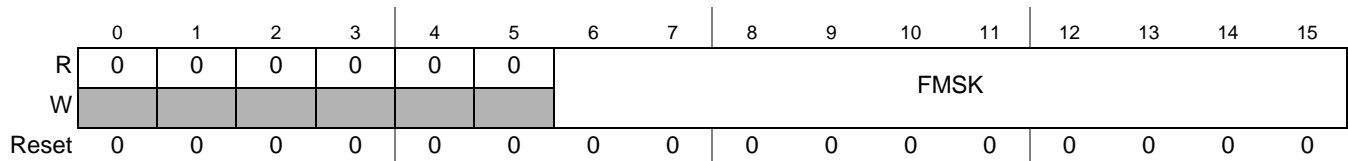
Write: *POC:config*

Figure 30-36. Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)

This register defines the sync frame acceptance filter mask. For details on filtering see [Section 30.6.15.1](#), “Sync Frame Acceptance Filtering”.

Table 30-43. SFIDAFMR Field Descriptions

Field	Description
FMSK	Filter Mask. This field defines the mask for the sync frame acceptance filtering.

30.5.2.37 Network Management Vector Registers (NMVR0–NMVR5)

Base + 0x004C (NMVR0)

Base + 0x004E (NMVR1)

Base + 0x0050 (NMVR2)

Base + 0x0052 (NMVR3)

Base + 0x0054 (NMVR4)

Base + 0x0056 (NMVR5)

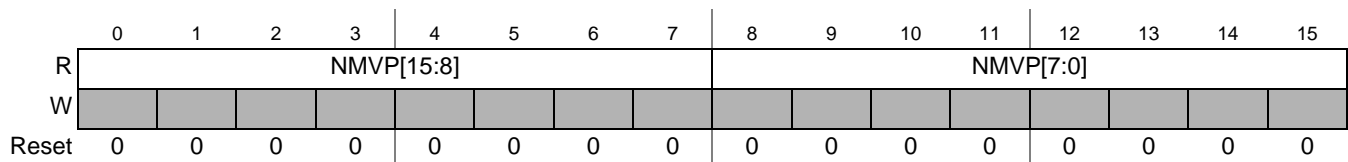


Figure 30-37. Network Management Vector Registers (NMVR0–NMVR5)

Each of these six registers holds one part of the Network Management Vector. The length of the Network Management Vector is configured in the [Network Management Vector Length Register \(NMVLR\)](#). If NMVLR is programmed with a value that is less than 12 bytes, the remaining bytes of the [Network Management Vector Registers \(NMVR0–NMVR5\)](#), which are not used for the Network Management Vector accumulating, will remain 0.

The NMVR provides accrued information over all received NMVs in the last communication cycle. All NMVs received in one cycle are ORed into the NMVR. The NMVR is updated at the end of the communication cycle.

Table 30-44. NMVR[0:5] Field Descriptions

Field	Description
NMVP	Network Management Vector Part. The mapping between the Network Management Vector Registers (NMVR0–NMVR5) and the receive message buffer payload bytes in NMV[0:11] is depicted in Table 30-45 .

Table 30-45. Mapping of NMVR_n to the Received Payload Bytes NMV_n

NMVR _n Register	NMV _n Received Payload
NMVR0[NMVP[15:8]]	NMV0
NMVR0[NMVP[7:0]]	NMV1
NMVR1[NMVP[15:8]]	NMV2
NMVR1[NMVP[7:0]]	NMV3
...	
NMVR5[NMVP[15:8]]	NMV10
NMVR5[NMVP[7:0]]	NMV11

30.5.2.38 Network Management Vector Length Register (NMVLR)

Base + 0x0058

Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	NMVL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-38. Network Management Vector Length Register (NMVLR)

This register defines the length of the network management vector in bytes.

Table 30-46. NMVLR Field Descriptions

Field	Description
NMVL	Network Management Vector Length. protocol related variable: gNetworkManagementVectorLength This field defines the length of the Network Management Vector in bytes. Legal values are between 0 and 12.

30.5.2.39 Timer Configuration and Control Register (TICCR)

Base + 0x005A

Write: T2_CFG: *POC:config*

T2_REP, T1_REP, T1SP, T2SP, T1TR, T2TR: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T2_CFG	T2_REP	0	0	0	T2ST	0	0	0	T1_REP	0	0	0	T1ST
W						T2SP	T2TR							T1SP	T1TR	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-39. Timer Configuration and Control Register (TICCR)

This register is used to configure and control the two timers T1 and T2. For timer details, see [Section 30.6.17, “Timer Support”](#). The Timer T1 is an absolute timer. The Timer T2 can be configured as an absolute or relative timer.

Table 30-47. TICCR Field Descriptions

Field	Description
T2_CFG	Timer T2 Configuration. This bit configures the timebase mode of Timer T2. 0 T2 is absolute timer. 1 T2 is relative timer.
T2_REP	Timer T2 Repetitive Mode. This bit configures the repetition mode of Timer T2. 0 T2 is non repetitive 1 T2 is repetitive
T2SP	Timer T2 Stop. This trigger bit is used to stop timer T2. 0 no effect 1 stop timer T2
T2TR	Timer T2 Trigger. This trigger bit is used to start timer T2. 0 no effect 1 start timer T2
T2ST	Timer T2 State. This status bit provides the current state of timer T2. 0 timer T2 is idle 1 timer T2 is running
T1_REP	Timer T1 Repetitive Mode. This bit configures the repetition mode of timer T1. 0 T1 is non repetitive 1 T1 is repetitive
T1SP	Timer T1 Stop. This trigger bit is used to stop timer T1. 0 no effect 1 stop timer T1
T1TR	Timer T1 Trigger. This trigger bit is used to start timer T1. 0 no effect 1 start timer T1
T1ST	Timer T1 State. This status bit provides the current state of timer T1. 0 timer T1 is idle 1 timer T1 is running

NOTE

Both timers are deactivated immediately when the protocol enters a state different from *POC:normal active* or *POC:normal passive*.

30.5.2.40 Timer 1 Cycle Set Register (TI1CYSR)

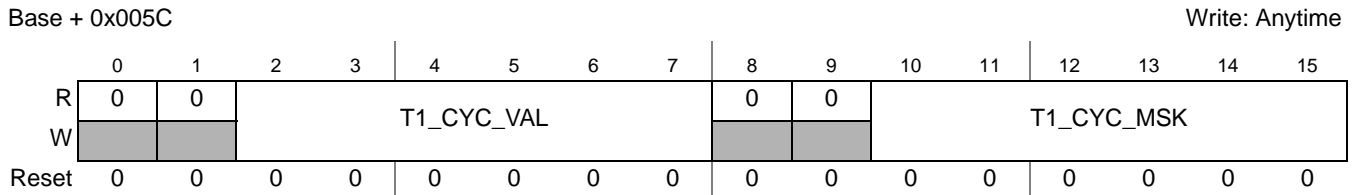


Figure 30-40. Timer 1 Cycle Set Register (TI1CYSR)

This register defines the cycle filter value and the cycle filter mask for timer T1. For a detailed description of timer T1, refer to [Section 30.6.17.1, “Absolute Timer T1”](#).

Table 30-48. TI1CYSR Field Descriptions

Field	Description
T1_CYC_VAL	Timer T1 Cycle Filter Value. This field defines the cycle filter value for timer T1.
T1_CYC_MSK	Timer T1 Cycle Filter Mask. This field defines the cycle filter mask for timer T1.

NOTE

If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

30.5.2.41 Timer 1 Macrotick Offset Register (TI1MTOR)

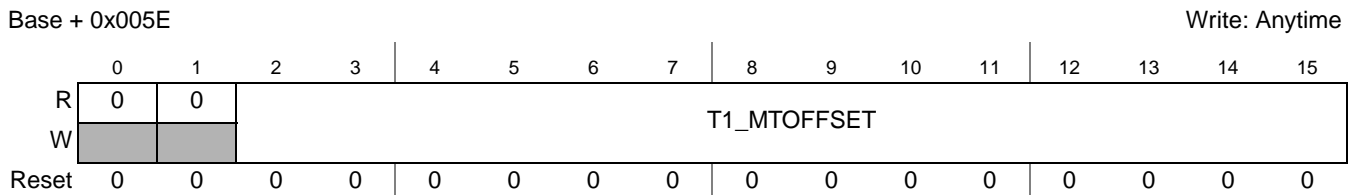


Figure 30-41. Timer 1 Macrotick Offset Register (TI1MTOR)

This register holds the macrotick offset value for timer T1. For a detailed description of timer T1, refer to [Section 30.6.17.1, “Absolute Timer T1”](#).

Table 30-49. TI1MTOR Field Descriptions

Field	Description
T1_MTOFFSET	Timer 1 Macrotick Offset. This field defines the macrotick offset value for timer 1.

NOTE

If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

30.5.2.42 Timer 2 Configuration Register 0 (TI2CR0)

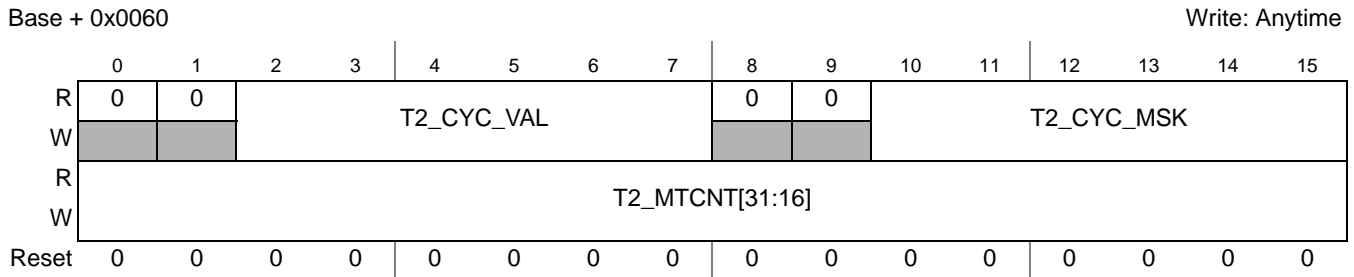


Figure 30-42. Timer 2 Configuration Register 0 (TI2CR0)

The content of this register depends on the value of the T2_CFG bit in the [Timer Configuration and Control Register \(TICCR\)](#). For a detailed description of timer T2, refer to [Section 30.6.17.2, “Absolute / Relative Timer T2”](#).

Table 30-50. TI2CR0 Field Descriptions

Field	Description
Fields for absolute timer T2 (TICCR[T2_CFG] = 0)	
T2_CYC_VAL	Timer T2 Cycle Filter Value. This field defines the cycle filter value for timer T2.
T2_CYC_MSK	Timer T2 Cycle Filter Mask. This field defines the cycle filter mask for timer T2.
Fields for relative timer T2 (TICCR[T2_CFG] = 1)	
T2_MTCNT[31:16]	Timer T2 Macrotick High Word. This field defines the high word of the macrotick count for timer T2.

NOTE

If timer T2 is configured as an *absolute* timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and timer T2 will expire according to the changed values.

If timer T2 is configured as a *relative* timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

30.5.2.43 Timer 2 Configuration Register 1 (TI2CR1)

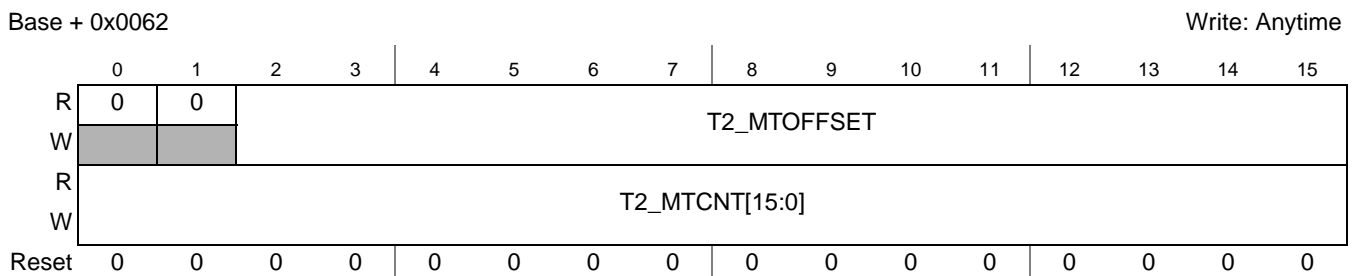


Figure 30-43. Timer 2 Configuration Register 1 (TI2CR1)

The content of this register depends on the value of the T2_CFG bit in the [Timer Configuration and Control Register \(TICCR\)](#). For a detailed description of timer T2, refer to [Section 30.6.17.2, “Absolute / Relative Timer T2”](#).

Table 30-51. TI2CR1 Field Descriptions

Field	Description
Fields for absolute timer T2 (TICCR[T2_CFG] = 0)	
T2_MTOFFSET	Timer T2 Macrotick Offset. This field holds the macrotick offset value for timer T2.
Fields for relative timer T2 (TICCR[T2_CFG] = 1)	
T2_MTCNT[15:0]	Timer T2 Macrotick Low Word. This field defines the low word of the macrotick value for timer T2.

NOTE

If timer T2 is configured as an *absolute* timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and the timer T2 will expire according to the changed values.

If timer T2 is configured as a *relative* timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

30.5.2.44 Slot Status Selection Register (SSSR)

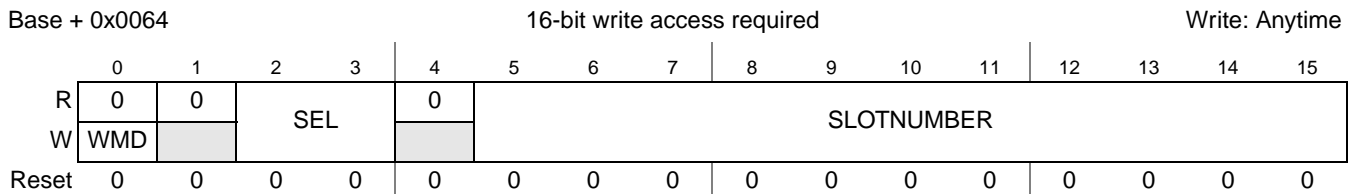


Figure 30-44. Slot Status Selection Register (SSSR)

This register is used to access the four internal non memory-mapped slot status selection registers SSSR0 to SSSR3. Each internal registers selects a slot, or symbol window/NIT, whose status vector will be saved in the corresponding [Slot Status Registers \(SSR0–SSR7\)](#) according to [Table 30-53](#). For a detailed description of slot status monitoring, refer to [Section 30.6.18, “Slot Status Monitoring”](#).

Table 30-52. SSSR Field Descriptions

Field	Description
WMD	Write Mode. This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.

Table 30-52. SSSR Field Descriptions (continued)

Field	Description
SEL	Selector. This field selects one of the four internal slot status selection registers for access. 00 select SSSR0. 01 select SSSR1. 10 select SSSR2. 11 select SSSR3.
SLOTNUMBER	Slot Number. This field specifies the number of the slot whose status will be saved in the corresponding slot status registers. Note: If this value is set to 0, the related slot status register provides the status of the symbol window after the NIT start, and provides the status of the NIT after the cycle start.

Table 30-53. Mapping Between SSSRn and SSRn

Internal Slot Status Selection Register	Write the Slot Status of the Slot Selected by SSSRn for each			
	Even Communication Cycle		Odd Communication Cycle	
	For Channel B to	For Channel A to	For Channel B to	For Channel A to
SSSR0	SSR0[15:8]	SSR0[7:0]	SSR1[15:8]	SSR1[7:0]
SSSR1	SSR2[15:8]	SSR2[7:0]	SSR3[15:8]	SSR3[7:0]
SSSR2	SSR4[15:8]	SSR4[7:0]	SSR5[15:8]	SSR5[7:0]
SSSR3	SSR6[15:8]	SSR6[7:0]	SSR7[15:8]	SSR7[7:0]

30.5.2.45 Slot Status Counter Condition Register (SSCCR)

Base + 0x0066		16-bit write access required										Write: Anytime				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	CNTCFG		MCY	VFR	SYF	NUF	SUF	STATUSMASK			
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-45. Slot Status Counter Condition Register (SSCCR)

This register is used to access and program the four internal non-memory mapped Slot Status Counter Condition Registers SSCCR0 to SSCCR3. Each of these four internal slot status counter condition registers defines the mode and the conditions for incrementing the counter in the corresponding [Slot Status Counter Registers \(SSCR0–SSCR3\)](#). The correspondence is given in [Table 30-55](#). For a detailed description of slot status counters, refer to [Section 30.6.18.4, “Slot Status Counter Registers”](#).

Table 30-54. SSCCR Field Descriptions

Field	Description
WMD	Write Mode. This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	Selector. This field selects one of the four internal slot counter condition registers for access. 00 select SSCCR0. 01 select SSCCR1. 10 select SSCCR2. 11 select SSCCR3.
CNTCFG	Counter Configuration. These bit field controls the channel related incrementing of the slot status counter. 00 increment by 1 if condition is fulfilled on channel A. 01 increment by 1 if condition is fulfilled on channel B. 10 increment by 1 if condition is fulfilled on at least one channel. 11 increment by 2 if condition is fulfilled on both channels channel. increment by 1 if condition is fulfilled on only one channel.
MCY	Multi Cycle Selection. This bit defines whether the slot status counter accumulates over multiple communication cycles or provides information for the previous communication cycle only. 0 The Slot Status Counter provides information for the previous communication cycle only. 1 The Slot Status Counter accumulates over multiple communication cycles.
VFR	Valid Frame Restriction. This bit is used to restrict the counter to received valid frames. 0 The counter is not restricted to valid frames only. 1 The counter is restricted to valid frames only.
SYF	Sync Frame Restriction. This bit is used to restrict the counter to received frames with the sync frame indicator bit set to 1. 0 The counter is not restricted with respect to the sync frame indicator bit. 1 The counter is restricted to frames with the sync frame indicator bit set to 1.
NUF	Null Frame Restriction. This bit is used to restrict the counter to received frames with the null frame indicator bit set to 0. 0 The counter is not restricted with respect to the null frame indicator bit. 1 The counter is restricted to frames with the null frame indicator bit set to 0.
SUF	Startup Frame Restriction. This bit is used to restrict the counter to received frames with the startup frame indicator bit set to 1. 0 The counter is not restricted with respect to the startup frame indicator bit. 1 The counter is restricted to received frames with the startup frame indicator bit set to 1.
STATUSMASK	Slot Status Mask. This bit field is used to enable the counter with respect to the four slot status error indicator bits. STATUSMASK[3] – This bit enables the counting for slots with the syntax error indicator bit set to 1. STATUSMASK[2] – This bit enables the counting for slots with the content error indicator bit set to 1. STATUSMASK[1] – This bit enables the counting for slots with the boundary violation indicator bit set to 1. STATUSMASK[0] – This bit enables the counting for slots with the transmission conflict indicator bit set to 1.

Table 30-55. Mapping between internal SSCCRn and SSCRn

Condition Register	Condition Defined for Register
SSCCR0	SSCR0
SSCCR1	SSCR1
SSCCR2	SSCR2
SSCCR3	SSCR3

30.5.2.46 Slot Status Registers (SSR0–SSR7)

Base + 0x0068 (SSR0)
 Base + 0x006A (SSR1)
 Base + 0x006C (SSR2)
 Base + 0x006E (SSR3)
 Base + 0x0070 (SSR4)
 Base + 0x0072 (SSR5)
 Base + 0x0074 (SSR6)
 Base + 0x0076 (SSR7)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-46. Slot Status Registers (SSR0–SSR7)

Each of these eight registers holds the status vector of the slot specified in the corresponding internal slot status selection register, which can be programmed using the [Slot Status Selection Register \(SSSR\)](#). Each register is updated after the end of the corresponding slot as shown in [Figure 30-141](#). The register bits are directly related to the protocol variables and described in more detail in [Section 30.6.18, “Slot Status Monitoring”](#).

Table 30-56. SSR0–SSR7 Field Descriptions

Field	Description
VFB	Valid Frame on Channel B. Protocol related variable: vSS!ValidFrame channel B 0 vSS!ValidFrame = 0 1 vSS!ValidFrame = 1
SYB	Sync Frame Indicator Channel B. Protocol related variable: vRF!Header!SyFIndicator channel B 0 vRF!Header!SyFIndicator = 0 1 vRF!Header!SyFIndicator = 1
NFB	Null Frame Indicator Channel B. Protocol related variable: vRF!Header!NFIndicator channel B 0 vRF!Header!NFIndicator = 0 1 vRF!Header!NFIndicator = 1
SUB	Startup Frame Indicator Channel B. Protocol related variable: vRF!Header!SuFIndicator channel B 0 vRF!Header!SuFIndicator = 0 1 vRF!Header!SuFIndicator = 1
SEB	Syntax Error on Channel B. Protocol related variable: vSS!SyntaxError channel B 0 vSS!SyntaxError = 0 1 vSS!SyntaxError = 1
CEB	Content Error on Channel B. Protocol related variable: vSS!ContentError channel B 0 vSS!ContentError = 0 1 vSS!ContentError = 1
BVB	Boundary Violation on Channel B. Protocol related variable: vSS!BViolation channel B 0 vSS!BViolation = 0 1 vSS!BViolation = 1
TCB	Transmission Conflict on Channel B. Protocol related variable: vSS!TxConflict channel B 0 vSS!TxConflict = 0 1 vSS!TxConflict = 1

Table 30-56. SSR0–SSR7 Field Descriptions (continued)

Field	Description
VFA	Valid Frame on Channel A. Protocol related variable: <i>vSS!ValidFrame</i> channel A 0 <i>vSS!ValidFrame</i> = 0 1 <i>vSS!ValidFrame</i> = 1
SYA	Sync Frame Indicator Channel A. Protocol related variable: <i>vRF!Header!SyFIndicator</i> channel A 0 <i>vRF!Header!SyFIndicator</i> = 0 1 <i>vRF!Header!SyFIndicator</i> = 1
NFA	Null Frame Indicator Channel A. Protocol related variable: <i>vRF!Header!NFIndicator</i> channel A 0 <i>vRF!Header!NFIndicator</i> = 0 1 <i>vRF!Header!NFIndicator</i> = 1
SUA	Startup Frame Indicator Channel A. Protocol related variable: <i>vRF!Header!SuFIndicator</i> channel A 0 <i>vRF!Header!SuFIndicator</i> = 0 1 <i>vRF!Header!SuFIndicator</i> = 1
SEA	Syntax Error on Channel A. Protocol related variable: <i>vSS!SyntaxError</i> channel A 0 <i>vSS!SyntaxError</i> = 0 1 <i>vSS!SyntaxError</i> = 1
CEA	Content Error on Channel A. Protocol related variable: <i>vSS!ContentError</i> channel A 0 <i>vSS!ContentError</i> = 0 1 <i>vSS!ContentError</i> = 1
BVA	Boundary Violation on Channel A. Protocol related variable: <i>vSS!BViolation</i> channel A 0 <i>vSS!BViolation</i> = 0 1 <i>vSS!BViolation</i> = 1
TCA	Transmission Conflict on Channel A. Protocol related variable: <i>vSS!TxConflict</i> channel A 0 <i>vSS!TxConflict</i> = 0 1 <i>vSS!TxConflict</i> = 1

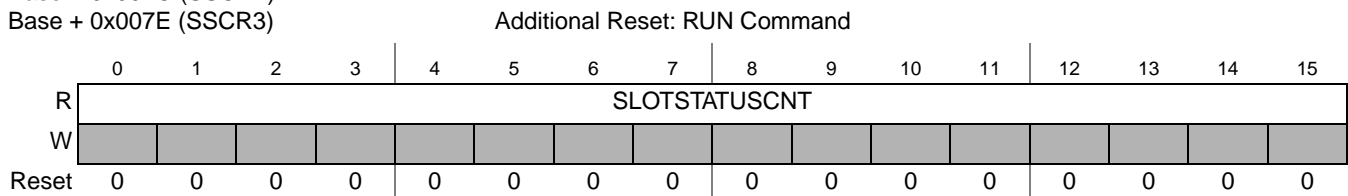
30.5.2.47 Slot Status Counter Registers (SSCR0–SSCR3)

Base + 0x0078 (SSCR0)

Base + 0x007A (SSCR1)

Base + 0x007C (SSCR2)

Base + 0x007E (SSCR3)

**Figure 30-47. Slow Status Counter Registers (SSCR0–SSCR3)**

Each of these four registers provides the slot status counter value for the previous communication cycle(s) and is updated at the cycle start. The provided value depends on the control bits and fields in the related internal slot status counter condition register SSCRn, which can be programmed by using the [Slot Status Counter Condition Register \(SSCCR\)](#). For more details, see [Section 30.6.18.4, “Slot Status Counter Registers”](#).

NOTE

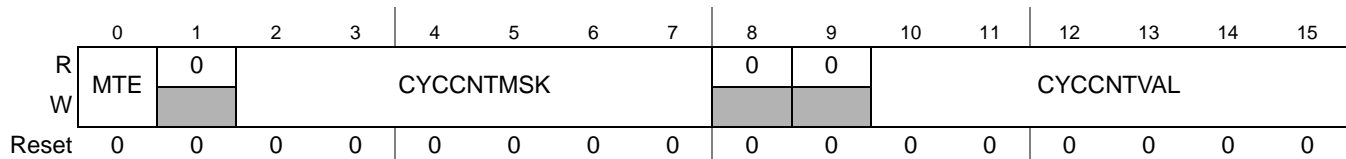
If the counter has reached its maximum value 0xFFFF and is in the multicycle mode, i.e. SSSCCRn[MCY] = 1, the counter is not reset to 0x0000. The application can reset the counter by clearing the SSSCCRn[MCY] bit and waiting for the next cycle start, when the FlexRay block clears the counter. Subsequently, the counter can be set into the multicycle mode again.

Table 30-57. SSCR0–SSCR3 Field Descriptions

Field	Description
SLOTSTATUSCNT	Slot Status Counter. This field provides the current value of the Slot Status Counter.

30.5.2.48 MTS A Configuration Register (MTSACFR)

Base + 0x0080

Write: MTE: Anytime
CYCCNTMSK, CYCCNTVAL: *POC:config***Figure 30-48. MTS A Configuration Register (MTSACFR)**

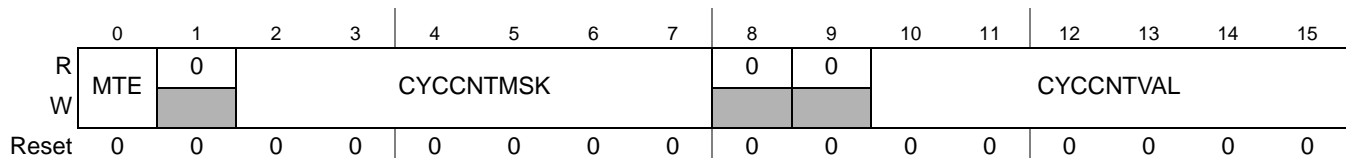
This register controls the transmission of the Media Access Test Symbol MTS on channel A. For more details, see [Section 30.6.13, “MTS Generation”](#).

Table 30-58. MTSACFR Field Descriptions

Field	Description
MTE	Media Access Test Symbol Transmission Enable. This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled 1 MTS transmission enabled
CYCCNTMSK	Cycle Counter Mask. This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	Cycle Counter Value. This field provides the filter value for the MTS cycle count filter.

30.5.2.49 MTS B Configuration Register (MTSBCFR)

Base + 0x0082

Write: MTE: Anytime
CYCCNTMSK, CYCCNTVAL: *POC:config***Figure 30-49. MTS B Configuration Register (MTSBCFR)**

This register controls the transmission of the Media Access Test Symbol MTS on channel B. For more details, see [Section 30.6.13, “MTS Generation”](#).

Table 30-59. MTSBCFR Field Descriptions

Field	Description
MTE	Media Access Test Symbol Transmission Enable. This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled 1 MTS transmission enabled
CYCCNTMSK	Cycle Counter Mask. This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	Cycle Counter Value. This field provides the filter value for the MTS cycle count filter.

30.5.2.50 Receive Shadow Buffer Index Register (RSBIR)

Base + 0x0084

16-bit write access required

Write: WMD, SEL: Any Time
RSBIDX: *POC:config*

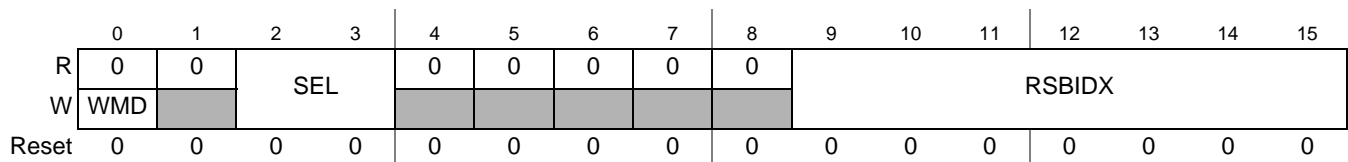


Figure 30-50. Receive Shadow Buffer Index Register (RSBIR)

This register is used to provide and retrieve the indices of the message buffer header fields currently associated with the receive shadow buffers. For more details on the receive shadow buffer concept, refer to [Section 30.6.6.3.5, “Receive Shadow Buffers Concept”](#).

Table 30-60. RSBIR Field Descriptions

Field	Description
WMD	Write Mode. This bit controls the write mode for this register. 0 update SEL and RSBIDX field on register write 1 update only SEL field on register write
SEL	Selector. This field is used to select the internal receive shadow buffer index register for access. 00 RSBIR_A1 — receive shadow buffer index register for channel A, segment 1 01 RSBIR_A2 — receive shadow buffer index register for channel A, segment 2 10 RSBIR_B1 — receive shadow buffer index register for channel B, segment 1 11 RSBIR_B2 — receive shadow buffer index register for channel B, segment 2
RSBIDX	Receive Shadow Buffer Index. This field contains the current index of the message buffer header field of the receive shadow message buffer selected by the SEL field. The FlexRay block uses this index to determine the physical location of the shadow buffer header field in the FlexRay memory. The FlexRay block will update this field during receive operation. The application provides initial message buffer header index value in the configuration phase. FlexRay block: Updates the message buffer header index after successful reception. Application: Provides initial message buffer header index.

30.5.2.51 Receive FIFO Selection Register (RFSR)

Base + 0x0086

Write: Anytime

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

Figure 30-51. Receive FIFO Selection Register (RFSR)

This register is used to select a receiver FIFO for subsequent access through the receiver FIFO configuration registers summarized in [Table 30-61](#).

Table 30-61. SEL Controlled Receiver FIFO Registers

Register
Receive FIFO Start Index Register (RFSIR)
Receive FIFO Depth and Size Register (RFDSR)
Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)
Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)
Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)
Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)
Receive FIFO Range Filter Configuration Register (RFRFCFR)
Receive FIFO Range Filter Control Register (RFRFCTR)

Table 30-62. RFSR Field Descriptions

Field	Description
SEL	Select. This control bit selects the receiver FIFO for subsequent programming. 0 Receiver FIFO for channel A selected 1 Receiver FIFO for channel B selected

30.5.2.52 Receive FIFO Start Index Register (RFSIR)

Base + 0x0088

Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	SIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-52. Receive FIFO Start Index Register (RFSIR)

This register defines the message buffer header index of the first message buffer of the selected FIFO.

Table 30-63. RFSIR Field Descriptions

Field	Description
SIDX	Start Index. This field defines the number of the message buffer header field of the first message buffer of the selected receive FIFO. The FlexRay block uses the value of the SIDX field to determine the physical location of the receiver FIFO's first message buffer header field.

30.5.2.53 Receive FIFO Depth and Size Register (RFDSR)

Base + 0x008A

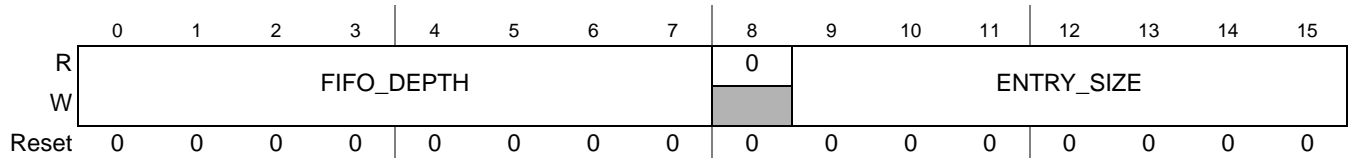
Write: *POC:config*

Figure 30-53. Receive FIFO Depth and Size Register (RFDSR)

This register defines the structure of the selected FIFO, i.e. the number of entries and the size of each entry.

Table 30-64. RFDSR Field Descriptions

Field	Description
FIFO_DEPTH	FIFO Depth. This field defines the depth of the selected receive FIFO, i.e. the number of entries.
ENTRY_SIZE	Entry Size. This field defines the size of the frame data sections for the selected receive FIFO in 2 byte entities.

30.5.2.54 Receive FIFO A Read Index Register (RFARIR)

Base + 0x008C

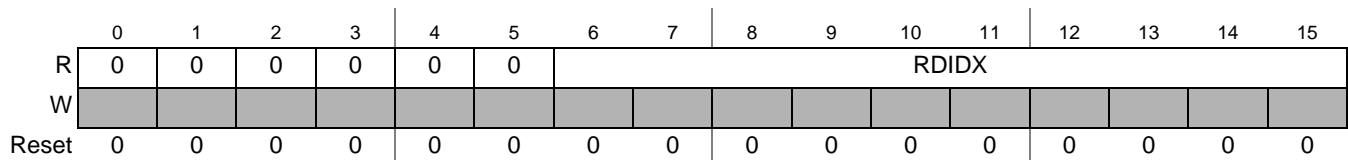


Figure 30-54. Receive FIFO A Read Index Register (RFARIR)

This register provides the message buffer header index of the next available receive FIFO A entry that the application can read.

Table 30-65. RFARIR Field Descriptions

Field	Description
RDIDX	Read Index. This field provides the message buffer header index of the next available receive FIFO message buffer that the application can read. The FlexRay block increments this index when the application writes to the FNEAIF flag in the Global Interrupt Flag and Enable Register (GIFER) . The index wraps back to the first message buffer header index if the end of the FIFO was reached.

NOTE

If the receive FIFO not empty flag FNEAIF is not set, the RDIDX field points to an physical message buffer which content is not valid. Only when FNEAIF is set, the message buffer indicated by RDIDX contains valid data.

30.5.2.55 Receive FIFO B Read Index Register (RFBRIR)

Base + 0x008E

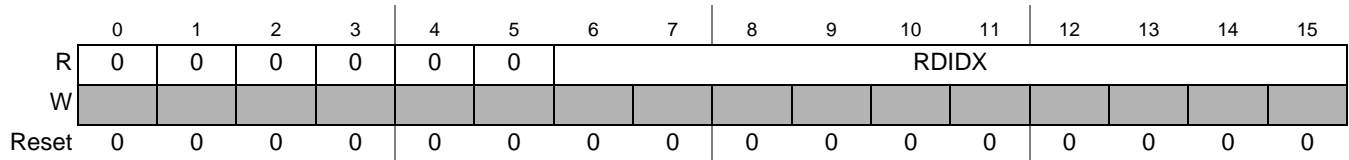


Figure 30-55. Receive FIFO B Read Index Register (RFBRIR)

This register provides the message buffer header index of the next available receive FIFO B entry that the application can read.

Table 30-66. RFBRIR Field Descriptions

Field	Description
RDIDX	Read Index. This field provides the message buffer header index of the next available receive FIFO entry that the application can read. The FlexRay block increments this index when the application writes to the FNEBIF flag in the Global Interrupt Flag and Enable Register (GIFER) . The index wraps back to the first message buffer header index if the end of the FIFO was reached.

NOTE

If the receive FIFO not empty flag FNEBIF is not set, the RDIDX field points to an physical message buffer which content is not valid. Only when FNEBIF is set, the message buffer indicated by RDIDX contains valid data.

30.5.2.56 Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)

Base + 0x0090

Write: *POC:config*

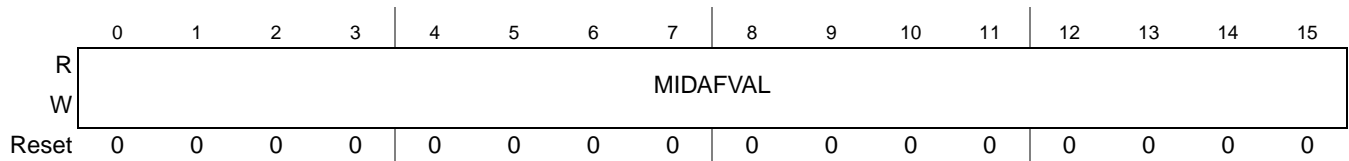


Figure 30-56. Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)

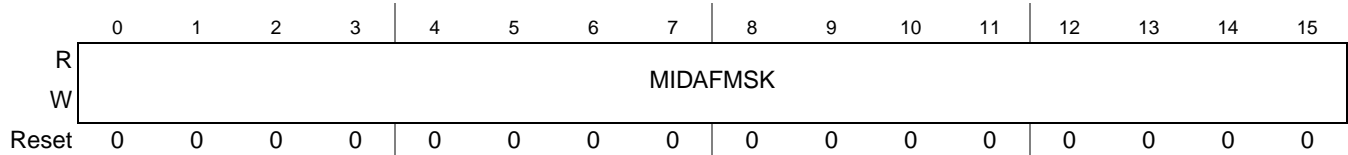
This register defines the filter value for the message ID acceptance filter of the selected receive FIFO. For details on message ID filtering see [Section 30.6.9.5, “Receive FIFO filtering.”](#)

Table 30-67. RFMIDAFVR Field Descriptions

Field	Description
MIDAFVAL	Message ID Acceptance Filter Value. Filter value for the message ID acceptance filter.

30.5.2.57 Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)

Base + 0x0092

Write: *POC:config***Figure 30-57. Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)**

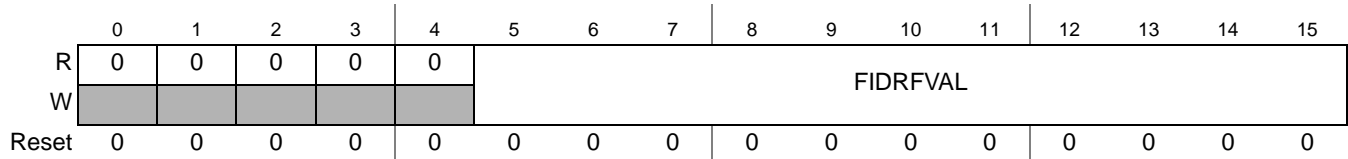
This register defines the filter mask for the message ID acceptance filter of the selected receive FIFO. For details on message ID filtering see [Section 30.6.9.5, “Receive FIFO filtering.”](#)

Table 30-68. RFMIAFMR Field Descriptions

Field	Description
MIDAFMSK	Message ID Acceptance Filter Mask. Filter mask for the message ID acceptance filter.

30.5.2.58 Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)

Base + 0x0094

Write: *POC:config***Figure 30-58. Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)**

This register defines the filter value for the frame ID rejection filter of the selected receive FIFO. For details on frame ID filtering see [Section 30.6.9.5, “Receive FIFO filtering.”](#)

Table 30-69. RFFIDRFVR Field Descriptions

Field	Description
FIDRFVAL	Frame ID Rejection Filter Value. Filter value for the frame ID rejection filter.

30.5.2.59 Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)

Base + 0x0096

Write: *POC:config*

Figure 30-59. Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)

This register defines the filter mask for the frame ID rejection filter of the selected receive FIFO. For details on frame ID filtering see [Section 30.6.9.5, “Receive FIFO filtering.”](#)

Table 30-70. RFFIDRFMR Field Descriptions

Field	Description
FIDRFMSK	Frame ID Rejection Filter Mask. Filter mask for the frame ID rejection filter.

30.5.2.60 Receive FIFO Range Filter Configuration Register (RFRFCFR)

Base + 0x0098

16-bit write access required

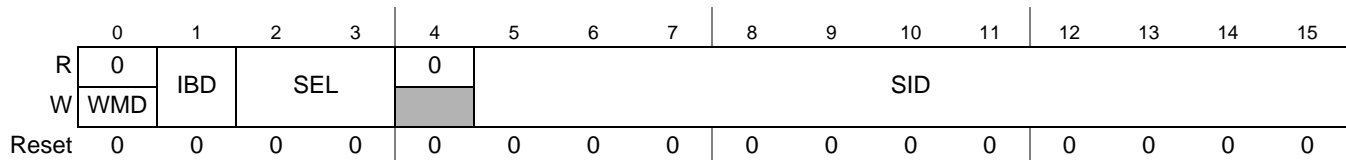
Write: WMD, IBD, SEL: Any Time
SID: *POC:config*

Figure 30-60. Receive FIFO Range Filter Configuration Register (RFRFCFR)

This register provides access to the four internal frame ID range filter boundary registers of the selected receive FIFO. For details on frame ID range filter see [Section 30.6.9.5, “Receive FIFO filtering”](#).

Table 30-71. RFRFCFR Field Descriptions

Field	Description
WMD	Write Mode. This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL and IBD field only on write access.
IBD	Interval Boundary. This control bit selects the interval boundary to be programmed with the SID value. 0 program lower interval boundary 1 program upper interval boundary
SEL	Filter Selector. This control field selects the frame ID range filter to be accessed. 00 select frame ID range filter 0. 01 select frame ID range filter 1. 10 select frame ID range filter 2. 11 select frame ID range filter 3.
SID	Slot ID. Defines the IBD-selected frame ID boundary value for the SEL-selected range filter.

30.5.2.61 Receive FIFO Range Filter Control Register (RFRFCTR)

Base + 0x009A

Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	F3MD	F2MD	F1MD	F0MD	0	0	0	0	F3EN	F2EN	F1EN	F0EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-61. Receive FIFO Range Filter Control Register (RFRFCTR)

This register is used to enable and disable each frame ID range filter and to define whether it is running as acceptance or rejection filter.

Table 30-72. RFRFCTR Field Descriptions

Field	Description
F3MD	Range Filter 3 Mode. This control bit defines the filter mode of the frame ID range filter 3. 0 range filter 3 runs as acceptance filter 1 range filter 3 runs as rejection filter
F2MD	Range Filter 2 Mode. This control bit defines the filter mode of the frame ID range filter 2. 0 range filter 2 runs as acceptance filter 1 range filter 2 runs as rejection filter
F1MD	Range Filter 1 Mode. This control bit defines the filter mode of the frame ID range filter 1. 0 range filter 1 runs as acceptance filter 1 range filter 1 runs as rejection filter
F0MD	Range Filter 0 Mode. This control bit defines the filter mode of the frame ID range filter 0. 0 range filter 0 runs as acceptance filter 1 range filter 0 runs as rejection filter
F3EN	Range Filter 3 Enable. This control bit is used to enable and disable the frame ID range filter 3. 0 range filter 3 disabled 1 range filter 3 enabled
F2EN	Range Filter 2 Enable. This control bit is used to enable and disable the frame ID range filter 2. 0 range filter 2 disabled 1 range filter 2 enabled
F1EN	Range Filter 1 Enable. This control bit is used to enable and disable the frame ID range filter 1. 0 range filter 1 disabled 1 range filter 1 enabled
F0EN	Range Filter 0 Enable. This control bit is used to enable and disable the frame ID range filter 0. 0 range filter 0 disabled 1 range filter 0 enabled

30.5.2.62 Last Dynamic Transmit Slot Channel A Register (LDTXSLAR)

Base + 0x009C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-62. Last Dynamic Slot Channel A Register (LDTXSLAR)

This register provides the number of the last transmission slot in the dynamic segment for channel A. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 30-73. LDTXSLAR Field Descriptions

Field	Description
LASTDYNTX SLOTA	Last Dynamic Transmission Slot Channel A. Protocol related variable: zLastDynTxSlot channel A. Number of the last transmission slot in the dynamic segment for channel A. If no frame was transmitted during the dynamic segment on channel A, the value of this field is set to 0.

30.5.2.63 Last Dynamic Transmit Slot Channel B Register (LDTXSLBR)

Base + 0x009E

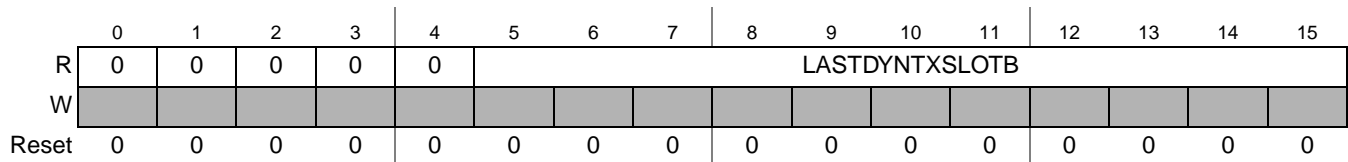


Figure 30-63. Last Dynamic Slot Channel B Register (LDTXSLBR)

This register provides the number of the last transmission slot in the dynamic segment for channel B. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 30-74. LDTXSLBR Field Descriptions

Field	Description
LASTDYNTX SLOTB	Last Dynamic Transmission Slot Channel B. Protocol related variable: zLastDynTxSlot channel B. Number of the last transmission slot in the dynamic segment for channel B. If no frame was transmitted during the dynamic segment on channel B the value of this field is set to 0.

30.5.2.64 Protocol Configuration Registers

The following configuration registers provide the necessary configuration information to the protocol engine. The individual values in the registers are described in [Table 30-75](#). For more details about the FlexRay related configuration parameters and the allowed parameter ranges, see *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

Table 30-75. Protocol Configuration Register Fields (Sheet 1 of 3)

Name	Description ¹	Min	Max	Unit	PCR
coldstart_attempts	gColdstartAttempts			number	3
action_point_offset	gdActionPointOffset - 1			MT	0
cas_rx_low_max	gdCASRxLowMax - 1			gdBit	4
dynamic_slot_idle_phase	gdDynamicSlotIdlePhase			minislot	28
minislot_action_point_offset	gdMinislotActionPointOffset - 1			MT	3
minislot_after_action_point	gdMinislot - gdMinislotActionPointOffset - 1			MT	2
static_slot_length	gdStaticSlot			MT	0

Table 30-75. Protocol Configuration Register Fields (Sheet 2 of 3)

Name	Description ¹	Min	Max	Unit	PCR
static_slot_after_action_point	$gdStaticSlot - gdActionPointOffset - 1$			MT	13
symbol_window_exists	$gdSymbolWindow \neq 0$	0	1	bool	9
symbol_window_after_action_point	$gdSymbolWindow - gdActionPointOffset - 1$			MT	6
tss_transmitter	$gdTSSTransmitter$			$gdBit$	5
wakeup_symbol_rx_idle	$gdWakeupSymbolRxIdle$			$gdBit$	5
wakeup_symbol_rx_low	$gdWakeupSymbolRxLow$			$gdBit$	3
wakeup_symbol_rx_window	$gdWakeupSymbolRxWindow$			$gdBit$	4
wakeup_symbol_tx_idle	$gdWakeupSymbolTxIdle$			$gdBit$	8
wakeup_symbol_tx_low	$gdWakeupSymbolTxLow$			$gdBit$	5
noise_listen_timeout	$(gListenNoise * pdListenTimeout) - 1$			μT	16/17
macro_initial_offset_a	$pMacroInitialOffset[A]$			MT	6
macro_initial_offset_b	$pMacroInitialOffset[B]$			MT	16
macro_per_cycle	$gMacroPerCycle$			MT	10
macro_after_first_static_slot	$gMacroPerCycle - gdStaticSlot$			MT	1
macro_after_offset_correction	$gMacroPerCycle - gOffsetCorrectionStart$			MT	28
max_without_clock_correction_fatal	$gMaxWithoutClockCorrectionFatal$			cyclepairs	8
max_without_clock_correction_passive	$gMaxWithoutClockCorrectionPassive$			cyclepairs	8
minislot_exists	$gNumberOfMinislots \neq 0$	0	1	bool	9
minislots_max	$gNumberOfMinislots - 1$			minislot	29
number_of_static_slots	$gNumberOfStaticSlots$			static slot	2
offset_correction_start	$gOffsetCorrectionStart$			MT	11
payload_length_static	$gPayloadLengthStatic$			2-bytes	19
max_payload_length_dynamic	$pPayloadLengthDynMax$			2-bytes	24
first_minislot_action_point_offset	$\max(gdActionPointOffset, gdMinislotActionPointOffset) - 1$			MT	13
allow_halt_due_to_clock	$pAllowHaltDueToClock$			bool	26
allow_passive_to_active	$pAllowPassiveToActive$			cyclepairs	12
cluster_drift_damping	$pClusterDriftDamping$			μT	24
comp_accepted_startup_range_a	$pdAcceptedStartupRange - pDelayCompensationChA$			μT	22
comp_accepted_startup_range_b	$pdAcceptedStartupRange - pDelayCompensationChB$			μT	26
listen_timeout	$pdListenTimeout - 1$			μT	14/15
key_slot_id	$pKeySlotId$			number	18
key_slot_used_for_startup	$pKeySlotUsedForStartup$			bool	11
key_slot_used_for_sync	$pKeySlotUsedForSync$			bool	11
latest_tx	$gNumberOfMinislots - pLatestTx$			minislot	21
sync_node_max	$gSyncNodeMax$			number	30
micro_initial_offset_a	$pMicroInitialOffset[A]$			μT	20
micro_initial_offset_b	$pMicroInitialOffset[B]$			μT	20

Table 30-75. Protocol Configuration Register Fields (Sheet 3 of 3)

Name	Description ¹	Min	Max	Unit	PCR
micro_per_cycle	<i>pMicroPerCycle</i>			μT	22/23
micro_per_cycle_min	<i>pMicroPerCycle - pdMaxDrift</i>			μT	24/25
micro_per_cycle_max	<i>pMicroPerCycle + pdMaxDrift</i>			μT	26/27
micro_per_macro_nom_half	round(<i>pMicroPerMacroNom</i> / 2)			μT	7
offset_correction_out	<i>pOffsetCorrectionOut</i>			μT	9
rate_correction_out	<i>pRateCorrectionOut</i>			μT	14
single_slot_enabled	<i>pSingleSlotEnabled</i>			bool	10
wakeup_channel	<i>pWakeupChannel</i>	see Table 30-76			10
wakeup_pattern	<i>pWakeupPattern</i>			number	18
decoding_correction_a	<i>pDecodingCorrection</i> + <i>pDelayCompensation[A]</i> + 2			μT	19
decoding_correction_b	<i>pDecodingCorrection</i> + <i>pDelayCompensation[B]</i> + 2			μT	7
key_slot_header_crc	header CRC for key slot	0x000	0x7FF	number	12
extern_offset_correction	<i>pExternOffsetCorrection</i>			μT	29
extern_rate_correction	<i>pExternRateCorrection</i>			μT	21

¹ See *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* for detailed protocol parameter definitions

Table 30-76. Wakeup Channel Selection

wakeup_channel	Wakeup Channel
0	A
1	B

30.5.2.64.1 Protocol Configuration Register 0 (PCR0)

Base + 0x00A0

Write: *POC:config*

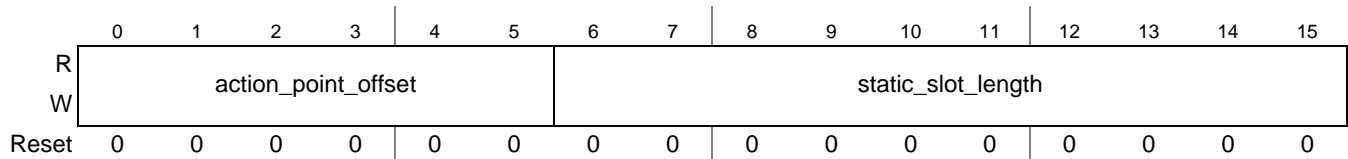


Figure 30-64. Protocol Configuration Register 0 (PCR0)

30.5.2.64.2 Protocol Configuration Register 1 (PCR1)

Base + 0x00A2

Write: *POC:config*

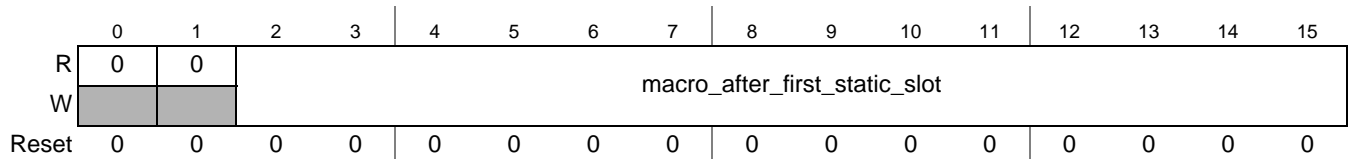


Figure 30-65. Protocol Configuration Register 1 (PCR1)

30.5.2.64.3 Protocol Configuration Register 2 (PCR2)

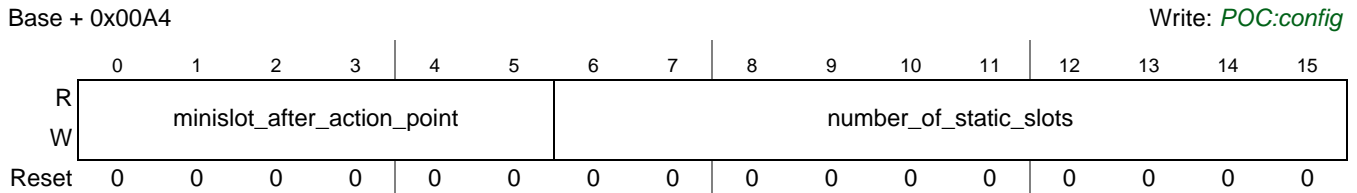


Figure 30-66. Protocol Configuration Register 2 (PCR2)

30.5.2.64.4 Protocol Configuration Register 3 (PCR3)

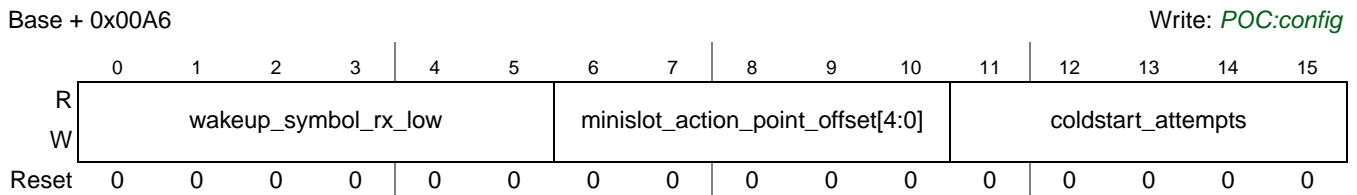


Figure 30-67. Protocol Configuration Register 3 (PCR3)

30.5.2.64.5 Protocol Configuration Register 4 (PCR4)

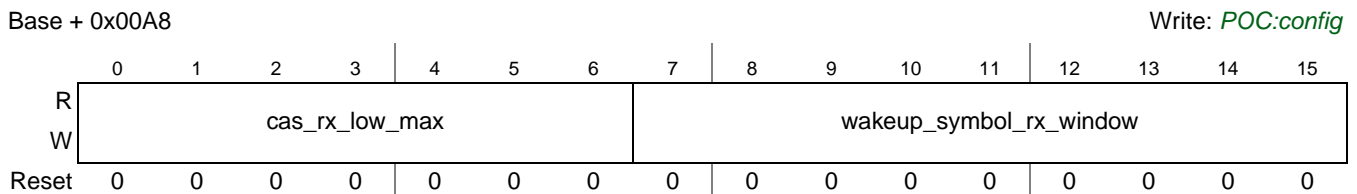


Figure 30-68. Protocol Configuration Register 4 (PCR4)

30.5.2.64.6 Protocol Configuration Register 5 (PCR5)

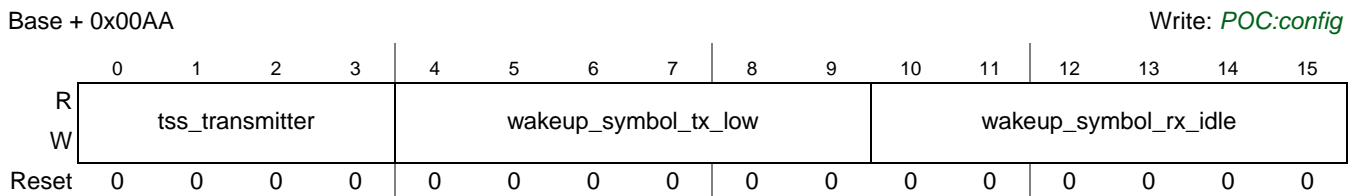


Figure 30-69. Protocol Configuration Register 5 (PCR5)

30.5.2.64.7 Protocol Configuration Register 6 (PCR6)

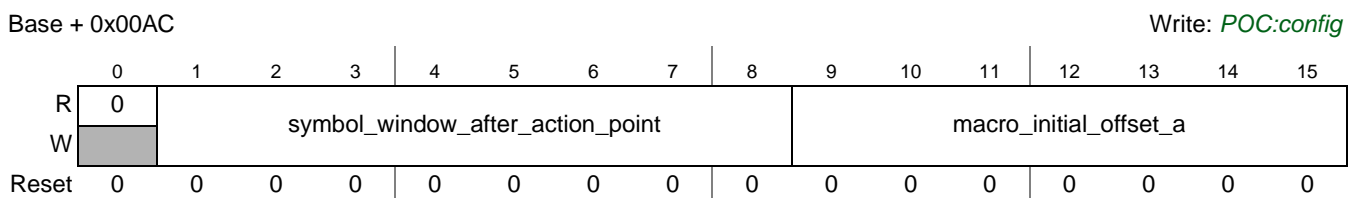


Figure 30-70. Protocol Configuration Register 6 (PCR6)

30.5.2.64.8 Protocol Configuration Register 7 (PCR7)

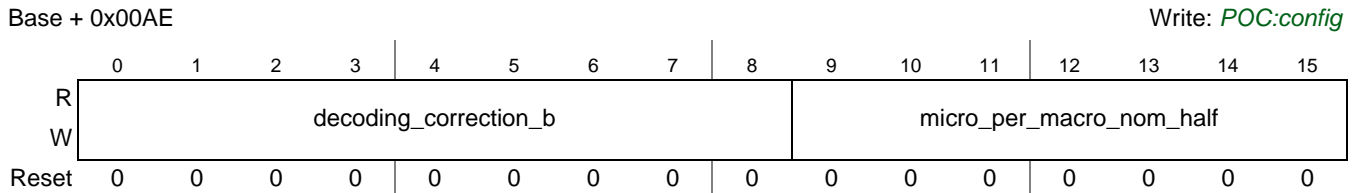


Figure 30-71. Protocol Configuration Register 7 (PCR7)

30.5.2.64.9 Protocol Configuration Register 8 (PCR8)

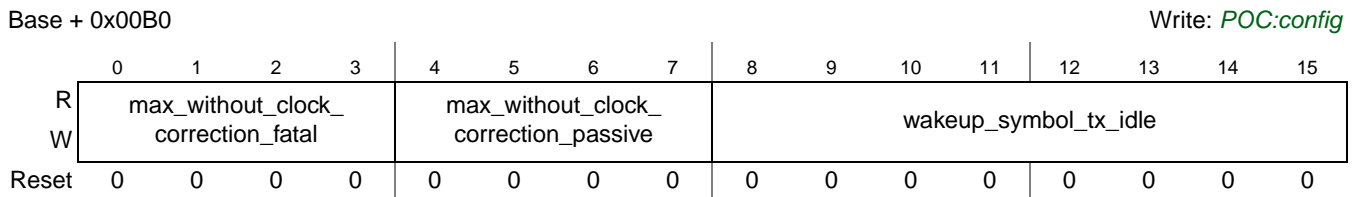


Figure 30-72. Protocol Configuration Register 8 (PCR8)

30.5.2.64.10 Protocol Configuration Register 9 (PCR9)

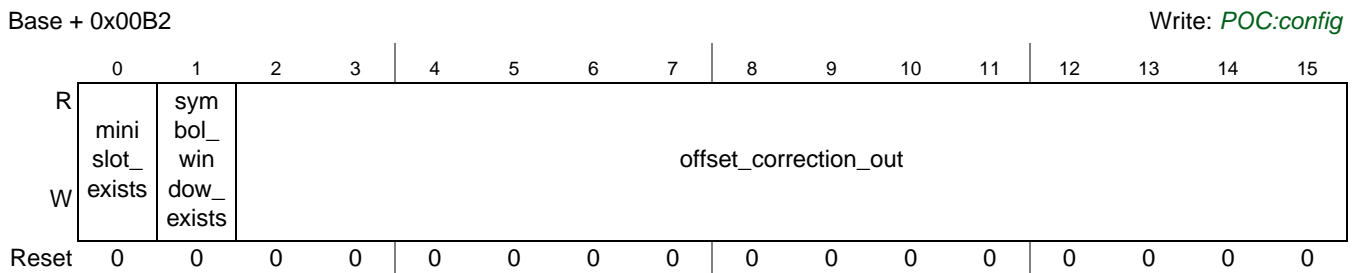


Figure 30-73. Protocol Configuration Register 9 (PCR9)

30.5.2.64.11 Protocol Configuration Register 10 (PCR10)

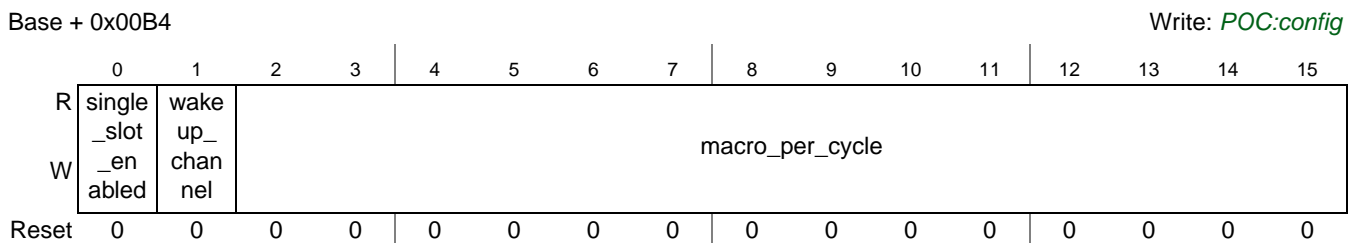


Figure 30-74. Protocol Configuration Register 10 (PCR10)

30.5.2.64.12 Protocol Configuration Register 11 (PCR11)

Base + 0x00B6

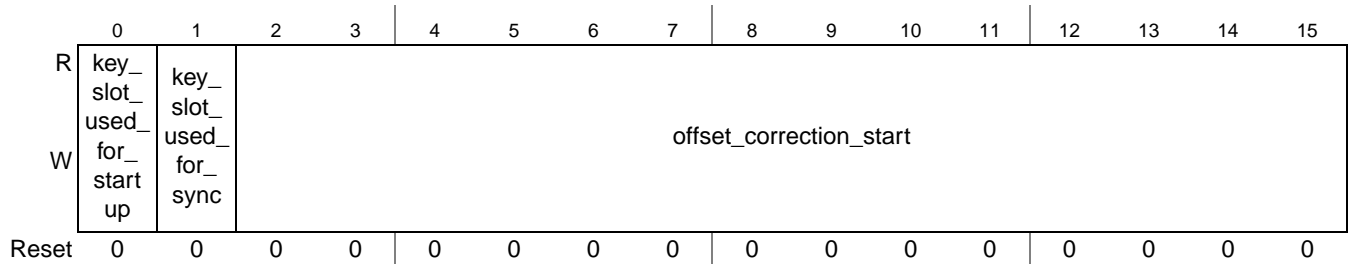
Write: *POC:config*

Figure 30-75. Protocol Configuration Register 11 (PCR11)

30.5.2.64.13 Protocol Configuration Register 12 (PCR12)

Base + 0x00B8

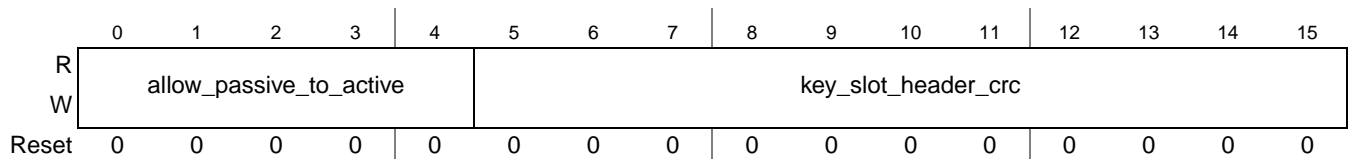
Write: *POC:config*

Figure 30-76. Protocol Configuration Register 12 (PCR12)

30.5.2.64.14 Protocol Configuration Register 13 (PCR13)

Base + 0x00BA

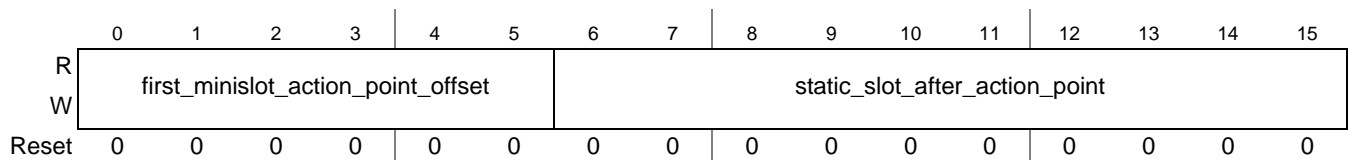
Write: *POC:config*

Figure 30-77. Protocol Configuration Register 13 (PCR13)

30.5.2.64.15 Protocol Configuration Register 14 (PCR14)

Base + 0x00BC

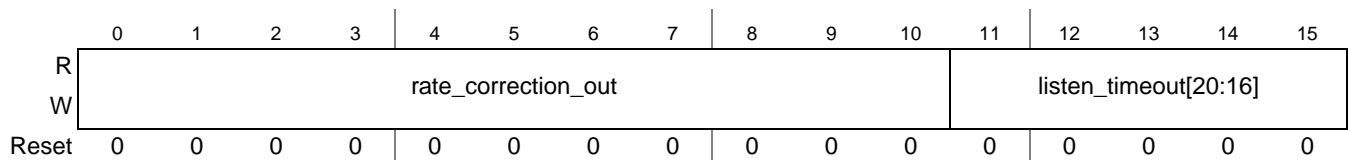
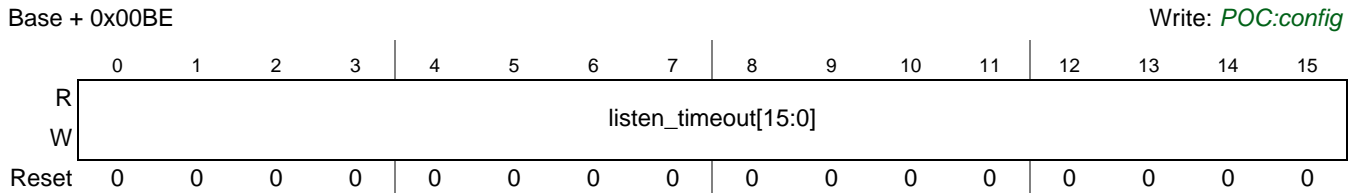
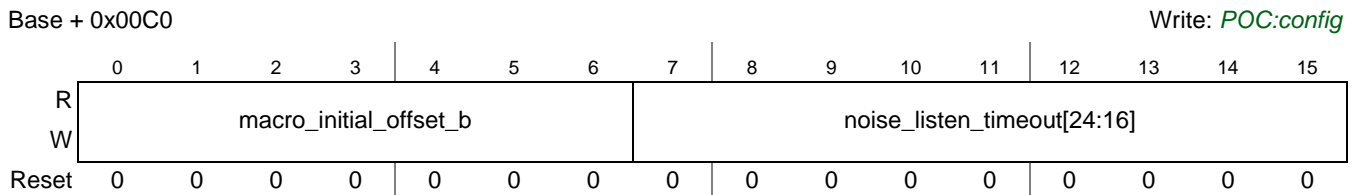
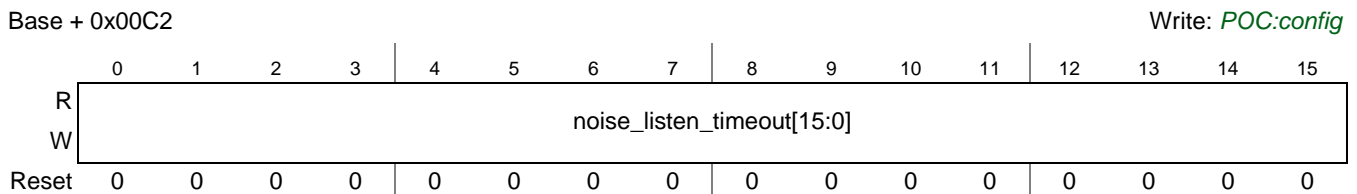
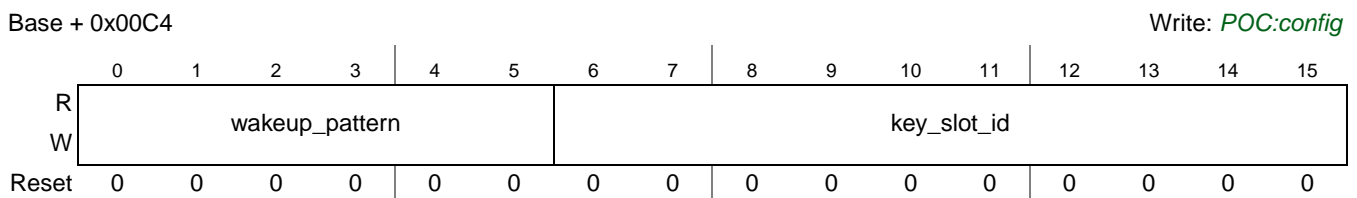
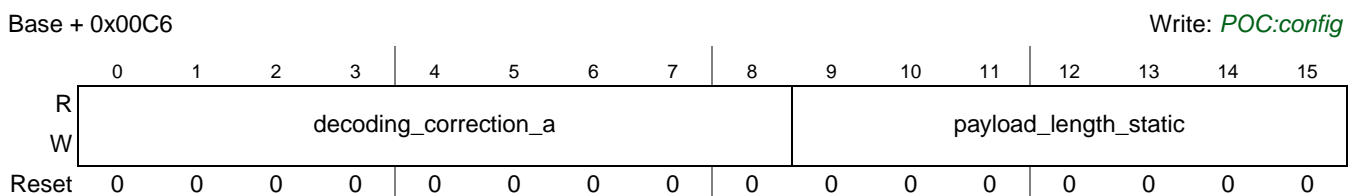
Write: *POC:config*

Figure 30-78. Protocol Configuration Register 14 (PCR14)

30.5.2.64.16 Protocol Configuration Register 15 (PCR15)**Figure 30-79. Protocol Configuration Register 15 (PCR15)****30.5.2.64.17 Protocol Configuration Register 16 (PCR16)****Figure 30-80. Protocol Configuration Register 16 (PCR16)****30.5.2.64.18 Protocol Configuration Register 17 (PCR17)****Figure 30-81. Protocol Configuration Register 17 (PCR17)****30.5.2.64.19 Protocol Configuration Register 18 (PCR18)****Figure 30-82. Protocol Configuration Register 18 (PCR18)****30.5.2.64.20 Protocol Configuration Register 19 (PCR19)****Figure 30-83. Protocol Configuration Register 19 (PCR19)**

30.5.2.64.21 Protocol Configuration Register 20 (PCR20)

Base + 0x00C8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	micro_initial_offset_b								micro_initial_offset_a							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-84. Protocol Configuration Register 20 (PCR20)

30.5.2.64.22 Protocol Configuration Register 21 (PCR21)

Base + 0x00CA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	extern_rate_correction				latest_tx											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-85. Protocol Configuration Register 21 (PCR21)

30.5.2.64.23 Protocol Configuration Register 22 (PCR22)

Base + 0x00CC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R*	comp_accepted_startup_range_a										micro_per_cycle[19:16]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-86. Protocol Configuration Register 22 (PCR22)

30.5.2.64.24 Protocol Configuration Register 23 (PCR23)

Base + 0x00CE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	micro_per_cycle[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-87. Protocol Configuration Register 23 (PCR23)

30.5.2.64.25 Protocol Configuration Register 24 (PCR24)

Base + 0x00D0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	cluster_drift_damping				max_payload_length_dynamic								micro_per_cycle_min [19:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-88. Protocol Configuration Register 24 (PCR24)

30.5.2.64.26 Protocol Configuration Register 25 (PCR25)

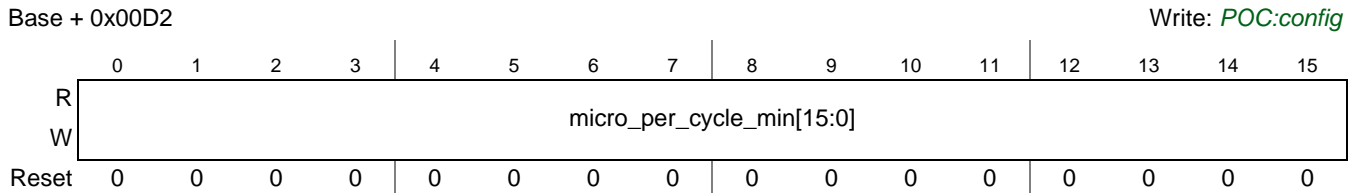


Figure 30-89. Protocol Configuration Register 25 (PCR25)

30.5.2.64.27 Protocol Configuration Register 26 (PCR26)

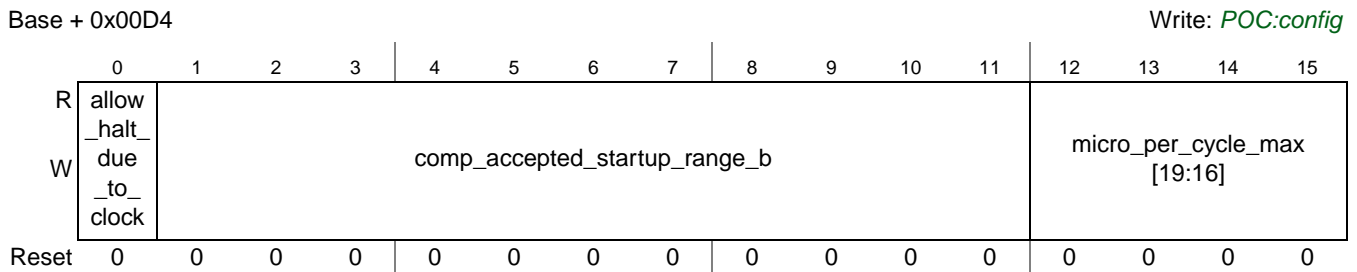


Figure 30-90. Protocol Configuration Register 26 (PCR26)

30.5.2.64.28 Protocol Configuration Register 27 (PCR27)

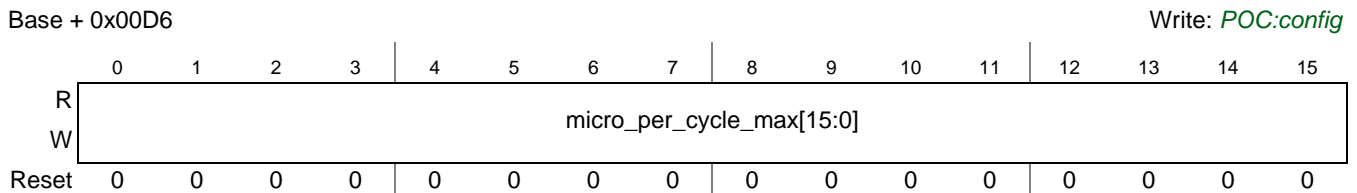


Figure 30-91. Protocol Configuration Register 27 (PCR27)

30.5.2.64.29 Protocol Configuration Register 28 (PCR28)

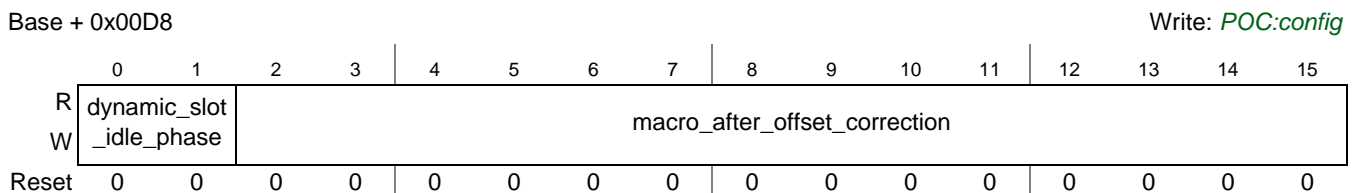


Figure 30-92. Protocol Configuration Register 28 (PCR28)

30.5.2.64.30 Protocol Configuration Register 29 (PCR29)

Base + 0x00DA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	extern_offset_correction			minislots_max												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-93. Protocol Configuration Register 29 (PCR29)

30.5.2.64.31 Protocol Configuration Register 30 (PCR30)

Base + 0x00DC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	sync_node_max			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-94. Protocol Configuration Register 30 (PCR30)

30.5.2.65 Message Buffer Configuration, Control, Status Registers (MBCCSRn)

Base + 0x0100 (MBCCSR0) Write: MCM, MBT, MTD: *POC:config* or MB_DIS
 Base + 0x0108 (MBCCSR1) CMT: MB_LCK
 ... EDT, LCKT, MBIE, MBIF: Normal Mode
 Base + 0x02F8 (MBCCSR63)

Additional Reset: CMT, DUP, DVAL, MBIF: Message Buffer Disable

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MCM	MBT	MTD	CMT	0	0	MBIE	0	0	0	DUP	DVAL	EDS	LCKS	MBIF
W					rwm	EDT	LCKT									w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-95. Message Buffer Configuration, Control, Status Registers (MBCCSRn)

The content of these registers comprises message buffer configuration data, message buffer control data, message buffer status information, and message buffer interrupt flags.

