

# **88F6180, 88F6190, 88F6192, and 88F6281**

## Integrated Controller

## Functional Specifications

Doc. No. MV-S104860-U0, Rev. C  
December 2, 2008, Preliminary



## Document Conventions

	<b>Note:</b> Provides related information or information of special importance.
	<b>Caution:</b> Indicates potential damage to hardware or software, or loss of data.
	<b>Warning:</b> Indicates a risk of personal injury.

## Document Status

Doc Status: Preliminary	Technical Publication: 0.xx
-------------------------	-----------------------------

For more information, visit our website at: [www.marvell.com](http://www.marvell.com)

### Disclaimer

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose, without the express written permission of Marvell. Marvell retains the right to make changes to this document at any time, without notice. Marvell makes no warranty of any kind, expressed or implied, with regard to any information contained in this document, including, but not limited to, the implied warranties of merchantability or fitness for any particular purpose. Further, Marvell does not warrant the accuracy or completeness of the information, text, graphics, or other items contained within this document.

Marvell products are not designed for use in life-support equipment or applications that would cause a life-threatening situation if any such products failed. Do not use Marvell products in these types of equipment or applications.

With respect to the products described herein, the user or recipient, in the absence of appropriate U.S. government authorization, agrees:

- 1) Not to re-export or release any such information consisting of technology, software or source code controlled for national security reasons by the U.S. Export Control Regulations ("EAR"), to a national of EAR Country Groups D:1 or E:2;
- 2) Not to export the direct product of such technology or such software, to EAR Country Groups D:1 or E:2, if such technology or software and direct products thereof are controlled for national security reasons by the EAR; and,
- 3) In the case of technology controlled for national security reasons under the EAR where the direct product of the technology is a complete plant or component of a plant, not to export to EAR Country Groups D:1 or E:2 the direct product of the plant or major component thereof, if such direct product is controlled for national security reasons by the EAR, or is subject to controls under the U.S. Munitions List ("USML").

At all times hereunder, the recipient of any such information agrees that they shall be deemed to have manually signed this document in connection with their receipt of any such information.

Copyright © 2008. Marvell International Ltd. All rights reserved. Marvell, the Marvell logo, Moving Forward Faster, Alaska, Fastwriter, Datacom Systems on Silicon, Libertas, Link Street, NetGX, PHYAdvantage, Prestera, Raising The Technology Bar, The Technology Within, Virtual Cable Tester, and Yukon are registered trademarks of Marvell. Ants, AnyVoltage, Discovery, DSP Switcher, Feroceon, GalNet, GalTis, Horizon, Marvell Makes It All Possible, RADLAN, UniMAC, and VCT are trademarks of Marvell. All other trademarks are the property of their respective owners.

# Table of Contents

<b>Preface.....</b>	<b>17</b>
About this Document.....	17
Relevant Devices .....	17
Related Documentation.....	17
Document Conventions.....	19
<b>1     Overview.....</b>	<b>20</b>
1.1    Block Diagrams .....	21
1.2    Overview of Functions and Interfaces .....	25
1.3    Differences Between the 88F6180, 88F6190, 88F6192, and 88F6281 Devices.....	31
<b>2     Address Map .....</b>	<b>34</b>
2.1    Sheeva™ CPU Core Address Decoding.....	34
2.2    TDM (SLIC/Codec) Address Map (88F6192/88F6281 Only) .....	37
2.3    PCI Express Address Decoding .....	37
2.4    SATA Address Decoding (88F619x/88F6281).....	39
2.5    Gigabit Ethernet Address Decoding.....	39
2.6    USB Address Decoding .....	39
2.7    Security Accelerator Address Decoding.....	39
2.8    XOR Engine Address Decoding.....	40
2.9    TWSI Address Decoding.....	40
2.10   Audio Interface Address Map (88F6180/88F6192/88F6281 Only).....	40
2.11   SDIO Address Map .....	40
2.12   Transport Stream (TS) Address Map (88F6192/88F6281 Only) .....	41
2.13   Default Address Map.....	41
<b>3     Sheeva™ CPU Core.....</b>	<b>43</b>
<b>4     DDR SDRAM Controller .....</b>	<b>44</b>
4.1    SDRAM Controller Implementation .....	44
4.2    DDR SDRAM Addressing.....	45
4.3    SDRAM Timing Parameters.....	47
4.4    DRAM Burst .....	48
4.5    SDRAM Bank Interleaving .....	48
4.6    SDRAM Open Pages .....	49
4.7    SDRAM Refresh.....	49
4.8    SDRAM Initialization .....	49
4.9    SDRAM Operation Register .....	50



---

4.10	SDRAM Self Refresh Mode .....	50
4.11	Heavy Load Support .....	52
4.12	SDRAM Clocking .....	52
4.13	SDRAM Address/Data Drive .....	52
4.14	SDRAM Read Data Sample .....	53
4.15	DDR2 On Die Termination (ODT) .....	53
<b>5</b>	<b>Time Division Multiplexing (TDM) Unit (88F6192 and 88F6281 Only).....</b>	<b>56</b>
5.1	Functional Description.....	56
5.2	TDM Protocol Specification .....	59
5.3	TDM (SLIC/Codec) Registers Access via SPI.....	71
<b>6</b>	<b>PCI Express Interface.....</b>	<b>74</b>
6.1	Functional Description.....	74
6.2	Link Initialization .....	76
6.3	Master Memory Transactions.....	76
6.4	Master I/O Transactions .....	77
6.5	Master Configuration Transactions .....	77
6.6	Target Memory Transactions .....	78
6.7	Target I/O Transactions .....	78
6.8	Target Configuration Transactions .....	78
6.9	Target Special Cases .....	79
6.10	Messages.....	79
6.11	Message Signaled Interrupts (MSI).....	81
6.12	Locked Transactions .....	81
6.13	Arbitration and Ordering .....	81
6.14	PCI Express Register Access .....	82
6.15	Hot Reset .....	83
6.16	Link Disable.....	83
6.17	Power Management .....	83
6.18	Error Handling .....	84
6.19	Loopback Modes .....	86
6.20	Peer-to-Peer Traffic.....	88
<b>7</b>	<b>Serial-ATA (SATA) II Interface (88F619x and 88F6281 Only).....</b>	<b>89</b>
7.1	Serial ATA II Host Controller (SATAHC) .....	89
7.2	SATAHC Block Diagram .....	90
7.3	SATAHC Initialization .....	90
7.4	Host Direct Control Over the Hard Disk Drive .....	90
7.5	LED Indications .....	91
7.6	EDMA Operation .....	91
7.7	BIST .....	111

7.8	Vendor Unique .....	112
7.9	Protocol Based Port Select .....	112
<b>8</b>	<b>Gigabit Ethernet Controller .....</b>	<b>113</b>
8.1	Port Features .....	113
8.2	Functional Overview.....	114
8.3	DMA Functionality .....	116
8.4	Receive Frame Processing .....	134
8.5	Marvell® Header Support.....	136
8.6	Distributed Switching Architecture (DSA) Tag Support.....	139
8.7	Ethernet Interrupts .....	143
8.8	Transmit Weighted Round-Robin Arbitration.....	144
8.9	Token Rate Configuration .....	146
8.10	Transmit Queues Egress Jitter Pacing (EJP) Arbitration .....	147
8.11	Network Interface (10/100/1000 Mbps).....	150
8.12	Auto-Negotiation .....	153
8.13	Data Binder .....	154
8.14	Inter-packet Gap .....	154
8.15	Illegal Frames.....	154
8.16	Backpressure Mode .....	155
8.17	Flow Control .....	155
8.18	Serial Management Interface (SMI) .....	157
8.19	Link Detection and Link Detection Bypass (ForceLinkPass*) .....	158
8.20	Precise Time Protocol (PTP).....	158
8.21	Network Management Interface Counters.....	168
8.22	Port MIB Counters.....	168
<b>9</b>	<b>Universal Serial Bus (USB 2.0) Interface.....</b>	<b>173</b>
<b>10</b>	<b>Cryptographic Engines and Security Accelerator (CESA) .....</b>	<b>174</b>
10.1	Cryptographic Engine Features .....	175
10.2	Security Accelerator Features .....	175
10.3	Cryptographic Engines Operational Description .....	175
10.4	Security Accelerator Operational Description.....	189
10.5	TDMA Controller .....	200
<b>11</b>	<b>XOR Engine.....</b>	<b>205</b>
11.1	Theory of Operation .....	205
11.2	Descriptor Chain .....	209
11.3	Address Decoding.....	213
11.4	Arbitration.....	214
11.5	XOR Engine Programming.....	215



---

11.6	Burst Limit .....	219
11.7	Errors and Interrupts .....	220
<b>12</b>	<b>Two-Wire Serial Interface (TWSI) .....</b>	<b>221</b>
12.1	TWSI Bus Operation .....	221
12.2	TWSI Port Operation.....	222
12.3	TWSI Serial ROM Initialization.....	227
<b>13</b>	<b>UART Interface.....</b>	<b>228</b>
13.1	Features .....	228
13.2	UART Interface Pin Assignment .....	228
13.3	Operation .....	228
13.4	Programmable Baud-Rate Generator .....	229
<b>14</b>	<b>8-bit NAND Flash Interface .....</b>	<b>231</b>
14.1	NAND Flash Interface Pin Assignment .....	231
14.2	NAND Flash Types .....	231
14.3	Software Responsibilities .....	231
14.4	NAND Flash Interface Read Timing Parameters .....	232
14.5	NAND Flash Interface Write Timing Parameters.....	234
14.6	Boot from NAND Flash.....	234
<b>15</b>	<b>Serial Peripheral Interface (SPI) .....</b>	<b>236</b>
15.1	SPI Interface Signals.....	236
15.2	Indirect Mode .....	237
15.3	Direct Mode .....	237
<b>16</b>	<b>Audio (I<sup>2</sup>S / S/PDIF) Interface (88F6180, 88F6192, and 88F6281 Only) .....</b>	<b>240</b>
16.1	Recording Data Flow.....	242
16.2	Playback Flow .....	246
16.3	Error Handling .....	251
16.4	Audio Unit Memory Structure .....	252
<b>17</b>	<b>Secure Digital Input/Output (SDIO) Interface.....</b>	<b>256</b>
17.1	Features .....	257
17.2	SDMem, MMC, and SDIO Arbitration Scheme .....	257
17.3	Difference Between SD Cards and MMC Cards .....	259
17.4	SDIO / SDMem / MMC Host Controller Initialization .....	259
17.5	SDIO / SDMem / MMC Command Execution.....	259
17.6	SDIO / SDMem / MMC Interrupts.....	261
<b>18</b>	<b>Transport Stream (TS) Interface (88F6192 and 88F6281 Only) .....</b>	<b>262</b>
18.1	TS Port Architecture .....	263
18.2	TS Interface.....	263

18.3	Clocks .....	266
18.4	TS Input Data Flow .....	267
18.5	TS Output Data Flow.....	267
18.6	DMA Engine .....	267
18.7	TS Timestamp Mechanism.....	272
18.8	TS Packet Aggregation .....	273
18.9	TS Port Interrupts.....	275
18.10	Loopback Mode.....	276
<b>19</b>	<b>General-Purpose I/O (GPIO) Port Interface .....</b>	<b>277</b>
19.1	GPIO Control Registers .....	277
19.2	GPIO Blink Enable Register.....	277
19.3	GPIO Interrupts .....	277
<b>20</b>	<b>Real-Time Clock (RTC) Unit.....</b>	<b>278</b>
20.1	Features .....	278
20.2	Functionality .....	278
<b>21</b>	<b>Interrupt Controller.....</b>	<b>280</b>
21.1	Local Interrupt Cause and Mask Registers .....	280
21.2	Main Interrupt Cause and Mask Registers .....	281
21.3	Doorbell Interrupt .....	281
21.4	Device Interrupt Controller Scheme .....	282
<b>22</b>	<b>Timers and Counters.....</b>	<b>283</b>
22.1	32-bit General-Purpose Timers .....	283
22.2	Watchdog Timer.....	283
22.3	RTC Alarm .....	283
22.4	SYSRSTn Duration Counter .....	285
<b>23</b>	<b>eFuse .....</b>	<b>286</b>
23.1	Typical eFuse Applications .....	286
23.2	eFuse Power Supply .....	286
23.3	eFuse Program and Lock.....	286
23.4	eFuse Read.....	287
<b>24</b>	<b>System Considerations.....</b>	<b>288</b>
24.1	Big and Little Endian Support.....	288
24.2	BootROM Firmware .....	290
24.3	Power Management .....	303
24.4	Error Handling Functional Description.....	309



<b>25</b>	<b>Internal Architecture .....</b>	<b>312</b>
25.1	Mbus-L—Sheeva™ CPU Core Local Bus.....	312
25.2	Mbus—Device Internal Bus.....	315
25.3	Mbus-L to Mbus Bridge .....	317
25.4	Transaction Ordering .....	317
<b>A</b>	<b>88F6180/88F619x/88F6281 Register Set.....</b>	<b>352</b>
A.1	Registers Overview .....	352
A.2	Internal Registers Address Map .....	354
<b>B</b>	<b>Revision History .....</b>	<b>786</b>

# List of Tables

<b>Preface.....</b>	<b>17</b>
<b>1   Overview.....</b>	<b>20</b>
Table 1:   88F6180, 88F619x, and 88F6281 Device Differences and Similarities.....	31
<b>2   Address Map .....</b>	<b>34</b>
Table 2:   Units IDs and Attributes—CPU .....	35
Table 3:   Unit IDs and Attributes—PCI Express .....	38
Table 4:   Device Default Address Map .....	41
<b>3   Sheeva™ CPU Core.....</b>	<b>43</b>
<b>4   DDR SDRAM Controller.....</b>	<b>44</b>
Table 5:   DDR2 DRAM Addressing .....	45
Table 6:   Address Multiplex for 16b Interface, AddrSel = 0 .....	46
Table 7:   Address Multiplex for 16b Interface, AddrSel = 1 .....	46
Table 8:   SDRAM Timing Parameters .....	47
Table 9:   M_STARTBURST Output Assertion Point Configuration.....	53
<b>5   Time Division Multiplexing (TDM) Unit (88F6192 and 88F6281 Only) .....</b>	<b>56</b>
Table 10:   Time Division Multiplexing (TDM) Interface Signals .....	58
<b>6   PCI Express Interface.....</b>	<b>74</b>
Table 11:   Supported Message Groups—Root Complex Mode .....	79
Table 12:   Supported Message Groups—Endpoint Mode .....	80
Table 13:   Physical Layer Error List.....	84
Table 14:   Data Link Layer Error List .....	84
Table 15:   Transaction Layer Error List .....	85
<b>7   Serial-ATA (SATA) II Interface (88F619x and 88F6281 Only) .....</b>	<b>89</b>
Table 16:   Disc Status LED State Settings .....	91
Table 17:   EDMA CRQB Data Structure Map .....	104
Table 18:   CRQB DW0—cPRD Descriptor Table Base Low Address .....	105
Table 19:   CRQB DW1—cPRD Descriptor Table Base High Address .....	105
Table 20:   CRQB DW2—Control Flags .....	105
Table 21:   CRQB DW3—Data Region Byte Count .....	106
Table 22:   CRQB DW4—ATA Command .....	106
Table 23:   CRQB DW5—ATA Command .....	106
Table 24:   CRQB DW6—ATA Command .....	107
Table 25:   CRQB DW7—ATA Command .....	107
Table 26:   ePRD Table Data Structure Map .....	108
Table 27:   ePRD DWORD 0 .....	108
Table 28:   ePRD DWORD 1 .....	109



---

Table 29:	ePRD DWORD 2 .....	109
Table 30:	ePRD DWORD 3 .....	109
Table 31:	EDMA CRPB Data Structure Map .....	110
Table 32:	CRPB ID Register .....	110
Table 33:	CRPB Response Flags Register .....	110
Table 34:	CRPB Time Stamp Register .....	111
<b>8</b>	<b>Gigabit Ethernet Controller.....</b>	<b>113</b>
Table 35:	Transmit Descriptor Command/Status .....	123
Table 36:	Transmit Descriptor Byte Count.....	125
Table 37:	Transmit Descriptor Buffer Pointer .....	125
Table 38:	Transmit Descriptor Next Descriptor Pointer .....	125
Table 39:	Receive Descriptor Command/Status.....	132
Table 40:	Receive Descriptor Byte Count.....	134
Table 41:	Receive Descriptor Buffer Pointer .....	134
Table 42:	Receive Descriptor Next Descriptor Pointer .....	134
Table 43:	Marvell Header Fields .....	137
Table 44:	DSA Tag Fields (TO_CPU Format) .....	140
Table 45:	DSA Tag Fields (FORWARD Format) .....	141
Table 46:	Token Rate Configuration Examples .....	147
Table 47:	SMI Bit Stream Format .....	157
Table 48:	Definitions for MAC MIB Counters .....	168
<b>9</b>	<b>Universal Serial Bus (USB 2.0) Interface .....</b>	<b>173</b>
<b>10</b>	<b>Cryptographic Engines and Security Accelerator (CESA) .....</b>	<b>174</b>
Table 49:	Acronyms, Abbreviations, and Definitions .....	174
Table 50:	Security Accelerator Data Structure Dword 0—Configuration .....	196
Table 51:	Security Accelerator Data Structure Dword 1—Encryption Pointers .....	197
Table 52:	Security Accelerator Data Structure Dword 2—Encryption Data Length.....	198
Table 53:	Security Accelerator Data Structure Dword 3—Encryption Keys Pointer .....	198
Table 54:	Security Accelerator Data Structure Dword 4—Encryption Initial Values Pointer.....	198
Table 55:	Security Accelerator Data Structure Dword 5—MAC Source Pointer.....	199
Table 56:	Security Accelerator Data Structure Dword 6—MAC Digest .....	199
Table 57:	Security Accelerator Data Structure Dword 7—MAC Initial Values Pointers .....	199
Table 58:	TDMA Descriptor Definitions .....	201
<b>11</b>	<b>XOR Engine .....</b>	<b>205</b>
Table 59:	Descriptor Status Word Definition.....	211
Table 60:	Descriptor CRC-32 Result Word Definition.....	211
Table 61:	Descriptor Command Word Definition .....	211
Table 62:	Descriptor Next Descriptor Address Word .....	212
Table 63:	Descriptor Byte Count Word .....	212
Table 64:	Descriptor Destination Address Word .....	212
Table 65:	Descriptor Source Address #N Words .....	213
Table 66:	EOC/EOD interpretation .....	220

---

<b>12</b>	<b>Two-Wire Serial Interface (TWSI) .....</b>	<b>221</b>
	Table 67: TWSI Control Register Bits .....	223
	Table 68: TWSI Status Codes .....	224
<b>13</b>	<b>UART Interface.....</b>	<b>228</b>
	Table 69: Typical Baud Rates where TCLK = 166 MHz .....	230
<b>14</b>	<b>8-bit NAND Flash Interface .....</b>	<b>231</b>
	Table 70: Device Controller Pin Assignments .....	231
<b>15</b>	<b>Serial Peripheral Interface (SPI) .....</b>	<b>236</b>
	Table 71: SPI Interface Signals .....	236
<b>16</b>	<b>Audio (I<sup>2</sup>S / S/PDIF) Interface (88F6180, 88F6192, and 88F6281 Only) .....</b>	<b>240</b>
	Table 72: Audio Unit Memory Bit Description .....	255
<b>17</b>	<b>Secure Digital Input/Output (SDIO) Interface .....</b>	<b>256</b>
	Table 73: Software Flow .....	260
<b>18</b>	<b>Transport Stream (TS) Interface (88F6192 and 88F6281 Only).....</b>	<b>262</b>
	Table 74: Transport Stream (TS) Interface Signal Assignment .....	264
<b>19</b>	<b>General-Purpose I/O (GPIO) Port Interface .....</b>	<b>277</b>
<b>20</b>	<b>Real-Time Clock (RTC) Unit .....</b>	<b>278</b>
<b>21</b>	<b>Interrupt Controller .....</b>	<b>280</b>
<b>22</b>	<b>Timers and Counters .....</b>	<b>283</b>
	Table 75: Alarm Interrupt Valid Bit Usage .....	284
<b>23</b>	<b>eFuse .....</b>	<b>286</b>
<b>24</b>	<b>System Considerations .....</b>	<b>288</b>
	Table 76: MMU Virtual-to-Physical Address Translation Table .....	291
	Table 77: Main Header Format.....	293
	Table 78: Header Extension Format.....	294
	Table 79: Types of NAND Flash Read Commands Supported.....	301
	Table 80: Types of ECC Protocols Supported per Flash Type .....	302
	Table 81: Bad Block Indicators per NAND Flash Cell Type.....	302
	Table 82: 512 Mb—SDRAM IDD Values .....	304
	Table 83: CPU Address Decoding Error Handling.....	310
	Table 84: PCI Express Error Handling.....	310
	Table 85: USB Error Handling .....	311
<b>25</b>	<b>Internal Architecture.....</b>	<b>312</b>
	Table 86: Mbus Units .....	315



---

<b>A</b>	<b>88F6180/88F619x/88F6281 Register Set .....</b>	<b>352</b>
Table 87:	Register Field Type Codes .....	352
Table 88:	Device Internal Registers Address Map .....	354
Table 89:	Register Map Table for the Mbus-L to Mbus Bridge Registers.....	355
Table 159:	Register Map Table for the DDR SDRAM Controller Registers.....	389
Table 194:	Register Map Table for the Time Division Multiplexing (TDM) Unit Registers.....	413
Table 242:	Register Map Table for the PCI Express Interface Registers .....	437
Table 326:	Register Map Table for the Serial-ATA Host Controller (SATAHC) Registers.....	491
Table 399:	Shadow Register Block Registers Map .....	549
Table 400:	Register Map Table for the Gigabit Ethernet Controller Registers.....	550
Table 465:	Register Map Table for the PTP Registers .....	597
Table 500:	Register Map Table for the USB 2.0 Registers.....	625
Table 515:	USB Controller Register Map (Offsets: 0x50000–0x502FF).....	632
Table 516:	Register Map Table for the Cryptographic Engine and Security Accelerator (CESA) Registers .....	634
Table 581:	Register Map Table for the XOR Engine Registers .....	657
Table 601:	Register Map Table for the TWSI Registers .....	671
Table 610:	Register Map Table for the NAND Flash Registers .....	675
Table 614:	Register Map Table for the UART Registers .....	678
Table 627:	Register Map Table for the SPI Registers .....	685
Table 636:	Register Map Table for the Audio Interface Registers .....	689
Table 677:	Register Map Table for the SDIO Registers .....	714
Table 729:	Register Map Table for the Transport Stream (TS) Registers .....	744
Table 740:	Register Map for TSU Registers .....	749
Table 766:	Register Map Table for the General Purpose Port Registers .....	762
Table 783:	Register Map Table for the RTC Registers.....	767
Table 790:	Register Map Table for the Boot ROM Registers .....	771
Table 792:	Register Map Table for the MPP Registers .....	773
Table 801:	Register Map Table for the eFuse Registers .....	779
Table 808:	Register Map Table for the Miscellaneous Registers .....	782
<b>B</b>	<b>Revision History .....</b>	<b>786</b>
Table 816:	Revision History .....	786