Table 30-77. MBCCSRn Field Descriptions (Sheet 1 of 3)

Field	Description
Message Buffer Configuration	
MCM	Message Buffer Commit Mode. This bit applies only to double buffered transmit message buffers and defines the commit mode. 0 Streaming commit mode 1 Immediate commit mode
MBT	Message Buffer Type. This bit applies only to transmit message buffers and defines the buffering type. 0 Single buffered transmit message buffer 1 Double buffered transmit message buffer

Table 30-77. MBCCSRn Field Descriptions (Sheet 2 of 3)

Field	Description
MTD	Message Buffer Transfer Direction. This bit defines the transfer direction of the message buffer. 0 Receive message buffer 1 Transmit message buffer
Message Buffer Control	
CMT	Commit for Transmission. This bit applies only to transmit message buffers and indicates whether the message buffer contains valid data that are ready for transmission. Both the application and the FlexRay block can modify this bit. <ul style="list-style-type: none"> Application: The application sets this bit to indicate that the transmit message buffer contains valid data ready for transmission. The application clears this bit to indicate that the message buffer data are no longer valid for transmission. FlexRay block: The FlexRay block clears this bit when the message buffer data are no longer valid for transmission. 0 Message buffer does not contain valid data. 1 Message buffer contains valid data.
EDT	Enable/Disable Trigger. This trigger bit is used to enable and disable a message buffer. The message buffer enable is triggered when the application writes 1 to this bit and the message buffer is disabled, i.e. the EDS status bit is 0. The message buffer disable is triggered when the application writes 1 to this bit and the message buffer is enabled, i.e. the EDS status bit is 1. 0 No effect 1 message buffer enable/disable triggered Note: If the application writes 1 to this bit, the write access to all other bits is ignored.
LCKT	Lock/Unlock Trigger. This trigger bit is used to lock and unlock a message buffer. The message buffer lock is triggered when the application writes 1 to this bit and the message buffer is not locked, i.e. the LCKS status bit is 0. The message buffer unlock is triggered when the application writes 1 to this bit and the message buffer is locked, i.e. the LCKS status bit is 1. 0 No effect 1 Trigger message buffer lock/unlock Note: If the application writes 1 to this bit and 0 to the EDT bit, the write access to all other bits is ignored.
MBIE	Message Buffer Interrupt Enable. This control bit defines whether the message buffer will generate an interrupt request when its MBIF flag is set. 0 Interrupt request generation disabled 1 Interrupt request generation enabled
Message Buffer Status	
DUP	Data Updated. This status bit applies only to receive message buffers. It is always 0 for transmit message buffers. This bit provides information whether the frame header in the message buffer header field and the message buffer data field were updated. See Section 30.6.6.3.3, "Message Buffer Status Update" for a detailed description of the update conditions. 0 Frame Header and Message buffer data field not updated. 1 Frame Header and Message buffer data field updated.

Table 30-77. MBCCSRn Field Descriptions (Sheet 3 of 3)

Field	Description
DVAL	Data Valid. The semantic of this status bit depends on the message buffer type and transfer direction. <ul style="list-style-type: none"> <i>Receive Message Buffer</i>: Indicates whether the message buffer data field contains valid frame data. See Section 30.6.6.3.3, “Message Buffer Status Update” for a detailed update description of the update conditions. <ul style="list-style-type: none"> 0 message buffer data field contains no valid frame data 1 message buffer data field contains valid frame data <i>Single Transmit Message Buffer</i>: Indicates whether the message is transferred again due to the state transmission mode of the message buffer. <ul style="list-style-type: none"> 0 Message transferred for the first time. 1 Message will be transferred again. <i>Double Transmit Message Buffer</i>: For the commit side it is always 0. For the transmit side it indicates whether the message is transferred again due to the state transmission mode of the message buffer. <ul style="list-style-type: none"> 0 Message transferred for the first time. 1 Message will be transferred again.
EDS	Enable/Disable Status. This status bit indicates whether the message buffer is enabled or disabled. <ul style="list-style-type: none"> 0 Message buffer is disabled. 1 Message buffer is enabled.
LCKS	Lock Status. This status bit indicates the current lock status of the message buffer. <ul style="list-style-type: none"> 0 Message buffer is not locked by the application. 1 Message buffer is locked by the application.
MBIF	Message Buffer Interrupt Flag. The semantic of this flag depends on the message buffer transfer direction. <ul style="list-style-type: none"> <i>Receive Message Buffer</i>: This flag is set when the slot status in the message buffer header field was updated and this slot was not an empty dynamic slot. See Section 30.6.6.3.3, “Message Buffer Status Update” for a detailed description of the update conditions. <ul style="list-style-type: none"> 0 slot status not updated 1 slot status updated and slot was not an empty dynamic slot <i>Transmit Message Buffer</i>: This flag is set when the slot status in the message buffer header field was updated. Additionally this flag is set immediately when a transmit message buffer was enabled. <ul style="list-style-type: none"> 0 slot status not updated 1 slot status updated / message buffer just enabled

30.5.2.66 Message Buffer Cycle Counter Filter Registers (MBCCFRn)

Base + 0x0102 (MBCCFR0)

Write: *POC:config* or MB_DIS

Base + 0x010A (MBCCFR1)

...

Base + 0x02FA (MBCCFR63)

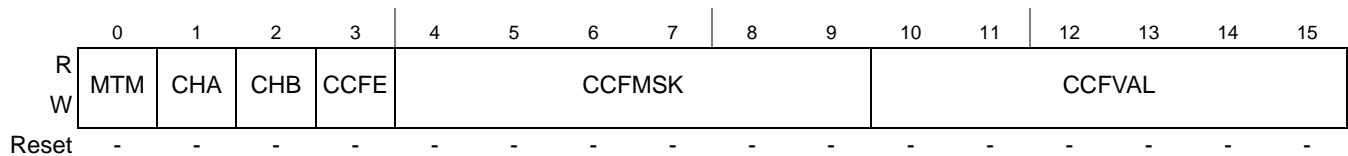


Figure 30-96. Message Buffer Cycle Counter Filter Registers (MBCCFRn)

This register contains message buffer configuration data for the transmission mode, the channel assignment, and for the cycle counter filtering. For detailed information on cycle counter filtering, refer to [Section 30.6.7.1, “Message Buffer Cycle Counter Filtering.”](#)

Table 30-78. MBCCFRn Field Descriptions

Field	Description
MTM	Message Buffer Transmission Mode. This control bit applies only to transmit message buffers and defines the transmission mode. 0 Event transmission mode 1 State transmission mode
CHA CHB	Channel Assignment. These control bits define the channel assignment and control the receive and transmit behavior of the message buffer according to Table 30-79 .
CCFE	Cycle Counter Filtering Enable. This control bit is used to enable and disable the cycle counter filtering. 0 Cycle counter filtering disabled 1 Cycle counter filtering enabled
CCFMSK	Cycle Counter Filtering Mask. This field defines the filter mask for the cycle counter filtering.
CCFVAL	Cycle Counter Filtering Value. This field defines the filter value for the cycle counter filtering.

Table 30-79. Channel Assignment Description

CHA	CHB	Transmit Message Buffer		Receive Message Buffer	
		static segment	dynamic segment	static segment	dynamic segment
1	1	transmit on both channel A and channel B	transmit on channel A only	store first valid frame received on either channel A or channel B	store first valid frame received on channel A, ignore channel B
0	1	transmit on channel B	transmit on channel B	store first valid frame received on channel B	store first valid frame received on channel B
1	0	transmit on channel A	transmit on channel A	store first valid frame received on channel A	store first valid frame received on channel A
0	0	no frame transmission	no frame transmission	no frame stored	no frame stored

NOTE

If at least one message buffer assigned to a certain slot is assigned to both channels, then all message buffers assigned to this slot have to be assigned to both channels. Otherwise, the message buffer configuration is illegal and the result of the message buffer search is not defined.

30.5.2.67 Message Buffer Frame ID Registers (MBFIDRn)

Base + 0x0104 (MBFIDR0)

Base + 0x010C (MBFIDR1)

...

Base + 0x02FC (MBFIDR63)

Write: *POC:config* or MB_DIS

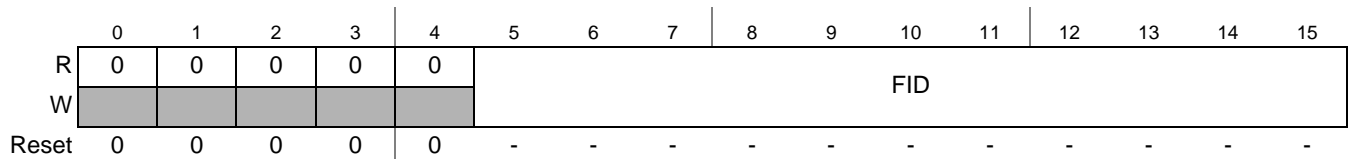


Figure 30-97. Message Buffer Frame ID Registers (MBFIDRn)

Table 30-80. MBFIDRn Field Descriptions

Field	Description
FID	<p>Frame ID — The semantic of this field depends on the message buffer transfer type.</p> <ul style="list-style-type: none"> <i>Receive Message Buffer</i>: This field is used as a filter value to determine if the message buffer is used for reception of a message received in a slot with the slot ID equal to FID. <i>Transmit Message Buffer</i>: This field is used to determine the slot in which the message in this message buffer should be transmitted.

30.5.2.68 Message Buffer Index Registers (MBIDXRn)

Base + 0x0106 (MBIDXR0)

Write: *POC:config* or MB_DIS

Base + 0x010E (MBIDXR1)

...

Base + 0x02FE (MBIDXR63)

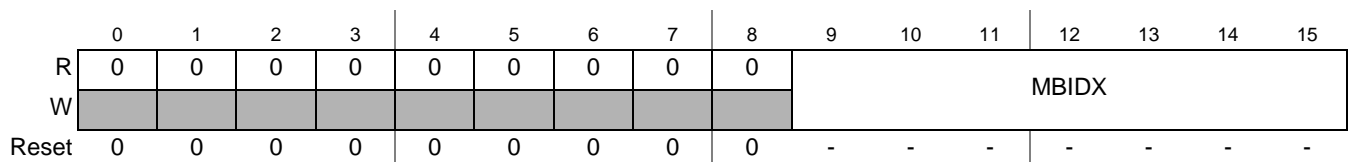


Figure 30-98. Message Buffer Index Registers (MBIDXRn)

Table 30-81. MBIDXRn Field Descriptions

Field	Description
MBIDX	<p>Message Buffer Index. This field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer.</p> <p>The application writes the index of the initially associated message buffer header field into this register. The FlexRay block updates this register after frame reception or transmission.</p>

30.6 Functional Description

This section provides a detailed description of the functionality implemented in the FlexRay block.

30.6.1 Message Buffer Concept

The FlexRay block uses a data structure called *message buffer* to store frame data, configuration, control, and status data. Each message buffer consists of two parts, the *message buffer control data* and the *physical message buffer*. The message buffer control data are located in dedicated registers. The structure of the message buffer control data depends on the message buffer type and is described in [Section 30.6.3, “Message Buffer Types”](#). The physical message buffer is located in the FRM and is described in [Section 30.6.2, “Physical Message Buffer”](#).

30.6.2 Physical Message Buffer

All FlexRay messages and related frame and slot status information of received frames and of frames to be transmitted to the FlexRay bus are stored in data structures called *physical message buffers*. The physical message buffers are located in the FRM. The structure of a physical message buffer is depicted in [Figure 30-99](#).

A physical message buffer consists of two fields, the *message buffer header field* and the *message buffer data field*. The message buffer header field contains the *frame header*, the *data field offset*, and the *slot status*. The message buffer data field contains the *frame data*.

The connection between the two fields is established by the *data field offset*.

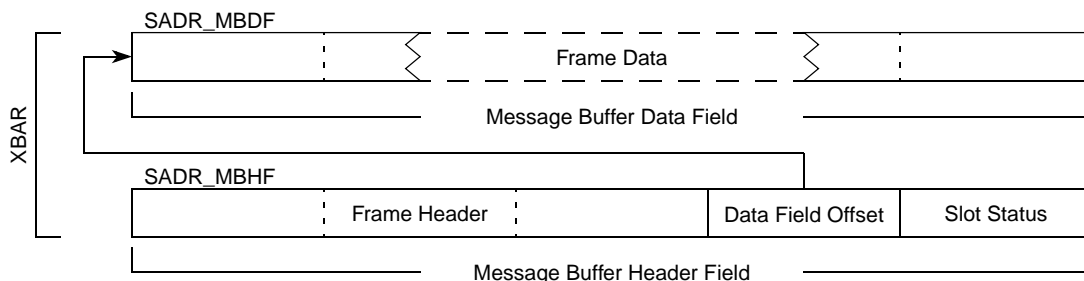


Figure 30-99. Physical Message Buffer Structure

30.6.2.1 Message Buffer Header Field

The message buffer header field is a contiguous region in the FRM and occupies ten bytes. It contains the frame header, the data field offset, and the slot status. Its structure is shown in [Figure 30-99](#). The physical start address *SADR_MBHF* of the message buffer header field must be 16-bit aligned.

30.6.2.1.1 Frame Header

The frame header occupies the first six bytes in the message buffer header field. It contains all FlexRay frame header related information according to the *FlexRay Communications System Protocol*

Specification, Version 2.1 Rev A. A detailed description of the usage and the content of the frame header is provided in [Section 30.6.5.2.1, “Frame Header Section Description”](#).

30.6.2.1.2 Data Field Offset

The data field offset follows the frame header in the message buffer data field and occupies two bytes. It contains the offset of the corresponding message buffer data field with respect to the FlexRay block FRM base address as provided by SYS_MEM_BASE_ADDR field in the [System Memory Base Address High Register \(SYMBADHR\)](#) and [System Memory Base Address Low Register \(SYMBADLR\)](#). The data field offset is used to determine the start address *SADR_MBDF* of the corresponding message buffer data field in the FRM according to [Equation 30-2](#).

$$\text{SADR_MBDF} = [\text{Data Field Offset}] + \text{SYS_MEM_BASE_ADDR} \quad \text{Eqn. 30-2}$$

30.6.2.1.3 Slot Status

The slot status occupies the last two bytes of the message buffer header field. It provides the slot and frame status related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the content and usage of the slot status is provided in [Section 30.6.5.2.3, “Slot Status Description”](#).

30.6.2.2 Message Buffer Data Field

The message buffer data field is a contiguous area of 2-byte entities. This field contains the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum length of this field depends on the specific message buffer configuration and is specified in the message buffer descriptions given in [Section 30.6.3, “Message Buffer Types”](#).

30.6.3 Message Buffer Types

The FlexRay block provides three different types of message buffers.

- Individual Message Buffers
- Receive Shadow Buffers
- Receive FIFO Buffers

For each message buffer type the structure of the physical message buffer is identical. The message buffer types differ only in the structure and content of message buffer control data, which control the related physical message buffer. The message buffer control data are described in the following sections.

30.6.3.1 Individual Message Buffers

The individual message buffers are used for all types of frame transmission and for dedicated frame reception based on individual filter settings for each message buffer. The FlexRay block supports three types of individual message buffers, which are described in [Section 30.6.6, “Individual Message Buffer Functional Description”](#).

Each individual message buffer consists of two parts, the physical message buffer, which is located in the FRM, and the message buffer control data, which are located in dedicated registers. The structure of an individual message buffer is given in Figure 30-100.

Each individual message buffer has a message buffer number n assigned, which determines the set of message buffer control registers associated to this individual message buffer. The individual message buffer with message buffer number n is controlled by the registers MBCCSR $_n$, MBCCFR $_n$, MBFIDR $_n$, and MBIDXR $_n$.

The connection between the message buffer control registers and the physical message buffer is established by the message buffer index field MBIDX in the Message Buffer Index Registers (MBIDXR $_n$). The start address SADR_MBHF of the related message buffer header field in the FRM is determined according to Equation 30-3.

$$\text{SADR_MBHF} = (\text{MBIDXR}_n[\text{MBIDX}] * 10) + \text{SYS_MEM_BASE_ADDR} \quad \text{Eqn. 30-3}$$

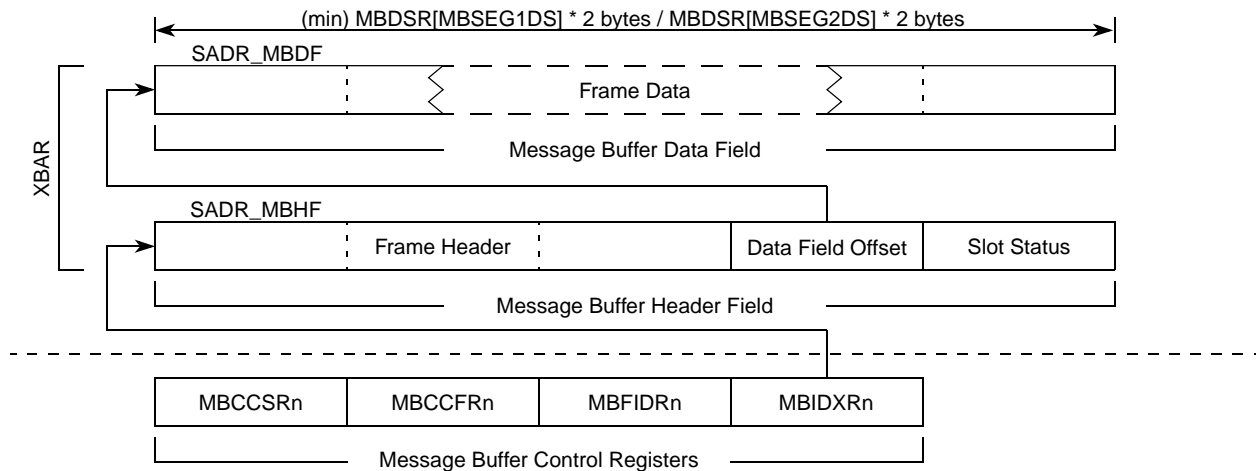


Figure 30-100. Individual Message Buffer Structure

30.6.3.1.1 Individual Message Buffer Segments

The set of the individual message buffers can be split up into two message buffer segments using the Message Buffer Segment Size and Utilization Register (MBSSUTR). All individual message buffers with a message buffer number $n \leq \text{MBSSUTR.LAST_MB_SEG1}$ belong to the first message buffer segment. All individual message buffers with a message buffer number $n > \text{MBSSUTR.LAST_MB_SEG1}$ belong to the second message buffer segment. The following rules apply to the length of the message buffer data field:

- all physical message buffers associated to individual message buffers that belong to the same message buffer segment must have message buffer data fields of the same length
- the minimum length of the message buffer data field for individual message buffers in the first message buffer segment is $2 * \text{MBDSR.MBSEG1DS}$ bytes
- the minimum length of the message buffer data field for individual message buffers assigned to the second segment is $2 * \text{MBDSR.MBSEG2DS}$ bytes.

30.6.3.2 Receive Shadow Buffers

The receive shadow buffers are required for the frame reception process for individual message buffers. The FlexRay block provides four receive shadow buffers, one receive shadow buffer per channel and per message buffer segment.

Each receive shadow buffer consists of two parts, the physical message buffer located in the FRM and the receive shadow buffer control registers located in dedicated registers. The structure of a receive shadow buffer is shown in Figure 30-101. The four internal shadow buffer control registers can be accessed by the Receive Shadow Buffer Index Register (RSBIR).

The connection between the receive shadow buffer control register and the physical message buffer for the selected receive shadow buffer is established by the receive shadow buffer index field RSBIDX in the Receive Shadow Buffer Index Register (RSBIR). The start address SADR_MBHF of the related message buffer header field in the FRM is determined according to Equation 30-4.

$$\text{SADR_MBHF} = (\text{RSBIR}[\text{RSBIDX}] * 10) + \text{SYS_MEM_BASE_ADDR} \quad \text{Eqn. 30-4}$$

The length required for the message buffer data field depends on the message buffer segment that the receive shadow buffer is assigned to. For the receive shadow buffers assigned to the first message buffer segment, the length must be the same as for the individual message buffers assigned to the first message buffer segment. For the receive shadow buffers assigned to the second message buffer segment, the length must be the same as for the individual message buffers assigned to the second message buffer segment. The receive shadow buffer assignment is described in Receive Shadow Buffer Index Register (RSBIR).

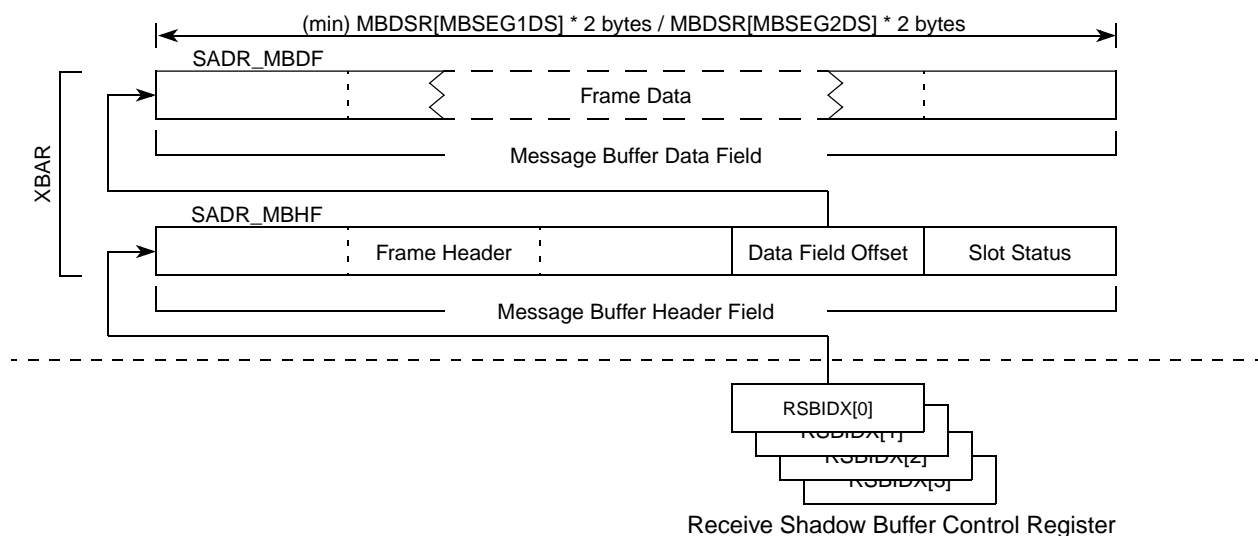


Figure 30-101. Receive Shadow Buffer Structure

30.6.3.3 Receive FIFO

The receive FIFO implements a frame reception system based on the FIFO concept. The FlexRay block provides two independent receive FIFOs, one per channel.

A receive FIFO consists of a set of physical message buffers in the FRM and a set of receive FIFO control registers located in dedicated registers. The structure of a receive FIFO is given in Figure 30-102.

The connection between the receive FIFO control registers and the set of physical message buffers is established by the start index field SIDX in the [Receive FIFO Start Index Register \(RFSIR\)](#), the FIFO depth field FIFO_DEPTH in the [Receive FIFO Depth and Size Register \(RFDSR\)](#), and the read index field RDIDX in the [Receive FIFO A Read Index Register \(RFARIR\)](#) / [Receive FIFO B Read Index Register \(RFBRIR\)](#). The start address SADR_MBHF_1 of the first message buffer header field that belongs to the receive FIFO in the FRM is determined according to [Equation 30-5](#).

$$SADR_MBHF[1] = (RFSIR[SIDX] * 10) + SYS_MEM_BASE_ADDR \quad \text{Eqn. 30-5}$$

The start address SADR_MBHF[n] of the last message buffer header field that belongs to the receive FIFO in the FRM is determined according to [Equation 30-6](#).

$$SADR_MBHF[n] = ((RFSIR[SIDX] + RFDSR[FIFO_DEPTH]) * 10) + SYS_MEM_BASE_ADDR \quad \text{Eqn. 30-6}$$

NOTE

All message buffer header fields assigned to a receive FIFO must be a contiguous region.

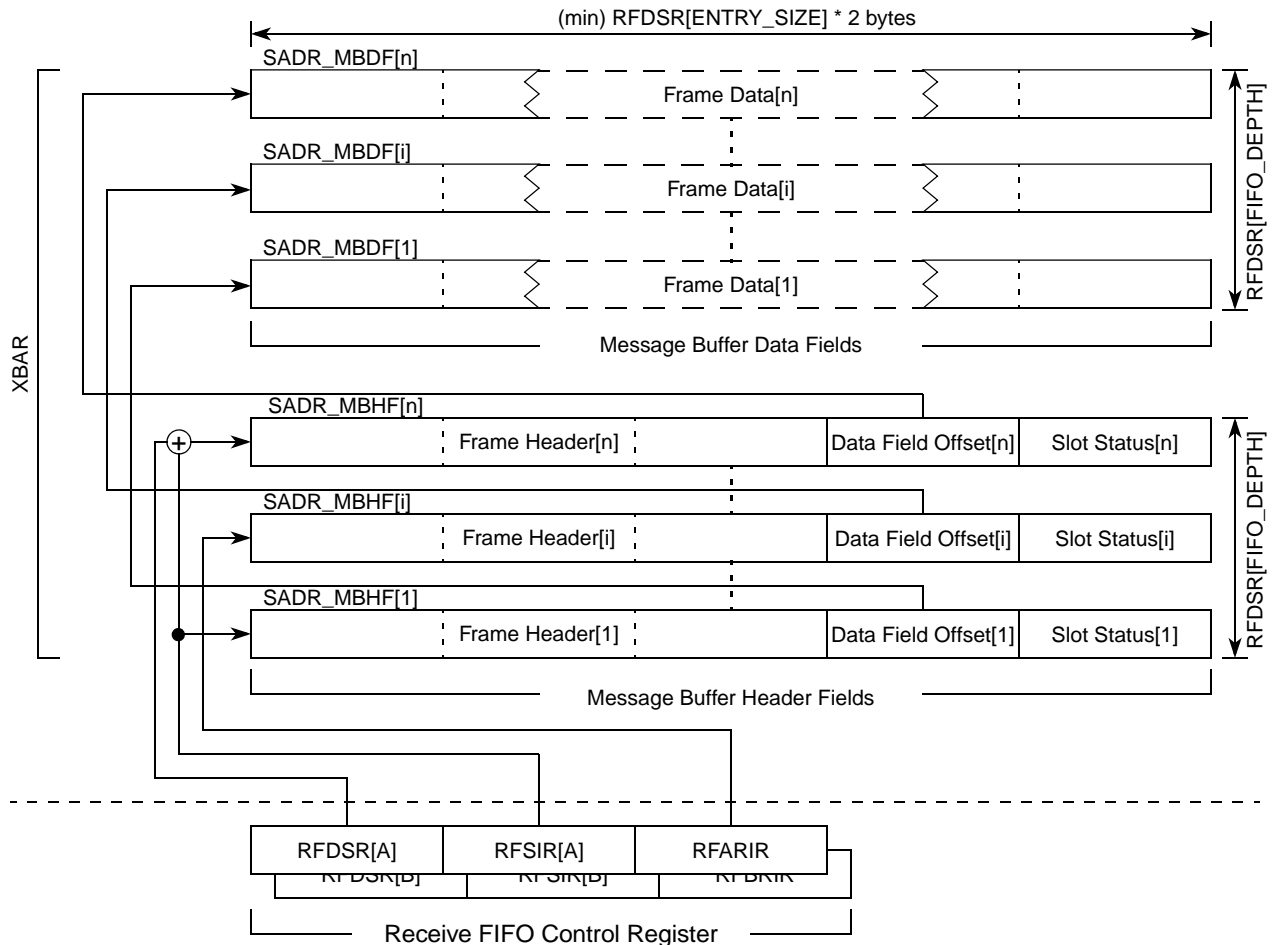


Figure 30-102. Receive FIFO Structure

30.6.3.4 Message Buffer Configuration and Control Data

This section describes the configuration and control data for each message buffer type.

30.6.3.4.1 Individual Message Buffer Configuration Data

Before an individual message buffer can be used for transmission or reception, it must be configured. There is a set of common configuration parameters that applies to all individual message buffers and a set of configuration parameters that applies to each message buffer individually.

Common Configuration Data

The set of common configuration data for individual message buffers is located in the following registers.

- [Message Buffer Data Size Register \(MBDSR\)](#)
The MBSEG2DS and MBSEG1DS fields define the minimum length of the message buffer data field with respect to the message buffer segment.
- [Message Buffer Segment Size and Utilization Register \(MBSSUTR\)](#)
The LAST_MB_SEG1 and LAST_MB_UTIL fields define the segmentation of the individual message buffers and the number of individual message buffers that are used. For more details, see [Section 30.6.3.1.1, “Individual Message Buffer Segments”](#)

Specific Configuration Data

The set of message buffer specific configuration data for individual message buffers is located in the following registers.

- [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#)
The MCM, MBT, MTD bits configure the message buffer type.
- [Message Buffer Cycle Counter Filter Registers \(MBCCFRn\)](#)
The MTM, CHA, CHB bits configure the transmission mode and the channel assignment. The CCFE, CCFMSK, and CCFVAL bits and fields configure the cycle counter filter.
- [Message Buffer Frame ID Registers \(MBFIDRn\)](#)
For a transmit message buffer, the FID field is used to determine the slot in which the message in this message buffer will be transmitted.
- [Message Buffer Index Registers \(MBIDXRn\)](#)
This MBIDX field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer.

30.6.3.5 Individual Message Buffer Control Data

During normal operation, each individual message buffer can be controlled by the control and trigger bits CMT, LCKT, EDT, and MBIE in the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#).

30.6.3.6 Receive Shadow Buffer Configuration Data

Before frame reception into the individual message buffers can be performed, the receive shadow buffers must be configured. The configuration data are provided by the [Receive Shadow Buffer Index Register \(RSBIR\)](#). For each receive shadow buffer, the application provides the message buffer header index. When the protocol is in the *POC:normal active* or *POC:normal passive* state, the receive shadow buffers are under full FlexRay block control.

30.6.3.7 Receive FIFO Control and Configuration Data

This section describes the configuration and control data for the two receive FIFOs.

30.6.3.7.1 Receive FIFO Configuration Data

The FlexRay block provides two completely independent receive FIFOs, one per channel. Each FIFO has its own set of configuration data. The configuration data are located in the following registers:

- [Receive FIFO Start Index Register \(RFSIR\)](#)
- [Receive FIFO Depth and Size Register \(RFDSR\)](#)
- [Receive FIFO Message ID Acceptance Filter Value Register \(RFMIDAFVR\)](#)
- [Receive FIFO Message ID Acceptance Filter Mask Register \(RFMIAFMR\)](#)
- [Receive FIFO Frame ID Rejection Filter Value Register \(RFFIDRFVR\)](#)
- [Receive FIFO Frame ID Rejection Filter Mask Register \(RFFIDRFMR\)](#)
- [Receive FIFO Range Filter Configuration Register \(RFRFCFR\)](#)

30.6.3.7.2 Receive FIFO Control Data

The application can access the receive FIFO at any time using the values provided in the [Receive FIFO A Read Index Register \(RFARIR\)](#) and [Receive FIFO B Read Index Register \(RFBRIR\)](#). To update the [Receive FIFO A Read Index Register \(RFARIR\)](#), the application must write 1 to the FIFO A Not Empty Interrupt Flag FNEAIF in the [Global Interrupt Flag and Enable Register \(GIFER\)](#). To update the [Receive FIFO B Read Index Register \(RFBRIR\)](#) the application must write 1 to the FIFO B Not Empty Interrupt Flag FNEBIF in the [Global Interrupt Flag and Enable Register \(GIFER\)](#). As long as the FIFO is not empty, each update increments the related read index. If the read index has reached the last FIFO entry, it wraps back to the FIFO start index.

30.6.4 FlexRay Memory Layout

The FlexRay block supports a wide range of possible layouts for the FRM. [Figure 30-103](#) shows an example layout. The following set of rules applies to the layout of the FRM:

- The FRM is a contiguous region.
- The FRM size is maximum 64 Kbytes.
- The FRM starts at a 16 byte boundary.

The FRM contains three areas: the *message buffer header area*, the *message buffer data area*, and the *sync frame table area*. The areas are described in this section.

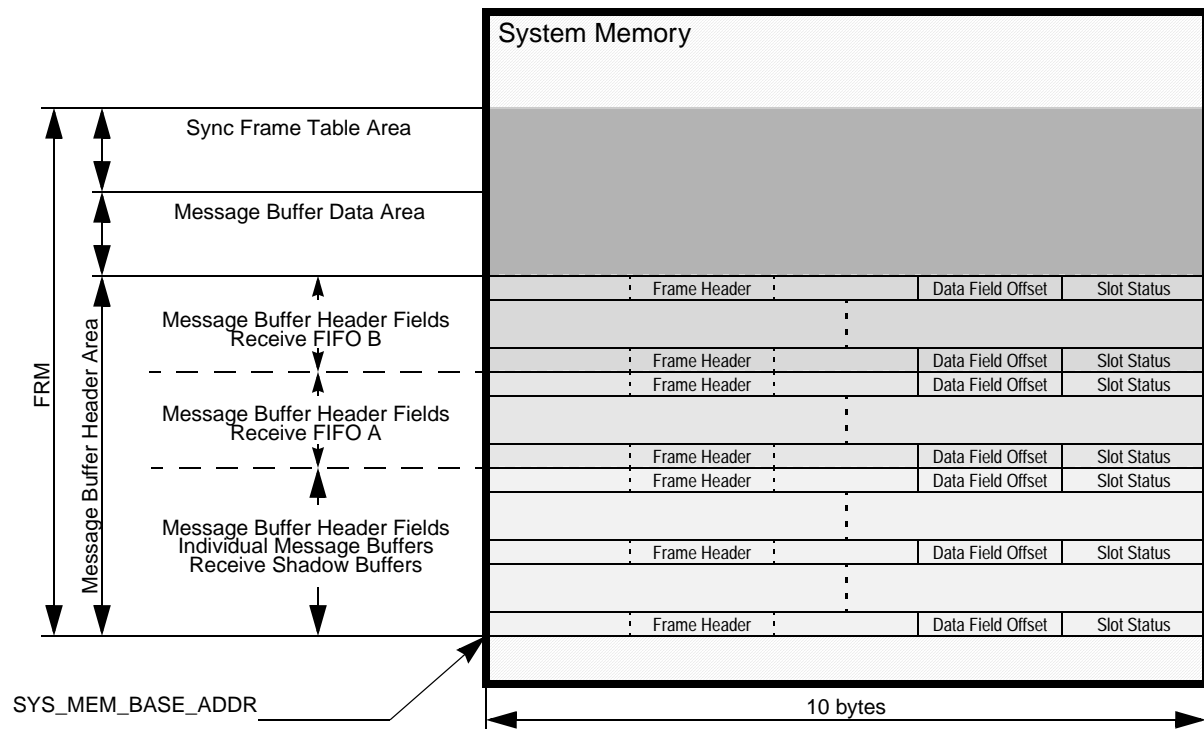


Figure 30-103. Example of FRM Layout

30.6.4.1 Message Buffer Header Area

The message buffer header area contains all message buffer header fields of the physical message buffers for all message buffer types. The following rules apply to the message buffer header fields for the three type of message buffers.

1. The start address `SADR_MBHF` of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill [Equation 30-7](#).

$$\text{SADR_MBHF} = (i * 10) + \text{SYS_MEM_BASE_ADDR}; (0 \leq i < 128) \quad \text{Eqn. 30-7}$$

2. The start address `SADR_MBHF` of each message buffer header field for the *receive FIFO* must fulfill [Equation 30-8](#).

$$\text{SADR_MBHF} = (i * 10) + \text{SYS_MEM_BASE_ADDR}; (0 \leq i < 1024) \quad \text{Eqn. 30-8}$$

3. The message buffer header fields for a receive FIFO have to be a contiguous area.

30.6.4.2 Message Buffer Data Area

The message buffer data area contains all the message buffer data fields of the physical message buffers. Each message buffer data field must start at a 16-bit boundary.

30.6.4.3 Sync Frame Table Area

The sync frame table area is used to provide a copy of the internal sync frame tables for application access. Refer to [Section 30.6.12, “Sync Frame ID and Sync Frame Deviation Tables”](#) for the description of the sync frame table area.

30.6.5 Physical Message Buffer Description

This section provides a detailed description of the usage and the content of the two parts of a physical message buffer, the message buffer header field and the message buffer data field.

30.6.5.1 Message Buffer Protection and Data Consistency

The physical message buffers are located in the FRM. The FlexRay block provides no means to protect the FRM from uncontrolled or illegal host or other client write access. To ensure data consistency of the physical message buffers, the application must follow the write access scheme that is given in the description of each of the physical message buffer fields.

30.6.5.2 Message Buffer Header Field Description

This section provides a detailed description of the usage and content of the message buffer header field. A description of the structure of the message buffer header fields is given in [Section 30.6.2.1, “Message Buffer Header Field”](#). Each message buffer header field consists of three sections: the frame header section, the data field offset, and the slot status section. For a detailed description of the Data Field Offset, see [Section 30.6.2.1.2, “Data Field Offset”](#).

30.6.5.2.1 Frame Header Section Description

Frame Header Section Content

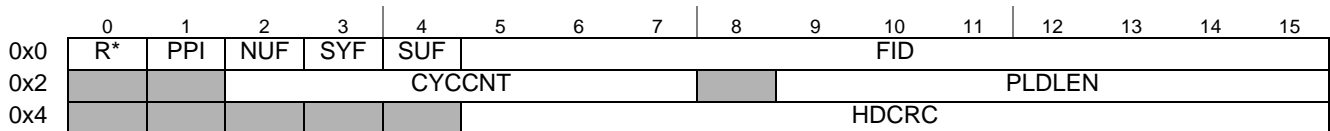
The semantic and content of the frame header section depends on the message buffer type.

For individual receive message buffers and receive FIFOs, the frame header receives the frame header data of the *first valid frame* received on the assigned channels. If a receive message buffer is assigned to both channels, the first valid frame received on either channel A or channel B is stored.

For receive shadow buffers, the frame header receives the frame header data of the current frame received regardless of whether the frame is valid or not.

For single and double transmit message buffers, the application writes the frame header of the frame to be transmitted into this location. The frame header will be read out when the frame is transferred to the FlexRay bus.

The structure of the frame header in the message buffer header field is given in [Figure 30-104](#). A detailed description of the frame header fields is given in [Table 30-83](#).



 = not used for TX message buffers, not updated for RX message buffers

Figure 30-104. Frame Header Structure

Frame Header Section Access

The frame header is located in the FRM. To ensure data consistency, the application must follow the write access scheme described below.

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the frame header field.

For transmit message buffers, the application must follow the write access restrictions given in [Table 30-82](#). This table shows the condition under which the application can write to the frame header entries. In general, the application can modify all frame header entries when the protocol is in the *POC:config* state or when the message buffer is disabled. For message buffers assigned to the dynamic segment, the application can modify all frame header entries except the frame ID when the message buffer is locked.

Table 30-82. Frame Header Write Access Constraints

Field	TX					
	Single Buffered		Double Buffered			
	Static Segment	Dynamic Segment	Static Segment		Dynamic Segment	
			Commit Side	Transmit Side	Commit Side	Transmit Side
FID	<i>POC:config</i> or MB_DIS					
R*, PPI NUF, SYF SUF CYCCNT PLDLEN HDCRD	<i>POC:config</i> or MB_DIS	<i>POC:config</i> or MB_DIS or MB_LCK	<i>POC:config</i> or MB_DIS		<i>POC:config</i> or MB_DIS or MB_LCK	<i>POC:config</i> or MB_DIS

The frame header entries NUF, SYF, SUF, and CYCCNT are not used for frame transmission. These values are generated internally before frame transmission depending on the current transmission state and configuration.

For transmit message buffers assigned to the *static* segment, the PLDLEN value must be equal to the value of the payload_length_static field in the [Protocol Configuration Register 19 \(PCR19\)](#). If this is not fulfilled, the static payload length error flag SPL_EF in the [CHI Error Flag Register \(CHIERFR\)](#) is set when the message buffer is under transmission. The PE generates a syntactically and semantically correct frame with payload_length_static payload words and the payload length field in the frame header set to payload_length_static.

For transmit message buffers assigned to the *dynamic* segment, the PLDLEN value must be less than or equal to the value of the `max_payload_length_dynamic` field in the [Protocol Configuration Register 24 \(PCR24\)](#). If this is not fulfilled, the dynamic payload length error flag `DPL_EF` in the [CHI Error Flag Register \(CHIERFR\)](#) is set when the message buffer is under transmission. The PE generates a syntactically and semantically correct dynamic frame with PLDLEN payload words and the payload length field in the frame header set to PLDLEN.

Table 30-83. Frame Header Field Descriptions

Field	Description
R*	Reserved Bit. This bit corresponds to the Reserved bit in the FlexRay frame header. <ul style="list-style-type: none"> For receive and FIFO message buffers, this is a status bit and represents the value of the Reserved bit in the frame received on the FlexRay bus in the corresponding slot. For transmit message buffers, this is a control bit. The FlexRay block transmits this within the frame header. Note: For protocol compliant operation, this control bit must be set to 0 for transmit message buffers.
PPI	Payload Preamble Indicator. This bit corresponds to the Payload Preamble Indicator in the FlexRay frame header. <ul style="list-style-type: none"> For receive and FIFO message buffers, this is a status bit and represents the value of the Payload Preamble Indicator of the first valid frame received on the FlexRay in the slot indicated by the CYCCNT field. For transmit message buffers, this is a control bit. The FlexRay block uses this value to set the Payload Preamble Indicator in the frame header of the frame to transmit. 0 No network management vector or message ID in frame payload data 1 Static Segment: Frame payload data contains network management vector Dynamic Segment: Frame payload data contains message ID
NUF	Null Frame Indicator. This bit corresponds to the Null Frame Indicator in the FlexRay frame header. <ul style="list-style-type: none"> For receive message buffers and receive FIFOs, this is a status bit and represents the value of the Null Frame Indicator of the first valid frame received on the FlexRay bus in the slot indicated by the CYCCNT field. For transmit message buffers, the value of this bit is ignored. The FlexRay block determines internally whether a null frame or non-null frame must be transmitted and sets the Null Frame Indicator accordingly. 0 Null frame received 1 Normal frame received
SYF	Sync Frame Indicator. This bit corresponds to the Sync Frame Indicator in the FlexRay frame header. <ul style="list-style-type: none"> For receive message buffers and receive FIFOs, this is a status bit and represents the value of the Sync Frame Indicator of the first valid frame received on the FlexRay bus in the slot indicated by the CYCCNT field. For transmit message buffers, the value of this bit is ignored. The FlexRay block determines internally whether a sync frame must be transmitted and sets the Sync Frame Indicator accordingly.
SUF	Startup Frame Indicator. This bit corresponds to the Startup Frame Indicator in the FlexRay frame header. <ul style="list-style-type: none"> For receive message buffers and receive FIFOs, this is a status bit and represents the value of the Startup Frame Indicator of the first valid frame received on the FlexRay bus in the slot indicated by the CYCCNT field. For transmit message buffers, the value of this bit is ignored. The FlexRay block determines internally whether a startup frame must be transmitted and sets the Startup Frame Indicator accordingly.
FID	Frame ID. <ul style="list-style-type: none"> For receive message buffers and receive FIFOs, this field provides the value of the Frame ID field of the first valid frame received on the FlexRay bus in the slot indicated by the CYCCNT field. For transmit message buffers, this field provides the value that will be transmitted in the Frame ID field of the FlexRay frame. Note: For transmit message buffers, the application must program this field to the same value as in the corresponding Message Buffer Frame ID Registers (MBFIDRn) . If the FlexRay block detects a mismatch while transmitting the frame header, it will set the frame ID error flag <code>FID_EF</code> in the CHI Error Flag Register (CHIERFR) . The value of the FID field will be ignored and replaced by the value provided in the Message Buffer Frame ID Registers (MBFIDRn) .

Table 30-83. Frame Header Field Descriptions (continued)

Field	Description
CYCCNT	<p>Cycle Count.</p> <ul style="list-style-type: none"> For receive message buffer and receive FIFOs, this field provides the number of the communication cycle in which the frame stored in this message buffer was received. For transmit message buffers, the value of this field is ignored. The FlexRay block will overwrite this value with the current cycle count value when it transmits the frame.
PLDLLEN	<p>Payload Length in 16-Bit Units.</p> <ul style="list-style-type: none"> For receive message buffers and receive FIFOs, this field provides the value of the payload length field of the first valid frame received on the FlexRay bus in the slot indicated by the FID field. For transmit message buffers assigned to the static segment, this value is ignored for the frame generation. The FlexRay block uses the value in the PCR19.payload_length_static to set the value of the <i>Payload length</i> field in the transmitted frame. For transmit message buffers assigned to the dynamic segment, this value is used to set the value of the <i>Payload length</i> field in the transmitted frame. <p>Note: The value of this field is given in numbers of 16-bit units</p>
HDCRC	<p>Header CRC.</p> <ul style="list-style-type: none"> For receive and FIFO message buffers, this field provides the value of the <i>Header CRC</i> of the received frame. For transmit message buffers, this field provides the <i>Header CRC</i> value as it was given by the application. The FlexRay block transmits this value in the <i>Header CRC</i> field of the transmitted frame.

30.6.5.2.2 Data Field Offset Description

Data Field Offset Content

For a detailed description of the Data Field Offset, see [Section 30.6.2.1.2, “Data Field Offset.”](#)

Data Field Offset Access

The application shall program the Data Field Offset when configuring the message buffers either in the *POC:config* state or when the message buffer is disabled.

30.6.5.2.3 Slot Status Description

The slot status is a read-only structure for the application and a write-only structure for the FlexRay block. The meaning and content of the slot status in the message buffer header field depends on the message buffer type.

Receive Message Buffer and Receive FIFO Slot Status Description

This section describes the slot status structure for the individual receive message buffers and receive FIFOs. The content of the slot status structure for receive message buffers depends on the message buffer type and on the channel assignment for individual receive message buffers as given by [Table 30-84](#).

Table 30-84. Receive Message Buffer Slot Status Content

Receive Message Buffer Type	Slot Status Content
Individual Receive Message Buffer assigned to both channels MBCCSRn.CHA=1 and MBCCSRn.CHB=1	see Figure 30-105
Individual Receive Message Buffer assigned to channel A MBCCSRn.CHA=1 and MBCCSRn.CHB=0	see Figure 30-106
Individual Receive Message Buffer assigned to channel B MBCCSRn.CHA=0 and MBCCSRn.CHB=1	see Figure 30-107
Receive FIFO Channel A Message Buffer	see Figure 30-106
Receive FIFO Channel B Message Buffer	see Figure 30-107

The meaning of the bits in the slot status structure is explained in [Table 30-85](#).

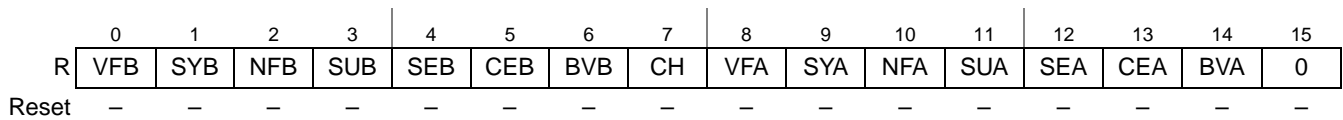


Figure 30-105. Receive Message Buffer Slot Status Structure (ChAB)

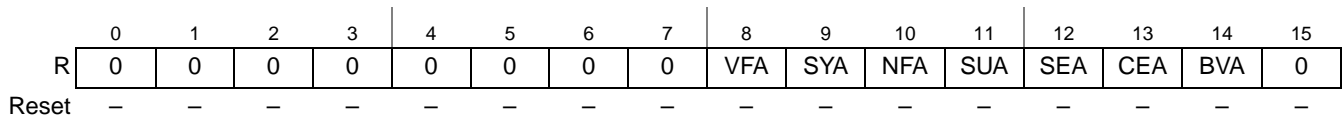


Figure 30-106. Receive Message Buffer Slot Status Structure (ChA)

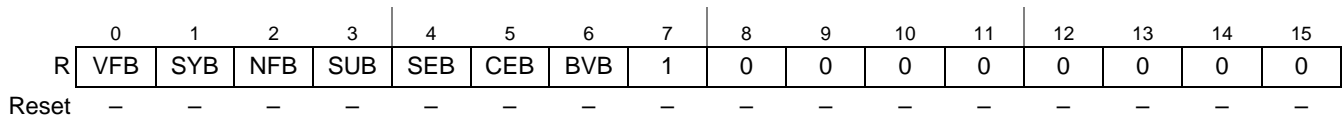


Figure 30-107. Receive Message Buffer Slot Status Structure (ChB)

Table 30-85. Receive Message Buffer Slot Status Field Descriptions

Field	Description
Common Message Buffer Status Bits	
VFB	Valid Frame on Channel B. Protocol related variable: <i>vSS!ValidFrame</i> channel B 0 <i>vSS!ValidFrame</i> = 0 1 <i>vSS!ValidFrame</i> = 1
SYB	Sync Frame Indicator Channel B. Protocol related variable: <i>vRF!Header!SyFIndicator</i> channel B 0 <i>vRF!Header!SyFIndicator</i> = 0 1 <i>vRF!Header!SyFIndicator</i> = 1
NFB	Null Frame Indicator Channel B. Protocol related variable: <i>vRF!Header!NFIndicator</i> channel B 0 <i>vRF!Header!NFIndicator</i> = 0 1 <i>vRF!Header!NFIndicator</i> = 1
SUB	Startup Frame Indicator Channel B. Protocol related variable: <i>vRF!Header!SuFIndicator</i> channel B 0 <i>vRF!Header!SuFIndicator</i> = 0 1 <i>vRF!Header!SuFIndicator</i> = 1

Table 30-85. Receive Message Buffer Slot Status Field Descriptions (continued)

Field	Description
SEB	Syntax Error on Channel B. Protocol related variable: <i>vSS!SyntaxError</i> channel B 0 <i>vSS!SyntaxError</i> = 0 1 <i>vSS!SyntaxError</i> = 1
CEB	Content Error on Channel B. Protocol related variable: <i>vSS!ContentError</i> channel B 0 <i>vSS!ContentError</i> = 0 1 <i>vSS!ContentError</i> = 1
BVB	Boundary Violation on Channel B. Protocol related variable: <i>vSS!BViolation</i> channel B 0 <i>vSS!BViolation</i> = 0 1 <i>vSS!BViolation</i> = 1
CH	Channel First Valid Received. This status bit applies only to receive message buffers assigned to the static segment and to both channels. It indicates the channel that has received the <i>first valid</i> frame in the slot. This flag is set to 0 if no valid frame was received at all in the subscribed slot. 0 first valid frame received on channel A, or no valid frame received at all 0 first valid frame received on channel B
VFA	Valid Frame on Channel A. protocol related variable: <i>vSS!ValidFrame</i> channel A 0 <i>vSS!ValidFrame</i> = 0 1 <i>vSS!ValidFrame</i> = 1
SYA	Sync Frame Indicator Channel A. Protocol related variable: <i>vRF!Header!SyFIndicator</i> channel A 0 <i>vRF!Header!SyFIndicator</i> = 0 1 <i>vRF!Header!SyFIndicator</i> = 1
NFA	Null Frame Indicator Channel A. Protocol related variable: <i>vRF!Header!NFIndicator</i> channel A 0 <i>vRF!Header!NFIndicator</i> = 0 1 <i>vRF!Header!NFIndicator</i> = 1
SUA	Startup Frame Indicator Channel A. Protocol related variable: <i>vRF!Header!SuFIndicator</i> channel A 0 <i>vRF!Header!SuFIndicator</i> = 0 1 <i>vRF!Header!SuFIndicator</i> = 1
SEA	Syntax Error on Channel A — protocol related variable: <i>vSS!SyntaxError</i> channel A 0 <i>vSS!SyntaxError</i> = 0 1 <i>vSS!SyntaxError</i> = 1
CEA	Content Error on Channel A — protocol related variable: <i>vSS!ContentError</i> channel A 0 <i>vSS!ContentError</i> = 0 1 <i>vSS!ContentError</i> = 1
BVA	Boundary Violation on Channel A — protocol related variable: <i>vSS!BViolation</i> channel A 0 <i>vSS!BViolation</i> = 0 1 <i>vSS!BViolation</i> = 1

Transmit Message Buffer Slot Status Description

This section describes the slot status structure for transmit message buffers. Only the TCA and TCB status bits are directly related to the transmission process. All other status bits in this structure are related to a receive process that may have occurred. The content of the slot status structure for transmit message buffers depends on the channel assignment as given by [Table 30-86](#).

Table 30-86. Transmit Message Buffer Slot Status Content

Transmit Message Buffer Type	Slot Status Content
Individual Transmit Message Buffer assigned to both channels MBCCSRn.CHA=1 and MBCCSRn.CHB=1	see Figure 30-108
Individual Transmit Message Buffer assigned to channel A MBCCSRn.CHA=1 and MBCCSRn.CHB=0	see Figure 30-109
Individual Transmit Message Buffer assigned to channel B MBCCSRn.CHA=0 and MBCCSRn.CHB=1	see Figure 30-110

The meaning of the bits in the slot status structure is described in [Table 30-85](#).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 30-108. Transmit Message Buffer Slot Status Structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 30-109. Transmit Message Buffer Slot Status Structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 30-110. Transmit Message Buffer Slot Status Structure (ChB)**Table 30-87. Transmit Message Buffer Slot Status Structure Field Descriptions**

Field	Description
VFB	Valid Frame on Channel B — protocol related variable: <i>vSS!ValidFrame</i> channel B 0 <i>vSS!ValidFrame</i> = 0 1 <i>vSS!ValidFrame</i> = 1
SYB	Sync Frame Indicator Channel B — protocol related variable: <i>vRF!Header!SyFIndicator</i> channel B 0 <i>vRF!Header!SyFIndicator</i> = 0 1 <i>vRF!Header!SyFIndicator</i> = 1
NFB	Null Frame Indicator Channel B — protocol related variable: <i>vRF!Header!NFIndicator</i> channel B 0 <i>vRF!Header!NFIndicator</i> = 0 1 <i>vRF!Header!NFIndicator</i> = 1
SUB	Startup Frame Indicator Channel B — protocol related variable: <i>vRF!Header!SuFIndicator</i> channel B 0 <i>vRF!Header!SuFIndicator</i> = 0 1 <i>vRF!Header!SuFIndicator</i> = 1
SEB	Syntax Error on Channel B — protocol related variable: <i>vSS!SyntaxError</i> channel B 0 <i>vSS!SyntaxError</i> = 0 1 <i>vSS!SyntaxError</i> = 1
CEB	Content Error on Channel B — protocol related variable: <i>vSS!ContentError</i> channel B 0 <i>vSS!ContentError</i> = 0 1 <i>vSS!ContentError</i> = 1

Table 30-87. Transmit Message Buffer Slot Status Structure Field Descriptions (continued)

Field	Description
BVB	Boundary Violation on Channel B — protocol related variable: <i>vSS!BViolation</i> channel B 0 <i>vSS!BViolation</i> = 0 1 <i>vSS!BViolation</i> = 1
TCB	Transmission Conflict on Channel B — protocol related variable: <i>vSS!TxConflict</i> channel B 0 <i>vSS!TxConflict</i> = 0 1 <i>vSS!TxConflict</i> = 1
VFA	Valid Frame on Channel A — protocol related variable: <i>vSS!ValidFrame</i> channel A 0 <i>vSS!ValidFrame</i> = 0 1 <i>vSS!ValidFrame</i> = 1
SYA	Sync Frame Indicator Channel A — protocol related variable: <i>vRF!Header!SyFIndicator</i> channel A 0 <i>vRF!Header!SyFIndicator</i> = 0 1 <i>vRF!Header!SyFIndicator</i> = 1
NFA	Null Frame Indicator Channel A — protocol related variable: <i>vRF!Header!NFIndicator</i> channel A 0 <i>vRF!Header!NFIndicator</i> = 0 1 <i>vRF!Header!NFIndicator</i> = 1
SUA	Startup Frame Indicator Channel A — protocol related variable: <i>vRF!Header!SuFIndicator</i> channel A 0 <i>vRF!Header!SuFIndicator</i> = 0 1 <i>vRF!Header!SuFIndicator</i> = 1
SEA	Syntax Error on Channel A — protocol related variable: <i>vSS!SyntaxError</i> channel A 0 <i>vSS!SyntaxError</i> = 0 1 <i>vSS!SyntaxError</i> = 1
CEA	Content Error on Channel A — protocol related variable: <i>vSS!ContentError</i> channel A 0 <i>vSS!ContentError</i> = 0 1 <i>vSS!ContentError</i> = 1
BVA	Boundary Violation on Channel A — protocol related variable: <i>vSS!BViolation</i> channel A 0 <i>vSS!BViolation</i> = 0 1 <i>vSS!BViolation</i> = 1
TCA	Transmission Conflict on Channel A — protocol related variable: <i>vSS!TxConflict</i> channel A 0 <i>vSS!TxConflict</i> = 0 1 <i>vSS!TxConflict</i> = 1

30.6.5.3 Message Buffer Data Field Description

The message buffer data field is used to store the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum required length of this field depends on the message buffer type that the physical message buffer is assigned to and is given in Table 30-88. The structure of the message buffer data field is given in Figure 30-111.

Table 30-88. Message Buffer Data Field Minimum Length

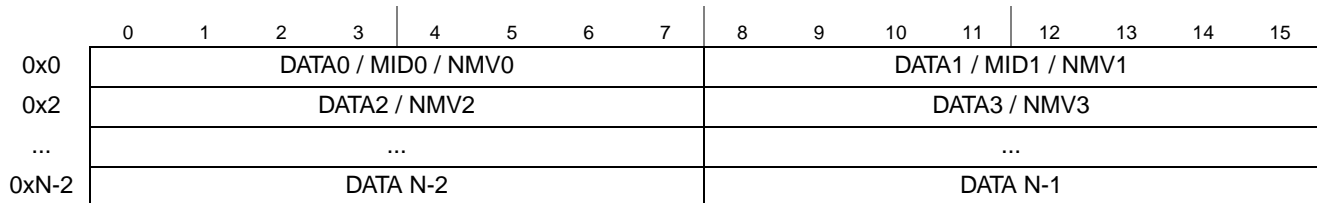
physical message buffer assigned to	minimum length defined by
Individual Message Buffer in Segment 1	MBDSR.MBSEG1DS
Receive Shadow Buffer in Segment 1	MBDSR.MBSEG1DS
Individual Message Buffer in Segment 2	MBDSR.MBSEG2DS
Receive Shadow Buffer in Segment 2	MBDSR.MBSEG2DS
Receive FIFO for channel A	RFDSR.ENTRY_SIZE (RFSR.SEL = 0)

Table 30-88. Message Buffer Data Field Minimum Length

physical message buffer assigned to	minimum length defined by
Receive FIFO for channel B	RFDSR.ENTRY_SIZE (RFSR.SEL = 1)

NOTE

The FlexRay block will not access any locations outside the message buffer data field boundaries given by [Table 30-88](#).

**Figure 30-111. Message Buffer Data Field Structure**

The message buffer data field is located in the FRM; thus, the FlexRay block has no means to control application write access to the field. To ensure data consistency, the application must follow a write and read access scheme.

30.6.5.3.1 Message Buffer Data Field Read Access

For transmit message buffers, the FlexRay block will not modify the content of the Message Buffer Data Field. Thus the application can read back the data at any time without any impact on data consistency.

For receive message buffers the application must lock the related receive message buffer and retrieve the message buffer header index from the [Message Buffer Index Registers \(MBIDXRn\)](#). While the message buffer is locked, the FlexRay block will not update the Message Buffer Data Field.

For receive FIFOs, the application can read the message buffer indicated by the [Receive FIFO A Read Index Register \(RFARIR\)](#) or the [Receive FIFO B Read Index Register \(RFBRIR\)](#) when the related receive FIFO non-empty interrupt flag FNEAIF or FNEBIF is set in the [Global Interrupt Flag and Enable Register \(GIFER\)](#). While the non-empty interrupt flag is set, the FlexRay block will not update the Message Buffer Data Field related to message buffer indicated by [Receive FIFO A Read Index Register \(RFARIR\)](#) or the [Receive FIFO B Read Index Register \(RFBRIR\)](#).

30.6.5.3.2 Message Buffer Data Field Write Access

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the message buffer data field.

For transmit message buffers, the application must follow the write access restrictions given in [Table 30-89](#).

Table 30-89. Frame Data Write Access Constraints

Field	single buffered	double buffered	
		commit side	transmit side
DATA, MID, NMV	<i>POC:config</i> or MB_DIS or MB_LCK	<i>POC:config</i> or MB_DIS or MB_LCK	<i>POC:config</i> or MB_DIS

Table 30-90. Frame Data Field Descriptions

Field	Description
DATA[0:N-1]	Message Data — Provides the message data received or to be transmitted. For receive message buffer and receive FIFOs, this field provides the message data received for this message buffer. For transmit message buffers, the field provides the message data to be transmitted.
MID[0:1]	Message Identifier — If the payload preamble bit PPI is set in the message buffer frame header, the MID field holds the message ID of a dynamic frame located in the message buffer. The receive FIFO filter uses the received message ID for message ID filtering.
NMV[0:11]	Network Management Vector — If the payload preamble bit PPI is set in the message buffer frame header, the network management vector field holds the network management vector of a static frame located in the message buffer. Note: The MID and NMV bytes replace the corresponding DATA bytes.

30.6.6 Individual Message Buffer Functional Description

The FlexRay block provides three basic types of individual message buffers:

1. Single Transmit Message Buffers
2. Double Transmit Message Buffers
3. Receive Message Buffers

Before an individual message buffer can be used, it must be configured by the application. After the initial configuration, the message buffer can be reconfigured later. The set of the configuration data for individual message buffers is given in [Section 30.6.3.4.1, “Individual Message Buffer Configuration Data”](#).

30.6.6.1 Individual Message Buffer Configuration

The individual message buffer configuration consists of two steps. The first step is the allocation of the required amount of memory for the FRM. The second step is the programming of the message buffer configuration registers, which is described in this section.

30.6.6.1.1 Common Configuration Data

One part of the message buffer configuration data is common to all individual message buffers and the receive shadow buffers. These data can only be set when the protocol is in the *POC:config* state.

The application configures the number of utilized individual message buffers by writing the message buffer number of the last utilized message buffer into the LAST_MB_UTIL field in the [Message Buffer Segment Size and Utilization Register \(MBSSUTR\)](#).

The application configures the size of the two segments of individual message buffers by writing the message buffer number of the last message buffer in the first segment into the LAST_MB_SEG1 field in the [Message Buffer Segment Size and Utilization Register \(MBSSUTR\)](#)

The application configures the length of the message buffer data fields for both of the message buffer segments by writing to the MBSEG2DS and MBSEG1DS fields in the [Message Buffer Data Size Register \(MBDSR\)](#).

Depending on the current receive functionality of the FlexRay block, the application must configure the receive shadow buffers. For each segment and for each channel with at least one individual receive message buffer assigned, the application must configure the related receive shadow buffer using the [Receive Shadow Buffer Index Register \(RSBIR\)](#).

30.6.6.1.2 Specific Configuration Data

The second part of the message buffer configuration data is specific for each message buffer.

These data can be changed only when either

- the protocol is in the *POC:config* state or
- the message buffer is disabled, i.e. MBCCSRn.EDS = 0

The individual message buffer type is defined by the MTD and MBT bits in the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#) as given in [Table 30-91](#).

Table 30-91. Individual Message Buffer Types

MBCCSRn.MTD	MBCCSRn.MBT	Individual Message Buffer Description
0	0	Receive Message Buffer
0	1	Reserved
1	0	Single Transmit Message Buffer
1	1	Double Transmit Message Buffer

The message buffer specific configuration data are

1. MCM, MBT, MTD bits in [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#)
2. all fields and bits in [Message Buffer Cycle Counter Filter Registers \(MBCCFRn\)](#)
3. all fields and bits in [Message Buffer Frame ID Registers \(MBFIDRn\)](#)
4. all fields and bits in [Message Buffer Index Registers \(MBIDXRn\)](#)

The meaning of the specific configuration data depends on the message buffer type, as given in the detailed message buffer type descriptions [Section 30.6.6.2, “Single Transmit Message Buffers”](#), [Section 30.6.6.3, “Receive Message Buffers”](#), and [Section 30.6.6.4, “Double Transmit Message Buffer”](#).

30.6.6.2 Single Transmit Message Buffers

The section provides a detailed description of the functionality of single buffered transmit message buffers.

A single transmit message buffer is used by the application to provide message data to the FlexRay block that will be transmitted over the FlexRay Bus. The FlexRay block uses the transmit message buffers to

provide information about the transmission process and status information about the slot in which message was transmitted.

The individual message buffer with message buffer number n is configured to be a single transmit message buffer by the following settings:

- MBCCSRn.MBT = 0 (single buffered message buffer)
- MBCCSRn.MTD = 1 (transmit message buffer)

30.6.6.2.1 Access Regions

To certain message buffer fields, both the application and the FlexRay block have access. To ensure data consistency, a message buffer locking scheme is implemented, which is used to control the access to the data, control, and status bits of a message buffer. The access regions for single transmit message buffers are depicted in Figure 30-112. A description of the regions is given in Table 30-92. If an region is active as indicated in Table 30-93, the access scheme given for that region applies to the message buffer.

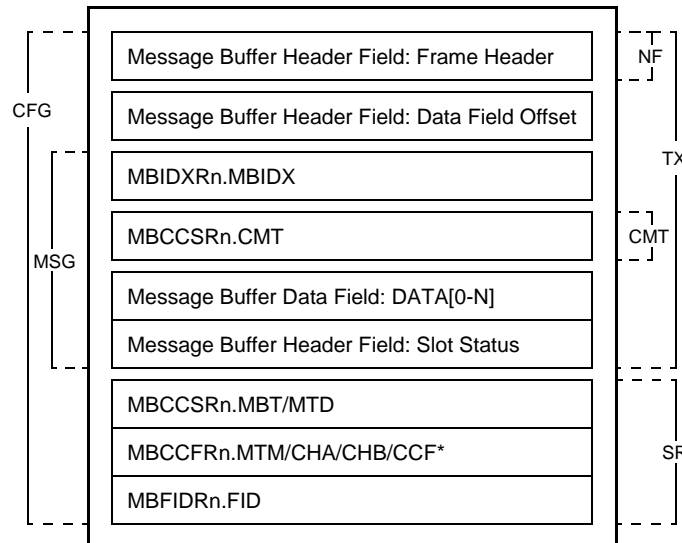


Figure 30-112. Single Transmit Message Buffer Access Regions

Table 30-92. Single Transmit Message Buffer Access Regions Description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	-	Message Buffer Configuration
MSG	read/write	-	Message Data and Slot Status Access
NF	-	read-only	Message Header Access for Null Frame Transmission
TX	-	read/write	Message Transmission and Slot Status Update
CM	-	read-only	Message Buffer Validation
SR	-	read-only	Message Buffer Search

The trigger bits MBCCSRn.EDT and MBCCSRn.LCKT, and the interrupt enable bit MBCCSRn.MBIE are not under access control and can be accessed from the application at any time. The status bits

MBCCSRn.EDS and MBCCSRn.LCKS are not under access control and can be accessed from the FlexRay block at any time.

The interrupt flag MBCCSnR.MBIF is not under access control and can be accessed from the application and the FlexRay block at any time. FlexRay block clear access has higher priority.

The FlexRay block restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The transmit message buffer states are given in Figure 30-113. A description of the states is given in Table 30-93, which also provides the access scheme for the access regions.

The status bits MBCCSRn.EDS and MBCCSRn.LCKS provide the application with the required message buffer status information. The internal status information is not visible to the application.

30.6.6.2.2 Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

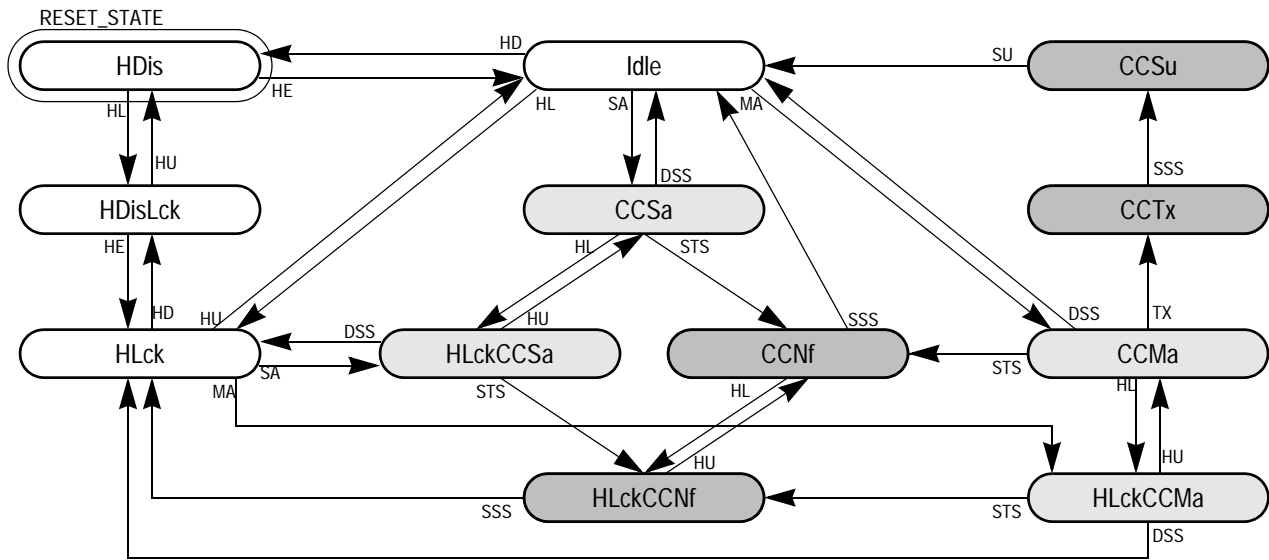


Figure 30-113. Single Transmit Message Buffer States

Table 30-93. Single Transmit Message Buffer State Description (Sheet 1 of 2)

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	–	CM, SR	Idle - Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	–	Disabled - Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	–	Disabled and Locked - Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	SR	Locked - Applications access to data, control, and status. Included in message buffer search.

Table 30-93. Single Transmit Message Buffer State Description (Sheet 2 of 2)

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
CCSa	1	0	–	–	Slot Assigned - Message buffer assigned to next static slot. Ready for Null Frame transmission.
HLckCCSa	1	1	MSG	–	Locked and Slot Assigned - Applications access to data, control, and status. Message buffer assigned to next static slot
CCNf	1	0	–	NF	Null Frame Transmission Header is used for null frame transmission.
HLckCCNf	1	1	MSG	NF	Locked and Null Frame Transmission - Applications access to data, control, and status. Header is used for null frame transmission.
CCMa	1	0	–	CM	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
HLckCCMa	1	1	MSG	–	Locked and Message Available - Applications access to data, control, and status. Message buffer is assigned to next slot and cycle counter filter matches.
CCTx	1	0	–	TX	Message Transmission - Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	–	TX	Status Update - Message buffer status update. Update of status flags, the slot status field, and the header index.

30.6.6.2.3 Message Buffer Transitions

Application Transitions

The application transitions can be triggered by the application using the commands described in [Table 30-94](#). The application issues the commands by writing to the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#). Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

The enable and disable commands issued by writing 1 to the trigger bit MBCCSRn.EDT. The transition that will be triggered by each of these command depends on the current value of the status bit MBCCSRn.EDS. If the command triggers the disable transition HD and the message buffer is in one of the states CCSa, HLckCCSa, CCMa, HLckCCMa, CCNf, HLckCCNf, or CCTx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

The lock and unlock commands issued by writing 1 to the trigger bit MBCCSRn.LCKT. The transition that will be triggered by each of these commands depends on the current value of the status bit MBCCSRn.LCKS. If the command triggers the lock transition HL and the message buffer is in the state CCTx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the [CHI Error Flag Register \(CHIERFR\)](#) is set.

Table 30-94. Single Transmit Message Buffer Application Transitions

Transition	Command	Condition	Description
HE	MBCCSRn.EDT:= 1	MBCCSRn.EDS = 0	Application triggers message buffer enable.
HD		MBCCSRn.EDS = 1	Application triggers message buffer disable.

Table 30-94. Single Transmit Message Buffer Application Transitions

Transition	Command	Condition	Description
HL	MBCCSRn.LCKT:= 1	MBCCSRn.LCKS = 0	Application triggers message buffer lock.
HU		MBCCSRn.LCKS = 1	Application triggers message buffer unlock.

Module Transitions

The module transitions that can be triggered by the FlexRay block are described in [Table 30-95](#). Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 30-95. Single Transmit Message Buffer Module Transitions

Transition	Condition	Description
SA	slot match and static slot	Slot Assigned - Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and MBCCSRn.CMT = 1	Transmission Slot Start - Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated - Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start - Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start . - Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of static slot or dynamic slot or symbol window or NIT.

Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of [Table 30-96](#), the module transitions have a higher priority than the application transitions. For all states except the CCMA state, both a lock/unlock transition HL/HD and a module transition can be executed at the same time. The result state is reached by first applying the application transition and subsequently the module transition to the intermediately reached state. For example, if the message buffer is in the HLck state and the application unlocks the message buffer by the HU transition and the module triggers the slot assigned transition SA, the intermediate state is Idle and the resulting state is CCSa.

The priorities among the module transitions is given in the second part of [Table 30-96](#).

Table 30-96. Single Transmit Message Buffer Transition Priorities

State	Priorities	Description
module vs. application		
Idle, HLck	SA > HD MA > HD	Slot Assigned > Message Buffer Disable Message Available > Message Buffer Disable
CCMA	TX > HL	Transmission Start > Message Buffer Lock

Table 30-96. Single Transmit Message Buffer Transition Priorities

State	Priorities	Description
module internal		
Idle, HLck	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS	Transmission Slot Start > Static Slot Start
	TX > DSS	Transmission Slot Start > Dynamic Slot Start

30.6.6.2.4 Transmit Message Setup

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field and sets the commit bit CMT in the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#). The physical access to the message buffer data field is described in [Section 30.6.3.1, “Individual Message Buffers”](#).

As indicated by [Table 30-93](#), the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa. The application can change the state of a message buffer if it issues the appropriate commands shown in [Table 30-94](#). The state change is indicated through the MBCCSRn.EDS and MBCCSRn.LCKS status bits.

If the transmit message buffer enters one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa the MBCCSRn.DVAL flag is negated.

30.6.6.2.5 Message Transmission

As a result of the message buffer search described in [Section 30.6.7, “Individual Message Buffer Search”](#), the FlexRay block triggers the message available transition MA for up to two transmit message buffers. This changes the message buffer state from Idle to CCMa and the message buffers can be used for message transmission in the next slot.

The FlexRay block transmits a message from a message buffer if both of the following two conditions are fulfilled at the start of the transmission slot:

1. the message buffer is in the message available state CCMa
2. the message data are still valid, i.e. MBCCSRn.CMT = 1

In this case, the FlexRay block triggers the TX transition and changes the message buffer state to CCTx. A transmit message buffer timing and state change diagram for message transmission is given in [Figure 30-114](#). In this example, the message buffer with message buffer number n is Idle at the start of the search slot, matches the slot and cycle number of the next slot, and message buffer data are valid, i.e. MBCCSRn.CMT = 1.

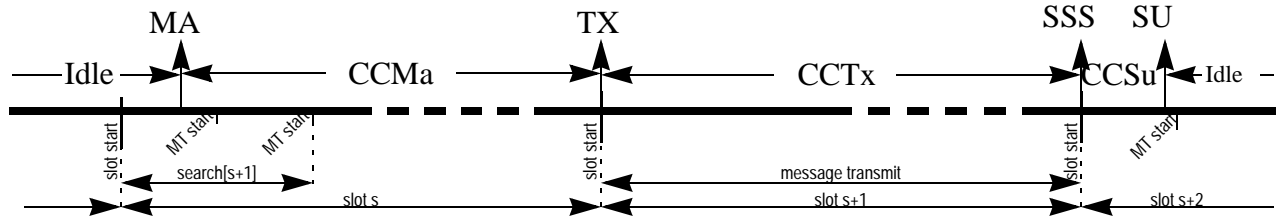


Figure 30-114. Message Transmission Timing

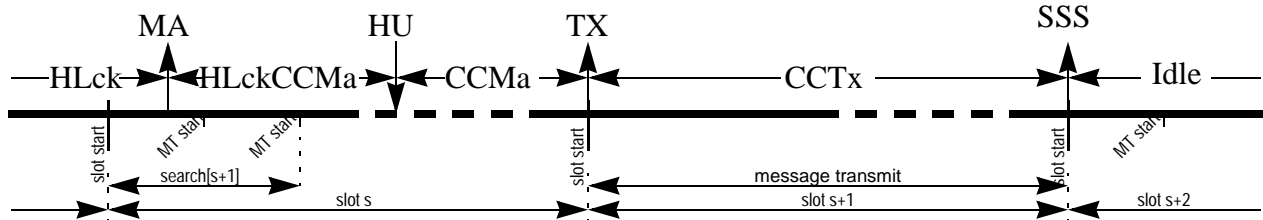


Figure 30-115. Message Transmission from HLck state with unlock

The amount of message data read from the FRM and transferred to the FlexRay bus is determined by the following three items

1. the message buffer segment that the message buffer is assigned to, as defined by the [Message Buffer Segment Size and Utilization Register \(MBSSUTR\)](#).
2. the message buffer data field size, as defined by the related field of the [Message Buffer Data Size Register \(MBDSR\)](#)
3. the value of the PLDLEN field in the message buffer header field, as described in [Section 30.6.5.2.1, “Frame Header Section Description”](#)

If a message buffer is assigned to message buffer segment 1, and $PLDLEN > MBSEG1DS$, then $2 * MBSEG1DS$ bytes will be read from the message buffer data field and zero padding is used for the remaining bytes for the FlexRay bus transfer. If $PLDLEN \leq MBSEG1DS$, the FlexRay block reads and transfers $2 * PLDLEN$ bytes. The same holds for segment 2 and $MBSEG2DS$.

30.6.6.2.6 Null Frame Transmission

A static slot with slot number S is assigned to the FlexRay block for channel A, if at least one transmit message buffer is configured with the $MBFIDRn.FID$ set to S and $MBCCFRn.CHA$ set to 1. A Null Frame is transmitted in the static slot S on channel A, if this slot is assigned to the FlexRay block for channel A, and all transmit message buffers with $MBFIDRn.FID = s$ and $MBCCFRn.CHA = 1$ are either not committed, i.e. $MBCCSRn.CMT = 0$, or locked by the application, i.e. $MBCCSRn.LCKS = 1$, or the cycle counter filter is enabled and does not match.

Additionally, the application can clear the commit bit of a message buffer that is in the $CCMa$ state, which is called *uncommit* or *transmit abort*. This message buffer will be used for null frame transmission.

As a result of the message buffer search described in [Section 30.6.7, “Individual Message Buffer Search”](#), the FlexRay block triggers the slot assigned transition SA for up to two transmit message buffers if at least

one of the conditions mentioned above is fulfilled for these message buffers. The transition SA changes the message buffer states from either Idle to CCSa or from HLck to HLckCCSa. In each case, these message buffers will be used for null frame transmission in the next slot. A message buffer timing and state change diagram for null frame transmission from Idle state is given in [Figure 30-116](#).

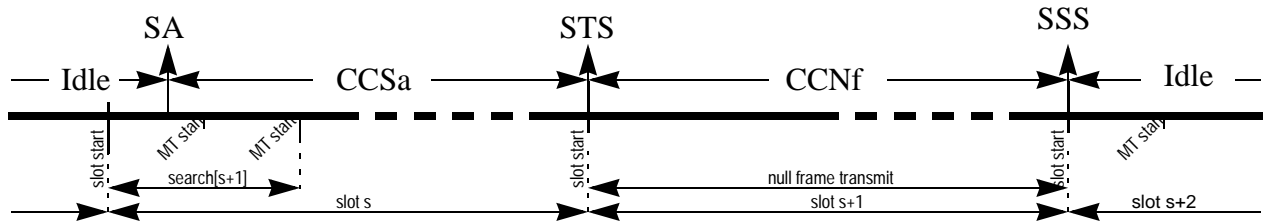


Figure 30-116. Null Frame Transmission from Idle state

A message buffer timing and state change diagram for null frame transmission from HLck state is given in [Figure 30-117](#).

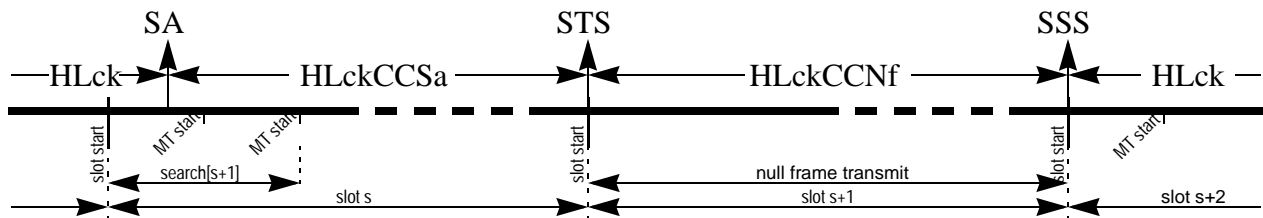


Figure 30-117. Null Frame Transmission from HLck state

If a transmit message buffer is in the CCSa or HLckCCSa state at the start of the transmission slot, a null frame is transmitted in any case, even if the message buffer is unlocked or committed before the transmission slot starts. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in [Figure 30-118](#).

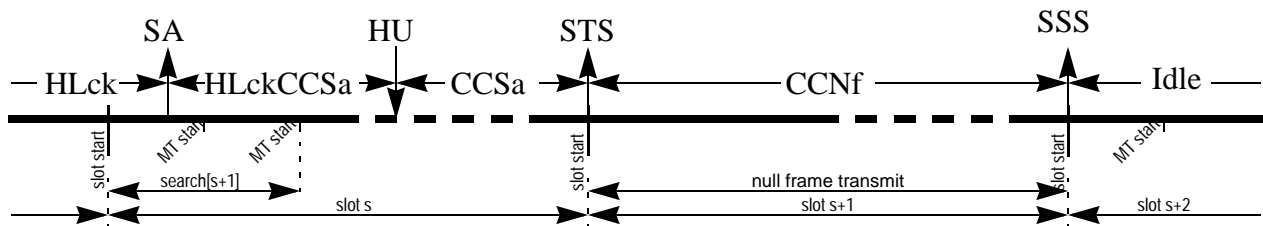


Figure 30-118. Null Frame Transmission from HLck state with unlock

Since the null frame transmission will not use the message buffer data, the application can lock/unlock the message buffer during null frame transmission. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in [Figure 30-119](#).

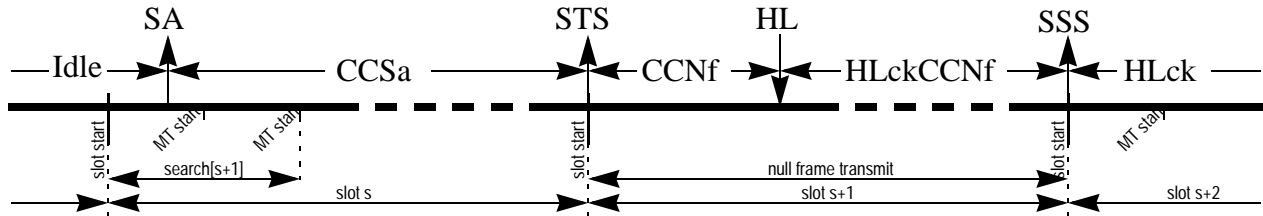


Figure 30-119. Null Frame Transmission from Idle state with locking

30.6.6.2.7 Message Buffer Status Update

After the end of each slot, the PE generates the slot status vector. Depending on the this status, the transmitted frame type, and the amount of transmitted data, the message buffer status is updated.

Message Buffer Status Update after Complete Message Transmission

The term complete message transmission refers to the fact that all payload data stored in the message buffer were send to FlexRay bus. In this case, the FlexRay block updates the slot status field of the message buffer and triggers the status updated transition SU. With the SU transition, the FlexRay block sets the message buffer interrupt flag MBCCSn.MBIF to indicate the successful message transmission.

Depending on the transmission mode flag MBCCFRn.MTM, the FlexRay block changes the commit flag MBCCSRn.CMT and the valid flag MBCCSRn.DVAL. If the MBCCFRn.MTM flag is negated, the message buffer is in the *event transmission mode*. In this case, each committed message is transmitted only once. The commit flag MBCCSRn.CMT is cleared with the SU transition. If the MBCCFRn.MTM flag is asserted, the message buffer is in the *state transmission mode*. In this case, each committed message is transmitted as long as the application provides new data or locks the message buffers. The FlexRay block will not clear the MBCCSRn.CMT flag at the end of transmission and will set the valid flag MBCCSRn.DVAL to indicate that the message will be transmitted again.

Message Buffer Status Update after Incomplete Message Transmission

The term incomplete message transmission refers to the fact that not all payload data that should be transmitted were send to FlexRay bus. This may be caused by the following regular conditions in the dynamic segment:

1. The transmission slot starts in a minislot with a minislot number greater than *pLatestTx*.
2. The transmission slot did not exist in the dynamic segment at all.

Additionally, an incomplete message transmission can be caused by internal communication errors. If those error occur, the Protocol Engine Communication Failure Interrupt Flag PECF_IF is set in the [Protocol Interrupt Flag Register 1 \(PIFR1\)](#).

In any of these two cases, the status of the message buffer is not changed at all with the SU transition. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

Message Buffer Status Update after Null Frame Transmission

After the transmission of a null frame, the status of the message buffer that was used for the null frame transmission is not changed at all. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

30.6.6.3 Receive Message Buffers

The section provides a detailed description of the functionality of the receive message buffers.

A receive message buffer is used to receive a message from the FlexRay Bus based on individual filter criteria. The FlexRay block uses the receive message buffer to provide the following data to the application

1. message data received
2. information about the reception process
3. status information about the slot in which the message was received

A individual message buffer with message buffer number n is configured as a receive message buffer by the following configuration settings

- MBCCSRn.MBT = 0 (single buffered message buffer)
- MBCCSRn.MTD = 0 (receive message buffer)

To certain message buffer fields, both the application and the FlexRay block have access. To ensure data consistency, a message buffer locking scheme is implemented that is used to control the access to the data, control, and status bits of a message buffer. The access regions for receive message buffers are depicted in [Figure 30-120](#). A description of the regions is given in [Table 30-97](#). If an region is active as indicated in [Table 30-98](#), the access scheme given for that region applies to the message buffer.

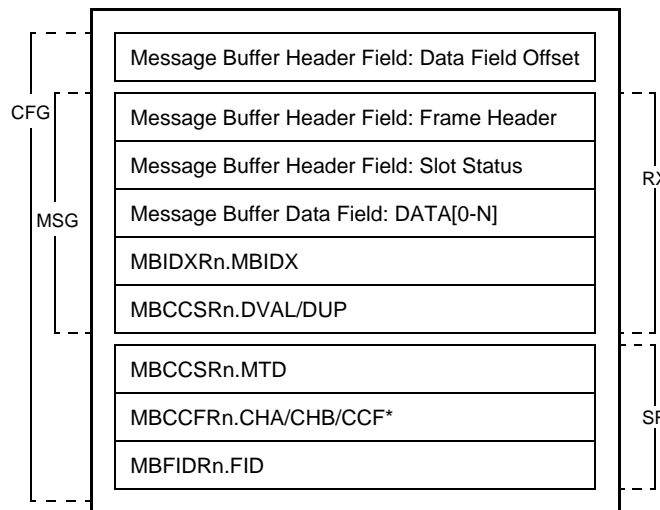


Figure 30-120. Receive Message Buffer Access Regions

Table 30-97. Receive Message Buffer Access Region Description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	-	Message Buffer Configuration, Message Data and Status Access
MSG	read/write	-	Message Data, Header, and Status Access
RX	-	write-only	Message Reception and Status Update
SR	-	read-only	Message Buffer Search Data

The trigger bits MBCCSRn.EDT and MBCCSRn.LCKT and the interrupt enable bit MBCCSRn.MBIE are not under access control and can be accessed from the application at any time. The status bits MBCCSRn.EDS and MBCCSRn.LCKS are not under access control and can be accessed from the FlexRay block at any time.

The interrupt flag MBCCSRn.MBIF is not under access control and can be accessed from the application and the FlexRay block at any time. FlexRay block set access has higher priority.

The FlexRay block restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The receive message buffer states are given in Figure 30-121. A description of the message buffer states is given in Table 30-93, which also provides the access scheme for the access regions.

The status bits MBCCSRn.EDS and MBCCSRn.LCKS provide the application with the required status information. The internal status information is not visible to the application.

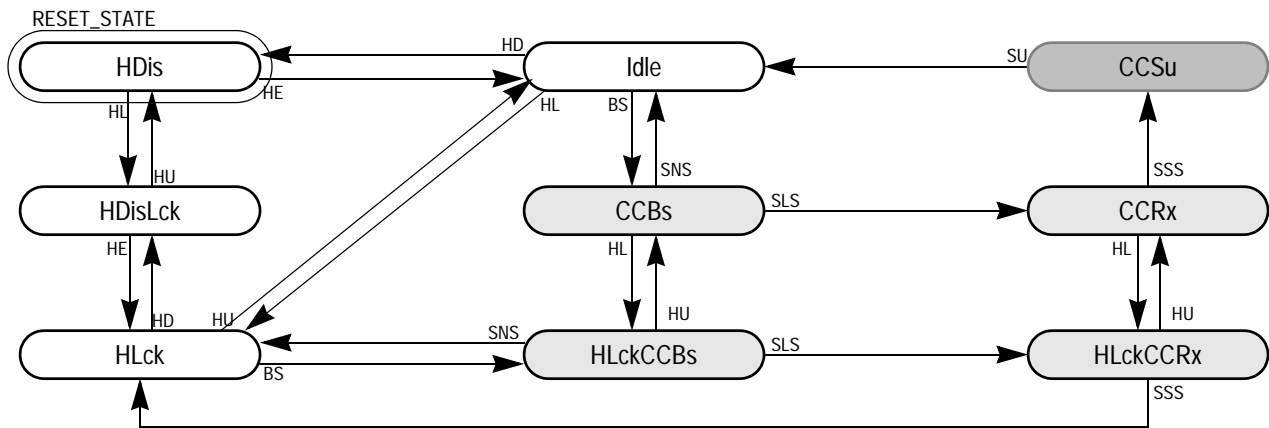


Figure 30-121. Receive Message Buffer States

Table 30-98. Receive Message Buffer States and Access (Sheet 1 of 2)

State	MBCCSRn		Access from		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	-	SR	Idle - Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	-	Disabled - Message Buffer under configuration. Excluded from message buffer search.

Table 30-98. Receive Message Buffer States and Access (Sheet 2 of 2)

State	MBCCSRn		Access from		Description
	EDS	LCKS	Appl.	Module	
HDisLck	0	1	CFG	–	Disabled and Locked - Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	–	Locked - Applications access to data, control, and status. Included in message buffer search.
CCBs	1	0	–	–	Buffer Subscribed - Message buffer subscribed for reception. Filter matches next (slot, cycle, channel) tuple.
HLckCCBs	1	1	MSG	–	Locked and Buffer Subscribed - Applications access to data, control, and status. Message buffer subscribed for reception.
CCRx	1	0	–	–	Message Receive - Message data received into related shadow buffer.
HLckCCRx	1	1	MSG	–	Locked and Message Receive - Applications access to data, control, and status. Message data received into related shadow buffer.
CCSu	1	0	–	RX	Status Update - Message buffer status update. Update of status flags, the slot status field, and the header index.

30.6.6.3.1 Message Buffer Transitions

Application Transitions

The application transitions that can be triggered by the application using the commands described in [Table 30-99](#). The application issues the commands by writing to the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#). Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

The enable and disable commands issued by writing 1 to the trigger bit MBCCSRn.EDT. The transition that will be triggered by each of these command depends on the current value of the status bit MBCCSRn.EDS. If the command triggers the disable transition HD and the message buffer is in one of the states CCBs, HLckCCBs, or CCRx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

The lock and unlock commands issued by writing 1 to the trigger bit MBCCSRn.LCKT. The transition that will be triggered by each of these commands depends on the current value of the status bit MBCCSRn.LCKS. If the command triggers the lock transition HL while the message buffer is in the state CCRx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the [CHI Error Flag Register \(CHIERFR\)](#) is set.

Table 30-99. Receive Message Buffer Application Transitions

Transition	Host Command	Condition	Description
HE	MBCCSRn.EDT:= 1	MBCCSRn.EDS = 0	Application triggers message buffer enable.
HD		MBCCSRn.EDS = 1	Application triggers message buffer disable.
HL	MBCCSRn.LCKT:= 1	MBCCSRn.LCKS = 0	Application triggers message buffer lock.
HU		MBCCSRn.LCKS = 1	Application triggers message buffer unlock.

Module Transitions

The module transitions that can be triggered by the FlexRay block are described in [Table 30-100](#). Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 30-100. Receive Message Buffer Module Transitions

Transition	Condition	Description
BS	slot match and CycleCounter match	Buffer Subscribed - The message buffer filter matches next slot and cycle.
SLS	slot start	Slot Start - Start of either Static Slot or Dynamic Slot.
SNS	symbol window start or NIT start	Symbol Window or NIT Start - Start of either Symbol Window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of either Static Slot, Dynamic Slot, Symbol Window, or NIT.
SU	status updated	Status Updated - Slot Status field, message buffer status flags, header index updated. Interrupt flag set.

Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in [Table 30-101](#), the module transitions have a higher priority than the application transitions. For all states except the CCRx state, a module transition and the application lock/unlock transition HL/HU and can be executed at the same time. The result state is reached by first applying the module transition and subsequently the application transition to the intermediately reached state. For example, if the message buffer is in the buffer subscribed state CCBs and the module triggers the slot start transition SLS at the same time as the application locks the message buffer by the HL transition, the intermediate state is CCRx and the resulting state is locked buffer subscribed state HLckCCRx.

Table 30-101. Receive Message Buffer Transition Priorities

State	Priorities	Description
module vs. application		
Idle	BS > HD	Buffer Subscribed > Message Buffer Disable
HLck	BS > HD	Buffer Subscribed > Message Buffer Disable
CCRx	SSS > HL	Slot or Segment Start > Message Buffer Lock

30.6.6.3.2 Message Reception

As a result of the message buffer search, the FlexRay block changes the state of up to two enabled receive message buffers from either idle state Idle or locked state HLck to the either subscribed state CCBs or locked buffer subscribed state HLckCCBs by triggering the buffer subscribed transition BS.

If the receive message buffers for the next slot are assigned to both channels, then at most one receive message buffer is changed to a buffer subscribed state.

If more than one matching message buffers assigned to a certain channel, then only the message buffer with the lowest message buffer number is in one of the states mentioned above.

With the start of the next static or dynamic slot the module trigger the slot start transition SLS. This changes the state of the subscribed receive message buffers from either CCBs to CCRx or from HLckCCBs to HLckCCRx, respectively.

During the reception slot, the received frame data are written into the shadow buffers. For details on receive shadow buffers, see [Section 30.6.6.3.5, “Receive Shadow Buffers Concept”](#). The data and status of the receive message buffers that are the CCRx or HLckCCRx are not modified in the reception slot.

30.6.6.3.3 Message Buffer Status Update

With the start of the next static or dynamic slot or with the start of the symbol window or NIT, the module trigger the slot or segment start transition SSS. This transition changes the state of the receiving receive message buffers from either CCRx to CCSu or from HLckCCRx to HLck, respectively.

If a message buffer was in the locked state HLckCCRx, no update will be performed. The received data are lost. This is indicated by setting the Frame Lost Channel A/B Error Flag FRLA_EF/FRLB_EF in the [CHI Error Flag Register \(CHIERFR\)](#).

If a message buffer was in the CCRx state it is now in the CCSu state. After the evaluation of the slot status provided by the PE the message buffer is updated. The message buffer update depends on the slot status bits and the segment the message buffer is assigned to. This is described in [Table 30-102](#).

Table 30-102. Receive Message Buffer Update

<i>vSS!ValidFrame</i>	<i>vRF!Header!NFIndicator</i>	Update description
1	1	Valid non-null frame received. <ul style="list-style-type: none"> - Message Buffer Data Field updated. - Frame Header Field updated. - Slot Status Field updated. - DUP:= 1 - DVAL:= 1 - MBIF:= 1
1	0	Valid null frame received. <ul style="list-style-type: none"> - Message Buffer Data Field <i>not</i> updated. - Frame Header Field <i>not</i> updated. - Slot Status Field updated. - DUP:= 0 - DVAL <i>not</i> changed - MBIF:= 1
0	x	No valid frame received. <ul style="list-style-type: none"> - Message Buffer Data Field not updated. - Frame Header Field not updated. - Slot Status Field updated. - DUP:= 0 - DVAL <i>not</i> changed. - MBIF:= 1, if the slot was not an empty dynamic slot. <p>Note: An empty dynamic slot is indicated by the following frame and slot status bit values: <i>vSS!ValidFrame</i> = 0 and <i>vSS!SyntaxError</i> = 0 and <i>vSS!ContentError</i> = 0 and <i>vSS!BViolation</i> = 0.</p>

NOTE

If the number of the last slot in the current communication cycle on a given channel is n , then all receive message buffers assigned to this channel with $MBFIDRn.FID > n$ will not be updated at all.

When the receive message buffer update has finished the status updated transition SU is triggered, which changes the buffer state from CCSu to Idle. An example receive message buffer timing and state change diagram for a normal frame reception is given in [Figure 30-122](#).

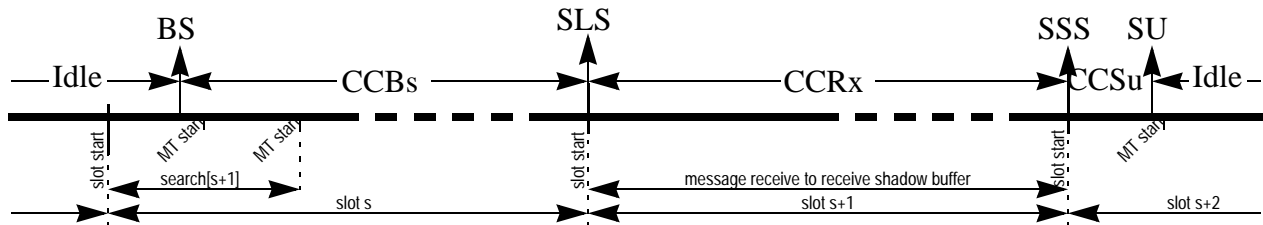


Figure 30-122. Message Reception Timing

The amount of message data written into the message buffer data field of the receive shadow buffer is determined by the following two items:

1. the message buffer segment that the message buffer is assigned to, as defined by the [Message Buffer Segment Size and Utilization Register \(MBSSUTR\)](#).
2. the message buffer data field size, as defined by the related field of the [Message Buffer Data Size Register \(MBDSR\)](#)
3. the number of bytes received over the FlexRay bus

If the message buffer is assigned to the message buffer segment 1, and the number of received bytes is greater than $2 * MBDSR.MBSEG1DS$, the FlexRay block writes only $2 * MBDSR.MBSEG1DS$ bytes into the message buffer data field of the receive shadow buffer. If the number of received bytes is less than $2 * MBDSR.MBSEG1DS$, the FlexRay block writes only the received number of bytes and will not change the trailing bytes in the message buffer data field of the receive shadow buffer. The same holds for the message buffer segment 2 with $MBDSR.MBSEG2DS$.

30.6.6.3.4 Received Message Access

To access the message data received over the FlexRay bus, the application reads the message data stored in the message buffer data field of the corresponding receive message buffer. The access to the message buffer data field is described in [Section 30.6.3.1, “Individual Message Buffers”](#).

The application can read the message buffer data field if the receive message buffer is one of the states HDis, HDisLck, or HLck. If the message buffer is in one of these states, the FlexRay block will not change the content of the message buffer.

30.6.6.3.5 Receive Shadow Buffers Concept

The receive shadow buffer concept applies only to individual receive message buffers. The intention of this concept is to ensure that only syntactically and semantically valid received non-null frames are

presented to the application in a receive message buffer. The basic structure of a receive shadow buffer is described in [Section 30.6.3.2, “Receive Shadow Buffers”](#).

The receive shadow buffers temporarily store the received frame header and message data. After the slot boundary the slot status information is generated. If the slot status information indicates the reception of the valid non-null frame (see [Table 30-102](#)), the FlexRay block writes the slot status into the slot status field of the receive shadow buffer and exchanges the content of the [Message Buffer Index Registers \(MBIDXR_n\)](#) with the content of the corresponding internal shadow buffer index register. In all other cases, the FlexRay block writes the slot status into the identified receive message buffer, depending on the slot status and the FlexRay segment the message buffer is assigned to.

The shadow buffer concept, with its index exchange, results in the fact that the FRM located message buffer associated to an individual receive message buffer changes after successful reception of a valid frame. This means that the message buffer area in the FRM accessed by the application for reading the received message is different from the initial setting of the message buffer. Therefore, the application must not rely on the index information written initially into the [Message Buffer Index Registers \(MBIDXR_n\)](#). Instead, the index of the message buffer header field must be fetched from the [Message Buffer Index Registers \(MBIDXR_n\)](#).

30.6.6.4 Double Transmit Message Buffer

The section provides a detailed description of the functionality of the double transmit message buffers.

Double transmit message buffers are used by the application to provide the FlexRay block with the message data to be transmitted over the FlexRay Bus. The FlexRay block uses this message buffer to provide information to the application about the transmission process, and status information about the slot in which message data was transmitted.

In contrast to the single transmit message buffers, the application can provide new transmission data while the transmission of the previously provided message data is running. This scheme is called double buffering and can be considered as a FIFO of depth 2.

Double transmit message buffers are implemented by combining two individual message buffers that form the two sides of an double transmit message buffer. One side is called the *commit side* and will be accessed by the application to provide the message data. The other side is called the *transmit side* and is used by the FlexRay block to transmit the message data to the FlexRay bus. The two sides are located in adjacent individual message buffers. The message buffer that implements the commit side has an even message buffer number $2n$. The transmit side message buffer follows the commit side message buffer and has the message buffer number $2n+1$. The basic structure and data flow of a double transmit message buffer is given in [Figure 30-123](#).

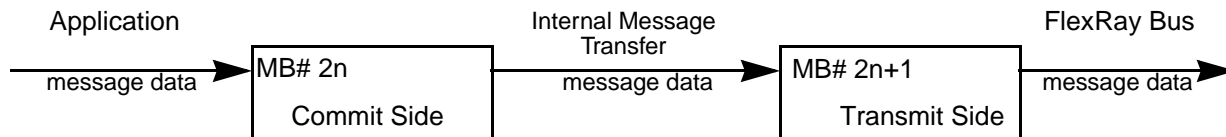


Figure 30-123. Double Transmit Buffer Structure and Data Flow

NOTE

Both the commit and the transmit side must be configured with identical values except for the [Message Buffer Index Registers \(MBIDXRn\)](#).

30.6.6.4.1 Access Regions

To certain message buffer fields, both the application and the FlexRay block have access. To ensure data consistency, a message buffer locking scheme is implemented, which controls the exclusive access to the data, control, and status bits of the message buffer.

The access scheme for double transmit message buffers is depicted in [Figure 30-124](#). The given regions represent fields that can be accessed from both the application and the FlexRay block and, thus, require access restrictions. A description of the regions is given in [Table 30-103](#).

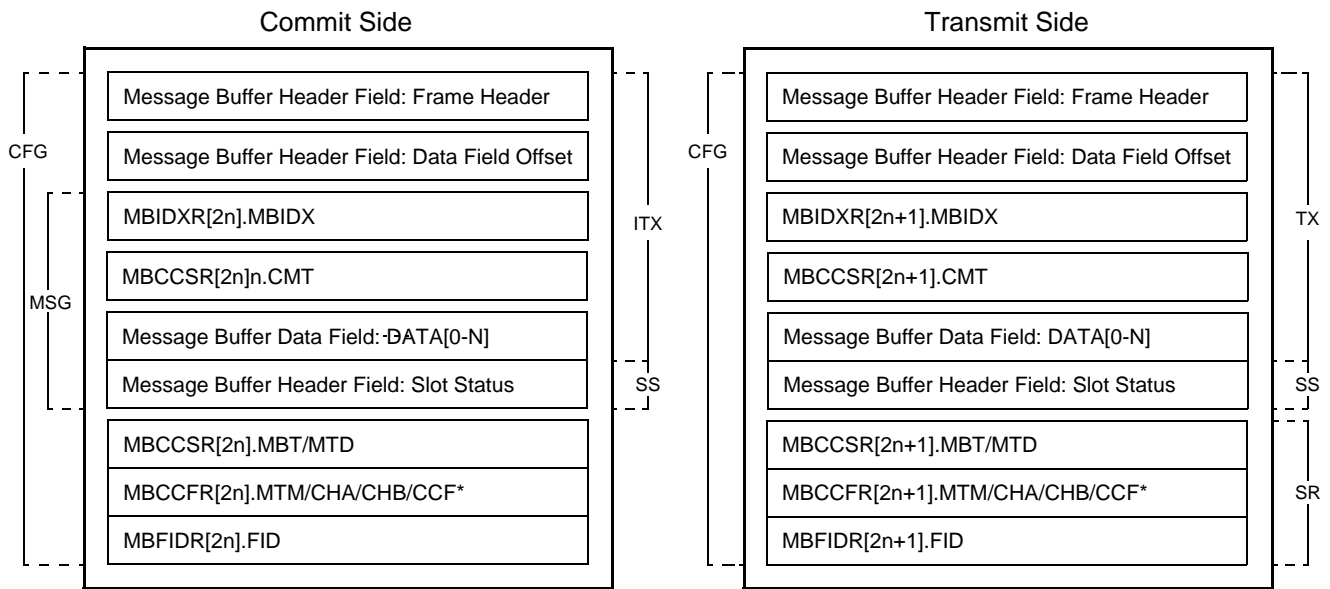


Figure 30-124. Double Transmit Message Buffer Access Regions Layout

Table 30-103. Double Transmit Message Buffer Access Regions Description

Access			Description
Region	Type		
	Application	Module	
Commit Side			
CFG	read/write	-	Message Buffer Configuration
MSG	read/write	-	Message Buffer Data and Control access
ITX	-	read/write	Internal Message Transfer.
SS	-	write-only	Slot Status Update
Transmit Side			
CFG	read/write	-	Message Buffer Configuration
SR	-	read-only	Message Buffer Search
TX	-	read-only	Internal Message Transfer, Message Transmission

Table 30-103. Double Transmit Message Buffer Access Regions Description

Access			Description
Region	Type		
	Application	Module	
SS	-	write-only	Slot Status Update

The trigger bits MBCCSRn.EDT and MBCCSRn.LCKT, and the interrupt enable bit MBCCSRn.MBIE are not under access control and can be accessed from the application at any time. The status bits MBCCSRn.EDS and MBCCSRn.LCKS are not under access control and can be accessed from the FlexRay block at any time.

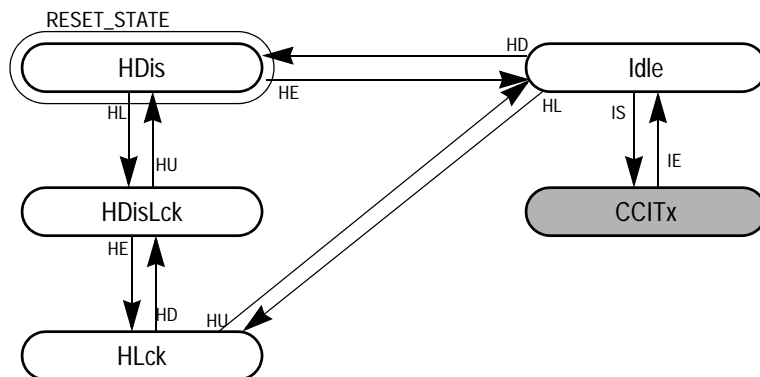
The interrupt flag MBCCSnR.MBIF is not under access control and can be accessed from the application and the FlexRay block at any time. FlexRay block set access has higher priority.

The FlexRay block restricts its access to the regions, depending on the current state of the corresponding part of the double transmit message buffer. The application must adhere to these restrictions in order to ensure data consistency. The states for the commit side of a double transmit message buffer are given in [Figure 30-125](#). A description of the states is given in [Table 30-105](#). The states for the transmit side of a double transmit message buffer are given in [Figure 30-126](#). A description of the states is given in [Table 30-105](#). The description tables also provide the access scheme for the access regions.

The status bits MBCCSRn.EDS and MBCCSRn.LCKS provide the application with the required message buffer status information. The internal status information is not visible to the application.

30.6.6.4.2 Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

**Figure 30-125. Double Transmit Message Buffer State Diagram (Commit Side)**

A description of the states of the commit side of a double transmit message buffer is given in [Table 30-104](#).

Table 30-104. Double Transmit Message Buffer State Description (Sheet 1 of 2)(Commit Side)

State	MBCCSR[2n]		Access Region		Description
	EDS	LCKS	Appl.	Module	
common states					

Table 30-104. Double Transmit Message Buffer State Description (Sheet 2 of 2)(Commit Side)

State	MBCCSR[2n]		Access Region		Description
	EDS	LCKS	Appl.	Module	
HDis	0	0	CFG	–	Disabled - Message Buffer under configuration. Commit Side can <i>not</i> be used for internal message transfer.
CCITx	1	0	–	ITX	Internal Message Transfer - Message Buffer Data transferred from commit side to transmit side.
commit side specific states					
Idle	1	0	–	ITX, SS	Idle - Message Buffer Commit Side is idle. Commit Side can be used for internal message transfer.
HDisLck	0	1	CFG	SS	Disabled and Locked - Message Buffer under configuration. Commit Side can <i>not</i> be used for internal message transfer.
HLck	1	1	MSG	SS	Locked - Applications access to data, control, and status. Commit Side can <i>not</i> be used for internal message transfer.

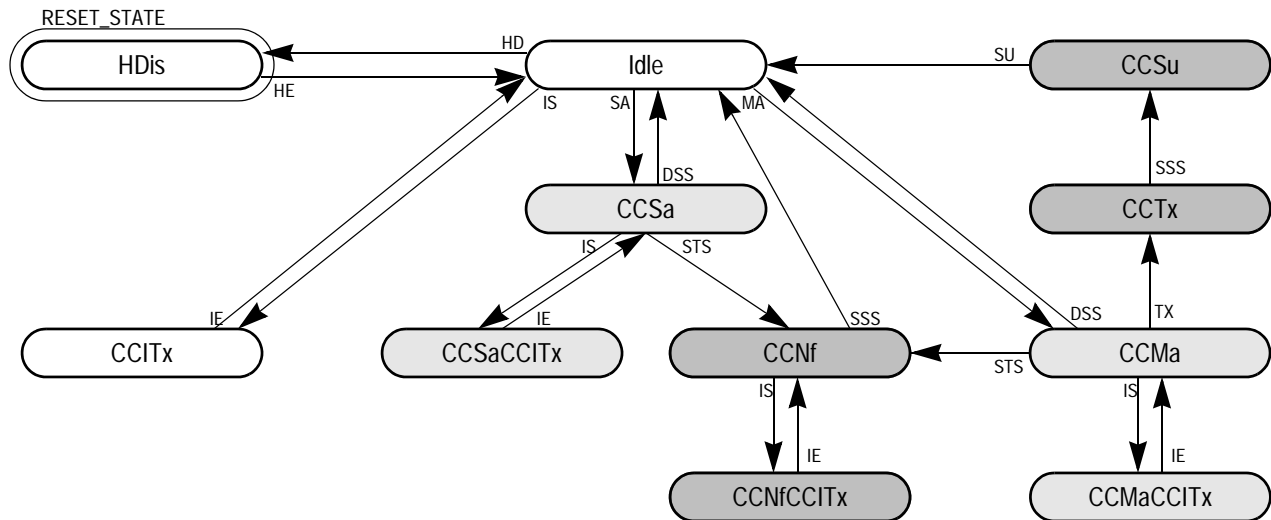


Figure 30-126. Double Transmit Message Buffer State Diagram (Transmit Side)

A description of the states of the transmit side of a double transmit message buffer is given in [Table 30-105](#).

Table 30-105. Double Transmit Message Buffer State Description (Transmit Side) (Sheet 1 of 2)

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
common states					
HDis	0	0	CFG	–	Disabled - Message Buffer under configuration. Excluded from message buffer search.
CCITx	1	0	–	TX	Internal Message Transfer - Message Buffer Data transferred from commit side to transmit side.
transmit side specific states					

Table 30-105. Double Transmit Message Buffer State Description (Transmit Side) (Sheet 2 of 2)

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	–	SR	Idle - Message Buffer Transmit Side is idle. Transmit Side is included in message buffer search.
CCSa	1	0	–	–	Slot Assigned - Message buffer assigned to next static slot. Ready for Null Frame transmission.
CCSaCCITx	1	0	–	TX	Slot Assigned and Internal Message Transfer - Message buffer assigned to next static slot and Message Buffer Data transferred from commit side to transmit side.
CCNf	1	0	–	TX	Null Frame Transmission Header is used for null frame transmission.
CCNfCCITx	1	0	–	TX	Null Frame Transmission and Internal Message Transfer - Header is used for null frame transmission and Message Buffer Data transferred from commit side to transmit side.
CCMa	1	0	–	–	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
CCMaCCITx	1	0	–	–	Message Available and Internal Message Transfer - Message buffer is assigned to next slot and cycle counter filter matches and Message Buffer Data transferred from commit side to transmit side.
CCTx	1	0	–	TX	Message Transmission - Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	–	SS	Status Update - Message buffer status update. Update of status flags, the slot status field, and the header index. Note: The slot status field of the commit side is updated too, even if the application has locked the commit side.

30.6.6.4.3 Message Buffer Transitions

Application Transitions

The application transitions that can be triggered by the application using the commands described in [Table 30-106](#). The application issues the commands by writing to the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#). Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

The enable and disable commands can be issued on the transmit side only. Any enable or disable command issued on the commit side will be ignored without notification. The transitions that will be triggered depends on the value of the EDS bit. The enable and disable commands will affect both the commit side and the transmit side at the same time. If the application triggers the disable transition HD while the transmit side is in one of the states CCSa, CCSaCCITx, CCNf, CCNfCCITx, CCMa, CCMaCCITx, CCTx, or CCSu, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

The lock and unlock commands can be issued on the commit side only. Any lock or unlock command issued on the transmit side will be ignored and the double transmit buffer lock error flag DBL_EF in the [CHI Error Flag Register \(CHIERFR\)](#) will be set. The transitions that will be triggered depends on the current value of the LCKS bit. The lock and unlock commands will only affect the commit side. If the application triggers the lock transition HL while the commit side is in the state CCITx, the message buffer

state will not be changed and the message buffer lock error flag LCK_EF in the [CHI Error Flag Register \(CHIERFR\)](#) will be set.

Table 30-106. Double Transmit Message Buffer Host Transitions

Transition	Host Command	Condition	Description
HE	MBCCSR[2n+1].EDT:= 1	MBCCSR[2n+1].EDS = 0	Application triggers message buffer enable.
HD		MBCCSR[2n+1].EDS = 1	Application triggers message buffer disable.
HL	MBCCSR[2n].LCKT:= 1	MBCCSR[2n].LCKS = 0	Application triggers message buffer lock.
HU		MBCCSR[2n].LCKS = 1	Application triggers message buffer unlock.

Module Transitions

The module transitions that can be triggered by the FlexRay block are described in [Table 30-107](#). The transitions C1 and C2 apply to both sides of the message buffer and are applied at the same time. All other FlexRay block transitions apply to the transmit side only.

Table 30-107. Double Transmit Message Buffer Module Transitions

Transition	Condition	Description
common transitions		
IS	see Section 30.6.6.4.5 , "Internal Message Transfer"	Internal Message Transfer Start - Start transfer of message data from commit side to transmit side.
IE		Internal Message Transfer End - Stop transfer of message data from commit side to transmit side. Note: The internal message transfer is stopped before the slot or segment start.
transmit side specific transitions		
SA	slot match and static slot	Slot Assigned - Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and MBCCSR[2n+1].CMT = 1	Transmission Slot Start - Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated - Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start - Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start - Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of static slot or dynamic slot or symbol window or NIT.

Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of [Table 30-108](#), the module transitions have a higher priority than the application transitions. The priorities among the FlexRay block transitions and the related states are given in the second part of [Table 30-108](#). These priorities apply only to the transmit side. The internal message transmit start transition IS has the lowest priority.

Table 30-108. Double Transmit Message Buffer Transition Priorities

State	Priority	Description
module vs. application		
Idle	IS > HD	Internal Message Transfer Start > Message Buffer Disable
	IS > HL	Internal Message Transfer Start > Message Buffer Lock
module internal		
Idle	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS	Transmission Slot Start > Static Slot Start
	TX > DSS	Transmission Slot Start > Dynamic Slot Start

30.6.6.4.4 Message Preparation

The application provides the message data through the commit side. The transmission itself is executed from the transmit side. The transfer of the message data from the commit side to the transmit side is done by the *Internal Message Transfer*, which is described in [Section 30.6.6.4.5, “Internal Message Transfer](#)

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field of the commit side and sets the commit bit CMT in the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#). The physical access to the message buffer data field is described in [Section 30.6.3.1, “Individual Message Buffers”](#).

As indicated by [Table 30-104](#), the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, or HLck. The application can change the state of a message buffer if it issues the appropriate commands shown in [Table 30-106](#). The state change is indicated through the MBCCSRn.EDS and MBCCSRn.LCKS status bits.

30.6.6.4.5 Internal Message Transfer

The internal message transfer transfers the message data from the commit side to the transmit side. The internal message transfer is implemented as the swapping of the content of the [Message Buffer Index Registers \(MBIDXRn\)](#) of the commit side and the transmit side. After the swapping, the commit side CMT bit is cleared, the commit side interrupt flag MBIF is set, the transmit side CMT bit is set, and the transmit side DVAL bit is cleared.

The conditions and the point in time when the internal message transfer is started are controlled by the message buffer commit mode bit MCM in the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#). The MCM bit configures the message buffer for either the streaming commit mode or the immediate commit mode. A detailed description is given in [Streaming Commit Mode](#) and [Immediate Commit Mode](#). The Internal Message Transfer is triggered with the transition IS. Both sides of the message buffer enter one of the CCITx states. The internal message transfer is finished with the transition IE.

Streaming Commit Mode

The intention of the streaming commit mode is to ensure that each committed message is transmitted *at least once*. The FlexRay block will not start the Internal Message Transfer for a message buffer as long as the message data on the transmit side is not transmitted at least once.

The streaming commit mode is configured by clearing the message buffer commit mode bit MCM in the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#).

In this mode, the internal message transfer from the commit side to the transmit side is started for a double transmit message buffer when all of the following conditions are fulfilled

1. the commit side is in the Idle state
2. the commit site message data are valid, i.e. $MBCCSR[2n].CMT = 1$
3. the transmit side is in one of the states Idle, CCSa, or CCMa
4. the transmit side contains either no valid message data, i.e. $MBCCSR[2n+1].CMT = 0$ or the message data were transmitted at least once, i.e. $MBCCSR[2n+1].DVAL = 1$

An example of a streaming commit mode state change diagram is given in [Figure 30-127](#). In this example, both the commit and the transmit side do not contain valid message data and the application provides two messages. The message buffer does not match the next slot.

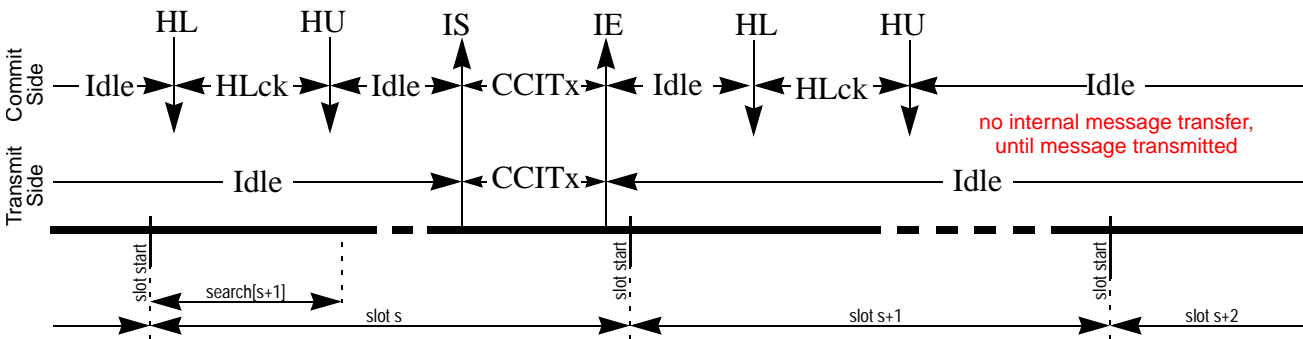


Figure 30-127. Internal Message Transfer in Streaming Commit Mode

Immediate Commit Mode

The intention of the immediate commit mode is to transmit the *latest* data provided by the application. This implies that it is not guaranteed that each provided message will be transmitted at least once.

The immediate commit mode is configured by setting the message buffer commit mode bit MCM in the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#).

In this mode, the internal message transfer from the commit side to the transmit side is started for one double transmit message buffer when all of the following conditions are fulfilled

1. the commit side is in the Idle state
2. the commit site message data are valid, i.e. $MBCCSR[2n].CMT = 1$
3. the transmit side is in one of the states Idle, CCSa, or CCMa

It is not checked whether the transmit side contains no valid message data or valid message data were transmitted at least once. If message data are valid and not transmitted, they may be overwritten.

An example of a streaming commit mode state change diagram is given in [Figure 30-128](#). In this example, both the commit and the transmit side do not contain valid message data, and the application provides two messages and the first message is gets overwritten. The message buffer does not match the next slot.

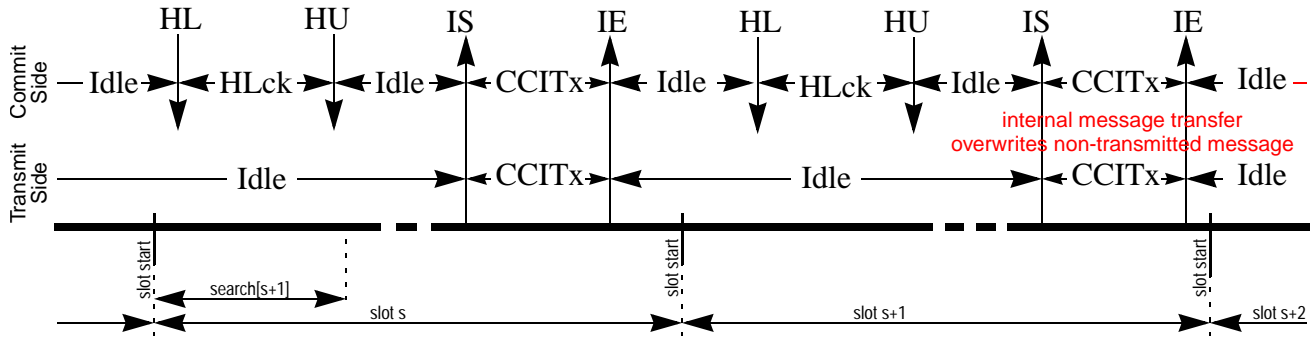


Figure 30-128. Internal Message Transfer in Immediate Commit Mode

30.6.6.4.6 Message Transmission

For double transmit message buffers, the message buffer search checks only the transmit side part. The internal scheduling ensures, that the internal message transfer is stopped on the message buffer search start. Thus, the transmit side of message buffer, that is not in its transmission or status update slot, is always in the Idle state.

The message transmit behavior and transmission state changes of the transmit side of a double transmit message buffer are the same as for single buffered transmit buffers, except that the transmit side of double buffers can not be locked by the application, i.e. the HU and HL transition do not exist. Therefore, refer to [Section 30.6.6.2.5, “Message Transmission”](#)

30.6.6.4.7 Message Buffer Status Update

The message buffer status update behavior of the transmit side of a double transmit message buffer is the same as for single transmit message buffers which is described in [Section 30.6.6.2.7, “Message Buffer Status Update”](#).

Additionally, the slot status field of the commit side is update after the update of the slot status field of the transmit side, even if the commit side is locked by the application. This is implemented to provide the slot status of the most recent transmission slot.

30.6.7 Individual Message Buffer Search

This section provides a detailed description of the message buffer search algorithm.

The message buffer search determines for each enabled channel if a slot s in a communication cycle c is assigned for frame or null frame transmission or if it is subscribed for frame reception on that channel.

The message buffer search is a sequential algorithm which is invoked at the following protocol related events:

1. NIT start
2. slot start in the static segment
3. minislot start in the dynamic segment

The message buffer search within the NIT searches for message buffers assigned or subscribed to slot 1. The message buffer search within slot n searches for message buffers assigned or subscribed to slot $n+1$.

In general, the message buffer search for the next slot n considers only message buffers which are

1. enabled, i.e. $MBCCSRn.EDS = 1$, and
2. matches the next slot n , i.e. $MBFIDRn.FID = n$, and
3. are the transmit side buffer in case of a double transmit message buffer.

On top of that, for the static segment only those message buffers are considered, that match the condition of at least one row of [Table 30-109](#). For the dynamic segment only those message buffers are considered, that match the condition of at least one row of [Table 30-110](#). These message buffers are called *matching* message buffers.

For each enabled channel the message buffer search may identify multiple *matching* message buffers. Among all matching message buffers the message buffers with highest priority according to [Table 30-109](#) for the static segment and according to [Table 30-110](#) for the dynamic segment are selected.

Table 30-109. Message Buffer Search Priority (static segment)

Priority	MTD	LCKS	CMT	CCFM ¹	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	1	-	0	1	transmit buffer, matches cycle count, not committed	SA
	1	1	-	1	transmit buffer, matches cycle count, locked	SA
2	1	-	-	-	transmit buffer	SA
3	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 4	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

¹ Cycle Counter Filter Match, see [Section 30.6.7.1, "Message Buffer Cycle Counter Filtering"](#)

Table 30-110. Message Buffer Search Priority (dynamic segment)

Priority	MTD	LCKS	CMT	CCFM ¹	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 2	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

¹ Cycle Counter Filter Match, see [Section 30.6.7.1, "Message Buffer Cycle Counter Filtering"](#)

If there are multiple message buffer with highest priority, the message buffer with the lowest message buffer number is selected. All message buffer which have the highest priority must have a consistent channel assignment as specified in [Section 30.6.7.2, "Message Buffer Channel Assignment Consistency"](#).

Depending on the message buffer channel assignment the same message buffer can be found for both channel A and channel B. In this case, this message buffer is used as described in [Section 30.6.3.1, "Individual Message Buffers"](#).

30.6.7.1 Message Buffer Cycle Counter Filtering

The message buffer cycle counter filter is a value-mask filter defined by the CCFE, CCFMSK, and CCFVAL fields in the [Message Buffer Cycle Counter Filter Registers \(MBCCFR_n\)](#). This filter determines a set of communication cycles in which the message buffer is considered for message reception or message transmission. If the cycle counter filter is disabled, i.e. CCFE = 0, this set of cycles consists of all communication cycles.

If the cycle counter filter of a message buffer does not match a certain communication cycle number, this message buffer is not considered for message transmission or reception in that communication cycle. In case of a transmit message buffer assigned to a slot in the static segment, though, this buffer is added to the matching message buffers to indicate the slot assignment and to trigger the null frame transmission.

The cycle counter filter of a message buffer matches the communication cycle with the number CYCCNT if at least one of the following conditions evaluates to true:

$$\text{MBCCFR}_n[\text{CCFE}] = 0 \quad \text{Eqn. 30-9}$$

$$\text{CYCCNT} \wedge \text{MBCCFR}_n[\text{CCFMSK}] = \text{MBCCFR}_n[\text{CCFVAL}] \wedge \text{MBCCFR}_n[\text{CCFMSK}] \quad \text{Eqn. 30-10}$$

30.6.7.2 Message Buffer Channel Assignment Consistency

The message buffer channel assignment given by the CHA and CHB bits in the [Message Buffer Cycle Counter Filter Registers \(MBCCFR_n\)](#) defines the channels on which the message buffer will receive or transmit. The message buffer with number *n* transmits or receives on channel A if MBCCFR_n[CHA] = 1 and transmits or receives on channel B if MBCCFR_n[CHB] = 1.

To ensure correct message buffer operation, all message buffers assigned to the same slot and with the same priority must have a *consistent* channel assignment. That means they must be either assigned to one channel only, or must be assigned to *both* channels. The behavior of the message buffer search is not defined, if both types of channel assignments occur for one slot and priority. An inconsistent channel assignment for message buffer 0 and message buffer 1 is depicted in [Figure 30-129](#).


MB0	MBFIDR0[FID] = 10	MBCCFR0[CHA] = 1, MBCCFR0[CHB] = 0		single channel assignment
MB1	MBFIDR1[FID] = 10	MBCCFR1[CHA] = 1, MBCCFR1[CHB] = 1		

Figure 30-129. Inconsistent Channel Assignment

30.6.7.3 Node Related Slot Multiplexing

The term *Node Related Slot Multiplexing* applies to the dynamic segment only and refers to the functionality if there are transmit as well as receive message buffers are configured for the same slot.

According to [Table 30-110](#) the transmit buffer is only found if the cycle counter filter matches, and the buffer is not locked and committed. In all other cases, the receive buffer will be found. Thus, if the block has no data to transmit in a dynamic slot, it is able to receive frames on that slot.

30.6.7.4 Message Buffer Search Error

If the message buffer search is running while the next message buffer search start event appears, the message buffer search is stopped and the Message Buffer Search Error Flag MSB_EF is set in the [CHI Error Flag Register \(CHIERFR\)](#). This appears only if the CHI frequency is too low to search through all message buffers within the NIT or a minislot. The message buffer result is not defined in this case. For more details see [Section 30.7.3, “Number of Usable Message Buffers”](#).

30.6.8 Individual Message Buffer Reconfiguration

The initial configuration of each individual message buffer can be changed even when the protocol is not in the *POC:config* state. This is referred to as individual message buffer *reconfiguration*. The configuration bits and fields that can be changed are given in the section on [Specific Configuration Data](#). The common configuration data given in the section on [Specific Configuration Data](#) can not be reconfigured when the protocol is out of the *POC:config* state.

30.6.8.1 Reconfiguration Schemes

Depending on the target and destination basic state of the message buffer that is to be reconfigured, there are three reconfiguration schemes.

30.6.8.1.1 Basic Type Not Changed (RC1)

A reconfiguration will not change the basic type of the individual message buffer, if both the message buffer transfer direction bit MBCCSn.MTD and the message buffer type bit MBCCSn.MBT are not changed. This type of reconfiguration is denoted by RC1 in [Figure 30-130](#). Single transmit and receive message buffers can be RC1-reconfigured when in the HDis or HDisLck state. Double transmit message buffers can be RC1-reconfigured if both the transmit side and the commit side are in the HDis state.

30.6.8.1.2 Buffer Type Not Changed (RC2)

A reconfiguration will not change the buffer type of the individual message buffer if the message buffer type bit MBCCSRn.MBT is not changed. This type of reconfiguration is denoted by RC2 in [Figure 30-130](#). It applies only to single transmit and receive message buffers. Single transmit and receive message buffers can be RC2-reconfigured when in the HDis or HDisLck state.

30.6.8.1.3 Buffer Type Changed (RC3)

A reconfiguration will change the buffer type of the individual message buffer if the message buffer type bit MBCCSRn.MBT is changed. This type of reconfiguration is denoted by RC3 in [Figure 30-130](#). The RC3 reconfiguration splits one double buffer into two single buffers or combines two single buffer into one double buffer. In the later case, the two single message buffers must have consecutive message buffer numbers and the smaller one must be even. Message Buffers can be RC3 reconfigured if they are in the HDis state.

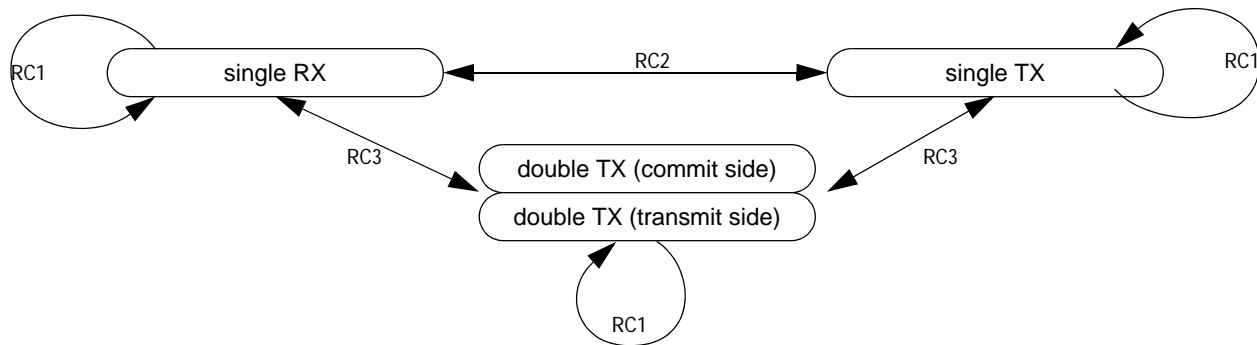


Figure 30-130. Message Buffer Reconfiguration Scheme

30.6.9 Receive FIFO

This section provides a detailed description of the two receive FIFOs.

30.6.9.1 Overview

The receive FIFOs implement the queued receive buffer defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. One receive FIFO is assigned to channel A, the other receive FIFO is assigned to channel B. Both FIFOs work completely independent from each other.

The message buffer structure of each FIFO is described in [Section 30.6.3.3, “Receive FIFO”](#). The area in the FRM for each of the two receive FIFOs is characterized by:

- The index of the first FIFO entry given by [Receive FIFO Start Index Register \(RFSIR\)](#)
- The number of FIFO entries and the length of each FIFO entry as given by [Receive FIFO Depth and Size Register \(RFDSR\)](#)

30.6.9.2 Receive FIFO Configuration

The receive FIFO control and configuration data are given in [Section 30.6.3.7, “Receive FIFO Control and Configuration Data”](#). The configuration of the receive FIFOs consists of two steps.

The first step is the allocation of the required amount of FRM for the FlexRay window. This includes the allocation of the message buffer header area and the allocation of the message buffer data fields. For more details see [Section 30.6.4, “FlexRay Memory Layout”](#).

The second step is the programming of the configuration data register while the PE is in *POC:config*.

The following steps configure the layout of the FIFO.

- The number of the first message buffer header index that belongs to the FIFO is written into the [Receive FIFO Start Index Register \(RFSIR\)](#).
- The depth of the FIFO is written into the FIFO_DEPTH field in the [Receive FIFO Depth and Size Register \(RFDSR\)](#).
- The length of the message buffer data field for the FIFO is written into the ENTRY_SIZE field in the [Receive FIFO Depth and Size Register \(RFDSR\)](#).

NOTE

To ensure, that the read index RDIDX always points to a message buffer that contains valid data, the receive FIFO must have at least 2 entries.

The FIFO filters are configured through the fifo filter registers.

30.6.9.3 Receive FIFO Reception

The frame reception to the receive FIFO is enabled, if for a certain slots no message buffer is assigned or subscribed. In this case the FIFO filter path shown in [Figure 30-131](#) is activated.

When the receive FIFO filter path indicates that the received frame must be appended to the FIFO, the FlexRay block writes the received frame header and slot status into the message buffer header field indicated by the internal FIFO header write index. The payload data are written in the message buffer data field. If the status of the received frame indicates a valid frame, the internal FIFO header write index is updated and the fifo not-empty interrupt flag FNEAIF/FNEBIF in the [Global Interrupt Flag and Enable Register \(GIFER\)](#) is set.

30.6.9.4 Receive FIFO Message Access

If the fifo not-empty interrupt flag FNEAIF/FNEBIF in the [Global Interrupt Flag and Enable Register \(GIFER\)](#) is set, the receive FIFO contains valid received messages, which can be accessed by the application.

The receive FIFO does not require locking to access the message buffers. To access the message the application first reads the receive FIFO read index RDIDX from the [Receive FIFO A Read Index Register \(RFARIR\)](#) or [Receive FIFO B Read Index Register \(RFBRIR\)](#), respectively. This index points to the message buffer header field of the next message buffer that contains valid data. The application can access the message data as described in [Section 30.6.3.3, “Receive FIFO”](#). When the application has read all message buffer data and status information, it writes 1 to the fifo not-empty interrupt flags FNEAIF or FNEBIF. This clears the interrupt flag and updates the RDIDX field in the [Receive FIFO A Read Index Register \(RFARIR\)](#) or [Receive FIFO B Read Index Register \(RFBRIR\)](#), respectively. When the RDIDX value has reached the last message buffer header field that belongs to the fifo, it wraps around to the index of the first message buffer header field that belongs to the fifo. This value is provided by the SIDX field in the [Receive FIFO Start Index Register \(RFSIR\)](#).

30.6.9.5 Receive FIFO filtering

The receive FIFO filtering is activated after all enabled individual receive message buffers have been searched without success for a message buffer to receive the current frame.

The FlexRay block provides three sets of FIFO filters. The FIFO filters are applied to valid non-null frames only. The FIFO will not receive invalid or null-frames. For each FIFO filter, the pass criteria is specified in the related section given below. Only frames that have passed all filters will be appended to the FIFO. The FIFO filter path is depicted in [Figure 30-131](#).

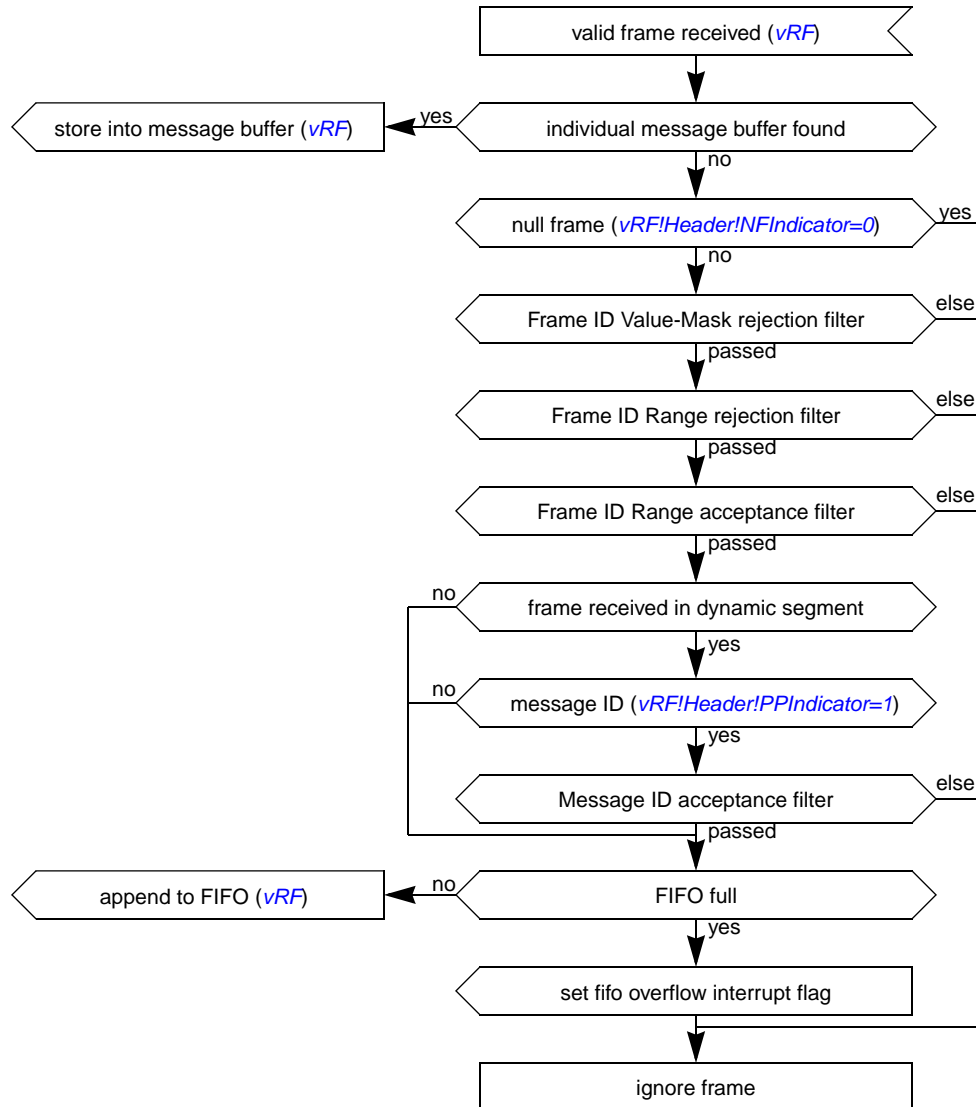


Figure 30-131. Received Frame FIFO Filter Path

A received frame passes the FIFO filtering if it has passed all three type of filter.

30.6.9.5.1 RX FIFO Frame ID Value-Mask Rejection Filter

The frame ID value-mask rejection filter is a value-mask filter and is defined by the fields in the [Receive FIFO Frame ID Rejection Filter Value Register \(RFFIDRFVR\)](#) and the [Receive FIFO Frame ID Rejection Filter Mask Register \(RFFIDRFMR\)](#). Each received frame with a frame ID FID that does not match the value-mask filter value passes the filter, i.e. is not rejected.

Consequently, a received valid frame with the frame ID FID passes the RX FIFO Frame ID Value-Mask Rejection Filter if [Equation 30-11](#) is fulfilled.

$$ID \wedge RFFIDRFMR[FIDRFMSK] \neq RFFIDRFVR[FIDRFVAL] \wedge RFFIDRFMR[FIDRFMSK] \quad \text{Eqn. 30-11}$$

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to pass all frames by the following settings.

- RFFIDRFVVR.FIDRFVAL:= 0x000 and RFFIDRFMR.FIDRFMSK:= 0x7FF

Using the settings above, only the frame with frame ID 0 will be rejected, which is an invalid frame. All other frames will pass.

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to reject all frames by the following settings.

- RFFIDRFMR.FIDRFMSK:= 0x000

Using the settings above, [Equation 30-11](#) can never be fulfilled ($0 \neq 0$) and thus all frames are rejected; no frame will pass. This is the reset value for the RX FIFO.

30.6.9.5.2 RX FIFO Frame ID Range Rejection Filter

Each of the four RX FIFO Frame ID Range filters can be configured as a rejection filter. The filters are configured by the [Receive FIFO Range Filter Configuration Register \(RFRFCFR\)](#) and controlled by the [Receive FIFO Range Filter Control Register \(RFRFCTR\)](#). The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range rejection filters if either no rejection filter is enabled, or, for all of the enabled RX FIFO Frame ID Range rejection filters, i.e. RFRFCTR.FiMD = 1 and RFRFCTR.FiEN = 1, [Equation 30-12](#) is fulfilled.

$$FID < RFRFCFR_{SEL}[SID_{IBD=0}] \text{ or } (RFRFCFR_{SEL}[SID_{IBD=1}] < FID) \quad \text{Eqn. 30-12}$$

Consequently, all frames with a frame ID that fulfills [Equation 30-13](#) for at least one of the enabled rejection filters will be rejected and thus not pass.

$$\exists RFRFCFR_{SEL}[SID_{IBD=0}] \leq FID \leq RFRFCFR_{SEL}[SID_{IBD=1}] \quad \text{Eqn. 30-13}$$

30.6.9.5.3 RX FIFO Frame ID Range Acceptance filter

Each of the four RX FIFO Frame ID Range filters can be configured as an acceptance filter. The filters are configured by the [Receive FIFO Range Filter Configuration Register \(RFRFCFR\)](#) and controlled by the [Receive FIFO Range Filter Control Register \(RFRFCTR\)](#). The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range acceptance filters if either no acceptance filter is enabled, or, for at least one of the enabled RX FIFO Frame ID Range acceptance filters, i.e. RFRFCTR.FiMD = 0 and RFRFCTR.FiEN = 1, [Equation 30-14](#) is fulfilled.

$$\exists RFRFCFR_{SEL}[SID_{IBD=0}] \leq FID \leq RFRFCFR_{SEL}[SID_{IBD=1}] \quad \text{Eqn. 30-14}$$

30.6.9.5.4 RX FIFO Message ID Acceptance Filter

The RX FIFO Message ID Acceptance Filter is a value-mask filter and is defined by the [Receive FIFO Message ID Acceptance Filter Value Register \(RFMIDAFVR\)](#) and the [Receive FIFO Message ID Acceptance Filter Mask Register \(RFMIAFMR\)](#). This filter applies only to valid frames received in the dynamic segment with the payload preamble indicator bit PPI set to 1. All other frames will pass this filter.

A received valid frame in the dynamic segment with the payload preamble indicator bit PPI set to 1 and with the message ID MID (the first two bytes of the payload) will pass the RX FIFO Message ID Acceptance Filter if [Equation 30-15](#) is fulfilled.

$$MID \wedge RFMIDAFMR[MIDAFMSK] = RFMIDAFMR[MIDAFVAL] \wedge RFMIDAFMR[MIDAFMSK] \quad \text{Eqn. 30-15}$$

The RX FIFO Message ID Acceptance Filter can be configured to accept all frames by setting

- `RFMIDAFMR.MIDAFMSK := 0x000`

Using the settings above, [Equation 30-15](#) is always fulfilled and all frames will pass.

30.6.10 Channel Device Modes

This section describes the two FlexRay channel device modes that are supported by the FlexRay block.

30.6.10.1 Dual Channel Device Mode

In the dual channel device mode, both FlexRay ports are connected to physical FlexRay bus lines. The FlexRay port consisting of `FR_A_RX`, `FR_A_TX`, and `FR_A_TX_EN` is connected to the physical bus channel A and the FlexRay port consisting of `FR_B_RX`, `FR_B_TX`, and `FR_A_TX_EN` is connected to the physical bus channel B. The dual channel system is shown in [Figure 30-132](#).

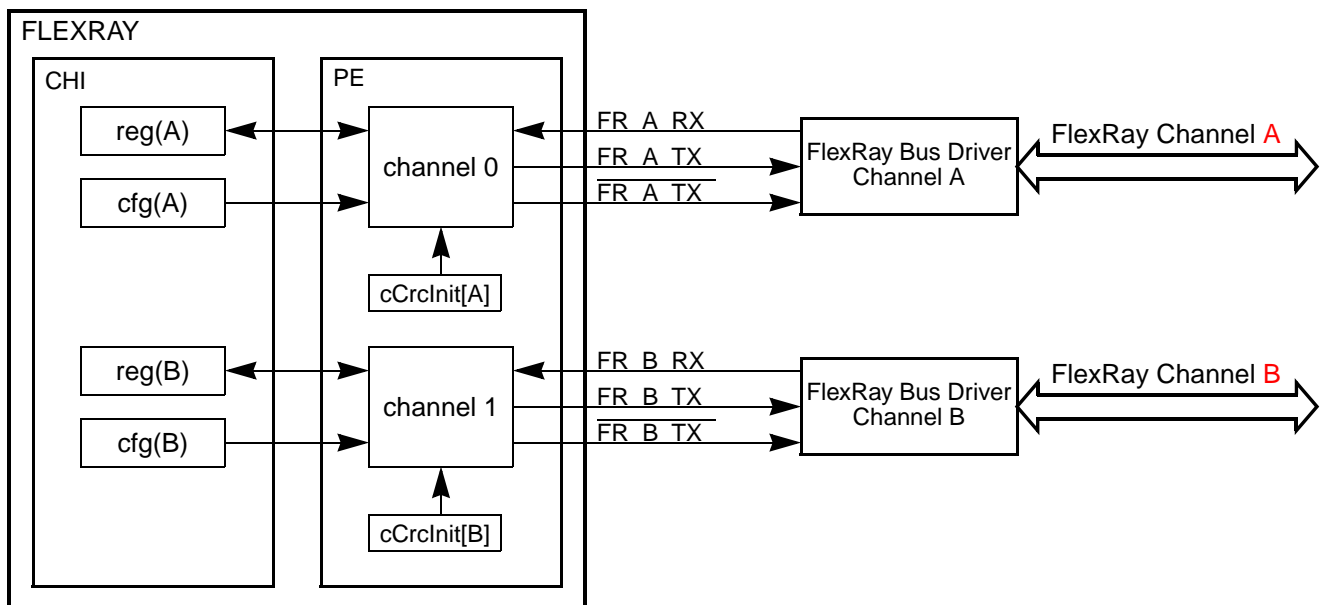


Figure 30-132. Dual Channel Device Mode

30.6.10.2 Single Channel Device Mode

The single channel device mode supports devices that have only one FlexRay port available. This FlexRay port consists of the signals `FR_A_RX`, `FR_A_TX`, and `FR_A_TX_EN` and can be connected to either the physical bus channel A (shown in [Figure 30-133](#)) or the physical bus channel B (shown in [Figure 30-134](#)).

If the device is configured as a single channel device by setting MCR.SCD to 1, only the internal channel A and the FlexRay Port A is used. Depending on the setting of MCR.CHA and MCR.CHB, the internal channel A behaves either as a FlexRay Channel A or FlexRay Channel B. The bit MCR.CHA must be set, if the FlexRay Port A is connected to a FlexRay Channel A. The bit MCR.CHB must be set if the FlexRay Port A is connected to a FlexRay Channel B. The two FlexRay channels differ only in the initial value for the frame CRC *cCrclnit*. For a single channel device, the application can access and configure only the registers related to internal channel A.

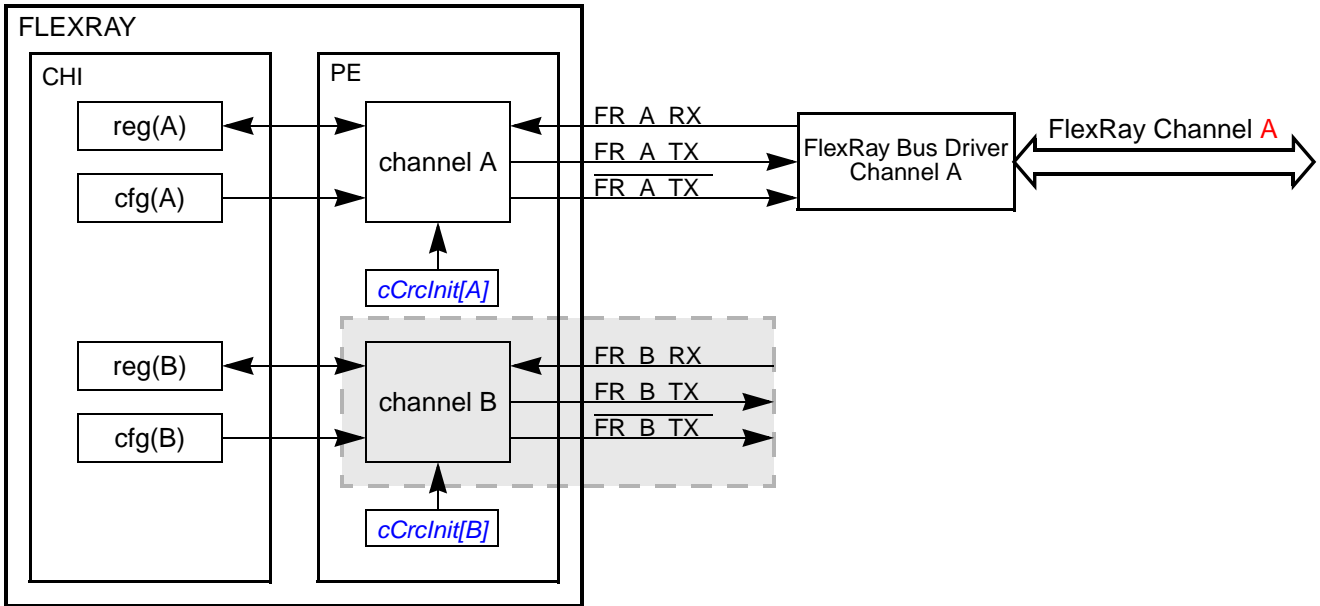


Figure 30-133. Single Channel Device Mode (Channel A)

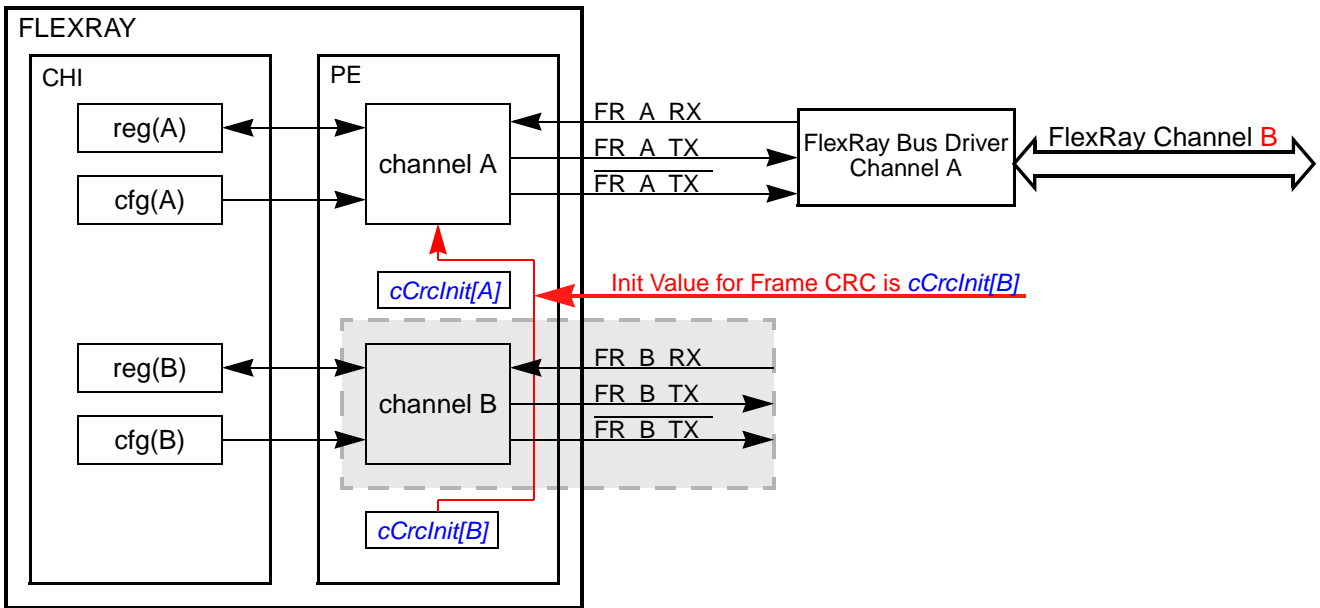


Figure 30-134. Single Channel Device Mode (Channel B)

30.6.11 External Clock Synchronization

The application of the external rate and offset correction is triggered when the application writes to the EOC_AP and ERC_AP fields in the **Protocol Operation Control Register (POCR)**. The PE applies the external correction values in the next even-odd cycle pair as shown in [Figure 30-135](#) and [Figure 30-136](#).