# List of Figures

<b>Preface.....</b>	<b>17</b>
<b>1 Overview.....</b>	<b>20</b>
Figure 1: 88F6180 Interface Block Diagram .....	21
Figure 2: 88F6190 Interface Block Diagram .....	22
Figure 3: 88F6192 Interface Block Diagram .....	23
Figure 4: 88F6281 Interface Block Diagram .....	24
<b>2 Address Map .....</b>	<b>34</b>
<b>3 Sheeva™ CPU Core.....</b>	<b>43</b>
<b>4 DDR SDRAM Controller.....</b>	<b>44</b>
Figure 5: DDR2 I/O Buffer .....	54
<b>5 Time Division Multiplexing (TDM) Unit (88F6192 and 88F6281 Only).....</b>	<b>56</b>
Figure 6: SLIC/Codec Connection Example .....	56
Figure 7: TDM Unit Block Diagram .....	57
Figure 8: TDM Operation Time Slot 0 .....	59
Figure 9: TDM Wideband Mode Operation .....	60
Figure 10: TDM Transmit Path.....	63
Figure 11: TDM Receive Path.....	68
Figure 12: Codec Register Write Operation .....	72
Figure 13: Codec Register Read Operation.....	73
<b>6 PCI Express Interface.....</b>	<b>74</b>
Figure 14: High-level Block Diagram .....	75
Figure 15: Shallow Internal Loopback.....	87
Figure 16: Deep Internal Loopback.....	87
<b>7 Serial-ATA (SATA) II Interface (88F619x and 88F6281 Only).....</b>	<b>89</b>
Figure 17: SATAHC Block Diagram.....	90
Figure 18: Disc Status LED Indication Diagram.....	91
Figure 19: Command Request Queue—32 Entries .....	92
Figure 20: Command Response Queue—32 Entries .....	93
Figure 21: EDMA Interrupt Hierarchy.....	100
<b>8 Gigabit Ethernet Controller.....</b>	<b>113</b>
Figure 22: Ethernet Descriptors and Buffers.....	116
Figure 23: Ethernet Packet Transmission Example .....	119
Figure 24: Transmit Descriptor Description .....	122
Figure 25: Receive Descriptor Description .....	131
Figure 26: Rx Packet Marvell Header Example .....	137



---

Figure 27: Tx Packet with Marvell Header Example .....	138
Figure 28: Rx Packet with DSA Tag Example (4 bytes tag, TO_CPU Format).....	139
Figure 29: Tx Packet with a DSA Tag Example (FROM_CPU format, use_vidx = 0) .....	143
Figure 30: MII Connection.....	150
Figure 31: GMII Connection.....	152
Figure 32: RGMII Pin Interconnection Between MAC and PHY .....	152
Figure 33: PTP Common Header Format .....	160
Figure 34: PTP over UDP Frame .....	161
Figure 35: PTP 2.1 Pipe Block Diagram .....	162
Figure 36: Time Stamping Pipeline Stages.....	164
Figure 37: Ethernet Frame Classification.....	170
Figure 38: Bad Frame Procedure .....	171
<b>9 Universal Serial Bus (USB 2.0) Interface .....</b>	<b>173</b>
<b>10 Cryptographic Engines and Security Accelerator (CESA) .....</b>	<b>174</b>
Figure 39: Authentication of a Data Chunk .....	178
Figure 40: Typical Authentication Flow for a Packet.....	179
Figure 41: DES Engine Pipeline .....	181
Figure 42: Typical DES/3DES Encryption Flow for Packet.....	184
Figure 43: Typical AES Encryption Flow for a Data Block .....	187
Figure 44: Typical AES Decryption Flow for a Data Block.....	189
Figure 45: Security Accelerator Main Decision Flow .....	190
Figure 46: Security Acceleration Flow for Packet Processing.....	191
Figure 47: Security Acceleration Flow for Packet Processing—Enhanced Mode.....	192
Figure 48: TDMA Descriptors Structure for Security Accelerator Packet Processing in Enhanced Mode .....	193
Figure 49: TDMA Descriptors .....	201
Figure 50: Chained Mode TDMA .....	203
<b>11 XOR Engine .....</b>	<b>205</b>
Figure 51: Schematic Diagram of the Two XOR Engines .....	205
Figure 52: XOR Operation with Multiple Incoming Data Blocks.....	207
Figure 53: XOR iSCSI CRC32C Operation.....	208
Figure 54: XOR Descriptor Format .....	210
Figure 55: Programmable Channel Pizza Arbiter .....	214
Figure 56: Software and Hardware Synchronization .....	218
<b>12 Two-Wire Serial Interface (TWSI) .....</b>	<b>221</b>
Figure 57: TWSI Examples .....	222

<b>13</b>	<b>UART Interface .....</b>	<b>228</b>
	Figure 58: Example UART Data Frame (Two Stop Bits).....	229
	Figure 59: Example UART Data Frame (One Stop Bit) .....	229
<b>14</b>	<b>8-bit NAND Flash Interface .....</b>	<b>231</b>
	Figure 60: 8-bit NAND Flash Read Parameters Example.....	233
	Figure 61: 8-bit NAND Flash Write Parameters Example.....	234
<b>15</b>	<b>Serial Peripheral Interface (SPI) .....</b>	<b>236</b>
<b>16</b>	<b>Audio (I<sup>2</sup>S / S/PDIF) Interface (88F6180, 88F6192, and 88F6281 Only) .....</b>	<b>240</b>
	Figure 62: Audio Unit Block Diagram .....	240
	Figure 63: Recording Flow .....	244
	Figure 64: Playback Flow.....	249
	Figure 65: Memory Structure for Transmit and Receive .....	254
<b>17</b>	<b>Secure Digital Input/Output (SDIO) Interface .....</b>	<b>256</b>
	Figure 66: SD_MMC Host Controller Hardware Block Diagram .....	256
	Figure 67: Host Initialization Flow .....	258
<b>18</b>	<b>Transport Stream (TS) Interface (88F6192 and 88F6281 Only).....</b>	<b>262</b>
	Figure 68: TSU Block Diagram .....	262
	Figure 69: TS Interface Block Diagram.....	263
	Figure 70: TS Parallel Protocol (Example).....	266
	Figure 71: TS Continuous Serial Data Protocol (Example) .....	266
	Figure 72: TS Input Descriptor Queue .....	269
	Figure 73: TS Output Descriptor Structure—No Packet Aggregation.....	270
	Figure 74: TS Output Descriptor Queue—No Packet Aggregation.....	271
	Figure 75: Impact of Timestamp on the Average TS Data Output Data Rate .....	273
	Figure 76: Aggregated TS Input Mode.....	274
	Figure 77: Aggregated TS Output Mode.....	275
	Figure 78: TS Loopback Modes.....	276
<b>19</b>	<b>General-Purpose I/O (GPIO) Port Interface .....</b>	<b>277</b>
<b>20</b>	<b>Real-Time Clock (RTC) Unit .....</b>	<b>278</b>
<b>21</b>	<b>Interrupt Controller .....</b>	<b>280</b>
	Figure 79: Device Interrupt Controller Scheme.....	282



---

<b>22</b>	<b>Timers and Counters.....</b>	<b>283</b>
<b>23</b>	<b>eFuse .....</b>	<b>286</b>
<b>24</b>	<b>System Considerations.....</b>	<b>288</b>
	Figure 80: Binary Image Layout in the Boot Device.....	292
	Figure 81: Initialization and Boot Method Selection Flow .....	296
	Figure 82: Header Decoding, DDR Initialization, and Image Execution Flowchart.....	299
	Figure 83: Endpoint Power Supply Control .....	307
<b>25</b>	<b>Internal Architecture.....</b>	<b>312</b>
	Figure 84: 88F6180 and 88F619x Bus Interface Unit Mbus-L Block Diagram.....	312
	Figure 85: 88F6281 Bus Interface Unit Mbus-L Block Diagram.....	313
	Figure 86: CPU to DDR Mbus-L Timing Diagrams—CPU2MbusLTickDrv=0, CPU2MbusLTickSample=0.....	314
	Figure 87: CPU to DDR Mbus-L Timing Diagrams—CPU2MbusLTickDrv=2, CPU2MbusLTickSample=2 .....	314
	Figure 88: Masters Request Default Arbitration Cycle.....	316
<b>A</b>	<b>88F6180/88F619x/88F6281 Register Set .....</b>	<b>352</b>
	Figure 89: PTP Configuration Data Structure Registers .....	600
	Figure 90: PTP Global Status Data Structure Registers.....	602
	Figure 91: PTP Port Configuration Data Structure Registers .....	603
	Figure 92: PTP Port Status Data Structure Registers.....	607
	Figure 93: TAI Global Configuration Data Structure .....	615
	Figure 94: PTP Time Application Interface Global Status Data Structure.....	620
<b>B</b>	<b>Revision History .....</b>	<b>786</b>

# Preface

## About this Document

This document provides the functional specifications for the 88F6180, 88F6190, 88F6192, and 88F6281 integrated controllers. This datasheet also provides detailed definitions for the registers implemented in these devices.

This document is intended to be the basic source of information for designers of new systems.

All feature descriptions and specifications described in this document refer to all the devices, unless otherwise specified. In this document, the 88F6180, 88F6190, 88F6192, and 88F6281 are often referred to as "the device/s". In addition, the 88F6190 and 88F6192 are often referred to as the 88F619x.

## Relevant Devices

- 88F6180
- 88F6190
- 88F6192
- 88F6281

## Related Documentation

The following documents contain additional information related to the 88F6180, 88F619x, and 88F6281:

- *88F6180 Hardware Specifications*, Doc No. MV-S104988-U0
- *88F6190 and 88F6192 Hardware Specifications*, Doc No. MV-S104987-U0
- *88F6281 Hardware Specifications*, Doc No. MV-S104859-U0
- *88F6180, 88F6190, 88F6192, and 88F6281 Design Guide*, Doc No. MV-S301398-00<sup>1</sup>
- *Sheeva™ 88SV131 ARM v5TE Processor Core with MMU and L1/L2 Cache Datasheet*, Doc No. MV-S104950-U0
- *Unified Layer 2 (L2) Cache for Sheeva™ CPU Cores Addendum*, Doc No. MV-S104858-U0
- *AN-179 TWSI Software Guidelines for Discovery™, Horizon™, and Feroceon® Devices*, Doc No. MV-S300754-00<sup>1</sup>
- *AN-183, 88F5181 and 88F5281 Big Endian and Little Endian Support*, Doc No. MV-S300767-00<sup>1</sup>
- *AN-249: Configuring the Marvell® SATA PHY to Transmit Predefined Test Patterns*, Doc No. MV-S301342-00<sup>1</sup>
- *AN-260 System Power-Saving Methods for 88F6180, 88F6190, 88F6192, and 88F6281*, Doc No. MV-S301454-00<sup>1</sup>
- *TB-227: Differences Between the 88F6192, and 88F6281 Stepping Z0 and A0*, Doc No. MV-S105223-00<sup>1</sup>
- *ARM Architecture Reference Manual*, Second Edition
- *PCI Express Base Specification*, Revision 1.1
- *Universal Serial Bus Specification*, Revision 2.0, April 2000, Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips

---

1. This document is a Marvell proprietary, confidential document, requiring an NDA and can be downloaded from the Marvell Extranet.



## 88F6180/88F619x/88F6281 Functional Specifications

- Enhanced Host Controller Interface Specification for Universal Serial Bus, Revision 0.95, November 2000, Intel Corporation
- ARC USB-HS OTG High-Speed USB On-The-Go Controller Core V 4.0.1 Reference.
- Federal Information Processing Standards (FIPS) 46-2 (Data Encryption Standard)
- FIPS 81 (DES Modes of Operation)
- FIPS 180-1 (Secure Hash Standard)
- FIPS draft - Advanced Encryption Standard (Rijndael)
- RFC 1321 (The MD5 Message-Digest Algorithm)
- RFC 1851 – The ESP Triple DES Transform
- RFC 2104 (HMAC: Keyed-Hashing for Message Authentication).
- RFC 2405 – The ESP DES-CBC Cipher Algorithm With Explicit IV
- IEEE standard, 802.3-2000 Clause 14
- ANSI standard X3.263-1995

See the Marvell Extranet website for the latest product documentation.

## Document Conventions

The following conventions are used in this document:

Signal Range	A signal name followed by a range enclosed in brackets represents a range of logically related signals. The first number in the range indicates the most significant bit (MSb) and the last number indicates the least significant bit (LSb).  Example: DB_Addr[12:0]
Active Low Signals #	An n letter at the end of a signal name indicates that the signal's active state occurs when voltage is low.  Example: INTn
State Names	State names are indicated in <i>italic</i> font.  Example: <i>linkfail</i>
Register Naming Conventions	Register field names are indicated by angle brackets. Example: <RegInit>  Register field bits are enclosed in brackets. Example: Field [1:0]  Register addresses are represented in hexadecimal format. Example: 0x0  Reserved: The contents of the register are reserved for internal use only or for future use. A lowercase <n> in angle brackets in a register indicates that there are multiple registers with this name. Example: Multicast Configuration Register<n>
Reset Values	Reset values have the following meanings: 0 = Bit clear 1 = Bit set
Abbreviations	Gb: gigabit GB: gigabyte Kb: kilobit KB: kilobyte Mb: megabit MB: megabyte
Numbering Conventions	Unless otherwise indicated, all numbers in this document are decimal (base 10). An 0x prefix indicates a hexadecimal number. An 0b prefix indicates a binary number.



# 1

## Overview

The Marvell® 88F6180, 88F6190, 88F6192, and 88F6281 devices are high-performance, highly integrated controllers. The devices are based on the ARMv5TE-compliant, high-speed Marvell® Sheeva™ 88SV131 CPU core with 256 KB L2 cache.

This section provides a brief description of the interfaces in each of these devices.



**Note**

The functions, interfaces, and registers/register bits described in this document do not necessarily apply to all of the devices.

## 1.1 Block Diagrams

Figure 1 is a block diagram of the 88F6180 interfaces.

**Figure 1: 88F6180 Interface Block Diagram**

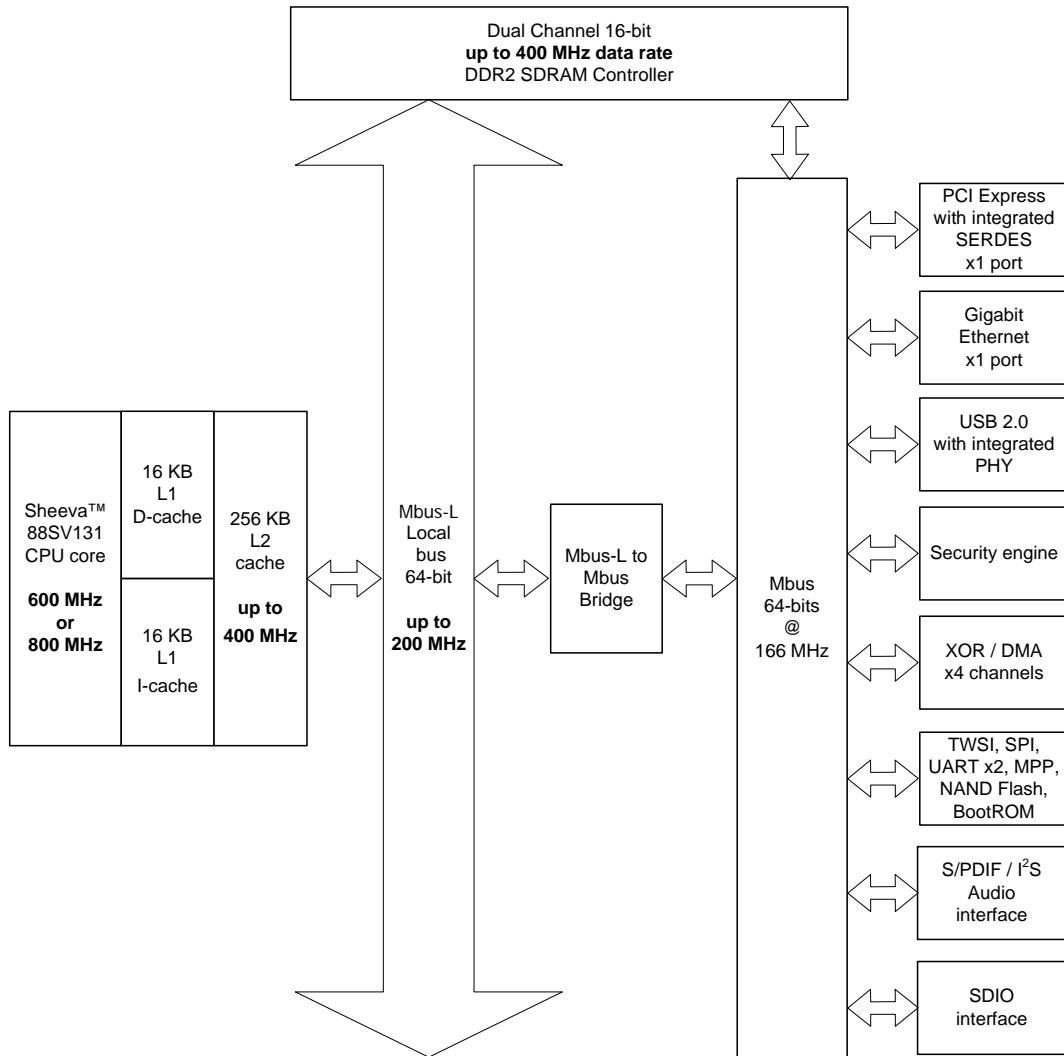


Figure 2 is a block diagram of the 88F6190 interfaces.

**Figure 2: 88F6190 Interface Block Diagram**

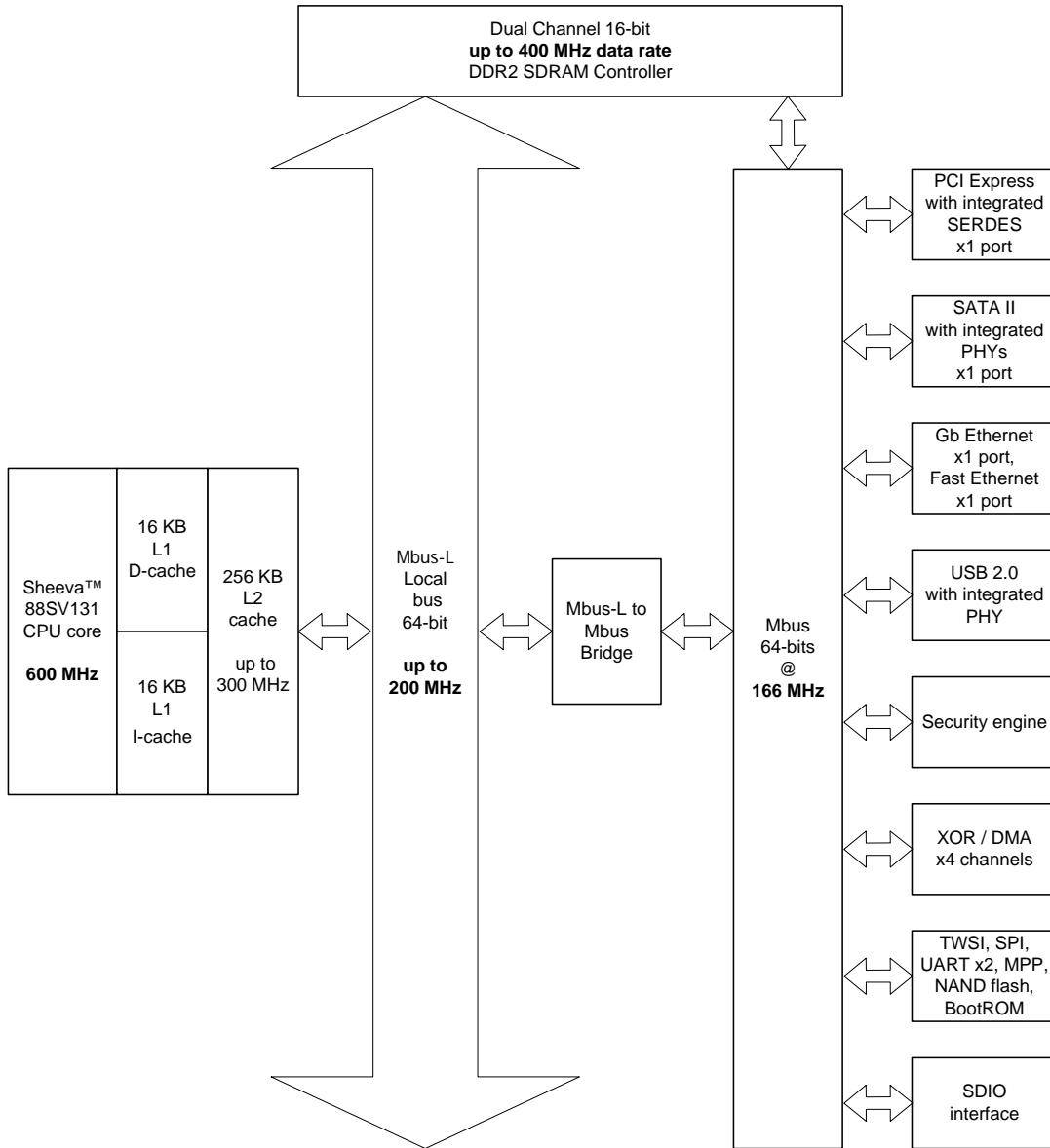


Figure 3 is a block diagram of the 88F6192 interfaces.