NOTE

The values provided in the EOC_AP and ERC_AP fields are the values that were written from the application most recently. If these value were already applied, they will not be applied in the current cycle pair again.

If the offset correction applied in the NIT of cycle $2n+1$ shall be affect by the external offset correction, the EOC_AP field must be written to after the start of cycle $2n$ and before the end of the static segment of cycle $2n+1$. If this field is written to after the end of the static segment of cycle $2n+1$, it is not guaranteed that the external correction value is applied in cycle $2n+1$. If the value is not applied in cycle $2n+1$, then the value will be applied in the cycle $2n+3$. Refer to [Figure 30-135](#) for timing details.

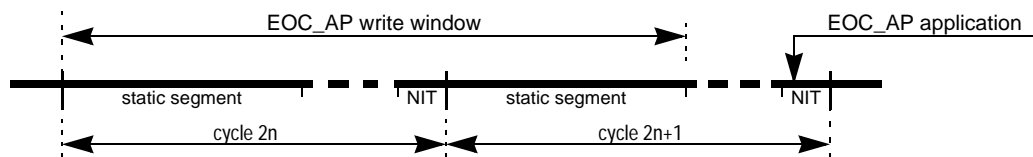


Figure 30-135. External Offset Correction Write and Application Timing

If the rate correction for the cycle pair $[2n+2, 2n+3]$ shall be affect by the external offset correction, the ERC_AP field must be written to after the start of cycle $2n$ and before the end of the static segment start of cycle $2n+1$. If this field is written to after the end of the static segment of cycle $2n+1$, it is not guaranteed that the external correction value is applied in cycle pair $[2n+2, 2n+3]$. If the value is not applied for cycle pair $[2n+2, 2n+3]$, then the value will be applied for cycle pair $[2n+4, 2n+5]$. Refer to [Figure 30-136](#) for details.

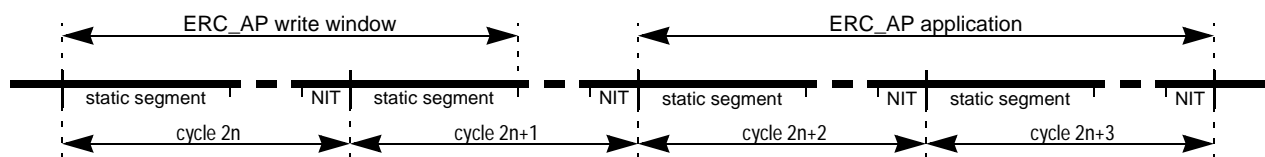


Figure 30-136. External Rate Correction Write and Application Timing

30.6.12 Sync Frame ID and Sync Frame Deviation Tables

The FlexRay protocol requires the provision of a snapshot of the Synchronization Frame ID tables for the even and odd communication cycle for both channels. The FlexRay block provides the means to write a copy of these internal tables into the FRM and ensures application access to consistent tables by means of table locking. Once the application has locked the table successfully, the FlexRay block will not overwrite these tables and the application can read a consistent snapshot.

NOTE

Only synchronization frames that have passed the synchronization frame filters are considered for clock synchronization and appear in the sync frame tables.

30.6.12.1 Sync Frame ID Table Content

The Sync Frame ID Table is a snapshot of the protocol related variables *vsSyncIdListA* and *vsSyncIdListB* for each even and odd communication cycle. This table provides a list of the frame IDs of the synchronization frames received on the corresponding channel and cycle that are used for the clock synchronization.

30.6.12.2 Sync Frame Deviation Table Content

The Sync Frame Deviation Table is a snapshot of the protocol related variable *zsDev(id)(oe)(ch)!Value*. Each Sync Frame Deviation Table entry provides the deviation value for the sync frame, with the frame ID presented in the corresponding entry in the Sync Frame ID Table.

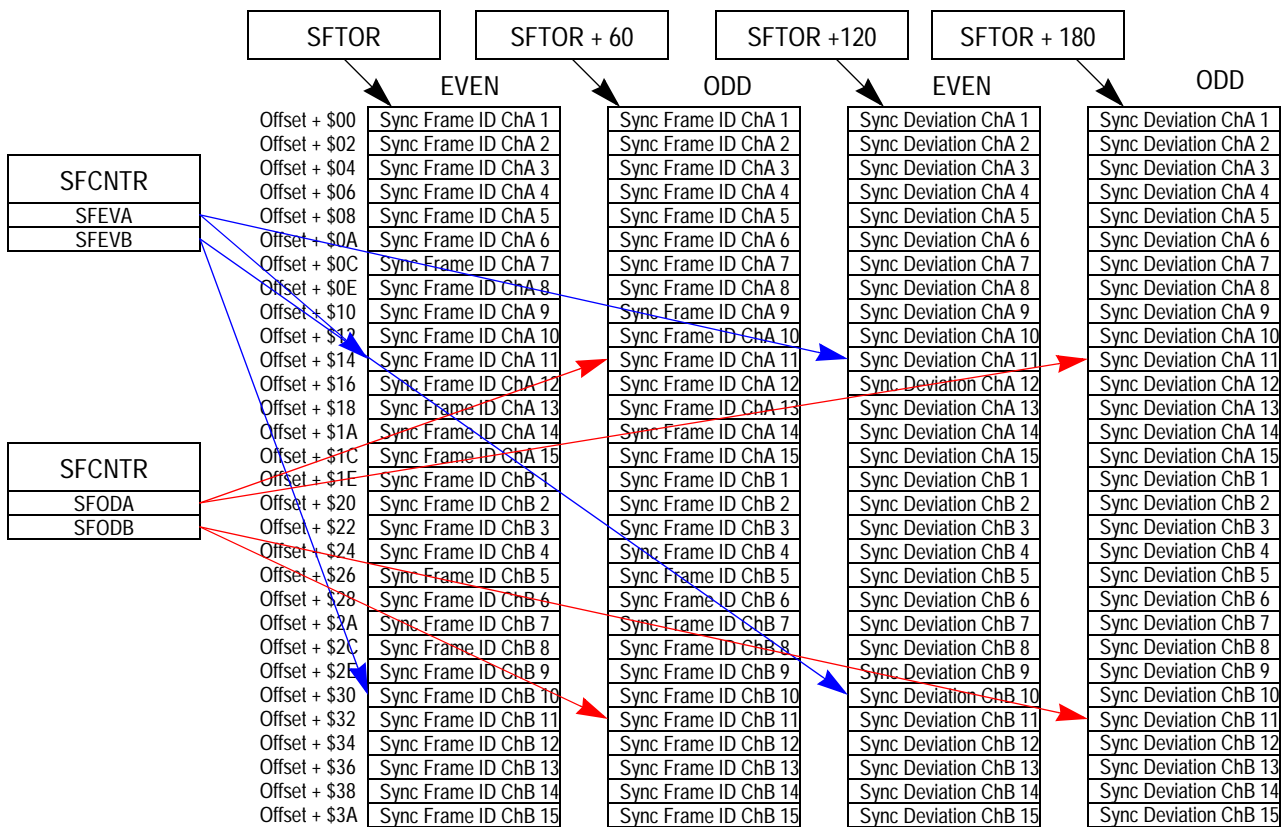


Figure 30-137. Sync Table Memory Layout

30.6.12.3 Sync Frame ID and Sync Frame Deviation Table Setup

The FlexRay block writes a copy of the internal synchronization frame ID and deviation tables into the FRM if requested by the application. The application must provide the appropriate amount of FRM for the tables. The memory layout of the tables is given in [Figure 30-137](#). Each table occupies 120 16-bit entries.

While the protocol is in *POC:config* state, the application must program the offsets for the tables into the [Sync Frame Table Offset Register \(SFTOR\)](#).

30.6.12.4 Sync Frame ID and Sync Frame Deviation Table Generation

The application controls the generation process of the Sync Frame ID and Sync Frame Deviation Tables into the FRM using the [Sync Frame Table Configuration, Control, Status Register \(SFTCCSR\)](#). A summary of the copy modes is given in [Table 30-111](#).

Table 30-111. Sync Frame Table Generation Modes

SFTCCSR			Description
OPT	SDVEN	SIDEN	
0	0	0	No Sync Frame Table copy
0	0	1	Sync Frame ID Tables will be copied continuously
0	1	0	Reserved
0	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables will be copied continuously
1	0	0	No Sync Frame Table copy
1	0	1	Sync Frame ID Tables for next even-odd-cycle pair will be copied
0	1	0	Reserved
1	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables for next even-odd-cycle pair will be copied

The Sync Frame Table generation process is described in the following for the even cycle. The same sequence applies to the odd cycle.

If the application has enabled the sync frame table generation by setting SFTCCSR.SIDEN to 1, the FlexRay block starts the update of the even cycle related tables after the start of the NIT of the next even cycle. The FlexRay block checks if the application has locked the tables by reading the SFTCCSR.ELKS lock status bit. If this bit is set, the FlexRay block will not update the table in this cycle. If this bit is cleared, the FlexRay block locks this table and starts the table update. To indicate that these tables are currently updated and may contain inconsistent data, the FlexRay block clears the even table valid status bit SFTCCSR.EVAL. Once all table entries related to the even cycle have been transferred into the FRM, the FlexRay block sets the even table valid bit SFTCCSR.EVAL and the Even Cycle Table Written Interrupt Flag EVT_IF in the [Protocol Interrupt Flag Register 1 \(PIFR1\)](#). If the interrupt enable flag EVT_IE is set, an interrupt request is generated.

To read the generated tables, the application must lock the tables to prevent the FlexRay block from updating these tables. The locking is initiated by writing a 1 to the even table lock trigger SFTCCSR.ELKT. When the even table is not currently updated by the FlexRay block, the lock is granted and the even table lock status bit SFTCCSR.ELKS is set. This indicates that the application has successfully locked the even sync tables and the corresponding status information fields SFRA, SFRB in

the [Sync Frame Counter Register \(SFCNTR\)](#). The value in the SFTCCSR.CYCNUM field provides the number of the cycle that this table is related to.

The number of available table entries per channel is provided in the SFCNTR.SFEVA and SFCNTR.SFEVB fields. The application can now start to read the sync table data from the locations given in [Figure 30-137](#).

After reading all the data from the locked tables, the application must unlock the table by writing to the even table lock trigger SFTCCSR.ELKT again. The even table lock status bit SFTCCSR.ELKS is reset immediately.

If the sync frame table generation is disabled, the table valid bits SFTCCSR.EVAL and SFTCCSR.EVAL are reset when the counter values in the [Sync Frame Counter Register \(SFCNTR\)](#) are updated. This is done because the tables stored in the FRM are no longer related to the values in the [Sync Frame Counter Register \(SFCNTR\)](#).

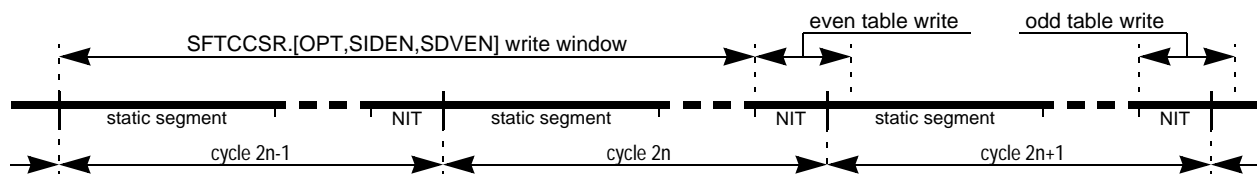


Figure 30-138. Sync Frame Table Trigger and Generation Timing

30.6.12.5 Sync Frame Table Access

The sync frame tables will be transferred into the FRM during the table write windows shown in [Figure 30-138](#). During the table write, the application can not lock the table that is currently written. If the application locks the table outside of the table write window, the lock is granted immediately.

30.6.12.5.1 Sync Frame Table Locking and Unlocking

The application locks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT in the [Sync Frame Table Configuration, Control, Status Register \(SFTCCSR\)](#). If the affected table is not currently written to the FRM, the lock is granted immediately, and the lock status bit ELKS/OLKS is set. If the affected table is currently written to the FRM, the lock is not granted. In this case, the application must issue the lock request again until the lock is granted.

The application unlocks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT. The lock status bit ELKS/OLKS is cleared immediately.

30.6.13 MTS Generation

The FlexRay block provides a flexible means to request the transmission of the Media Access Test Symbol MTS in the symbol window on channel A or channel B.

The application can configure the set of communication cycles in which the MTS will be transmitted over the FlexRay bus by programming the CYCCNTMSK and CYCCNTVAL fields in the [MTS A Configuration Register \(MTSACFR\)](#) and [MTS B Configuration Register \(MTSBCFR\)](#).

The application enables or disables the generation of the MTS on either channel by setting or clearing the MTE control bit in the [MTS A Configuration Register \(MTSACFR\)](#) or [MTS B Configuration Register \(MTSBCFR\)](#). If an MTS is to be transmitted in a certain communication cycle, the application must set the MTE control bit during the static segment of the preceding communication cycle.

The MTS is transmitted over channel A in the communication cycle with number CYCCNT, if [Equation 30-17](#), [Equation 30-18](#), and [Equation 30-18](#) are fulfilled.

$$\text{PSR0}[\text{PROTSTATE}] = \textit{POC:normal active} \quad \text{Eqn. 30-16}$$

$$\text{MTSACRF}[\text{MTE}] = 1 \quad \text{Eqn. 30-17}$$

$$\text{CYCCNT} \wedge \text{MTSACFR}[\text{CYCCNTMSK}] = \text{MTSACFR}[\text{CYCCNTVAL}] \wedge \text{MTSACFR}[\text{CYCCNTMSK}] \quad \text{Eqn. 30-18}$$

The MTS is transmitted over channel B in the communication cycle with number CYCCNT, if [Equation 30-16](#), [Equation 30-19](#), and [Equation 30-20](#) are fulfilled.

$$\text{MTSBCRF}[\text{MTE}] = 1 \quad \text{Eqn. 30-19}$$

$$\text{CYCCNT} \wedge \text{MTSBCFR}[\text{CYCCNTMSK}] = \text{MTSBCFR}[\text{CYCCNTVAL}] \wedge \text{MTSBCFR}[\text{CYCCNTMSK}] \quad \text{Eqn. 30-20}$$

30.6.14 Sync Frame and Startup Frame Transmission

The transmission of sync frames and startup frames is controlled by the following register fields:

- PCR18.key_slot_id: provides the number of the slot for sync or startup frame transmission
- PCR11.key_slot_used_for_sync: indicates sync frame transmission
- PCR11.key_slot_used_for_startup: indicates startup frame transmission
- PCR12.key_slot_header_crc: provides header crc for sync frame or startup frame
- Message Buffer with message buffer number $n = \text{PCR18.key_slot_id}$

The generation of the sync or startup frames depends on the current protocol state. In the *POC:startup* state, the generation is independent of the message buffer setup; in the *POC:normal active* state, the generation is affected by the current message buffer setup.

30.6.14.1 Sync Frame and Startup Frame Transmission in *POC:startup*

In the *POC:startup* state, the sync and startup frame transmission is independent of the message buffer setup. If at least one of the indication bits PCR11.key_slot_used_for_sync or PCR11.key_slot_used_for_startup is set, a Null Frame will be transmitted in the slot with slot number PCR18.key_slot_id. The header CRC for this Null Frame is taken from PCR12.key_slot_header_crc. The settings of the sync and startup frame indicators are taken from PCR11.key_slot_used_for_sync and PCR11.key_slot_used_for_startup.

30.6.14.2 Sync Frame and Startup Frame Transmission in *POC:normal active*

In the *POC:normal active* state, the sync and startup frame transmission depends on the message buffer setup. If at least one of the indication bits PCR11.key_slot_used_for_sync or

PCR11.key_slot_used_for_startup is set, or if a transmit message buffer with MBFIDRn.FID == PCR18.key_slot_id is configured and enabled, a Null Frame or Data Frame will be transmitted in the slot with slot number PCR18.key_slot_id. The header CRC for this frame is taken from PCR12.key_slot_header_crc, the settings of the sync and startup frame indicators are taken from PCR11.key_slot_used_for_sync and PCR11.key_slot_used_for_startup. A data frame will be transmitted if the message buffer is unlocked and committed and the cycle counter filter matches the current cycle.

30.6.15 Sync Frame Filtering

Each received synchronization frame must pass the Sync Frame Acceptance Filter and the Sync Frame Rejection Filter before it is considered for clock synchronization. If the synchronization frame filtering is globally disabled, i.e. the SFFE control bit in the [Module Configuration Register \(MCR\)](#) is cleared, all received synchronization frames are considered for clock synchronization. If a received synchronization frame did not pass at least one of the two filters, this frame is processed as a normal frame and is not considered for clock synchronization.

30.6.15.1 Sync Frame Acceptance Filtering

The synchronization frame acceptance filter is implemented as a value-mask filter. The value is configured in the [Sync Frame ID Acceptance Filter Value Register \(SFIDAFVR\)](#) and the mask is configured in the [Sync Frame ID Acceptance Filter Mask Register \(SFIDAFMR\)](#). A received synchronization frame with the frame ID FID passes the sync frame acceptance filter, if [Equation 30-21](#) or [Equation 30-22](#) evaluates to true.

$$\text{MCR[SFFE]} = 0 \quad \text{Eqn. 30-21}$$

$$\text{FID[9:0]} \wedge \text{SFIDAFMR[FMSK[9:0]]} = \text{SFIDAFVR[FVAL[9:0]]} \wedge \text{SFIDAFMR[FMSK[9:0]]} \quad \text{Eqn. 30-22}$$

NOTE

Sync frames are transmitted in the static segment only. Thus FID <= 1023.

30.6.15.2 Sync Frame Rejection Filtering

The synchronization frame rejection filter is a comparator. The compare value is defined by the [Sync Frame ID Rejection Filter Register \(SFIDRFR\)](#). A received synchronization frame with the frame ID FID passes the sync frame rejection filter if [Equation 30-23](#) or [Equation 30-24](#) evaluates to true.

$$\text{MCR[SFFE]} = 0 \quad \text{Eqn. 30-23}$$

$$\text{FID[9:0]} \neq \text{SFIDRFR[SYNFRID[9:0]]} \quad \text{Eqn. 30-24}$$

NOTE

Sync frames are transmitted in the static segment only. Thus FID <= 1023.

30.6.16 Strobe Signal Support

The FlexRay block provides a number of strobe signals for observing internal protocol timing related signals in the protocol engine. The signals are listed and described in [Table 30-13](#).

30.6.16.1 Strobe Signal Assignment

Each of the strobe signals listed in [Table 30-13](#) can be assigned to one of the four strobe ports using the [Strobe Signal Control Register \(STBSCR\)](#). To assign multiple strobe signals, the application must write multiple times to the [Strobe Signal Control Register \(STBSCR\)](#) with appropriate settings.

To read out the current settings for a strobe signal with number N, the application must execute the following sequence.

1. Write to STBSCR with WMD = 1 and SEL = N. (updates SEL field only)
2. Read STBCSR.

The SEL field provides N and the ENB and STBPSEL fields provides the settings for signal N.

30.6.16.2 Strobe Signal Timing

This section provides detailed timing information of the strobe signals with respect to the protocol engine clock.

The strobe signals display internal PE signals. Due to the internal architecture of the PE, some signals are generated several PE clock cycles before the actual action is performed on the FlexRay Bus. These signals are listed in [Table 30-13](#) with a negative clock offset. An example waveform is given in [Figure 30-139](#).

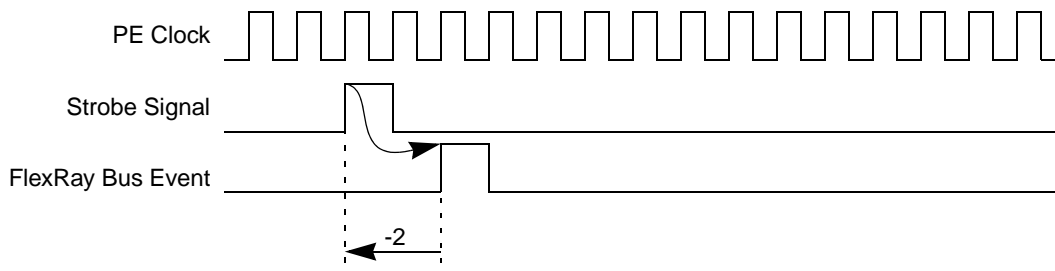


Figure 30-139. Strobe Signal Timing (type = pulse, clk_offset = -2)

Other signals refer to events that occurred on the FlexRay Bus some cycles before the strobe signal is changed. These signals are listed in [Table 30-13](#) with a positive clock offset. An example waveform is given in [Figure 30-140](#).

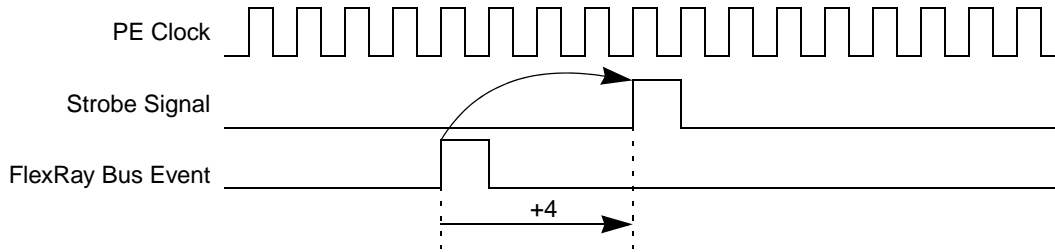


Figure 30-140. Strobe Signal Timing (type = pulse, clk_offset = +4)

30.6.17 Timer Support

The FlexRay block provides two timers, which run on the FlexRay time base. Each timer generates a maskable interrupt when it reaches a configured point in time. Timer T1 is an absolute timer. Timer T2 can be configured to be an absolute or a relative timer. Both timers can be configured to be repetitive. In the non-repetitive mode, timer stops if it expires. In repetitive mode, timer is restarted when it expires.

Both timers are active only when the protocol is in *POC:normal active* or *POC:normal passive* state. If the protocol is not in one of these modes, the timers are stopped. The application must restart the timers when the protocol has reached the *POC:normal active* or *POC:normal passive* state.

30.6.17.1 Absolute Timer T1

The absolute timer T1 has the protocol cycle count and the macrotick count as the time base. The timer 1 interrupt flag TI1_IF in the [Protocol Interrupt Flag Register 0 \(PIFR0\)](#) is set at the macrotick start event, if [Equation 30-25](#) and [Equation 30-26](#) are fulfilled

$$\text{CYCCTR.CYCCNT} \& \text{T1CYSR.T1_CYC_MSK} == \text{T1CYSR.T1_CYC_VAL} \& \text{T1CYSR.T1_CYC_MSK} \quad \text{Eqn. 30-25}$$

$$\text{MTCTR.MTCT} == \text{TI1MTOR.T1_MTOFFSET} \quad \text{Eqn. 30-26}$$

If the timer 1 interrupt enable bit TI1_IE in the [Protocol Interrupt Enable Register 0 \(PIER0\)](#) is asserted, an interrupt request is generated.

The status bit T1ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T1ST bit is not cleared and the timer is restarted immediately. The T1ST is cleared when the timer is stopped.

30.6.17.2 Absolute / Relative Timer T2

The timer T2 can be configured to be an absolute or relative timer by setting the T2_CFG control bit in the [Timer Configuration and Control Register \(TICCR\)](#). The status bit T2ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T2ST bit is not cleared and the timer is restarted immediately. The T2ST is cleared when the timer is stopped.

30.6.17.2.1 Absolute Timer T2

If timer T2 is configured as an absolute timer, it has the same functionality timer T1 but the configuration from [Timer 2 Configuration Register 0 \(TI2CR0\)](#) and [Timer 2 Configuration Register 1 \(TI2CR1\)](#) is used.

On expiration of timer T2, the interrupt flag TI2_IF in the [Protocol Interrupt Flag Register 0 \(PIFR0\)](#) is set. If the timer 1 interrupt enable bit TI1_IE in the [Protocol Interrupt Enable Register 0 \(PIERO\)](#) is asserted, an interrupt request is generated.

30.6.17.2.2 Relative Timer T2

If the timer T2 is configured as a relative timer, the interrupt flag TI2_IF in the [Protocol Interrupt Flag Register 0 \(PIFR0\)](#) is set, when the programmed amount of macroticks MT[31:0], defined by [Timer 2 Configuration Register 0 \(TI2CR0\)](#) and [Timer 2 Configuration Register 1 \(TI2CR1\)](#), has expired since the trigger or restart of timer 2. The relative timer is implemented as a down counter and expires when it has reached 0. At the macrotick start event, the value of MT[31:0] is checked and then decremented. Thus, if the timer is started with $MT[31:0] == 0$, it expires at the next macrotick start.

30.6.18 Slot Status Monitoring

The FlexRay block provides several means for slot status monitoring. All slot status monitors use the same slot status vector provided by the PE. The PE provides a slot status vector for each static slot, for each dynamic slot, for the symbol window, and for the NIT, on a per channel base. The content of the slot status vector is described in [Table 30-112](#). The PE provides the slot status vector within the first macrotick after the end of the related slot/window/NIT, as shown in [Figure 30-141](#).

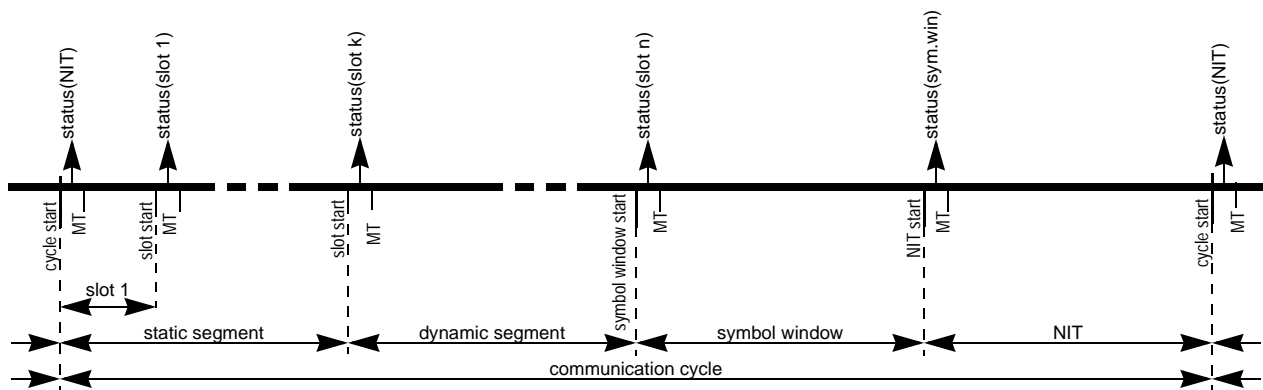


Figure 30-141. Slot Status Vector Update

NOTE

The slot status for the NIT of cycle n is provided after the start of cycle n+1.

Table 30-112. Slot Status Content

	Status Content
static / dynamic Slot	<p>slot related status</p> <p><i>vSS!ValidFrame</i> - valid frame received <i>vSS!SyntaxError</i> - syntax error occurred while receiving <i>vSS!ContentError</i> - content error occurred while receiving <i>vSS!BViolation</i> - boundary violation while receiving <i>for slots in which the module transmits:</i> <i>vSS!TxConflict</i> - reception ongoing while transmission starts <i>for slots in which the module does not transmit:</i> <i>vSS!TxConflict</i> - reception ongoing while transmission starts first valid - channel that has received the first valid frame</p> <p>received frame related status <i>extracted from</i></p> <p>a) header of valid frame, if <i>vSS!ValidFrame</i> = 1 b) last received header, if <i>vSS!ValidFrame</i> = 0 c) set to 0, if nothing was received</p> <p><i>vRF!Header!NFIndicator</i> - Null Frame Indicator (0 for null frame) <i>vRF!Header!SuFIndicator</i> - Startup Frame Indicator <i>vRF!Header!SyFIndicator</i> - Sync Frame Indicator</p>
Symbol Window	<p>window related status</p> <p><i>vSS!ValidFrame</i> - always 0 <i>vSS!ContentError</i> - content error occurred while receiving <i>vSS!SyntaxError</i> - syntax error occurred while receiving <i>vSS!BViolation</i> - boundary violation while receiving <i>vSS!TxConflict</i> - reception ongoing while transmission starts</p> <p>received symbol related status <i>vSS!ValidMTS</i> - valid Media Test Access Symbol received</p> <p>received frame related status <i>see static/dynamic slot</i></p>
NIT	<p>NIT related status</p> <p><i>vSS!ValidFrame</i> - always 0 <i>vSS!ContentError</i> - content error occurred while receiving <i>vSS!SyntaxError</i> - syntax error occurred while receiving <i>vSS!BViolation</i> - boundary violation while receiving <i>vSS!TxConflict</i> - always 0</p> <p>received frame related status <i>see static/dynamic slot</i></p>

30.6.18.1 Channel Status Error Counter Registers

The two channel status error counter registers, [Channel A Status Error Counter Register \(CASERCR\)](#) and [Channel B Status Error Counter Register \(CBSERCR\)](#), incremented by one, if at least one of four slot status error bits, *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, or *vSS!TxConflict* is set to 1. The status vectors for all slots in the static and dynamic segment, in the symbol window, and in the NIT are taken into account. The counters wrap round after they have reached the maximum value.

30.6.18.2 Protocol Status Registers

The [Protocol Status Register 2 \(PSR2\)](#) provides slot status information about the Network Idle Time NIT and the Symbol Window. The [Protocol Status Register 3 \(PSR3\)](#) provides aggregated slot status information.

30.6.18.3 Slot Status Registers

The eight slot status registers, [Slot Status Registers \(SSR0–SSR7\)](#), can be used to observe the status of static slots, dynamic slots, the symbol window, or the NIT without individual message buffers. These registers provide all slot status related and received frame / symbol related status information, as given in [Table 30-112](#), except of the *first valid* indicator for non-transmission slots.

30.6.18.4 Slot Status Counter Registers

The FlexRay block provides four slot status error counter registers, [Slot Status Counter Registers \(SSCR0–SSCR3\)](#). Each of these slot status counter registers is updated with the value of an internal slot status counter at the start of a communication cycle. The internal slot status counter is incremented if its increment condition, defined by the [Slot Status Counter Condition Register \(SSCCR\)](#), matches the status vector provided by the PE. All static slots, the symbol window, and the NIT status are taken into account. *Dynamic* slots are *excluded*. The internal slot status counting and update timing is shown in [Figure 30-142](#).

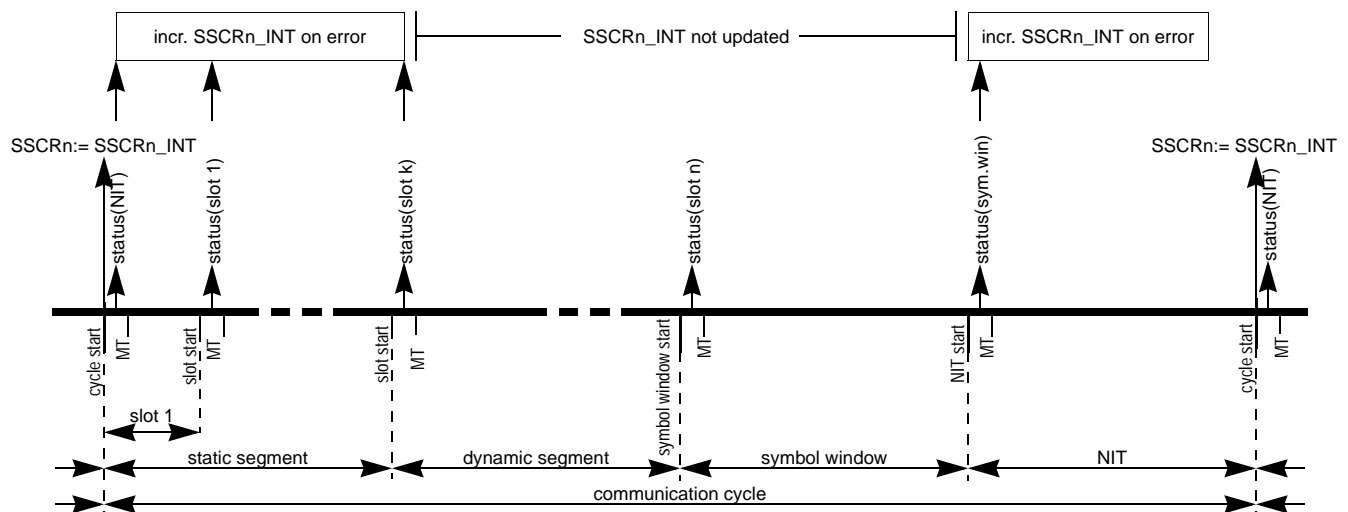


Figure 30-142. Slot Status Counting and SSCRn Update

The PE provides the status of the NIT in the first slot of the next cycle. Due to these facts, the SSCRn register reflects, in cycle n, the status of the NIT of cycle n-2, and the status of all static slots and the symbol window of cycle n-1.

The increment condition for each slot status counter consists of two parts, the frame related condition part and the slot related condition part. The internal slot status counter SSCRn_INT is incremented if at least one of the conditions is fulfilled:

1. frame related condition:

- (SSCCRn.VFR | SSSCCRn.SYF | SSSCCRn.NUF | SSSCCRn.SUF) // count on frame condition = 1;

and

- ((~SSCCRn.VFR | *vSS!ValidFrame*) & // valid frame restriction
(~SSCCRn.SYF | *vRF!Header!SyFIndicator*) & // sync frame indicator restriction
(~SSCCRn.NUF | *~vRF!Header!NFIndicator*) & // null frame indicator restriction
(~SSCCRn.SUF | *vRF!Header!SuFIndicator*)) // startup frame indicator restriction = 1;

NOTE

The indicator bits SYF, NUF, and SUF are valid only when a valid frame was received. Thus it is required to set the VFR always, whenever count on frame condition is used.

2. slot related condition:

- ((SSCCRn.STATUSMASK[3] & *vSS!ContentError*) | // increment on content error
(SSCCRn.STATUSMASK[2] & *vSS!SyntaxError*) | // increment on syntax error
(SSCCRn.STATUSMASK[1] & *vSS!BViolation*) | // increment on boundary violation
(SSCCRn.STATUSMASK[0] & *vSS!TxConflict*)) // increment on transmission conflict = 1;

If the slot status counter is in single cycle mode, i.e. SSSCCRn.MCY = 0, the internal slot status counter SSSCCRn_INT is reset at each cycle start. If the slot status counter is in the multicycle mode, i.e. SSSCCRn.MCY = 1, the counter is not reset and incremented, until the maximum value is reached.

30.6.18.5 Message Buffer Slot Status Field

Each individual message buffer and each FIFO message buffer provides a slot status field, which provides the information shown in [Table 30-112](#) for the static/dynamic slot. The update conditions for the slot status field depend on the message buffer type. Refer to the Message Buffer Update Sections in [Section 30.6.6](#), “[Individual Message Buffer Functional Description](#)”.

30.6.19 Interrupt Support

The FlexRay block provides 108 individual interrupt sources and five combined interrupt sources.

30.6.19.1 Individual Interrupt Sources

30.6.19.1.1 Message Buffer Interrupts

The FlexRay block provides 64 message buffer interrupt sources.

Each individual message buffer provides an interrupt flag MBCCSn.MBIF and an interrupt enable bit MBCCSn.MBIE. The FlexRay block sets the interrupt flag when the slot status of the message buffer was updated. If the interrupt enable bit is asserted, an interrupt request is generated.

30.6.19.1.2 Receive FIFO Interrupts

The FlexRay block provides 2 Receive FIFO interrupt sources.

Each of the 2 Receive FIFO provides a Receive FIFO Not Empty Interrupt Flag. The FlexRay block sets the Receive FIFO Not Empty Interrupt Flags (GIFER.FNEBIF, GIFER.FNEAIF) in the [Global Interrupt Flag and Enable Register \(GIFER\)](#) if the corresponding Receive FIFO is not empty.

30.6.19.1.3 Wakeup Interrupt

The FlexRay block provides one interrupt source related to the wakeup.

The FlexRay block sets the Wakeup Interrupt Flag GIFER.WUPIF when it has received a wakeup symbol on the FlexRay bus. The FlexRay block generates an interrupt request if the interrupt enable bit GIFER.WUPIE is asserted.

30.6.19.1.4 Protocol Interrupts

The FlexRay block provides 25 interrupt sources for protocol related events. For details, see [Protocol Interrupt Flag Register 0 \(PIFR0\)](#) and [Protocol Interrupt Flag Register 1 \(PIFR1\)](#). Each interrupt source has its own interrupt enable bit.

30.6.19.1.5 CHI Error Interrupts

The FlexRay block provides 16 interrupt sources for CHI related error events. For details, see [CHI Error Flag Register \(CHIERFR\)](#). There is one common interrupt enable bit GIFER.CHIIE for all CHI error interrupt sources.

30.6.19.2 Combined Interrupt Sources

Each combined interrupt source generates an interrupt request only when at least one of the interrupt sources that is combined generates an interrupt request.

30.6.19.2.1 Receive Message Buffer Interrupt

The combined receive message buffer interrupt request RBIRQ is generated when at least one of the individual receive message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit GIFER.RBIE is set.

30.6.19.2.2 Transmit Message Buffer Interrupt

The combined transmit message buffer interrupt request TBIRQ is generated when at least one of the individual transmit message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit GIFER.TBIE is asserted.

30.6.19.2.3 Protocol Interrupt

The combined protocol interrupt request PRTIRQ is generated when at least one of the individual protocol interrupt sources generates an interrupt request and the interrupt enable bit GIFER.PRIE is set.

30.6.19.2.4 CHI Error Interrupt

The combined CHI error interrupt request CHIIRQ is generated when at least one of the individual chi error interrupt sources generates an interrupt request and the interrupt enable bit GIFER.CHIE is set.

30.6.19.2.5 Module Interrupt

The combined module interrupt request MIRQ is generated if at least one of the combined interrupt sources generates an interrupt request and the interrupt enable bit GIFER.MIE is set.

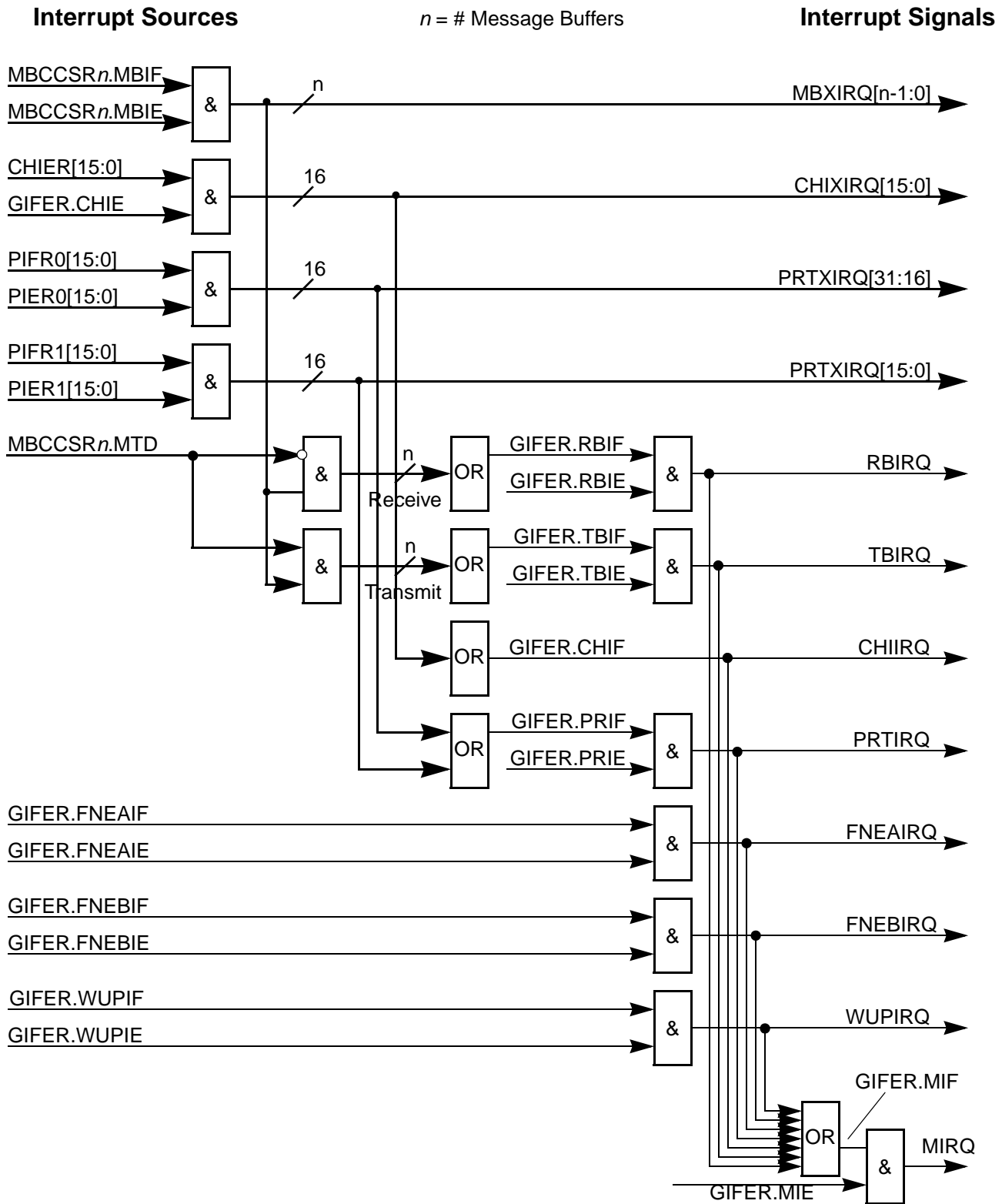


Figure 30-143. Scheme of Cascaded Interrupt Request

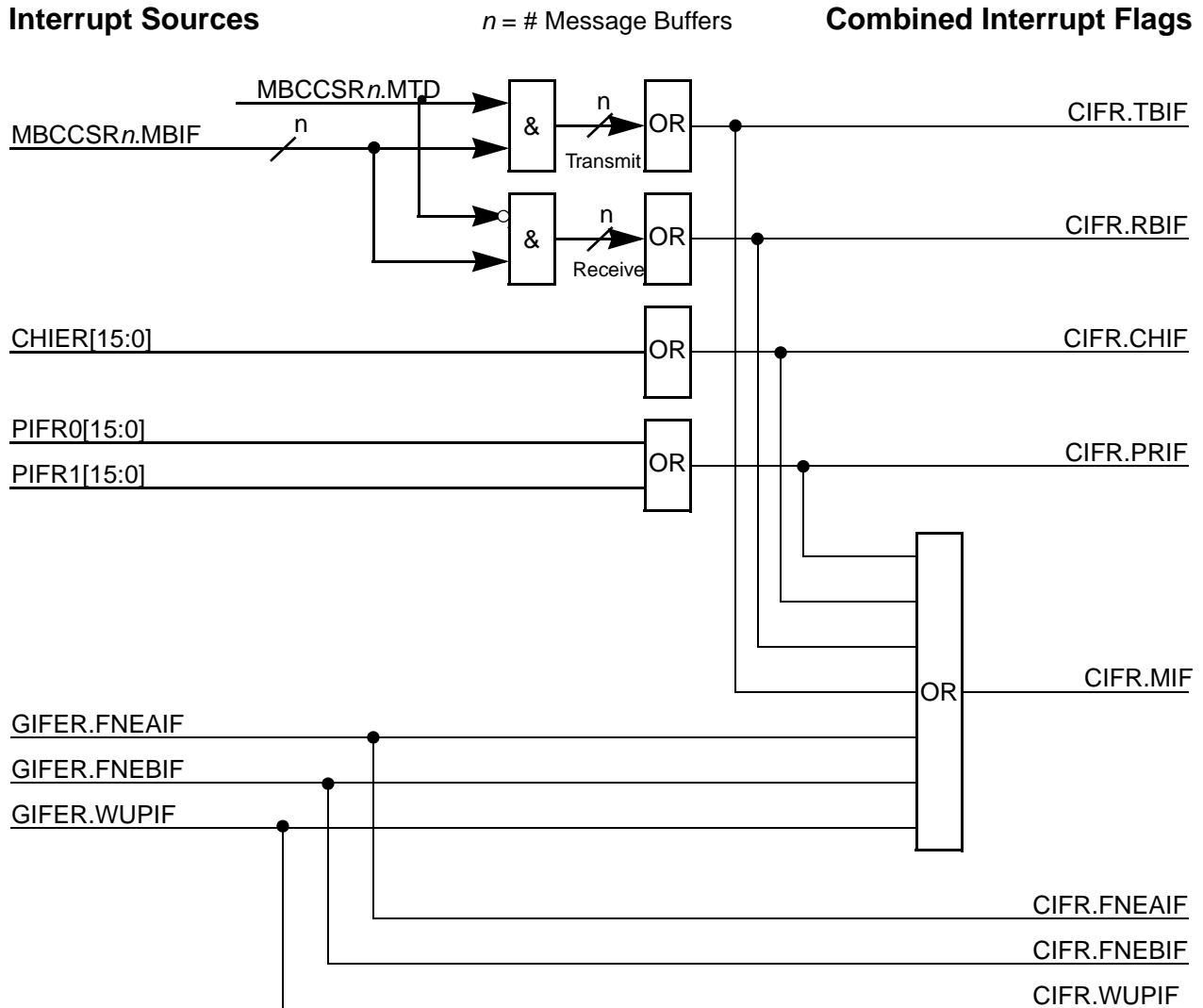


Figure 30-144. Scheme of Combined Interrupt Flags

30.6.20 Lower Bit Rate Support

The FlexRay block supports a number of lower bit rates on the FlexRay bus channels. The lower bit rates are implemented by modifying the duration of the microtick *pdMicrotick*, the number of samples per microtick *pSamplesPerMicrotick*, the number of samples per bit *cSamplesPerBit*, and the strobe offset *cStrobeOffset*. The application configures the FlexRay channel bit rate by setting the BITRATE field in the **Module Configuration Register (MCR)**. The protocol values are set internally. The available bit rates, the related BITRATE field configuration settings and related protocol parameter values are shown in [Table 30-113](#).

Table 30-113. FlexRay Channel Bit Rate Control

FlexRay Channel Bit Rate [Mbit/s]	MCR.BITRATE	$p_{dMicrotick}$ [ns]	$g_{dSampleClockPeriod}$ [ns]	$p_{SamplesPerMicrotick}$	$c_{SamplesPerBit}$	$c_{StrobeOffset}$
10.0	000	25.0	12.5	2	8	5
8.0	011	25.0	12.5	2	10	6
5.0	001	25.0	25.0	1	8	5
2.5	010	50.0	50.0	1	8	5

NOTE

The bit rate of 8 Mbit/s is not defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

30.7 Application Information

30.7.1 Initialization Sequence

This section describes the required steps to initialize the FlexRay block. The first subsection describes the steps required after a system reset, the second section describes the steps required after preceding shutdown of the FlexRay block.

30.7.1.1 Module Initialization

This section describes the module related initialization steps after a system reset.

1. Configure FlexRay block.
 - a) configure the control bits in the [Module Configuration Register \(MCR\)](#)
 - b) configure the system memory base address in [System Memory Base Address High Register \(SYMBADHR\)](#) and [System Memory Base Address Low Register \(SYMBADLR\)](#)
2. Enable the FlexRay block.
 - a) write 1 to the module enable bit MEN in the [Module Configuration Register \(MCR\)](#)

The FlexRay block now enters the Normal Mode. The application can commence with the protocol initialization described in [Section 30.7.1.2, “Protocol Initialization”](#).

30.7.1.2 Protocol Initialization

This section describes the protocol related initialization steps.

1. Configure the Protocol Engine.
 - a) issue CONFIG command via [Protocol Operation Control Register \(POCR\)](#)
 - b) wait for *POC:config* in [Protocol Status Register 0 \(PSR0\)](#)
 - c) configure the PCR0,..., PCR30 registers to set all protocol parameters
2. Configure the Message Buffers and FIFOs.
 - a) set the number of message buffers used and the message buffer segmentation in the [Message Buffer Segment Size and Utilization Register \(MBSSUTR\)](#)
 - b) define the message buffer data size in the [Message Buffer Data Size Register \(MBDSR\)](#)
 - c) configure each message buffer by setting the configuration values in the [Message Buffer Configuration, Control, Status Registers \(MBCCSRn\)](#), [Message Buffer Cycle Counter Filter Registers \(MBCCFRn\)](#), [Message Buffer Frame ID Registers \(MBFIDRn\)](#), [Message Buffer Index Registers \(MBIDXRn\)](#)
 - d) configure the receive FIFOs
 - e) issue CONFIG_COMPLETE command via [Protocol Operation Control Register \(POCR\)](#)
 - f) wait for *POC:ready* in [Protocol Status Register 0 \(PSR0\)](#)

After this sequence, the FlexRay block is configured as a FlexRay node and is ready to integrate into the FlexRay cluster.

30.7.2 Shut Down Sequence

This section describes a secure shut down sequence to stop the FlexRay block gracefully. The main targets of this sequence are

- finish all ongoing reception and transmission
- do not corrupt FlexRay bus and do not disturb ongoing FlexRay bus communication

For a graceful shutdown the application shall perform the following tasks:

1. Disable all enabled message buffers.
 - a) repeatedly write 1 to MBCCSRn[EDT] until MBCCSRn[EDS] == 0.
2. Stop Protocol Engine.
 - a) issue HALT command via [Protocol Operation Control Register \(POCR\)](#)
 - b) wait for *POC:halt* in [Protocol Status Register 0 \(PSR0\)](#)

30.7.3 Number of Usable Message Buffers

This section describes the relationship between the number of message buffers that can be utilized and the required minimum CHI clock frequency. Additional constraints for the minimum CHI clock frequency are given in [Section 30.3, “Controller Host Interface Clocking”](#).

The FlexRay block uses a sequential search algorithm to determine the individual message buffer assigned or subscribed to the next slot. This search must be finished within one FlexRay slot. The shortest FlexRay slot is an empty dynamic slot. An empty dynamic slot is a minislot and consists of $gdMinislot$ macroticks with a nominal duration of $gdMacrotick$. The minimum duration of a corrected macrotick is $gdMacrotick_{min} = 39 \mu\mu T$. This results in a minimum slot length of

$$\Delta_{slotmin} = 39 \cdot pdMicrotick \cdot gdMinislot \quad Eqn. 30-27$$

The search engine is located in the CHI and runs on the CHI clock. It evaluates one individual message buffer per CHI clock cycle. For internal status update and double buffer commit operations, and as a result of the clock domain crossing jitter, an additional amount of 10 CHI clock cycles is required to ensure correct operation. For a given number of message buffers and for a given CHI clock frequency f_{chi} , this results in a search duration of

$$\Delta_{search} = \frac{1}{f_{chi}} \cdot (\# MessageBuffers + 10) \quad Eqn. 30-28$$

The message buffer search must be finished within one slot which requires that Equation 30-29 must be fulfilled

$$\Delta_{search} \leq \Delta_{slotmin} \quad Eqn. 30-29$$

This results in the formula given in Equation 30-30 which determines the required minimum CHI frequency for a given number of message buffers that are utilized.

$$f_{chi} \geq \frac{\# MessageBuffers + 10}{39 \cdot pdMicrotick \cdot gdMinislot} \quad Eqn. 30-30$$

The minimum CHI frequency for a selected set of relevant protocol parameters is given in Table 30-114.

Table 30-114. Minimum f_{chi} [MHz] Examples (64 Message Buffers)

$pdMicrotick$ [ns]	$gdMinislot$					
	2	3	4	5	6	7
25.0	37.94	25.30	18.98	15.18	12.65	10.84
50.0	18.98	12.65	9.45	7.59	6.33	5.43

30.7.4 Protocol Control Command Execution

This section considers the issues of the protocol control command execution.

The application issues any of the protocol control commands listed in the POC CMD field of Table 30-16 by writing the command to the POC CMD field of the Protocol Operation Control Register (POCR). As a result the FlexRay block sets the BSY bit while the command is transferred to the PE. When the PE has accepted the command, the BSY flag is cleared. All commands are accepted by the PE.

The PE maintains a protocol command vector. For each command that was accepted by the PE, the PE sets the corresponding command bit in the protocol command vector. If a command is issued while the corresponding command bit is set, the command is not queued and is lost.

If the command execution block of the PE is idle, it selects the next accepted protocol command with the highest priority from the current protocol command vector according to the protocol control command priorities given in [Table 30-115](#). If the current protocol state does not allow the execution of this protocol command (see POC state changes in *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*) the FlexRay block asserts the illegal protocol command interrupt flag `IPC_IF` in the [Protocol Interrupt Flag Register 1 \(PIFR1\)](#). The protocol command is not executed in this case.

Some protocol commands may be interrupted by other commands or the detection of a fatal protocol error as indicated by [Table 30-115](#). If the application issues the `RESET`, `FREEZE`, or `READY` command, or if the the PE detects a fatal protocol error, some commands already stored in the command vector will be removed from this vector.

Table 30-115. Protocol Control Command Priorities

Protocol Command	Priority	Interrupted By	Cleared and Terminated By
RESET	(highest) 1	none	
FREEZE	2		RESET
READY	3		RESET
CONFIG_COMPLETE	3		RESET
ALL_SLOTS	4	RESET, FREEZE, READY, CONFIG_COMPLETE, fatal protocol error	RESET, FREEZE, READY, CONFIG_COMPLETE, fatal protocol error
ALLOW_COLDSTART	5		RESET
RUN	6		RESET, FREEZE, fatal protocol error
WAKEUP	7		RESET, FREEZE, fatal protocol error
DEFAULT_CONFIG	8		RESET, FREEZE, fatal protocol error
CONFIG	9		RESET
HALT	(lowest) 10		RESET, FREEZE, READY, CONFIG_COMPLETE, fatal protocol error

30.7.5 Protocol Reset Command

The section considers the issues of the protocol `RESET` command.

The application issues the protocol reset command by writing the `RESET` command code to the `POCCMD` field of the [Protocol Operation Control Register \(POCR\)](#). As a result, the PE stops its operation immediately, the FlexRay bus ports put into their idle state, and no more data or status information is sent to the CHI. The lack of PE signals stops all message buffer operations in the CHI. In particular, the message buffers that are currently under internal use remain internally locked. To overcome this message buffer internal lock situation, the application must put the protocol into the *POC:default config* state. This will release all internal message buffer locks.

The following sequence must be executed by the application to put the protocol into the *POC:default config* state.

1. Repeatedly send Protocol Command FREEZE via Protocol Operation Control Register (POCR), until the freeze bit FRZ in Protocol Status Register 1 (PSR1) is set.
2. Repeatedly send Protocol Command DEFAULT_CONFIG via Protocol Operation Control Register (POCR), until the freeze bit FRZ bit in Protocol Status Register 1 (PSR1) is cleared and the PROTSTATE field in Protocol Status Register 0 (PSR0) is set to *POC:default config*.

30.7.6 Message Buffer Search on Simple Message Buffer Configuration

This sections describes the message buffer search behavior for a simplified message buffer configuration. The receive FIFO behavior is not considered in this section.

30.7.6.1 Simple Message Buffer Configuration

A simple message buffer configuration is a configuration that has at most one transmit message buffer and at most one receive message buffer assigned to a slot S . The simple configuration used in this section utilizes two message buffers, one single buffered transmit message buffer and one receive message buffer.

The transmit message buffer has the message buffer number t and has following configuration

Table 30-116. Transmit Buffer Configuration

Register	Field	Value	Description
MBCCSR t	MCM	-	used only for double buffers
	MBT	0	single transmit buffer
	MTD	1	transmit buffer
MBCCFR t	MTM	0	event transition mode
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000011	cycle set = $\{4n\} = \{0,4,8,12,\dots\}$
	CCFVAL	000000	
MBFIDR t	FID	S	assigned to slot S

The availability of data in the transmit buffer is indicated by the commit bit MBCCSR t [CMT] and the lock bit MBCCSR t [LCKS].

The receive message buffer has the message buffer number r and has following configuration

Table 30-117. Receive Buffer Configuration

Register	Field	Value	Description
MBCCSR _r	MCM	-	n/a
	MBT	-	n/a
	MTD	0	receive buffer
MBCCF _r	MTM	-	n/a
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000001	cycle set = {2n} = {0,2,4,6, ... }
	CCFVAL	000000	
MBFIDR _r	FID	S	subscribed slot

Furthermore the assumption is that both message buffers are enabled (MBCCSR_t[EDS] = 1 and MBCCSR_r[EDS] = 1)

NOTE

The cycle set $\{4n+2\} = \{2,6,10, \dots\}$ is assigned to the receive buffer only.

The cycle set $\{4n\} = \{0,4,8,12, \dots\}$ is assigned to both buffers.

30.7.6.2 Behavior in static segment

In this case, both message buffers are assigned to a slot S in the *static* segment.

The configuration of a transmit buffer for a static slot S assigns this slot to the node as a transmit slot. The FlexRay protocol requires:

- When a slot occurs, if the slot is assigned to a node on a channel that node must transmit either a normal frame or a null frame on that channel. Specifically, a null frame will be sent if there is no data ready, or if there is no match on a transmit filter (cycle counter filtering, for example).

Regardless of the availability of data and the cycle counter filter, the node will transmit a frame in the static slot S . In any case, the result of the message buffer search will be the transmit message buffer t . The receive message buffer r will not be found, no reception is possible.

30.7.6.3 Behavior in dynamic segment

In this case, both message buffers are assigned to a slot S in the *dynamic* segment. The FlexRay protocol requires:

- When a slot occurs, if a slot is assigned to a node on a channel that node only transmits a frame on that channel if there is data ready and there is a match on relevant transmit filters (no null frames are sent).

Chapter 31

Enhanced Queued Analog-to-Digital Converter (eQADC)

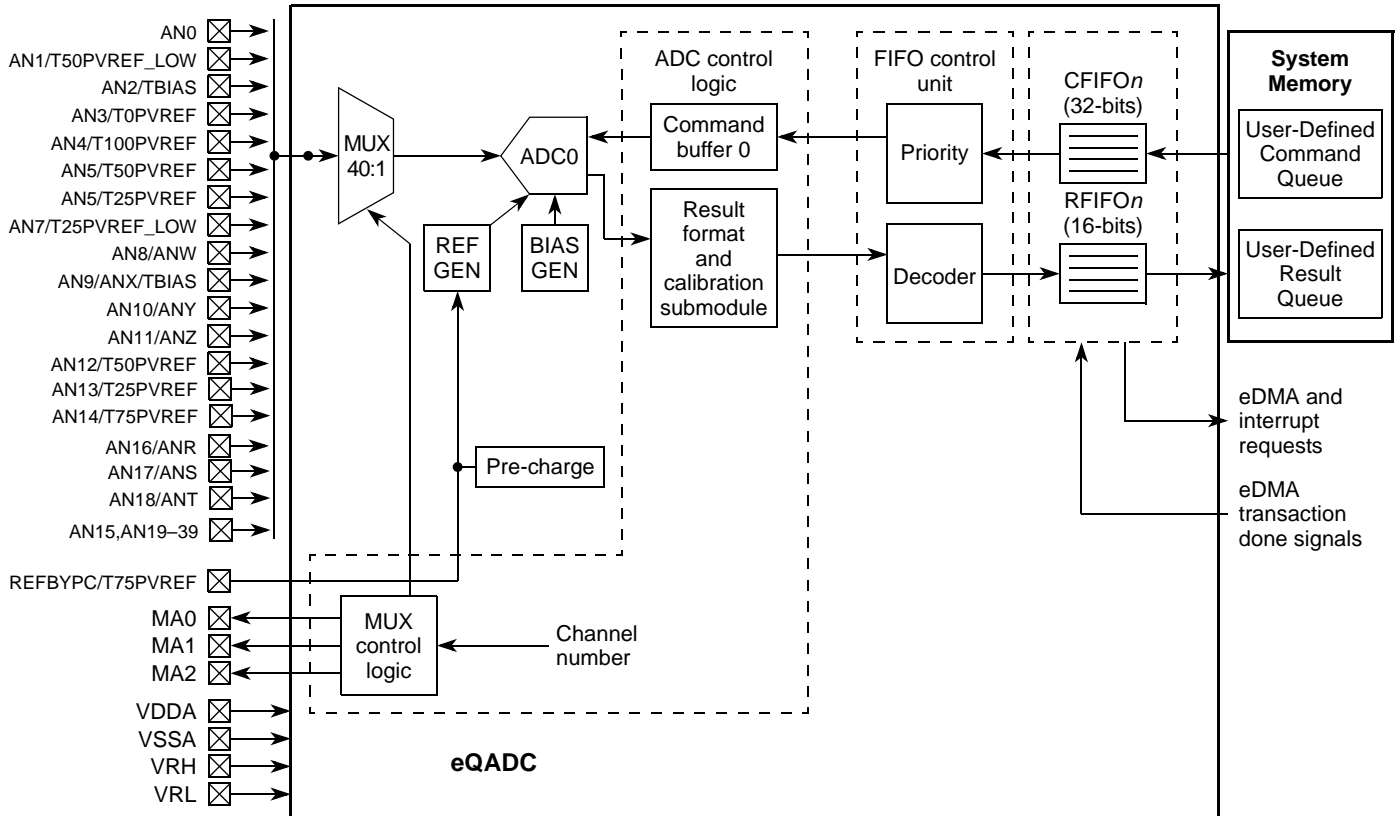
31.1 Introduction

The enhanced queued analog-to-digital converter (eQADC) provides accurate and fast conversions for a range of applications. The eQADC provides a parallel interface to a single on-chip analog-to-digital converter (ADC). The on-chip ADC is architected to allow access to all the analog channels.

The eQADC transfers commands from multiple command FIFOs (CFIFOs) to the on-chip ADC. The block can also receive data from the on-chip ADC into multiple result FIFOs (RFIFOs), in parallel to and independently of the CFIFOs. The eQADC supports software and external hardware triggers from other blocks to initiate transfers of commands from the CFIFOs to the on-chip ADC. It also monitors the fullness of CFIFOs and RFIFOs, and accordingly generates DMA or interrupt requests to control data movement between the FIFOs and the system memory.

31.1.1 Block Diagram

Figure 31-1 shows the primary components inside the eQADC.



NOTES: $n = 0, 1, 2, 3, 4, 5$


Some signals at pins denoted by  may be muxed on a single package pin.

Figure 31-1. eQADC Block Diagram

Figure 31-1 shows the primary components inside the eQADC. The eQADC consists of the FIFO control unit, which controls the CFIFOs and the RFIFOs, and the ADC control logic, which controls the on-chip ADC. There are six CFIFOs and six RFIFOs, each with four entries.

The FIFO control unit performs the following functions:

- It prioritizes the CFIFOs to determine what CFIFOs will have their commands transferred.
- Supports software and hardware triggers to start command transfers from a particular CFIFO.
- Decodes result data from the on-chip ADC, and transfers data to the appropriate RFIFO.

The ADC control logic manages the execution of commands bound for the on-chip ADC. It interfaces with the CFIFOs via one 2-entry command buffer (CBuffer) and with the RFIFOs via the result format and calibration sub-block. The ADC control logic performs the following functions:

- Buffers command data for execution.
- Decodes command data and accordingly generates control signals for the on-chip ADC.

- Formats and calibrates conversion result data coming from the on-chip ADC.
- Generates the internal multiplexer control signals and the select signals used by the external multiplexers.

Figure 31-1 also depicts data flow through the eQADC. Commands are contained in system memory in an user defined data structure. The most likely data structure to be used is a queue as depicted in the Figure 31-1¹. Command data is moved from the command queue (CQueue) to the CFIFOs by either the host CPU or by the DMAC. After a CFIFO is triggered and becomes the highest priority CFIFO using a certain CBuffer, command data is transferred from the CFIFO to the ADC on chip. The ADC executes the command, and the result is moved through the result format and calibration sub-block to the RFIFO specified by a field in the command that initiated the conversion. When data is stored in an RFIFO, data is moved from the RFIFO by the host CPU or by the DMAC to a data structure in system memory depicted in the Figure 31-1 as a result queue (RQueue).

31.1.2 Features

The eQADC has these major features:

- One on-chip RSD cyclic ADC
 - 12-bit AD resolution
 - Targets up to 9-bit accuracy at 400 KSample/s (ADC_CLK=6 MHz) (the actual accuracy is TBD, subject to the final characterization)
 - Single-ended signal range from 0 to 5 V
 - Sample times of 2 (default), 8, 64, or 128 ADC clock cycles
 - Provides time stamp information when requested
 - Parallel interface to eQADC CFIFOs and RFIFOs
 - Supports both right-justified unsigned and signed formats for conversion results
 - The REFBYPC is a stable reference voltage for the eQADC and is used to connect a 100 nF bypass capacitor between the REFBYPC pin and VRL.
- Automatic application of ADC calibration constants
 - Provision of reference voltages (25% VREF² and 75% VREF) for ADC calibration purposes
- 40 input channels available to the on-chip ADC
- Priority-based CFIFOs
 - Supports six CFIFOs with fixed priority. The lower the CFIFO number, the higher its priority. When commands of distinct CFIFOs are bound for the same CBuffer, the higher priority CFIFO is always served first.
 - Supports software and several hardware trigger modes to arm a particular CFIFO
 - Generates interrupt when command coherency is not achieved
- External hardware triggers

1. Command and result data can be stored in system memory in any user defined data structure. However, in this document it will be assumed that the data structure of choice is a queue, since it is the most likely data structure to be used and because queues are the only type of data structure supported by the DMAC.

2. VREF=VRH-VRL.

- Supports rising-edge, falling-edge, high-level, and low-level triggers
- Supports configurable digital filter

NOTE

If a PIT trigger is selected as the source of the trigger, then the trigger pulse width will be two PIT clocks long. The PIT clock may be the system clock divided by 1, 2, 4, or 8 as selected by the SIU_SYSCLK[LPCLKDIV1] register. Thus the eQADC digital filtering needs to be set appropriately.

- Supports seven external 8-to-1 muxes which can expand the input channel number from 40 to 68
- Upgrades the functionality provided by the QADC

31.1.3 Modes of Operation

31.1.4 Normal Mode

This is the default operational mode when the eQADC is not in background debug mode.

31.1.5 Debug Mode

Upon detection of a debug mode entry request, the eQADC enters debug mode if entry to this mode is enabled. During debug mode, the eQADC will not transfer commands from any CFIFOs, no data will be returned to any RFIFO, no hardware trigger event will be captured, and all eQADC registers can be accessed as in normal mode. If there are commands in the on-chip CBuffers that were already under execution at the time the debug mode entry request is detected, these commands will be completed but the generated results, if any, will not be sent to the RFIFOs until debug mode is exited. Commands whose execution has not started will not be executed until debug mode is exited. The clock associated with an on-chip ADC stops during its low phase, after the ADC ceases executing commands. The time base counter will stop only after the on-chip ADC ceases executing commands.

When exiting debug mode, the eQADC relies on the CFIFO operation modes and on the CFIFO status to determine the next command entry to transfer.

The eQADC internal behavior after the debug mode entry request is detected differs depending on the status of command transfers.

- No command transfer is in progress.
The eQADC immediately halts future command transfers from any CFIFO.
- Command transfer is in progress.
eQADC will complete the transfer and update CFIFO status before halting future command transfers from any CFIFO.

If the command message transmission is aborted, the eQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after debug mode is exited.

31.1.5.1 Stop Mode

Upon a stop mode entry request detection based on setting the SIU HLT bit for the eQADC, the eQADC progressively halts its operations until it reaches a static, stable state from which it can recover when returning to normal mode. The eQADC then asserts an acknowledge signal, indicating that it is static and that the clock input can be stopped. The acknowledge signal is visible via the SIU HLT_ACK bit for the eQADC.

If at the time the stop mode entry request is detected, there are commands in the ADC that were already under execution, these commands will be completed but the generated results, if any, will not be sent to the RFIFOs until stop mode is exited. Commands whose execution has not started will not be executed until stop mode is exited.

After these remaining commands are executed, the clock input to the ADCs is stopped. The time base counter will stop after all on-chip ADCs cease executing commands and then the stop acknowledge signal is asserted. When exiting stop mode, the eQADC relies on the CFIFO operation modes and on the CFIFO status to determine the next command entry to transfer.

The eQADC internal behavior after the stop mode entry request is detected differs depending on the status of the command transfer.

- No command transfer is in progress

The eQADC immediately halts future command transfers from any CFIFO.

If a null message is being transmitted, eQADC will complete the transmission before halting future command transfers. If valid data (conversion result or data read from an ADC register) is received at the end of the transmission, it will not be sent to an RFIFO until stop mode is exited.

If the null message transmission is aborted, the eQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after stop mode is exited.

- Command transfer is in progress.

The eQADC will complete the transfer and update CFIFO status before halting future command transfers from any CFIFO.

If the command message transmission is aborted, the eQADC will complete the abort procedure before halting future command transfers from any CFIFO. The message of the CFIFO that caused the abort of the previous serial transmission will only be transmitted after stop mode is exited.

31.2 External Signal Description

Refer to [Table 2-1](#) and [Section 2.7](#), “Detailed External Signal Descriptions,” for detailed signal descriptions.

31.3 Memory Map and Registers

This section provides a detailed description of all eQADC registers.

31.3.1 Module Memory Map

The eQADC memory map is shown in [Table 31-1](#). The address of each register is given as an offset to the eQADC base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

Table 31-1. eQADC Memory Map

Offset from EQADC_BASE (0xFFFF8_0000)	Register	Access	Reset Value	Section/Page
0x0000	eQADC Module Configuration Register (EQADC_MCR)	R/W		31.3.3.1/31-9
0x0004	Reserved			
0x0008	eQADC Null Message Send Format Register (EQADC_NMSFR)	R/W		31.3.3.2/31-10
0x000C	eQADC External Trigger Digital Filter Register (EQADC_ETDFR)	R/W		31.3.3.3/31-10
0x0010	eQADC CFIFO Push Register 0 (EQADC_CFPR0)	W		31.3.3.4/31-12
0x0014	eQADC CFIFO Push Register 1 (EQADC_CFPR1)	W		
0x0018	eQADC CFIFO Push Register 2 (EQADC_CFPR2)	W		
0x001C	eQADC CFIFO Push Register 3 (EQADC_CFPR3)	W		
0x0020	eQADC CFIFO Push Register 4 (EQADC_CFPR4)	W		
0x0024	eQADC CFIFO Push Register 5 (EQADC_CFPR5)	W		
0x0028	Reserved			
0x002C	Reserved			
0x0030	eQADC Result FIFO Pop Register 0 (EQADC_RFPR0)	R		31.3.3.5/31-12
0x0034	eQADC Result FIFO Pop Register 1 (EQADC_RFPR1)	R		
0x0038	eQADC Result FIFO Pop Register 2 (EQADC_RFPR2)	R		
0x003C	eQADC Result FIFO Pop Register 3 (EQADC_RFPR3)	R		
0x0040	eQADC Result FIFO Pop Register 4 (EQADC_RFPR4)	R		
0x0044	eQADC Result FIFO Pop Register 5 (EQADC_RFPR5)	R		
0x0048	Reserved			
0x004C	Reserved			
0x0050	eQADC CFIFO Control Register 0 (EQADC_CFCR0)	R/W		31.3.3.6/31-13
0x0052	eQADC CFIFO Control Register 1 (EQADC_CFCR1)	R/W		
0x0054	eQADC CFIFO Control Register 2 (EQADC_CFCR2)	R/W		
0x0056	eQADC CFIFO Control Register 3 (EQADC_CFCR3)	R/W		
0x0058	eQADC CFIFO Control Register 4 (EQADC_CFCR4)	R/W		
0x005A	eQADC CFIFO Control Register 5 (EQADC_CFCR5)	R/W		

Table 31-1. eQADC Memory Map (continued)

Offset from EQADC_BASE (0xFFFF8_0000)	Register	Access	Reset Value	Section/Page
0x005C	Reserved			
0x0060	eQADC Interrupt and eDMA Control Register 0 (EQADC_IDCR0)	R/W		31.3.3.7/31-15
0x0062	eQADC Interrupt and eDMA Control Register1 (EQADC_IDCR1)	R/W		
0x0064	eQADC Interrupt and eDMA Control Register 2 (EQADC_IDCR2)	R/W		
0x0066	eQADC Interrupt and eDMA Control Register 3 (EQADC_IDCR3)	R/W		
0x0068	eQADC Interrupt and eDMA Control Register 4 (EQADC_IDCR4)	R/W		
0x006A	eQADC Interrupt and eDMA Control Register 5 (EQADC_IDCR5)	R/W		
0x006C	Reserved			
0x0070	eQADC FIFO and Interrupt Status Register 0 (EQADC_FISR0)	R/W		31.3.3.8/31-17
0x0074	eQADC FIFO and Interrupt Status Register 1 (EQADC_FISR1)	R/W		
0x0078	eQADC FIFO and Interrupt Status Register 2 (EQADC_FISR2)	R/W		
0x007C	eQADC FIFO and Interrupt Status Register 3 (EQADC_FISR3)	R/W		
0x0080	eQADC FIFO and Interrupt Status Register 4 (EQADC_FISR4)	R/W		
0x0084	eQADC FIFO and Interrupt Status Register 5 (EQADC_FISR5)	R/W		
0x0088	Reserved			
0x008C	Reserved			

Table 31-1. eQADC Memory Map (continued)

Offset from EQADC_BASE (0xFFFF8_0000)	Register	Access	Reset Value	Section/Page
0x0090	eQADC CFIFO Transfer Counter Register 0 (EQADC_CFTCR0)	R/W		31.3.3.9/31-21
0x0092	eQADC CFIFO Transfer Counter Register 1 (EQADC_CFTCR1)	R/W		
0x0094	eQADC CFIFO Transfer Counter Register 2 (EQADC_CFTCR2)	R/W		
0x0096	eQADC CFIFO Transfer Counter Register 3 (EQADC_CFTCR3)	R/W		
0x0098	eQADC CFIFO Transfer Counter Register 4 (EQADC_CFTCR4)	R/W		
0x009A	eQADC CFIFO Transfer Counter Register 5 (EQADC_CFTCR5)	R/W		
0x009C	Reserved			
0x00A0	eQADC CFIFO Status Snapshot Register (EQADC_CFSSR)	R		31.3.3.10/31-21
0x00A4	Reserved			
0x00A8	Reserved			
0x00AC	eQADC CFIFO Status Register (EQADC_CFSR)	R		31.3.3.11/31-22
0x00B0	Reserved			
0x00B4	Reserved			
0x00B8	Reserved			
0x00BC– 0x00FC	Reserved			
0x0100–0x010C	eQADC CFIFO0 Registers (EQADC_CF0Rw) (w=0, .., 3)	R		31.3.3.12/31-23
0x0110–0x013C	Reserved			
0x0140–0x014C	eQADC CFIFO1 Registers (EQADC_CF1Rw) (w=0, .., 3)	R		31.3.3.12/31-23
0x0150–0x017C	Reserved			
0x0180– 0x018C	eQADC CFIFO2 Registers (EQADC_CF2Rw) (w=0, .., 3)	R		31.3.3.12/31-23
0x0190–0x01BC	Reserved			
0x01C0– 0x01CC	eQADC CFIFO3 Registers (EQADC_CF3Rw) (w=0, .., 3)	R		31.3.3.12/31-23
0x01D0–0x01FC	Reserved			
0x0200– 0x020C	eQADC CFIFO4 Registers (EQADC_CF4Rw) (w=0, .., 3)	R		31.3.3.12/31-23
0x0210–0x023C	Reserved			
0x0240–0x024C	eQADC CFIFO5 Registers (EQADC_CF5Rw) (w=0, .., 3)	R		31.3.3.12/31-23
0x0250–0x02FC	Reserved			

Table 31-1. eQADC Memory Map (continued)

Offset from EQADC_BASE (0xFFFF8_0000)	Register	Access	Reset Value	Section/Page
0x0300–0x030C	eQADC RFIFO0 Registers (EQADC_RF0Rw) (w=0, ..., 3)	R		31.3.3.13/31-24
0x0310–0x033C	Reserved			
0x0340 - 0x034C	eQADC RFIFO1 Registers (EQADC_RF1Rw) (w=0, ..., 3)	R		31.3.3.13/31-24
0x0350–0x037C	Reserved			
0x0380–0x038C	eQADC RFIFO2 Registers (EQADC_RF2Rw) (w=0, ..., 3)	R		31.3.3.13/31-24
0x0390–0x03BC	Reserved			
0x03C0–0x03CC	eQADC RFIFO3 Registers (EQADC_RF3Rw) (w=0, ..., 3)	R		31.3.3.13/31-24
0x03D0–0x03FC	Reserved			
0x0400–0x040C	eQADC RFIFO4 Registers (EQADC_RF4Rw) (w=0, ..., 3)	R		31.3.3.13/31-24
0x0410–0x043C	Reserved			
0x0440–0x044C	eQADC RFIFO5 Registers (EQADC_RF5Rw) (w=0, ..., 3)	R		31.3.3.13/31-24
0x0450–0x07FC	Reserved			

31.3.2 Register Descriptions

This section lists the eQADC registers in address order and describes the registers and their bit fields.

31.3.3 eQADC Register Descriptions

31.3.3.1 eQADC Module Configuration Register (EQADC_MCR)

The EQADC_MCR contains bits used to control how the eQADC responds to a debug mode entry request.

Offset: Base+ 0x0000

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	[Shaded]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DBG	
W	[Shaded]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-2. eQADC Module Configuration Register (EQADC_MCR)

Table 31-2. EQADC_MCR Field Descriptions

Field	Description
bits 0–29	Reserved.
DBG	Debug Enable. Defines the eQADC response to a debug mode entry request. 00 Do not enter debug mode 01 Reserved 10 Enter debug mode. If the eQADC SSI is enabled, FCK stops while the eQADC is in debug mode. 11 Enter debug mode. If the eQADC SSI is enabled, FCK is free running while the eQADC is in debug mode

31.3.3.2 eQADC Null Message Send Format Register (EQADC_NMSFR)

The EQADC_NMSFR defines the format of the null message sent to the external device.

Offset: Base + 0x0008

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	NMF									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	NMF															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-3. eQADC Null Message Send Format Register (EQADC_NMSFR)**Table 31-3. EQADC_NMSFR Field Descriptions**

Field	Description
bits 0–5	Reserved.
NMF	Null Message Format. Contains the programmable null message send value for the eQADC.

NOTE

The eQADC null message send format register affects only how the eQADC sends a null message, but it has no control on how the eQADC detects a null message on receiving data. The eQADC detects a null message by decoding the MESSAGE_TAG field on the receive data. Refer to [Table 31-26](#) for more information on the MESSAGE_TAG field.

31.3.3.3 eQADC External Trigger Digital Filter Register (EQADC_ETDFR)

The EQADC_ETDFR is used to set the minimum time a signal must be held in a logic state on the CFIFO triggers inputs to be recognized as an edge or level gated trigger. The digital filter length field specifies the minimum number of system clocks that the digital filter counter must count to recognize a logic state change.

Offset: Base + 0x000C

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	DFL			
R	0	0	0	0	0	0	0	0	0	0	0	0				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-4. eQADC External Trigger Digital Filter Register (EQADC_ETDFR)

Table 31-4. EQADC_ETDFR Field Description Table

Field	Description
bits 0–27	Reserved.
DFL	<p>Digital Filter Length. Specifies the minimum number of system clocks that must the digital filter counter must count to recognize a logic state change. The count specifies the sample period of the digital filter which is calculated according to the following equation:</p> $\text{FilterPeriod} = (\text{SystemClockPeriod} \times 2^{\text{DFL}}) + 1(\text{SystemClockPeriod})$ <p>Minimum clock counts for which an ETRIG signal needs to be stable to be passed through the filter are shown in Table 31-5.</p> <p>Note: The DFL field must only be written when the MODEn of all CFIFOs are configured to disabled.</p>

Table 31-5. Minimum Required Time to Valid ETRIG

DFL	Minimum Clock Count	Minimum Time (ns) (System Clock = 66 MHz)
0b0000	2	30.30
0b0001	3	45.45
0b0010	5	75.76
0b0011	9	136.36
0b0100	17	257.58
0b0101	33	500.00
0b0110	65	984.85
0b0111	129	1954.55
0b1000	257	3893.94
0b1001	513	7772.73
0b1010	1025	15530.30
0b1011	2049	31045.45
0b1100	4097	62075.76
0b1101	8193	124136.36

Table 31-5. Minimum Required Time to Valid ETRIG (continued)

DFL	Minimum Clock Count	Minimum Time (ns) (System Clock = 66 MHz)
0b1110	16385	248257.58
0b1111	32769	496500.00

31.3.3.4 eQADC CFIFO Push Registers 0–5 (EQADC_CFPR n)

The EQADC_CFPRs provide a mechanism to fill the CFIFOs with command messages from the command queues. Refer to [Section 31.4.3, “eQADC Command FIFOs,”](#) for more information on the CFIFOs and to [Section 31.4.1.1, “Message Format in eQADC,”](#) for a description on command message formats.

Offset: Base + 0x0010 (EQADC_CFPR0)

Access: Write

Base + 0x0014 (EQADC_CFPR1);

Base + 0x0018 (EQADC_CFPR2)

Base + 0x001C (EQADC_CFPR3)

Base + 0x0020 (EQADC_CFPR4)

Base + 0x0024 (EQADC_CFPR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	CF_PUSH n															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	CF_PUSH n															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-5. eQADC CFIFO Push Registers 0–5 (EQADC_CFPR n)Table 31-6. EQADC_CFPR n Field Descriptions

Field	Description
CF_PUSH n	<p>CFIFO Push Data n. When CFIFOn is not full, writing to the whole word or any bytes of EQADC_CFPRn will push the 32-bit CF_PUSHn value into CFIFOn. Writing to the CF_PUSHn field also increments the corresponding CFCTRn value by one in Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn),” When the CFIFOn is full, the eQADC ignores any write to the CF_PUSHn. Reading the EQADC_CFPRn always returns 0.</p> <p>Note: Only whole words must be written to EQADC_CFPR. Writing halfwords or bytes to EQADC_CFPR will still push the whole 32-bit CF_PUSH field into the corresponding CFIFO, but undefined data will fill the areas of CF_PUSH that were not specifically designated as target locations for the write.</p>

31.3.3.5 eQADC Result FIFO Pop Registers 0–5 (EQADC_RFPR n)

The eQADC_RFPRs provide a mechanism to retrieve data from RFIFOs.

NOTE

The EQADC_RFPR n must not be read speculatively. For future compatibility, the TLB entry covering the EQADC_RFPR n must be configured to be guarded.

Offset: Base + 0x0030 (EQADC_RFPR0)

Access: Read

Base + 0x0034 (EQADC_RFPR1)

Base + 0x0038 (EQADC_RFPR2)

Base + 0x003C (EQADC_RFPR3)

Base + 0x0040 (EQADC_RFPR4)

Base + 0x0044 (EQADC_RFPR5)

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

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RF_POP n															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-6. eQADC RFIFO Pop Registers 0–5 (EQADC_RFPR n)**Table 31-7. EQADC_RFPR n Field Descriptions**

Field	Description
bits 0–15	Reserved.
RF_POP n	Result FIFO Pop Data n . When RFIFO n is not empty, the RF_POP n contains the next unread entry value of RFIFO n . Reading the whole word, a halfword, or any bytes of EQADC_RFPR n will pop one entry from RFIFO n , and the corresponding RFCTR n value will be decremented by 1 (See Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”). When the RFIFO n is empty, any read on EQADC_RFPR n returns undefined data value and does not decrement the RFCTR n value. Writing to EQADC_RFPR n has no effect.

31.3.3.6 eQADC CFIFO Control Registers 0–5 (EQADC_CFCR n)

The eQADC_CFCRs contain bits that affect CFIFOs. These bits specify the CFIFO operation mode and can invalidate all of the CFIFO contents.

Offset: EQADC_BASE + 0x0050 (EQADC_CFCR0)

Access: Read/Write

EQADC_BASE + 0x0052 (EQADC_CFCR1)

EQADC_BASE + 0x0054 (EQADC_CFCR2)

EQADC_BASE + 0x0056 (EQADC_CFCR3)

EQADC_BASE + 0x0058 (EQADC_CFCR4);

EQADC_BASE + 0x005A (EQADC_CFCR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MODE n				0	0	0	0
W						SSE n	CFINV n		MODE n							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-7. eQADC CFIFO Control Registers (EQADC_CFCR n)

Table 31-8. EQADC_CFCR n Field Descriptions

Field	Description
bits 0–4	Reserved.
SSE n	CFIFO Single-Scan Enable Bit n . Used to set the SSS n bit, as described in Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)” . Writing a 1 to SSE n will set the SSS n if the CFIFO is in single-scan mode. When SSS n is already asserted, writing a 1 to SSE n has no effect. If the CFIFO is in continuous-scan mode or is disabled, writing a 1 to SSE n will not set SSS n . Writing a 0 to SSE n has no effect. SSE n always is read as 0. 0 No effect. 1 Set the SSS n bit.
CFINV n	CFIFO Invalidate Bit n . Causes the eQADC to invalidate all entries of CFIFO n . Writing a 1 to CFINV n will reset the value of CFCTR n in the EQADC_FISR register (refer to Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”). Writing a 1 to CFINV n also resets the push next data pointer, transfer next data pointer to the first entry of CFIFO n in Figure 31-27 . CFINV n always is read as 0. Writing a 0 has no effect. 0 No effect. 1 Invalidate all of the entries in the corresponding CFIFO. Note: Writing CFINV n only invalidates commands stored in CFIFO n ; previously transferred commands that are waiting for execution, that is commands stored in the ADC command buffer, will still be executed, and results generated by them will be stored in the appropriate RFIFO. Note: CFINV n must not be written unless the MODE n is configured to disabled, and CFIFO status is IDLE.
bit 7	Reserved.
MODE n	CFIFO Operation Mode n . Selects the CFIFO operation mode for CFIFO n . Refer to Section 31.4.3.5, “CFIFO Scan Trigger Modes,” for more information on CFIFO trigger mode. Note: If MODE n is not disabled, it must not be changed to any other mode besides disabled. If MODE n is disabled and the CFIFO status is IDLE, MODE n can be changed to any other mode.
bits 12–15	Reserved.

Table 31-9. CFIFO Operation Mode Table

MODE n	CFIFO Operation Mode
0b0000	Disabled
0b0001	Software trigger, single scan
0b0010	Low-level gated external trigger, single scan
0b0011	High-level gated external trigger, single scan
0b0100	Falling-edge external trigger, single scan
0b0101	Rising-edge external trigger, single scan
0b0110	Falling- or rising-edge external trigger, single scan
0b0111–0b1000	Reserved
0b1001	Software trigger, continuous scan
0b1010	Low-level gated external trigger, continuous scan
0b1011	High-level gated external trigger, continuous scan
0b1100	Falling-edge external trigger, continuous scan
0b1101	Rising-edge external trigger, continuous scan

Table 31-9. CFIFO Operation Mode Table (continued)

MODE n	CFIFO Operation Mode
0b1110	Falling- or rising-edge external trigger, continuous scan
0b1111	Reserved

31.3.3.7 eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCR n)

The eQADC_IDCRs contain bits to enable the generation of interrupt or eDMA requests when the corresponding flag bits are set in EQADC_FISR n (Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR n)”).

Offset: EQADC_BASE + 0x0060 (EQADC_IDCR0)
 EQADC_BASE + 0x0062 (EQADC_IDCR1)
 EQADC_BASE + 0x0064 (EQADC_IDCR2)
 EQADC_BASE + 0x0066 (EQADC_IDCR3)
 EQADC_BASE + 0x0068 (EQADC_IDCR4)
 EQADC_BASE + 0x006A (EQADC_IDCR5)

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCI	TORI	PIE n	EOQI	CFUI	0	CFF	CFF	0	0	0	0	RFOI	0	RFD	RFD
W	En	En		En	En		En	Sn					En		En	Sn
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-8. eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCR n)

Table 31-10. EQADC_IDCR n Field Descriptions

Field	Description
NCIE n	Non-Coherency Interrupt Enable n . Enables the eQADC to generate an interrupt request when the corresponding NCF n , described in Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR n)”, is asserted. 0 Disable non-coherency interrupt request 1 Enable non-coherency interrupt request
TORIE n	Trigger Overrun Interrupt Enable n . Enables the eQADC to generate an interrupt request when the corresponding TORF n (described in Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR n)”) is asserted. Apart from generating an independent interrupt request for a CFIFO n trigger overrun event, the eQADC also provides a combined interrupt at which the result FIFO overflow interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun interrupt requests of all CFIFOs are ORed. When RFOIE n , CFUIE n , and TORIE n are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOF n , CFUF n , and TORF n (assuming that all interrupts are enabled). See Section 31.4.7, “eQADC eDMA/Interrupt Request,” for details. 0 Disable trigger overrun interrupt request 1 Enable trigger overrun interrupt request
PIE n	Pause Interrupt Enable n . Enables the eQADC to generate an interrupt request when the corresponding PF x in EQADC_FISR n (See Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR n)”) is asserted. 0 Disable pause interrupt request 1 Enable pause interrupt request

Table 31-10. EQADC_IDCR n Field Descriptions (continued)

Field	Description
EOQIE n	End-of-Queue Interrupt Enable n . Enables the eQADC to generate an interrupt request when the corresponding EOQF n in EQADC_FISR n (See Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) is asserted. 0 Disable end of queue interrupt request. 1 Enable end of queue interrupt request.
CFUIE n	CFIFO Underflow Interrupt Enable n . Enables the eQADC to generate an interrupt request when the corresponding CFUF n in EQADC_FISR n (See Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) is asserted. Apart from generating an independent interrupt request for a CFIFO n underflow event, the eQADC also provides a combined interrupt at which the result FIFO overflow interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun interrupt requests of all CFIFOs are ORed. When RFOIE n , CFUIE n , and TORIE n are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOF n , CFUF n , and TORF n (assuming that all interrupts are enabled). See Section 31.4.7, “eQADC eDMA/Interrupt Request,” for details. 0 Disable underflow interrupt request 1 Enable underflow interrupt request
bit 5	Reserved.
CFFE n	CFIFO Fill Enable n . Enables the eQADC to generate an interrupt request (CFFS n is asserted) or eDMA request (CFFS n is negated) when CFFF n in EQADC_FISR n (Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) is asserted. 0 Disable CFIFO fill eDMA or interrupt request 1 Enable CFIFO fill eDMA or interrupt request Note: CFFE n must not be negated while an eDMA transaction is in progress.
CFFS n	CFIFO Fill Select n . Selects if an eDMA or interrupt request is generated when CFFF n in EQADC_FISR n (See Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) is asserted. If CFFE n is asserted, the eQADC generates an interrupt request when CFFS n is negated, or it generates an eDMA request if CFFS n is asserted. 0 Generate interrupt request to move data from the system memory to CFIFO n . 1 Generate eDMA request to move data from the system memory to CFIFO n . Note: DMA access is not supported on CFIFO 2 to 5. These FIFOs can be filled only by interrupt requests. Note: CFFS n must not be negated while an eDMA transaction is in progress.
bits 8–11	Reserved.
RFOIE n	RFIFO Overflow Interrupt Enable n . Enables the eQADC to generate an interrupt request when the corresponding RFOF n in EQADC_FISR n (See Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn)”) is asserted. Apart from generating an independent interrupt request for an RFIFO n overflow event, the eQADC also provides a combined interrupt at which the result FIFO overflow Interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun interrupt requests of all CFIFOs are ORed. When RFOIE n , CFUIE n , and TORIE n are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOF n , CFUF n , and TORF n (assuming that all interrupts are enabled). See Section 31.4.7, “eQADC eDMA/Interrupt Request,” for details. 0 Disable overflow interrupt request 1 Enable overflow Interrupt request
bit 13	Reserved.

Table 31-10. EQADC_IDCR n Field Descriptions (continued)

Field	Description
RFDE n	RFIFO Drain Enable n . Enables the eQADC to generate an interrupt request (RFDS n is asserted) or eDMA request (RFDS n is negated) when RFDF n in EQADC_FISR n (See Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR n)”) is asserted. 0 Disable RFIFO drain eDMA or interrupt request 1 Enable RFIFO drain eDMA or interrupt request Note: RFDE n must not be negated while an eDMA transaction is in progress.
RFDS n	RFIFO Drain Select n . Selects if an eDMA or interrupt request is generated when RFDF n in EQADC_FISR n (See Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR n)”) is asserted. If RFDE n is asserted, the eQADC generates an interrupt request when RFDS n is negated, or it generates an eDMA request when RFDS n is asserted. 0 Generate interrupt request to move data from RFIF n to the system memory 1 Generate eDMA request to move data from RFIFO n to the system memory Note: DMA access is not supported on CFIFO 2 to 5. These FIFOs can be filled only by interrupt requests. Note: RFDS n must not be negated while an eDMA transaction is in progress.

31.3.3.8 eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR n)

The EQADC_FISRs contain flag and status bits for each CFIFO and RFIFO pair. Writing 1 to a flag bit clears it. Writing 0 has no effect. Status bits are read only. These bits indicate the status of the FIFO itself.

Offset: Base + 0x0070 (EQADC_FISR0)

Access: Read/Write to clear

Base + 0x0074 (EQADC_FISR1)

Base + 0x0078 (EQADC_FISR2)

Base + 0x007C (EQADC_FISR3)

Base + 0x0080 (EQADC_FISR4)

Base + 0x0084 (EQADC_FISR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NCF n	TORF n	PF n	EOQF n	CFUF n	SSS n	CFFF n	0	0	0	0	0	RFOF n	0	RFDF n	0
W	w1c	w1c	w1c	w1c	w1c		w1c						w1c		w1c	
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFCTR n				TNXTPTR n				RFCTR n				POPXTPTR n			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-9. eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR n)

Table 31-11. EQADC_FISR n Field Descriptions

Field	Description
NCF n	<p>Non-Coherency Flag n. NCFn is set whenever a command sequence being transferred through CFIFOn becomes non-coherent. If NCIEn in EQADC_IDCRn (See Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and NCFn are asserted, an interrupt request will be generated. Writing a 1 clears NCFn. Writing a 0 has no effect. More for information on non-coherency refer to Section 31.4.3.6.5, “Command Sequence Non-Coherency Detection.”</p> <p>0 Command sequence being transferred by CFIFOn is coherent 1 Command sequence being transferred by CFIFOn became non-coherent</p> <p>Note: Non-coherency means that a command in the command FIFO was not immediately executed, but delayed. This may occur if the command is pre-empted, where a higher priority queue is triggered and has a competing conversion command for the same converter.</p>
TORF n	<p>Trigger Overrun Flag for CFIFO n. TORFn is set when trigger overrun occurs for the specified CFIFO in edge or level trigger mode. Trigger overrun occurs when an already triggered CFIFO receives an additional trigger. When EQADC_IDCRn[TORIEn] is set (See Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and TORFn are asserted, an interrupt request will be generated.</p> <p>Apart from generating an independent interrupt request for a CFIFOn trigger overrun event, the eQADC also provides a combined interrupt at which the result FIFO overflow interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun Interrupt requests of all CFIFOs are ORed. When RFOIEn, CFUIEn, and TORIEn are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFn, CFUFn, and TORFn (assuming that all interrupts are enabled). See Section 31.4.7, “eQADC eDMA/Interrupt Request,” for details.</p> <p>Write 1 to clear the TORFn bit. Writing 0 has no effect.</p> <p>0 No trigger overrun occurred 1 Trigger overrun occurred</p> <p>Note: The trigger overrun flag will not set for CFIFOs configured for software trigger mode.</p>
PF n	<p>Pause Flag n. PF behavior changes according to the CFIFO trigger mode.</p> <ul style="list-style-type: none"> In edge trigger mode, PFn is set when the eQADC completes the transfer of an entry with an asserted pause bit from CFIFOn. In level trigger mode, when CFIFOn is in the TRIGGERED state, PFn is set when CFIFO status changes from TRIGGERED due to the detection of a closed gate. <p>An interrupt routine, generated due to the asserted PF, can be used to verify if a complete scan of the user-defined command queue was performed. If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status. If a closed gate is detected while a command transfer to an on-chip ADC is taking place, it will only affect the CFIFO status when the transfer completes.</p> <p>The transfer of entries bound for the on-chip ADC is considered completed when they are stored in the ADC command buffer. In software trigger mode, PFn will never become asserted.</p> <p>If PIEn (See Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and PFn are asserted, an interrupt will be generated. Writing a 1 clears the PFn. Writing a 0 has no effect. Refer to Section 31.4.3.6.3, “Pause Status,” for more information on pause flag.</p> <p>0 Entry with asserted pause bit was not transferred from CFIFOn (CFIFO in edge trigger mode), or CFIFO status did not change from the TRIGGERED state due to detection of a closed gate (CFIFO in level trigger mode). 1 Entry with asserted pause bit was transferred from CFIFOn (CFIFO in edge trigger mode), or CFIFO status changes from the TRIGGERED state due to detection of a closed gate (CFIFO in level trigger mode).</p> <p>Note: In edge trigger mode, an asserted PFn only implies that the eQADC has finished transferring a command with an asserted pause bit from CFIFOn. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.</p> <p>Note: In software or level trigger mode, when the eQADC completes the transfer of an entry from CFIFOn with an asserted pause bit, PFn will not be set and transfer of commands will continue without pausing.</p>

Table 31-11. EQADC_FISR n Field Descriptions (continued)

Field	Description
EOQF n	<p>End-of-Queue Flag n. Indicates that an entry with an asserted EOQ bit was transferred from CFIFOn to the on-chip ADCs or to the external device. See Section 31.4.1.1, “Message Format in eQADC,” for details about command message formats. When the eQADC completes the transfer of an entry with an asserted EOQ bit from CFIFOn, EOQFn will be set. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the command buffer. If the EOQIEn bit (See Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and EOQFn are asserted, an interrupt will be generated. Writing a 1 clears the EOQFn bit. Writing a 0 has no effect. Refer to Section 31.4.3.6.2, “Command Queue Completion Status,” for more information on end-of-queue flag.</p> <p>0 Entry with asserted EOQ bit was not transferred from CFIFOn 1 Entry with asserted EOQ bit was transferred from CFIFOn</p> <p>Note: An asserted EOQFn only implies that the eQADC has finished transferring a command with an asserted EOQ bit from CFIFOn. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.</p>
CFUF n	<p>CFIFO Underflow Flag n. Indicates an underflow event on CFIFOn. CFUFn is set when CFIFOn is in the TRIGGERED state and it becomes empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. When CFUIEn (see Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and CFUFn are both asserted, the eQADC generates an interrupt request.</p> <p>Apart from generating an independent interrupt request for a CFIFOn underflow event, the eQADC also provides a combined interrupt at which the result FIFO overflow interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun interrupt requests of all CFIFOs are ORed. When RFOIEn, CFUIEn, and TORIEn are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFn, CFUFn, and TORFn (assuming that all interrupts are enabled). See Section 31.4.7, “eQADC eDMA/Interrupt Request,” for details. Writing a 1 clears CFUFn. Writing a 0 has no effect.</p> <p>0 No CFIFO underflow event occurred 1 A CFIFO underflow event occurred</p>
SSSn	<p>CFIFO Single-Scan Status Bit n. When asserted, enables the detection of trigger events for CFIFOs programmed into single-scan level- or edge-trigger mode, and works as trigger for CFIFOs programmed into single-scan software-trigger mode. Refer to Section 31.4.3.5.2, “Single-Scan Mode,” for further details. The SSSn bit is set by writing a 1 to the SSEn bit (see Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”). The eQADC clears the SSSn bit when a command with an asserted EOQ bit is transferred from a CFIFO in single-scan mode, when a CFIFO is in single-scan level-trigger mode and its status changes from the TRIGGERED state due to the detection of a closed gate, or when the value of the CFIFO operation mode MODEn (see Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn)”) is changed to disabled. Writing to SSSn has no effect. SSSn has no effect in continuous-scan or in disabled mode.</p> <p>0 CFIFO in single-scan level- or edge-trigger mode will ignore trigger events, or CFIFO in single-scan software-trigger mode is not triggered. 1 CFIFO in single-scan level- or edge-trigger mode will detect a trigger event, or CFIFO in single-scan software-trigger mode is triggered.</p>
CFFF n	<p>CFIFO Fill Flag n. CFFFn is set when the CFIFOn is not full. When CFFEn (see Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and CFFFn are both asserted, an interrupt or an eDMA request will be generated depending on the status of the CFFSn bit. When CFFSn is negated (interrupt requests selected), software clears CFFFn by writing a 1 to it. Writing a 0 has no effect. When CFFSn is asserted (eDMA requests selected), CFFFn is automatically cleared by the eQADC when the CFIFO becomes full.</p> <p>0 CFIFOn is full. 1 CFIFOn is not full.</p> <p>Note: When generation of interrupt requests is selected (CFFSn=0), CFFFn must only be cleared in the ISR after the CFIFOn push register is accessed.</p> <p>Note: CFFFn should not be cleared when CFFSn is asserted (eDMA requests selected).</p>
bits 7–11	Reserved.

Table 31-11. EQADC_FISR n Field Descriptions (continued)

Field	Description
RFOF n	<p>RFIFO Overflow Flag n. Indicates an overflow event on RFIFOn. RFOFn is set when RFIFOn is already full, and a new data is received from the on-chip ADCs or from the external device. The RFIFOn will not overwrite older data in the RFIFO, and the new data will be ignored. When RFOIEn (see Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and RFOFn are both asserted, the eQADC generates an interrupt request.</p> <p>Apart from generating an independent interrupt request for an RFIFOn overflow event, the eQADC also provides a combined interrupt at which the result FIFO overflow interrupt, the command FIFO underflow interrupt, and the command FIFO trigger overrun interrupt requests of all CFIFOs are ORed. When RFOIEn, CFUIEn, and TORIEn are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFn, CFUFn, and TORFn (assuming that all interrupts are enabled). See Section 31.4.7, “eQADC eDMA/Interrupt Request,” for details.</p> <p>Write 1 to clear RFOFn. Writing a 0 has no effect.</p> <p>0 No RFIFO overflow event occurred. 1 An RFIFO overflow event occurred.</p>
bit 13	Reserved.
RFDF n	<p>RFIFO Drain Flag n. Indicates if RFIFOn has valid entries that can be drained or not. RFDFn is set when the RFIFOn has at least one valid entry in it. When RFDEn (see Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn)”) and RFDFn are both asserted, an interrupt or an eDMA request will be generated depending on the status of the RFDSn bit. When RFDSn is negated (interrupt requests selected), software clears RFDFn by writing a 1 to it. Writing a 0 has no effect. When RFDSn is asserted (eDMA requests selected), RFDFn is automatically cleared by the eQADC when the RFIFO becomes empty.</p> <p>0 RFIFOn is empty. 1 RFIFOn has at least one valid entry.</p> <p>Note: In the interrupt service routine, RFDF must be cleared only after the RFIFOn pop register is read. Note: RFDFn should not be cleared when RFDSn is asserted (eDMA requests selected).</p>
bit 15	Reserved.
CFCTR n	CFIFO n Entry Counter. Indicates the number of commands stored in the CFIFO n . When the eQADC completes transferring a piece of new data from the CFIFO n , it decrements CFCTR n by 1. Writing a word or any bytes to the corresponding CFIFO Push Register (see Section 31.3.3.4, “eQADC CFIFO Push Registers 0–5 (EQADC_CFPRn)”) increments CFCTR n by 1. Writing any value to CFCTR n has no effect.
TNX TPTR n	CFIFO n Transfer Next Pointer. Indicates the index of the next entry to be removed from CFIFO n when it completes a transfer. When TNXTPTR n is 0, it points to the entry with the smallest memory-mapped address inside CFIFO n . TNXTPTR n is only updated when a command transfer is completed. If the maximum index number (CFIFO depth minus 1) is reached, TNXTPTR n is wrapped to 0, else, it is incremented by 1. For details refer to Section 31.4.3.1, “CFIFO Basic Functionality.” Writing any value to TNXTPTR n has no effect.
RFCTR n	RFIFO n entry counter. Indicates the number of data items stored in the RFIFO n . When the eQADC stores a piece of new data into RFIFO n , it increments RFCTR n by 1. Reading the whole word, halfword or any bytes of the corresponding Result FIFO pop register (see Section 31.3.3.5, “eQADC Result FIFO Pop Registers 0–5 (EQADC_RFPRn)”) decrements RFCTR n by 1. Writing any value to RFCTR n itself has no effect.
POPNX TPTR n	RFIFO n Pop Next Pointer. Indicates the index of the entry that will be returned when EQADC_RFPR n is read. When POPNXTPTR n is 0, it points to the entry with the smallest memory-mapped address inside RFIFO n . POPNXTPTR n is updated when EQADC_RFPR n is read. If the maximum index number (RFIFO depth minus 1) is reached, POPNXTPTR n is wrapped to 0, else, it is incremented by 1. For details refer to Section 31.4.4.1, “RFIFO Basic Functionality.” Writing any value to POPNXTPTR n has no effect.

31.3.3.9 eQADC CFIFO Transfer Counter Registers 0–5 (EQADC_CFTCR n)

The EQADC_CFTCRs record the number of commands transferred from a CFIFO. The EQADC_CFTCR supports the monitoring of command transfers from a CFIFO.

Offset: EQADC_BASE + 0x0090 (EQADC_CFTCR0)

Access: Read/Write

EQADC_BASE + 0x0092 (EQADC_CFTCR1)

EQADC_BASE + 0x0094 (EQADC_CFTCR2)

EQADC_BASE + 0x0096 (EQADC_CFTCR3)

EQADC_BASE + 0x0098 (EQADC_CFTCR4)

EQADC_BASE + 0x009A (EQADC_CFTCR5)

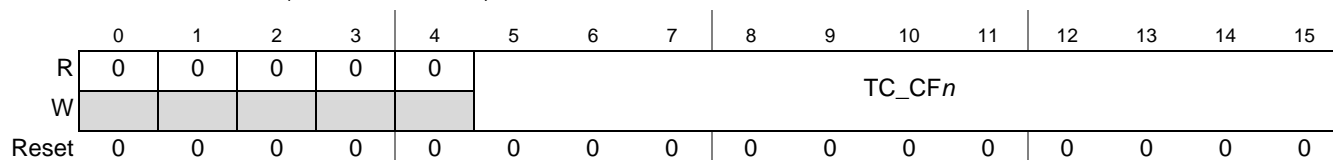


Figure 31-10. eQADC CFIFO Transfer Counter Registers (EQADC_CFTCR n)

Table 31-12. EQADC_CFTCR n Field Descriptions

Field	Description
bits 0–4	Reserved.
TC_CF n	Transfer Counter for CFIFO n . TC_CF n counts the number of commands that have been completely transferred from CFIFO n . TC_CF n =2, for example, signifies that two commands have been transferred. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the command buffer. The eQADC increments the TC_CF n value by 1 after a command is transferred. TC_CF n resets to 0 after eQADC completes transferring a command with an asserted EOQ bit. Writing any value to TC_CF n sets the counter to that written value. Note: If CFIFO n is in the TRIGGERED state when its MODE n field is programmed to disabled, the exact number of entries transferred from the CFIFO until that point (TC_CF n) is only known after the CFIFO status changes to IDLE, as indicated by CFS n . For details refer to Section 31.4.3.5.1, “Disabled Mode.”

31.3.3.10 eQADC CFIFO Status Snapshot Register (EQADC_CFSSR)

The EQADC_CFSSR contains status fields to track the operation status of each CFIFO and the transfer counter of the last CFIFO to initiate a command transfer to the internal ADC. All fields of EQADC_CFSSR are captured at the beginning of a command transfer to the buffer.

Note that captured status register values are associated with a previous command transfer. This means that the EQADC_CFSSR register captures the status register before the status register changes, because of the transfer of the current command that is about to be popped from the CFIFO. The EQADC_CFSSR is read only. Writing to EQADC_CFSSR has no effect.

Offset: Base + 0x00A0

Access: Read

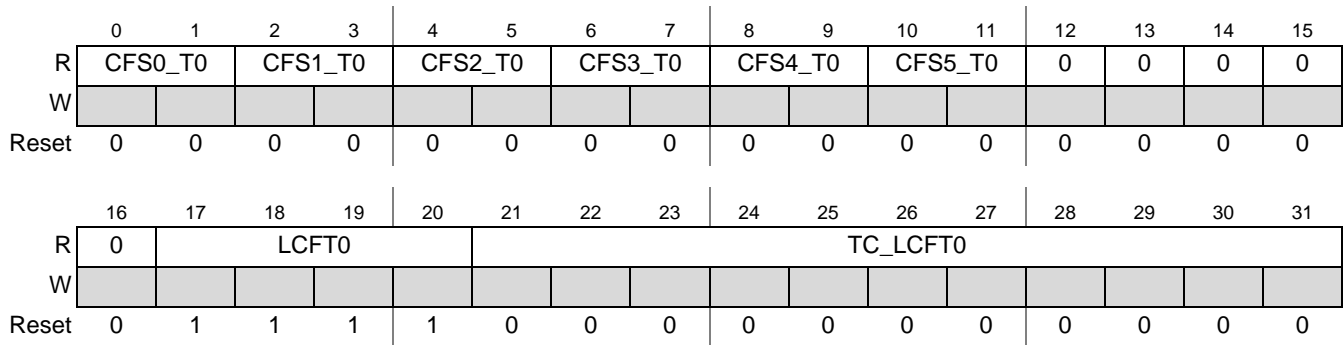


Figure 31-11. eQADC CFIFO Status Snapshot Register (EQADC_CFSSR)

Table 31-13. EQADC_CFSSR Field Descriptions

Field	Description																		
CFS _n _T0	CFIFO Status At Transfer to ADC0 Command Buffer. Indicates the CFIFO _n status at the time a command transfer to ADC0 command buffer is initiated. CFS _n _T0 is a copy of the corresponding CFS _n in EQADC_CFSSR (see Section 31.3.3.11, “eQADC CFIFO Status Register (EQADC_CFSSR)”) captured at the time a command transfer to buffer _n is initiated.																		
bits 12–16	Reserved.																		
LCFT0	Last CFIFO to Transfer to ADC0 Command Buffer. Holds the CFIFO number of last CFIFO to have initiated a command transfer to ADC0 command buffer. LCFT0 has the following values: <table border="1" style="margin-left: 40px;"> <thead> <tr> <th>LCFT0</th> <th>LCFT0 Meaning</th> </tr> </thead> <tbody> <tr> <td>0b0000</td> <td>Last command was transferred from CFIFO0</td> </tr> <tr> <td>0b0001</td> <td>Last command was transferred from CFIFO1</td> </tr> <tr> <td>0b0010</td> <td>Last command was transferred from CFIFO2</td> </tr> <tr> <td>0b0011</td> <td>Last command was transferred from CFIFO3</td> </tr> <tr> <td>0b0100</td> <td>Last command was transferred from CFIFO4</td> </tr> <tr> <td>0b0101</td> <td>Last command was transferred from CFIFO5</td> </tr> <tr> <td>0b0110–0b1110</td> <td>Reserved</td> </tr> <tr> <td>0b1111</td> <td>No command was transferred to ADC0 command buffer</td> </tr> </tbody> </table>	LCFT0	LCFT0 Meaning	0b0000	Last command was transferred from CFIFO0	0b0001	Last command was transferred from CFIFO1	0b0010	Last command was transferred from CFIFO2	0b0011	Last command was transferred from CFIFO3	0b0100	Last command was transferred from CFIFO4	0b0101	Last command was transferred from CFIFO5	0b0110–0b1110	Reserved	0b1111	No command was transferred to ADC0 command buffer
LCFT0	LCFT0 Meaning																		
0b0000	Last command was transferred from CFIFO0																		
0b0001	Last command was transferred from CFIFO1																		
0b0010	Last command was transferred from CFIFO2																		
0b0011	Last command was transferred from CFIFO3																		
0b0100	Last command was transferred from CFIFO4																		
0b0101	Last command was transferred from CFIFO5																		
0b0110–0b1110	Reserved																		
0b1111	No command was transferred to ADC0 command buffer																		
TC_LCFT0	Transfer Counter for Last CFIFO to Transfer Commands to ADC0 Command Buffer. Indicates the number of commands which have been completely transferred from CFIFO _n when a command transfer from CFIFO _n to ADC0 command buffer is initiated. TC_LCFT0 is a copy of the corresponding TC_CF _n in EQADC_CFTCR _n (see Section 31.3.3.9) captured at the time a command transfer from CFIFO _n to ADC0 command buffer is initiated. This field has no meaning when LCFT0 is 0b1111.																		

31.3.3.11 eQADC CFIFO Status Register (EQADC_CFSR)

The EQADC_CFSR contains the current CFIFO status. The EQADC_CFSRs are read only. Writing to the EQADC_CFSR has no effect.

Offset: Base + 0x00AC

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFS0		CFS1		CFS2		CFS3		CFS4		CFS5		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-12. eQADC CFIFO Status Register (EQADC_CFSR)

Table 31-14. EQADC_CFSR Field Descriptions

Field	Description
CFS n	CFIFO Status. Indicates the current status of CFIFO n . Refer to Table 31-15 for more information on CFIFO status.
bits 12–31	Reserved.

Table 31-15. Current CFIFO Status

CFIFO Status	Field Value	Explanation
IDLE	0b00	<ul style="list-style-type: none"> CFIFO is disabled. CFIFO is in single-scan edge or level trigger mode and does not have EQADC_FISRn[SSS] asserted. eQADC completed the transfer of the last entry of the user defined command queue in single-scan mode.
Reserved	0b01	Not applicable.
WAITING FOR TRIGGER	0b10	<ul style="list-style-type: none"> CFIFO mode is modified to continuous-scan edge or level trigger mode. CFIFO mode is modified to single-scan edge or level trigger mode and EQADC_FISRn[SSS] is asserted. CFIFO mode is modified to single-scan software trigger mode and EQADC_FISRn[SSS] is negated. CFIFO is paused. eQADC transferred the last entry of the queue in continuous-scan edge trigger mode.
TRIGGERED	0b11	CFIFO is triggered

31.3.3.12 eQADC CFIFO Registers (EQADC_CF[0–5]R n)

EQADC_CF[0–5]R n provide visibility of the contents of a CFIFO for debugging purposes. Each CFIFO has four registers that are uniquely mapped to its four 32-bit entries. Refer to [Section 31.4.3, “eQADC Command FIFOs,”](#) for more information on CFIFOs. These registers are read only. Data written to these registers is ignored.

Offset: CFIFO0: Base + 0x0100 (CF0R0)
 Base + 0x0104 (CF0R1)
 Base + 0x0108 (CF0R2)
 Base + 0x010C (CF0R3)
 CFIFO1: Base + 0x0140 (CF1R0)
 Base + 0x0144 (CF1R1)
 Base + 0x0148 (CF1R2)
 Base + 0x014C (CF1R3)
 CFIFO2: Base + 0x0180 (CF2R0)
 Base + 0x0184 (CF2R1)
 Base + 0x0188 (CF2R2)
 Base + 0x018C (CF2R3)
 CFIFO3: Base + 0x01C0 (CF3R0)
 Base + 0x01C4 (CF3R1)
 Base + 0x01C8 (CF3R2)
 Base + 0x01CC (CF3R3)
 CFIFO4: Base + 0x0200 (CF4R0)
 Base + 0x0204 (CF4R1)
 Base + 0x0208 (CF4R2)
 Base + 0x020C (CF4R3)
 CFIFO5: Base + 0x0240 (CF5R0)
 Base + 0x0244 (CF5R1)
 Base + 0x0248 (CF5R2)
 Base + 0x024C (CF5R3)

Access: Read

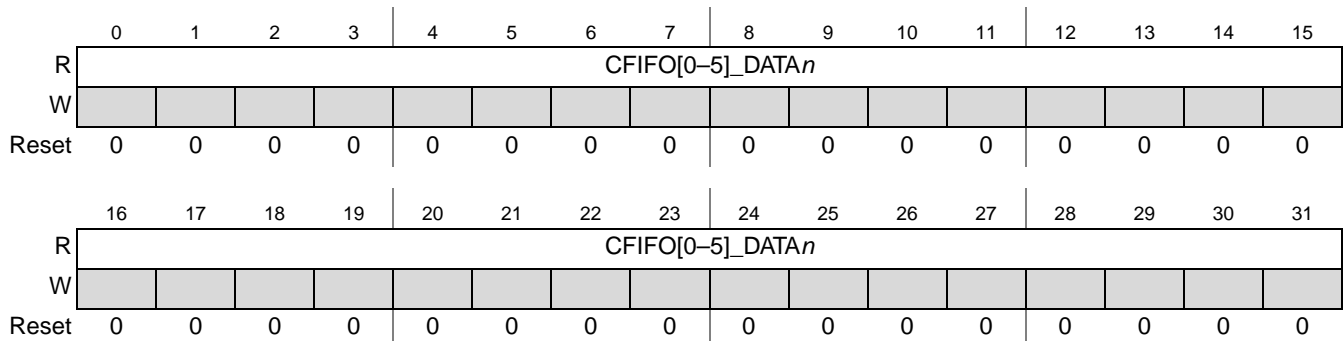


Figure 31-13. eQADC CFIFO[0-5] Registers (EQADC_CF[0-5]Rn)

Table 31-16. EQADC_CF[0-5]Rn Field Descriptions

Field	Description
CFIFO[0-5]_DATA _n	CFIFO[0-5]_data _n . Returns the value stored within the entry of CFIFO[0-5]. Each CFIFO is composed of four 32-bit entries, with register 0 being mapped to the entry with the smallest memory mapped address.

31.3.3.13 eQADC RFIFO Registers (EQADC_RF[0-5]Rn)

EQADC_RF[0-5]Rn provide visibility of the contents of a RFIFO for debugging purposes. Each RFIFO has four registers which are uniquely mapped to its four 16-bit entries. Refer to [Section 31.4.4, “Result FIFOs,”](#) for more information on RFIFOs. These registers are read only. Data written to these registers is ignored.

Offset: RFIFO0: Base + 0x0300 (RF0R0)

Base + 0x0304 (RF0R1)

Base + 0x0308 (RF0R2)

Base+0x030C (RF0R3)

RFIFO1: Base + 0x0340 (RF1R0)

Base + 0x0344 (RF1R1)

Base + 0x0348 (RF1R2)

Base + 0x034C (RF1R3)

RFIFO2: Base + 0x0380 (RF2R0)

Base + 0x0384 (RF2R1)

Base + 0x0388 (RF2R2)

Base + 0x038C (RF2R3)

RFIFO3: Base + 0x03C0 (RF3R0)

Base + 0x03C4 (RF3R1)

Base + 0x03C8 (RF3R2)

Base + 0x03CC (RF3R3)

RFIFO4: Base + 0x0400 (RF4R0)

Base + 0x0404 (RF4R1)

Base + 0x0408 (RF4R2)

Base + 0x040C (RF4R3)

RFIFO5: Base + 0x0440 (RF5R0)

Base + 0x0444 (RF5R1)

Base + 0x0448 (RF5R2)

Base + 0x044C (RF5R3)

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RFIFO[0-5]_DATA _n															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-14. eQADC RFIFO_n Registers (EQADC_RF[0-5]R_n)

Table 31-17. EQADC_RF[0-5]R_n Field Descriptions

Field	Description
RFIFO[0-5]_DATA _n	RFIFO[0-5] data <i>n</i> . Returns the value stored within the entry of RFIFO[0-5]. Each RFIFO is composed of four 16-bit entries, with register 0 being mapped to the entry with the smallest memory mapped address.

31.3.4 On-Chip ADC Registers

This section describes registers that control on-chip ADC operation. The ADC registers are not part of the CPU accessible memory map. These registers can be accessed indirectly through configuration commands only. There are five non-memory-mapped registers for ADC0. The address, usage, and access privilege of each register is shown in Table 31-18. Data written to or read from reserved areas of the memory map is undefined.

Their assigned addresses are the values used to set the ADC_REG_ADDRESS field of the read/write configuration commands bound for the on-chip ADC. These are halfword addresses. Further, the following restrictions apply when accessing these registers:

- Registers ADC0_CR, ADC0_GCCR, and ADC0_OCCR can be accessed by configuration commands sent to the ADC0 command buffer only.
- Registers ADC_TSCR and ADC_TBCR can be accessed by configuration commands sent to the ADC0 command buffer.

Table 31-18. ADC0 Registers

ADC0 Register Address	Register	Access	Reset Value	Section/Page
0x0000	ADC0 Address 0x00 is used for conversion command messages.			
0x0001	ADC0_CR — ADC0 Control Register	R/W		31.3.4.1/31-26
0x0002	ADC_TSCR — ADC Time Stamp Control Register	R/W		31.3.4.2/31-28
0x0003	ADC_TBCR — ADC Time Base Counter Register	R/W		31.3.4.3/31-29
0x0004	ADC0_GCCR — ADC0 Gain Calibration Constant Register	R/W		31.3.4.4/31-30
0x0005	ADC0_OCCR — ADC0 Offset Calibration Constant Register	R/W		31.3.4.5/31-31
0x0006–0x00FF	Reserved			

31.3.4.1 ADC0 Control Register (ADC0_CR)

The ADC0 control register (ADC0_CR) is used to configure the on-chip ADC.

Offset: 0x0001

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ADC0	0	0	0	ADC0_	0	0	0	0	0	0	ADC0_CLK_PS				
W	_EN					EMUX										
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Figure 31-15. ADC0 Control Registers (ADC0_CR)

Table 31-19. ADC0_CR Field Descriptions

Field	Description
ADC0_EN	<p>ADC0 Enable. Enables ADC0 to perform A/D conversions. Refer to Section 31.4.5.1, “Enabling and Disabling the on-chip ADC,” for details.</p> <p>0 ADC is disabled. Clock supply to ADC0 is stopped.</p> <p>1 ADC is enabled and ready to perform A/D conversions.</p> <p>Note: The bias generator circuit inside the ADC ceases functioning when this bit is negated.</p> <p>Note: Conversion commands sent to a disabled ADC are ignored by the ADC control hardware.</p> <p>Note: When the ADC0_EN status is changed from asserted to negated, the ADC clock will not stop until it reaches its low phase.</p>
bits 1–3	Reserved.
ADC0_EMUX	<p>ADC0 External Multiplexer Enable. When ADC0_EMUX is asserted, the MA pins will output digital values according to the number of the external channel being converted for selecting external multiplexer inputs. Refer to Section 31.4.6, “Internal/External Multiplexing,” for a detailed description about how ADC0_EMUX affects channel number decoding.</p> <p>0 External multiplexer disabled; no external multiplexer channels can be selected.</p> <p>1 External multiplexer enabled; external multiplexer channels can be selected.</p> <p>Note: The ADC0_EMUX bit must only be written when the ADC0_EN bit is negated. ADC0_EMUX can be set during the same write cycle used to set ADC0_EN.</p>
5–10	Reserved.
ADC0_CLK_PS	<p>ADC0 Clock Prescaler. The ADC0_CLK_PS field controls the system clock divide factor for the ADC0 clock as in Table 31-20. See Section 31.4.5.2, “ADC Clock and Conversion Speed,” for details about how to set ADC0_CLK_PS.</p> <p>The ADC0_CLK_PS field must only be written when the ADC0_EN bit is negated. This field can be configured during the same write cycle used to set ADC0_EN.</p>

Table 31-20. System Clock Divide Factor for ADC Clock

ADC0_CLK_PS	System Clock Divide Factor
0b00000	2
0b00001	4
0b00010	6
0b00011	8
0b00100	10
0b00101	12
0b00110	14
0b00111	16
0b01000	18
0b01001	20
0b01010	22
0b01011	24
0b01100	26

Table 31-20. System Clock Divide Factor for ADC Clock (continued)

ADC0_CLK_PS	System Clock Divide Factor
0b01101	28
0b01110	30
0b01111	32
0b10000	34
0b10001	36
0b10010	38
0b10011	40
0b10100	42
0b10101	44
0b10110	46
0b10111	48
0b11000	50
0b11001	52
0b11010	54
0b11011	56
0b11100	58
0b11101	60
0b11110	62
0b11111	64

31.3.4.2 ADC Time Stamp Control Register (ADC_TSCR)

The ADC_TSCR contains a system clock divide factor used in the making of the time base counter clock. It determines at what frequency the time base counter will run. ADC_TSCR can be accessed by configuration commands sent to ADC0.

Offset: 0x0002

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	TBC_CLK_PS			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-16. ADC Time Stamp Control Register (ADC_TSCR)

Table 31-21. ADC_TSCR Field Descriptions

Field	Description
0–11	Reserved.
TBC_CLK_PS	Time Base Counter Clock Prescaler. Contains the system clock divide factor for the time base counter. It controls the accuracy of the time stamp. The prescaler is disabled when TBC_CLK_PS is set to 0b0000.

Table 31-22. Clock Divide Factor for Time Stamp

TBC_CLK_PS	System Clock Divide Factor	Clock to Time Stamp Counter for a 66 MHz System Clock (MHz)
0b0000	Disabled	Disabled
0b0001	1	66
0b0010	2	33
0b0011	4	16.5
0b0100	6	11
0b0101	8	8.25
0b0110	10	6.60
0b0111	12	5.50
0b1000	16	4.13
0b1001	32	2.06
0b1010	64	1.03
0b1011	128	0.52
0b1100	256	0.26
0b1101	512	0.13
0b1110–0b1111	Reserved	—

NOTE

If TBC_CLK_PS is not set to disabled, it must not be changed to any other value besides disabled. If TBC_CLK_PS is set to disabled it can be changed to any other value.

31.3.4.3 ADC Time Base Counter Registers (ADC_TBCR)

The ADC_TBCR contains the current value of the time base counter. ADC_TBCR can be accessed by configuration commands sent to ADC0.

Offset: 0x0003

Access: Read/Write

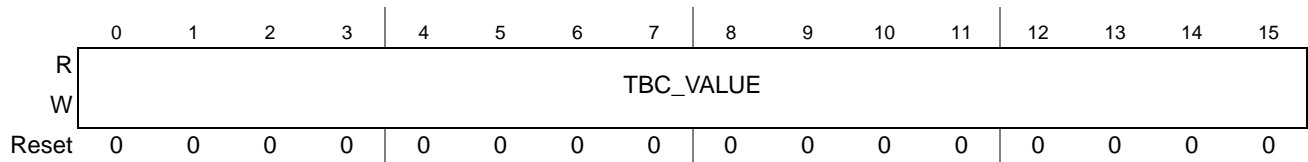


Figure 31-17. ADC Time Base Counter Register (ADC_TBCR)

Table 31-23. ADC_TBCR Field Descriptions

Field	Description
TBC_VALUE	Time Base Counter VALUE. Contains the current value of the time base counter. Reading TBC_VALUE returns the current value of time base counter. Writes to TBC_VALUE register load the written data to the counter. The time base counter counts from 0x0000 to 0xFFFF and wraps when reaching 0xFFFF.

31.3.4.4 ADC0 Gain Calibration Constant Register (ADC0_GCCR)

The ADC0_GCCR contains the gain calibration constant used to fine-tune the ADC0 conversion results. Refer to [Section 31.4.5.4, “ADC Calibration Feature,”](#) for details about the calibration scheme used in the eQADC.

Offset: 0x0004

Access: Read/Write

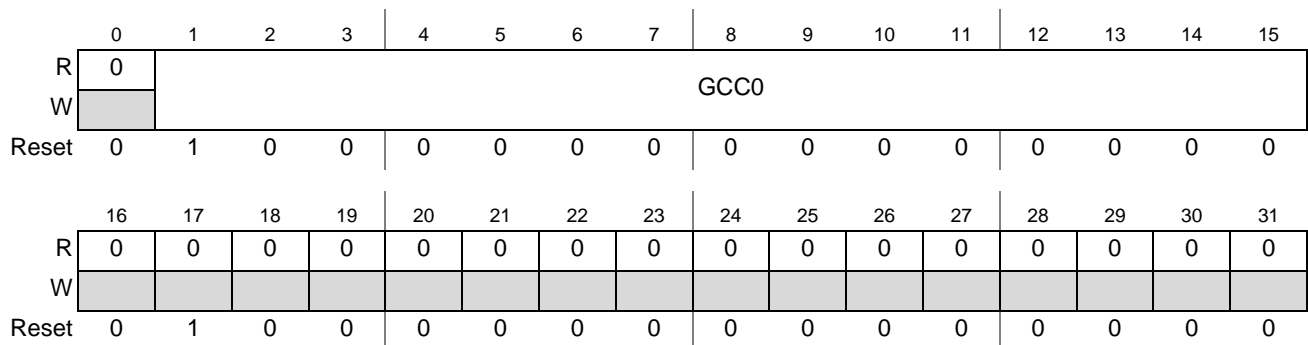


Figure 31-18. ADC0 Gain Calibration Constant Register (ADC0_GCCR)

Table 31-24. ADC0_GCCR Field Descriptions

Field	Description
bit 0	Reserved.
GCCn	ADC0 Gain Calibration Constant. Contains the gain calibration constant used to fine-tune ADC0 conversion results. It is a unsigned 15-bit fixed pointed value. The gain calibration constant is an unsigned fixed point number expressed in the <i>GCC_INT.GCC_FRAC</i> binary format. The integer part of the gain constant (GCC_INT) contains a single binary digit while its fractional part (GCC_FRAC) contains 14 digits. For details about the GCC data format refer to Section 31.4.5.4.2, “MAC Unit and Operand Data Format.”

31.3.4.5 ADC0 Offset Calibration Constant Register (ADC0_OCCR)

The ADC0_OCCR contains the offset calibration constant used to fine-tune the ADC0 conversion results. The offset constant is a signed 14-bit integer value. Refer to [Section 31.4.5.4, “ADC Calibration Feature,”](#) for details about the calibration scheme used in the eQADC.

Offset: 0x0005

Access: Read/Write

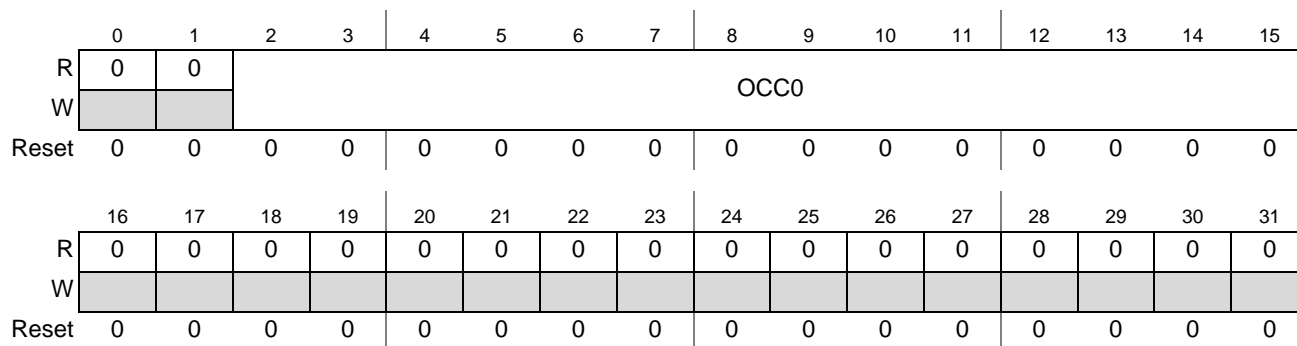


Figure 31-19. ADC0 Offset Calibration Constant Registers (ADC0_OCCR)

Table 31-25. ADC0_OCCR Field Descriptions

Field	Description
0–1	Reserved.
OCC n	ADC0 Offset Calibration Constant. Contains the offset calibration constant used to fine-tune ADC0 conversion results. Negative values should be expressed using the two's complement representation.

31.4 Functional Description

The eQADC provides an interface to an on-chip ADC.

Initially, command data is contained in system memory in a user-defined data queue structure. Command data is moved between the user-defined queues and CFIFOs by the host CPU or by the eDMA which responds to interrupt and eDMA requests generated by the eQADC. The eQADC supports software and hardware triggers from other modules or external pins to initiate transfers of commands from the multiple CFIFOs to the on-chip ADC.

CFIFOs can be configured to be in single-scan or continuous-scan mode. When a CFIFO is configured to be in single-scan mode, the eQADC scans the user-defined command queue one time. The eQADC stops transferring commands from the triggered CFIFO after detecting the EOQ bit set in the last transfer. After an EOQ bit is detected, software involvement is required to rearm the CFIFO so that it can detect new trigger events.

When a CFIFO is configured for continuous-scan mode, the whole user command queue is scanned multiple times. After the detection of an asserted EOQ bit in the last command transfer, command transfers can continue or not depending on the mode of operation of the CFIFO.

The eQADC can also in parallel and independently of the CFIFOs receive data from the on-chip ADC into multiple RFIFOs. Result data is moved from the RFIFOs to the user-defined result queues in system memory by the host CPU or by the eDMA.

31.4.1 Data Flow in the eQADC

Figure 31-20 shows how command data flows inside the eQADC system. A command message is the predefined format in which command data is stored in the user-defined command queues. A command message has 32 bits and is composed of two parts: a CFIFO header and an ADC command. Command messages are moved from the user command queues to the CFIFOs by the host CPU or by the eDMA as they respond to interrupt and eDMA requests generated by the eQADC. The eQADC generates these requests whenever a CFIFO is not full. The FIFO control unit will only transfer the command part of the command message to the selected ADC. Information in the CFIFO header together with the upper bit of the ADC command is used by the FIFO control unit to arbitrate which triggered CFIFO will be transferring the next command. Commands sent to the ADC are executed in a first-in-first-out (FIFO) basis and three types of results can be expected: data read from an ADC register, a conversion result, or a time stamp.

NOTE

While the eQADC pops commands out from a CFIFO, it also is checking the number of entries in the CFIFO and generating requests to fill it. The process of pushing and popping commands to and from a CFIFO can occur simultaneously.

The FIFO control unit expects all incoming results to be shaped in a pre-defined result message format. Figure 31-21 shows how result data flows inside the eQADC system. Results generated on the on-chip ADC are formatted into result messages inside the result format and calibration submodule. Results returning from the external device are already formatted into result messages and therefore bypass the result format and calibration submodule located inside the eQADC. A result message is composed of an RFIFO header and an ADC result. The FIFO control unit decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result should be sent. Once in an RFIFO, the ADC result is moved to the corresponding user result queue by the host CPU or by the eDMA as they respond to interrupt and eDMA requests generated by the eQADC. The eQADC generates these requests whenever an RFIFO has at least one entry.

NOTE

While conversion results are returned, the eQADC is checking the number of entries in the RFIFO and generating requests to empty it. The process of pushing and popping ADC results to and from an RFIFO can occur simultaneously.

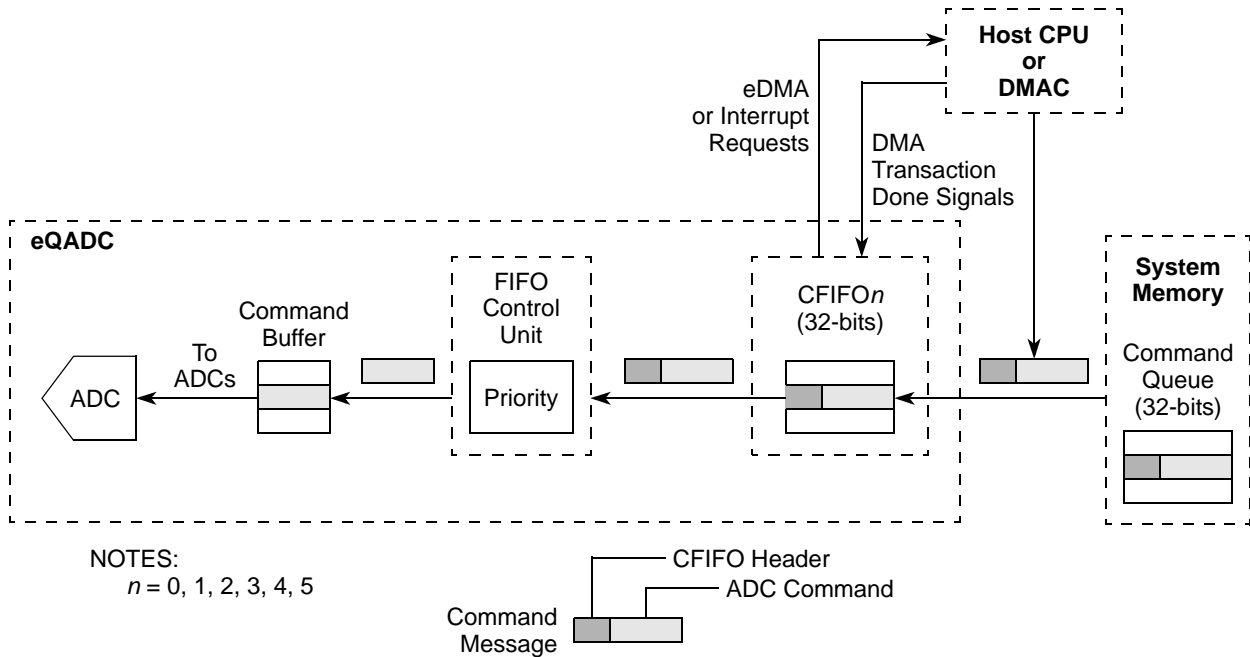


Figure 31-20. Command Flow During eQADC Operation

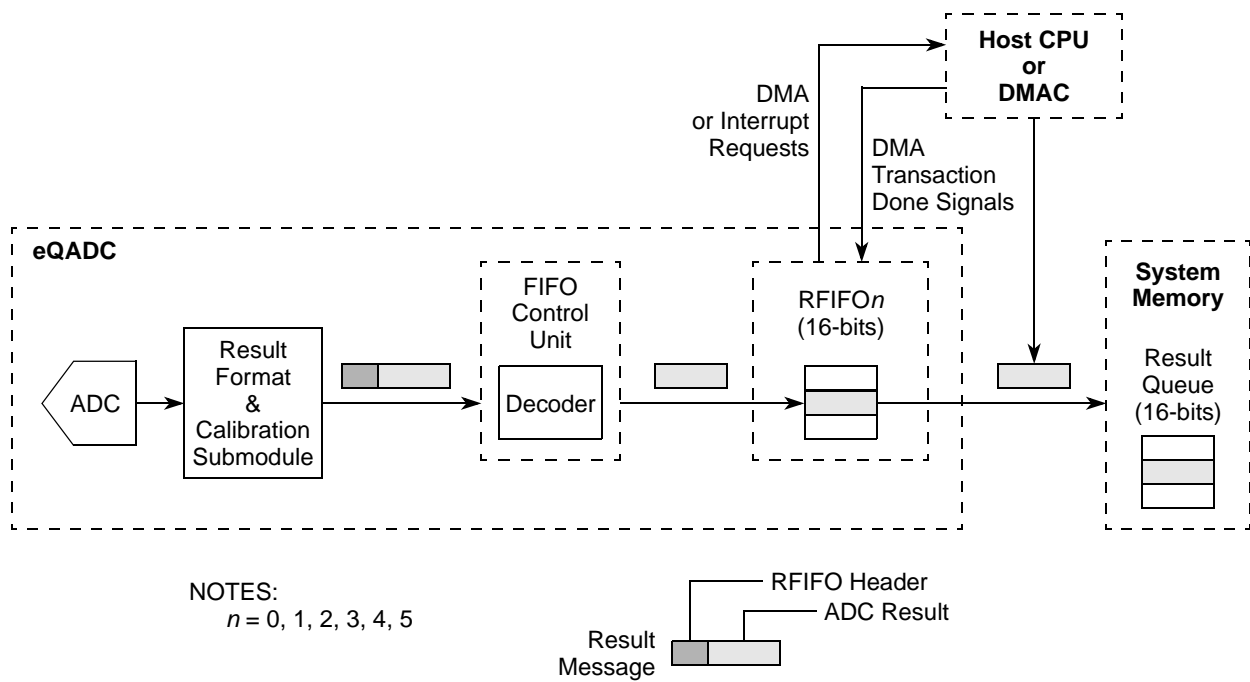


Figure 31-21. Result Flow During eQADC Operation

31.4.1.1 Message Format in eQADC

This section explains the command and result message formats used for on-chip ADC operation

A command message is the pre-defined format at which command data is stored in the user command queues. A command message has 32 bits and is composed of two parts: a CFIFO header and an ADC command. The size of the CFIFO header is fixed to 6 bits, and it works as inputs to the FIFO control unit. The header controls when a command queue ends and when it pauses. Information contained in the CFIFO header, together with the upper bit of the ADC command, is used by the FIFO control unit to arbitrate which triggered CFIFO will transfer the next command. ADC commands are encoded inside the least significant 26 bits of the command message.

A result message is composed of an RFIFO header and an ADC result. The FIFO control unit decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result should be sent. An ADC result is always 16 bits long.

31.4.1.1.1 Message Formats for On-Chip ADC Operation

This section describes the command/result message formats used for on-chip ADC operation.

Conversion Command Message Format for On-Chip ADC Operation

Figure 31-22 describes the command message format for conversion commands when interfacing with the on-chip ADC. A conversion result is always returned for conversion commands and time stamp information can be optionally requested. The lower byte of conversion commands is always set to 0 to distinguish it from configuration commands.

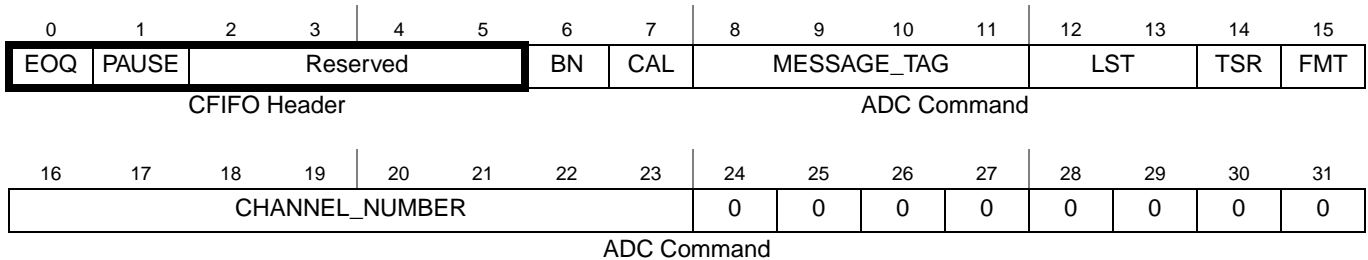


Figure 31-22. Conversion Command Message Format for On-Chip ADC Operation

Table 31-26. On-Chip ADC Field Descriptions: Conversion Command Message Format

Field	Description																								
EOQ	<p>End-of-Queue. Asserted in the last command of a command queue to indicate to the eQADC that a scan of the queue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to 0. Depending on the CFIFO mode of operation, the CFIFO status will also change upon the detection of an asserted EOQ bit on the last transferred command. See Section 31.4.3.5, “CFIFO Scan Trigger Modes,” for details.</p> <p>0 Not the last entry of the command queue. 1 Last entry of the command queue.</p> <p>Note: If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.</p>																								
PAUSE	<p>Pause. Allows software to create sub-queues within a command queue. When the eQADC completes the transfer of a command with an asserted pause bit, the CFIFO enters the WAITING FOR TRIGGER state. Refer to Section 31.4.3.6.1, “CFIFO Operation Status,” for a description of the state transitions. The pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode.</p> <p>0 Do not enter WAITING FOR TRIGGER state after transfer of the current command message. 1 Enter WAITING FOR TRIGGER state after transfer of the current command message.</p> <p>Note: If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.</p>																								
bits 2–5	Reserved.																								
BN	<p>Buffer Number.</p> <p>0 Message sent to ADC 0. 1 Reserved</p>																								
CAL	<p>Calibration. Indicates if the returning conversion result must be calibrated.</p> <p>0 Do not calibrate conversion result. 1 Calibrate conversion result.</p>																								
MESSAGE_TAG	<p>MESSAGE_TAG Field. Allows the eQADC to separate returning results into different RFIFOs. When the eQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the on-chip ADC returns the result with the same MESSAGE_TAG. The eQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data.</p> <table border="1" data-bbox="453 1220 1325 1709"> <thead> <tr> <th>MESSAGE_TAG</th> <th>MESSAGE_TAG Meaning</th> </tr> </thead> <tbody> <tr> <td>0b0000</td> <td>Result is sent to RFIFO 0</td> </tr> <tr> <td>0b0001</td> <td>Result is sent to RFIFO 1</td> </tr> <tr> <td>0b0010</td> <td>Result is sent to RFIFO 2</td> </tr> <tr> <td>0b0011</td> <td>Result is sent to RFIFO 3</td> </tr> <tr> <td>0b0100</td> <td>Result is sent to RFIFO 4</td> </tr> <tr> <td>0b0101</td> <td>Result is sent to RFIFO 5</td> </tr> <tr> <td>0b0110–0b0111</td> <td>Reserved</td> </tr> <tr> <td>0b1000</td> <td>Null message received</td> </tr> <tr> <td>0b1001</td> <td>Reserved for customer use. ¹</td> </tr> <tr> <td>0b1010</td> <td>Reserved for customer use. ¹</td> </tr> <tr> <td>0b1011–0b1111</td> <td>Reserved</td> </tr> </tbody> </table> <p>¹ These messages are treated as null messages. Therefore, they must obey the format for incoming null messages and return valid BUSY0/1 fields.</p>	MESSAGE_TAG	MESSAGE_TAG Meaning	0b0000	Result is sent to RFIFO 0	0b0001	Result is sent to RFIFO 1	0b0010	Result is sent to RFIFO 2	0b0011	Result is sent to RFIFO 3	0b0100	Result is sent to RFIFO 4	0b0101	Result is sent to RFIFO 5	0b0110–0b0111	Reserved	0b1000	Null message received	0b1001	Reserved for customer use. ¹	0b1010	Reserved for customer use. ¹	0b1011–0b1111	Reserved
MESSAGE_TAG	MESSAGE_TAG Meaning																								
0b0000	Result is sent to RFIFO 0																								
0b0001	Result is sent to RFIFO 1																								
0b0010	Result is sent to RFIFO 2																								
0b0011	Result is sent to RFIFO 3																								
0b0100	Result is sent to RFIFO 4																								
0b0101	Result is sent to RFIFO 5																								
0b0110–0b0111	Reserved																								
0b1000	Null message received																								
0b1001	Reserved for customer use. ¹																								
0b1010	Reserved for customer use. ¹																								
0b1011–0b1111	Reserved																								

Table 31-26. On-Chip ADC Field Descriptions: Conversion Command Message Format (continued)

Field	Description										
LST	<p>Long Sampling Time. These two bits determine the duration of the sampling time in ADC clock cycles. Note: For external mux mode, 64 or 128 sampling cycles is recommended.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>LST</th> <th>Sampling cycles (ADC Clock Cycles)</th> </tr> </thead> <tbody> <tr> <td>0b00</td> <td>2</td> </tr> <tr> <td>0b01</td> <td>8</td> </tr> <tr> <td>0b10</td> <td>64</td> </tr> <tr> <td>0b11</td> <td>128</td> </tr> </tbody> </table>	LST	Sampling cycles (ADC Clock Cycles)	0b00	2	0b01	8	0b10	64	0b11	128
LST	Sampling cycles (ADC Clock Cycles)										
0b00	2										
0b01	8										
0b10	64										
0b11	128										
TSR	<p>Time Stamp Request. TSR indicates the request for a time stamp. When TSR is asserted, the on-chip ADC control logic returns a time stamp for the current conversion command after the conversion result is sent to the RFIFOs. See Section 31.4.5.3, “Time Stamp Feature,” for details.</p> <p>0 Return conversion result only. 1 Return conversion time stamp after the conversion result.</p>										
FMT	<p>Conversion Data Format. FMT specifies to the eQADC how to format the 12-bit conversion data returned by the ADC into the 16-bit format which is sent to the RFIFOs. See Section , “ADC Result Format for On-Chip ADC Operation,” for details.</p> <p>0 Right justified unsigned. 1 Right justified signed.</p>										
CHANNEL_NUMBER	<p>Channel Number. Selects the analog input channel. The software programs this field with the channel number corresponding to the analog input pin to be sampled and converted. See Section 31.4.6.1, “Channel Assignment,” for details.</p>										
bits 24–31	Reserved.										

Write Configuration Command Message Format for On-Chip ADC Operation

[Figure 31-23](#) describes the command message format for a write configuration command when interfacing with the on-chip ADC. A write configuration command is used to set the control registers of the on-chip ADC. No conversion data will be returned for a write configuration command. Write configuration commands are differentiated from read configuration commands by a negated R/W bit.

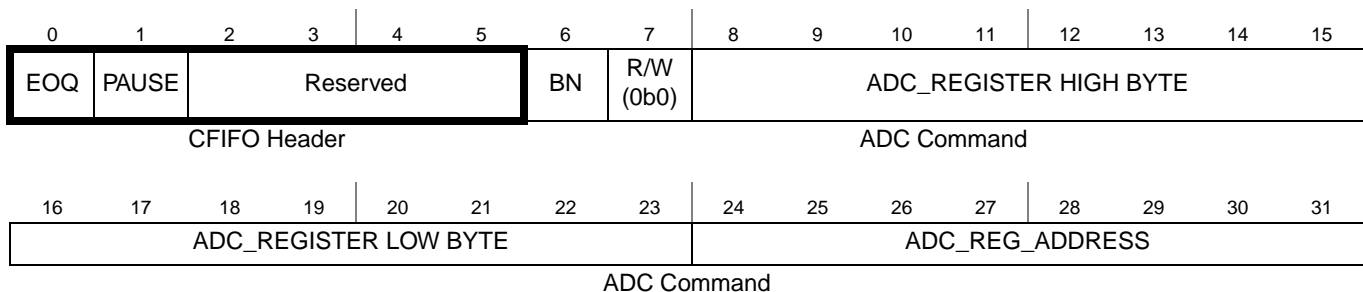
**Figure 31-23. Write Configuration Command Message Format for On-chip ADC Operation**

Table 31-27. On-Chip ADC Field Descriptions: Write Configuration

Field	Description
EOQ	End-of-Queue. Asserted in the last command of a command queue to indicate to the eQADC that a scan of the queue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to 0. Depending on the CFIFO mode of operation, the CFIFO status will also change upon the detection of an asserted EOQ bit on the last transferred command. See Section 31.4.3.5, “CFIFO Scan Trigger Modes,” for details. 0 Not the last entry of the command queue. 1 Last entry of the command queue. Note: If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
1 PAUSE	Pause Bit. Allows software to create sub-queues within a command queue. When the eQADC completes the transfer of a command with an asserted pause bit, the CFIFO enters the WAITING FOR TRIGGER state. Refer to Section 31.4.3.6.1, “CFIFO Operation Status,” for a description of the state transitions. The pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 0 Do not enter WAITING FOR TRIGGER state after transfer of the current command message. 1 Enter WAITING FOR TRIGGER state after transfer of the current command message. Note: If both the pause and EOQ bits are asserted in the same command message, the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
bits 2–5	Reserved.
BN	Buffer Number. Indicates which buffer the message will be stored in. 0 Message stored in buffer 0. 1 Message stored in buffer 1.
R/W	Read/Write. A negated R/W indicates a write configuration command. 0 Write 1 Read
ADC_ REGISTER _HIGH_ BYTE	ADC Register High Byte. The value to be written into the most significant 8 bits of control/configuration register when the R/W bit is negated.
ADC_ REGISTER _LOW_ BYTE	ADC Register Low Byte. The value to be written into the least significant 8 bits of a control/configuration register when the R/W bit is negated.
ADC_REG_ ADDRESS	ADC Register Address. Selects a register on the ADC register set to be written or read. Only halfword addresses can be used. See Table 31-18 .

Read Configuration Command Message Format for On-Chip ADC Operation

[Figure 31-24](#) describes the command message format for a read configuration command when interfacing with the on-chip ADC. A read configuration command is used to read the contents of the on-chip ADC registers which are only accessible via command messages. Read configuration commands are differentiated from write configuration commands by an asserted R/W bit.