**Figure 3: 88F6192 Interface Block Diagram**

---

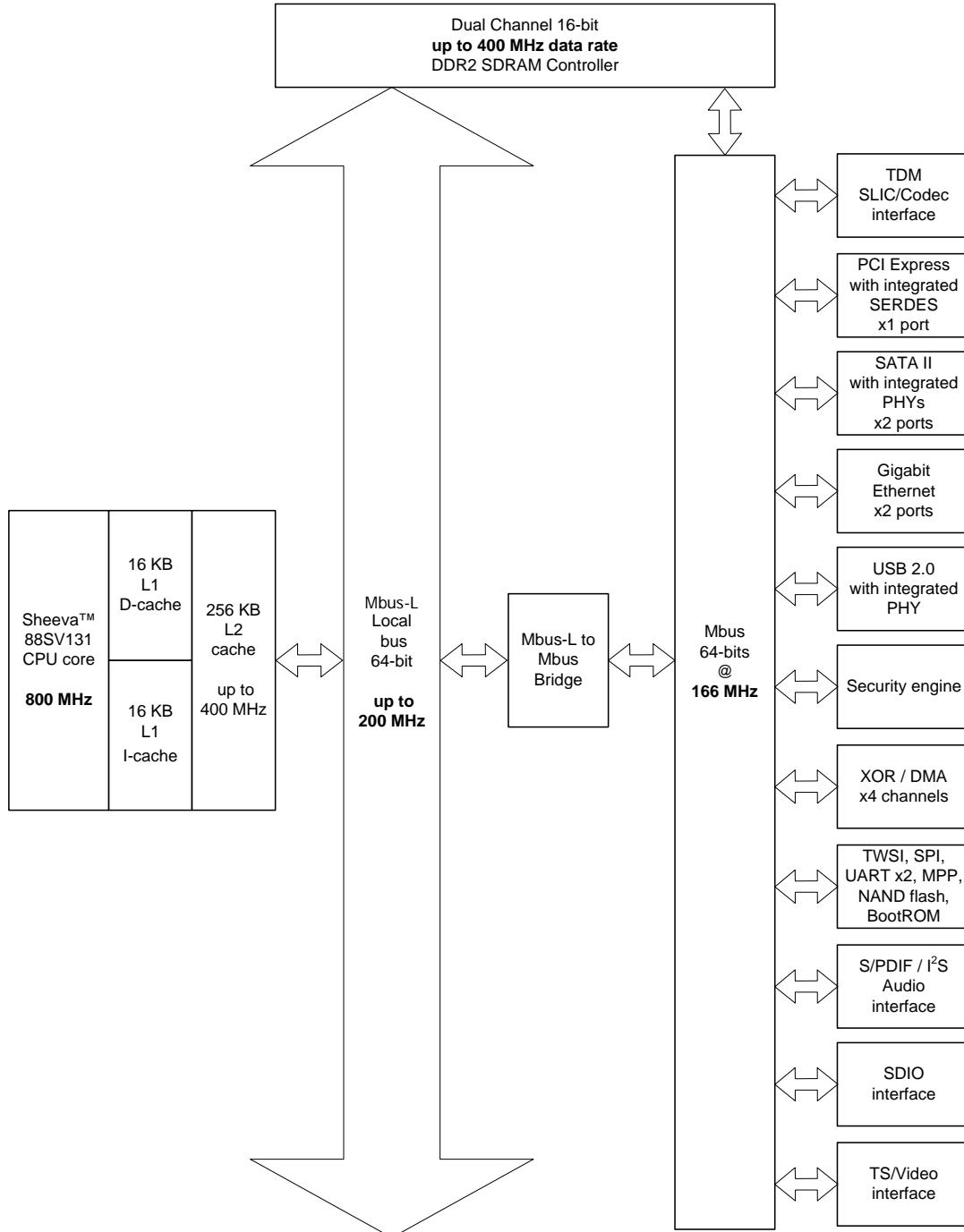
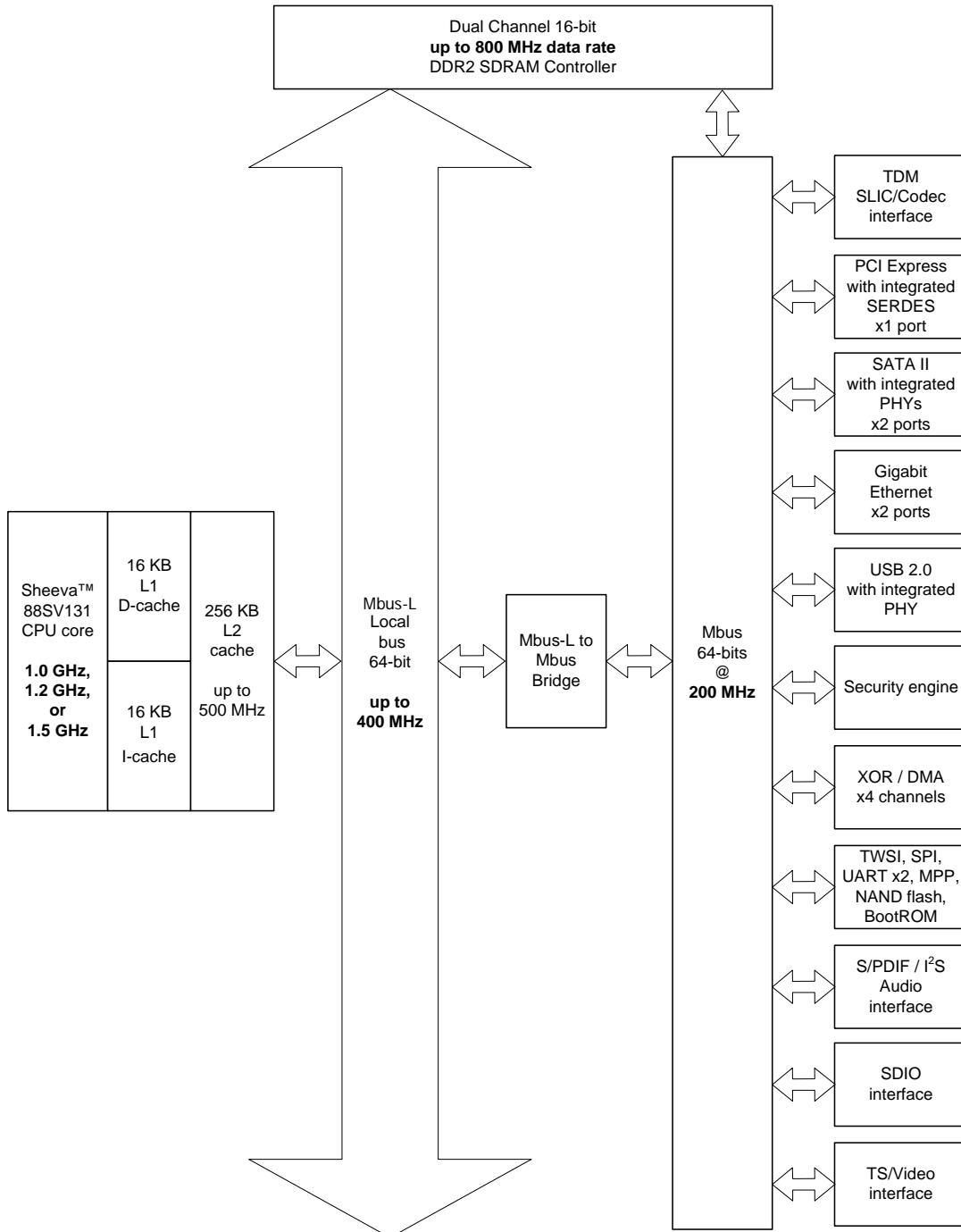


Figure 4 is a block diagram of the 88F6281 interfaces.

**Figure 4: 88F6281 Interface Block Diagram**



## 1.2

# Overview of Functions and Interfaces

The following is a list of the device functions and interfaces:

**Sheeva™ 88SV131 CPU Core** The device integrates the Sheeva 88SV131 CPU core. This core is compliant with ARMv5TE architecture, as published in the *ARM Architecture Reference Manual*, Second Edition. The Sheeva 88SV131 CPU core provides integrated 16/16 KB, four-way, set-associative I/D L1 caches and a unified 256 KB four-way, set-associative L2 cache.

- **88F6180:** Running at 600 MHz or 800 MHz
- **88F6190:** Running at 600 MHz
- **88F6192:** Running at 800 MHz
- **88F6281:** Running at 1.0 GHz, 1.2 GHz, or 1.5 GHz

The Sheeva 88SV131 CPU core also provides:

- 32-bit and 16-bit RISC architecture
- An MMU to support virtual memory features
- 64-bit internal data bus
- Branch Prediction Unit
- JTAG/ARM ICE support
- Big and Little Endian modes support

See [Section 3, Sheeva™ CPU Core, on page 43](#).

**DDR SDRAM Interface** The device integrates a 16-bit DDR2 SDRAM interface.

**88F6180 and 88F619x**

- Up to 200 MHz clock frequency with an 400 MHz data rate
- Supports two DRAM chip selects
- Supports all DDR2 devices with densities up to 1 Gb
- Supports up to 16 open pages (page per bank)
- Up to 512 MB total address space

**88F6281**

- Up to 400 MHz clock frequency with an 800 MHz data rate
- Supports four DRAM chip selects
- Supports all DDR2 devices with densities up to 2 Gb
- Supports up to 32 open pages (page per bank)
- Up to 2 GB total address space

**All of the devices**

Provide the following DDR SDRAM interface features:

- Support for on board DDR designs (no DIMM support)
- DDR SDRAM with a clock ratio of 1:N and 2:N between the DDR SDRAM and the CPU core, respectively
- SSTL 1.8V I/Os
- Auto calibration of I/Os output impedance
- Support for 2T mode to enable high-frequency operation with a heavy load configuration
- Supports DRAM bank interleaving
- Supports up to a 128-byte burst per single memory access

See [Section 4, DDR SDRAM Controller, on page 44](#).



<b>Time Division Multiplexing (SLIC/SLAC/Codec) Interface</b>	The 88F6192 and 88F6281 contain a Time Division Multiplexing (SLIC/SLAC/Codec) interface. The TDM is a generic interface to the standard SLIC/SLAC/codec devices. It provides: <ul style="list-style-type: none"><li>■ Compatibility with standard PCM highway formats</li><li>■ TDM protocol support for two channels, up to 128 time slots</li><li>■ SPI interface for codec register read/write access</li><li>■ Two integrated DMA engines to transfer voice data to/from memory buffer</li></ul> See <a href="#">Section 5, Time Division Multiplexing (TDM) Unit (88F6192 and 88F6281 Only), on page 56.</a>
<b>PCI Express Interface</b>	The device integrates a PCI Express Base 1.1 compatible interface containing a single PCI Express lane (x1) host port with an integrated low power SERDES, based on Marvell® SERDES technology. This interface can serve as a Root Complex or an Endpoint port with: <ul style="list-style-type: none"><li>■ x1 lane width</li><li>■ 2.5 Gbps data rate</li><li>■ Lane polarity reversal support</li><li>■ Maximum payload size of 128 bytes</li><li>■ Single Virtual Channel (VC-0)</li><li>■ Replay buffer support</li><li>■ Extended PCI Express configuration space</li><li>■ Advanced Error Reporting (AER) support</li><li>■ Power management: L0s and software L1 support</li><li>■ Interrupt emulation message support</li><li>■ Error message support</li></ul> As a master, the PCI Express interface contains: <ul style="list-style-type: none"><li>■ Single outstanding read transaction</li><li>■ Maximum read request of up to 128 bytes</li><li>■ Maximum write request of up to 128 bytes</li><li>■ Up to four outstanding read transactions in Endpoint mode</li></ul> As a target, the PCI Express interface contains: <ul style="list-style-type: none"><li>■ Supports up to eight read request transactions</li><li>■ Maximum read request size of 4 KB</li><li>■ Maximum write request of 128 bytes</li><li>■ Supports PCI Express access to all of the device's internal registers</li></ul> See <a href="#">Section 6, PCI Express Interface, on page 74.</a>

- Serial ATA II (SATA II) Interface** The 88F6192 and 88F6281 contain two and the 88F6190 contains one SATA II compliant 3 Gbps (Gen2i) SATA PHY(s). The SATA interface supports:
- SATA II Native command queuing, up to 128 outstanding commands per port
  - First party DMA (FPDMA) full support
  - Backwards compatibility to SATA I 1.5-Gbps speed and devices
  - Fully supports the SATA II Phase 1.0 specification, and the following advanced SATA II Phase 2.0 specification features:
    - 3 Gbps (Gen2i) SATA II speed
    - SATA II Port Multiplier performs FIS-Based Switching as defined in SATA working group Port Multiplier definition
    - SATA II Port Selector issues the protocol-based OOB sequence to select the active host port
  - Supports device 48-bit addressing
  - Supports ATA Tag Command Queuing
- The SATA Host Controller supports:
- Enhanced-DMA [EDMA] for the SATA ports
  - Automatic command execution without host intervention
  - Command queuing support, for up to 32 outstanding commands
  - Separate SATA request/response queues
  - 64-bit addressing support for descriptors and data buffers in system memory
  - Read ahead
  - Advanced interrupt coalescing
  - Target mode operation—Two devices can be attached back-to-back, through Serial ATA ports, enabling data communication between different 88F619x/88F6281 devices, with one acting as a host and the other emulate a device
  - Advanced drive diagnostics via the ATA SMART command
- See [Section 7, Serial-ATA \(SATA\) II Interface \(88F619x and 88F6281 Only\), on page 89.](#)



**Gigabit Ethernet  
Interface**

**88F6180**

The Gigabit Ethernet interface consists of a single full-duplex Gigabit Ethernet (GbE) port that supports an RGMII/MII/MMII interface.

**88F6190**

The Gigabit Ethernet interface consists of one full-duplex Gigabit Ethernet (GbE) port and one full-duplex Fast Ethernet (FE) port that supports the following modes:

- Port0 RGMII, Port1 MII/MMII
- Port0 GMII, Port1 N/A

**88F6192 and 88F6281**

The Gigabit Ethernet interface consists of two full-duplex Gigabit Ethernet (GbE) ports that support the following modes:

- Port0 RGMII, Port1 RGMII
- Port0 RGMII, Port1 MII/MMII
- Port0 MII/MMII, Port1 RGMII
- Port0 GMII, Port1 N/A

The Gigabit Ethernet interface supports 10/100/1000 Mbps speeds, as well as the 200 Mbps proprietary Marvell® MII (MMII).

Receive and transmit buffer management is based on buffer-descriptor linked lists. Data transfers are performed by the port dedicated SDMA (see [Section 8.3, DMA Functionality, on page 116](#)).

Each Ethernet port includes advanced Destination Address (DA) filtering on received packets that also detects packet type/encapsulations that can be used by the CPU for packet routing:

- Layer 2: BPDU,VLAN (programmable VLAN-EtherType), Ethernet v2, LLC/SNAP
- Layer 3: IPv4, IPv6 (according to Ethertype), other
- Layer 4 (only over IPv4): TCP and UDP

The port has eight receive priority queues. Queuing is performed based on DA, VLAN-Tag, IP-TOS, Marvell Header, and DSA Tag.

The port supports standard Ethernet frames (up to 1.5 KB) and, in addition, Jumbo frames (up to 9 KB).

It also supports hardware TCP and UDP checksum check on receive, and generate on transmit (checksum generation for Jumbo frames is not supported).

- Precise Timing Protocol (PTP) with:
  - Precise time stamping for packets, as defined in IEEE 1588 PTP v1 and v2 and IEEE 802.1AS draft standards
  - Flexible Time Application interface to distribute PTP clock and time to other devices in the system
  - Optionally accepts an external clock input for time stamping
- Audio Video Bridging Networks including:
  - IEEE 802.1Qav pre-draft Audio Video Bridging networks
  - Time- and priority-aware egress pacing algorithm to prevent bunching and bursting effects—suitable for audio/video applications
  - Egress Jitter Pacer for AVB-Class A and AVB-Class B traffic and strict priority for legacy traffic queues

See [Section 8, Gigabit Ethernet Controller, on page 113](#).

<b>USB 2.0 Interface</b>	<p>The device integrates a single USB 2.0 compliant high-speed port with an integrated PHY:</p> <ul style="list-style-type: none"> <li>■ Serves as a peripheral or host</li> <li>■ Enhanced Host Controller Interface (EHCI) compatible as a host</li> <li>■ As a host, supports direct connection to all peripheral types (LS, FS, HS)</li> <li>■ As a peripheral, connects to all host types (HS, FS) and hubs</li> <li>■ Integrates up to four independent Endpoints that support control, interrupt, bulk, and isochronous data transfers</li> <li>■ Integrates a dedicated DMA for data movement between memory and port</li> </ul> <p>See <a href="#">Section 9, Universal Serial Bus (USB 2.0) Interface, on page 173.</a></p>
<b>Cryptographic Engine and Security Accelerator</b>	<p>The device integrates a Cryptographic Engine and Security Accelerator to support data encryption and authentication. It also contains a dedicated Direct Memory Access (DMA) controller to perform the following:</p> <ul style="list-style-type: none"> <li>■ Hardware implementation of encryption and authentication engines to boost packet processing speed</li> <li>■ Dedicated DMA to feed the hardware engines with data from the internal SRAM memory or from the DDR memory</li> <li>■ Implements AES, DES, and 3DES encryption algorithms</li> <li>■ Implements SHA1 and MD5 authentication algorithms</li> </ul> <p>See <a href="#">Section 10, Cryptographic Engines and Security Accelerator (CESA), on page 174.</a></p>
<b>XOR / DMA Channel</b>	<p>The device integrates four XOR / DMA channels. Each channel has the capability to transfer data between the interfaces. The channels:</p> <ul style="list-style-type: none"> <li>■ Support chaining via linked-lists of descriptors</li> <li>■ Move data from source interface to destination interface</li> <li>■ Support increment or hold of source and/or destination address</li> <li>■ Support XOR operation on up to eight source blocks, useful for RAID application</li> <li>■ Support iSCSI CRC-32 calculation</li> </ul> <p>See <a href="#">Section 11, XOR Engine, on page 205.</a></p>
<b>Two-Wire Serial Interface (TWSI)</b>	<p>The device contains a single Two-Wire Serial Interface (TWSI) port that can be configured as either a master or a slave interface. This port can also be used for serial ROM initialization.</p> <p>The TWSI fully supports multiple TWSI master environments (clock synchronization, bus arbitration). The TWSI interface can be used for multiple applications such as a master to control other TWSI on board devices and to auto-load values from an external serial ROM device. It can be used as a slave for communication with some other TWSI masters</p> <p>See <a href="#">Section 12, Two-Wire Serial Interface (TWSI), on page 221.</a></p>
<b>UART Interface</b>	<p>The device supports a Universal Asynchronous Receiver/Transmitter (UART) Interface that consists of two Synopsis DW_16550 compatible UART ports.</p> <p>The UART interface integrates:</p> <ul style="list-style-type: none"> <li>■ Two pins for transmit and receive operations</li> <li>■ Two pins for modem control functions</li> </ul> <p>See <a href="#">Section 13, UART Interface, on page 228.</a></p>
<b>NAND Flash Interface</b>	<p>The device implements an 8-bit NAND Flash interface to boot from NAND Flash, or for any other non-volatile memory usage. The NAND Flash interface provides a glueless interface to CE care and CE don't care type NAND Flash devices.</p> <p>See <a href="#">Section 14, 8-bit NAND Flash Interface, on page 231.</a></p>



<b>SPI Serial Flash Interface</b>	The device implements an SPI interface for direct boot from external SPI flash memory. This interface operates at up to 41.6 MHz in the 88F6180 and 88F619x, and up to 50 MHz in the 88F6281.
	See <a href="#">Section 15, Serial Peripheral Interface (SPI), on page 236</a> .
<b>Audio I<sup>2</sup>S / S/PDIF Interface</b>	The 88F6180, 88F6192, and 88F6281 contain an I <sup>2</sup> S / S/PDIF interface for audio in and audio out: <ul style="list-style-type: none"><li>■ Either I<sup>2</sup>S / S/PDIF inputs can be active at one time</li><li>■ Both I<sup>2</sup>S or S/PDIF outputs can be simultaneously active (transferring the same PCM data)</li></ul> This interface supports the following I <sup>2</sup> S and S/PDIF specific features: <ul style="list-style-type: none"><li>■ I<sup>2</sup>S specific features:<ul style="list-style-type: none"><li>• Sample rates of 44.1/48/96 kHz</li><li>• I<sup>2</sup>S input and I<sup>2</sup>S output operate at the same sample rate</li><li>• 16/24-bit depths</li><li>• I<sup>2</sup>S in and I<sup>2</sup>S out support Independent bit depths (16 bit/24 bit)</li><li>• Supports plain I<sup>2</sup>S, right justified and left justified formats</li></ul></li><li>■ S/PDIF specific features:<ul style="list-style-type: none"><li>• Compliant to IEC 60958-1, IEC 60958-3, and IEC 61937 specifications</li><li>• Sample rates of 44.1/48/96 kHz</li><li>• 16/20/24-bit depths</li></ul></li></ul>
	See <a href="#">Section 16, Audio (I<sup>2</sup>S / S/PDIF) Interface (88F6180, 88F6192, and 88F6281 Only), on page 240</a> .
<b>SDIO Interface</b>	The device integrates an SD/SDIO/MMC host interface that operates at up to 50 MHz. This interface supports: <ul style="list-style-type: none"><li>■ 1-bit/4-bit SDMem, SDIO, and MMC cards</li><li>■ Hardware generate/check CRC on all command and data transaction on the card bus</li></ul> See <a href="#">Section 17, Secure Digital Input/Output (SDIO) Interface, on page 256</a> .
<b>MPEG Video / Transport Stream Interface (TS)</b>	The 88F6192 and 88F6281 implement an MPEG Video / TS interface of up to 80 Mbps. It is ISO/IEC 13818-1 standard compliant, supports any of the following modes: <ul style="list-style-type: none"><li>■ Parallel (8 bit) input</li><li>■ Parallel (8 bit) output</li><li>■ Two independent serial interfaces</li></ul> See <a href="#">Section 18, Transport Stream (TS) Interface (88F6192 and 88F6281 Only), on page 262</a> .
<b>General-Purpose I/O Port (GPIO)</b>	<b>88F6180</b> provides a 30-bit general-purpose I/O port <b>88F619x</b> provides a 36-bit general-purpose I/O port <b>88F6281</b> provides a 50-bit general-purpose I/O port Each of these general-purpose I/O pins can be used for peripheral functions or for general-purpose I/O. <ul style="list-style-type: none"><li>■ Each pin can be configured independently</li><li>■ GPIO inputs can be used to register interrupts from external devices, and generate maskable interrupts</li></ul> See <a href="#">Section 19, General-Purpose I/O (GPIO) Port Interface, on page 277</a> .

<b>Real-Time Clock (RTC)</b>	The device integrates a real-time clock that records second, minute, hour, date, day, month, and year.  While the system power is off, a backup battery (1.5V–1.8V) can operate the RTC unit. The RTC unit operates with an external 32.768 kHz crystal.  See <a href="#">Section 20, Real-Time Clock (RTC) Unit, on page 278</a> .
<b>Interrupt Controller</b>	The device integrates an advanced interrupt controller that handles maskable interrupts from all the various sources and forwards them to the Sheeva™ CPU core.  In Endpoint mode, the interrupts can also be forwarded to the Endpoint PCI Express interface.  The Sheeva CPU core has two interrupt inputs—low and high priority. Each of the chip interrupt events can be assigned to one of these two interrupts.  See <a href="#">Section 21, Interrupt Controller, on page 280</a> .
<b>Timers</b>	The device contains two general-purpose, 32-bit timers, and a single 32-bit watchdog timer.  See <a href="#">Section 22, Timers and Counters, on page 283</a> .
<b>Internal Architecture</b>	The device internal architecture is optimized for high-performance applications. It contains an Mbus-L bus for high-performance, low latency CPU core to DDR SDRAM connectivity and a proprietary Mbus architecture for I/O connectivity.  The internal architecture integrates: <ul style="list-style-type: none"><li>■ Advanced Mbus architecture</li><li>■ Dual port DDR SDRAM controller connectivity to both CPU and Mbus</li></ul> See <a href="#">Section 25, Internal Architecture, on page 312</a> .