Enhanced Queued Analog-to-Digital Converter (eQADC)

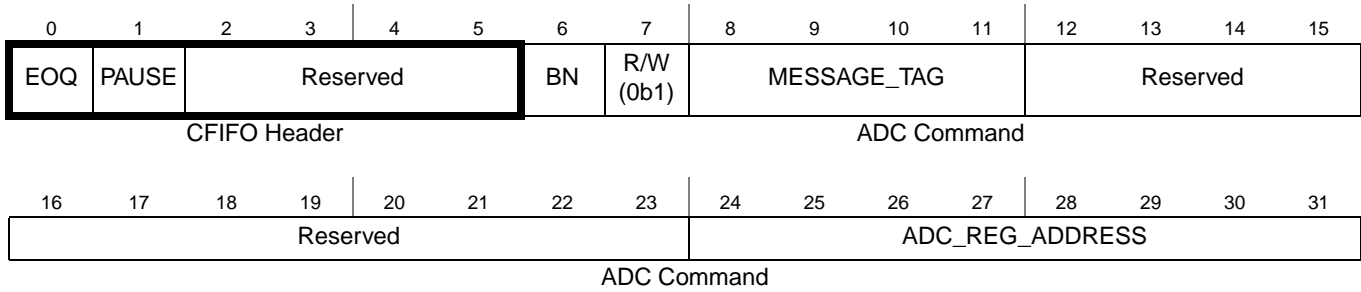


Figure 31-24. Read Configuration Command Message Format for On-Chip ADC Operation

Table 31-28. On-Chip ADC Field Descriptions: Read Configuration

Field	Description
EOQ	<p>End-of-Queue. Asserted in the last command of a command queue to indicate to the eQADC that a scan of the queue is completed. EOQ instructs the eQADC to reset its current CFIFO transfer counter value (TC_CF) to 0. Depending on the CFIFO mode of operation, the CFIFO status will also change upon the detection of an asserted EOQ bit on the last transferred command. See Section 31.4.3.5, “CFIFO Scan Trigger Modes,” for details.</p> <p>0 Not the last entry of the command queue. 1 Last entry of the command queue.</p> <p>Note: If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.</p>
PAUSE	<p>Pause Bit. Allows software to create sub-queues within a command queue. When the eQADC completes the transfer of a command with an asserted pause bit, the CFIFO enters the WAITING FOR TRIGGER state. Refer to Section 31.4.3.6.1, “CFIFO Operation Status,” for a description of the state transitions. The pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode.</p> <p>0 Do not enter WAITING FOR TRIGGER state after transfer of the current command message. 1 Enter WAITING FOR TRIGGER state after transfer of the current command message.</p> <p>Note: If both the pause and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.</p>
bits 2–5	Reserved.
BN	<p>Buffer Number. Indicates which buffer the message will be stored in.</p> <p>0 Message stored in buffer 0. 1 Message stored in buffer 1.</p>
R/W	<p>Read/Write. An asserted R/W bit indicates a read configuration command.</p> <p>0 Write 1 Read</p>

Table 31-28. On-Chip ADC Field Descriptions: Read Configuration (continued)

Field	Description																								
MESSAGE_TAG	<p>MESSAGE_TAG Field. Allows the eQADC to separate returning results into different RFIFOs. When the eQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the on-chip ADC returns the result with the same MESSAGE_TAG. The eQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data.</p> <table border="1"> <thead> <tr> <th>MESSAGE_TAG</th> <th>MESSAGE_TAG Meaning</th> </tr> </thead> <tbody> <tr> <td>0b0000</td> <td>Result is sent to RFIFO 0</td> </tr> <tr> <td>0b0001</td> <td>Result is sent to RFIFO 1</td> </tr> <tr> <td>0b0010</td> <td>Result is sent to RFIFO 2</td> </tr> <tr> <td>0b0011</td> <td>Result is sent to RFIFO 3</td> </tr> <tr> <td>0b0100</td> <td>Result is sent to RFIFO 4</td> </tr> <tr> <td>0b0101</td> <td>Result is sent to RFIFO 5</td> </tr> <tr> <td>0b0110–0b0111</td> <td>Reserved</td> </tr> <tr> <td>0b1000</td> <td>Null message received</td> </tr> <tr> <td>0b1001</td> <td>Reserved for customer use.¹</td> </tr> <tr> <td>0b1010</td> <td>Reserved for customer use.¹</td> </tr> <tr> <td>0b1011–0b1111</td> <td>Reserved</td> </tr> </tbody> </table> <p>¹ These messages are treated as null messages. Therefore, they must obey the format for incoming null messages and return valid BUSY0/1 fields.</p>	MESSAGE_TAG	MESSAGE_TAG Meaning	0b0000	Result is sent to RFIFO 0	0b0001	Result is sent to RFIFO 1	0b0010	Result is sent to RFIFO 2	0b0011	Result is sent to RFIFO 3	0b0100	Result is sent to RFIFO 4	0b0101	Result is sent to RFIFO 5	0b0110–0b0111	Reserved	0b1000	Null message received	0b1001	Reserved for customer use. ¹	0b1010	Reserved for customer use. ¹	0b1011–0b1111	Reserved
MESSAGE_TAG	MESSAGE_TAG Meaning																								
0b0000	Result is sent to RFIFO 0																								
0b0001	Result is sent to RFIFO 1																								
0b0010	Result is sent to RFIFO 2																								
0b0011	Result is sent to RFIFO 3																								
0b0100	Result is sent to RFIFO 4																								
0b0101	Result is sent to RFIFO 5																								
0b0110–0b0111	Reserved																								
0b1000	Null message received																								
0b1001	Reserved for customer use. ¹																								
0b1010	Reserved for customer use. ¹																								
0b1011–0b1111	Reserved																								
bits 12–23	Reserved.																								
ADC_REG_ADDRESS	ADC Register Address. Selects a register on the ADC register set to be written or read. Only halfword addresses can be used.																								

ADC Result Format for On-Chip ADC Operation

When the FIFO control unit receives a return data message, it decodes the MESSAGE_TAG field and stores the 16-bit data into the appropriate RFIFO. This section describes the ADC result portion of the result message returned by the on-chip ADC.

The 16-bit data stored in the RFIFOs can be the following:

- Data read from an ADC register with a read configuration command. In this case, the stored 16-bit data corresponds to the contents of the ADC register that was read.
- A time stamp. In this case, the stored 16-bit data is the value of the time base counter latched when the eQADC detects the end of the analog input voltage sampling. For details see [Section 31.4.5.3, “Time Stamp Feature.”](#)
- A conversion result. In this case, the stored 16-bit data contains a right justified 14-bit result data. The conversion result can be calibrated or not depending on the status of CAL bit in the command that requested the conversion. When the CAL bit is negated, this 14-bit data is obtained by executing a 2-bit left-shift on the 12-bit data received from the ADC. When the CAL bit is asserted, this 14-bit data is the result of the calculations performed in the EQADC MAC unit using the 12-bit

data received from the ADC and the calibration constants GCC and OCC (See [Section 31.4.5.4, “ADC Calibration Feature”](#)). Then, this 14-bit data is further formatted into a 16-bit format according to the status of the FMT bit in the conversion command. When FMT is asserted, the 14-bit result data is reformatted to look as if it was measured against an imaginary ground at $V_{REF}/2$ (the msb (most significant bit) bit of the 14-bit result is inverted), and is sign-extended to a 16-bit format as in [Figure 31-25](#). When FMT is negated, the eQADC zero-extends the 14-bit result data to a 16-bit format as in [Figure 31-26](#). Correspondence between the analog voltage in a channel and the calculated digital values is shown in [Table 31-31](#).

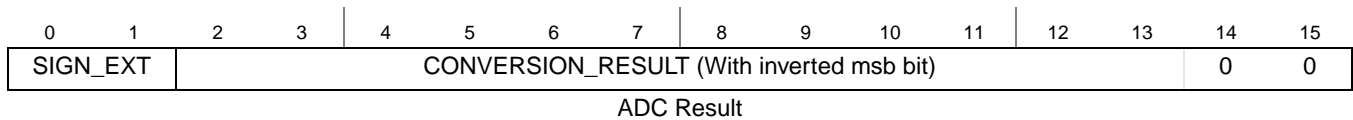


Figure 31-25. ADC Result Format when FMT = 1 (Right Justified Signed)—On-Chip ADC Operation

Table 31-29. ADC Result Format when FMT = 1 Field Descriptions

Field	Description
SIGN_EXT	Sign Extension. Only has meaning when FMT is asserted. SIGN_EXT is 0b00 when CONVERSION_RESULT is positive, and 0b11 when CONVERSION_RESULT is negative.
CONVERSION_RESULT	Conversion Result. A digital value corresponding to the analog input voltage in a channel when the conversion command was initiated. The two's complement representation is used to express negative values.

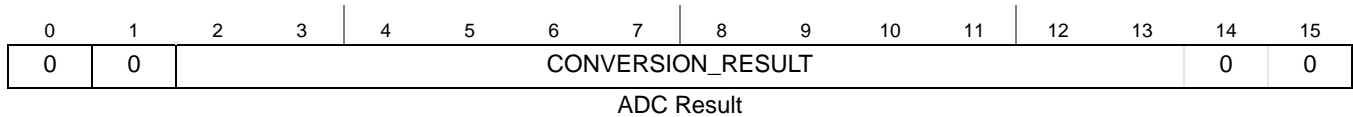


Figure 31-26. ADC Result Format when FMT = 0 (Right Justified Unsigned)—On-Chip ADC Operation

Table 31-30. ADC Result Format when FMT = 0 Field Descriptions

Field	Description
SIGN_EXT	Sign Extension. Only has meaning when FMT is asserted. SIGN_EXT is 0b00 when CONVERSION_RESULT is positive, and 0b11 when CONVERSION_RESULT is negative.
CONVERSION_RESULT	Conversion Result. A digital value corresponding to the analog input voltage in a channel when the conversion command was initiated.

Table 31-31. Correspondence between Analog Voltages and Digital Values^{1, 2}

	Voltage Level on Channel (V)	Corresponding 12-bit Conversion Result Returned by the ADC	16-bit Result Sent to RFIFOs (FMT=0) ³	16-bit Result Sent to RFIFOs (FMT=1) ³
Single-Ended Conversions	5.12	0xFFF	0x3FFC	0x1FFC
	5.12 – 1sb	0xFFF	0x3FFC	0x1FFC

	2.56	0x800	0x2000	0x0000

	1 1sb	0x001	0x0004	0xE004
	0	0x000	0x0000	0xE000

¹ $V_{REF}=V_{RH}-V_{RL}=5.12V$. Resulting in one 12-bit count (1sb) =1.25mV.

² The two's complement representation is used to express negative values.

³ Assuming uncalibrated conversion results.

31.4.2 Command/Result Queues

The command and result queues are actually part of the eQADC system although they are not hardware implemented inside the eQADC. Instead command and result queues are user-defined queues located in system memory. Each command queue entry is a 32-bit command message. The last entry of a command queue has the EOQ bit asserted to indicate that it is the last entry of the queue. The result queue entry is a 16-bit data item.

See [Section 31.1.3, “Modes of Operation,”](#) for a description of the message formats and their flow in eQADC.

Refer to [Section 31.5.5, “Command Queue and Result Queue Usage,”](#) for examples of how command queues and result queues can be used.

31.4.3 eQADC Command FIFOs

31.4.3.1 CFIFO Basic Functionality

There are six prioritized CFIFOs located in the eQADC. Each CFIFO is four entries deep, and each CFIFO entry is 32 bits long. A CFIFO serves as a temporary storage location for the command messages stored in the command queues in system memory. When a CFIFO is not full, the eQADC sets the corresponding CFFF bit in [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\).”](#) If CFFE is asserted as in [Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 \(EQADC_IDCRn\),”](#) the eQADC generates requests for more commands from a command queue. An interrupt request, served by the host CPU, is generated when CFFS is negated, and a eDMA request, served by the eDMA, is generated when CFFS is asserted. The host CPU or the eDMA respond to these requests

by writing to the [Section 31.3.3.4, “eQADC CFIFO Push Registers 0–5 \(EQADC_CFPRn\),”](#) to fill the CFIFO.

NOTE

Only whole words must be written to EQADC_CFPR. Writing halfwords or bytes to EQADC_CFPR will still push the whole 32-bit CF_PUSH field into the corresponding CFIFO, but undefined data will fill the areas of CF_PUSH that were not specifically designated as target locations for writing.

[Figure 31-27](#) describes the important components in the CFIFO. Each CFIFO is implemented as a circular set of registers to avoid the need to move all entries at each push/pop operation. The push next data pointer points to the next available CFIFO location for storing data written into the eQADC command FIFO push register. The transfer next data pointer points to the next entry to be removed from CFIFO_n when it completes a transfer. The CFIFO transfer counter control logic counts the number of entries in the CFIFO and generates eDMA or interrupt requests to fill the CFIFO. TNXTPTR in [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\),”](#) indicates the index of the entry that is currently being addressed by the transfer next data pointer, and CFCTR, in the same register, provides the number of entries stored in the CFIFO.

Using TNXTPTR and CFCTR, the absolute addresses for the entries indicated by the transfer next data pointer and by the push next data pointer can be calculated using the following formulas:

$$\begin{aligned} \text{Transfer Next Data Pointer Address} &= \text{CFIFO}_n\text{_BASE_ADDRESS} + \text{TNXTPTR}_n * 4 \\ \text{Push Next Data Pointer Address} &= \text{CFIFO}_n\text{_BASE_ADDRESS} + \\ &[(\text{TNXTPTR}_n + \text{CFCTR}_n) \bmod \text{CFIFO_DEPTH}] * 4 \end{aligned}$$

where

- $a \bmod b$ returns the remainder of the division of a by b .
- CFIFO_n_BASE_ADDRESS is the smallest memory mapped address allocated to a CFIFO_n entry.
- CFIFO_DEPTH is the number of entries contained in a CFIFO - four in this implementation.

When CFS_n in [Section 31.3.3.11, “eQADC CFIFO Status Register \(EQADC_CFSR\),”](#) is in the TRIGGERED state, the eQADC generates the proper control signals for the transfer of the entry pointed by transfer next data pointer. CFUF_n in [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\),”](#) is set when a CFIFO_n underflow event occurs. A CFIFO underflow occurs when the CFIFO is in the TRIGGERED state and it becomes empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. CFIFO_n is empty when the transfer next data pointer n equals the push next data pointer n and CFCTR_n is 0. CFIFO_n is full when the transfer next data pointer n equals the push next data pointer n and CFCTR_n is not 0.

When the eQADC completes the transfer of an entry from CFIFO_n: the transferred entry is popped from CFIFO_n, the CFIFO counter CFCTR in the [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\),”](#) is decremented by 1, and transfer next data pointer n is incremented by 1 (or wrapped around) to point to the next entry in the CFIFO. The transfer of entries bound for the on-chip ADC is considered completed when they are stored in the appropriate ADC command buffer.

When the EQADC_CFPR n is written and CFIFOn is not full, the CFIFO counter CFCTR n is incremented by 1, and the push next data pointer n then is incremented by 1 (or wrapped around) to point to the next entry in the CFIFO.

When the EQADC_CFPR n is written but CFIFOn is full, the eQADC will not increment the counter value and will not overwrite any entry in CFIFOn.

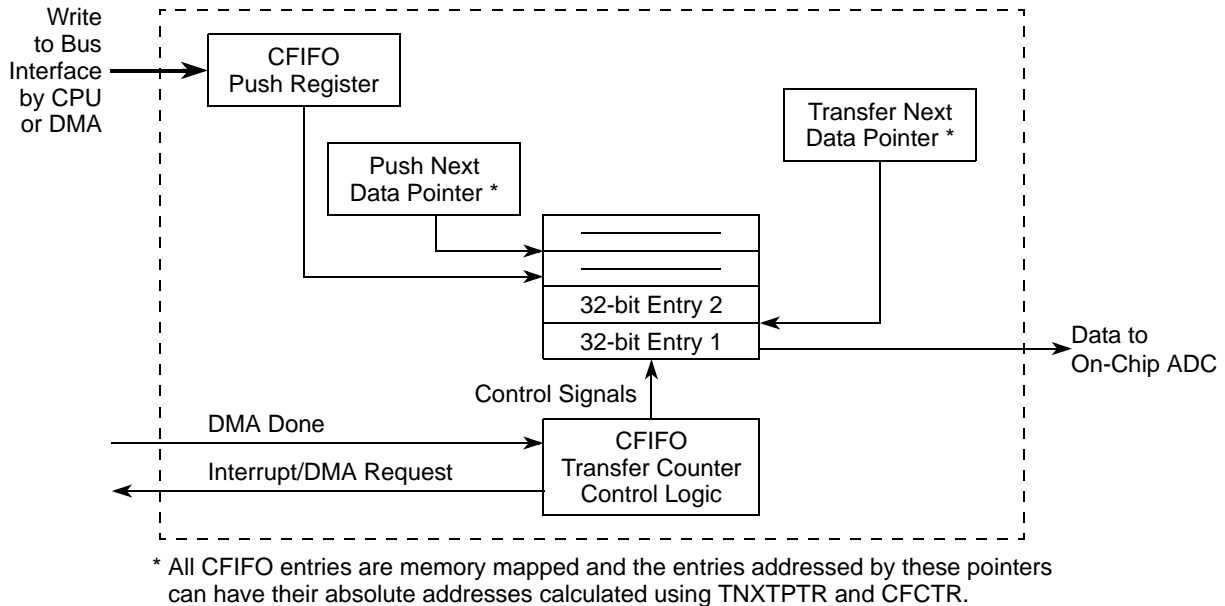


Figure 31-27. CFIFO Diagram

The detailed behavior of the push next data pointer and transfer next data pointer is described in the example shown in [Figure 31-28](#) where a CFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four entries. In this example, CFIFOn with 16 entries is shown in sequence after pushing and transferring entries.

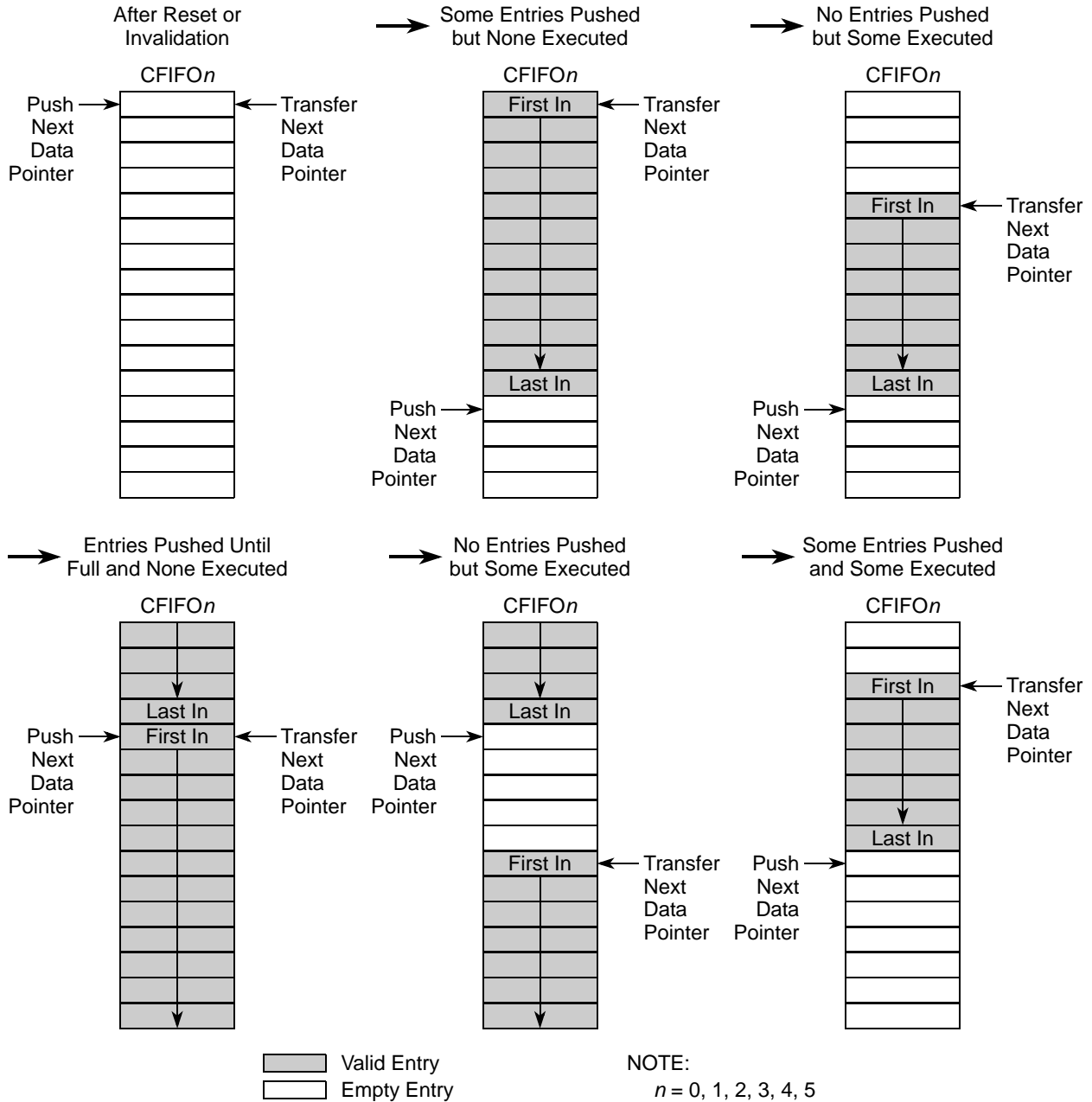


Figure 31-28. CFIFO Entry Pointer Example

31.4.3.2 CFIFO Prioritization and Command Transfer

The CFIFO priority is fixed according to the CFIFO number. A CFIFO with a smaller number has a higher priority. When commands of distinct CFIFOs are bound for the same destination (the same on-chip ADC), the higher priority CFIFO is always served first. A triggered, not-underflowing CFIFO will start the transfer of its commands when the following occur:

- Its commands are bound for an internal command buffer that is not full, and it is the highest priority triggered CFIFO sending commands to that buffer.

A triggered CFIFO with commands bound for a certain command buffer consecutively transfers its commands to the buffer until one of the following occurs:

- An asserted end of queue bit is reached.
- An asserted pause bit is encountered and the CFIFO is configured for edge trigger mode.
- CFIFO is configured for level trigger mode and a closed gate is detected.
- In case its commands are bound for an internal command buffer, a higher priority CFIFO that uses the same internal buffer is triggered.

The prioritization logic of the eQADC, depicted in [Figure 31-29](#), that prioritizes CFIFOs with commands bound for ADC0.

NOTE

Triggered but empty CFIFOs, underflowing CFIFOs, are not considered for prioritization. No data from these CFIFOs will be sent to the on-chip ADC, nor will they stop lower priority CFIFOs from transferring commands.

Whenever CBuffer0 is able to receive new commands, the prioritization submodule selects the highest-priority triggered CFIFO with a command bound for ADC0, and sends it to ADC0. If CBuffer0 is able to receive new entries but there are no triggered CFIFOs with commands bound for it, nothing is sent.

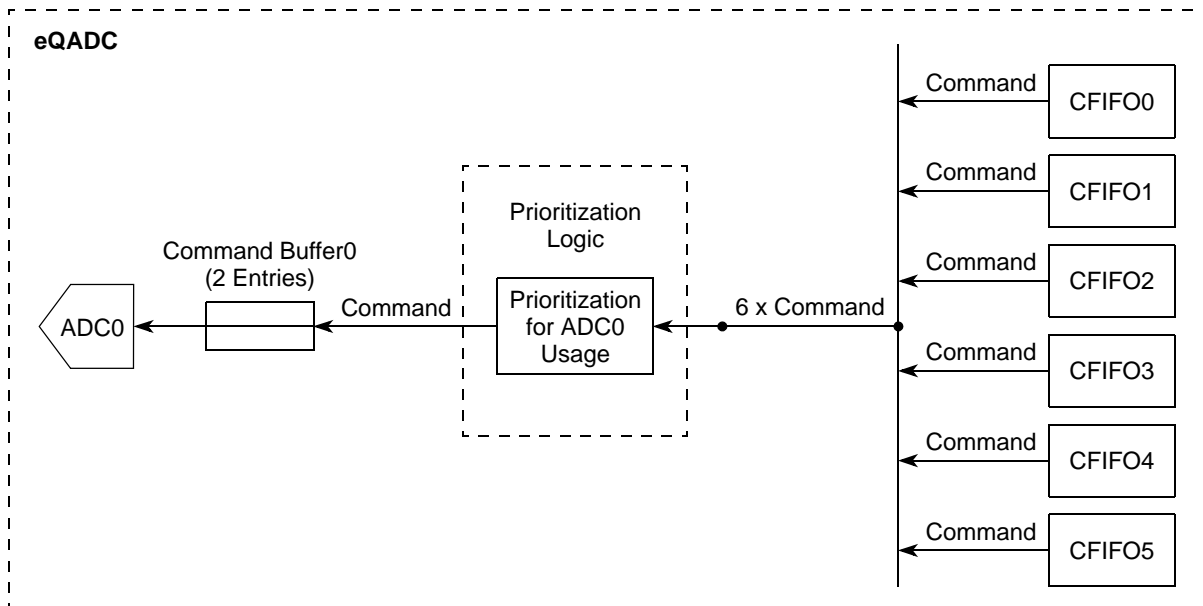


Figure 31-29. CFIFO Prioritization Logic

31.4.3.3 External Trigger from eTPU or eMIOS Channels

The six eQADC external trigger inputs can be connected to either an external pin (either ETRIG0, ETRIG1, GPIO206, or GPIO207), an eTPU channel, or an eMIOS channel. The input source for each eQADC external trigger is individually specified in the eQADC trigger input select register (SIU_ETISR) in the SIU block.

The eQADC trigger numbers specified by SIU_ETISR[TSEL(0-5)] correspond to CFIFO numbers 0-5. To calculate the CFIFO number that each trigger is connected to, divide the eDMA channel number by 2.

31.4.3.4 External Trigger Event Detection

The digital filter length field in Section 31.3.3.3, “eQADC External Trigger Digital Filter Register (EQADC_ETDFR),” specifies the minimum number of system clocks that the external trigger signals 0 and 1 must be held at a logic level to be recognized as valid. All ETRIG signals are filtered. A counter for each queue trigger is implemented to detect a transition between logic levels. The counter counts at the system clock rate. The corresponding counter is cleared and restarted each time the signal transitions between logic levels. When the corresponding counter matches the value specified by the digital filter length field in Section 31.3.3.3, “eQADC External Trigger Digital Filter Register (EQADC_ETDFR),” the eQADC considers the ETRIG logic level to be valid and passes that new logic level to the rest of the eQADC.

The filter is only for filtering the ETRIG signal. Logic after the filter checks for transitions between filtered values, such as for detecting the transition from a filtered logic level zero to a filtered logic level one in rising edge external trigger mode. The eQADC can detect rising edge, falling edge, or level gated external triggers. The digital filter will always be active independently of the status of the MODE_n field in Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 (EQADC_CFCRn),” but the edge, level detection logic is only active when MODE_n is set to a value different from disabled, and in case MODE_n is set to single scan mode, when the SSS bit is asserted. Note that the time necessary for a external trigger event to result into a CFIFO status change is not solely determined by the DFL field in the Section 31.3.3.3, “eQADC External Trigger Digital Filter Register (EQADC_ETDFR).” After being synchronized to the system clock and filtered, a trigger event is checked against the CFIFO trigger mode. Only then, after a valid trigger event is detected, the eQADC accordingly changes the CFIFO status. Refer to Figure 31-30 for an example.

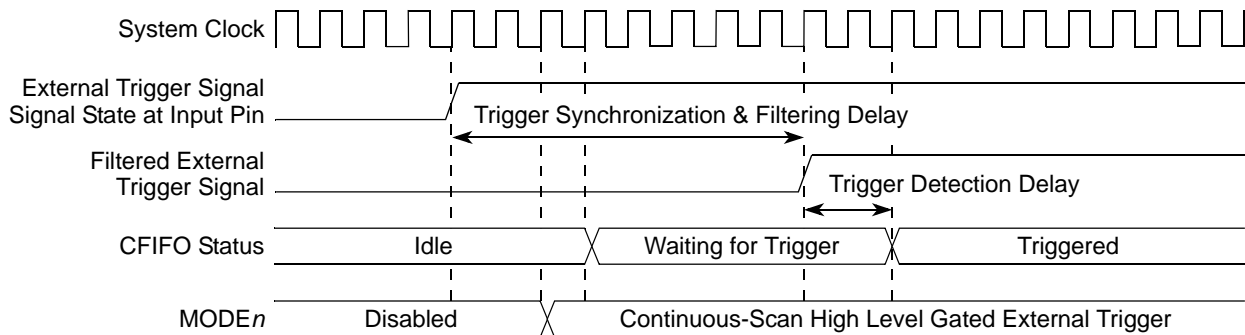


Figure 31-30. ETRIG Event Propagation Example

31.4.3.5 CFIFO Scan Trigger Modes

The eQADC supports two different scan modes, single-scan and continuous-scan. Refer to Table 31-32 for a summary of these two scan modes. When a CFIFO is triggered, the eQADC scan mode determines whether the eQADC will stop command transfers from a CFIFO, and wait for software intervention to rearm the CFIFO to detect new trigger events, upon detection of an asserted EOQ bit in the last transfer. Refer to Section 31.4.1.1, “Message Format in eQADC,” for details about command formats.

CFIFOs can be configured in single-scan or continuous-scan mode. When a CFIFO is configured in single-scan mode, the eQADC scans the command queue one time. The eQADC stops future command transfers from the triggered CFIFO after detecting the EOQ bit set in the last transfer. After a EOQ bit is detected, software involvement is required to rearm the CFIFO so that it can detect new trigger events.

When a CFIFO is configured for continuous-scan mode, no software involvement is necessary to rearm the CFIFO to detect new trigger events after an asserted EOQ is detected. In continuous-scan mode the whole command queue is scanned multiple times.

The eQADC also supports different triggering mechanisms for each scan mode. The eQADC will not transfer commands from a CFIFO until the CFIFO is triggered. The combination of scan modes and triggering mechanisms allows the support of different requirements for scanning input channels. The scan mode and trigger mechanism are configured by programming the $MODE_n$ field in [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\).”](#)

Enabled CFIFOs can be triggered by software or external trigger events. The elapsed time from detecting a trigger to transferring a command is a function of clock frequency, trigger synchronization, trigger filtering, programmable trigger events, command transfer, CFIFO prioritization, ADC availability, etc. Fast and predictable transfers can be achieved by ensuring that the CFIFO is not underflowing and that the target ADC can accept commands when the CFIFO is triggered.

31.4.3.5.1 Disabled Mode

The $MODE_n$ field in [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\).”](#) for all of the CFIFOs can be changed from any other mode to disabled at any time. No trigger event can initiate command transfers from a CFIFO which has its $MODE$ field programmed to disabled.

NOTE

If $MODE_n$ is not disabled, it must not be changed to any other mode besides disabled. If $MODE_n$ is disabled and the CFIFO status is IDLE, $MODE_n$ can be changed to any other mode.

If $MODE_n$ is changed to disabled:

- The CFIFO execution status will change to IDLE. The timing of this change depends on whether a command is being transferred or not:
 - When no command transfer is in progress, the eQADC switches the CFIFO to IDLE status immediately.
 - When a command transfer to an on-chip ADC is in progress, the eQADC will complete the transfer, update TC_CF , and switch CFIFO status to IDLE. Command transfers to the internal ADC are considered completed when a command is written to the relevant buffer.
- The CFIFOs are not invalidated automatically. The CFIFO still can be invalidated by writing a 1 to the $CFINV_n$ bit (see [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\).”](#)). Certify that CFS has changed to IDLE before setting $CFINV_n$.
- The TC_CF_n value also is not reset automatically, but it can be reset by writing 0 to it.
- The $EQADC_FISR_n[SSS]$ bit (see [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\).”](#)) is negated. The SSS bit can be set even if a 1 is written to the

EQADC_CFCR[SSE] bit (see [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\)”](#)) in the same write that the MODE n field is changed to a value other than disabled.

- The trigger detection hardware is reset. If MODE n is changed from disabled to an edge trigger mode, a new edge, matching that edge trigger mode, is needed to trigger the command transfers from the CFIFO.

NOTE

CFIFO fill requests, which generated when CFFF is asserted, are not automatically halted when MODE n is changed to disabled. CFIFO fill requests will still be generated until EQADC_IDCR n [CFFE] bit is cleared (see [Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 \(EQADC_IDCRn\)”](#)).

31.4.3.5.2 Single-Scan Mode

In single-scan mode, a single pass through a sequence of command messages in the user-defined command queue is performed.

In single-scan software trigger mode, the CFIFO is triggered by an asserted single-scan status bit, EQADC_FISR n [SSS] (see [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\)”](#)). The SSS bit is set by writing 1 to the single-scan enable bit, EQADC_CFCR n [SSE] (see [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\)”](#)).

In single-scan edge- or level-trigger mode, the respective triggers are only detected when the SSS bit is asserted. When the SSS bit is negated, all trigger events for that CFIFO are ignored. Writing a 1 to the SSE bit can be done during the same write cycle that the CFIFO operation mode is configured.

Only the eQADC can clear the SSS bit. Once SSS is asserted, it remains asserted until the eQADC completes the command queue scan, or the CFIFO operation mode, EQADC_CFCR n [MODE n] (see [Section 31.3.3.6](#)) is changed to disabled. The SSS n bit will be negated while MODE n is disabled.

Single-Scan Software Trigger

When single-scan software trigger mode is selected, the CFIFO is triggered by an asserted SSS bit. The SSS bit is asserted by writing 1 to the SSE bit. Writing to SSE while SSS is already asserted will not have any effect on the state of the SSS bit, nor will it cause a trigger overrun event.

The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC. When an asserted EOQ bit is encountered, the eQADC will clear the SSS bit. Setting the SSS bit is required for the eQADC to start the next scan of the queue.

The pause bit has no effect in single-scan software trigger mode.

Single-Scan Edge Trigger

When SSS is asserted and an edge triggered mode is selected for a CFIFO, an appropriate edge on the associated trigger signal causes the CFIFO to become triggered. For example, if rising-edge trigger mode is selected, the CFIFO becomes triggered when a rising edge is sensed on the trigger signal. The CFIFO

commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC.

When an asserted EOQ bit is encountered, the eQADC clears SSS and stops command transfers from the CFIFO. An asserted SSS bit and a subsequent edge trigger event are required to start the next scan for the CFIFO. When an asserted pause bit is encountered, the eQADC stops command transfers from the CFIFO, but SSS remains set. Another edge trigger event is required for command transfers to continue. A trigger overrun happens when the CFIFO is in a TRIGGERED state and an edge trigger event is detected.

Single-Scan Level Trigger

When SSS is asserted and a level gated trigger mode is selected, the input level on the associated trigger signal puts the CFIFO in a TRIGGERED state. When the CFIFO is set to high-level gated trigger mode, a high level signal opens the gate, and a low level closes the gate. When the CFIFO is set to low-level gated trigger mode, a low level signal opens the gate, and a high level closes the gate. If the corresponding level is already present, setting the SSS bit triggers the CFIFO. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC.

The eQADC clears the SSS bit and stops transferring commands from a triggered CFIFO when an asserted EOQ bit is encountered or when CFIFO status changes from triggered due to the detection of a closed gate. If a closed gate is detected while no command transfers are taking place and the CFIFO status is triggered, the CFIFO status is immediately changed to IDLE, the SSS bit is negated, and the PF flag is asserted. An asserted SSS bit and a level trigger are required to restart the CFIFO. Command transfers will restart from the point they have stopped.

The pause bit has no effect in single-scan level-trigger mode.

31.4.3.5.3 Continuous-Scan Mode

In continuous-scan mode, multiple passes looping through a sequence of command messages in a command queue are executed. When a CFIFO is programmed for a continuous-scan mode, the EQADC_CFCRn[SSE] (see [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\)”](#)) does not have any effect.

Continuous-Scan Software Trigger

When a CFIFO is programmed to continuous-scan software trigger mode, the CFIFO is triggered immediately. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC. When a CFIFO is programmed to run in continuous-scan software trigger mode, the eQADC will not halt transfers from the CFIFO until the CFIFO operation mode is modified to disabled or a higher priority CFIFO preempts it. Although command transfers will not stop upon detection of an asserted EOQ bit, the EOQF is set and, if enabled, an EOQ interrupt request is generated.

The pause bit has no effect in continuous-scan software trigger mode.

Continuous-Scan Edge Trigger

When rising, falling, or either edge trigger mode is selected for a CFIFO, a corresponding edge on the associated ETRIG signal places the CFIFO in a TRIGGERED state. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC.

When an EOQ or a pause is encountered, the eQADC halts command transfers from the CFIFO and, if enabled, the appropriate interrupt requests are generated. Another edge trigger event is required to resume command transfers but no software involvement is required to rearm the CFIFO in order to detect such event.

A trigger overrun happens when the CFIFO is already in a TRIGGERED state and a new edge trigger event is detected.

Continuous-Scan Level Trigger

When high or low level gated trigger mode is selected, the input level on the associated trigger signal places the CFIFO in a TRIGGERED state. When high-level gated trigger is selected, a high-level signal opens the gate, and a low level closes the gate. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using an available on-chip ADC. Although command transfers will not stop upon detection of an asserted EOQ bit at the end of a command transfer, the EOQF is asserted and, if enabled, an EOQ interrupt request is generated.

The eQADC stops transferring commands from a triggered CFIFO when CFIFO status changes from triggered due to the detection of a closed gate. If a closed gate is detected while no command transfers are taking place and the CFIFO status is TRIGGERED, the CFIFO status is immediately changed to waiting for trigger and the PF flag is asserted. Command transfers will restart as the gate opens.

The pause bit has no effect in continuous-scan level-trigger mode.

31.4.3.5.4 CFIFO Scan Trigger Mode Start/Stop Summary

Table 31-32 summarizes the start and stop conditions of command transfers from CFIFOs for all of the single-scan and continuous-scan trigger modes.

Table 31-32. CFIFO Scan Trigger Mode—Command Transfer Start/Stop Summary

Trigger Mode	Requires Asserted SSS to Recognize Trigger Events?	Command Transfer Start/Restart Condition	Stop on asserted EOQ bit ¹ ?	Stop on asserted Pause bit ² ?	Other Command Transfer Stop Condition ^{3 4}
Single Scan Software	Not Applicable	Asserted SSS bit.	Yes	No	None.
Single Scan Edge	Yes	A corresponding edge occurs when the SSS bit is asserted.	Yes	Yes	None.

Table 31-32. CFIFO Scan Trigger Mode—Command Transfer Start/Stop Summary (continued)

Trigger Mode	Requires Asserted SSS to Recognize Trigger Events?	Command Transfer Start/Restart Condition	Stop on asserted EOQ bit ¹ ?	Stop on asserted Pause bit ² ?	Other Command Transfer Stop Condition ^{3 4}
Single Scan Level	Yes	Gate is opened when the SSS bit is asserted.	Yes	No	The eQADC also stops transfers from the CFIFO when CFIFO status changes from triggered due to the detection of a closed gate. ⁵
Continuous Scan Software	No	CFIFO starts automatically after being configured into this mode.	No	No	None.
Continuous Scan Edge	No	A corresponding edge occurs.	Yes	Yes	None.
Continuous Scan Level	No	Gate is opened.	No	No	The eQADC also stops transfers from the CFIFO when CFIFO status changes from triggered due to the detection of a closed gate. ⁵

¹ Refer to [Section 31.4.3.6.2, “Command Queue Completion Status,”](#) for more information on EOQ.

² Refer to [Section 31.4.3.6.3, “Pause Status,”](#) for more information on pause.

³ The eQADC always stops command transfers from a CFIFO when the CFIFO operation mode is disabled.

⁴ The eQADC always stops command transfers from a CFIFO when a higher priority CFIFO is triggered. Refer to [Section 31.4.3.2, “CFIFO Prioritization and Command Transfer,”](#) for information on CFIFO priority.

⁵ If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status.

31.4.3.6 CFIFO and Trigger Status

31.4.3.6.1 CFIFO Operation Status

Each CFIFO has its own CFIFO status field. CFIFO status (CFS) can be read from EQADC_CFSSR (see [Section 31.3.3.11, “eQADC CFIFO Status Register \(EQADC_CFSR\).”](#) [Figure 31-31](#) and [Table 31-33](#) indicate the CFIFO status switching condition. Refer to [Table 31-15](#) for the meaning of each CFIFO operation status. The last CFIFO to transfer a command to an on-chip ADC can be read from the LCFT_n ($n=0,1$) fields (see [Section 31.3.3.10, “eQADC CFIFO Status Snapshot Register \(EQADC_CFSSR\).”](#))

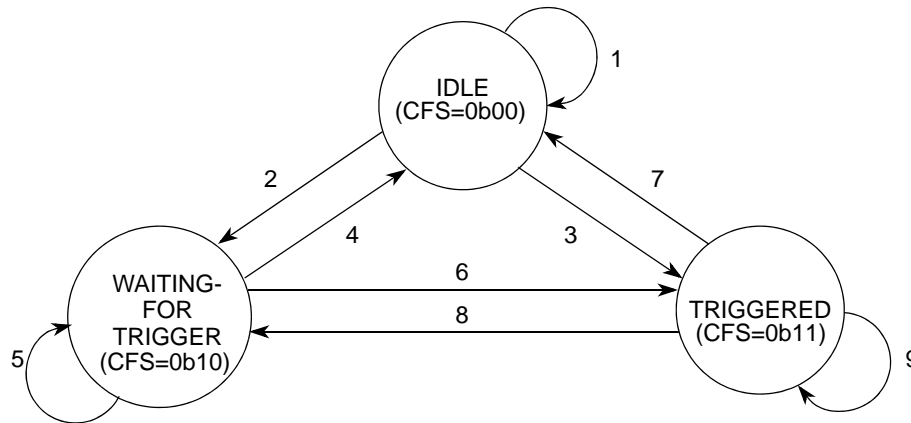


Figure 31-31. State Machine of CFIFO Status

Table 31-33. Command FIFO Status Switching Condition

No.	From Current CFIFO Status (CFS)	To New CFIFO Status (CFS)	Status Switching Condition
1	IDLE (00)	IDLE (0b00)	<ul style="list-style-type: none"> CFIFO mode is programmed to disabled, OR CFIFO mode is programmed to single-scan edge or level trigger mode and SSS is negated.
2		WAITING FOR TRIGGER (0b10)	<ul style="list-style-type: none"> CFIFO mode is programmed to continuous-scan edge or level trigger mode, OR CFIFO mode is programmed to single-scan edge or level trigger mode and SSS is asserted, OR CFIFO mode is programmed to single-scan software trigger mode.
3		TRIGGERED (0b11)	<ul style="list-style-type: none"> CFIFO mode is programmed to continuous-scan software trigger mode
4	WAITING FOR TRIGGER (10)	IDLE (0b00)	<ul style="list-style-type: none"> CFIFO mode is modified to disabled mode.
5		WAITING FOR TRIGGER (0b10)	<ul style="list-style-type: none"> No trigger occurred.
6		TRIGGERED (0b11)	<ul style="list-style-type: none"> Appropriate edge or level trigger occurred, OR CFIFO mode is programmed to single-scan software trigger mode and SSS bit is asserted.

Table 31-33. Command FIFO Status Switching Condition (continued)

No.	From Current CFIFO Status (CFS)	To New CFIFO Status (CFS)	Status Switching Condition
7	TRIGGERED (11)	IDLE (0b00)	<ul style="list-style-type: none"> CFIFO in single-scan mode, eQADC detects the EOQ bit asserted at end of command transfer, and CFIFO mode is not modified to disabled, OR CFIFO, in single-scan level trigger mode, and the gate closes while no commands are being transferred from the CFIFO, and CFIFO mode is not modified to disabled, OR CFIFO, in single-scan level trigger mode, and eQADC detects a closed gated at end of command transfer, and CFIFO mode is not modified to disabled, OR CFIFO mode is modified to disabled mode and CFIFO was not transferring commands. CFIFO mode is modified to disabled mode while CFIFO was transferring commands, and CFIFO completes or aborts the transfer.
8		WAITING FOR TRIGGER (0b10)	<ul style="list-style-type: none"> CFIFO in single or continuous-scan edge trigger mode, eQADC detects the pause bit asserted at the end of command transfer, the EOQ bit in the same command is negated, and CFIFO mode is not modified to disabled, OR CFIFO in continuous-scan edge trigger mode, eQADC detects the EOQ bit asserted at the end of command transfer, and CFIFO mode is not modified to disabled, OR CFIFO, in continuous-scan level trigger mode, and the gate closes while no commands are being transferred from the CFIFO, and CFIFO mode is not modified to disabled, OR CFIFO, in continuous-scan level trigger mode, and eQADC detects a closed gated at end of command transfer, and CFIFO mode is not modified to disabled.
9		TRIGGERED (0b11)	<ul style="list-style-type: none"> No event to switch to IDLE or WAITING FOR TRIGGER status has happened.

31.4.3.6.2 Command Queue Completion Status

The end of queue flag, EQADC_FISRn[EOQF] (see [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\)”](#)) is asserted when the eQADC completes the transfer of a CFIFO entry with an asserted EOQ bit. Software sets the EOQ bit in the last command message of a user-defined command queue to indicate that this entry is the end of the queue. See [Section 31.4.1.1, “Message Format in eQADC,”](#) for information on command message formats. The transfer of entries bound for the on-chip ADC is considered completed when they are stored in the appropriate command buffer.

The command with a EOQ bit asserted is valid and will be transferred. When EQADC_CFCRn[EOQIE] (see [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\)”](#)) and EQADC_FISRn[EOQF] are asserted, the eQADC will generate an end of queue interrupt request.

In single-scan modes, command transfers from the corresponding CFIFO will cease when the eQADC completes the transfer of a entry with an asserted EOQ. Software involvement is required to rearm the CFIFO so that it can detect new trigger events.

NOTE

An asserted $EOQFn$ only implies that the eQADC has finished transferring a command with an asserted EOQ bit from $CFIFOn$. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

31.4.3.6.3 Pause Status

In edge trigger mode, when the eQADC completes the transfer of a CFIFO entry with an asserted pause bit, the eQADC will stop future command transfers from the CFIFO and set $EQADC_FISRn[PF]$ (see [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\)”](#)). Refer to [Section 31.4.1.1, “Message Format in eQADC,”](#) for information on command message formats. The eQADC ignores the pause bit in command messages in any software level trigger mode. The eQADC sets the PF flag upon detection of an asserted pause bit only in single or continuous-scan edge trigger mode. When the PF flag is set for a CFIFO in single-scan edge trigger mode, the $EQADC_FISRn[SSS]$ bit will not be cleared (see [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\)”](#)).

In level trigger mode, the definition of the PF flag has been redefined. In level trigger mode, when $CFIFOn$ is in TRIGGERED status, PFn is set when the CFIFO status changes from TRIGGERED due to detection of a closed gate. The pause flag interrupt routine can be used to verify if the a complete scan of the command queue was performed. If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status.

When $EQADC_CFCR[PIE]$ (see [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\)”](#)) and $EQADC_FISRn[PF]$ are asserted, the eQADC will generate a pause interrupt request.

NOTE

In edge trigger mode, an asserted PFn only implies that the eQADC finished transferring a command with an asserted pause bit from $CFIFOn$. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

NOTE

In software or level trigger mode, when the eQADC completes the transfer of an entry from $CFIFOn$ with an asserted pause bit, PFn will not be set and command transfers will continue without pausing.

31.4.3.6.4 Trigger Overrun Status

When a CFIFO is configured for edge- or level-trigger mode and is in a TRIGGERED state, an additional trigger occurring for the same CFIFO results in a trigger overrun. The trigger overrun bit for the corresponding CFIFO will be set ($EQADC_FISRn[TORFn] = 1$, see [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\)”](#)). When $EQADC_CFCRn[TORIE]$ (see [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\)”](#)) and $EQADC_FISRn[TORF]$ are asserted, the eQADC generates a trigger overrun interrupt request.

31.4.3.6.5 Command Sequence Non-Coherency Detection

The eQADC provides a mechanism to indicate if a command sequence has been completely executed without interruptions. A command sequence is defined as a group of consecutive commands bound for the same ADC and it is expected to be executed without interruptions. A command sequence is coherent if its commands are executed in order without interruptions. Because commands are stored in the ADC's command buffers before being executed in the eQADC, a command sequence is coherent if, while it is transferring commands to an on-chip ADC command buffer, the buffer is only fed with commands from that sequence without ever becoming empty.

A command sequence starts when:

- A CFIFO in TRIGGERED state transfers its first command to an on-chip ADC.
- The CFIFO is constantly transferring commands and the previous command sequence ended.
- The CFIFO resumes command transfers after being interrupted.

And a command sequence ended when:

- An asserted EOQ bit is detected on the last transferred command.
- CFIFO is in edge-trigger mode and asserted pause bit is detected on the last transferred command.

Figure 31-32 shows examples of how the eQADC would detect command sequences when transferring commands from a CFIFO.

User Command Queue with Two Command Sequences	
1	CF5_ADC0_CM0
2	CF5_ADC0_CM1
3	CF5_ADC0_CM2
4	CF5_ADC0_CM3(Pause=1)
5	CF5_ADC0_CM4
6	CF5_ADC0_CM5
7	CF5_ADC0_CM6(EOQ=1)

Assuming that these commands are transferred by a CFIFO configured for edge trigger mode and the command transfers are never interrupted, the eQADC would check for non-coherency of two command sequences: one formed by commands 0, 1, 2, 3, and the other by commands 4, 5, 6.

Figure 31-32. Command Sequence Example

The NCF flag is used to indicate command sequence non-coherency. When the NCF_n flag is asserted, it indicates that the command sequence being transferred through CFIFO $_n$ became non-coherent. The NCF flag only becomes asserted for CFIFOs in a TRIGGERED state.

A command sequence is non-coherent when, after transferring the first command of a sequence from a CFIFO to a buffer, it cannot successively send all the other commands of the sequence before any of the following conditions are true:

- The CFIFO through which commands are being transferred is pre-empted by a higher priority CFIFO which sends commands to the same CBuffer. The NCF flag becomes asserted immediately after the first command transfer from the pre-empting CFIFO, that is the higher priority CFIFO, to the ADC in use is completed. See Figure 31-43.

31.4.4 Result FIFOs

31.4.4.1 RFIFO Basic Functionality

There are six RFIFOs located in the eQADC. Each RFIFO is four entries deep, and each RFIFO entry is 16 bits long. Each RFIFO serves as a temporary storage location for the one of the result queues allocated in system memory. All result data is saved in the RFIFOs before being moved into the system result queues. When an RFIFO is not empty, the eQADC sets the corresponding EQADC_FISR_n[RFDF] (see Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR_n)”). If EQADC_IDCR_n[RFDE] is asserted (see Section 31.3.3.7), the eQADC generates a request so that the RFIFO entry is moved to a result queue. An interrupt request, served by the host CPU, is generated when EQADC_IDCR_n[RFDS] is negated, and an eDMA request, served by the eDMA, is generated when RFDS is asserted. The host CPU or the eDMA responds to these requests by reading EQADC_RFPR_n (see Section 31.3.3.5, “eQADC Result FIFO Pop Registers 0–5 (EQADC_RFPR_n)”) to retrieve data from the RFIFO.

NOTE

Reading a word, halfword, or any bytes from EQADC_RFPR_n will pop an entry from RFIFO_n, and the RFCTR_n field will be decremented by 1.

The eDMA controller should be configured to read a single result (16-bit data) from the RFIFO pop registers for every asserted eDMA request it acknowledges. Refer to Section 31.5.2, “eQADC/eDMA Controller Interface” for eDMA controller configuration guidelines.

Figure 31-33 describes the important components in the RFIFO. Each RFIFO is implemented as a circular set of registers to avoid the need to move all entries at each push/pop operation. The pop next data pointer always points to the next RFIFO message to be retrieved from the RFIFO when reading EQADC_RFPR. The receive next data pointer points to the next available RFIFO location for storing the next incoming message from the on-chip ADC. The RFIFO counter logic counts the number of entries in RFIFO and generates interrupt or eDMA requests to drain the RFIFO.

EQADC_FISR_n[POPNEXTPTR] (see Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR_n)”) indicates which entry is currently being addressed by the pop next data pointer, and EQADC_FISR_n[RFCTR] provides the number of entries stored in the RFIFO. Using POPNEXTPTR and RFCTR, the absolute addresses for pop next data pointer and receive next data pointer can be calculated using the following formulas:

$$\begin{aligned} \text{Pop Next Data Pointer Address} &= \text{RFIFO}_n\text{_BASE_ADDRESS} + \text{POPNEXTPTR}_n * 4 \\ \text{Receive Next Data Pointer Address} &= \text{RFIFO}_n\text{_BASE_ADDRESS} + \\ &[(\text{POPNEXTPTR}_n + \text{RFCTR}_n) \bmod \text{RFIFO_DEPTH}] * 4 \end{aligned}$$

where

- $a \bmod b$ returns the remainder of the division of a by b .
- RFIFO_n_BASE_ADDRESS is the smallest memory mapped address allocated to an RFIFO_n entry.
- RFIFO_DEPTH is the number of entries contained in a RFIFO - four in this implementation.

When a new message arrives and RFIFO_n is not full, the eQADC copies its contents into the entry pointed by receive next data pointer. The RFIFO counter $\text{EQADC_FISR}_n[\text{RFCTR}_n]$ (see Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR $_n$)”) is incremented by 1, and the receive next data pointer n is also incremented by 1 (or wrapped around) to point to the next empty entry in RFIFO_n . However, if the RFIFO_n is full, the eQADC sets the $\text{EQADC_FISR}_n[\text{RFOF}]$ (see Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISR $_n$)”). The RFIFO_n will not overwrite the older data in the RFIFO , the new data will be ignored, and the receive next data pointer n is not incremented or wrapped around. RFIFO_n is full when the receive next data pointer n equals the pop next data pointer n and RFCTR_n is not 0. RFIFO_n is empty when the receive next data pointer n equals the pop next data pointer n and RFCTR_n is 0.

When the eQADC RFIFO pop register n is read and the RFIFO_n is not empty, the RFIFO counter RFCTR_n is decremented by 1, and the pop next data pointer is incremented by 1 (or wrapped around) to point to the next RFIFO entry.

When the eQADC RFIFO pop register n is read and RFIFO_n is empty, eQADC will not decrement the counter value and the pop next data pointer n will not be updated. The read value will be undefined.

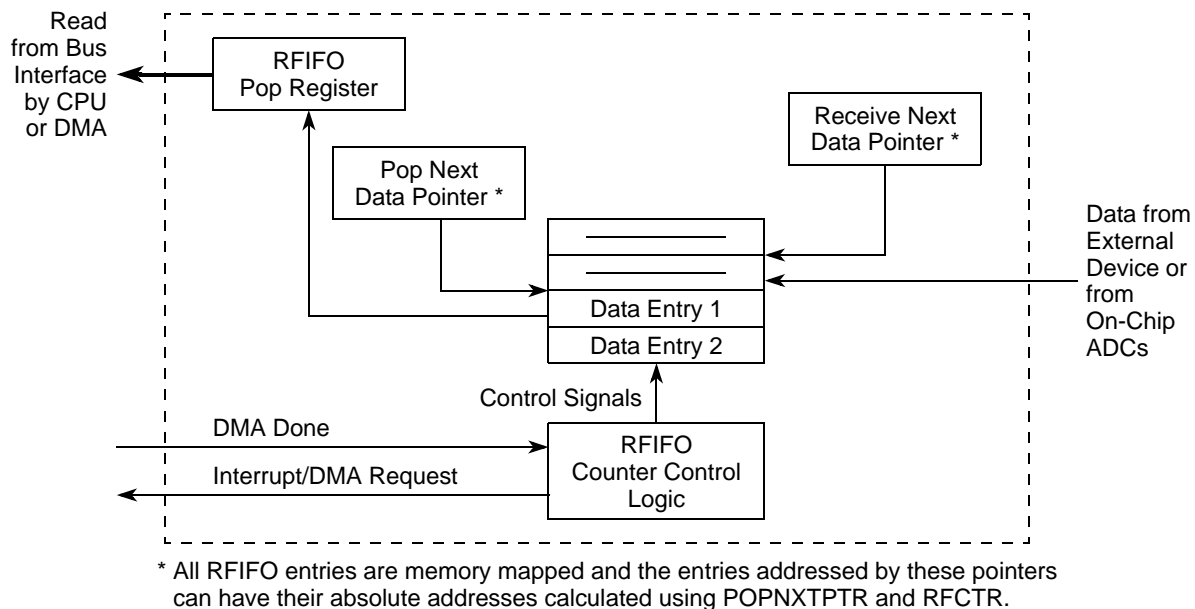


Figure 31-33. RFIFO Diagram

The detailed behavior of the pop next data pointer and receive next data pointer is described in the example shown in Figure 31-34 where an RFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four entries. In this example, RFIFO_n with 16 entries is shown in sequence after popping or receiving entries.

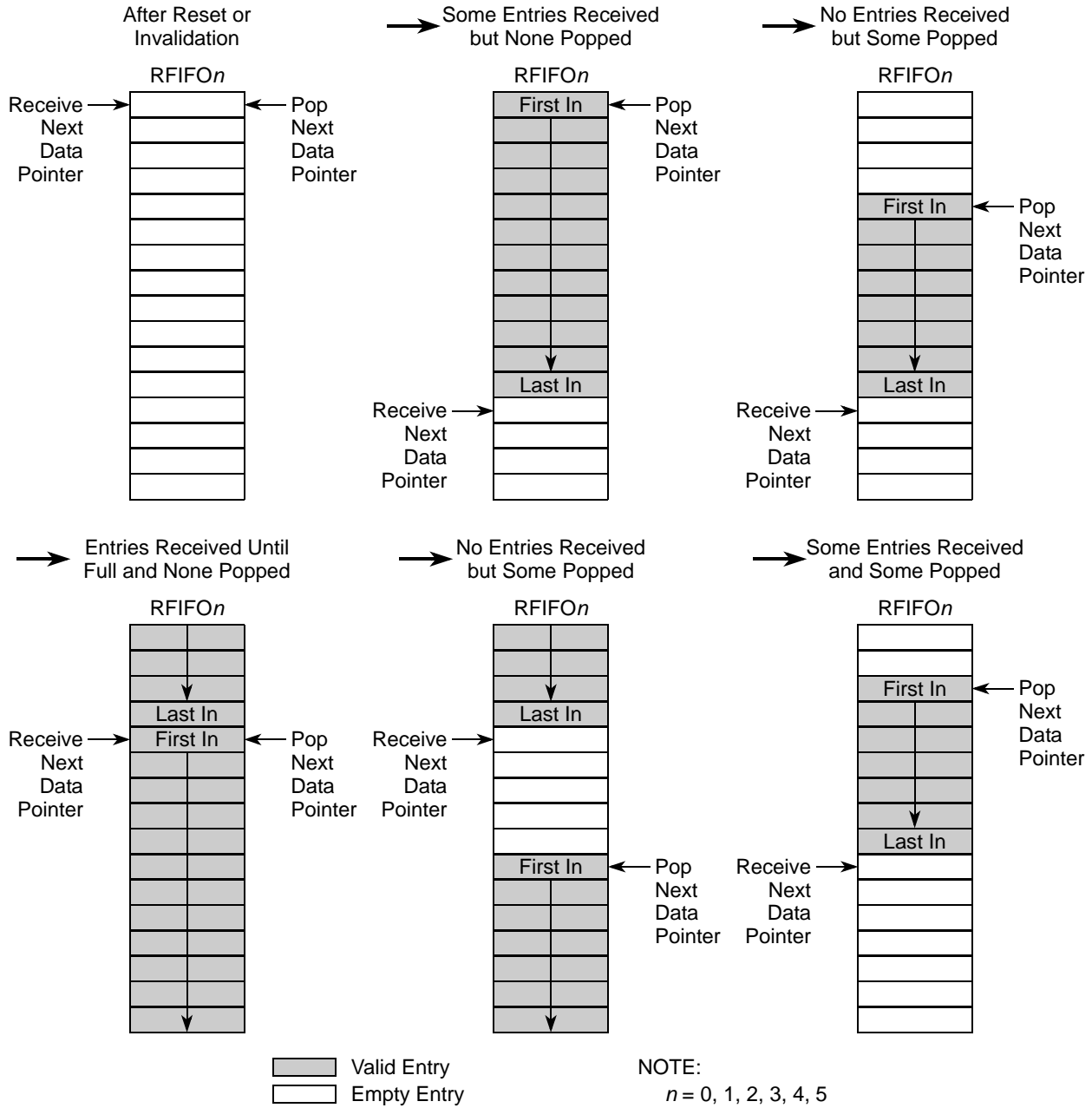


Figure 31-34. RFIFO Entry Pointer Example

31.4.4.2 Distributing Result Data into RFIFOs

Data to be moved into the RFIFOs comes from ADC0. All result data comes with a MESSAGE_TAG field defining what should be done with the received data. The FIFO control unit decodes the MESSAGE_TAG field and:

- Stores the 16-bit data into the appropriate RFIFO if the MESSAGE_TAG indicates a valid RFIFO number or
- Ignores the data in case of a null or “reserved for customer use” MESSAGE_TAG

In general, received data is moved into RFIFOs as they become available.

When time-stamped results return from the on-chip ADC, the conversion result and the time stamp are always moved to the RFIFOs in consecutive clock cycles in order to guarantee they are always stored in consecutive RFIFO entries.

31.4.5 On-Chip ADC Configuration and Control

31.4.5.1 Enabling and Disabling the on-chip ADC

The on-chip ADC has an enable bit (ADC0_CR[ADC0_EN], see [Section 31.3.4.1, “ADC0 Control Register \(ADC0_CR\)”](#)) which allows the enabling of the ADC only when necessary. When the enable bit for an ADC is negated, the clock input to that ADC is stopped. The ADC is disabled out of reset - ADC0_EN negated - to allow for their safe configuration. The ADC must only be configured when its enable bit is negated. Once the enable bit of the ADC is asserted, clock input is started, and the bias generator circuit is turned on. When the enable bits of the ADC is negated, the bias circuit generator is stopped.

NOTE

Conversion commands sent to a disabled ADC are ignored by the ADC control hardware.

NOTE

An 8ms wait time from VDDA power up to enabling ADC or exiting from stop or sleep mode is required to pre-charge the external 100nf capacitor on REFBYPC. This time must be guaranteed by crystal startup time plus reset duration or the user. The ADC internal bias generator circuit will start up after 10us upon VRH/VRL power up and produces a stable/required bias current to the pre-charge circuit, but the current to the other analog circuits are disabled until the ADC is enabled. As soon as the ADC is enabled, the bias currents to other analog circuits will be ready.

NOTE

Because of previous design versions, the eQADC will always wait 120 ADC clocks before issuing the first conversion command following the enabling the on-chip ADC, or the exiting of stop mode. There is an independent counter checking for this delay Conversion commands can start to be executed whenever the counter completes counting 120 ADC clocks.

31.4.5.2 ADC Clock and Conversion Speed

The clock input to the ADC is defined by setting the ADC0_CR[ADC0_CLK_PS](see [Section 31.3.4.1, “ADC0 Control Register \(ADC0_CR\)”](#)) The ADC0_CLK_PS field selects the clock divide factor by which the system clock will be divided as showed in [Table 31-20](#). The ADC clock frequency is calculated as below and it must not exceed 12 MHz.

$$\text{ADCClockFrequency} = \frac{\text{SystemClockFrequency(MHz)}}{\text{SystemClockDivideFactor}}; (\text{ADCClockFrequency} \leq 12\text{MHz})$$

Figure 31-35 depicts how the ADC clocks for ADC0 are generated.

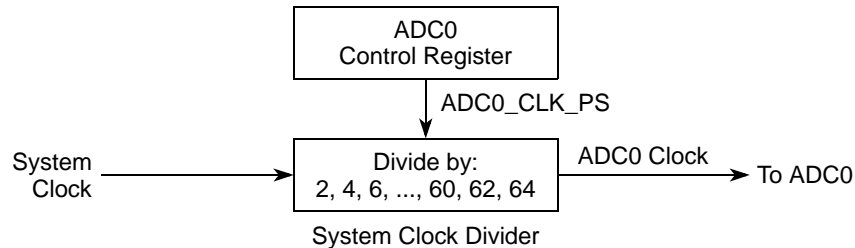


Figure 31-35. ADC0 Clock Generation

The ADC conversion speed (in kilosamples per second – ksamp/s) is calculated by the following formula. The number of sampling cycles is determined by the LST bits in the command message — see [Section , “Conversion Command Message Format for On-Chip ADC Operation,”](#) — and it can take one of the following values: 2, 8, 64, or 128 ADC clock cycles. The number of AD conversion cycles is 14. The maximum conversion speed is achieved when the ADC Clock frequency is set to its maximum (12 MHz) and the number of sampling cycles set to its minimum (2 cycles). The maximum conversion speed is 750ksamp/s.

$$\text{ADCConversionSpeed} = \frac{\text{ADCClockFrequency(MHz)}}{(\text{NumberOfSamplingCycles} + \text{NumberOfADConversionCycles})}$$

31.4.5.3 Time Stamp Feature

The on-chip ADC can provide a time stamp for the conversions they execute. A time stamp is the value of the time base counter latched when the eQADC detects the end of the analog input voltage sampling. A time stamp for a conversion command is requested by setting the TSR bit in the corresponding command. When TSR is negated, that is a time stamp is not requested, the ADC returns a single result message containing the conversion result. When TSR is asserted, that is a time stamp is requested, the ADC returns two result messages; one containing the conversion result, and another containing the time stamp for that conversion. The result messages are sent in this order to the RFIFOs and both messages are sent to the same RFIFO as specified in the MESSAGE_TAG field of the executed conversion command.

The time base counter is a 16-bit up counter and wraps after reaching 0xFFFF. It is disabled after reset and it is enabled according to the setting of ADC_TSCR[TBC_CLK_PS] field (see [Section 31.3.4.2, “ADC Time Stamp Control Register \(ADC_TSCR\)”](#)). TBC_CLK_PS defines if the counter is enabled or disabled, and, if enabled, at what frequency it is incremented. The time stamps are returned regardless of whether the time base counter is enabled or disabled. The time base counter can be reset by writing 0x0000 to the ADC_TBCR ([Section 31.3.4.3, “ADC Time Base Counter Registers \(ADC_TBCR\)”](#)) with a write configuration command.

31.4.5.4 ADC Calibration Feature

31.4.5.4.1 Calibration Overview

The eQADC provides a calibration scheme to remove the effects of gain and offset errors from the results generated by the on-chip ADC. Only results generated by the on-chip ADC are calibrated. The main component of calibration hardware is a multiply-and-accumulate (MAC) unit, one per on-chip ADC, that is used to calculate the following transfer function which relates a calibrated result to a raw, uncalibrated one.

$$\text{CAL_RES} = \text{GCC} * \text{RAW_RES} + \text{OCC} + 2;$$

where:

- CAL_RES is the calibrated result corresponding the input voltage V_i .
- GCC is the gain calibration constant.
- RAW_RES is the raw, uncalibrated result corresponding to an specific input voltage V_i .
- OCC is the offset calibration constant.
- The addition of two reduces the maximum quantization error of the ADC. See [Section 31.5.6.3, “Quantization Error Reduction During Calibration.”](#)

Calibration constants GCC and OCC are determined by taking two samples of known reference voltages and using these samples to calculate their values. For details and an example about how to calculate the calibration constants and use them in result calibration refer to [Section 31.5.6, “ADC Result Calibration.”](#) Once calculated, GCC is stored in ADC0_GCCR (see [Section 31.3.4.4, “ADC0 Gain Calibration Constant Register \(ADC0_GCCR\)”](#)) and OCC in ADC0_OCCR(see [Section 31.3.4.5, “ADC0 Offset Calibration Constant Register \(ADC0_OCCR\)”](#)) from where their values are fed to the MAC unit.

A conversion result is calibrated according to the status of CAL bit in the command that initiated the conversion. If the CAL bit is asserted, the eQADC will automatically calculate the calibrated result before sending the result to the appropriate RFIFO. If the CAL bit is negated, the result is not calibrated, it bypasses the calibration hardware, and is directly sent to the appropriate RFIFO.

31.4.5.4.2 MAC Unit and Operand Data Format

The MAC unit diagram is shown in [Figure 31-36](#). Each on-chip ADC has a separate MAC unit to calibrate its conversion results.

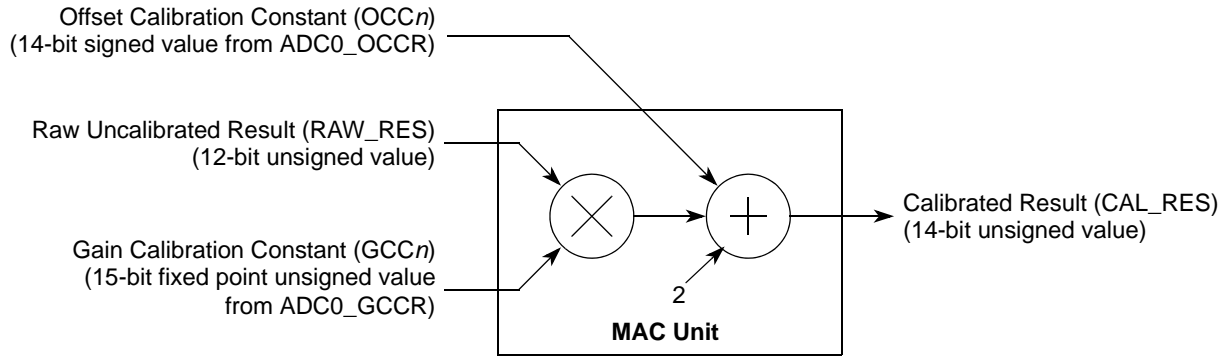


Figure 31-36. MAC Unit Diagram

The OCC_n operand is a 14-bit signed value and it is the upper 14 bits of the value stored in ADC0_OCCR. The RAW_RES operand is the raw uncalibrated result, and it is a direct output from the on-chip ADC.

The GCC_n operand is a 15-bit fixed point unsigned value, and it is the upper 15 bits of the value stored in ADC0_GCCR. The GCC is expressed in the $GCC_INT.GCC_FRAC$ binary format. The integer part of the GCC ($GCC_INT = GCC[1]$) contains a single binary digit while its fractional part ($GCC_FRAC = GCC[2:15]$) contains 14 bits. See Figure 31-37 for more information. The gain constant equivalent decimal value ranges from 0 to 1.999938..., as shown in Table 31-35. Two is always added to the MAC output: see Section 31.5.6.3, “Quantization Error Reduction During Calibration. CAL_RES output is the calibrated result, and it is a 14-bit unsigned value. CAL_RES is truncated to 0x3FFF, in case of a overflow, and to 0x0000, in case of an underflow.

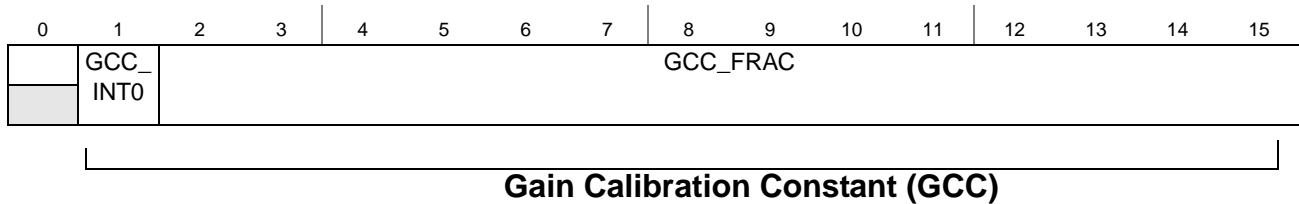


Figure 31-37. Gain Calibration Constant Format

Table 31-34. Gain Calibration Constant Format Field Descriptions

Field	Description
bit 0	Reserved
GCC_INT0	Integer part of the gain calibration constant for ADC0. GCC_INT is the integer part of the gain calibration constant (GCC) for ADC0/1.
GCC_FRAC	Fractional part of the gain calibration constant for ADC0. GCC_FRAC is the fractional part of the gain calibration constant (GCC) for ADC0. GCC_FRAC can express decimal values ranging from 0 to 0.999938...

Table 31-35. Correspondence between Binary and Decimal Representations of the Gain Constant

Gain Constant (GCC_INT.GCC_FRAC binary format)	Corresponding Decimal Value
0.0000_0000_0000_00	0
...	...
0.1000_0000_0000_00	0.5
...	...
0.1111_1111_1111_11	0.999938...
1.0000_0000_0000_00	1
...	...
1.1100_0000_0000_00	1.75
...	...
1.1111_1111_1111_11	1.999938...

31.4.5.5 ADC Control Logic Overview and Command Execution

Figure 31-38 shows the basic logic blocks involved in the ADC control and how they interact. CFIFOs/RFIFOs interact with ADC command/result message return logic through the FIFO control unit. The EB and BN bits in the command message uniquely identify the ADC to which a command should be sent. The FIFO control unit decodes these bits and sends the ADC command to the proper ADC. Other blocks of logic are the result format and calibration submodule, the time stamp logic, and the MUX control logic.

The result format and calibration submodule formats the returning data into result messages and sends them to the RFIFOs. The returning data can be data read from an ADC register, a conversion result, or a time stamp. The formatting and calibration of conversion results also take place inside this submodule.

The time stamp logic latches the value of the time base counter when detecting the end of the analog input voltage sampling, and sends it to the result format and calibration submodule as time stamp information.

The MUX control logic generates the proper MUX control signals and, when the ADC0/1_EMUX bits are asserted, the MA signals based on the channel numbers extracted from the ADC Command.

ADC commands are stored in the ADC command buffers (2 entries) as they come in and they are executed on a first-in-first-out basis. After the execution of a command in ENTRY1 finishes, all commands are shifted one entry. After the shift, ENTRY0 is always empty and ready to receive a new command. Execution of configuration commands only starts when they reach ENTRY1. Consecutive conversion commands are pipelined, and their execution can start while in ENTRY0. This is explained below.

A/D conversion accuracy can be affected by the settling time of the input channel multiplexers. Some time is required for the channel multiplexer's internal capacitances to settle after the channel number is changed. If the time prior to and during sampling is not long enough to permit this settling, then the voltage on the sample capacitors will not accurately represent the voltage to be read. This is a problem in particular when external muxes are used.

To maximize settling time, when a conversion command is in buffer ENTRY1 and another conversion command is identified in ENTRY0, then the channel number of ENTRY0 is sent to the *MUX control logic* half an ADC clock before the start of the sampling phase of the command in ENTRY0. This pipelining of sample and settling phase is shown in Figure 31-39(b).

This provides more accurate sampling, which is specially important for applications that require high conversion speeds, i.e., with the ADC running at maximum clock frequency and with the analog input voltage sampling time set to a minimum (2 ADC clock cycles). In this case the short sampling time may not allow the multiplexers to completely settle. The second advantage of pipelining conversion commands is to provide equal conversion intervals even though the sample time increases on second and subsequent conversions. See Figure 31-39. This is important for any digital signal process application.

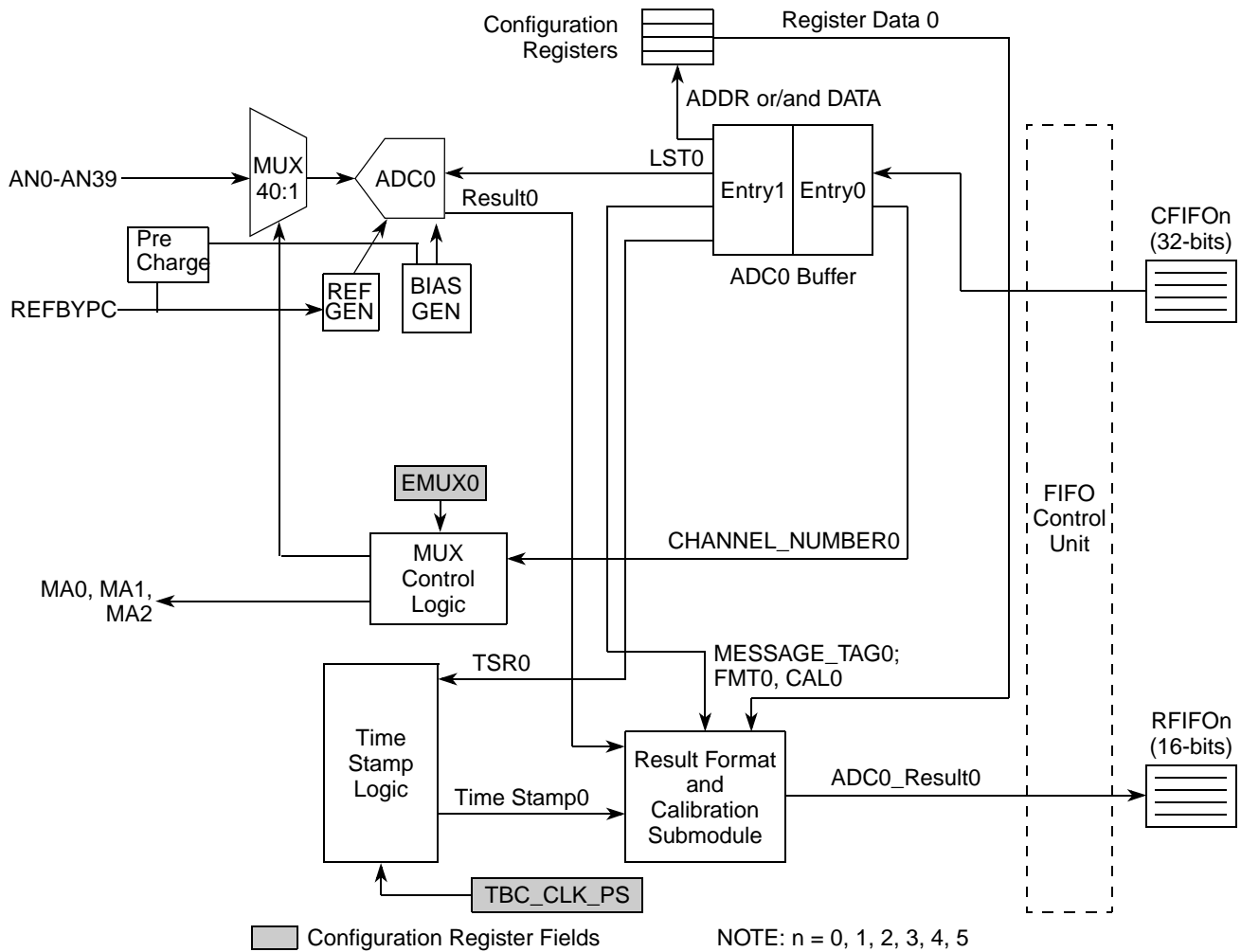
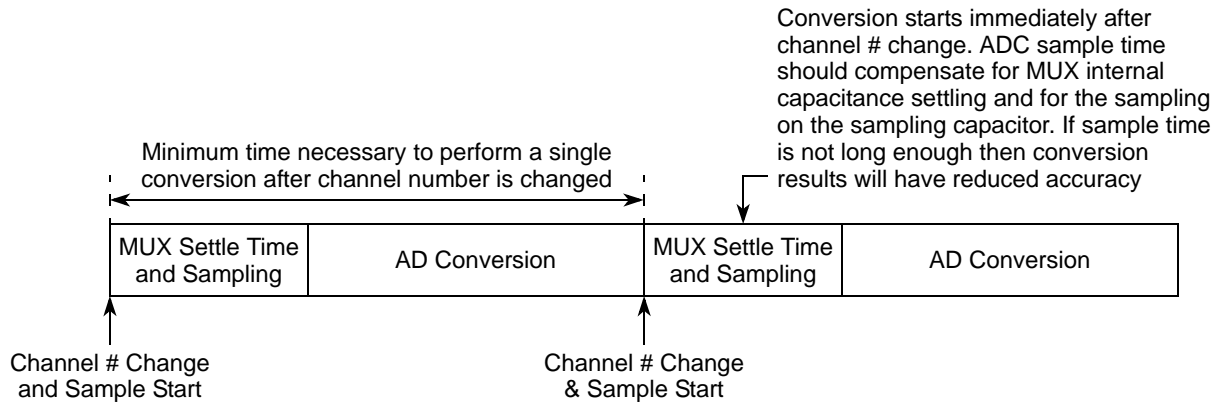
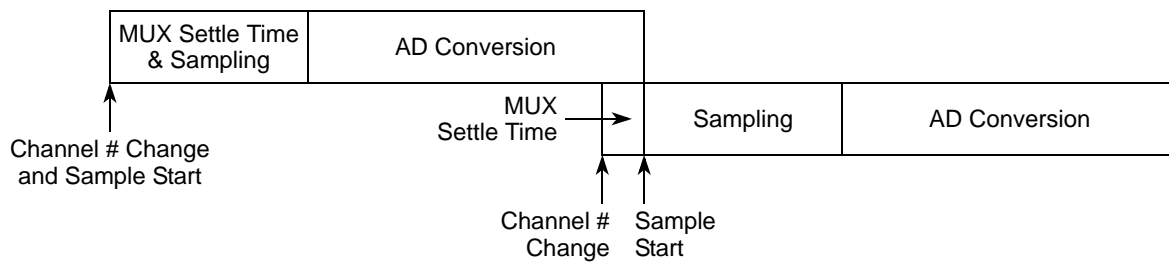


Figure 31-38. On-Chip ADC Control Scheme



(a) Command Execution Sequence for Two Non-Overlapped Commands



(b) Command Execution Sequence for Two Overlapped Commands

Figure 31-39. Overlapping Consecutive Conversion Commands

31.4.6 Internal/External Multiplexing

31.4.6.1 Channel Assignment

The internal analog multiplexers select one of the 40 analog input pins for conversion, based on the CHANNEL_NUMBER field of a Command Message.

Table 31-36 shows the channel number assignments for the non-multiplexed mode. The 40 single-ended channels are provided by the ADC.

Table 31-36. Non-multiplexed Channel Assignments

Input Pins			Channel Number in CHANNEL_NUMBER Field	
Analog Pin Name	Other Functions	Conversion Type	Binary	Decimal
AN0 to AN39		Single-ended	0000_0000 to 0010_0111	0 to 39
VRH		Single-ended	0010_1000	40
VRL		Single-ended	0010_1001	41

Table 31-36. Non-multiplexed Channel Assignments

Input Pins			Channel Number in CHANNEL_NUMBER Field	
	$(VRH - VRL)/2$ see footnote ¹	Single-ended	0010_1010	42
	75% x (VRH - VRL)	Single-ended	0010_1011	43
	25% x (VRH - VRL)	Single-ended	0010_1100	44
Reserved			0010_1101 to 1111_1111	45 to 255

¹ This equation only applies before calibration. After calibration, the 50% reference point will actually return approximately 20mV lower than the expected 50% of the difference between the High Reference Voltage (VRH) and the Low Reference Voltage (VRL). For calibration of the ADC only the 25% and 75% points should be used as described in [Section 31.5.6.1, “MAC Configuration Procedure”](#)

[Table 31-37](#) shows the channel number assignments for multiplexed mode. The ADC with the ADC0_EMUX bit asserted can access 39 single-ended and, at most, 56 externally multiplexed channels. Refer to [Section 31.4.6.2, “External Multiplexing,”](#) for a detailed explanation about how external multiplexing can be achieved.

Table 31-37. Multiplexed Channel Assignments

Input Pins			Channel Number in CHANNEL_NUMBER Field	
Analog Pin Name	Other Functions	Conversion Type	Binary	Decimal
AN0 to AN7		Single-ended	0000_0000 to 0000_0111	0 to 7
Reserved			0000_1000 to 0000_1011	8 to 11
AN12 to AN15		Single-ended	0000_1100 to 0000_1111	12 to 15
Reserved			0001_0000 to 0001_0010	16 to 18
AN19 to AN39		Single-ended	0001_0011 to 0010_0111	19 to 39
VRH		Single-ended	0010_1000	40
VRL		Single-ended	0010_1001	41
	$(VRH - VRL)/2$	Single-ended	0010_1010	42
	75% x (VRH - VRL)	Single-ended	0010_1011	43
	25% x (VRH - VRL)	Single-ended	0010_1100	44
Reserved			0010_1101 to 0011_1111	45 to 63
ANW	—	Single-ended	0100_0xxx	64 to 71
ANX	—	Single-ended	0100_1xxx	72 to 79
ANY	—	Single-ended	0101_0xxx	80 to 87
ANZ	—	Single-ended	0101_1xxx	88 to 95
Reserved			0110_0000 to 1101_1111	96 to 223

Table 31-37. Multiplexed Channel Assignments (continued)

Input Pins			Channel Number in CHANNEL_NUMBER Field	
ANR	—	Single-ended	1110_0xxx	224 to 231
ANS	—	Single-ended	1110_1xxx	232 to 239
ANT	—	Single-ended	1111_0xxx	240 to 247
Reserved			1111_1000 to 1111_1111	248 to 255

31.4.6.2 External Multiplexing

The eQADC can use from one to seven external multiplexers to expand the number of analog signals that may be converted. Up to 56 analog channels can be converted through external multiplexer selection. The externally multiplexed channels are automatically selected by the CHANNEL_NUMBER field of a command message, in the same way done with internally multiplexed channels. The software selects the external multiplexed mode by setting the ADC0_EMUX bit in ADC0_CR. Figure 31-37 shows the channel number assignments for the multiplexed mode. There are 40 single-ended and, at most, 56 externally multiplexed channels that can be selected.

Figure 31-40 shows a typical configuration of four external multiplexer chips connected to the eQADC. The external multiplexer chip selects one of eight analog inputs and connects it to a single analog output, which is fed to a specific input of the eQADC. The eQADC provides three multiplexed address signals, MA0, MA1, and MA2, to select one of eight inputs. These three multiplexed address signals are connected to all four external multiplexer chips. The analog output of the four multiplex chips are each connected to four separate eQADC inputs, ANW, ANX, ANY, and ANZ. The MA pins correspond to the three least significant bits of the channel number that selects ANW, ANX, ANY, and ANZ with MA0 being the most significant bit - See Table 31-38.

Table 31-38. Encoding of MA Pins¹

Channel Number selecting ANW, ANX, ANY, ANZ, ANR, ANS, ANT (decimal)							MA0	MA1	MA2
ANW	ANX	ANY	ANZ	ANR	ANS	ANT			
64	72	80	88	224	232	240	0	0	0
65	73	81	89	225	233	241	0	0	1
66	74	82	90	226	234	242	0	1	0
67	75	83	91	227	235	243	0	1	1
68	76	84	92	228	236	244	1	0	0
69	77	85	93	229	237	245	1	0	1
70	78	86	94	230	238	246	1	1	0
71	79	87	95	231	239	247	1	1	1

¹ 0 means pin is driven LOW and 1 that pin is driven HIGH.

When the external multiplexed mode is selected, the eQADC automatically creates the MA output signals from CHANNEL_NUMBER field of a command message. The eQADC also converts the proper input

channel (ANR, ANS, ANT, ANW, ANX, ANY, and ANZ) by interpreting the CHANNEL_NUMBER field. As a result, up to 56 externally multiplexed channels appear to the conversion queues as directly connected signals.

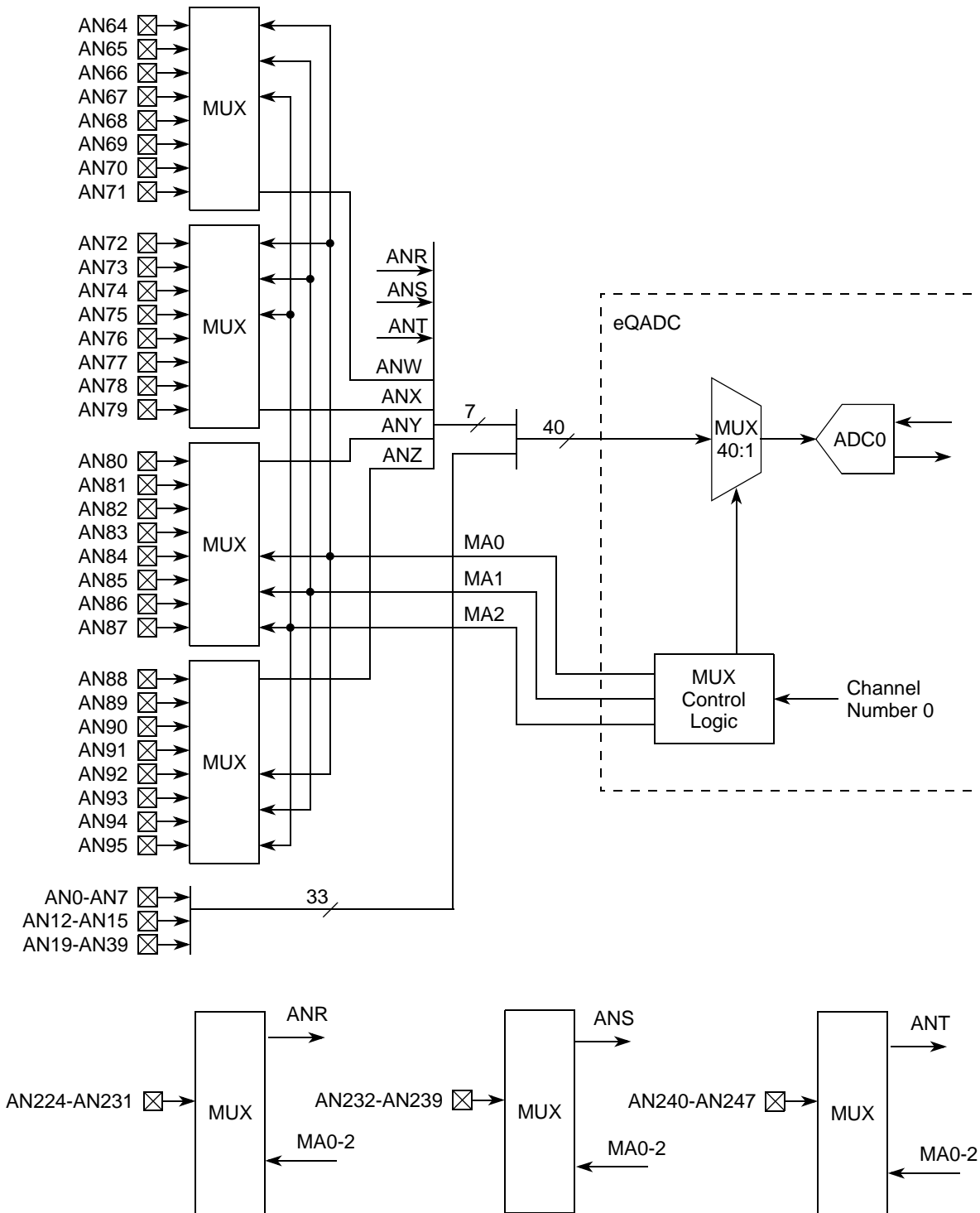


Figure 31-40. Example of External Multiplexing

31.4.7 eQADC eDMA/Interrupt Request

Table 31-39 lists methods to generate interrupt requests in the eQADC queuing control and triggering control. The eDMA/interrupt request select bits and the eDMA/interrupt enable bits are described in Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn),” and the interrupt flag bits are described in Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn).” Table 31-41 depicts all interrupts and eDMA requests generated by the eQADC.

Table 31-39. eQADC FIFO Interrupt Summary¹

Interrupt	Condition	Clearing Mechanism
Non Coherency Interrupt	NCIE _n = 1 NCF _n = 1	Clear NCF _n bit by writing a 1 to the bit.
Trigger Overrun Interrupt ²	TORIE _n = 1 TORF _n = 1	Clear TORF _n bit by writing a 1 to the bit.
Pause Interrupt	PIE _n = 1 PF _n = 1	Clear PF _n bit by writing a 1 to the bit.
End of Queue Interrupt	EOQIE _n = 1 EOQF _n = 1	Clear EOQF _n bit by writing a 1 to the bit.
Command FIFO Underflow Interrupt ²	CFUIE _n = 1 CFUF _n = 1	Clear CFUF _n bit by writing a 1 to the bit.
Command FIFO Fill Interrupt	CFFE _n = 1 CFFS _n = 0 CFFF _n = 1	Clear CFFF _n bit by writing a 1 to the bit.
Result FIFO Overflow Interrupt ²	RFOIE _n = 1 RFOF _n = 1	Clear RFOF _n bit by writing a 1 to the bit.
Result FIFO Drain Interrupt	RFDE _n = 1 RFDS _n = 0 RFDF _n = 1	Clear RFDF _n bit by writing a 1 to the bit.

¹ For details refer to Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn),” and Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn).”

² Apart from generating an independent interrupt request for when a RFIFO overflow interrupt, a CFIFO underflow interrupt, and a CFIFO trigger overrun interrupt occurs, the eQADC also provides a combined interrupt request at which these requests from ALL CFIFOs are ORed. Refer to Figure 31-41 for details.

Table 31-40 describes a list of methods to generate eDMA requests by the eQADC.

Table 31-40. eQADC FIFO eDMA Summary¹

eDMA Request	Condition	Clearing Mechanism
Result FIFO Drain eDMA Request	RFDE _n = 1 RFDS _n = 1 RFDF _n = 1	The eQADC automatically clears the RFDF _n when RFIFOn becomes empty. Writing 1 to the RFDF _n bit is not allowed while RDFS = 1.
Command FIFO Fill eDMA Request	CFFE _n = 1 CFFS _n = 1 CFFF _n = 1	The eQADC automatically clears the CFFF _n when CFIFOn becomes full. Writing 1 to the CFFF _n bit is not allowed while CFDS = 1.

¹ For details refer to Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 (EQADC_FISRn),” and Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn).”

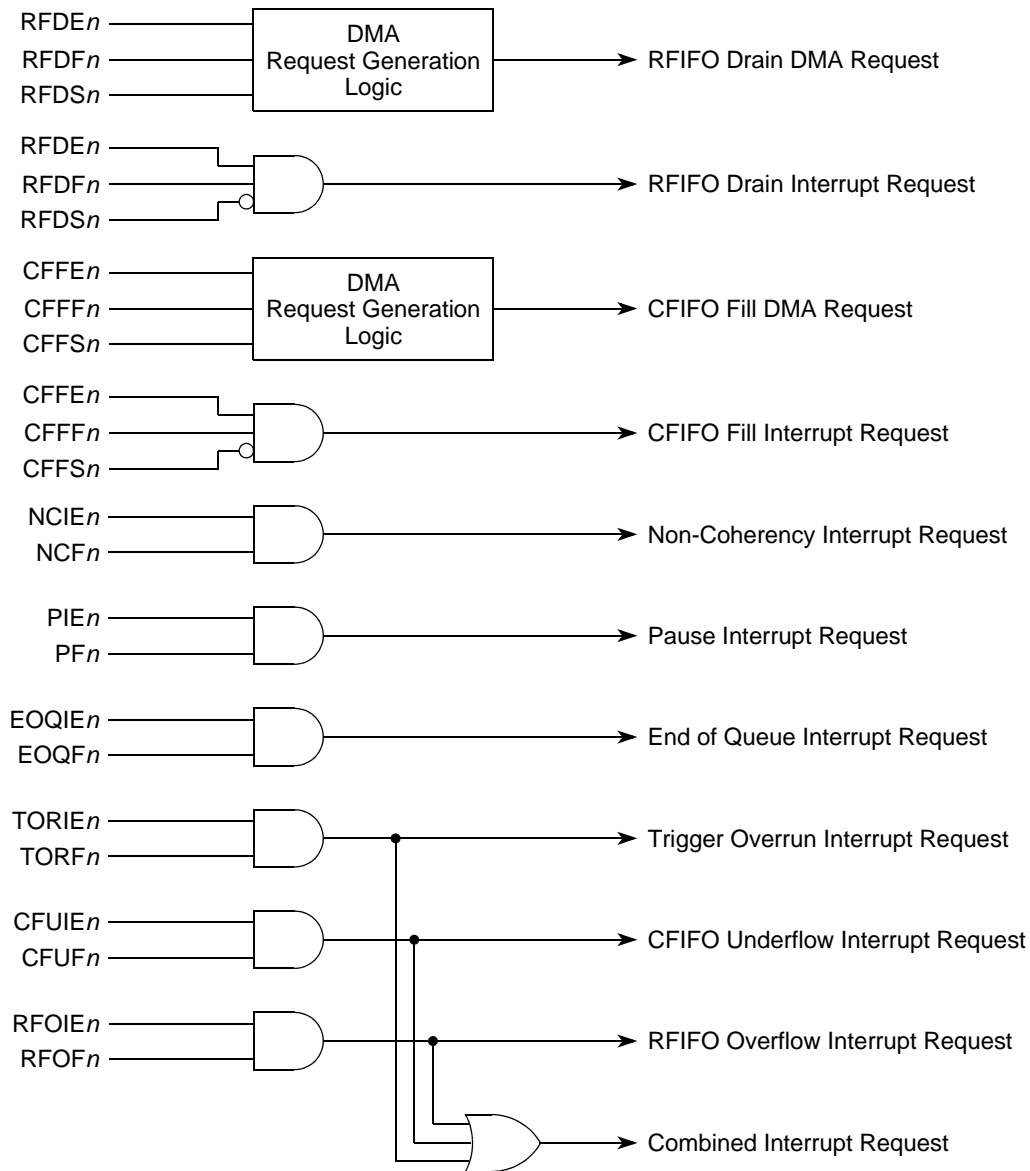


Figure 31-41. eQADC eDMA and Interrupt Requests

31.4.8 Analog Submodule

31.4.8.1 Reference Bypass

The reference bypass capacitor (REFBYPC) signal requires a 100 nF capacitor connected to VRL to filter noise on the internal reference used by the ADC.

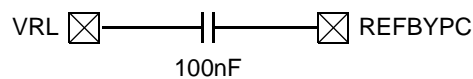


Figure 31-42. Reference Bypass Circuit

31.4.8.2 Analog-to-Digital Converter (ADC)

31.4.8.2.1 ADC Overview

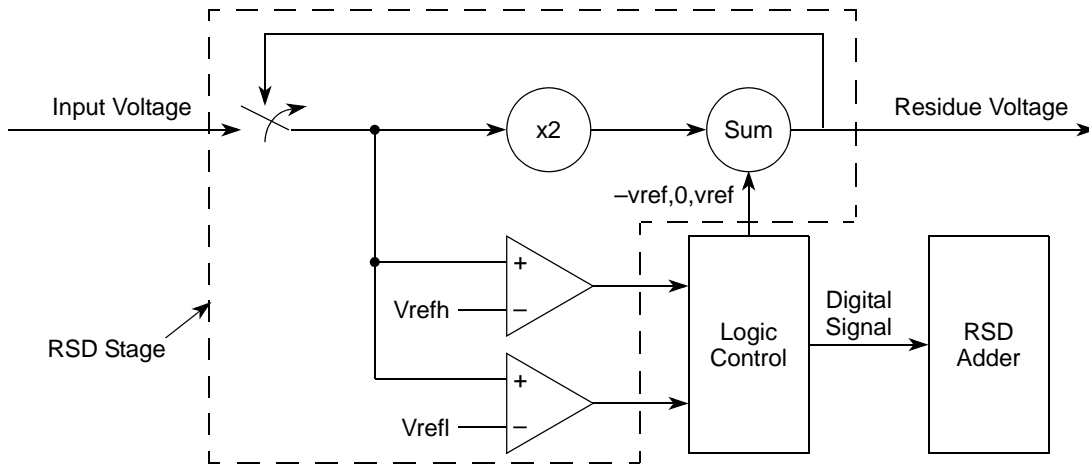


Figure 31-43. RSD ADC Block Diagram

The redundant signed digit (RSD) cyclic ADC consists of two main portions, the analog RSD stage, and the digital control and calculation module, as shown in Figure 31-43. To begin an analog-to-digital conversion, an input voltage is passed into the analog RSD stage and then from the RSD stage output, back to its input to be passed again. To complete a 12-bit conversion, the signal must pass through the RSD stage 13 times. Each time an input signal is read into the RSD stage, a digital sample is taken by the digital control/calculation module. The digital control/calculation module uses this sample to tell the analog module how to condition the signal. The digital module also saves each successive sample and adds them according to the RSD algorithm at the end of the entire conversion cycle.

On each pass through the RSD stage, the input signal will be multiplied by exactly two, and summed with either $-v_{ref}$, 0, or v_{ref} , depending on the logic control. The logic control will determine $-v_{ref}$, 0, or v_{ref} depending on the two comparator inputs. As the logic control sets the summing operation, it also sends a digital value to the RSD adder. Each time an analog signal passes through the RSD single-stage, a digital value is collected by the RSD adder. At the end of an entire AD conversion cycle, the RSD adder uses these collected values to calculate the 12-bit digital output.

Figure 31-44 shows the transfer function for the RSD stage. Note how the digital value (AB) is dependent on the two comparator inputs.

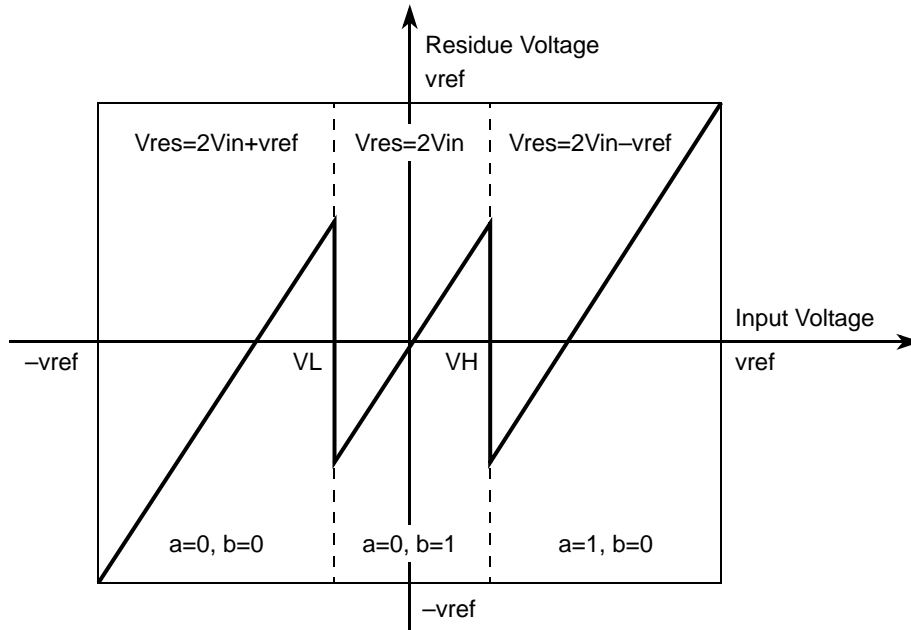


Figure 31-44. RSD Stage Transfer Function

In each pass through the RSD stage, the residue will be sent back to be the new input, and the digital signals, a and b, will be stored. For the 12-bit ADC, the input signal is sampled during the input phase, and after each of the 12 passes through the RSD stage. Thus, 13 total a and b values are collected. Upon collecting all these values, they will be added according to the RSD algorithm to create the 12-bit digital representation of the original analog input. The bits are added in the following manner:

31.4.8.2.2 RSD Adder

The array, s1 to s12, will be the digital output of the RSD ADC with s1 being the msb and s12 being the lsb (least significant bit).

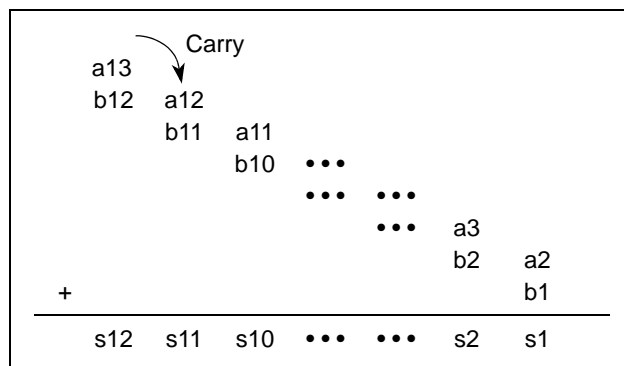


Figure 31-45. RSD Adder

31.5 Initialization/Application Information

31.5.1 Multiple Queues Control Setup Example

This section provides an example of how to configure multiple user command queues. [Table 31-41](#) describes how each queue can be used for a different application. Also documented in this section are general guidelines on how to initialize the on-chip ADC, and how to configure the command queues and the eQADC.

Table 31-41. Example Applications of Each Command Queue

Command Queue Number	Queue Type	Running Speed	Number of Contiguous Conversions	Example
0	Very fast burst time-based queue	every 2 μ s for 200 μ s; pause for 300 μ s and then repeat	2	Injector current profiling
1	Fast hardware-triggered queue	every 900 μ s	3	Current sensing of PWM controlled actuators
2	Fast repetitive time-based queue	every 2 ms	8	Throttle position
3	Software-triggered queue	every 3.9 ms	3	Command triggered by software strategy
4	Repetitive angle-based queue	every 625 μ s	7	Airflow read every 30 degrees at 8000 RPM
5	Slow repetitive time-based queue	every 100 ms	10	Temperature sensors

31.5.1.1 Initialization of On-Chip ADC

The following steps provide an example of configuring the eQADC to initialize the on-chip ADC. In this example, commands will be sent through CFIFO0.

1. Load all required configuration commands in the RAM in such way that they form a queue; this data structure will be referred below as Queue0. [Figure 31-46](#) shows an example of a command queue able to configure the on-chip ADC.
2. Configure [Section 31.3.3.2, “eQADC Null Message Send Format Register \(EQADC_NMSFR\).”](#)
3. Configure the eDMA to transfer data from Queue0 to CFIFO0 in the eQADC.
4. Configure [Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 \(EQADC_IDCRn\).”](#)
 - a) Set CFFS0 to configure the eQADC to generate an eDMA request to load commands from Queue0 to the CFIFO0.
 - b) Set CFFE0 to enable the eQADC to generate an eDMA request to transfer commands from Queue0 to CFIFO0; Command transfers from the RAM to the CFIFO0 will start immediately.

- c) Set EOQIE0 to enable the eQADC to generate an interrupt after transferring all of the commands of Queue0 through CFIFO0.
5. Configure [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\).”](#)
 - a) Write 0b0001 to the MODE0 field in eQADC_CFCR0 to program CFIFO0 for software single-scan mode.
 - b) Write 1 to SSE0 to assert SSS0 and trigger CFIFO0.
6. Because CFIFO0 is in single-scan software mode and it is also the highest priority CFIFO, the eQADC starts to transfer configuration commands to the on-chip ADC.
7. When all of the configuration commands have been transferred, EQADC_FISR_n[CF0] (see [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\)”](#)) will be set. The eQADC generates an end-of-queue interrupt. The initialization procedure is complete.

		Command Queue in System Memory		
Command Address	0x0 0x1	<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="border-bottom: 1px solid black; padding: 2px 5px;">Configuration Command to ADC0—Ex: Write ADC0_CR</td> </tr> <tr> <td style="padding: 2px 5px;">Configuration Command to ADC0—Ex: Write ADC_TSCR</td> </tr> </table>	Configuration Command to ADC0—Ex: Write ADC0_CR	Configuration Command to ADC0—Ex: Write ADC_TSCR
Configuration Command to ADC0—Ex: Write ADC0_CR				
Configuration Command to ADC0—Ex: Write ADC_TSCR				

Figure 31-46. Example of a Command Queue Configuring the On-Chip ADC

31.5.1.2 Configuring eQADC for Applications

This section provides an example based on the applications in [Table 31-41](#). The example describes how to configure multiple command queues to be used for those applications and provides a step-by-step procedure to configure the eQADC and the associated command queue structures. In the example, the “Fast hardware-triggered command queue,” described on the second row of [Table 31-41](#), will have its commands transferred to ADC; the conversion commands will be executed by ADC. The generated results will be returned to RFIFO3 before being transferred to the result queues in the RAM by the eDMA.

NOTE

There is no fixed relationship between CFIFOs and RFIFOs with the same number. The results of commands being transferred through CFIFO1 can be returned to any RFIFO, regardless of its number. The destination of a result is determined by the MESSAGE_TAG field of the command that requested the result. See [Section 31.4.1.1, “Message Format in eQADC,”](#) for details.

Step One: Set up the command queues and result queues.

1. Load the RAM with configuration and conversion commands. [Table 31-42](#) is an example of how command queue 1 commands should be set.
 - a) Each trigger event will cause four commands to be executed. When the eQADC detects the pause bit asserted, it will wait for another trigger to restart transferring commands from the CFIFO.
 - b) At the end of the command queue, the “EOQ” bit is asserted as shown in [Table 31-42](#).
 - c) Results will be returned to RFIFO3 as specified in the MESSAGE_TAG field of commands.
2. Reserve memory space for storing results.

Table 31-42. Example of Command Queue Commands¹

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
	EOQ	PAUSE	RESERVED	ABORT_ST	EB (0b1)	BN	CAL	MESSAGE TAG	ADC COMMAND																											
CMD1	0	0	0	0	0	1	0	0b0011	Conversion Command																											
CMD2	0	0	0	0	0	1	0	0b0011	Conversion Command																											
CMD3	0	0	0	0	0	1	0	0b0011	Conversion Command																											
CMD4	0	1	0	0	0	1	0	0b0011 ²	Configure peripheral device for next conversion sequence																											
CMD5	0	0	0	0	0	1	0	0b0011	Conversion Command																											
CMD6	0	0	0	0	0	1	0	0b0011	Conversion Command																											
CMD7	0	0	0	0	0	1	0	0b0011	Conversion Command																											
CMD8	0	1	0	0	0	1	0	0b0011 ²	Configure peripheral device for next conversion sequence																											
⋮									etc.....																											
CMDEOQ	1	0	0	0	0	1	0	0b0011	EOQ Message																											
	CFIFO Header								ADC Command																											

¹ Fields LST, TSR, FMT, and CHANNEL_NUMBER are not shown for clarity. See Section , “Conversion Command Message Format for On-Chip ADC Operation,” for details.

² MESSAGE_TAG field is only defined for read configuration commands.

Step Two: Configure the eDMA to handle data transfers between the command/result queues in RAM and the CFIFOs/RFIFOs in the eQADC.

1. For transferring, set the source address of the eDMA TCD_n to point to the start address of command queue 1. Set the destination address of the eDMA to point to EQADC_CFPR1. Refer to Section 31.3.3.4, “eQADC CFIFO Push Registers 0–5 (EQADC_CFPRn).”
2. For receiving, set the source address of the eDMA TCD_n to point to EQADC_RFPR3. Refer to Section 31.3.3.5, “eQADC Result FIFO Pop Registers 0–5 (EQADC_RFPRn).” Set the destination address of the eDMA to point to the starting address of result queue 1.

Step Three: Configure the eQADC control registers.

3. Configure Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 (EQADC_IDCRn).”
 - a) Set EOQIE1 to enable the End of Queue Interrupt request.
 - b) Set CFFS1 and RFDS3 to configure the eQADC to generate eDMA requests to push commands into CFIFO1 and to pop result data from RFIF03.
 - c) Set CFINV1 to invalidate the contents of CFIFO1.
 - d) Set RFDE3 and CFFE1 to enable the eQADC to generate eDMA requests. Command transfers from the RAM to the CFIFO1 will start immediately.

- e) Set RFOIE3 to indicate if RFIFO3 overflows.
 - f) Set CFUIE1 to indicate if CFIFO1 underflows.
4. Configure MODE1 to continuous-scan rising edge external trigger mode in [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\).”](#)

Step Four: Command transfer to ADCs and result data reception.

When an external rising edge event occurs for CFIFO1, the eQADC automatically will begin transferring commands from CFIFO1 when it becomes the highest priority CFIFO trying to send commands to ADC0. The received results will be placed in RFIFO3 and then moved to result queue 1 by the eDMA.

31.5.2 eQADC/eDMA Controller Interface

This section provides an overview of the EQADC/eDMA interface and general guidelines about how the eDMA should be configured in order for it to correctly transfer data between the queues in system memory and the EQADC FIFOs.

31.5.2.1 Command Queue/CFIFO Transfers

In transfers involving command queues and CFIFOs, the eDMA moves data from a queued source to a single destination as shown in [Figure 31-47](#). The location of the data to be moved is indicated by the source address, and the final destination for that data, by the destination address. The eDMA has transfer control descriptors (TCDs) containing these addresses and other parameters used in the control of data transfers (See [Section 12.3.2.16, “Transfer Control Descriptor \(TCD\)”](#) for more information). For every eDMA request issued by the EQADC, the eDMA must be configured to transfer a single command (32-bit data) from the command queue, pointed to by the source address, to the CFIFO push register, pointed to by the destination address. After the service of an eDMA request is completed, the source address has to be updated to point to the next valid command. The destination address remains unchanged. When the last command of a queue is transferred one of the following actions is recommended. Refer to [Chapter 12, “Enhanced Direct Memory Access \(eDMA\)”](#) for details about how this functionality is supported.

- The corresponding eDMA channel should be disabled. This might be desirable for CFIFOs in single scan mode.
- The source address should be updated to pointed to a valid command which can be the first command in the queue that has just been transferred (cyclic queue), or the first command of any other command queue. This is desirable for CFIFOs in continuous scan mode, or in some cases, for CFIFOs in single scan mode.

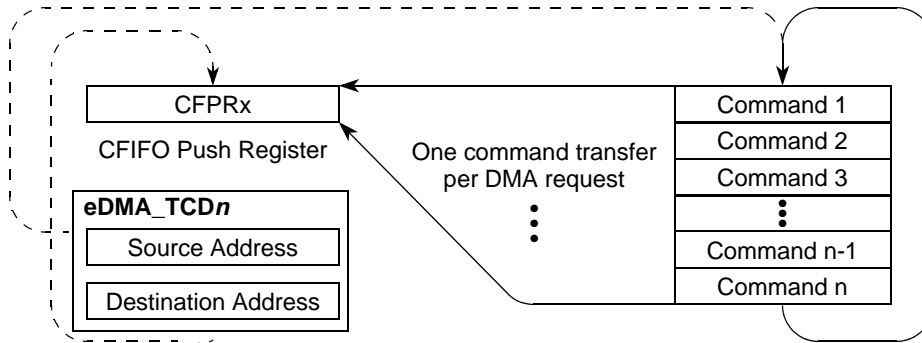


Figure 31-47. Command Queue/CFIFO Interface

31.5.2.2 Receive Queue/RFIFO Transfers

In transfers involving receive queues and RFIFOs, the eDMA controller moves data from a single source to a queue destination as shown in Figure 31-48. The location of the data to be moved is indicated by the source address, and the final destination for that data, by the destination address. For every eDMA request issued by the EQADC, the eDMA controller has to be configured to transfer a single result (16-bit data), pointed to by the source address, from the RFIFO pop register to the receive queue, pointed to by the destination address. After the service of an eDMA request is completed, the destination address has to be updated to point to the location where the next 16-bit result will be stored. The source address remains unchanged. When the last expected result is written to the receive queue, one of the following actions is recommended. Refer to Chapter 12, “Enhanced Direct Memory Access (eDMA)” for details about how this functionality is supported.

- The corresponding eDMA channel should be disabled.
- The destination address should be updated pointed to the next location where new coming results are stored, which can be the first entry of the current receive queue (cyclic queue), or the beginning of a new receive queue.

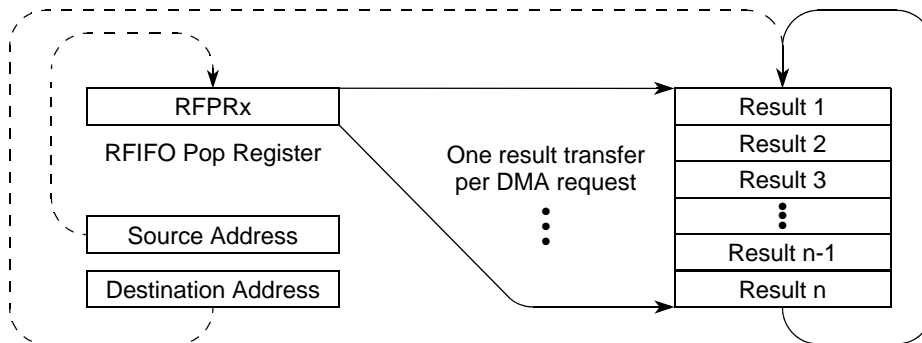


Figure 31-48. Receive Queue/RFIFO Interface

31.5.3 Sending Immediate Command Setup Example

In the eQADC, there is no immediate command register for sending a command immediately after writing to that register. However, a CFIFO can be configured to perform the same function as an immediate

command register. The following steps illustrate how to configure CFIFO5 as an immediate command CFIFO. This eliminates the use of the eDMA. The results will be returned to RFIFO5.

1. Configure the [Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 \(EQADC_IDCRn\).”](#)
 - a) Clear CFIFO fill enable5 (CFFE5 = 0) in EQADC_IDCR5.
 - b) Clear CFIFO underflow interrupt enable5 (CFUIE5 = 0) in EQADC_IDCR2.
 - c) Clear RFDS5 to configure the eQADC to generate interrupt requests to pop result data from RFIFO5.
 - d) Set RFIFO drain enable5 (RFDE5 = 1) in EQADC_IDCR5.
2. Configure the [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\).”](#)
 - a) Write 1 to CFINV5 in EQADC_CFCR5. This will invalidate the contents of CFIFO5.
 - a) Set MODE5 to continuous-scan software trigger mode in EQADC_CFCR5.
3. To transfer a command, write it to the eQADC CFIFO push register 5 (EQADC_CFPR5) with message tag = 0b0101. Refer to [Section 31.3.3.4, “eQADC CFIFO Push Registers 0–5 \(EQADC_CFPRn\).”](#)
4. Up to 4 commands can be queued in CFIFO5. Check the CFCTR5 status in EQADC_FISR5 before pushing another command to avoid overflowing the CFIFO. Refer to [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\).”](#)
5. When the eQADC receives a conversion result for RFIFO5, it generates an interrupt request. RFIFO pop register 5 (EQADC_RFPR5) can be popped to read the result. Refer to [Section 31.3.3.5, “eQADC Result FIFO Pop Registers 0–5 \(EQADC_RFPRn\).”](#)

31.5.4 Modifying Queues

More command queues may be needed than the six supported by the eQADC. These additional command queues can be supported by interrupting command transfers from a configured CFIFO, even if it is triggered and transferring, modifying the corresponding command queue in the RAM or associating another command queue to it, and restarting the CFIFO. More details on disabling a CFIFO are described in [Section 31.4.3.5.1, “Disabled Mode.”](#)

1. Determine the resumption conditions when later resuming the scan of the command queue at the point before it was modified.
 - a) Change EQADC_CFCRn[MODEn] (see [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\)”](#)) to disabled. Refer to [Section 31.4.3.5.1, “Disabled Mode,”](#) for a description of what happens when MODEn is changed to disabled.
 - b) Poll EQADC_CFSR[CFSn] until it becomes IDLE (see [Section 31.3.3.11, “eQADC CFIFO Status Register \(EQADC_CFSR\)”](#)).
 - c) Read and save EQADC_CFTCRn[TC_CFn] (see [Section 31.3.3.9, “eQADC CFIFO Transfer Counter Registers 0–5 \(EQADC_CFTCRn\)”](#)) for later resuming the scan of the queue. The TC_CFn provides the point of resumption.
 - d) Since all result data may not have been stored in the appropriate RFIFO at the time MODEn is changed to disable, wait for all expected results to be stored in the RFIFO/result queue before

reconfiguring the eDMA to work with the modified result queue. The number of results that must return can be estimated from the TC_CFn value obtained above.

2. Disable the eDMA from responding to the eDMA request generated by EQADC_FISRn[CFFFn] and EQADC_FISRn[RFDFn] (see [Section 31.3.3.8, “eQADC FIFO and Interrupt Status Registers 0–5 \(EQADC_FISRn\)”](#)).
3. Write “0x0000” to the TC_CFn field.
4. Load the new configuration and conversion commands into RAM. Configure the eDMA to support the new command/result queue, but do not configure it yet to respond to eDMA requests from CFIFOn/RFIFOn.
5. If necessary, modify the EQADC_IDCRn registers (see [Section 31.3.3.7, “eQADC Interrupt and eDMA Control Registers 0–5 \(EQADC_IDCRn\)”](#)) to suit the modified command queue.
6. Write 1 to EQADC_CFCRn[CFINVn] (see [Section 31.3.3.6, “eQADC CFIFO Control Registers 0–5 \(EQADC_CFCRn\)”](#)) to invalidate the entries of CFIFOn.
7. Configure the eDMA to respond to eDMA requests generated by CFFFn and RFDFn.
8. Change MODEn to the modified CFIFO operation mode. Write 1 to SSEn to trigger CFIFOn if MODEn is software trigger.

31.5.5 Command Queue and Result Queue Usage

[Figure 31-49](#) is an example of command queue and result queue usage. It shows the command queue 0 commands requesting results that will be stored in result queue 0 and result queue 1, and command queue 1 commands requesting results that will be stored only in result queue 1. Some command messages request data to be returned from the on-chip ADC, but some only configure them and do not request returning data. When a command queue contains both write and read commands like command queue 0, the command queue and result queue entries will not be aligned, in [Figure 31-49](#), the result for the second command of command queue 0 is the first entry of result queue 0. The figure also shows that command queue and result queue entries can also become unaligned even if all commands in a command queue request data as command queue 1. Command queue 1 entries became unaligned to result queue 1 entries because a result requested by the forth command queue 0 command was sent to result queue 1. This happens because the system can be configured so that several command queues can have results sent to a single result queue.

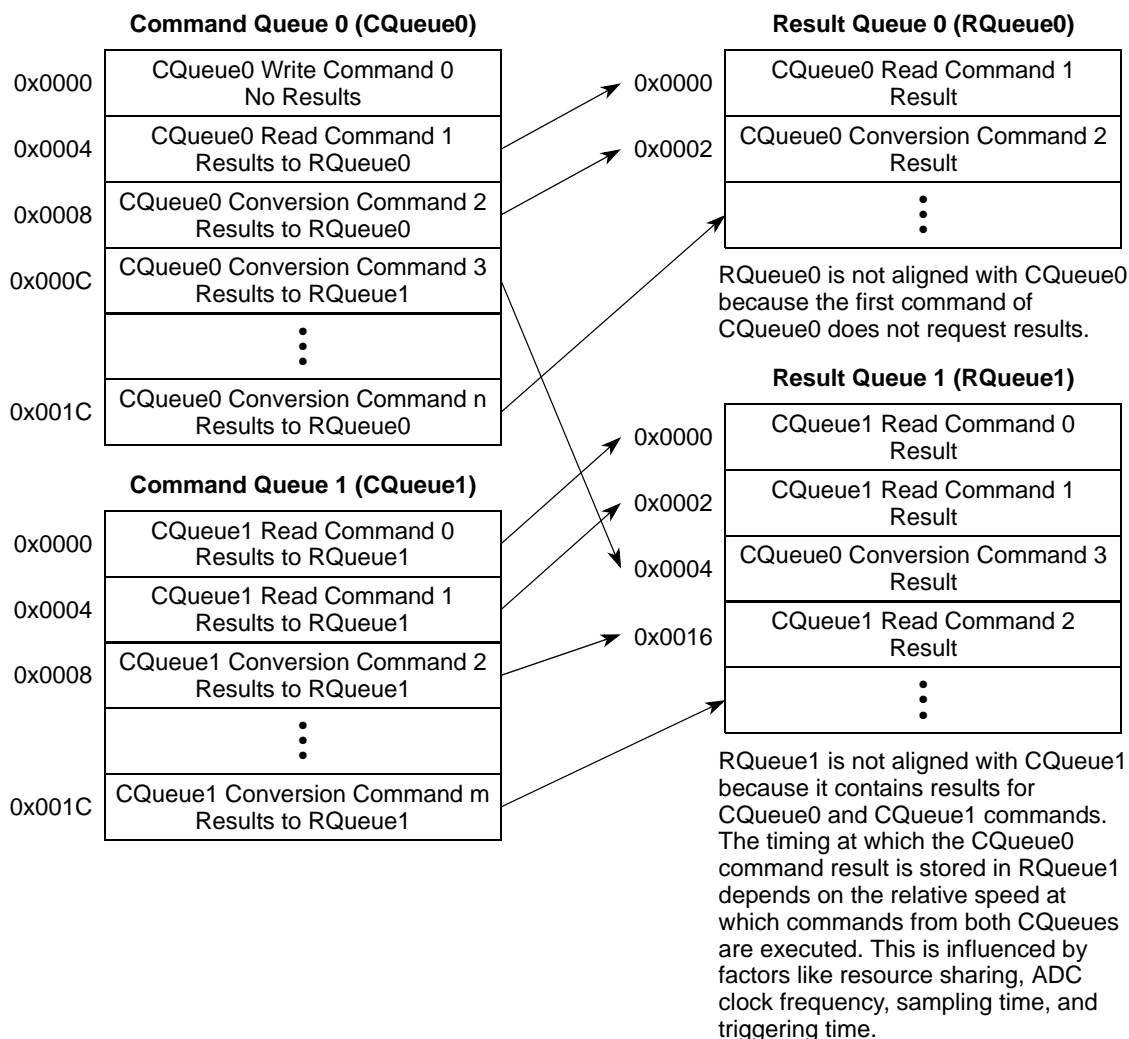


Figure 31-49. eQADC Command and Result Queues

31.5.6 ADC Result Calibration

The ADC result calibration process consists of two steps: determining the gain and offset calibration constants, and calibrating the raw results generated by the on-chip ADC by solving the following equation discussed in [Section 31.4.5.4.1, “Calibration Overview.”](#)

$$\text{CAL_RES} = \text{GCC} * \text{RAW_RES} + \text{OCC} + 2; \quad \text{Eqn. 31-1}$$

The calibration constants GCC and OCC can be calculated from [Equation 31-1](#) provided that two pairs of expected (CAL_RES) and measured (RAW_RES) result values are available for two different input voltages. Most likely calibration points to be used are 25% VREF¹ and 75% VREF since they are far apart but not too close to the end points of the full input voltage range. This allows for calculations of more representative calibration constants. The eQADC provides these voltages via channel numbers 43 and 44.

1. $V_{\text{REF}} = V_{\text{RH}} - V_{\text{RL}}$

The raw, uncalibrated results for these input voltages are obtained by converting these channels with conversion commands that have the CAL bit negated.

The transfer equations for when sampling these reference voltages are:

$$\text{CAL_RES}_{75\%VREF} = \text{GCC} * \text{RAW_RES}_{75\%VREF} + \text{OCC} + 2; \quad \text{Eqn. 31-2}$$

$$\text{CAL_RES}_{25\%VREF} = \text{GCC} * \text{RAW_RES}_{25\%VREF} + \text{OCC} + 2; \quad \text{Eqn. 31-3}$$

Thus;

$$\text{GCC} = (\text{CAL_RES}_{75\%VREF} - \text{CAL_RES}_{25\%VREF}) / (\text{RAW_RES}_{75\%VREF} - \text{RAW_RES}_{25\%VREF}); \quad \text{Eqn. 31-4}$$

$$\text{OCC} = \text{CAL_RES}_{75\%VREF} - \text{GCC} * \text{RAW_RES}_{75\%VREF} - 2; \quad \text{Eqn. 31-5}$$

or

$$\text{OCC} = \text{CAL_RES}_{25\%VREF} - \text{GCC} * \text{RAW_RES}_{25\%VREF} - 2; \quad \text{Eqn. 31-6}$$

After being calculated, the GCC and OCC values must be written to ADC0_GCCR (see [Section 31.3.4.4, “ADC0 Gain Calibration Constant Register \(ADC0_GCCR\)”](#)) and the ADC0_OCCR (see [Section 31.3.4.5, “ADC0 Offset Calibration Constant Register \(ADC0_OCCR\)”](#)) using write configuration commands.

The eQADC will automatically calibrate the results, according to [Equation 31-1](#), of every conversion command that has its CAL bit asserted using the GCC and OCC values stored in the ADC calibration registers.

31.5.6.1 MAC Configuration Procedure

The following steps illustrate how to configure the calibration hardware, that is, determining the values of the gain and offset calibration constants, and the writing these constants to the calibration registers.

1. Convert channel 44 with a command that has its CAL bit negated and obtain the raw, uncalibrated result for 25% VREF (RAW_RES_{25%VREF}).
2. Convert channel 43 with a command that has its CAL bit negated and obtain the raw, uncalibrated result for 75% VREF (RAW_RES_{75%VREF}).
3. Because the expected values for the conversion of these voltages are known (CAL_RES_{25%VREF} and CAL_RES_{75%VREF}), GCC and OCC values can be calculated from [Equation 31-4](#) and [Equation 31-5](#) using these values, and the results determined in steps 1 and 2.
4. Reformat GCC and OCC to the proper data formats as specified in [Section 31.4.5.4.2, “MAC Unit and Operand Data Format.”](#) GCC is an unsigned 15-bit fixed point value and OCC is a signed 14-bit value.
5. Write the GCC value to ADC0 gain calibration register (see [Section 31.3.4.4, “ADC0 Gain Calibration Constant Register \(ADC0_GCCR\)”](#)) and the OCC value to ADC0 offset calibration constant register (see [Section 31.3.4.5, “ADC0 Offset Calibration Constant Register \(ADC0_OCCR\)”](#)) using write configuration commands.

31.5.6.2 Example Calculation of Calibration Constants

The raw results obtained when sampling reference voltages 25% VREF and 75% VREF were, respectively, 3798 and 11592. The results that should have been obtained from the conversion of these reference voltages are, respectively, 4096 and 12288. Therefore, using Equation 31-4 and Equation 31-5, the gain and offset calibration constants are:

$$\begin{aligned} \text{GCC} &= (12288 - 4096) / (11592 - 3798) = 1.05106492 \rightarrow 1.05102539^1 = 0x4344 \\ \text{OCC} &= 12288 - 1.05106492 * 11592 - 2 = 102.06 \rightarrow 102 = 0x0066 \end{aligned}$$

Table 31-43 shows, for this particular case, examples of how the result values change according to GCC and OCC when result calibration is executed (CAL=1) and when it is not (CAL=0).

Table 31-43. Calibration Example

Input Voltage	Raw result (CAL=0)		Calibrated result (CAL=1)	
	Hexadecimal	Decimal	Hexadecimal	Decimal
25% VREF	0x0ED6	3798	0x1000	4095.794
75% VREF	0x2D48	11592	0x3000	12287.486

31.5.6.3 Quantization Error Reduction During Calibration

Figure 31-50 shows how the ADC transfer curve changes due to the addition of two to the MAC output during the calibration - see MAC output equation in Section 31.4.5.4, “ADC Calibration Feature.” The maximum absolute quantization error is reduced by half leading to an increase in accuracy.

1. This calculation is rounded down due to binary approximation.

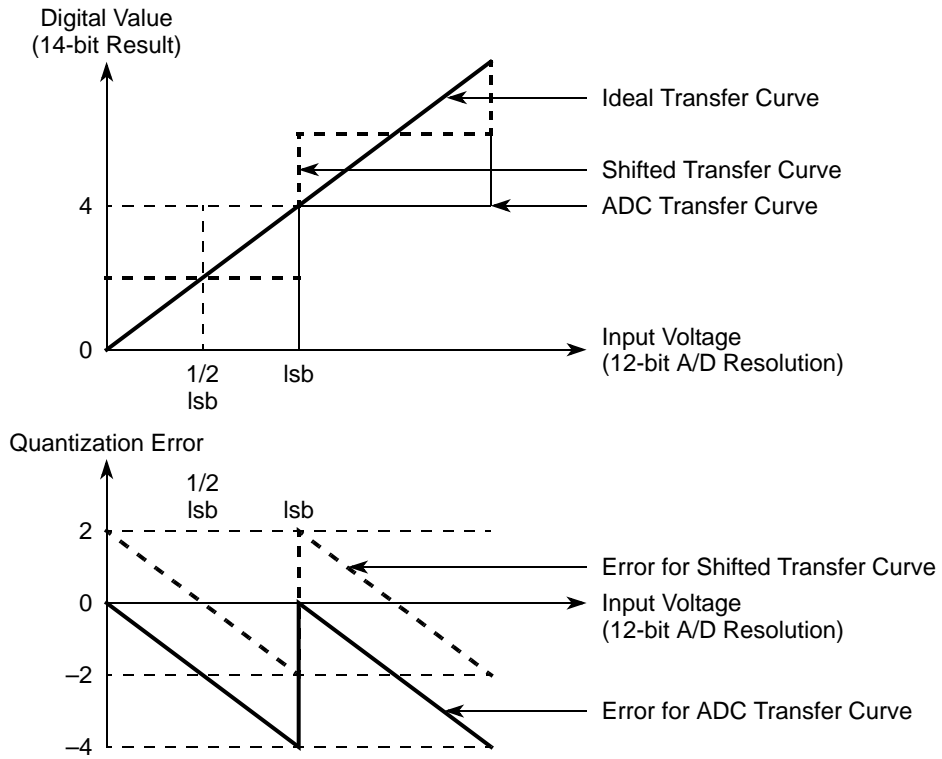


Figure 31-50. Quantization Error Reduction During Calibration

threeand

Chapter 32

Boot Assist Module (BAM)

32.1 Introduction

The MPC5510 boot assist module (BAM) is a 4-KB block of read-only memory (ROM) that contains the BAM program. The BAM program is compiled to variable length encoding (VLE) code. The BAM program is executed by the e200z1 when the MPC5510 performs a power-on-reset (POR), or any other reset, when the CRP_Z1VEC register remains in its reset state. The BAM program initializes the MCU, transitions to the user application code in the internal flash or downloads the user code into internal RAM via CAN or SCI serial links, and passes control to the user code.

32.1.1 Features

The BAM program provides the following functionality:

- Initial e200z1 core MMU setup with no address translation to allow the core to access all internal MCU resources and external memory address space
- Location and detection of user code in the internal flash
- Automatic switch to serial boot mode if internal flash is blank or invalid
- Supports user programmable 64-bit password protection for serial boot mode
- Supports serial bootloading via CAN bus or eSCI to the internal SRAM
- Supports censorship protection for internal flash memory
- Provides an option to disable the miscellaneous control module (MCM) software watchdog timer (enabled by default)
- Configures MMU to start user application, compiled in either classic Power Book E code or as Freescale VLE code

NOTE

The BAM program is intended to be run on e200z1 (main core) only. Attempting to execute the BAM program by the e200z0 core may cause erratic MCU behavior.

NOTE

During BAM program execution, the default reset values of various system registers (e.g. SIU, FlexCAN, eSCI, MCM SWT) may be updated.

32.1.2 Modes of Operation

32.1.3 Normal Mode

In normal operation the BAM responds to all read requests within its address space. The e200z1 core executes the BAM program after the negation of reset if the CRP CRP_Z1VEC register value is 0xFFFF_FFFC.

32.1.4 Debug Mode

The BAM program is not executed when the MCU comes out of reset in OnCE debug mode. The development tool must initialize the MCU instead of BAM before starting the user application.

32.1.5 Internal Boot Mode

This mode of operation is for systems that boot from internal memory. The internal flash is used for all code and the boot configuration data. After the BAM has completed the boot process, user code may enable the external bus interface if required.

32.1.6 Serial Boot Mode

This mode of operation can be used for initial MCU programming or for user system initialization. It allows a user program to be loaded into system RAM, using the eSCI or FlexCAN serial interface, then executed. The loaded program can be used to control the download of data, and the erasing or programming of internal or external flash memory.

32.2 Memory Map and Registers

This section provides a detailed description of the BAM memory map.

32.2.1 Module Memory Map

[Table 32-1](#) shows the BAM memory map.

The BAM ROM module occupies the last 16 KB of the MCU memory space; however, only the last 4 KB is physically present.

NOTE

Attempting to execute instructions from addresses in the range 0xFFFF_C000–0xFFFF_EFFF may cause unpredictable results.

Some important absolute addresses are presented in [Table 32-1](#).

Table 32-1. BAM Absolute Addresses

Address	Comment
0xFFFF_FFFC	BAM reset vector—first executed address after the reset
0xFFFF_F000	BAM start address

32.2.2 Register Descriptions

The BAM module does not have any registers.

32.3 Functional Description

32.3.1 BAM Program Resources

The BAM program uses/initializes these MCU resources:

- The BOOTCFG field in the reset status register (SIU_RSR) determines the boot mode
- The location and value of the reset configuration halfword (RCHW) determines the location of the boot code and the boot configuration options
- FlexCAN_A and eSCI_A modules for serial boot mode
- FlexCAN_A message buffers RAM for stack and global variables during serial boot mode
- The DISNEX bit in the SIU_CCR register to determine if the Nexus port is enabled

The BAM program:

- Configures the e200z1 MMU to allow access to all available MCU address space (internal flash, EBI, peripheral bridge, and SRAM) without address translation
- Configures FlexCAN_A and eSCI_A when performing serial boot mode
- Uses the eDMA during serial boot mode.

32.3.2 BAM Program Operation

If the CRP_Z1VEC register remains in its POR state, the BAM code is executed after the negation of reset and before user code starts. To prevent the execution of the BAM code upon exiting sleep mode, change the value of the CPR_Z1VEC register before entering the sleep mode. See [Section 5.2.2.6, “Z1 Reset Vector Register \(CRP_Z1VEC\),”](#) in [Chapter 5, “Clock, Reset, and Power Control \(CRP\),”](#) for more detail about the CRP_Z1VEC register.

The BAM reads the status of the BOOTCFG bit from the reset status register (SIU_RSR) and the appropriate boot sequence is started (see [Table 32-2](#)). The BOOTCFG bit reflects state of the BOOTCFG pin during the MCU reset.

[Table 32-2](#) shows the boot mode selection depending on BOOTCFG bit in the SIU_RSR, how the value stored in the Censorship word in the shadow row of internal flash memory effects whether the internal flash memory is enabled or disabled, whether the Nexus port is enabled or disabled, and whether the password downloaded in serial boot mode is compared to a fixed public password or to a user program-

mable flash password.

Table 32-2. Boot Modes

Boot Mode Name	BOOTCFG	Censorship Control 0x00FF_FDE0	Serial Boot Control 0x00FF_FDE2	Internal Flash State	Nexus State	Serial Password
Internal—Censored	0	Any other value	Don't care	Enabled	Disabled	Flash
Internal—Public		0x55AA		Enabled	Enabled	Public
Serial—Flash Password	1	Don't care	0x55AA	Enabled	Disabled	Flash
Serial—Public Password			Any other value	Disabled	Enabled	Public

The censorship control word is a 32-bit word of data stored in the shadow row of internal flash memory. This memory location is read and interpreted by hardware as part of the boot process. The memory address of the censorship control word is 0x00FF_FDE0. The censorship control word is factory programmed to be 0x55AA_55AA. This results in a device that is not censored and uses a flash-based password for serial boot mode.

Censorship control word at 0x00FF_FDE0:

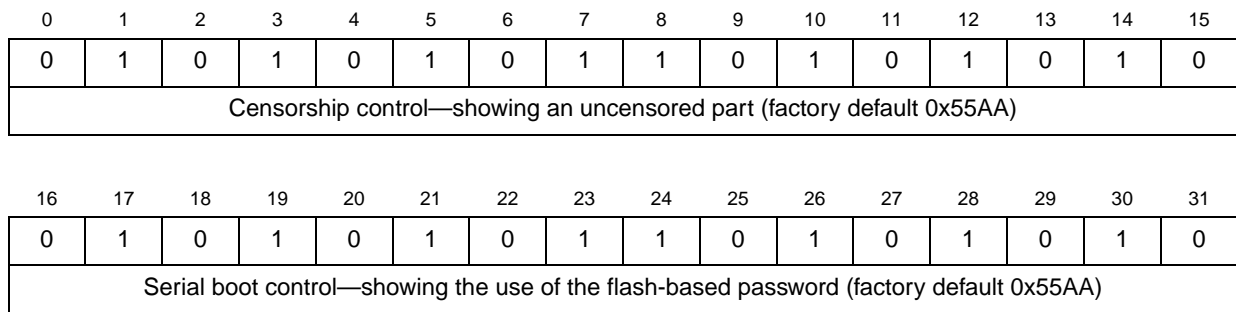


Figure 32-1. Censorship Control Word

The BAM code uses the state of the DISNEX bit (reflecting disabled status of the NEXUS port) to determine if the serial password downloaded in serial boot mode should be compared to a fixed public value (0xFEED_FACE_CAFE_BEEF) or to a flash value stored in the shadow row of internal flash at address 0x00FF_FDD8.

Serial boot flash password at 0x00FF_FDD8 – 0x00FF_FDDF:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1
Serial boot password (0x00FF_FDD8)–0xFEED (factory default)															
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0
Serial boot password (0x00FF_FDDA)–0xFACE (factory default)															
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0
Serial boot password (0x00FF_FDDC)–0xCAFE (factory default)															
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1	0	1	1	1	1	1	0	1	1	1	0	1	1	1	1
Serial boot password (0x00FF_FDDE)– 0xBEEF (factory default)															

Figure 32-2. Serial Boot Flash Password

Because the BAM program enters serial boot mode if it fails to find a valid RCHW in internal boot mode, a valid serial password must be programmed. The factory preprogrammed value of the serial password is the public password value (0xFEED_FACE_CAFE_BEEF).

32.3.3 Features

Because the MMU default out of reset is to allow access to the 4 KB range around the reset vector only, the BAM program sets up the e200z1 core MMU to enable accesses to all MCU resources, as described in [Table 32-3](#).

Table 32-3. MMU Configuration for an Internal Boot

TLB Entry	Region	Logical Base Address	Physical Base Address	Size	Attributes
0	Peripheral bridge and BAM	0xFFFF0_0000	0xFFFF0_0000	1 MB	Big Endian Global PID
1	Internal flash	0x0000_0000	0x0000_0000	256 MB	Big Endian Global PID
2	EBI	0x2000_0000	0x2000_0000	256 MB	Big Endian Global PID
3	SRAM	0x4000_0000	0x4000_0000	256 KB	Big Endian Global PID

Code type attributes for all TLB entries are set to be VLE from the beginning.

Code type attributes for TLB entries 1–3 are set later according to the VLE bit in the RCHW (see [Figure 32-3](#)) or the VLE bit received during serial boot mode (see [Section 32.3.3.2.2, “Serial Boot Mode Download Protocol”](#)) and should match the user application encoding (VLE or classic Power Book E).

After configuring the MMU, the BAM determines the selected boot mode and provides the following features for each of the boot modes:

32.3.3.1 Internal Boot Mode

When the core determines that internal boot mode has been selected, a machine check exception is configured to handle possible ECC read errors that may occur while searching the internal flash to find the reset configuration halfword (RCHW).

32.3.3.1.1 Reset Configuration Halfword Read

The BAM searches the internal flash memory for a valid RCHW. A valid RCHW is a 16-bit value that contains a fixed 8-bit boot identifier and some configuration bits. The RCHW is expected to be the first halfword in one of the locations shown in [Table 32-4](#).

Table 32-4. LAS Block Memory Addresses

Block	Address
0	0x0000_0000
1	0x0000_4000
4	0x0001_0000
7	0x0001_C000
8	0x0002_0000
9	0x0003_0000

The BOOT_BLOCK_ADDRESS used in the register descriptions below is the first address from [Table 32-4](#) where the BAM finds a valid RCHW.

[Figure 32-3](#) shows the fields of the RCHW.

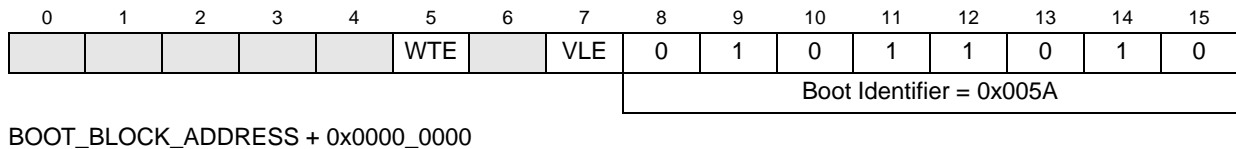


Figure 32-3. RCHW Fields

Table 32-5. Internal Boot RCHW Field Descriptions

Field	Description
bits 0–4	Reserved. These bit values are ignored when the halfword is read. Write to 0 for future compatibility.
WTE	Watchdog timer enable. This bit determines if the MCM software watchdog timer is disabled. 0 Disable software watchdog timer 1 Software watchdog timer maintains its default state out of reset, i.e. enabled. The timeout period is programmed to be 2^{17} system clocks.
	Reserved.
VLE	VLE Code Indicator. This bit is used to configure the MMU to execute the user code as either Classic Book E code or as Freescale VLE code. 0 User code executes as classic Book E code 1 User code executes as Freescale VLE code
BOOTID	Boot identifier. This field serves two functions. First, it is used to indicate which block in flash memory contains the boot program. Second, it identifies whether the flash memory is programmed or invalid. The value of a valid boot identifier is 0x005A (0b01011010). The BAM program checks the first halfword of each flash memory block starting at block 0 until a valid boot identifier is found. If all blocks in the low- address space of the internal flash are checked and no valid boot identifier is found, the internal flash is assumed to be invalid and a CAN/SCI boot is initiated.

If the BAM fails to find a valid RCHW, it switches to serial boot mode.

If the BAM finds a valid RCHW, the configuration bits are parsed as shown in [Table 32-5](#). The BAM then fetches the reset vector from the address of the RCHW + 4, and branches to that address.

BOOT_BLOCK_ADDRESS + 0x0000_0004

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31

Figure 32-4. Reset Boot Vector

32.3.3.2 Serial Boot Mode Features

In this mode of operation, the BAM code configures FlexCAN_A and eSCI_A for serial download of a user program. Unused message buffers in FlexCAN_A are used for stack and global variables. The system clock is selected directly from main crystal oscillator output; thus, the crystal frequency defines baud rates for serial interfaces, used to download the user application.

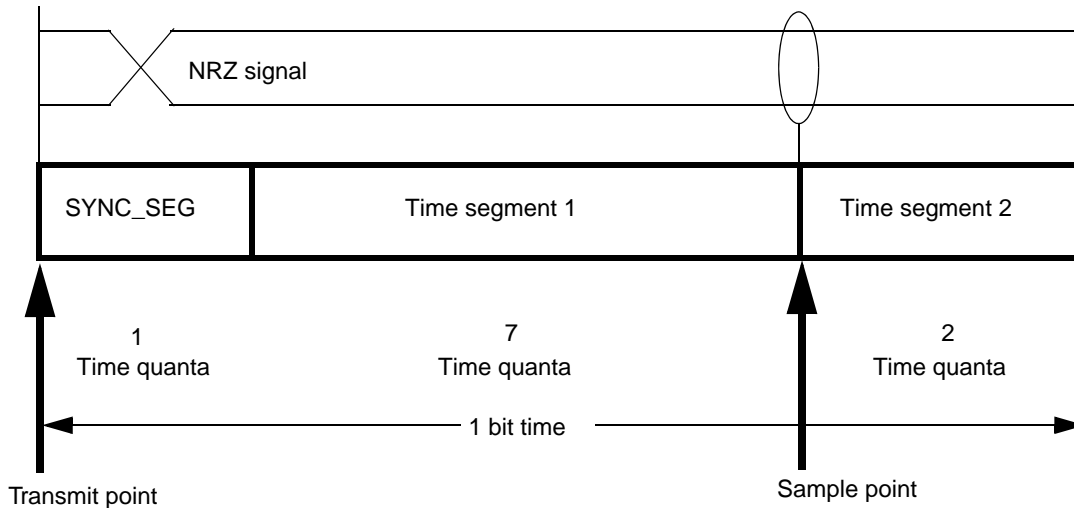
32.3.3.2.1 FlexCAN and eSCI Configuration

The BAM program configures FlexCAN_A and eSCI_A for reception. The CNRX_A and the RXD_A pads are configured as inputs to the FlexCAN and eSCI modules. The CNTX_A pad is configured as an output from the FlexCAN module. The TXD_A pad remains configured as GPIO input until a valid eSCI byte is received before a valid CAN message.

The FlexCAN controller is configured to operate at a baud rate equal to the system clock frequency divided by 40, using the standard 11-bit identifier format detailed in the CAN 2.0A specification. (See Table 32-6 for examples of baud rates.)

The BAM program ignores all possible errors that may happen during the serial communication. All received data is assumed to be good and is echoed on the CNTX signal.

The CAN controller bit timing is programmed with 10 time quantas and the sample point is two time quantas before the end (see Figure 32-5).



1 time quanta = 4 system clock periods

Figure 32-5. FlexCAN Bit Timing

The eSCI is configured for one start bit, eight data bits, no parity, and one stop bit. It operates at a baud rate equal to the system clock divided by 33. (See Table 32-6 for baud rate examples.)

The BAM program ignores eSCI errors. All received data is assumed to be good and is echoed out on the TXD_A signal.

Table 32-6. Serial Boot Mode—Baud Rates and Watchdog Summary

Crystal Frequency (MHz)	SCI Baud Rate (baud)	CAN Baud Rate (baud)	Watchdog Time-out period (seconds)
f_{extal}	$f_{\text{extal}} / 833$	$f_{\text{extal}} / 40$	$2^{17} / f_{\text{extal}}$
8	9600	200K	16.8
12	14400	300K	11.2
16	19200	400K	8.4
20	24000	500K	6.7
40	48000	1M	3.4

Upon reception of a valid CAN message with ID = 0x011 that contains 8 bytes of data, or an eSCI byte, the BAM program transitions to one of two serial boot sub-modes: CAN serial boot mode or eSCI serial boot mode.

In CAN serial boot mode, the eSCI_A RXD_A pad reverts to GPIO input. The ensuing download protocol is assumed to be all through the CAN. The eSCI is disabled.

If the eSCI byte is received first, the CAN_A controller is disabled and its pads reprogrammed to the GPIO. The eSCI TXD_A pad is reconfigured as an output.

Table 32-7. CAN/eSCI Reset Configuration for CAN/eSCI Pins in Serial Boot Mode

Pins	Reset Function	Initial Serial Boot Mode	Serial Boot Mode After a Valid CAN Message Received	Serial Boot Mode After a Valid eSCI Message Received
CNTX_A	GPIO	CNTX_A	CNTX_A	GPIO
CNRX_A	GPIO	CNRX_A	CNRX_A	GPIO
TXD_A	GPIO	GPIO	GPIO	TXD_A
RXD_A	GPIO	RXD_A	GPIO	RXD_A

Table 32-8. CAN/eSCI Reset Pin Configuration

Pins	I/O	Hysteresis	Driver Configuration
CNTX_A / TXD_A	Output	—	Push/Pull
CNRX_A / RXD_A	Input	Y	—

32.3.3.2.2 Serial Boot Mode Download Protocol

The download protocol follows four steps:

1. Host sends 64-bit password.
2. Host sends start address, size of download code in bytes, and VLE bit.
3. Host sends the application code data.
4. The MCU switches to the loaded code at the start address.

The communication is done in half-duplex manner, any transmission from host is followed by the MCU transmission. The host computer will not send data until it receives echo from the MCU. All multibyte data structures are sent most significant byte (MSB) first.

When the CAN is used for serial download, the data is packed into standard CAN messages in the following manner:

- A message with 0x0011 ID and 8-byte length is used to send the password. The MCU echoes with the same data, but 0x0001 ID.
- A message with 0x0012 ID and 8-byte length is used to send the start address, length, and the VLE mode bit. The MCU echoes with a message with 0x0002 ID.

- Messages with 0x0013 ID are used to send the downloaded data. The MCU echoes with 0x003 ID.

When the SCI is used for serial download, the data is sent byte-by-byte.

32.3.3.2.3 Serial Boot Mode Processing

The BAM program executes the serial boot as following:

1. Download 64-bit password.

The received 8-byte password is checked for validity. For a password to be valid, none of its four 16-bit halfwords must equal 0x0000 or 0xFFFF. Also, a password must have at least one 0 and one 1 in each halfword lane to be considered valid.

The BAM program then checks the censorship status of the MCU by checking the DISNEX bit in the SIU_CCR register. If Nexus is disabled, the MCU is considered to be censored and the password is compared with a password stored in the shadow row in internal flash memory.

If Nexus is enabled, the MCU is not considered to be censored, or the MCU is booting from external flash and the password is compared to the fixed value = 0xFEED_FACE_CAFE_BEEF.

If the password fails the validity test, the MCU stops responding to all stimuli. To get the MCU out of that state, the RESET signal must be asserted.

If the password is valid, the BAM refreshes the MCM software watchdog timer and performs the next step in the protocol.

2. Download start address, size of download, and VLE bit.

The next 8 bytes received by the MCU are considered to contain a 32-bit start address, the VLE mode bit, and a 31-bit code length (see [Figure 32-6](#)).

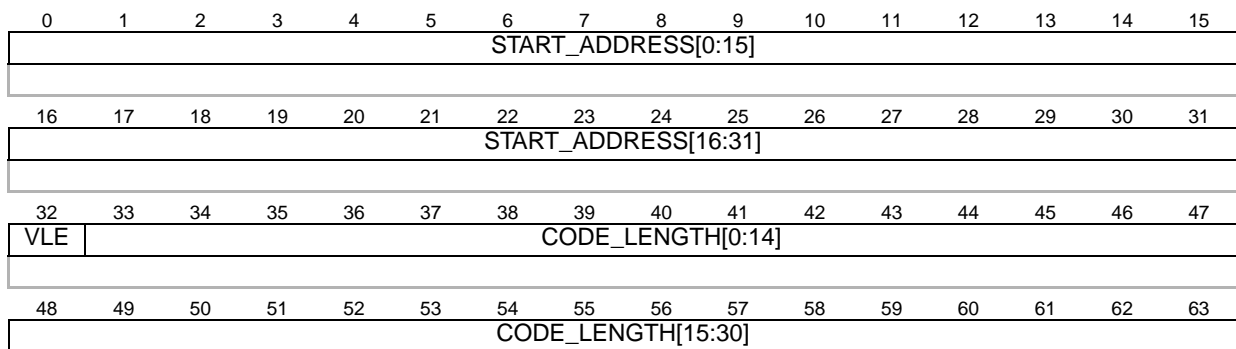


Figure 32-6. Start Address, VLE Bit and Download Size in Bytes

The start address defines where the received data will be stored and where the MCU will branch after the download is finished. The two least significant bits of the start address are ignored by the BAM program, such that the loaded code should be 32-bit word aligned.

The length defines how many data bytes to be loaded.

The VLE mode bit instructs the MCU to program MMU pages with VLE attribute. If it is 1, the downloaded code must be compiled to VLE instructions, if it is 0 the code contains classic Power Book E architecture instructions.

3. Download data.

Each byte of data received is stored in the MCU's memory, starting at the address specified in the previous protocol step, and incrementing through memory until the number of bytes of data received and stored in memory matches the number specified in the previous protocol step.

NOTE

In the MPC5510, the SRAM is protected by 32-bit wide error correction code (ECC), but other MPC55XX devices protect SRAM with 64-bit wide ECC. In the general case, this means any write to uninitialized SRAM must be 64 bits wide, otherwise an ECC error may occur. Therefore the BAM buffers downloaded data until 8 bytes have been received, and then does a single 64-bit wide write. Only system RAM supports 64-bit writes; therefore, attempting to download data to other RAM apart from system RAM will cause errors. If the start address of the downloaded data is not on an 8-byte boundary, the BAM will write 0x0000 to the memory locations from the preceding 8-byte boundary to the start address (maximum 4 bytes). The BAM will also write 0x0000 to all memory locations from the last byte of data downloaded to the following 8 byte boundary (maximum 7 bytes).