## 1.3 Differences Between the 88F6180, 88F6190, 88F6192, and 88F6281 Devices

[Table 1](#) provides a list of the differences between the 88F6180, 88F6190, 88F6192, and 88F6281 devices. It also lists those interfaces that are the same in all of the devices.

**Table 1: 88F6180, 88F619x, and 88F6281 Device Differences and Similarities**

Feature	88F6180	88F6190	88F6192	88F6281
<i>Differences</i>				
<b>Sheeva™ CPU Core</b>	Running at 600 MHz or 800 MHz L2 cache at up to 400 MHz	Running at 600 MHz L2 cache at up to 300 MHz	Running at 800 MHz L2 cache at up to 400 MHz	Running at 1.0 GHz, 1.2 GHz, or 1.5 GHz L2 cache at up to 500 MHz
<b>TCLK frequency</b>	166 MHz			200 MHz



Table 1: 88F6180, 88F619x, and 88F6281 Device Differences and Similarities (Continued)

Feature	88F6180	88F6190	88F6192	88F6281
<b>DDR SDRAM Interface</b>	<ul style="list-style-type: none"><li>Up to 200 MHz clock frequency with an 400 MHz data rate</li><li>Supports two DRAM chip selects</li><li>Supports all DDR2 devices with densities up to 1Gb</li><li>Supports up to 16 open pages (page per bank)</li><li>Up to 512 MB total address space</li></ul>			<ul style="list-style-type: none"><li>Up to 400 MHz clock frequency with an 800 MHz data rate</li><li>Supports four DRAM chip selects</li><li>Supports all DDR2 devices with densities up to 2 Gb</li><li>Supports up to 32 open pages (page per bank)</li><li>Up to 2 GB total address space</li></ul>
<b>Time Division Multiplexing (SLIC/codec) Interface</b>		No		Yes <sup>1</sup>
<b>Serial ATA II (SATA II) Interface</b>	None	1 port		2 ports
<b>Gigabit Ethernet Interface</b>	1 GbE RGMII/MII/MMII port	1 Gigabit Ethernet port and 1 Fast Ethernet port <ul style="list-style-type: none"><li>Port0 RGMII, Port1 MII/MMII</li><li>Port0 GMII, Port1 N/A</li></ul>		2 GbE ports <sup>1</sup> <ul style="list-style-type: none"><li>Port0 RGMII, Port1 RGMII</li><li>Port0 RGMII, Port1 MII/MMII</li><li>Port0 MII/MMII, Port1 RGMII</li><li>Port0 GMII, Port1 N/A</li></ul>
<b>Audio S/PDIF / I2S Interface</b>	Yes	No		Yes <sup>1</sup>
<b>MPEG Video / Transport Stream Interface (TS)</b>		No		Yes <sup>1</sup>
<b>General-Purpose I/O Port</b>	30-bits	36-bits		50-bits
<b>Similarities</b>				
<b>PCI Express Interface</b>		Yes		
<b>USB 2.0 Interface</b>		Yes		
<b>Cryptographic Engine and Security Accelerator</b>		Yes		

**Table 1: 88F6180, 88F619x, and 88F6281 Device Differences and Similarities (Continued)**

Feature	88F6180	88F6190	88F6192	88F6281
XOR engine and DMA			Yes	
Two-Wire Serial Interface (TWSI)			1 port	
UART Interface			2 ports	
NAND Flash Interface			Yes	
SPI Serial Flash Interface			Yes	
SDIO Interface			Yes	
Real-Time Clock (RTC)			Yes	

1. The following interfaces are multiplexed:

- Audio
- TS
- TDM
- GbE port1 or GbE port0 in GMII mode (see the note below)

For the 88F6192, only one of these interfaces may be selected at a time.

For the 88F6281, only two of these interfaces may be selected at a time.



**Note**

- GbE port0 configured to GMII mode utilizes the pins of port0 and port1. This configuration uses the multiplex pins.
- When GbE port0 is configured to GMII mode, the port cannot support MII/MMII, due to multiplexing limitations.
- GbE port0 configured to RGMII or MII/MMII mode utilizes dedicated pins, and can be activated independently, regardless of the above multiplexed interfaces.

## 2 Address Map

The device has a fully programmable address map. There is a separate address map for each of the master units (units that can initiate transactions over the device Mbus).

- Sheeva™ CPU core address space
- PCI Express address space
- GbE MAC address space
- USB address space
- SATA address space (88F619x and 88F6281 only)
- XOR engine address space
- Security accelerator address space
- SDIO address space
- TDM (SLIC/codec) address space (88F6192 and 88F6281 only)
- Audio address space (88F6180, 88F6192, and 88F6281 only)
- TS address space (88F6192 and 88F6281 only)

Each of these interfaces includes programmable address windows that allow it to access different device resources. Each window can map up to 4 GB of address space.

**Note**

- Throughout this section, the term “BAR” means Base Address Register.
- Although each master has independent address windows, when a resource (e.g., DRAM M\_CS<sub>n</sub>[0]) is used by multiple masters, all masters must use the same address map for this resource. This means that all masters must use an identical address window setting for each resource.

### 2.1

### Sheeva™ CPU Core Address Decoding

The Sheeva CPU core address decoding map consists of 13 address windows as follows:

- Four windows dedicated for Sheeva CPU core access to the four DRAM chip selects
- One window for Sheeva CPU core access to the chip internal registers space
- Eight configurable windows for Sheeva CPU core access to the remainder of the chip resources

**DRAM windows** Each DRAM window can have from a minimum of 16 MB of address space up to a maximum of 4 GB of address space. Each window is defined by specific Base and Size registers. The Base and Size are 8-bits wide, corresponds to address bits[31:24]. The Size must be programmed as a set of 1s (staring from the LSB) followed by a set of 0s. The set of 1s defines the size. For example, if Size[7:0] is set to 0x0F, it defines a size of 256 MB (the number of 1s is 4,  $2^4 \times 16\text{ MB} = 256\text{ MB}$ ).

By default, each of the DRAM windows corresponds to a different DRAM chip select (M\_CS<sub>n</sub>[3:0]). However, each window can be set to support any of the DRAM chip selects. This feature provides more flexibility in mapping DRAM space.

**Registers space** The chip internal registers window has a 1 MB fixed size (It has a Base register only, but no Size register).

**NOTE:** Only part of this space is populated with registers.

**Configurable address windows** Each of the configurable address windows can have from a minimum of 64 KB of address space up to a maximum of 4 GB of address space. Each window is defined by a Window Base register and by a Window Control register's <Size> field. The Base and Size are 16-bits wide, corresponding to address bits[31:16]. The Size must be programmed as a set of 1s (staring from the LSB) followed by a set of 0s. The set of 1s defines the size. For example, if Size[15:0] is set to 0x03FF, it defines a size of 64 MB (the number of 1s is 10,  $2^{10} \times 64\text{ KB} = 64\text{ MB}$ ).

Address decoding starts with the address being compared with the values in the various Base Address registers. The Size sets which address bits are significant for the comparison. In the previous example of a 64 MB size, the CPU address bits[31:26] are compared against Base Address bits[15:10] (the Size masks address bits[25:0]). An address is considered as a window hit if it matches the Base Address register bits (the bits that are not masked by the Size).



**Note**

- Never program the Base and Size registers so that they result in an address window overlap.
- The address decoding scheme restricts the address window to a size of  $2^n$  and to a start address aligned to the window size.

Upon a hit in one of the configurable windows, the transaction is forwarded to a specific target interface specified by Window Control register's <Target> bits[7:4]), and with specific transaction attributes specified by Window Control register's <Attr> bits [15:8].

[Table 2](#) shows a summary of target units IDs and attributes.

**Table 2: Units IDs and Attributes—CPU**

Field	Description
Target Unit ID	0x0 = Reserved 0x1 = NAND flash, SPI flash, or bootROM 0x2 = Reserved 0x3 = Security Accelerator SRAM 0x4 = PCI Express 0x5–0xF = Reserved

**Table 2: Units IDs and Attributes—CPU (Continued)**

Field	Description
Attributes[7:0]	<p>If the target is the NAND flash, set the &lt;Attr&gt; field (bits [7:0]) to 0x2F.</p> <p>If the target device is the SPI flash, set the &lt;Attr&gt; field (bits [7:0]) to 0x1E.</p> <p>If the target device is the bootROM, set the &lt;Attr&gt; field (bits [7:0]) to 0x1D.</p> <p>If the target is PCI Express interface:</p> <ul style="list-style-type: none"> <li>• To generate I/O transactions, set the &lt;Attr&gt; field (bits [7:0]) to 0xE0.</li> <li>• To generate memory transactions, set the &lt;Attr&gt; field (bits [7:0]) to 0xE8.</li> </ul> <p>If the target is the Security Accelerator SRAM:</p> <ul style="list-style-type: none"> <li>• Bits[1:0]—Data swapping—set to 0x1.</li> <li>    00 = byte swap</li> <li>    01 = no swap</li> <li>    10 = byte and word swap</li> <li>    11 = word swap</li> <li>• Bits[7:2] = Reserved. Must be 0.</li> </ul>

Each of the four DRAM windows and each of the configurable windows have an enable bit (Size register's <En> bit[0]).

- When set to 1, the window is enabled.
- When set to 0, the window is disabled and not taking part in the address decoding process.

The device internal registers space has a fixed size of 1 MB, even though only part of this space is really populated with chip internal registers. Upon a write to a non-implemented register, the data is discarded. A read to a non-implemented register will return undefined data.

The registers are located in different units of the device (distributed register file). Therefore, ordering is not guaranteed upon CPU back-to-back writes to different registers. If ordering is required, perform a read after each write.

For Sheeva core address decoding registers and their default values, see [Section A.3.1, CPU Address Map Registers, on page 357](#)).



**Note** Access to internal registers is limited to the WORD boundary (There is no support of burst access to the register files). This means that the register file space must never be cacheable.

## 2.1.1

### Sheeva™ CPU Core-to-PCI Express Address Remapping

The device supports address remapping on the Sheeva CPU core accesses to the PCI Express interface. This enables relocating a CPU-to-PCI Express address window to a new location in the PCI Express address space, decoupling the Sheeva CPU core and the PCI Express memory allocation.

Each of the configurable address windows has an associated Remap register. Upon a hit in one of these windows, the upper bits of the CPU address are replaced by the corresponding bits of the Remap Low register, before the data is transferred to the PCI Express interface. The number of bits to be replaced is determined according to the Size register. For example, with a 64-MB window, the CPU address bits[31:26] are replaced with bits[31:26] of the Remap Low register.

Each of these windows also has a 32-bit Remap High register that may be used for 64-bit addressing on the PCI Express interface. When this register is not set to 0, a CPU address hit in this window results in the PCI Express master generating a 64-bit addressing transaction.

## 2.1.2

### CPU Address Decoding Errors

When an address decoding error occurs (for example, no hit in any of the address windows, or an attempt to burst to an internal registers space):

1. An interrupt is set.
2. If it is a write transaction, the data is discarded.  
If it is a read, dummy data is driven back to the Sheeva CPU core, with an erroneous data indication (resulting in a CPU exception).

The SDRAM address decoding windows also supports the write-protection feature. A CPU attempt to write to a write-protected memory space is discarded and treated as the other address decoding errors described above.

## 2.2

### TDM (SLIC/Codec) Address Map (88F6192/88F6281 Only)

The TDM (SLIC/codec) interface address map consists of four programmable address windows for the different interfaces (see [Section A.5, Time Division Multiplexing \(TDM\) Unit Registers, on page 413](#)).



**Note**

The TDM port is restricted to access only the SDRAM and the PCI Express interfaces. Setting the TDM port address decoding windows differently results in unpredictable behavior.

## 2.3

### PCI Express Address Decoding

The PCI Express port has its own address map. The PCI Express interface address map consists of three Base Address registers (BARs) that map the chip address space. One BAR is dedicated for the chip internal registers space. The other two BARs are further sub-decoded by six programmable address windows to the different interfaces of the chip.

The three BARs are 64-bit BARs. The internal registers space has a fixed size of 1 MB. The other two BARs have corresponding size registers. Each programmable address window can map from a minimum of 64 KB of address space up to a maximum of 4 GB of address space.

PCI Express address decoding is similar to the CPU address decoding scheme. An address is considered as window hit if it matches the Base Address register bits. These are the bits not masked by the Size register.



**Note**

- Do not program the Base and Size registers so that they result in an address window overlap.
- The PCI Express address decoding scheme restricts the address window to a size of  $2^n$ , and to a start address that is aligned to the window size.

Each of the two programmable BARs has an enable bit. If the bit is enabled, the BAR can be configured to couple with one of the address windows. If a BAR is disabled, no address decoding is performed to it.

Upon an address hit, the address is further sub-decoded to any of the six address windows that are configured to the specific BAR. Each of these windows also has Base and Size registers. Based on this address decoding, the transaction is forwarded to a specific target interface (e.g., SDRAM controller) with specific transaction attributes (e.g., M\_CS[0]). [Table 3](#) shows a summary of target units IDs and attributes.

**Table 3: Unit IDs and Attributes—PCI Express**

Field	Description
Target Unit ID	0x0 = DRAM controller 0x1 = NAND Flash, SPI Flash, or boot ROM 0x2 = Reserved 0x3 = Security accelerator SRAM 0x4 = PCI Express 0x5–0xF = Reserved
Attributes[7:0]	If the target is DRAM controller: <ul style="list-style-type: none"> <li>• To access M_CSn[0], set the &lt;Attr&gt; field (bits [7:0]) to 0xE.</li> <li>• To access M_CSn[1], set the &lt;Attr&gt; field (bits [7:0]) to 0xD.</li> <li>• To access M_CSn[2], set the &lt;Attr&gt; field (bits [7:0]) to 0xB (88F6281 only).</li> <li>• To access M_CSn[3], set the &lt;Attr&gt; field (bits [7:0]) to 0x7 (88F6281 only).</li> </ul> If the target is the NAND Flash, set the <Attr> field (bits [7:0]) to 0x2F. If the target device is the SPI Flash, set the <Attr> field (bits [7:0]) to 0x1E. If the target device is the bootROM, set the <Attr> field (bits [7:0]) to 0x1D. If the target is PCI Express interface: <ul style="list-style-type: none"> <li>• To generate I/O transactions, set the &lt;Attr&gt; field (bits [7:0]) to 0xE0.</li> <li>• To generate memory transactions, set the &lt;Attr&gt; field (bits [7:0]) to 0xE8.</li> </ul> If the target is the Security Accelerator SRAM, set the <Attr> field (bits [7:0]) to 0x1.

### 2.3.1

### PCI Express-to-Memory Address Remapping

The device supports PCI Express address remapping. Each of the six PCI Express address windows has an associated Remap register. Upon a hit in one of these windows, the upper bits of the address in that PCI Express Window register, are replaced by the corresponding bits of the Remap register before being transferred to the target interface. The number of bits to be replaced is determined according to the Size register.

Each Remap register has an enable bit. If disabled, no remap action takes place, and the original address is transferred to the destination.

### 2.3.2

### PCI Express Address Decoding Errors

If the device PCI Express port receives a transaction that does not match any of the BARs:

1. An interrupt is set and the error is registered.
2. The transaction is terminated as an unsupported request.

If the device PCI Express port receives a transaction that hits one of the BARs, but does not match any of the sub-decoding windows:

1. An interrupt is set and the error is registered.
2. The transaction is forwarded to a default target, as defined in the PCI Express Default Window Control Register ([Table 263 p. 447](#)).



Note

Set the PCI Express Default Window Control Register to point to a dummy target device, so no destructive operation is performed due to the address decoding error.

## 2.4

### SATA Address Decoding (88F619x/88F6281)

The SATA port DMA uses an address decoding logic consisting of four address windows. The address decoding scheme is the same as the Gigabit Ethernet address decoding logic.



Note

The SATA port is restricted to access only the SDRAM and the PCI Express interfaces. Setting SATA port address decoding windows differently results in unpredictable behavior.

## 2.5

### Gigabit Ethernet Address Decoding

The Gigabit Ethernet port address decoding logic consisting of six address windows. Whenever the port's SDMA generates a read or a write transaction (for example, fetch descriptor), the address is compared against these address windows, to determine which interface must be accessed.

The address decoding scheme is the same as for the XOR/DMA logic, with one exception. For an address miss match, the SDMA transaction is retargeted to a fixed address and to the target interface, as defined in the Ethernet Unit Default Address (EUDA) Register ([Table 403 p. 555](#)).



Note

The GbE port is restricted to access only the SDRAM interface. Setting the GbE port address decoding windows differently, results in unpredictable behavior.

## 2.6

### USB Address Decoding

Each USB port uses an address decoding logic consisting of four address windows. Whenever one of the ports generates a read or a write transaction (for example, fetch descriptor), the address is compared against these address windows, to determine which interface must be accessed.



Note

The USB port is restricted to access only the SDRAM interface. Setting the USB port address decoding windows differently results in unpredictable behavior.

## 2.7

### Security Accelerator Address Decoding

The Security Accelerator DMA uses an address decoding logic consisting of four address windows. The address decoding scheme is the same as the Gigabit Ethernet address decoding logic.

**Note**

The Security accelerator DMA is restricted to access only the SDRAM interface. Setting the security accelerator DMA address decoding windows differently results in unpredictable behavior.

## 2.8

### XOR Engine Address Decoding

The two XOR engines each contain two XOR/DMA channels for a total of four XOR/DMA channels. The two XOR engines share a single address decoding logic consisting of eight address windows. Each of these windows is defined by a Base and a Size register. Each window can map from a minimum of 64 KB of address space up to a maximum of 4 GB of address space.

Each of the eight windows can be configured to a specific target interface, and to specific transaction attributes as shown in [Table 3](#). Whenever one of the ports generates a read or a write transaction (for example, fetch descriptor), the address is compared against these address windows, to determine which interface must be accessed.

Four of the eight windows also have a Remap High register. Use these registers to generate an address beyond the standard 4-GB space. This is useful for 64-bit PCI Express addressing.

When a DMA channel attempts to access an unmapped address, an interrupt is set, the error status is registered, and the channel halts.

**Note**

The XOR ports are restricted to access only the SDRAM and the PCI Express interfaces. Setting the XOR port address decoding windows differently results in unpredictable behavior.

## 2.9

### TWSI Address Decoding

The device TWSI serial ROM initialization and the TWSI debug port allow for access to the chip's internal resources (see the Reset Pins and Configuration section in the *Hardware Specifications*).

The serial ROM initialization and the debug port access to the chip resources is composed of a 32-bit address followed by 32-bit data. Bit[0] of the transaction address must be set to 0.

## 2.10

### Audio Interface Address Map (88F6180/88F6192/88F6281 Only)

The Audio interface address map consists of two programmable address windows for the different interfaces (see [Section A.16, Audio Interface Registers, on page 689](#)).

**Note**

The Audio port is restricted to access only the SDRAM and the PCI Express interfaces. Setting the Audio port address decoding windows differently results in unpredictable behavior.

## 2.11

### SDIO Address Map

The SDIO interface address map consists of four programmable address windows for the different interfaces (see [Section A.17, SDIO Registers, on page 714](#)).

**Note**

The SDIO port is restricted to access only the SDRAM and the PCI Express interfaces. Setting the SDIO port address decoding windows differently results in unpredictable behavior.

**2.12****Transport Stream (TS) Address Map  
(88F6192/88F6281 Only)**

The TS interface address map consists of four programmable address windows for the different interfaces (see [Section A.18, Transport Stream \(TS\) Registers, on page 744](#)).

**Note**

- The TS port is restricted to access only the SDRAM and the PCI Express interfaces. Setting the TS port address decoding windows differently results in unpredictable behavior.
- The SDRAM access from TS unit must not cross 32-byte boundary.

**2.13****Default Address Map**

[Table 4](#) lists the default target address map for the device, including each target interface address ID and attribute value.

**Table 4: Device Default Address Map**

Target Interface	Target Interface ID	Target Interface Attribute	Address Space Size	Address Range in Hexadecimal
DDR SDRAM CS0	0	0x0E	256 MB	0000.0000–0FFF.FFFF
DDR SDRAM CS1	0	0x0D	256 MB	1000.0000–1FFF.FFFF
DDR SDRAM CS2	0	0x0B	256 MB	2000.0000–2FFF.FFFF
DDR SDRAM CS3	0	0x07	256 MB	3000.0000–3FFF.FFFF
Reserved			–	4000.0000–7FFF.FFFF
PCI Express Memory	4	0xE8	512 MB	8000.0000–9FFF.FFFF
Reserved			512 MB	A000.0000–BFFF.FFFF
PCI Express I/O	4	0xE0	64 KB	C000.0000–C001.FFFF
Reserved			–	C001.0000–C001.FFFF
Reserved			–	C002.0000–C800.FFFF
Security Accelerator Internal SRAM Memory	3	0x00	64 KB	C801.0000–C801.FFFF <b>NOTE:</b> Only 2-KB SRAM is implemented.
<b>NOTE:</b> There is no access to the Security Accelerator Internal SRAM Memory from the PCI Express interface.				



Table 4: Device Default Address Map (Continued)

Target Interface	Target Interface ID	Target Interface Attribute	Address Space Size	Address Range in Hexadecimal
Reserved			—	C802.0000–CFFF.FFFF
Internal Address Space <sup>1</sup>			1 MB	D000.0000–D00F.FFFF
Reserved			—	D010.0000–D7FF.FFFF
NAND Flash	1	0x2F	128 MB	D800.0000–DFFF.FFFF
Reserved			—	E000.0000–E7FF.FFFF
SPI Serial Flash	1	0x1E	128 MB	E800.0000–EFFF.FFFF
BootROM	1	0x1D	—	F000.0000–F7FF.FFFF
Boot device <sup>2</sup> (set by bootstrap): • NAND flash • SPI Serial flash • BootROM	1	Set by bootstrap: 2F or 1E or 1D	128 MB	F800.0000–FFFF.FFFF

1. For the device Internal Address Map, see [Table 88, Device Internal Registers Address Map, on page 354](#).

2. The actual default is determined by the selected boot option (see the *Boot Mode* in the Reset Configuration table in the *Hardware Specifications*).

# 3 Sheeva™ CPU Core

The device uses the Marvell® Sheeva™ 88SV131 CPU core, compliant with the ARMv5TE architecture. This Sheeva CPU core<sup>1</sup> integrates a 256-KB L2 cache.

For full details and specifications about the Sheeva CPU core refer to the following documents:

- *Sheeva™ 88SV131 ARM v5TE Processor Core with MMU and L1/L2 Cache Datasheet*  
(Doc No. MV-S104950-00)
- *Unified Layer 2 (L2) Cache for Sheeva™ CPU Cores Addendum* (Doc. No. MV-S104858-00).