4. Switch to the loaded code.

The BAM program waits for the last echo message transmission to complete, then the active communication controller is disabled. Its pins revert to GPIO inputs. The BAM code passes control to the loaded code at start address, which was received in step 2 of the protocol.

NOTE

The code that is downloaded and executed must periodically refresh the MCM watchdog timer or change the timeout period to a value that will not cause resets during normal operation.

The serial download protocol is summarized in the [Table 32-9](#) and [Table 32-10](#)

Table 32-9. CAN Serial Boot Mode Download Protocol

Protocol Step	Host Sent Message	BAM Response Message	Action
1	CAN ID 0x0011 + 64-bit password	CAN ID 0x0001 + 64 bit password	Password checked for validity and compared against stored password. Platform watchdog timer is refreshed if the password check is successful.
2	CAN ID 0x0012 + 32-bit store address + VLE bit + 31-bit number of bytes	CAN ID 0x0002 + 32-bit store address + VLE bit + 31-bit number of bytes	Load address and size of download are stored for future use. The VLE bit determines whether the MMU entry for the SRAM, EBI, and flash is configured to run Book E or VLE code.

Table 32-9. CAN Serial Boot Mode Download Protocol

Protocol Step	Host Sent Message	BAM Response Message	Action
3	CAN ID 0x0013 + 8 to 64 bits of raw binary data	CAN ID 0x0003 + 8 to 64 bits of raw binary data	Each byte of data received is stored in MCU memory, starting at the address specified in the previous step and incrementing until the amount of data received and stored, matches the size as specified in the previous step.
4	None	None	The BAM returns IO pins to their reset state, disables FlexCAN_A module and then branches to the first address the data was stored to (As specified in step 2).

Table 32-10. eSCI Serial Boot Mode Download Protocol

Protocol Step	Host Sent Message	BAM Response Message	Action
1	64-bit password MSB first	64-bit password	Password checked for validity and compared against stored password. Platform Watchdog timer is refreshed if the password check is successful.
2	32-bit store address + VLE bit + 31-bit number of bytes (MSB first)	32-bit store address + VLE bit + 31-bit number of bytes	Load address and size of download are stored for future use. The VLE bit determines whether the MMU entry for the SRAM, EBI and Flash is configured to run Book E or VLE code.
3	8 bits of raw binary data	8 bits of raw binary data	Each byte of data received is store in MCU memory, starting at the address specified in the previous step and incrementing until the amount of data received and stored, matched the size as specified in the previous step.
4	None	None	The BAM returns IO pins to their reset state and disables the ESCI_A module. Then it branches to the first address the data was stored to (as specified in step 2).

Chapter 33

Media Local Bus (MLB)

33.1 Introduction

The Media Local Bus (MLB) is an inter-chip communication bus that gives external devices real-time access to the Media Oriented Systems Transport (MOST) bus. It has been designed to standardize a common hardware interface and common software API library, and allows transport of synchronous, asynchronous, isochronous and control data. The standardization allows an application (or multiple applications) to access the MOST network via a MOST transceiver. An MLB system consists of one MLB controller and one or several MLB devices. The MLB controller is typically an Intelligent Network Interface Chip, for example, the INIC/OS81050 (INIC), which functions as a transceiver for the MOST network and as a controller for the Media Local Bus.

The MLB interface consists of an MLB clock (generated by the MLB controller), MLB signal, and MLB data lines. There are two possible interface configurations: 3-pin and 5-pin. In the 3-pin interface, the MLBSIG and MLBDAT are bidirectional and the MLBCLK is unidirectional. In the 5-pin interface, all signals are unidirectional (MLBSI, MLBSO, MLBDI, MLBDO, and MLBCLK).

The MPC551xE/G implements a software¹ emulated MLB solution (SoftMLB) that is based on the e200Z0 core (IOP), but also uses system RAM, 2xDSPIs, and the eDMA module. In addition, SoftMLB Interface Logic has been integrated to detect the *FRAMESYNC* pattern and generate the appropriate triggers to the IOP, eDMA, and DSPIs. The MLB signals are routed out on Port E and Port F, which are on VDDE1 and VDDE3 power domains respectively. The MLB signals on Port E must be externally level shifted to be compatible with the 2.5 V MLB bus (as the VDDE1 domain must be 5 V to cater for other signals on this domain). The VDDE3 is a smaller domain that is shared with Nexus, JTAG, and some EBI signals.

33.1.1 Block Diagram

Figure 33-1 is a simplified block diagram of the SoftMLB concept and shows the functionality and interdependence on the other major blocks in the SoC.

1. A software driver for MLB emulation will be available from freescale. Available date - TBD

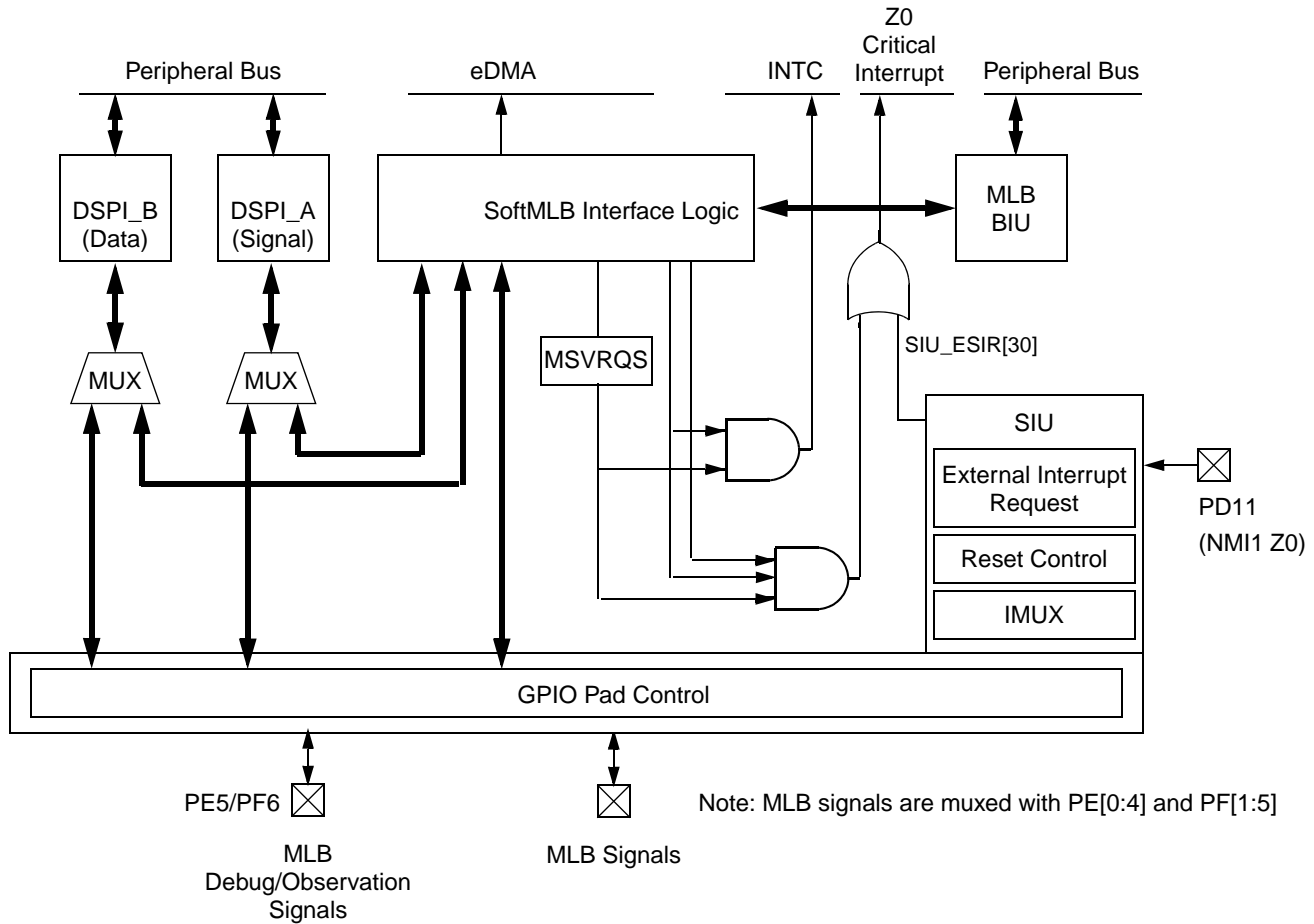


Figure 33-1. SoftMLB Block Diagram

33.1.2 Features

- 3-pin interface or 5-pin interface
- 256Fs operation
- External level shift control signals for 3-pin interface
- Multiple pin out options to increase flexibility
- MLBCLK clock adjust
- Visibility of debug signals

33.1.3 Modes of Operation

The SoftMLB Interface Logic has two modes of operation: Normal mode and Stop mode

33.1.3.1 Normal Mode

This mode is the main operational mode. The module transmits and receives MLB data, interfaces with DSPI_A and DSPI_B, and generates synchronization signals for other modules. In this mode, debug signals can be output to help in the setup. These signals are selected through the MLB Configuration Register (MLB_MCR) and the appropriate SIU_PCR[PA] field.

33.1.3.2 Stop Mode

This mode is the main low-power mode of the SoftMLB Interface Logic. It is enabled by setting the MDIS bit in the MLB_MCR register or by setting the HLT[9] bit in the SIU_HLT register. In this mode, the SoftMLB Interface Logic clocks are disabled, eliminating dynamic power consumption.

This module is power gated when the SoC enters SLEEP mode.

33.2 External Signal Description

The SoftMLB Interface Logic can be configured, via the MIFSEL bit in the MLB_MCR register, for a 3-pin interface (MIFSEL=0) or a 5-pin interface (MIFSEL=1). The state of this bit determines the signals that are multiplexed out by the SoftMLB Interface Logic. In 3-pin mode, the MLBCLK (input), MLBSIG (bidirectional), MLBDAT (bidirectional), MLBSIG_BUFEN (output), and MLBDAT_BUFEN (output) are available and can be routed to PE[0:4] or PF[1:5]. In 5-pin mode, MLBCLK (input), MLBSI (input), MLBDI (input), MLBSO (output), and MLBDO (output) are available and can be routed to PE[0:4] or PF[1:5], respectively. Furthermore, the clock adjust observation signals, MLB_SIGOBS, MLB_DATOBS, and the MLB_SLOT debug signal can be output and routed to PE5 or PF6. [Table 33-1](#) provides a summary of the external signals used for the SoftMLB interface.

Table 33-1. SoftMLB External Signal Summary

SoftMLB External Signal Summary			
Signal Name	Signal Description	Interface	Pin Muxing Options
MLBCLK	MLB Clock	5-pin & 3-pin	PE0, PF1
MLBSI MLBSIG	MLB Signal In MLB Bi-directional Signal	5-pin 3-pin	PE1, PF2
MLBDI MLBDAT	MLB Data In MLB Bi-directional Data	5-pin 3-pin	PE2, PF3
MLBSO MLBSIG_BUFEN	MLB Signal Out MLB Signal Level Shifter Enable	5-pin 3-pin	PE3, PF4
MLBDO MLBDAT_BUFEN	MLB Data Out MLB Data Level Shifter Enable	5-pin 3-pin	PE4, PF5
MLB_SLOT MLB_SIGOBS MLB_DATOBS	MLB Slot Debug MLB Clock Adjust Observe Signal Output MLB Clock Adjust Observe Data Output	5-pin & 3-pin 5-pin & 3-pin 5-pin & 3-pin	PE5, PF6

In each configuration, the MLB interface signals can be routed out in multiple pin positions on the MPC551xE/G. The selection is made via the associated SIU_PCR[PA] field for the particular pin. See [Table 2-1](#) for specific details.

NOTE

Port E and Port F are used to support the MLB signals. For timing reasons, the MLB signals should not be mixed between the two ports at the same time.

33.3 Memory Map and Registers

The SoftMLB Interface Logic memory map is shown in [Table 33-2](#). The address of each register is given as an offset to the MLB base address. Registers are listed in address order, identified by complete name and mnemonic, and show the type of access allowed. The memory map consists of a block of 64 address locations which are aliased within the 16K block reserved for the MLB starting at the MLB_BASE address.

Table 33-2. MLB Memory Map

Offset from MLB_BASE (0xFFF8_4000)	Register	Access	Reset Value	Section/Page
General Registers				
0x00	MLB_MCR – Module Configuration Register	R/W	0x8000_0000	33.3.1.1/33-5
0x04	MLB_MBR – MLB Blank Register	R/W	0x0000_0000	33.3.1.2/33-7
0x08	MLB_MSR – Module Status Register	R	0x0000_0000	33.3.1.3/33-8
0x0C	MLB_RXCCHAR – RX Control Channel Address Register	R/W	0x0000_0000	33.3.1.4/33-9
0x10	MLB_RXACHAR – RX Async Channel Address Register	R/W	0x0000_0000	33.3.1.5/33-10
0x14	MLB_TXCCHAR – TX Control Channel Address Register	R/W	0x0000_0000	33.3.1.6/33-11
0x18	MLB_TXACHAR – TX Async Channel Address Register	R/W	0x0000_0000	33.3.1.7/33-12
0x1C	MLB_TXSCHAR – TX Sync Channel Address Register	R/W	0x0000_0000	33.3.1.8/33-13
0x20	MLB_TXSCHAMR – TX Sync Channel Address Mask Register	R/W	0x0000_003E	33.3.1.9/33-14
0x24	MLB_CLKACR – MLBCLK Clock Adjust Control Register	R/W	0x0000_0000	33.3.1.10/33-15
0x28	MLB_RXICCHAR – RX Isochronous Channel Address Register	R/W	0x0000_0000	33.3.1.11/33-16
0x2C	MLB_TXICCHAR – TX Isochronous Channel Address Register	R/W	0x0000_0000	33.3.1.12/33-17

Note: Unimplemented locations always read 0. Writes to unimplemented locations have no effect.

33.3.1 Register Descriptions

This section lists the registers and bits that are used to control the SoftMLB Interface Logic.

33.3.1.1 MLB Module Configuration Register (MLB_MCR)

The MLB_MCR contains bits that configure the SoftMLB Interface Logic.

Offset MLB_BASE+0x0000

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MDATOBSE	MSIGOBSE	MSLOTE	0	0	MSVRQIE	MDATRQE	0	0	MSVRQDL			MSVRQCIE	MIFSEL	MSBFEPOL	MDBFEPOL
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-2. MLB Module Configuration Register (MLB_MCR)

Table 33-3. MLB Module Configuration (MLB_MCR) Register Field Descriptions

Field	Description
MDIS	Module Disable. Controls whether the SoftMLB Interface Logic is enabled or not. When MDIS is set, the SoftMLB Interface logic is asynchronously held in reset and disabled. When disabled, clocks are stopped to the non-memory mapped logic. Register reads and writes are still accessible. The DSPI_SS is driven high (DSPI are de-selected). The SoftMLB Interface Logic also masks out all received bits on DSPI_S_OUT and DSPI_D_OUT. Once this signal is cleared, the SoftMLB Interface Logic requires some time to synchronize to the MLB Bus. MLB_MSR[MSYSS] = 1 indicates that the logic has synchronized. 0 Enable the SoftMLB Interface Logic 1 Disable the SoftMLB Interface Logic – Default out of reset
bits 1–15	Reserved.

Table 33-3. MLB Module Configuration (MLB_MCR) Register Field Descriptions (continued)

Field	Description																																				
MDATOBSE MSIGOBSE MSLOTE	<p>MLB Observation Output Enables and MLB Slot Debug Output Enable. These bits control the output of the SoftMLB Interface Logic to be the MLB_SLOT signal, or the MLB_SIGOBS or MLB_DATOBS signals (output of the DSPI_A and DSPI_B input latch is routed to PTE5 or PTF6 for use in the clock adjust scheme). The correct SIU_PCR[PA] selection must be chosen to route the desired signal to either PTE5 or PTF6. The table below summarizes the multiplexing between MLB_SLOT, MLB_SIGOBS and MLB_DATOBS.</p> <table border="1"> <thead> <tr> <th>MSLOTE</th> <th>MDATOBSE</th> <th>MSIGOBSE</th> <th>SoftMLB Interface Logic Output</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>Disabled</td> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> <td>MLB_SIGOBS</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>MLB_DATOBS</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> <td>Reserved (defaults to MLB_SIGOBS)</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>MLB_SLOT</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> <td>MLB_SLOT</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> <td>MLB_SLOT</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>MLB_SLOT</td> </tr> </tbody> </table>	MSLOTE	MDATOBSE	MSIGOBSE	SoftMLB Interface Logic Output	0	0	0	Disabled	0	0	1	MLB_SIGOBS	0	1	0	MLB_DATOBS	0	1	1	Reserved (defaults to MLB_SIGOBS)	1	0	0	MLB_SLOT	1	0	1	MLB_SLOT	1	1	0	MLB_SLOT	1	1	1	MLB_SLOT
MSLOTE	MDATOBSE	MSIGOBSE	SoftMLB Interface Logic Output																																		
0	0	0	Disabled																																		
0	0	1	MLB_SIGOBS																																		
0	1	0	MLB_DATOBS																																		
0	1	1	Reserved (defaults to MLB_SIGOBS)																																		
1	0	0	MLB_SLOT																																		
1	0	1	MLB_SLOT																																		
1	1	0	MLB_SLOT																																		
1	1	1	MLB_SLOT																																		
bits 19–20	Reserved.																																				
MSVRQIE	<p>MLB Service Request Interrupt Enable. This bit is used to enable interrupt requests (SRV_REQ) to the IOP via the interrupt controller. The SRV_REQ is the MSVRQS flag gated with MSVRQIE. MSVRQS is the reflection of a free toggle (SRV_REQ_GLUE) and is self clearing. If the ISR does not respond fast enough the interrupt request will be lost. Likewise if the ISR is too fast the request can still be pending and should not be re-enabled until the MSVRQS is clear. The interrupt request can also directly request a critical interrupt to the IOP. See the description of the MSVRQCIE bit below for details.</p> <p>Note: The MSYSS needs to be set before an SRV_REQ_GLUE signal will be generated.</p> <p>0 Interrupt request disabled (default out of reset) 1 Interrupt request enabled and generated every 32 MLBCLK cycles</p>																																				
MDATRQE	<p>MLB DATA eDMA Request Enable. This bit is used to enable the SoftMLB Interface Logic to generate eDMA requests. The eDMA request is a free running toggle. The eDMA controller moves the data, then the SoftMLB Interface request to the eDMA is cleared until the next eDMA request. This is independent of the MDATRQS which is a reflection of the free running toggle.</p> <p>Note: The MSYSS needs to be set before an eDMA request shall be generated.</p> <p>Note: In order to generate an eDMA request the eDMA channel mux needs to enable an eDMA channel and associate the MLB_DMA_REQ(0x35).</p> <p>0 SoftMLB Interface Logic does not trigger an eDMA request (default out of reset) 1 SoftMLB Interface Logic does trigger an eDMA request every MLBDATA word (32 MLBCLK cycles)</p>																																				
bits 23–24	Reserved.																																				

Table 33-3. MLB Module Configuration (MLB_MCR) Register Field Descriptions (continued)

Field	Description
MSVRQDL	<p>MLB Service Request Delay in MLBCLK cycles. These control bits are used to control the number of cycles that the SRV_REQ_GLUE is driven before the real 32-bit MLB word boundary. MSVRQDL should only be updated when MDIS is set.</p> <p>000 0 cycles (default out of reset) 001 1 cycle 010 2 cycles 011 3 cycles 100 4 cycles 101 5 cycles 110 Reserved 111 Reserved</p>
MSVRQCIE	<p>MLB Service Request Critical Interrupt Enable. This bit is used to enable critical interrupt requests (SRV_REQ_CI) to the IOP. The SRV_REQ_CI is ORed with the IOP NMI interrupt to create the IOP critical interrupt input. The SRV_REQ_CI is the MSVRQS flag gated with MSVRQCIE and MSVRQIE. MSVRQS is the reflection of a free toggle (SRV_REQ_GLUE) and is self clearing.</p> <p>MSVRQCIE is a write once bit and thus once written, MSVRQCIE will hold its value until the next reset.</p> <p>Note: The MSYSS needs to be set before an SRV_REQ_CI will be generated.</p> <p>0 Critical interrupt request disabled (default out of reset) 1 Critical interrupt request enabled</p>
MIFSEL	<p>MLB Interface Select. This bit selects between the 3-pin and 5-pin MLB interfaces. Depending on its value, the SIU_PCR[PA] field routes different signals to the pins. In the 3-pin interface this field allows the MLBDAT, MLBSIG, MLBCLK, MLBSIG_BUFEN, and MLBDAT_BUFEN signals to be routed to the corresponding pins. In the 5-pin interface the MLBSI, MLBSO, MLBDI, MLBDO, and MLBCLK signals are routed to the pins.</p> <p>0 3-pin interface selected (default out of reset) 1 5-pin interface selected</p>
MSBFEPOL	<p>MLBSIG_BUFEN Polarity Select. This bit selects the active polarity for the MLBSIG_BUFEN. If the default condition (MSBFEPOL=0) is selected, the MLBSIG_BUFEN is driven high when active and low when inactive. If MSBFEPOL=1, the signal is active low.</p> <p>0 Active High (default out of reset) 1 Active Low</p>
MDBFEPOL	<p>MLBDAT_BUFEN Polarity Select. This bit selects the active polarity for the MLBDAT_BUFEN. If the default condition (MDBFEPOL=0) is selected, the MLBDAT_BUFEN is driven high when active and low when inactive. If MDBFEPOL=1, the signal is active low.</p> <p>0 Active High (default out of reset) 1 Active Low</p>

33.3.1.2 MLB Blank Register (MLB_MBR)

The MLB_MBR register contains the blank request bit to cancel data that has been queued in the DSPI FIFO.

Offset MLB_BASE+0x0004

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	BLANK
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-3. MLB Blank Register (MLB_MBR)

Table 33-4. MLB Blank (MLB_MBR) Register Field Descriptions

Field	Description
bit 0–30	Reserved.
BLANK	<p>MLB Blank Request. This bit is used to cancel data that has been previously queued in the DSPI FIFO due to change in the receiver status. (for example, RxBusy)</p> <p>To save IOP cycles required to clear this bit again, the SoftMLB Interface Logic automatically clears this bit, after the MLB_GATE signal has been generated internally. The SoftMLB Interface Logic asserts the MLB_GATE signal at the start of the next MLB quadlet and resets the BLANK signal. The self clearing function is only active when MDIS is cleared. The BLANK bit is cleared when MDIS is set. Due to clock synchronization between the system clock and MLBCLK, it may take up to two MLBCLK clocks before BLANK is updated in the MLBCLK domain.</p> <p>0 MLB Blank not requested (default out of Reset) 1 MLB Blank requested</p>

33.3.1.3 MLB Module Status Register (MLB_MSR)

The MLB_MSR contains the status bits that are used to determine detection of the system channel and status flags for service and eDMA requests.

Offset MLB_BASE+0x0008

Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	MDATRQS	MSYSS	MSVRQS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-4. MLB Module Status Register (MLB_MSR)

Table 33-5. MLB Module Status (MLB_MSR) Register Field Descriptions

Field	Description
bits 0–28	Reserved.
MDATRQS	MLB Data eDMA Request Status. This bit indicates the state of the internal eDMA request signal. It is free running and high for 16 bits of the MLBCLK signal then low for the following 16 bits. MDATRQS is cleared when MDIS is set. MDATRQS will only be set after MSYSS is set. 0 eDMA request is not asserted 1 eDMA request is asserted
MSYSS	MLB System Stable. This signal is asserted by the hardware after it detects 256 occurrences of the <i>FRAMESYNC</i> pattern (0x1FE). It can be cleared only by setting MDIS. 0 System is not stable 1 System is stable
MSVRQS	MLB Service Request Status. This bit indicates the state of the internal SRV_REQ_GLUE signal. It is high for 16 bits of the MLBCLK signal then low for the following 16 bits. SRV_REQ is MSVRQS gated with MSVRQIE. Likewise, SRV_REQ_CI is MSVRQS gated with MSVRQCIE and MSVRQIE. MSVRQS is cleared when MDIS is set. MSVRQS will only be set after MSYSS is set. 0 Service request not active 1 Service request active

33.3.1.4 RX Control Channel Address Register (MLB_RXCCHAR)

The MLB_RXCCHAR contains the RX Control Channel Address for this device.

Offset MLB_BASE+0x000C

Access: Read/Write

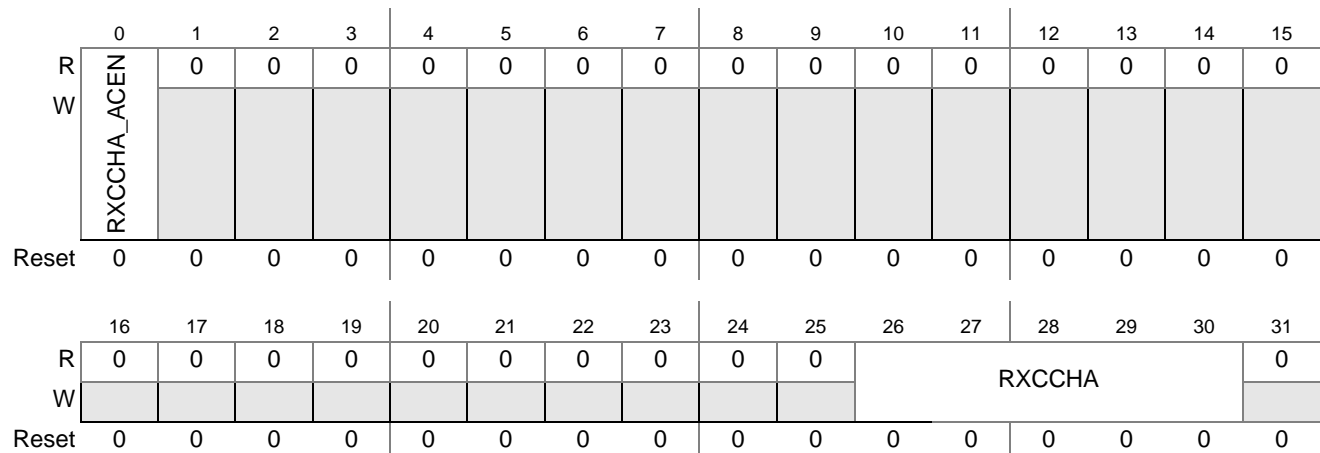


Figure 33-5. MLB RX Control Channel Address Register (MLB_RXCCHAR)

Table 33-6. MLB RX Control Channel Address Register (MLB_RXCCHAR) Field Descriptions

Field	Description
RXCCHA_ACEN	RX Control Channel Address Comparison Enable. When enabled, a received Channel Address is compared against the RX Control Channel Address configured in this register. RXCCHA_ACEN should only be updated when MDIS is set. 0 RX Control Channel Address comparison disabled (default out of reset) 1 RX Control Channel Address comparison enabled
bits 1–25	Reserved.
RXCCHA	RX Control Channel Address. If RXCCHA_ACEN=1, this address will be compared against the logical address that was driven on the bus by the MLB controller (INIC). If the received channel address matches the programmed value in the MLB_RXCCHAR register the appropriate output buffer enables are driven (Section 33.4.2.1.4, “MLBSIG_BUFEN and MLBDAT_BUFEN”). Although Channel Addresses are defined to be sixteen bits wide, bits 15 through 9 and the LSB are always zero. The odd addresses are reserved and Channel Address 0x0000 is the bus idle state. Only the 31 even addresses between 0x0002 and 0x003E are allowed; therefore, only five bits per Channel Address are required to be configured. RXCCHA should only be updated when MDIS is set. An address match occurs when RXCCHA_ACEN is set and the received 16 bit Channel Address equals 16b0000_0000_00_{RXCCHA}_0.
bit 31	Reserved.

33.3.1.5 RX Async Channel Address Register (MLB_RXACHAR)

The MLB_RXACHAR contain the RX Async Channel Address for this device.

Offset MLB_BASE+0x0010

Access: Read/Write

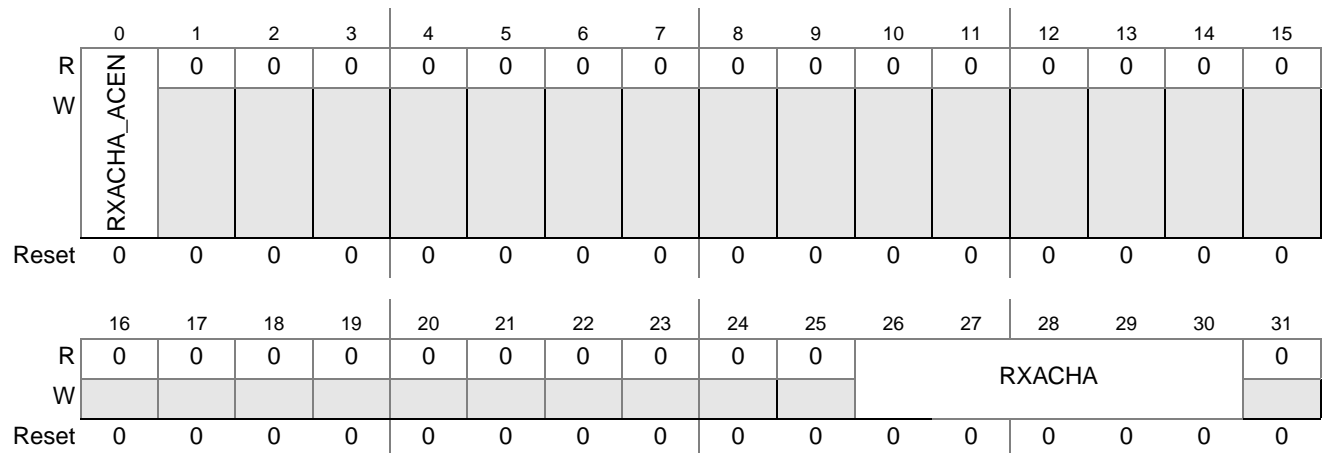


Figure 33-6. MLB RX Async Channel Address Register (MLB_RXACHAR)

Table 33-7. MLB RX Async Channel Address Register (MLB_RXACHAR) Field Descriptions

Field	Description
RXACHA_ACEN	RX Async Channel Address Comparison Enable. When enabled a received Channel Address is compared against the RX Async Channel Address configured in this register. RXACHA_ACEN should only be updated when MDIS is set. 0 RX Async Channel Address comparison disabled (default out of reset) 1 RX Async Channel Address comparison enabled
bits 1–25	Reserved.
RXACHA	RX Async Channel Address. If RXACHA_ACEN=1, this address will be compared against the logical address that was driven on the bus by the MLB controller (INIC). If the received channel address matches the programmed value in the MLB_RXACHAR register the appropriate output buffer enables are driven (Section 33.4.2.1.4, “MLBSIG_BUFEN and MLBDAT_BUFEN”). Although Channel Addresses are defined to be sixteen bits wide, bits 15 through 9 and the LSB are always zero. The odd addresses are reserved and Channel Address 0x0000 is the bus idle state. Only the 31 even addresses between 0x0002 and 0x003E are allowed; therefore, only five bits per Channel Address are required to be configured. RXACHA should only be updated when MDIS is set. An address match occurs when RXACHA_ACEN is set and the received 16 bit Channel Address equals 16b0000_0000_00_{RXACHA}_0.
bit 31	Reserved.

33.3.1.6 TX Control Channel Address Register (MLB_TXCCHAR)

The MLB_TXCCHAR contains the TX Control Channel Address for this device.

Offset MLB_BASE+0x0014

Access: Read/Write

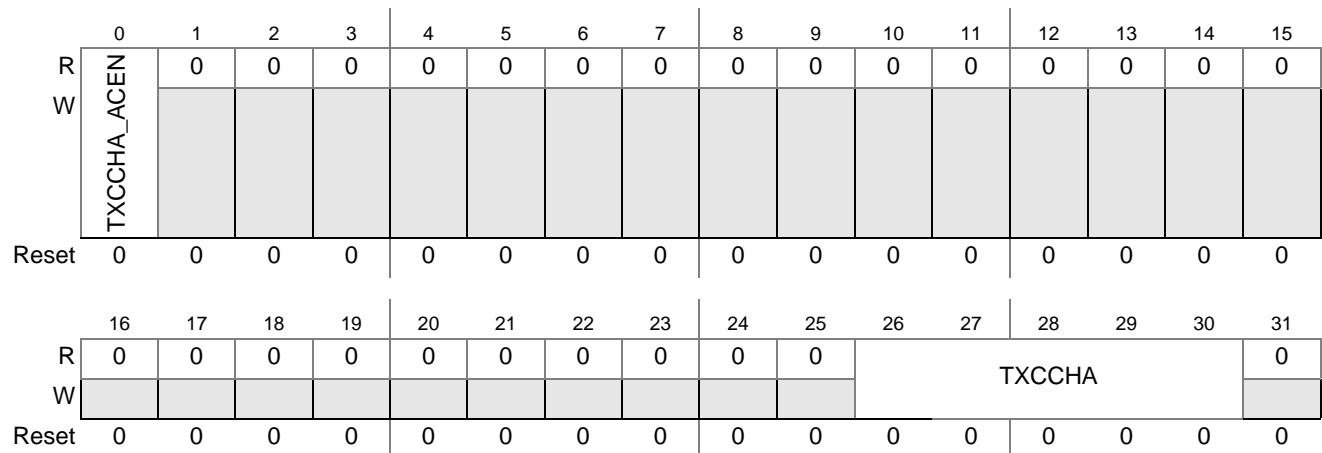


Figure 33-7. MLB TX Control Channel Address Register (MLB_TXCCHAR)

Table 33-8. MLB TX Control Channel Address Register (MLB_TXCCHAR) Field Descriptions

Field	Description
TXCCHA_ACEN	TX Control Channel Address Comparison Enable. When enabled, a received Channel Address is compared against the TX Control Channel Address configured in this register. TXCCHA_ACEN should only be updated when MDIS is set. 0 TX Control Channel Address comparison disabled (default out of reset) 1 TX Control Channel Address comparison enabled
bits 1–25	Reserved.
TXCCHA	TX Control Channel Address. If TXCCHA_ACEN=1, this address will be compared against the logical address that was driven on the bus by the MLB controller (INIC). If the received channel address matches the programmed value in the MLB_TXCCHAR register the appropriate output buffer enables are driven (Section 33.4.2.1.4, “MLBSIG_BUFEN and MLBDAT_BUFEN”). Although Channel Addresses are defined to be sixteen bits wide, bits 15 through 9 and the LSB are always zero. The odd addresses are reserved and Channel Address 0x0000 is the bus idle state. Only the 31 even addresses between 0x0002 and 0x003E are allowed; therefore, only five bits per Channel Address are required to be configured. TXCCHA should only be updated when MDIS is set. An address match occurs when TXCCHA_ACEN is set and the received 16 bit Channel Address equals 16b0000_0000_00_{TXCCHA}_0.
bit 31	Reserved.

33.3.1.7 TX Async Channel Address Register (MLB_TXACHAR)

The MLB_TXACHAR contain the TX Async Channel Address for this device.

Offset MLB_BASE+0x0018

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	TXACHA					0
W											TXACHA					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-8. MLB TX Async Channel Address Register (MLB_TXACHAR)

Figure 33-9. MLB TX Async Channel Address Register (MLB_TXACHAR) Field Descriptions

Field	Description
TXACHA_ACEN	TX Async Channel Address Comparison Enable. When enabled a received Channel Address is compared against the TX Async Channel Address configured in this register. TXACHA_ACEN should only be updated when MDIS is set. 0 TX Async Channel Address comparison disabled (default out of reset) 1 TX Async Channel Address comparison enabled
bits 1–25	Reserved.
TXACHA	TX Async Channel Address. If TXACHA_ACEN=1, this address will be compared against the logical address that was driven on the bus by the MLB controller (INIC). If the received channel address matches the programmed value in the MLB_TXACHAR register the appropriate output buffer enables are driven (Section 33.4.2.1.4, “MLBSIG_BUFEN and MLBDAT_BUFEN”). Although Channel Addresses are defined to be sixteen bits wide, bits 15 through 9 and the LSB are always zero. The odd addresses are reserved and Channel Address 0x0000 is the bus idle state. Only the 31 even addresses between 0x0002 and 0x003E are allowed; therefore, only five bits per Channel Address are required to be configured. TXACHA should only be updated when MDIS is set. An address match occurs when TXACHA_ACEN is set and the received 16 bit Channel Address equals 16b0000_0000_00_{TXACHA}_0.
bit 31	Reserved.

33.3.1.8 TX Sync Channel Address Register (MLB_TXSCHAR)

The MLB_TXSCHAR contains the TX Sync Channel Address for this device.

Offset MLB_BASE+0x001C

Access: Read/Write

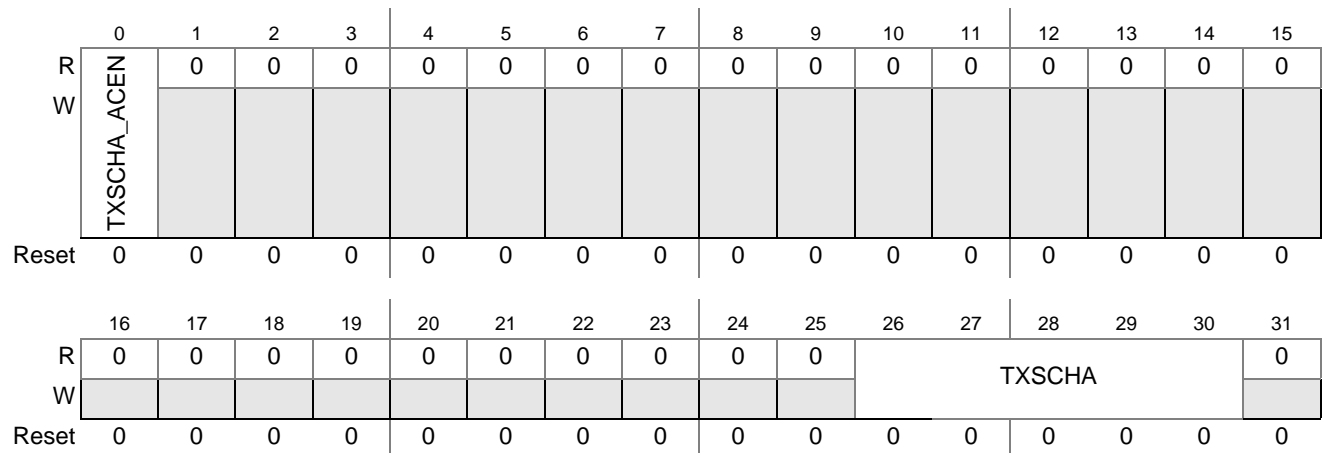


Figure 33-10. MLB TX Sync Channel Address Register (MLB_TXSCHAR)

Table 33-9. MLB TX Sync Channel Address Register (MLB_TXSCHAR) Field Descriptions

Field	Description
TXSCHA_ACEN	TX Sync Channel Address Comparison Enable. When enabled a received Channel Address is compared against the TX Sync Channel Address configured in this register. TXSCHA_ACEN should only be updated when MDIS is set. 0 TX Sync Channel Address comparison disabled (default out of reset) 1 TX Sync Channel Address comparison enabled
bits 1–25	Reserved.
TXSCHA	TX Sync Channel Address. If TXSCHA_ACEN=1, this address will be compared against the logical address that was driven on the bus by the MLB controller (INIC). If the received channel address matches the programmed value in the MLB_TXSCHAR register the appropriate output buffer enables are driven (Section 33.4.2.1.4, “MLBSIG_BUFEN and MLBDAT_BUFEN”). Although Channel Addresses are defined to be sixteen bits wide, bits 15 through 9 and the LSB are always zero. The odd addresses are reserved and Channel Address 0x0000 is the bus idle state. Only the 31 even addresses between 0x0002 and 0x003E are allowed; therefore, only five bits per Channel Address are required to be configured. TXSCHA should only be updated when MDIS is set. An address match occurs when TXSCHA_ACEN is set and the received 16 bit Channel Address equals 16b0000_0000_00_{TXSCHA}_0.
bit 31	Reserved.

33.3.1.9 TX Sync Channel Address Mask Register (MLB_TXSCHAMR)

The MLB_TXSCHAMR contains the TX Sync Channel Address Mask for this device.

Offset MLB_BASE+0x0020

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	TXSCHAM				0	
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0

Figure 33-11. MLB TX Sync Channel Address Mask Register (MLB_TXSCHAMR)

Table 33-10. MLB TX Sync Channel Address Register (MLB_TXSCHAMR) Field Descriptions

Field	Description
bits 0–25	Reserved.
TXSCHAM	TX Sync Channel Address Mask Register. These user configuration bits are used to define bit-wise masking on the TX Sync Channel address that will allow the device to recognize multiple TX Channel Sync Addresses. If the mask is cleared, the corresponding bit in the address is ignored. If the mask is set the corresponding bit in the received address must match for a valid comparison. TXSCHAM should only be updated when MDIS is set. 0 Ignore corresponding bit in the address (filter open) 1 Compare corresponding bit in the address (All 1s – default out of Reset – match on single address in the MLB_TXSCHAMR register.
bit 31	Reserved.

33.3.1.10 MLBCLK Clock Adjust Control Register (MLB_CLKACR)

The MLB_CLKACR contains bits that are used to control the delay of the DSPI_DS_CLK relative to the MLBCLK input clock.

Offset MLB_BASE+0x0024

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PDLY															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-12. MLBCLK Clock Adjust Control Register (MLB_CLKACR)

Table 33-11. MLBCLK Clock Adjust Control (MLB_CLKACR) Register Field Descriptions

Field	Description
bits 0–15	Reserved.
PDLY	<p>Programmable Delay. The PDLY value determines the programmable delay added to the incoming MLBCLK before being used to capture data. All incoming MLB data is captured on the rising edge of the delayed MLBCLK. Each PDLY bit enables one of sixteen equal delay stages which are concatenated together.</p> <p>0000_0000_0000_0000 All delay stages disabled (bypass) 0000_0000_0000_0001 1 delay stages enabled 0000_0000_0000_0011 2 delay stages enabled 0000_0000_0000_0111 3 delay stages enabled 0000_0000_0000_1111 4 delay stages enabled 0000_0000_0001_1111 5 delay stages enabled 0000_0000_0011_1111 6 delay stages enabled 0000_0000_0111_1111 7 delay stages enabled 0000_0000_1111_1111 8 delay stages enabled 0000_0001_1111_1111 9 delay stages enabled 0000_0011_1111_1111 10 delay stages enabled 0000_0111_1111_1111 11 delay stages enabled 0000_1111_1111_1111 12 delay stages enabled 0001_1111_1111_1111 13 delay stages enabled 0011_1111_1111_1111 14 delay stages enabled 0111_1111_1111_1111 15 delay stages enabled 1111_1111_1111_1111 16 delay stages enabled Other Reserved</p>

33.3.1.11 RX Isochronous Channel Address Register (MLB_RXICHA)

The MLB_RXICHA contains the RX Isochronous Channel Address for this device.

Offset MLB_BASE+0x0028

Access: Read/Write

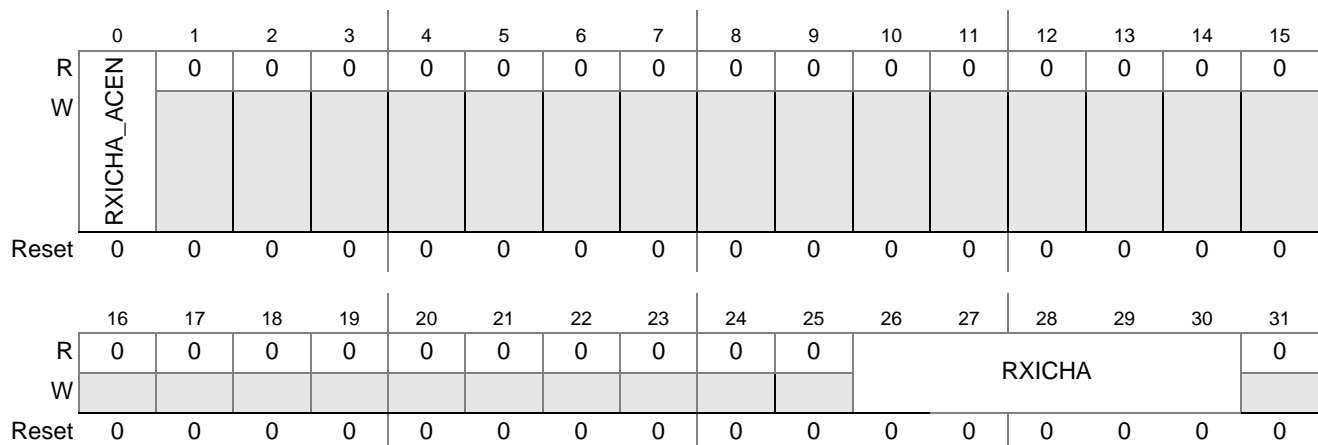


Figure 33-13. MLB RX Isochronous Channel Address Register (MLB_RXICHA)

Table 33-12. MLB RX Isochronous Channel Address Register (MLB_RXICHA) Field Descriptions

Field	Description
RXICHA_ACEN	RX Isochronous Channel Address Comparison Enable. When enabled a received Channel Address is compared against the RX Isochronous Channel Address configured in this register. RXICHA_ACEN should only be updated when MDIS is set. 0 RX Isochronous Channel Address comparison disabled (default out of reset) 1 RX Isochronous Channel Address comparison enabled
bits 1–25	Reserved.
RXICHA	RX Isochronous Channel Address. If RXICHA_ACEN=1, this address will be compared against the logical address that was driven on the bus by the MLB controller (INIC). If the received channel address matches the programmed value in the MLB_RXICHA register the appropriate output buffer enables are driven (Section 33.4.2.1.4, “MLBSIG_BUFEN and MLBDAT_BUFEN”). Although Channel Addresses are defined to be sixteen bits wide, bits 15 through 9 and the LSB are always zero. The odd addresses are reserved and Channel Address 0x0000 is the bus idle state. Only the 31 even addresses between 0x0002 and 0x003E are allowed; therefore, only five bits per Channel Address are required to be configured. RXICHA should only be updated when MDIS is set. An address match occurs when RXICHA_ACEN is set and the received 16-bit Channel Address equals 16b0000_0000_00_{RXICHA}_0.
bit 31	Reserved.

33.3.1.12 TX Isochronous Channel Address Register (MLB_TXICHA)

The MLB_TXICHA contain the TX Isochronous Channel Address for this device.

Offset MLB_BASE+0x0028

Access: Read/Write

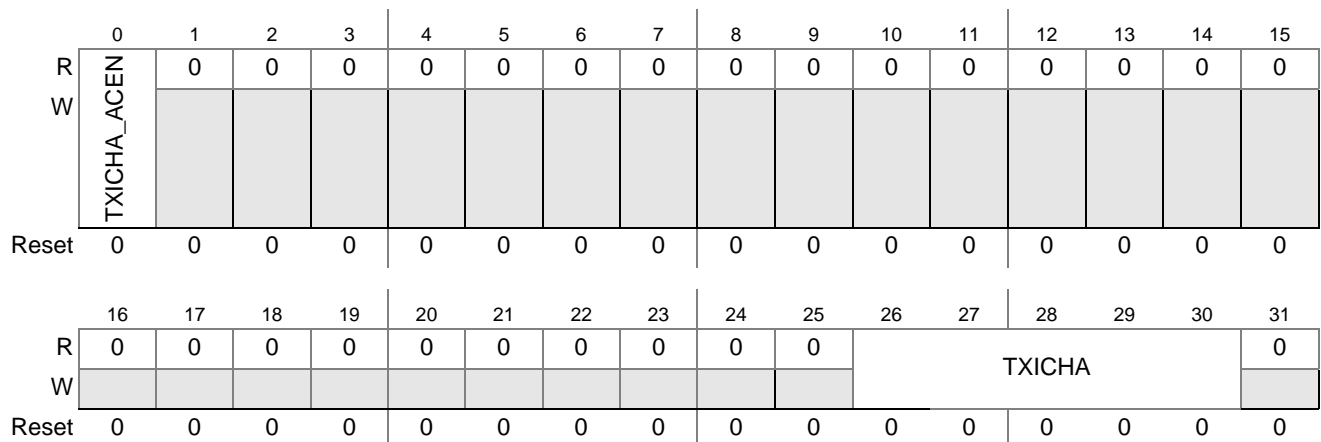


Figure 33-14. MLB TX Isochronous Channel Address Register (MLB_TXICHA)

Table 33-13. MLB TX Isochronous Channel Address Register (MLB_TXICHAR) Field Descriptions

Field	Description
TXICHA_ACEN	TX Isochronous Channel Address Comparison Enable. When enabled a received Channel Address is compared against the TX Isochronous Channel Address configured in this register. TXICHA_ACEN should only be updated when MDIS is set. 0 TX Isochronous Channel Address comparison disabled (default out of reset) 1 TX Isochronous Channel Address comparison enabled
bits 1–25	Reserved.
TXICHA	TX Isochronous Channel Address. If TXICHA_ACEN=1, this address will be compared against the logical address that was driven on the bus by the MLB controller (INIC). If the received channel address matches the programmed value in the MLB_TXICHAR register the appropriate output buffer enables are driven (Section 33.4.2.1.4, “MLBSIG_BUFEN and MLBDAT_BUFEN”). Although Channel Addresses are defined to be sixteen bits wide, bits 15 through 9 and the LSB are always zero. The odd addresses are reserved and Channel Address 0x0000 is the bus idle state. Only the 31 even addresses between 0x0002 and 0x003E are allowed; therefore, only five bits per Channel Address are required to be configured. TXICHA should only be updated when MDIS is set. An address match occurs when TXICHA_ACEN is set and the received 16-bit Channel Address equals 16b0000_0000_00_{RXICHA}_0.
bit 31	Reserved.

33.4 Functional Description

The MLB topology supports communications between a single MLB controller (INIC) and one or several MLB devices (for example, MPC551xE/G). The 3-pin interface consists of an MLBCLK clock (generated by the MLB controller), an MLBSIG signal, and MLBDAT data lines. In this interface, MLBSIG and MLBDAT are bidirectional. In the 5-pin interface, all signals are unidirectional (MLBSI, MLBSO, MLBDI, MLBDO, and MLBCLK). The implementation on the MPC551xE/G also includes Level Shifter Enable control signals (MLBSIG_BUFEN and MLBDAT_BUFEN) for the 3-pin interface, and setup and debug signals (MLB_SIGOBS, MLB_DATOBS, and MLB_SLOT). See [Section 33.2, “External Signal Description”](#) description for a full details.

The MLB controller is the interface between the MLB devices and the MOST network. [Figure 33-15](#) shows the MLB topology with the MPC551xE/G as the MLB device and the INIC as the MLB controller. The 5-pin interface is shown.

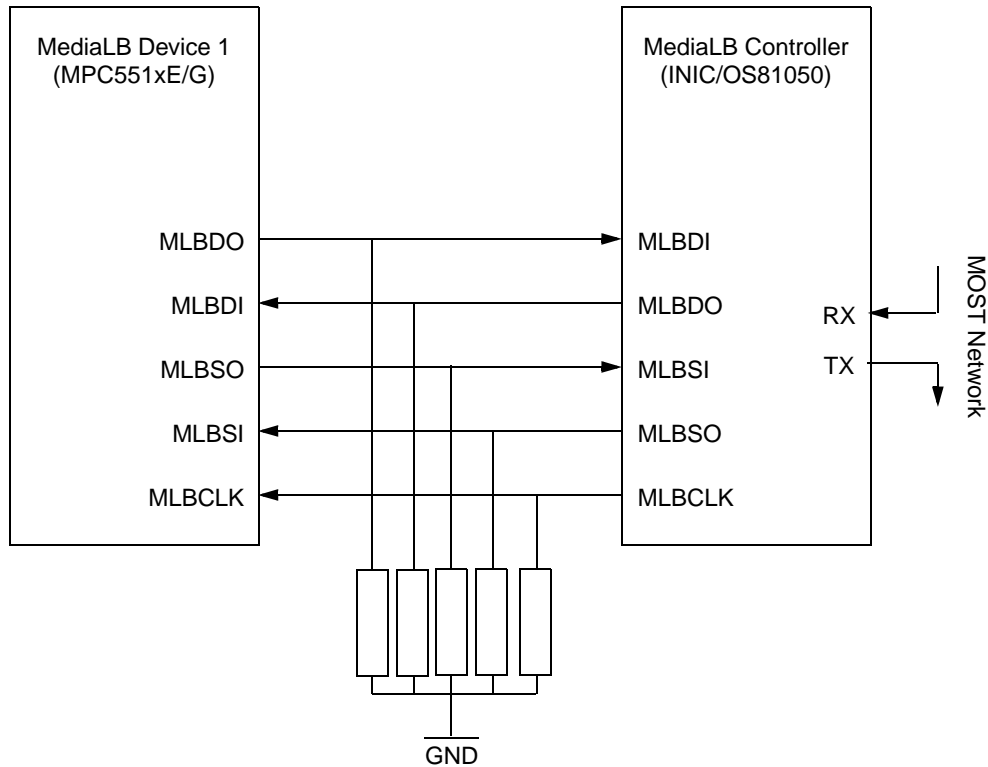


Figure 33-15. 5-pin MLB Topology

The INIC generates the MLBCLK clock, which is synchronized to the MOST network and provides the timing for the entire MOST interface.

The MLBSIG (or MLBSO) carries the Channel Address generated by the MLB controller (INIC). The Channel Address indicates which MLB device is transmitting and which MLB device is receiving in the following physical channel. The transmitting MLB device outputs a Command Byte on its MLBSIG (or MLBSO) and the receiving MLB device outputs an RxStatus byte in the quadlet following the Command Byte.

The MLBDAT (or MLBDO) line is driven by the transmitting MLB device and is received by all other devices, including the controller. This line carries the actual data (synchronous, asynchronous, isochronous, control).

The IOP based softMLB solution uses the following resources to emulate an MLB device.

- e200z0 (IOP)
- DSPI_A and DSPI_B
- eDMA (four channels)
- SoftMLB Interface Logic
- Pin multiplexing options within the SIU
- System RAM

The DSPIs are used in slave mode and clocked continuously by a delayed version of the MLBCLK. DSPI_A and DSPI_B are used to implement the signal and data channels, respectively. Frame synchronization is achieved by the slave select signal generated by the SoftMLB Interface Logic, which is derived from the *FRAMESYNC* signal on the MLBSIG (or MLBSO) line.

The SoftMLB emulation uses four eDMA channels in the soft MLB concept. One channel is used to transfer the MLBSIG values into the internal RAM; this is triggered by DSPI_A drain flag. The second channel is used to transfer the MLBDAT values into the internal RAM, triggered by the SoftMLB Interface Logic (MLB_DMA_REQ) and aligned with every MLB word. The third and fourth channels are used to transfer the data from internal SRAM to DSPI_A and DSPI_B respectively, and are triggered by the IOP, which in turn is triggered by the SoftMLB Interface Logic (SRV_REQ) every 32 MLBCLK cycles. The source of the IOP interrupt can be selected at initialization to be from the interrupt controller, source 293, or the IOP critical interrupt. The IOP also implements the low level MLB protocol, processes the data, and transmits the command and RXStatus bytes (depending on whether it is a transmitter or receiver) from the MLB device. [Figure 33-16](#) shows the data flow of the soft MLB solution.

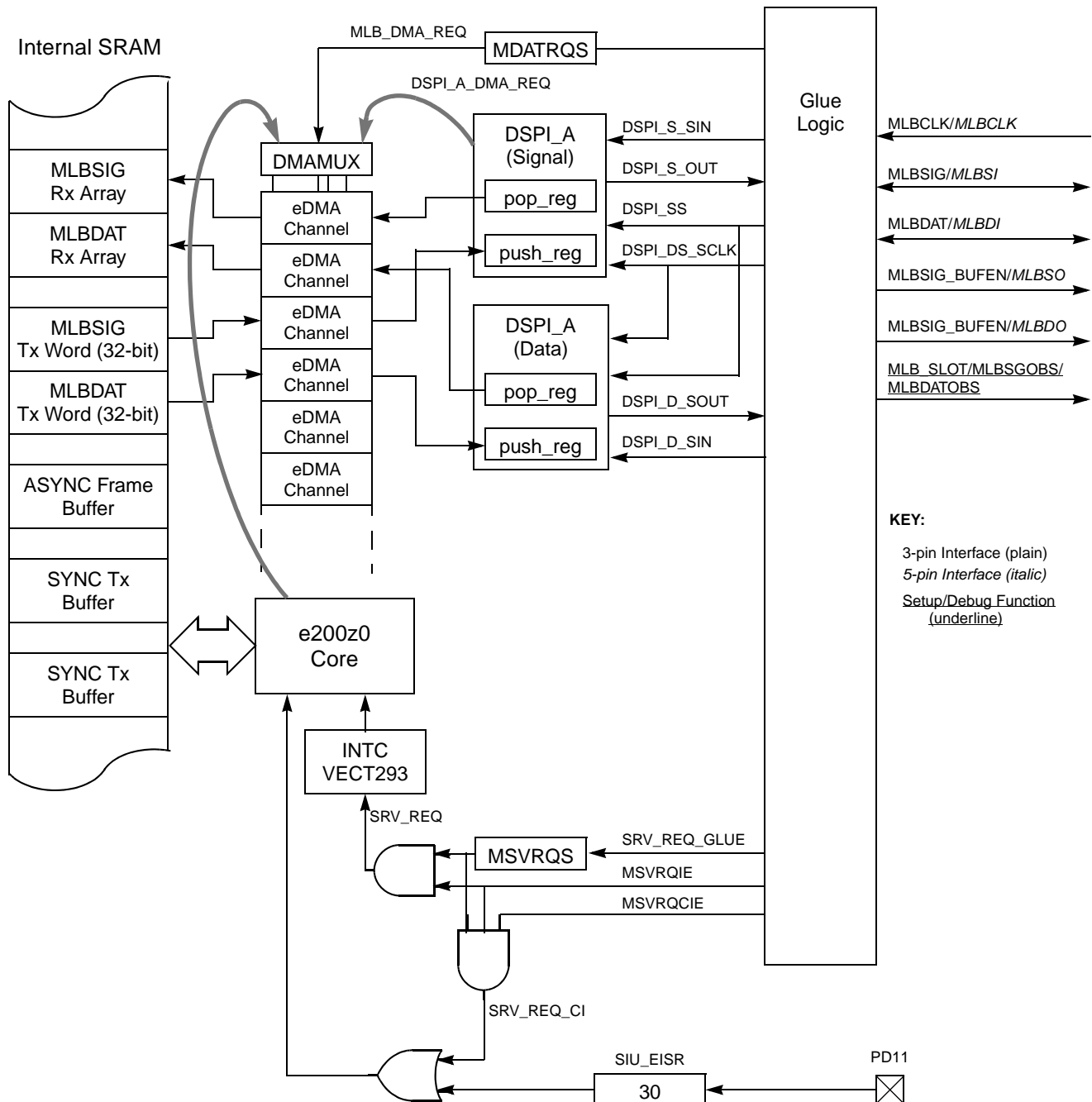


Figure 33-16. SoftMLB Data Flow

33.4.1 SoftMLB Interface Logic Description

The internal SoftMLB Interface Logic is an essential component of the softMLB solution. It is the interface between the 3-pin or 5-pin MLB interface and DSPI_A and DSPI_B on the MPC551xE/G. It also performs synchronization, eDMA triggering, IOP triggering, startup, clock adjust, and internal and external buffer control for the 3-pin and 5-pin configurations. The SoftMLB interface logic is enabled by asserting the

MLB_MCR[MDIS] bit. When enabled, it looks for the *FRAMESYNC* pattern (0x1FE) on the MLBSIG (or MLBSI) line. Detection of this pattern is used to synchronize to the bus. The logic detects 256 occurrences of the pattern before it is enabled to transmit data to/from the MLB bus. This is mandatory to prevent erroneous behavior of the MOST network controller (INIC) and to guarantee stable operation. Stable operation is signalled by setting the MLB_MSR[MSYSS] bit.

The Logic also generates an IOP trigger signal (SRV_REQ_GLUE) every 32 MLBCLK cycles that activates the IOP. This signal can be shifted up to 5 MLBCLK cycles from the MLB word boundary by the MLB_MCR[MSVRQDL] bits. The signal is an interrupt source into the INTC (see [Section 33.3.1, “Register Descriptions”](#) for details) or a critical interrupt which is Ored with the IOP external NMI. An eDMA request can also be generated from the SoftMLB Interface logic. This is input into the eDMA Mux (input 35) and generated every 32 MLBCLK cycles on the MLB word boundary.

In addition, the SoftMLB Interface Logic is responsible for selecting the 3-pin or 5-pin interface. It controls the output of the correct signals to the SIU in the 3-pin or 5-pin interface, and the level shifter enable signals in the 3-pin interface. See section [Section 33.4.2.1.4, “MLBSIG_BUFEN and MLBDAT_BUFEN”](#) for specific details of the level shifter control signals.

It also performs the MLBCLK adjust. The MLBCLK clock must be delayed to ensure proper setup and hold times for internal signals. The delay relative to the MLBCLK input is controlled via the MLB_CLKACR.

33.4.2 SoftMLB Interface Logic Signal Description

[Figure 33-17](#) shows a block diagram of the SoftMLB Interface Logic with all input and output signals. Integration of the SoftMLB Interface Logic in the SoC is shown in [Figure 33-16](#). Only the signals that are routed to the SIU are described below.

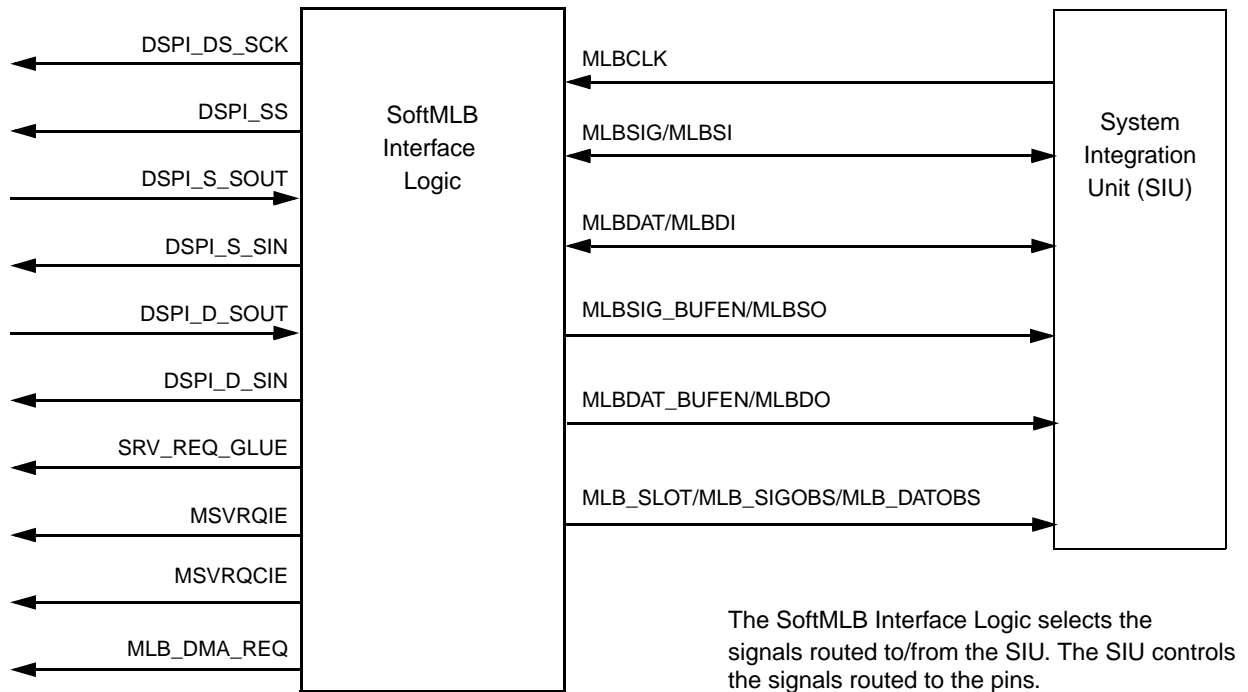


Figure 33-17. SoftMLB Interface Logic

33.4.2.1 Three-pin Interface

The 3-pin MLB interface has a separate unidirectional clock driven from the INIC and bidirectional signal and data lines. These are multiplexed out on the same pins as the 5pin interface.

33.4.2.1.1 MLBCLK

MLBCLK is the MLB bus clock input that is driven by the MLB master (INIC) and is synchronized with the MOST network. This signal has a typical frequency of 12.288 MHz. The maximum frequency is 12.3136 MHz (256 Fs at 48.1 kHz).

NOTE

This clock is asynchronous to the MCU system clock and is used to clock the SoftMLB Interface Logic.

33.4.2.1.2 MLBDAT

MLBDAT is a bidirectional data line that transfers data (synchronous, asynchronous, isochronous, control) to or from the network controller (INIC). The data word is 32 bits and the MSB is sent first.

33.4.2.1.3 MLBSIG

MLBSIG is a bidirectional signal line that transfers the bus management data to or from the network controller (INIC). The 32-bit word is comprised of the following three parts:

- Command — a byte-wide value that specifies the type of user data

- RxStatus — a byte-wide value that provides a handshaking mechanism
- Channel Address — a 16-bit token that is sent by the MLB master (INIC) and specifies a logical-channel.

33.4.2.1.4 MLBSIG_BUFEN and MLBDAT_BUFEN

These signals are technically not part of the 3-pin MLB specification, but are required to control the external level shifters and output enable of the internal pads when 3 pin mode is used. To drive the level shifter enable signals at the end of the MLB quadlet (32-bit timeslot), the content of the shift register is compared against potential MLB receive and transmit slots assigned to the device. These are configured in the Channel Address registers. See [Section 33.3.1, “Register Descriptions”](#) for a full description. If a valid comparison is detected, the appropriate external level shifter enable is asserted by the SoftMLB Interface Logic. The level of assertion is controlled by the MLB[MSBFEPOL:MDBFEPOL] bits. Details of when these signals are asserted are given below.

If the received Channel Address matches the programmed Rx Channel Address the MLBSIG output buffer is enabled in the next timeslot for eight bits during the RxStatus byte field. The MLBDAT output buffer is de-asserted

If the received Channel Address matches the programmed Tx Channel Address the MLBSIG output buffer is enabled in the next timeslot for 8 bits during the Command byte field. The MLBDAT output buffer is enabled in the next timeslot for 32 bits during Data word field.

NOTE

Multiple SYNC address are supported via the [Section 33.3.1.9, “TX Sync Channel Address Mask Register \(MLB_TXSCHAMR\)”](#)

If the received Channel Address matches the programmed Tx and Rx Channel Address this is an error condition. Both buffer are de-asserted

If the received Channel Address does not match the programmed Tx and Rx Channel Addresses this is a valid condition and both buffer are de-asserted.

33.4.2.2 Five-pin Interface

The 5-pin MLB interface has separate data and signal lines. The interface is half-duplex for consistency with the 3-pin interface. These signals are multiplexed out in the same positions as the 3-pin interface.

33.4.2.2.1 MLBCLK

MLBCLK is the MLB bus clock that is driven by the MLB master (INIC) and is synchronized with the MOST network. This signal has a typical frequency of 12.288 MHz. The maximum frequency is 12.3136 MHz (256 Fs at 48.1 kHz).

33.4.2.2.2 MLBDO

MLBDO transmits MLB Data from the MPC551xE/G onto the MLB bus.

33.4.2.2.3 MLBDI

MLBDI receives MLB Data from the MLB bus into the MPC551xE/G.

33.4.2.2.4 MLBSO

MLBSO transmits the MLB Signal from the MPC551xE/G onto the MLB bus.

33.4.2.2.5 MLBSI

MLBSI receives the MLB signal from the MLB bus into the MPC551xE/G.

33.4.2.2.6 MLB_DATOBS and MLB_SIGOBS

These signals are not part of the MLB specification. They are used during setup to observe the internal MLBDAT and MLBSIG signals during clock adjust.

33.4.2.2.7 MLB_SLOT

This signal toggles (low to high) exactly at the 32-bit boundary of an MLB word. It is a debug signal that can be used during development.

Appendix A

Revision History

This appendix describes corrections to the *MPC5510 Reference Manual*. For convenience, the corrections are grouped by revision.

A.1 Changes Between Revisions 0 and 1

Table A-1. Changes Between Revisions 0 and 1

Chapter	Description
1	<p>Removed references to SMSRUN and SMSSTOP and inserted "64K" in SLEEP mode description in section 1.5.</p> <p>Corrected Execution Speed row in Table 1-2.</p> <p>In Table 1-1, changed maximum number of DSPIs on 5517S from 3 to 4.</p> <p>Modified Figure 1-1.</p> <p>Inserted note on bit numbering in Section 1.1.</p> <p>Added new section 1.3.1.</p> <p>Modified Figure 1-1.</p> <p>Modified Section 1.5 to show that RTC continues to receive clock during STOP and SLEEP.</p> <p>Imported revised Table 1-1 and Table 1-5 from MPC5510PB Rev. 2.</p>
2	<p>Inserted pin number for V_{DDF} in Table 2-2.</p> <p>Changed "32,47" to "15,32,47" for V_{SSE2} in 144-pin package in Table 2-2.</p> <p>Corrected labels on pins 51, 52, 64 of 144-pin package.</p> <p>Changed "I/O" to "O" for PCS_C[0] in Table 2-1.</p> <p>Corrected labels on pins 52, 53 in 176-pin package diagram.</p> <p>Modified Table 2-1, row for PD2.</p> <p>Corrected signal descriptions to ensure eMIOS I/O capability agrees with Table 2-1.</p> <p>Modified Section 2.7.11.3.</p> <p>Modified Sections 2.7.11.9, 2.7.12.13, and 2.7.12.14.</p> <p>Modified Section 2.7.12.9.</p> <p>Modified Section 2.7.12.6.</p> <p>Modified Table 2-1, Table 2-2, and Figure 2-4.</p>
3	<p>Deleted PIT_RTI from Table 3-2.</p> <p>Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify.</p> <p>Changed 32KOSCEN to OSC32KEN, 32KRCTRIM to TRIM32IRC and IRCTRIM to TRIMIRC in block diagram to agree with chapter 5.</p> <p>Modified sections 3.2.1 and 3.2.2 to remove frequency spec values and refer to data sheet.</p> <p>In Table 3-1 added "PIT" to the list of modules in LPCLKDIV 1.</p> <p>Modified section 3.5.3,</p>
4	<p>Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify.</p> <p>Changed flag bits to show w1c where cleared by writing 1.</p>

Table A-1. Changes Between Revisions 0 and 1 (continued)

Chapter	Description
5	<p>Appended two notes to definition of WKCLKSEL in CRP_WKSE Field Descriptions table.</p> <p>Added INIT box to Sleep Mode Transition Diagram (Part 1) and Stop Mode Transition Diagram (Part 1).</p> <p>Significant content and editorial changes resulting from review of previous document revision.</p> <p>Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify.</p> <p>Changed bit names from “32KIRCEN” to “IRC32KEN” and “32KOSCEN” to “OSC32KEN”.</p> <p>Added w1c to flag bits cleared by writing 1.</p> <p>Added note to Clock Source register and modified TRIMIRC description.</p> <p>Modified WKPDETx bits description.</p> <p>Modified description of Z0VEC.</p> <p>Added notes below Z1VEC and Z0VEC descriptions.</p> <p>More changes to TRIM bits in Figure 5-2 and Table 5-2.</p> <p>More changes to PWKSRIE = WKPSEL bits.</p> <p>Added note to APIVAL bit description.</p>
6	<p>Changed Section 6.2.1.4, “Core Non-maskable Interrupt Pins (PD10 and PD11)”.</p> <p>Changed “HALTACK” to “HLTACK” in title of Table 6-28.</p> <p>Replaced Figure 6-49 with Figure 8-5.</p> <p>In SIU_SRCR register, changed bits 4 and 28 from read-only to R/W, and removed refs to SPR0 and SPRL0 from the footnotes.</p> <p>Various content and editorial changes.</p> <p>Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify.</p> <p>Fixed conditional text error in SIU_SRCR diagram.</p> <p>Changed SIU_CCR figure and bit descriptions.</p> <p>Added “w1c” to SIU_EISR and SIU_OSR registers and associated text.</p> <p>Modified section 6.3.2.5 and 6.3.2.12.</p> <p>Added table in LPCLKDIVn field description.</p>
7	<p>Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify.</p> <p>Minor editorial changes.</p> <p>Edited sections 7.1 and 7.3.2.6 to remove references to debug reset.</p> <p>Numerous editorial changes.</p>
8	<p>Changed Section 8.2.1, “Core Interrupts” and Section 8.4.3, “Non-Maskable Interrupt (NMI)”.</p> <p>Changed Figure 8.5. Copied this figure across to section 6 SIU.</p> <p>Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify.</p> <p>Modified Section 8.1, Table 8-1 and Section 8.2.4.</p> <p>More changes to Section 8.1 and Section 8.2.4.</p>
9	<p>Modified INTC Priority Select Register diagram (Figure 9-12) and field descriptions (Table 9-9).</p> <p>Added text to VTES_PRC1/0 bit description in INT_MCR register.</p> <p>Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify.</p> <p>Appended sentence and revised coding in section 9.5.5.2 Ensuring Coherency.</p> <p>Various editorial changes.</p>
10	<p>Added note to bit field description tables explaining bit numbering of Power PC Book E 64-bit registers.</p>
11	<p>Removed “Book E” from two locations in introduction.</p>

Table A-1. Changes Between Revisions 0 and 1 (continued)

Chapter	Description
12	Modified EDMA_CR diagram and field descriptions. Removed all references to groups and group functions. Modified SSIZE encoding. Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify.
13	Modified Figure 13-3 and Figure 13-6. Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Minor editorial changes. Changed “n-1” to “n” in Figure 13-2 and all italic x to n.
14	Modified sections 14.4.1 and 14.4.2.
15	Replaced Figure 15-1 with Figure 17-1. Modified Figure 15-1 and added footnote to Figure 15-3.
16	Corrected the reset value of MUDCR in Table 16-1. Modified MUDCR diagram and field descriptions. Changed MCMTIR to SWTIR in section 16.2.2.3. Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Changed flag bits in registers to show w1c., where appropriate. Modified note in section 16.1 and bullet in section 16.1.1.
17	Corrected Figures 17-5, 17-6, 17-7, 17-8, 17-9. Added note to reserved bit descriptions in Tables 17-7, 17-9. Corrected MPU base address in Table 17-1. Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Various editorial changes.
18	Corrected section 18.3.1 and Table 18-1. Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Changed CP0INE and CP1INE registers and bit description tables from 32-bit to 16-bit.
19	Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Inserted text in first para.
20	Added note explaining bit numbering convention used. Removed bit numbers in register bit field descriptions. Redrew flowchart diagrams to allow conversion to sans serif font. Cleaned up fonts in other diagrams. Removed bit numbers from bit names used in body text and modified surrounding text for clarification. Removed everything using MSB=0 condition. Added SYSCLK/1 as default frequency to Table 20-4 and section 20.5.5.
21	Modified section 21.4.2, and modified sections 21.6 and 21.7.
22	Deleted four lines of text in RWSC row in Table 22-11. Corrected LLOCK bit in LML register and field descriptions. Corrected SLOCK bits in SLL register and field descriptions. Corrected LSEL bits in LMS register and field descriptions. Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Added w1c to flag bits in register diagrams, where appropriate. Add information to Table 22-1.

Table A-1. Changes Between Revisions 0 and 1 (continued)

Chapter	Description
23	Corrected section 23.3.1. Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Changed "DIS_TX" to "DIS_TXF" in Table 23-2. Removed shading from w1c in register diagrams, where appropriate. Corrected reset value of TFFF bit in status register. Modified section 23.1 and added note to section 23.1.1.
24	Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Removed shading from w1c in register diagrams, where appropriate.
25	Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify.
26	Added note to section 26.2.1. Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Amended flag bits to use w1c clearing convention.
27	Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Changed IBCR bit names: "TX/RX" to "TX" and "MS/SL" to "MS". Changed all instances of "IIC" to "I2C". Amended flag bits to use w1c clearing convention.
28	Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Amended flag bits to use w1c clearing convention.
29	Changed ADDR[3:31] to ADDR[8:31] in Figure 29-19. Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Amended flag bits to use w1c clearing convention.
30	Changed "system clock / 3" to "system clock / 2" and "120 MHz" to "80 MHz" in section 30.4.2 PLL Clocking". Added base address to heading row in Memory Map table. Inserted "Reserved" at address 0x000A in memory map.
31	Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Edited text to show only one CFSSR register (and only one ADC). Cleaned up eQADC and EQADC. Cleaned up w1c bits.
32	Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify. Significant content and editorial changes resulting from review of previous document revision.
33	Changed "defaults to Disabled" to "defaults to MLB_SIGOBS" in Table 33-4. Removed superfluous bit numbers in bit field names. Modified text where necessary to clarify.