---

1. In this document, the Sheeva™ 88SV131 CPU core is often referred to as the CPU.

# 4

## DDR SDRAM Controller

The device integrates an DDR SDRAM (Double Data Rate-Synchronous DRAM) controller that supports up to four DRAM banks (four DRAM chip selects). It incorporates an 18-bit address bus (M\_A[14:0] and M\_BA[2:0]) and a 16-bit data bus (M\_DQ[15:0]).

The device supports 256 Mb, 512 Mb, 1 Gb, and 2 Gb DDR2 SDRAM devices, with up to 2 GB total address space.

The DRAM can be accessed from any of the device interfaces. The DRAM controller supports up to a 128-byte burst per single transaction from the Mbus port and up to a 32-byte burst from the Mbus-L port. It supports DRAM bank interleaving, as well as open pages (up to eight pages per chip select). Typically, this is useful on long DMA bursts to/from the DRAM.

The following optional DDR2 features are not supported:

- Additive latency
- Separate read and write DQS (RDQS signal)
- Off Chip Driver (OCD) Impedance Adjustment
- Power Down mode
- Burst Length (B)L 8
- Frequency change during self refresh

### 4.1

## SDRAM Controller Implementation

The DRAM controller receives read and write requests from any of the chip units through the device Mbus fabric, or from the CPU via Mbus-L path, and translates these requests to DDR SDRAM transactions.

The SDRAM controller contains a transaction queue, read and write buffers. It can absorb up to four transactions of 128 byte each, in its buffers.

Transactions from the Mbus are pushed into the transaction queue. The SDRAM controller arbitrates between the transaction from the top of the queue and transactions received from the CPU Mbus-L path. It drives part of the address bits of the selected transaction on M\_A[14:0] and M\_BA[2:0] during the activate cycle (M\_RASn), and the remaining bits during the command cycle (M\_CASn).

#### 4.1.1

### Write Data Path

For a write transaction, write data coming from the requesting unit is placed in the write buffer. Use of the write buffer is required to compensate for the data-rate differences between the received write data rate (running at core clock domain) and the rate that it is driven to the DRAM (DRAM clock domain, double data rate).

The SDRAM interface write data path is 32b wide. Write data received from the 64b-wide Mbus is unpacked to 32b, and driven on the 16b SDRAM interface at double data rate.

#### 4.1.2

### Read Data Path

For a read transaction, after the command cycle (M\_CASn), the SDRAM controller samples read data driven by the DRAM (sample window depends on the CAS Latency (CL) parameter), pushes the data into the read buffer, and drives it back to the requesting unit. Use of the read buffer is required to compensate for the data rate differences between the received read data from the DRAM

(running at DRAM clock domain, double data rate) and data rate of the requesting unit (core clock domain). It is also used for temporary storage of read data, when the originator unit can not absorb this data.

For a CPU read from the DRAM, read data is not pushed to the read buffer. It goes directly to the CPU bus interface unit via a 64-bit wide Mbus-L path. This minimizes read latency.

The SDRAM interface read data path is 32 bits wide. Read data received from the 16b SDRAM interface at double data rate is packed to the 64b-wide Mbus. Similarly, for CPU reads, read data received from DRAM is packed to the 64b-wide Mbus-L.

#### 4.1.3

### Arbitration and Ordering

Transactions coming from the device Mbus fabric are pushed into a transaction queue. The SDRAM controller arbitrates between the transaction at the top of the queue and transactions coming from CPU Mbus-L path, always giving priority to the CPU.

While serving one transaction, the arbiter selects the next transaction to be served (from CPU or from Mbus). When the next transaction is targeted to a different SDRAM bank, the SDRAM controller utilizes its SDRAM bank interleaving capability (see [Section 4.5, SDRAM Bank Interleaving, on page 48](#)). When the next transaction is targeted to the same page, in the same bank, the SDRAM controller utilizes its open pages capability (see [Section 4.6, SDRAM Open Pages, on page 49](#))

The SDRAM controller transaction queue maintains transaction ordering between the source unit over the device Mbus and the DRAM (no transactions re-ordering).

When receiving a CPU-to-DRAM transaction over the Mbus-L path, the SDRAM controller performs an address lookup, against pending Mbus write transactions to the DRAM. When an address match occurs, the CPU transaction is postponed until the matched Mbus transaction is forwarded to the DRAM. This lookup mechanism guarantees proper producer-consumer operation.

The CPU also supports LOCK transactions. Upon receiving a LOCK transaction on the Mbus-L path, the SDRAM controller blocks any pending Mbus transaction, until LOCK de-assertion occurs.

## 4.2

### DDR SDRAM Addressing

The device supports 256-Mb, 512-Mb, 1-Gb, and 2-Gb DDR2 SDRAM devices. The different DRAM devices differ in the usage of M\_A[14:0] and M\_BA[2:0] lines, as shown in [Table 5](#).

**Table 5: DDR2 DRAM Addressing**

DRAM Type		Bank Address	Row Address	Column Address	Auto Precharge
256 Mb	32Mx8	M_BA[1:0]	M_A[12:0]	M_A[9:0]	M_A[10]
	16Mx16	M_BA[1:0]	M_A[12:0]	M_A[8:0]	M_A[10]
512 Mb	64Mx8	M_BA[1:0]	M_A[13:0]	M_A[9:0]	M_A[10]
	32Mx16	M_BA[1:0]	M_A[12:0]	M_A[9:0]	M_A[10]
1 Gb	128Mx8	M_BA[2:0]	M_A[13:0]	M_A[9:0]	M_A[10]
	64Mx16	M_BA[2:0]	M_A[12:0]	M_A[9:0]	M_A[10]
2 Gb	256Mx8	M_BA[2:0]	M_A[14:0]	M_A[9:0]	M_A[10]
	128Mx16	M_BA[2:0]	M_A[13:0]	M_A[9:0]	M_A[10]



- Note**
- The Bank Address is the same in both RAS and CAS cycles.
  - An auto-precharge indication, during the CAS cycle, is driven on M\_A[10].

The SDRAM controller supports up to four SDRAM physical banks (four SDRAM chip selects). The total SDRAM bank address space is determined by the nature of the DDR SDRAM devices. For example, when using 512 Mb x8 devices (64Mx8), the bank, built up of eight such devices (a 64b SDRAM interface), has a 512-MB address space.

#### 4.2.1 DDR SDRAM Address Multiplex

The <CSxAddrSel> fields in the SDRAM Address Control Register ([Table 172 p. 398](#)) define how the address bits driven by the requesting unit to the SDRAM controller are translated to row and column address bits on M\_DA[14:0] and M\_BA[2:0].

The row and column address translation is different for 256-Mb, 512-Mb, 1-Gb, or 2-Gb SDRAM densities, as well as for x8 or x16 SDRAM organization (see [Table 6](#) and [Table 7](#)).

**Table 6: Address Multiplex for 16b Interface, AddrSel = 0**

SDRAM Configuration		M_BA[2:0]	Row M_A[14:0]	Column M_A[14:0]
256 Mb	32Mx8	12:11	25:13	10:1
	16Mx16	12:11	24:13,10	9:1
512 Mb	64Mx8	12:11	26:13	10:1
	32Mx16	12:11	25:13	10:1
1 Gb	128Mx8	13:11	27:14	10:1
	64Mx16	13:11	26:14	10:1
2 Gb	256Mx8	13:11	28:14	10:1
	128Mx16	13:11	27:14	10:1

**Table 7: Address Multiplex for 16b Interface, AddrSel = 1**

DRAM Configuration		M_BA[2:0]	Row M_A[14:0]	Column M_A[14:0]
256 Mb	32Mx8	25:24	23:11	10:1
	16Mx16	24:23	22:10	9:1
512 Mb	64Mx8	26:25	24:11	10:1
	32Mx16	25:24	23:11	10:1
1 Gb	128Mx8	27:25	24:11	10:1
	64Mx16	26:24	23:11	10:1

**Table 7: Address Multiplex for 16b Interface, AddrSel = 1 (Continued)**

DRAM Configuration		M_BA[2:0]	Row M_A[14:0]	Column M_A[14:0]
2 Gb	256Mx8	28:26	25:11	10:1
	128Mx16	27:25	24:11	10:1

By default, all four physical banks (M\_CS<sub>n</sub>[3:0]) must be populated with the same DRAM devices (the same density and configuration). However, the device also supports having a different DRAM configuration for each of M\_CS<sub>n</sub>[3:0]. This is especially useful for systems that support memory expansion.

## 4.3 SDRAM Timing Parameters

The SDRAM controller supports a wide range of SDRAM timing parameters. These parameters can be configured through the following registers:

- SDRAM Timing (Low) Register ([Table 170 p. 397](#))
- SDRAM Timing (High) Register ([Table 171 p. 397](#))
- SDRAM Mode Register ([Table 175 p. 401](#))



The DRAM controller does not support different timing parameters for each physical bank.

**Table 8: SDRAM Timing Parameters**

SDRAM Timing Parameters	Description
CAS Latency (CL)	The number of cycles from M_CAS <sub>n</sub> assertion to the sampling of the first read data. The SDRAM controller supports a CL of 3, 4, 5, 6, or 7 cycles.
RAS Precharge (Trp)	The minimum number of cycles from precharge to a new activate cycle, in the same SDRAM bank.
M_RAS <sub>n</sub> to M_CAS <sub>n</sub> (Trcd)	The minimum number of cycles between activate cycle and command cycle, in the same SDRAM bank.
Row Active Time (Tras)	The minimum number of cycles between activate cycle and precharge cycle, in the same SDRAM bank.
Write to Precharge (Tw <sub>r</sub> )	The minimum number of cycles between a write command and precharge, in the same SDRAM bank.
Write to Read (Tw <sub>tr</sub> )	The minimum number of cycles between a write command and a read command in the same SDRAM device
Active to Active (Trrd)	The minimum number of cycles between activate bank A and activate bank B (in the same SDRAM device)
Refresh Command (Trfc)	The minimum number of cycles between a refresh command and a new activate command

**Table 8: SDRAM Timing Parameters**

SDRAM Timing Parameters	Description
Read to Read (Tr2r)	The minimum number of cycles between consecutive read commands to different devices. This is not part of the JEDEC specification. This is used to prevent contention between consecutive reads to different SDRAM devices (different chip selects). The SDRAM controller supports Tr2r of 1 or 2 cycles.
Read to Write and Write to Read (Tr2w_w2r)	The minimum number of cycles between a read command and a write command. This is not part of the JEDEC specification. This is used to prevent contention between consecutive read after write or write after read commands. The SDRAM controller supports Tr2w_w2r of 1 or 2 cycles.

## 4.4 DRAM Burst

A DDR SDRAM device can be configured to different burst lengths (BL) and burst ordering. The device SDRAM controller only supports a BL setting of four, and linear wraparound burst type (<BT> field in the SDRAM Mode Register ([Table 175 p. 401](#)) must be set to 0).

A single DRAM access request in the device SDRAM controller can vary from a single byte up to a 128-byte burst (a burst of 64 16-bit words). The SDRAM controller drives the SDRAM address and control signals accordingly, concatenating multiple BL accesses as a one, long burst.

Even when the required DRAM access is not a full multiple of the DRAM burst lengths, the SDRAM controller always completes the burst to the next BL boundary. In a write transaction, the controller masks the redundant cycles with a data mask.

### 4.4.1 Burst Chop Support

The CPU maximum transaction size is 32B (cache line) while an Mbus request can be as long as 128B. This fact represents a fairness issue on the CPU versus Mbus arbitration scheme. Since the CPU read access is latency-sensitive, waiting for an entire 128B Mbus transaction to complete before serving a CPU read request can result in a CPU performance penalty.

To resolve this conflict, the SDRAM controller supports a burst-chop feature, where the SDRAM controller splits every Mbus transaction on the 32B boundaries (A 128B transaction is split into four 32B transactions). If the SDRAM controller receives a CPU request while in the middle of serving an Mbus transaction, it forwards the CPU transaction in between the 32B segments of the Mbus transaction.



The burst-chop feature does not introduce any performance penalty on the Mbus transactions, as long as there is no CPU transaction interference.

## 4.5 SDRAM Bank Interleaving

The device supports both physical bank (M\_CS[3:0]) interleaving and virtual bank (M\_BA[2:0]) interleaving.

Interleaving provides higher system performance by hiding a new transaction's active cycles during a previous transaction's data cycles. This technique gains maximum utilization of the SDRAM-bus bandwidth.

The SDRAM controller performs bank interleaving between the current active transaction and the next transaction to be executed, if it is targeted to a different bank.

Proper selection of SDRAM address multiplexing, via the SDRAM Address Control Register ([Table 172 p. 398](#)), can sometimes increase the probability of virtual bank interleaving.

## 4.6

### SDRAM Open Pages

It is possible to configure the device SDRAM controller to keep SDRAM pages open, using the SDRAM Open Pages Control Register ([Table 173 p. 400](#)). It supports up to 32 open pages (one page per each virtual bank).

When a page is kept open at the end of a burst (no precharge cycle), and if the next cycle to the same virtual bank hits the same page (same row address), there is no need for a new activate cycle. This is typically useful for large DMA transfers to/from the SDRAM. Once a page is open, it is kept open until one of the following events occur:

- There is an access to the same bank but to a different row address. The SDRAM controller performs a precharge (closes the page) and opens a new one (the new row address).
- The refresh counter expires. The SDRAM controller closes all open pages and performs a refresh to all banks.

## 4.7

### SDRAM Refresh

The device implements standard CAS before RAS refreshing.

The refresh rate for all banks is determined according to the 14-bit Refresh value in the SDRAM Configuration Register ([Table 168 p. 393](#)). For example, when the value of <Refresh> is 0x200, if the M\_CLK\_OUT frequency is 166 MHz (a 6-ns cycle), a refresh sequence occurs every 3.072 us.

Every time the refresh counter reaches its terminal count, a refresh request is sent to the SDRAM controller. The refresh request has a higher priority than any other SDRAM access request. As soon as the current outstanding SDRAM transactions complete, the SDRAM controller precharges all banks (both the ones that are opened, and those that are not open), and performs an auto-refresh command to all SDRAM banks.

## 4.8

### SDRAM Initialization

The SDRAM controller starts the DDR SDRAM initialization sequence as soon as the <InitEn> field in the SDRAM Initialization Control Register ([Table 184 p. 407](#)) is set to 1. The software must initialize the DDR SDRAM Control registers prior setting this field.

The <InitEn> can be set only once by the CPU. To change the SDRAM mode and SDRAM extended mode values after initialization has finished, see [Section 4.9, SDRAM Operation Register](#).

The SDRAM controller postpones any attempt to access the SDRAM before the initialization sequence completes. It is recommended that the CPU set the <Cmd> field in the SDRAM Operation Register ([Table 174 p. 400](#)) to the NOP command, and perform read polling until the register returns to a normal operation value.

The DDR SDRAM specification requires at least 200 us of stable clock after SDRAM power up, before starting the initialization. Since the SDRAM controller starts driving valid clock to the SDRAM only upon reset de-assertion, do not activate initialization earlier than 200 us after reset de-assertion.

The SDRAM initialization sequence consists of the following steps:

1. Precharge to all SDRAM banks (all four physical banks).
2. Issue the EMRS2 (EMRS = Extended Mode Register Set) command based on the Extended DRAM Mode 2 Register ([Table 185 p. 407](#)) value.
3. Issue the EMRS3 command based on the Extended DRAM Mode 3 Register ([Table 186 p. 407](#)) value.

4. Issue the EMRS command based on the Extended DRAM Mode Register ([Table 176 p. 402](#)) value, to enable the SDRAM DLL.
5. Issue MRS (Mode Register Set) command based on the SDRAM Mode Register ([Table 175 p. 401](#)) value, with the <DLLRst> activated.
6. Wait 200 cycles.
7. Precharge all banks.
8. Generate two auto-refresh cycles.
9. Issue the MRS command based on the [SDRAM Mode Register](#) value, and with the <DLLRst> de-activated.
10. Issue the EMRS command with the <OCD> field in the Extended DRAM Mode Register ([Table 176 p. 402](#)) set to 1111 (OCD Calibration Default), followed by another EMRS command with the <OCD> field set to 000 (OCD Calibration Exit).

## 4.9 SDRAM Operation Register

In addition to the normal SDRAM operation mode, the SDRAM controller also supports special SDRAM commands through the SDRAM Operation Register ([Table 174 p. 400](#)). These operations include:

- Normal SDRAM Mode (default mode)
- NOP Commands
- Precharge All Banks
- SDRAM Mode Register Setting (MRS)
- SDRAM Extended Mode Register Setting (EMRS)
- EMRS2
- EMRS3
- Force a Refresh Cycle
- Enter Self Refresh

The register contains four command type bits. Once the CPU changes the register default to one of the command types, the SDRAM controller executes the required command, resets the register back to the default value, and returns to normal operation. The CPU must poll this register to identify when the SDRAM controller has returned to normal operation mode.

When using DDR SDRAM DIMMs, the SDRAM parameters are recorded in the DIMM Serial Presence Detect (SPD) serial ROM. The CPU can read the SPD via the device TWSI interface and program the SDRAM parameters accordingly, using the Load Mode register command.

The CPU must not attempt to change the SDRAM Mode Register ([Table 175 p. 401](#)) setting prior to SDRAM controller completion of the SDRAM initialization sequence. To guarantee this restriction, it is recommended that the CPU set the SDRAM Operation Register ([Table 174 p. 400](#)) to the NOP command, perform read polling until the register has returned to a normal operation value, and then set [SDRAM Mode Register](#) to its new value.

## 4.10 SDRAM Self Refresh Mode

The SDRAM controller also supports SDRAM Self Refresh mode. This feature is useful for two applications:

- Power saving
- Battery backup (in case of power failure)

The SDRAM controller puts the SDRAM in Self Refresh mode by generating a refresh cycle with M\_CKEx driven low. When exiting self refresh, it drives M\_CKEx high, and waits for 200 cycles, before generating any new transaction to SDRAM.

The SDRAM controller supports these two applications a bit differently:

- For power saving set the [\*\*<SRMode>\*\*](#) to 1. Normal operation resumes after any new SDRAM access request.
- For battery backup, set the [\*\*<SRMode>\*\*](#) field in the SDRAM Configuration Register ([Table 168 p. 394](#)) to 0 (Entered Self Refresh mode). Once in Self Refresh mode, the SDRAM can no longer be accessed and only returns to normal operation after power on reset.



**Note**

There are four CKE signals (M\_CKE[3:0]) that behave the same. The device does not support the separate placement of each physical bank into self refresh. If the board topology does not require the use of all of these signals, it is possible to only use some of the signals.

During self refresh, all of the SDRAM signals (excluding M\_CLK\_OUT, M\_CKE, and M\_STARTBURST) are floated. This significantly reduces the power consumption. If the [\*\*<SRClk>\*\*](#) field in the DDR Controller Control (Low) Register ([Table 169 p. 394](#)) is set to 1, the device also stops driving M\_CLK\_OUT and M\_CLK\_OUTn when SDRAM is in Self Refresh mode.

The SDRAM controller keeps M\_CKE low from power up until the software triggers the initialization sequence.

When the SDRAM controller wakes up from reset, it does recognize that the SDRAM is in Self Refresh mode. It starts an initialization sequence, as if it was a normal power up. This initialization sequence has no effect on the SDRAM content.

#### 4.10.1 Power Saving Mode

To place the SDRAM into self refresh set the [\*\*<Cmd>\*\*](#) field in the SDRAM Operation Register ([Table 174 p. 400](#)) to 0x7. The SDRAM controller waits for 256 cycles and then generates a self refresh command to SDRAM, and clears the SDRAM Operation register.

If there are new pending transactions to SDRAM, the SDRAM controller sets the M\_CKE signals and waits 200 cycles before generating a new transaction to SDRAM.



**Note**

The SDRAM controller exits the Self Refresh mode only as a result of a SDRAM read/write access. Attempts to access the SDRAM with one of the operation commands (for example, the MRS command), while in Self Refresh mode, results in a system hang.

#### 4.10.2 Battery Backup Mode

In some applications, data loss is unacceptable and a battery-backup mechanism is implemented. In case the system detects a power failure, the SDRAM enters Self Refresh mode, just before power goes down, and the SDRAM is kept “alive” via the battery backup. This means the SDRAM content is not lost. It is available for re-use when power is restored.

The device does not tolerate powering of the I/O without powering the core. Since the SDRAM power is driven from the battery during battery backup, the board must have separate power planes to the SDRAM and to the device VDD\_M.

To implement battery backup, follow these steps:

1. When the external logic detects a power failure, it interrupts the CPU. The logic can use the device GPIO interrupts to perform this function.
2. The CPU interrupt handler sets the [\*\*<Cmd>\*\*](#) field in the SDRAM Operation Register ([Table 174 p. 400](#)) to 0x7.

3. The SDRAM controller waits for 256 cycles and generates a Self Refresh command to the SDRAM. This stops the SDRAM controller. It does not serve any pending SDRAM transactions.



The CPU may notify the external hardware, via a GPIO pin, that it has placed the SDRAM in Battery Backup mode.

**Note**

4. When the external logic turns the device power off, it must keep M\_CKE0/1 driven LOW. This keeps the SDRAM in Self Refresh mode while powered by the battery.
5. When the power is restored, the external logic stops driving the M\_CKE signals.
6. The device SDRAM controller drives any idle commands as long as SYSRSTn is asserted, and commences the SDRAM initialization sequence when reset is de-asserted. The initialization starts when the CPU enables the [<InitEn>](#) field in the SDRAM Initialization Control Register ([Table 184 p. 407](#)).

## 4.11 Heavy Load Support

When using multiple physical banks, the address and control signals are heavily loaded, and may be unable to meet the SDRAM AC timing requirements. By configuring one of the registers described below, the address/control signals are buffered and this AC timing issue is resolved.

- When using registers, all address and control signals (M\_A[14:0], M\_BA[2:0], M\_RASn, M\_CASn, M\_WEn, M\_CSn[3:0], and M\_CKE) arrive at the SDRAM device one cycle after they are driven by the SDRAM controller. Also read data arrives back at the SDRAM controller one cycle later.  
To enable this mode, set the [<RegDIMM>](#) field in the SDRAM Configuration Register ([Table 168 p. 393](#)).
- An alternative solution for the heavy load configuration is using 2T mode. In this mode, all address/control signals except for M\_CSn[3:0] are asserted for two cycles, instead of one cycle. The SDRAM protocol is still maintained, since all of the signals are qualified with M\_CSn[3:0] signals. These signals are still asserted for only one cycle. This operation results in an easing of the timing requirement on the address/control signals.  
To enable this mode, set the [<2T>](#) field in the DDR Controller Control (Low) Register ([Table 169 p. 394](#)).

## 4.12 SDRAM Clocking

The CPU Bus Interface Unit (BIU) and the SDRAM run from the same clock source (HCLK). HCLK is derived from the same PLL that also generates the CPU core clock (CPU\_CLK). The clocks are edge aligned, and the entire CPU to SDRAM path runs synchronously, resulting in very low latency.

As described, the device—besides the CPU and SDRAM interfaces—runs at a different clock domain (TCLK). Any request for SDRAM access from other units over the device Mbus passes through synchronization logic.

## 4.13 SDRAM Address/Data Drive

The SDRAM clock is driven by the device M\_CLK\_OUT/M\_CLK\_OUTn differential pair. All SDRAM address and control signals driven by the device (single data rate signals) are coupled to the rising or to the falling edge of this clock. The clock edge is configured by the [<CtrlPos>](#) field in the DDR Controller Control (Low) Register ([Table 169 p. 395](#)).



**Note**

Typically, address and control signals should be driven with the rising edge of M\_CLK\_OUT. However, under certain board topology and SDRAM load, there may be a hold time problem on these signals. Then, use the falling edge setting (0).

The front-end logic of the SDRAM controller is responsible for correctly driving the double data rate data with the M\_DQS signals, as well as unpacking the data from 32-bit SDR to 16-bit DDR.

During a write transaction, 32-bit wide data is pulled out of the write buffer and driven as 16-bit DDR on the bus. The first 16 bits are driven with rising edge of M\_CLK\_OUT and the second 16 bits are driven with the falling edge of M\_CLK\_OUT. The SDRAM controller drives DQS (data strobe) along with the data. The DDR SDRAM specification requires very accurate DQS timing in respect to the SDRAM clock. The SDRAM controller uses a Fine Tune DLL (FTDLL) and a delay line to achieve the correct timing (shift DQ by  $\frac{1}{4}$  cycle).

## 4.14 SDRAM Read Data Sample

The front-end logic of the SDRAM controller is responsible for correct sampling of the double data rate data with M\_DQS signals, as well as the packing of data from 16-bit DDR to 32-bit SDR. A 16-bit DDR read data is latched via the received DQS (shifted by  $\frac{1}{4}$  cycle, via the delay line). The first 16-bits are sampled with the rising edge of DQS, and the next 64-bits with the falling edge of DQS.

To meet the DDR SDRAM AC specification, packed 32-bit read data cannot simply be sampled with the internal SDRAM controller clock. The exact sample point depends on class latency, silicon process, and board topology. The controller drives a M\_STARTBURSTn signal, as an envelope of the read data phase. Route this signal on the board all the way to the SDRAM, and back to the controller as M\_STARTBURST\_INn feedback. This signal is used as a reference for proper data sample.

The assertion point of M\_STARTBURST is controlled by the [\*\*<SBOutDel>\*\*](#) field in the DDR Controller Control (Low) Register ([Table 169 p. 396](#)). The default setting of this field should be according to [Table 9](#).

**Table 9: M\_STARTBURST Output Assertion Point Configuration**

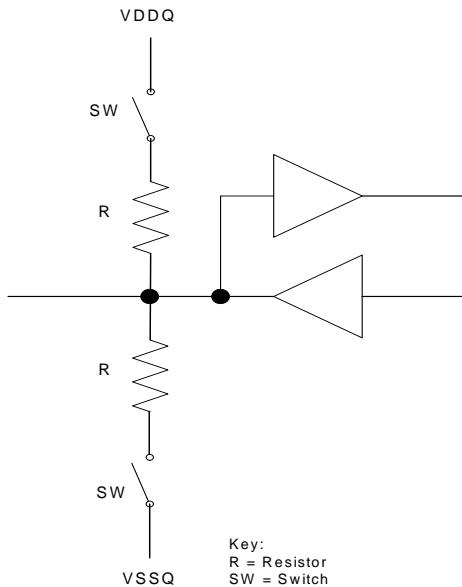
SDRAM Topology	CL=3	CL=4	CL=5	CL = 6	CL = 7
Unbuffered	0x1	0x3	0x5	0x7	0x9
Registered	0x3	0x5	0x7	0x9	0xB

The sampling point of the read data arriving from the SDRAM is determined by the [\*\*<SBInDel>\*\*](#) field in the DDR Controller Control (Low) Register ([Table 169 p. 396](#)).

## 4.15 DDR2 On Die Termination (ODT)

The DDR2 supports dynamic turn ON and OFF termination resistors within the SDRAM I/O buffers, as well as within the SDRAM controller I/O buffers. [Figure 5](#) shows a schematic of a DDR2 I/O buffer.

**Figure 5: DDR2 I/O Buffer**



The R nominal value is configurable to one of these values: 300, 150, or 100 ohm. This results in  $R_{tt}$  ( $R/2$ ) of: 150, 75, or 50 ohm, respectively.

The SDRAM controller has two ODT signals in the 88F619x and 88F6281 devices (M\_ODT[1:0]) and one ODT signal in the 88F6180 (M\_ODT). There is an additional ODT signal internal to the device that controls the termination inside the chip I/O buffers. The ODT signals can dynamically turn the SDRAM termination ON and OFF. This is useful for maintaining proper signal integrity, with minimum reflection on the lines, without requiring any external termination resistors.

The device supports ODT at each of the  $R_{tt}$  values—150, 75, or 50 ohm—on both the SDRAM I/O buffers and the device I/O buffers.



**Note**

As defined in the JEDEC specification, ODT applies only to the DM, DQ, and DQS signals. External termination is still required on the address/control signals.

The the termination value during ODT operation or disabling of termination is controlled by the following fields in the Extended DRAM Mode Register ([Table 176 p. 402](#)):

- [\*\*<Rtt\[0\]>\*\*](#)
- [\*\*<Rtt\[1\]>\*\*](#)

Typically, when driving a signal and eliminating reflection, place a termination resistor at the end of the line. When the device drives data on the DQ lines (write transactions), it is advisable to turn ON termination on the SDRAM. On the other hand, when the SDRAM drives DQ signals (read transactions), it is required to turn ON the termination inside the device I/O buffer.

In a multiple-SDRAM bank environment, termination topology is more complex, and requires some board simulation. The device SDRAM controller provides full flexibility to select which of the four

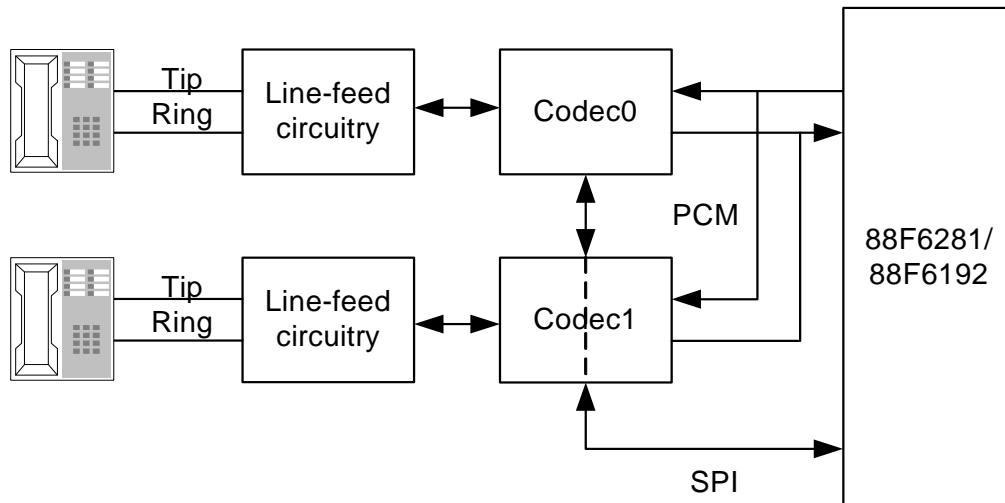
SDRAM banks terminations to turn ON or OFF, for any read or write transaction to any of the four banks.

# 5

## Time Division Multiplexing (TDM) Unit (88F6192 and 88F6281 Only)

The 88F6192 and 88F6281 devices integrate a Time Division Multiplexing (TDM) Unit that is used for interfacing with external SLIC (Subscriber Line Interface Circuit)/SLAC (Subscriber Line Audio-processing Circuit)/codec devices, as shown in [Figure 6](#). This interface is useful for VoIP (Voice over IP) applications.

**Figure 6: SLIC/Codec Connection Example**



The main features of the TDM unit are:

- Two voice channels (interfaces two SLIC devices)
- TDM interface with up to 128 full-duplex time slots; selectable time slot per each SLIC device
- SPI interface for SLIC/SLAC registers read/write access
- Support for various bit clock rates (256 kHz to 8.192 MHz in increments by powers of two).
- TDM as master or slave of frame sync and PCM (Pulse-Code Modulation) clock
- Support for compound (A-law/U-law) or linear voice samples
- Support for wideband voice channels
- Support for various flavors of PCM (short and long frame synchronization, inverted frame synchronization, positive-edge/negative-edge PCM data drive/sample and others)

### 5.1

### Functional Description

The TDM interface defines up to 128 full-duplex time slots (at 8.192 MHz PCM clock).

The TDM interface provides two independent transmit and two receive channels internally, via a

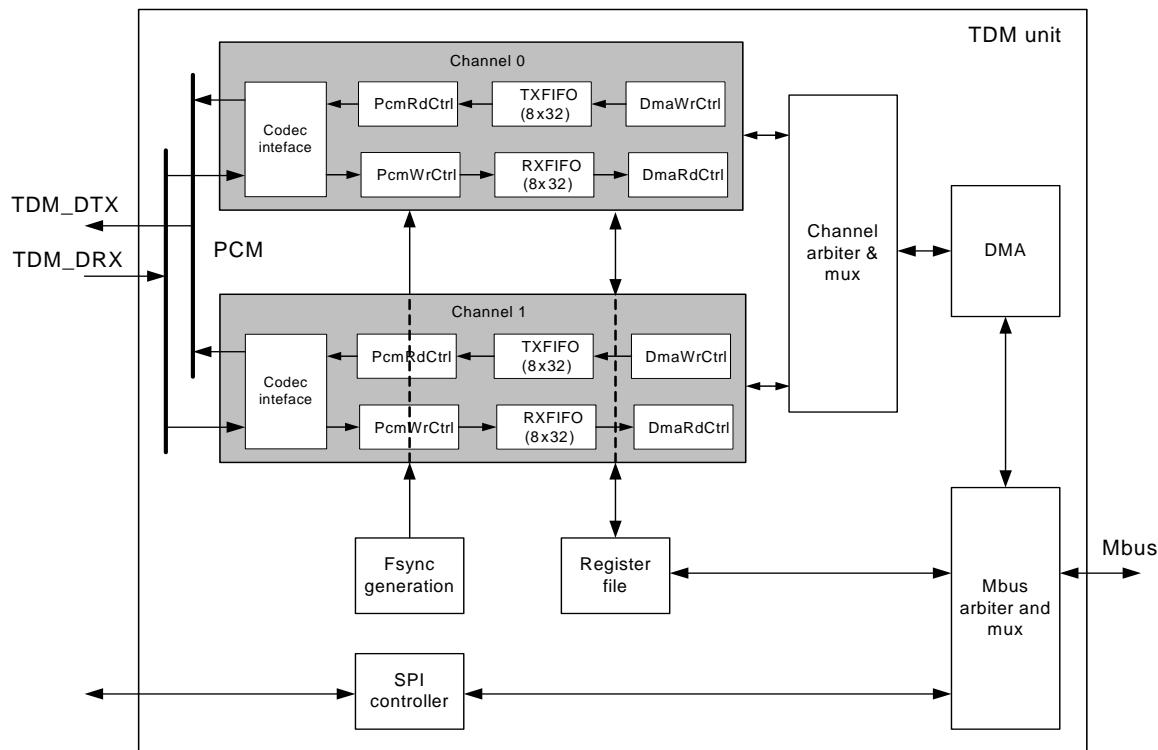
tightly integrated DMA engine (see [Figure 7](#)). These channels can be mapped onto one of the 128 (maximum) time slots contained in a frame, as defined in the control registers. CPU intervention is required only for overflow/underflow management and timely, buffer management.

The TDM interface supports the following functionality:

- PCM bus interface for telephony SLIC/SLAC/codec devices
- Internal register read/write access
- Interrupt generation to the internal CPU
- DMA operation to move data between the PCM bus and memory
- Codec control register interface (via the SPI interface)

**Figure 7: TDM Unit Block Diagram**

---



The size and location of the buffers are programmable through the registers. If the DMA engine fails to service the requirements of the TDM interface in a timely fashion (due to increased internal bus latencies), an underflow/overflow occurs, and the TDM interface generates a CPU interrupt and signals the error via the Interrupt Status Register ([Table 219 p. 427](#)) (ISR). The channel in which the error occurs is then switched off. In cases of Tx underflow, the TDM sends 0's on a programmed time slot. In cases of Rx overflow, it discards incoming data from the programmed time slot.

The TDM interface signals are provided in [Table 10](#).

**Table 10: Time Division Multiplexing (TDM) Interface Signals**

Pin Name	I/O	Description
TDM_CH0_TX_QL	O	TDM Channel0 Transmit Qualifier
TDM_CH2_TX_QL	O	TDM Channel2 Transmit Qualifier
TDM_CH0_RX_QL	O	TDM Channel0 Receive Qualifier
TDM_CH2_RX_QL	O	TDM Channel2 Receive Qualifier
TDM_CODEC_INTn	I	Interrupt Signal FROM the SLIC/codec
TDM_CODEC_RSTn	O	SLIC/codec Reset Signal
TDM_PCLK	I/O	PCM Audio Bit Clock
TDM_FS	I/O	TDM Frame Sync Signal
TDM_DRX	I	PCM Audio Input Data (for recording)
TDM_DTX	O	PCM Audio Output Data (for playback)
TDM_SPI_CS[1:0]	O	Active low SPI chip selects driven by the host to the codec for register access. Always asserted for eight SCLK cycles at a time. Only Byte-by-Byte mode codec register read/write is supported.
TDM_SPI_SCK	O	Serial SPI clock from the host to the codec for register access. This is an RTO (return to one) clock. It toggles for eight cycles at a time (for 1 byte transfer) during codec register access, then it returns to high. The host drives write data on TDM_SPI_MOSI on the negative edge of TDM_SPI_SCK, and captures read data from the codec on the positive edge of TDM_SPI_SCK.
TDM_SPI_MOSI	O	Serial SPI data from the host to the codec for register access. When TDM_SPI_CS[1:0] is asserted low, the data is driven from the host on the negative edge of TDM_SPI_SCK. It is always driven for eight TDM_SPI_SCK cycles at a time. In a byte, the data can be driven MSB or LSB first.
TDM_SPI_MISO	I	Serial SPI read data from the codec to the host for register access. When TDM_SPI_CS is asserted low, this data is driven from codec on negative edge of TDM_SPI_SCK. It is always driven for eight TDM_SPI_SCK cycles at a time. The codec drives data on this line only for a read operation, when it receives the command and address in previous bytes from the host on TDM_SPI_MOSI In a byte, the data can be driven MSB or LSB first.



These pins are multiplexed on the device bus and MPP pins. For further information, see the Pin Multiplexing section in the respective device *Hardware Specifications*.

The Frame Sync (FS) is generated (or sampled) indicating the start of the time slot. The time slot corresponds to one sample of 1 byte (A-law or U-law encoded) or 2 bytes (linear encoded). For 2 bytes, each sample occupies two adjacent time slots. In transmit operation, the data is driven out to the codec on the Data Tx (DTX) line. The TDM can drive data on the positive edge or negative edge of PCLK. The TDM can be the master of FS and PCLK (it drives both of these) or it can be the slave. As a slave, it receives PCLK and FS from an outside master.

## 5.2

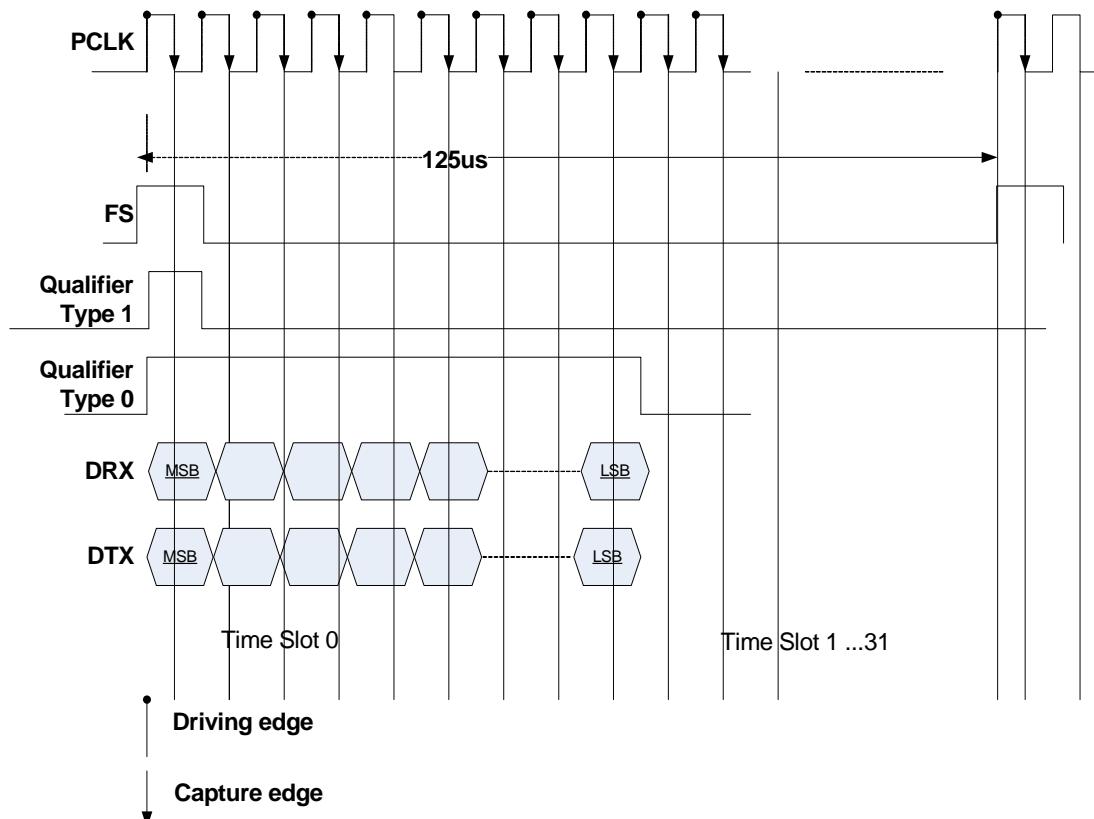
## TDM Protocol Specification

The TDM interface defines time slots for PCM data transmit and receive. The start of the time slot is indicated by an FS. The time slot consists of 8 PCM clocks (PCLK). The FS is always asserted at an interval of 125 µs, which corresponds to a frequency of 8 kHz. In each FS, one sample of PCM data is received and transmitted.

The FS indicates the start of the Time Slot 0, as shown in the [Figure 8](#).

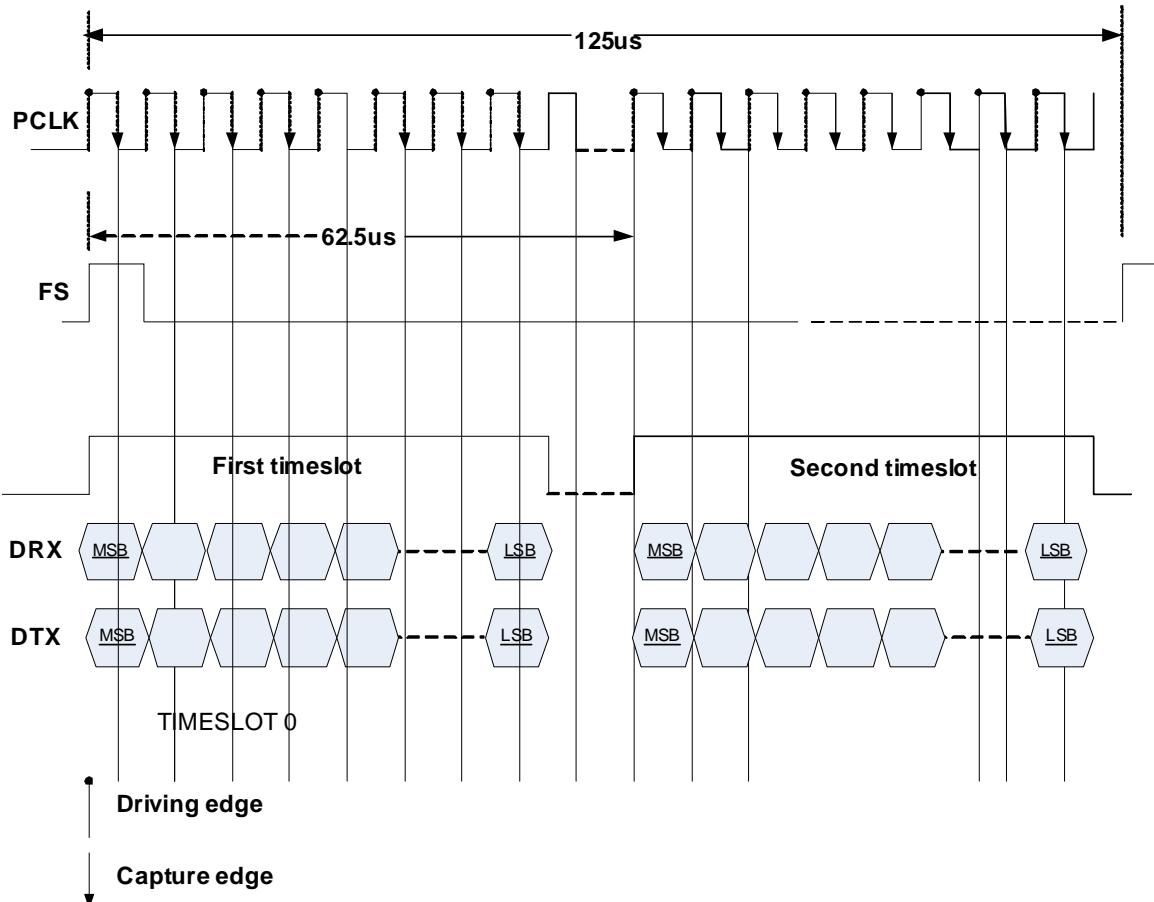
- **Data Rx:** The codec drives the Data Rx (DRX) line on the positive edge of the PCLK, and the TDM captures on the negative edge of the PCLK. In receive operations, the data is received from the codec on the DRX line serially, in MSB first order.
- **Data Tx:** The TDM drives data on the positive or negative edge of PCLK on the DTX line. The codec captures data on the negative edge of the PCLK. In transmit operations, the data is driven out to the codec on DTX serially, in MSB first order.

**Figure 8: TDM Operation Time Slot 0**



TDM also supports Wideband mode, in which two PCM samples are transmitted and received in one frame sync (125 µs) from TDM to codec and vice-versa. The wideband codec is used for enhanced voice quality in VoIP networks. The voice quality is improved because the effective sampling rate becomes 16 kHz (instead of the 8 kHz rate in the standard Narrowband mode). The second sample is sent in a time slot that is 62.5 µs from the first time slot, as shown in [Figure 9, TDM Wideband Mode Operation, on page 60](#).

**Figure 9: TDM Wideband Mode Operation**



The TDM drives data on the positive or negative edge of PCLK on the DTX line. The FS can be short, long, inverted, and driven on the positive or negative edge of PCLK, depending upon the programmed value.

The TDM also supports the generation of four qualifier signals (two per channel) for receive and transmit, as shown in [Figure 8, TDM Operation Time Slot 0, on page 59](#). The qualifiers are used as data enable for codecs that do not have programmable time slots. These qualifiers can be programmed active for either:

- One PCLK, indicating the MSB for that time slot (Type1)  
or
- The full duration of the time slot (Type0)

### 5.2.1 Tx Data Flow



The term CH<sub>x</sub> in the following sections stands for CH0 or CH1 (Channel 0 or Channel 1).

**Note**

The TDM supports two channels for Tx operation. Both channels are independent of one another, and each channel has a separate set of control and configuration registers.

The sequence for sending PCM data from the TDM to the codec is:

1. Firmware processes the incoming data from the IP network and creates a sample buffer in memory (typically a 10-ms sample buffer). The sample size must be in multiples of 4 bytes. The maximum size is the 30-ms width of the sample.
2. CPU initializes the TDM.

These registers are programmed only at the beginning of the Tx operation. Changing register values in the middle of TDM operation is not permitted.

- TDM PCM Clock Rate Divisor Register ([Table 215 p. 426](#))—selecting PCM clock.
- Number of Time Slots Register ([Table 214 p. 425](#))—FS generation.
- Miscellaneous Control Register ([Table 221 p. 429](#))—assessing reset to codec.
- PCM Control Register ([Table 203 p. 418](#))—TDM features supported by TDM. By default, bits[1:0] are set to 0x3, which indicates target device mode TDM operation. For initiator device operation mode, set bits[1:0] to 0x0. The codec is then reset by programming the [`<CODEC\_RST>`](#) field in the Miscellaneous Control Register ([Table 221 p. 429](#)) to allow sufficient time for the codec reset (see individual codec specification for details). The codec checks and loads the ratio of the PCM and FS clocks during reset, and uses the values for internal operation.
- Channel Time Slot Control Register ([Table 204 p. 420](#))—selecting the time slot in which PCM data is received. The same time slot value should also be programmed in the codec. The Rx time slot of the TDM is the Tx time slot of the codec.
- Channel 0/1 Total Sample Count Register (n=0–1) ([Table 213 p. 425](#))—program with the [`<CH0/1\_TOTAL\_SMPL\_CNT>`](#) and [`<CH0/1\_INT\_SMPL\_CNT>`](#) fields that are used by the TDM to synchronize with firmware. Firmware typically creates 10-ms sample widths (The width may be any Dword size buffer up to the 30-ms sample width) of buffer in memory. The sample from the codec can be 1- or 2-bytes in size, depending upon codec register settings.

3. CPU implements ping-pong buffers and enables transmit operation in TDM.

These registers can also be programmed during TDM operation.

- The CPU checks the [`<TX\_DMA\_ST\_ADDR\_OWN\_CHx>`](#) field in the Channel 0/1 Buffer Ownership Register (n=0–1) ([Table 208 p. 423](#)) for a value 0. A value of 0 indicates that software can program the [`<TX\_DMA\_ST\_ADDR\_CH0>`](#) field in the Channel 0 Transmit Data Start Address Register ([Table 209 p. 423](#)) with the buffer address.
- The CPU programs the [`<TX\_DMA\_ST\_ADDR\_OWN\_CHx>`](#) field in the Channel 0/1 Buffer Ownership Register (n=0–1) ([Table 208 p. 423](#)) with 1, which indicates this buffer is now owned by the hardware.
- The hardware makes a copy of the buffer address as programmed in the [`<TX\_DMA\_ST\_ADDR\_CH0>`](#) field in the Channel 0 Transmit Data Start Address Register ([Table 209 p. 423](#)), and then resets the [`<TX\_DMA\_ST\_ADDR\_OWN\_CHx>`](#) field in the Channel 0/1 Buffer Ownership Register (n=0–1) ([Table 208 p. 423](#)) to 0. When this bit is not 0, the CPU cannot program the [`<TX\_DMA\_ST\_ADDR\_OWN\_CHx>`](#) field with a new value. In this way, the ping-pong buffers can be implemented in memory by software.
- The [`<TX\_DMA\_ST\_ADDR\_OWN\_CHx>`](#) field points to the address of the Tx buffer in memory.
- Codec register initialization (through SPI): See individual codec specification for register addresses and programming values.
- The CPU programs the [`<CHnTxEn>`](#) field in the Channel 0/1 Enable and Disable Register (n=0–1) ([Table 207 p. 422](#)) to enable the channel transmit operation.



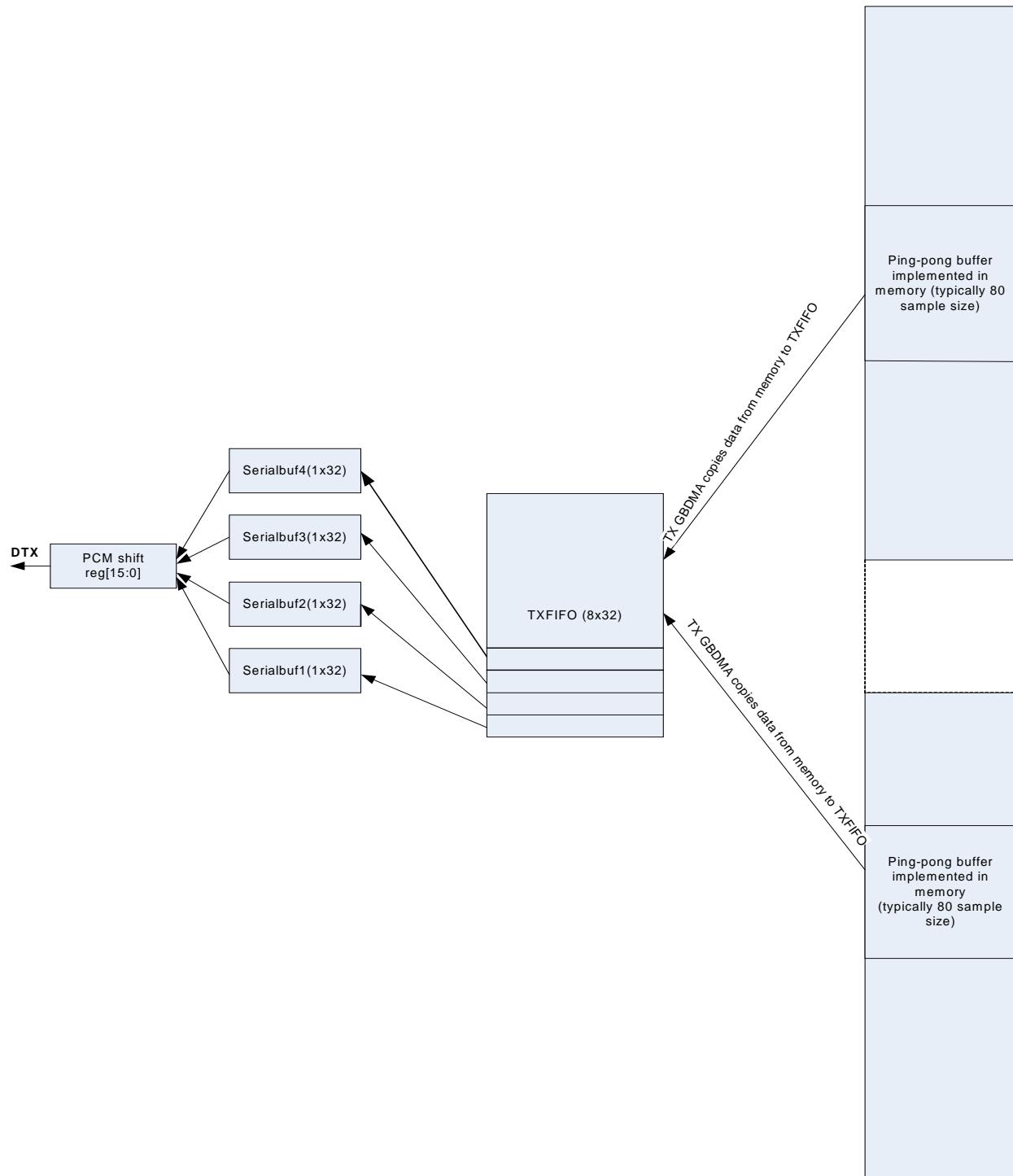
**Note**

The [`<TX\_DMA\_ST\_ADDR\_CH0>`](#) field must be 32-byte aligned.

4. DMA is accessed.
  - Active channels request access to DMA.
  - DMA performs a burst read of 32 bytes from memory.
  - Channel TXFIFO is filled up in one burst.
5. Once the TXFIFO is full, the TDM Tx read controller reads the TXFIFO and copies four Dwords to the serial buffer as shown in [Figure 10](#). When the TXFIFO becomes empty, the DMA is activated again, to fetch the next 32 bytes from memory. This procedure continues until the total sample count is reached (see the [`<CH0/1\_TOTAL\_SMPL\_CNT>`](#) field in the Channel 0/1 Total Sample Count Register (n=0–1) ([Table 213 p. 425](#))). For example, if the [`<CH0/1\_TOTAL\_SMPL\_CNT>`](#) field is set to 80 bytes, the DMA is activated three times—fetching the first 32B, fetching the next 32B, fetching the last 16B.

**Figure 10: TDM Transmit Path**

---



6. As the sample sent to the codec becomes equal to the <CH0/1\_INT\_SMPL\_CNT> field in the Channel 0/1 Total Sample Count Register (n=0–1) ([Table 213 p. 425](#)), an interrupt—the <SCOCH0\_TX\_INT> or <SCOCH1\_TX\_INT> field in the Interrupt Status Register ([Table 219 p. 427](#))—is issued to the CPU indicating the need for a new buffer. When the CPU receives the SCOCHx\_TX\_INT interrupt, three conditions arise:
  - Channel active: If the channel is still active, the CPU makes another buffer with the buffer size as programmed in the Channel 0/1 Total Sample Count Register (n=0–1) ([Table 213 p. 425](#)). The CPU checks if the <TX\_DMA\_ST\_ADDR\_OWN\_CHx> field in the Channel 0/1 Buffer Ownership Register (n=0–1) ([Table 208 p. 423](#)) is 0, programs the <TX\_DMA\_ST\_ADDR\_CH0> field in the Channel 0 Transmit Data Start Address Register ([Table 209 p. 423](#)) with the new buffer address, and sets the buffer ownership to 1. When all the data is fetched from the current buffer, the TDM checks that the <TX\_DMA\_ST\_ADDR\_OWN\_CHx> field is 1 and makes a copy of the <TX\_DMA\_ST\_ADDR\_CH0> field for the next DMA operation. It also resets the <TX\_DMA\_ST\_ADDR\_OWN\_CHx> field to 0 when the previous buffer operation is complete (i.e., all the data has been send out on the PCM bus). In this way, the software can poll the <TX\_DMA\_ST\_ADDR\_OWN\_CHx> field to check the status of the buffer and can implement the ping-pong buffer.  
For normal operation, when the channel is active, Steps 4, 5, and 6 are repeated.
  - Channel closed from IP side: Programs <CHnTxEn> field in the Channel 0/1 Enable and Disable Register (n=0–1) ([Table 207 p. 422](#)) to close the channel. The TDM sends all the samples in the existing buffer to the PCM interface and then moves to IDLE state. The TDM also, generates the CHx\_TX\_IDLE interrupt—the <CH0\_TX\_IDLE> or <CH1\_TX\_IDLE> field in the Interrupt Status Register ([Table 219 p. 428](#)).
  - The CPU fails to create a new buffer (if the channel is active) or fails to close the channel: The TDM underflows (see [Section 5.2.1.1, Tx Underflow, on page 64](#)).

## 5.2.1.1

### Tx Underflow

An underflow occurs when TDM is unable to meet the PCM transmit rate (one PCM sample every 125 µs in the Narrowband mode, or two samples in Wideband mode). There are two cases that might cause Tx underflow: GbDMA underflow or software underflow.

During normal Tx operation, if the TXFIFO empties, because the last 4 Dwords are copied to the serial buffer, and there are still samples to be fetched from the buffer in memory, the GbDMA is triggered to fill the TXFIFO again. At that point, there are 17 PCM samples stored in the TDM to be sent on the DTX line (1 in the PCM shift register and 16 in the serial buffers). If GbDMA fails to fetch the next 32 bytes from memory, before these 17 PCM samples are sent on the DTX line (17x125 µs), an underflow flag is set, which indicates that Tx underflow has occurred in the TDM. However, the underflow interrupt is not triggered until the GbDMA fills the TXFIFO. This delay occurs avoid the GbDMA filling the TXFIFO at some later time and causing the TXFIFO write pointer to change.

In case of underflow the following actions are performed by the TDM:

1. The TDM generates an UFLOW\_CHx\_INT interrupt—the <UFLOW\_CH0\_INT> or <UFLOW\_CH1\_INT> field in the Interrupt Status Register ([Table 219 p. 427](#))—to the CPU if the interrupt is enabled in the Interrupt Status Register ([Table 219 p. 427](#)) and Interrupt Status Mask Register ([Table 217 p. 426](#)).
2. The TDM switches off the channel transmit, by setting the <CHnTxEn> field in the Channel 0/1 Enable and Disable Register (n=0–1) ([Table 207 p. 422](#)) to 0, and moves to IDLE state. The TDM also flushes the TXFIFO. It keeps sending 0s on the PCM interface. The buffer that was being processed at the time that the underflow occurred becomes invalid. The audio is disrupted.

To start the transmit operation again, software has to re-initialize the channel, by following the Step 3 as explained in [Section 5.2.1, Tx Data Flow, on page 60](#).

Software underflow occurs when the CPU fails to make a new buffer (if the channel is active) or fails to close the channel.

Once the SCOCCHx\_TX\_INT is generated, there is a time window available for software to either point to a new buffer to TDM or to close the channel, if it is closed from the IP side. This window starts from the assertion of interrupt SCOCCHx\_TX\_INT and ends at the buffer switch condition. The buffer switch condition is defined as "The existing buffer is already fetched to TXFIFO earlier, data from the TX FIFO is copied to serial buffers, and now there are four samples (500 µs in terms of time) of PCM data remaining in the serial buffers." The TDM switches to the next buffer at this time and checks the [`<TX\_DMA\_ST\_ADDR\_OWN\_CHx>`](#) field in the Channel 0/1 Buffer Ownership Register (n=0–1) ([Table 208 p. 423](#)) for a value of 1. The TDM needs to switch at this time to a new buffer so that it can perform the next DMA operation, in a timely manner, and fill the TX FIFO again. If at the buffer switch event, the TDM does not see [`<TX\_DMA\_ST\_ADDR\_OWN\_CHx>`](#) = 1 and the channel is still enabled, the underflow flag is set.

Setting of this flag means that the TDM has underflow. However, the UFLOW\_CHx\_INT is still not generated, as the TDM has to send all the PCM samples that it is still storing. At this moment, there are 4 or 8 samples stored in the TDM (one in the PCM shift register and three in the serial buffer for Narrowband mode, 1 in the PCM shift register and seven in the serial buffer for Wideband mode). After sending these samples out on the DTX line, the UFLOW\_CHx\_INT is triggered.

The following actions are performed by TDM in case of underflow:

1. The TDM generates an UFLOW\_CHx\_INT interrupt—the [`<UFLOW\_CH0\_INT>`](#) or [`<UFLOW\_CH1\_INT>`](#) field in the Interrupt Status Register ([Table 219 p. 427](#))—to the CPU if the interrupt is enabled in the Interrupt Status Register ([Table 219 p. 427](#)) and Interrupt Status Mask Register ([Table 217 p. 426](#)).
2. The TDM switches off the channel transmit by setting the [`<CHnTxEn>`](#) field in the Channel 0/1 Enable and Disable Register (n=0–1) ([Table 207 p. 422](#)) to 0 and moves to IDLE state. It also flushes the TXFIFO. It keeps sending 0s on the PCM interface. The audio is disrupted.

To start the transmit operation again, software has to re-initialize the channel, by following Step 3 in [Section 5.2.1, Tx Data Flow, on page 60](#).

### 5.2.1.2

#### Setting INT\_SAMPLE\_CNT

It is important to set the [`<CH0/1\_INT\_SMPL\_CNT>`](#) field in the Channel 0/1 Total Sample Count Register (n=0–1) ([Table 213 p. 425](#)) for normal Tx operation of the TDM. Once this number of samples is sent on the DTX line, the SCOCCHx\_TX\_INT interrupt is generated. Software then has a window of time to point to the new buffer or close the channel, if it is closed from the IP side.

Software should set the [`<CH0/1\_INT\_SMPL\_CNT>`](#) field to allow the CPU sufficient time to point to the new buffer or close the channel. This ensures that a Tx underflow does not occur.

In quantitative terms the window size is given by the following equation (see the software underflow description in [Section 5.2.1.1, Tx Underflow, on page 64](#)):

- For Narrowband mode(1-byte or 2-byte sample)  

$$\text{Window size} = (\text{TOTAL_SAMPLE_CNT} - 4 - \text{INT_SAMPLE_CNT}) * 125 \mu\text{s}$$
- For Wideband mode (1-byte or 2-byte sample)  

$$\text{Window size} = (\text{TOTAL_SAMPLE_CNT} - 8 - \text{INT_SAMPLE_CNT}) * 125 \mu\text{s}$$

### 5.2.2

#### Rx Data Flow

The TDM supports two channels for Rx operation. Both channels are independent of one another, each with a separate set of control and configuration registers.

The sequence for receiving PCM data from the codec (codec to TDM) is:

1. When the telephone is picked up to start a conversation, an off-hook state is detected, and the codec generates an interrupt. The dial tone is passed to the POTS by the codec, and the CPU reads the interrupt. Once the receive channel is active:



- A codec interrupt is indicated by the <CODEC\_INT> field in the Interrupt Status Register ([Table 219 p. 428](#)).
  - The CPU reads the Interrupt Status Register ([Table 219 p. 427](#)) in the codec.
  - The receive channel is now active.
  - The CPU starts the TDM initialization process.
2. CPU initializes the TDM.
- These registers are programmed only at the beginning of the Rx operation. Changing register values in the middle of TDM operation is not permitted.
- TDM PCM Clock Rate Divisor Register ([Table 215 p. 426](#))—selecting PCM clock
  - Number of Time Slots Register ([Table 214 p. 425](#))—FS generation
  - Miscellaneous Control Register ([Table 221 p. 429](#))—assessing reset to codec
  - PCM Control Register ([Table 203 p. 418](#))—TDM features supported by TDM.  
By default, bits[1:0] are set to 0x3, which indicates target device mode TDM operation.  
For initiator device operation mode, set bits[1:0] to 0x0. The codec is then reset by programming the <CODEC\_RST> field in the Miscellaneous Control Register ([Table 221 p. 429](#)) for to allow sufficient time for the codec reset (see individual codec specification for details). The codec checks and loads the ratio of PCM and FS clocks during reset and uses the values for internal operation.
  - Channel Time Slot Control Register ([Table 204 p. 420](#))—selecting the time slot in which PCM data is received. The same time slot value should also be programmed in the codec. The Rx time slot of the TDM is the Tx time slot of the codec.
  - Channel 0/1 Total Sample Count Register (n=0–1) ([Table 213 p. 425](#))—program with the <CH0/1\_TOTAL\_SMPL\_CNT> and <CH0/1\_INT\_SMPL\_CNT> fields, which are used by the TDM to synchronize with the firmware. Firmware typically makes 10-ms sample width (The width may be any Dword size buffer up to the 30-ms sample width. The size must be a multiple of 4 bytes.) of buffer in memory. The sample from the codec can be 1- or 2-bytes in size, depending upon codec register settings.
3. CPU implements ping-pong buffers and enables receive operation in the TDM.
- These registers can also be programmed during TDM operation.
- The CPU checks the <RX\_DMA\_ST\_ADDR\_OWN\_CHx> field in the Channel 0/1 Buffer Ownership Register (n=0–1) ([Table 208 p. 423](#)) for a value 0. A value of 0 indicates that software can program the <RX\_DMA\_ST\_ADDR\_CH0> field in the Channel 0 Receive Data Start Address Register ([Table 210 p. 424](#)) with the buffer address.
  - The CPU programs the <RX\_DMA\_ST\_ADDR\_OWN\_CHx> field in the Channel 0/1 Buffer Ownership Register (n=0–1) ([Table 208 p. 423](#)) with 1, which indicates that this buffer is now owned by hardware.
  - TDM makes a copy of the buffer address as programmed in the <RX\_DMA\_ST\_ADDR\_CH0> field, and then resets the <RX\_DMA\_ST\_ADDR\_OWN\_CHx> field to 0. When this bit is not 0, the CPU cannot program the <RX\_DMA\_ST\_ADDR\_CH0> field with a new value. In this way, ping-pong buffers can be implemented in memory by software.
  - <RX\_DMA\_ST\_ADDR\_CH0>—points to the address of the Rx buffer in memory.
  - Codec register initialization (through SPI): See the individual codec specifications for register addresses and programming values.
  - The CPU programs the <CHnRxEn> field in the Channel 0/1 Enable and Disable Register (n=0–1) ([Table 207 p. 422](#)) to enable the channel receive operation.



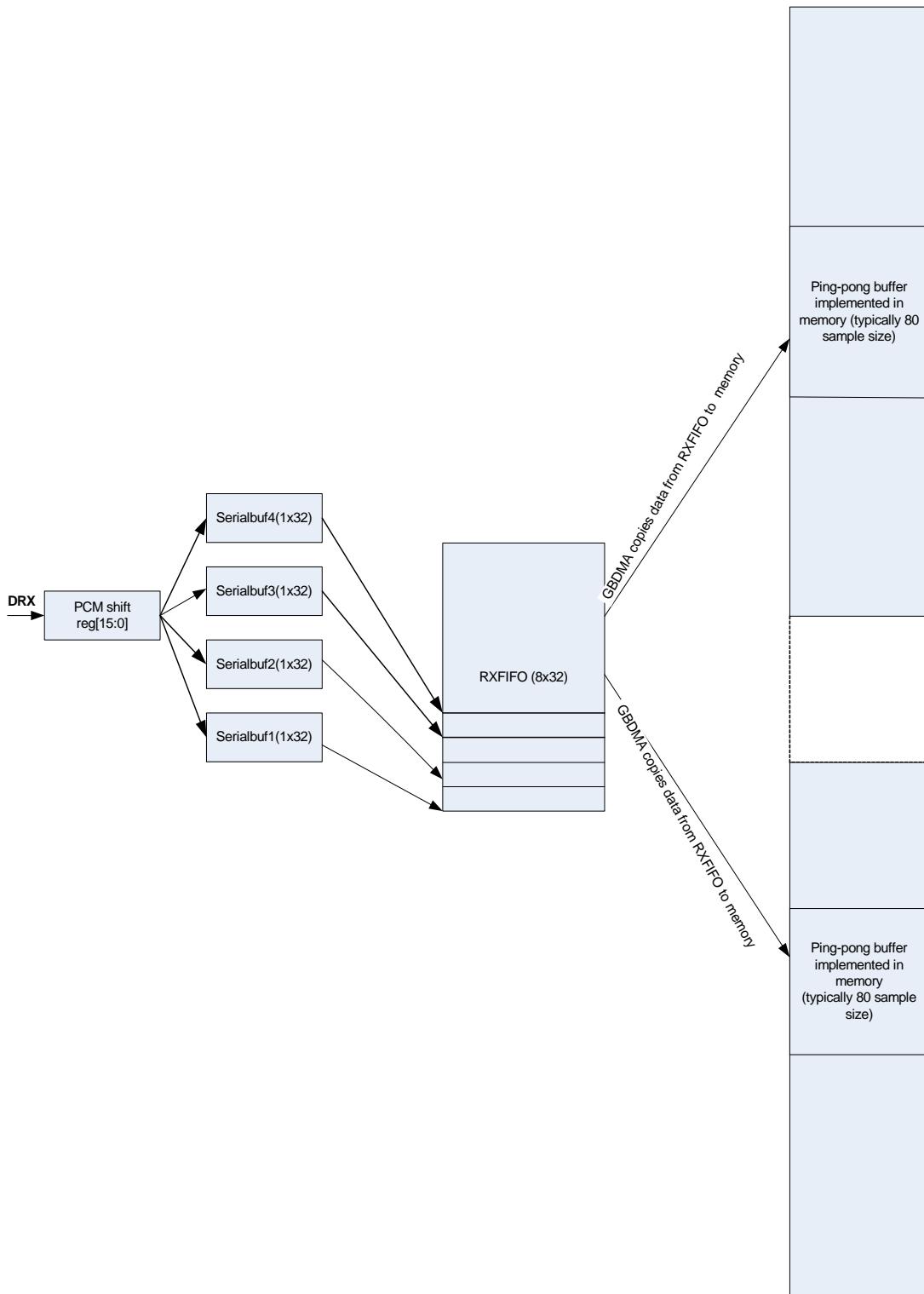
**Note**

The [`<RX\_DMA\_ST\_ADDR\_CH0>`](#) field must be 32-byte aligned.

4. Once the channel is enabled, the TDM begins receiving data serially (from the next FS) on the DRX line in the programmed time slot as shown in [Figure 11](#). Once a voice sample is received in PCM shift register, it is copied to a serial buffer. This serial buffer helps to overcome the latency on the internal bus (backup buffer). There are four serial buffers (1 Dword in size each). Filling up one or all of these buffers is set by the [`<PerfBit>`](#) field in the PCM Control Register ([Table 203 p. 420](#)). If the [`<PerfBit>`](#) field is set to 1 (default), all serial buffers are filled with PCM samples, and then they are copied to the RXFIFO.
5. Once the RXFIFO becomes full, the DMA is accessed:
  - The channel requests access to the DMA.
  - The DMA performs a burst write of 32 bytes to memory (the channel RXFIFO is emptied in one burst).

When the RXFIFO becomes full, the DMA is activated again to write the next 32 bytes to memory. This procedure continues until the total sample count is reached. For example, if the the [`<CH0/1\_TOTAL\_SMPL\_CNT>`](#) field in the Channel 0/1 Total Sample Count Register ( $n=0-1$ ) ([Table 213 p. 425](#)) is set to 80 bytes, the DMA will be activated three times—write first 32 bytes, write next 32 bytes, write last 16 bytes.

**Figure 11: TDM Receive Path**



6. As samples are received from codec, and the number becomes equal the `<CH0/1_INT_SMPL_CNT>` field in the Channel 0/1 Total Sample Count Register ( $n=0-1$ ) ([Table 213 p. 425](#)) value, a SCOCNx\_RX\_INT interrupt is issued to the CPU indicating that a new buffer should be created. The CPU evaluates the conditions that can arise during TDM Rx operations.
    - Channel is still active—if the channel is still active, the CPU creates another buffer of the same size as that programmed in the Channel 0/1 Total Sample Count Register ( $n=0-1$ ) ([Table 213 p. 425](#)) (after receiving a SCOCNx\_RX\_INT interrupt). The CPU checks the `<RX_DMA_ST_ADDR_OWN_CHx>` field in the Channel 0/1 Buffer Ownership Register ( $n=0-1$ ) ([Table 208 p. 423](#)) for a value of 0, programs the `<RX_DMA_ST_ADDR_CH0>` field in the Channel 0 Receive Data Start Address Register ([Table 210 p. 424](#)) with the new buffer address, and sets the buffer ownership to 1. When all the PCM data is written to the current buffer, the TDM then checks that the `<RX_DMA_ST_ADDR_OWN_CHx>` field is 1 and makes a copy of the `<RX_DMA_ST_ADDR_CH0>` field for the next DMA operation, and reset the `<RX_DMA_ST_ADDR_OWN_CHx>` field to 0. In this way, software can poll the `<RX_DMA_ST_ADDR_OWN_CHx>` field to check the status of the buffer and to start further processing of the buffer.
- For normal operation, when the channel is active, Steps 4, 5, and 6 are repeated.
- Channel closed due to telephone hang up (on-hook)—The user can hang up the telephone at any time during normal Rx operation. This can occur before the buffer is completely full. When the telephone hangs up, the codec generates an interrupt to the CPU (the `<CODEC_INT>` field in the Interrupt Status Register ([Table 219 p. 428](#))). After receiving the interrupt, the CPU can close the channel Rx by programming the `<CHnRxEn>` field in the Channel 0/1 Enable and Disable Register ( $n=0-1$ ) ([Table 207 p. 422](#)) to 0. In this case, the TDM stops receiving data from the DRX line, and the remaining PCM samples in the buffer are filled with dummy data (programmed in the `<DUMMY_DATA>` field in the Dummy Data for Dummy RX Write Register ([Table 220 p. 428](#))). The TDM then enters an IDLE state and generates a CHx\_RX\_IDLE interrupt.
  - The CPU fails to make a new buffer (if the channel is active) or fails to close the channel after the telephone is hung up—the TDM overflows (see [Section 5.2.2.1, Rx Overflow, on page 69](#)).

### 5.2.2.1 Rx Overflow

If the TDM is unable to meet the PCM transmit rate (one PCM sample every 125 µs in Narrowband mode, or two samples in Wideband mode), an overflow occurs. There are two cases that might cause Rx overflow—GbDMA overflow or software overflow.

During normal Receive operation, the RXFIFO becomes full when the last 4 Dwords are copied from the serial buffers. The GbDMA is then triggered to empty the RXFIFO. From that moment, the TDM can store 17 PCM samples (9 samples for linear mode) coming from DRX line. It can store 1 PCM sample in the PCM shift register and 16 in the serial buffers, if using 1-byte samples (or 1 in the PCM shift register and 8 in the serial buffer, if using 2-byte samples). If GbDMA fails to write the 32 bytes to memory before these 17 PCM samples are received on the DDX line (17 x 125 µs), an overflow flag is set, which indicates that RX overflow has occurred in the TDM. However, the overflow interrupt is not triggered until GbDMA empties the RXFIFO. This is to avoid GbDMA reading the RXFIFO at some later time and causing the RXFIFO read pointer to change.

In case of overflow, the following actions are performed by the TDM:

1. The TDM generates an interrupt OFLOW\_CHx\_INT to the CPU if the interrupt is enabled in the Interrupt Status Register ([Table 219 p. 427](#)) and Interrupt Status Mask Register ([Table 217 p. 426](#)).
2. The TDM switches off the channel receive by setting the `<CHnRxEn>` field in the Channel 0/1 Enable and Disable Register ( $n=0-1$ ) ([Table 207 p. 422](#)) to 0 and goes to IDLE state. The TDM

also flush the RXFIFO. It stops receiving data from PCM interface. The buffer that was being processed at the time overflow occurred becomes invalid. The audio is disrupted.

To start the receive operation again, software has to re-initialize the channel by following the Step 3 in [Section 5.2.2, Rx Data Flow, on page 65](#).

Software overflow occurs when the CPU fails to create a new buffer (if the channel is active) or fails to close the channel after the telephone is hang up.

Once SCOCCHx\_TX\_INT is generated, there is a time window available for software to either point to new buffer to TDM or to close the channel if it is closed from the IP side. This window starts from assertion of interrupt SCOCCHx\_RX\_INT and ends at the buffer full condition. The buffer full condition is defined as "All the PCM samples of the existing buffer have been written to the buffer in memory." The TDM switches to the next buffer at that time and checks the [`<RX\_DMA\_ST\_ADDR\_OWN\_CHx>`](#) field in the Channel 0/1 Buffer Ownership Register ( $n=0-1$ ) ([Table 208 p. 423](#)) for a value of 1 (to fetch the address of a new buffer). If the TDM detects [`<RX\_DMA\_ST\_ADDR\_OWN\_CHx>`](#) as 0 and the channel is still enabled, overflow interrupt OFLOW\_CHx\_INT is set.

The following actions are performed by the TDM in the case of overflow:

1. The TDM generates an interrupt OFLOW\_CHx\_INT to CPU if the interrupt is enabled in the Interrupt Status Register ([Table 219 p. 427](#)) and Interrupt Status Mask Register ([Table 217 p. 426](#)).
2. The TDM switches off the channel receive by setting the [`<CHnRxEn>`](#) field in the Channel 0/1 Enable and Disable Register ( $n=0-1$ ) ([Table 207 p. 422](#)) to 0 and moves to IDLE state. It also flushes the RXFIFO. It stops receiving data from the PCM interface. The audio is disrupted.

To start the receive operation again, software has to re-initialize the channel by following the Step 3 in [Section 5.2.2, Rx Data Flow, on page 65](#).

## 5.2.2.2

### Setting Interrupt Sample Count

Setting the [`<CH0/1\_INT\_SMPL\_CNT>`](#) field in the Channel 0/1 Total Sample Count Register ( $n=0-1$ ) ([Table 213 p. 425](#)) plays an important role in normal Rx operation of the TDM. Once the number of samples set in that field are received on the DRX line, the SCOCCHx\_RX\_INT interrupt is generated. Software then has a window of time to point to a new buffer or close the channel, if the user hangs up the telephone. Software should set [`<CH0/1\_INT\_SMPL\_CNT>`](#) so that the CPU has sufficient time to point to the new buffer or close the channel. This ensures that a Rx overflow does not occur.

In quantitative terms, the window size is given by the following equation (see the software overflow description in [Section 5.2.2.1, Rx Overflow](#)):

- For 1 byte sample  
 $\text{Window size} = (\text{TOTAL_SAMPLE_CNT} - \text{INT_SAMPLE_CNT}) * 125 \mu\text{s}$
- For 2 byte sample  
 $\text{Window size} = (\text{TOTAL_SAMPLE_CNT} - \text{INT_SAMPLE_CNT}) * 125 \mu\text{s}$

## 5.2.3

### TDM Wideband Mode Operation

In Wideband mode, two PCM samples are transmitted and received in one frame sync (125  $\mu\text{s}$ ) from TDM to codec and vice-versa. The wideband codec is used for enhanced voice quality in Voice over IP (VoIP) networks. The voice quality is improved because the effective sampling rate becomes 16 kHz. The second sample is sent in a time slot that is 62.5  $\mu\text{s}$  from the first time slot.

Each of the two channels of the TDM can be individually configured in Wideband or Narrowband mode. The [`<CH0WBand>`](#) field and the [`<CH1WBand>`](#) field in the PCM Control Register ([Table 203 p. 420](#)) define the Wideband versus Narrowband mode selection. For the Wideband time slot programming, two registers are provided in each channel. The transmit and receive flow remains the same as explained in [Section 5.2.1, Tx Data Flow, on page 60](#) and [Section 5.2.2, Rx Data Flow, on page 65](#).

To set Channel x to Wideband mode:

1. Set the <CH0WBand> field in the PCM Control Register ([Table 203 p. 420](#)) to 1 to select Wideband mode for CHx.
2. Set the <CH0DlyEn> field in the PCM Control Register ([Table 203 p. 419](#)) to 1 to select delay control for time slot programming.
3. Program the Channel 0 Delay Control Register ([Table 205 p. 421](#)) for the first time slot for transmit and receive of PCM data.
4. Program the TDM Channel0 Wideband Delay Control Register ([Table 229 p. 431](#)) for the second time slot for transmit and receive of PCM data. For wideband codecs, this time slot is generally 62.5 µs from the first time slot.

## 5.3

## TDM (SLIC/Codec) Registers Access via SPI

The codec register read/write interface is a generic design, to support various PCM codecs, which follow SPI (4-wire) type protocol for register read/write. The pins used for this interface are:

- TDM\_SPI\_CS[1:0]
- TDM\_SPI\_SCK
- TDM\_SPI\_MOSI
- TDM\_SPI\_MISO

For a description of these pins see [Table 10](#).



**Note**

The 88F6192 and 88F6281 support two methods for interfacing two SLIC/codec devices via SPI. The first method is chaining the SLIC devices, in which case only one TDM\_SPI\_CS[1:0] signal is required. The second method uses two separate TDM\_SPI\_CS[1:0] signals, one signal per each SLIC.

### 5.3.1

### Codec Register Write Operation

A typical codec register write is shown in [Figure 12](#). The transaction takes place in a Byte-by-Byte mode. The CPU performs the following sequence for a codec register write:

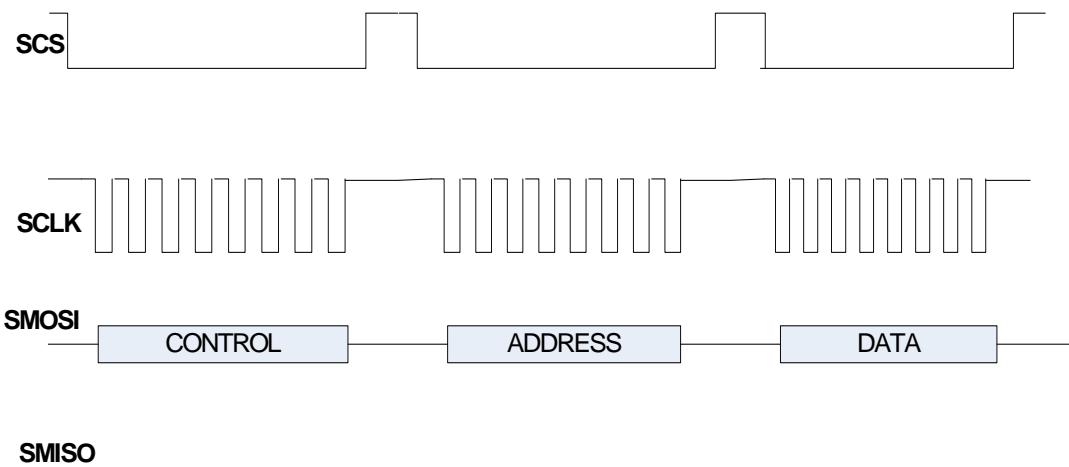
1. The CPU must check that the status of the <SPISStat> field in the SPI Control Register ([Table 197 p. 415](#)) is 0 before starting any codec register read/write operation.
2. The CPU programs the CSU Global Control Register ([Table 196 p. 415](#)).
3. The CPU programs Codec Access Command Low Register ([Table 198 p. 415](#)), Codec Access Command High Register ([Table 199 p. 416](#)), and Codec Registers Access Control ([Table 200 p. 416](#)) with the appropriate values, depending on the codec specification requirements.
4. After all of these steps, the CPU writes 1 in the <SPISStat> field. By writing 1 in this field, it starts the codec register read/write operation. The 88F6192/88F6281 device drives the active low chip select (TDM\_SPI\_CS[1:0]) and it toggles TDM\_SPI\_SCK for exactly eight cycles. The command byte is sent to the codec first (depending upon the individual codec, e.g., SI3210 requires address in the first byte) on TDM\_SPI\_MOSI. The device drives TDM\_SPI\_MOSI on the negative edge of TDM\_SPI\_SCK and the codec captures on the positive edge of TDM\_SPI\_SCK.

Each byte can be sent by MSB/LSB first, depending upon the individual codec. This is set by programming the <LSB\_MSB> field in the Codec Registers Access Control ([Table 200 p. 416](#)). TDM\_SPI\_CS[1:0] becomes high after each byte transfer.

After the command byte is sent, TDM\_SPI\_CS[1:0] toggles again for eight TDM\_SPI\_SCK cycles and the address byte is set on TDM\_SPI\_MOSI. The write data is sent after this step.

5. When the write operation is complete, the device resets the [\*\*<SPISStat>\*\*](#) field in the SPI Control Register ([Table 197 p. 415](#)) to 0. The CPU polls this bit. When it becomes 0, the CPU can start the next codec write transaction.

**Figure 12: Codec Register Write Operation**



### 5.3.2 Codec Register Read Operation

A typical codec register read is shown in [Figure 13, Codec Register Read Operation, on page 73](#). The transaction takes place in Byte-by-Byte mode. The CPU performs the following sequence for a codec register read:

1. The CPU must check the status of the [\*\*<SPISStat>\*\*](#) field in the SPI Control Register ([Table 197 p. 415](#)).
2. The CPU programs the CSU Global Control Register ([Table 196 p. 415](#)).
3. The CPU programs the Codec Access Command Low Register ([Table 198 p. 415](#)), Codec Access Command High Register ([Table 199 p. 416](#)), and Codec Registers Access Control ([Table 200 p. 416](#)) with the appropriate values, depending on the codec specification requirements.
4. The CPU writes 1 in the [\*\*<SPISStat>\*\*](#) field. Writing 1 in this field starts the codec register read/write operation. The 88F6192/88F6281 device drives the active low chip select (TDM\_SPI\_CS[1:0]) and it toggles TDM\_SPI\_SCK for exactly eight cycles. The command byte is sent to codec first (depending upon the individual codec) on SDO. The device drives SDO on the negative edge of TDM\_SPI\_SCK and the codec captures on the positive edge of TDM\_SPI\_SCK.

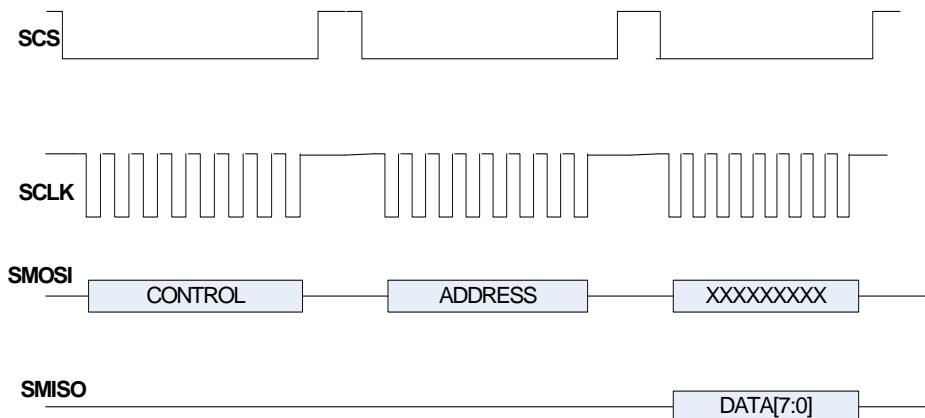
Each byte can be sent MSB/LSB first, depending upon the individual codec. This is set by programming the [\*\*<LSB\\_MSB>\*\*](#) field in the Codec Registers Access Control ([Table 200 p. 416](#)). TDM\_SPI\_CS[1:0] becomes high after each byte transfer.

After the command byte is sent, TDM\_SPI\_CS[1:0] toggles again for eight TDM\_SPI\_SCK cycles and the address byte is sent on TDM\_SPI\_MISO. After the address byte, the codec responds by sending the read data on TDM\_SPI\_MISO, when TDM\_SPI\_CS[1:0] and

TDM\_SPI\_SCK are toggled again by the device. The codec drives read data on the negative edge of TDM\_SPI\_SCK, and the device captures it on the positive edge of TDM\_SPI\_SCK.

5. When the read operation is complete, the 88F6192/88F6281 resets [<SPISStat>](#) field in the SPI Control Register ([Table 197 p. 415](#)) to 0. The CPU polls this bit. When it becomes 0, the CPU can read the Codec Read Data Register ([Table 201 p. 417](#)) for the read data value.

**Figure 13: Codec Register Read Operation**



# 6 PCI Express Interface

The device integrates one PCI Express x1 port.

The PCI Express interface has the following features:

- PCI Express Base 1.1 compatible
- Root Complex port or Endpoint port
- Embedded PCI Express PHY based on proven Marvell® SERDES technology
- x1 link width
- 2.5 GHz signalling
- Lane polarity inversion support
- Replay buffer
- Maximum payload size of 128 bytes
- Single Virtual Channel (VC-0)
- Ingress and egress flow control
- Extended Tag support
- Interrupt emulation message support
- Power Management (PM):
  - Software power management states  $D_1$ ,  $D_2$ ,  $D_{3\text{hot}}$ , and  $D_{3\text{cold}}$  support
  - Active state power management L0s and L1 support
- Advanced Error Reporting (AER) capability support
- Single function device configuration header
- Message Signaled Interrupts (MSI) support
- Expansion ROM support
- Programmable address map

## 6.1

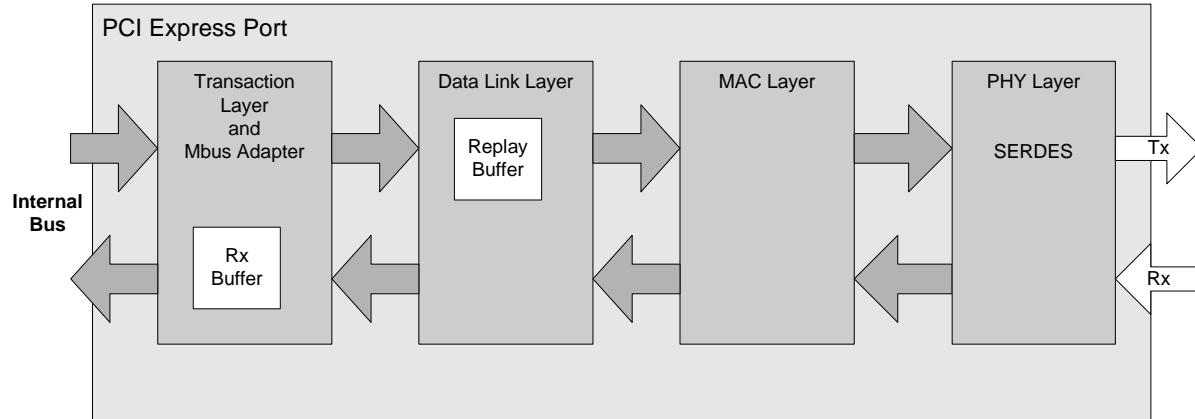
### Functional Description

The PCI Express interface uses a layered architecture, according to the PCI Express specifications. The main layers are the PHY layer, MAC layer, Data Link layer, and Transaction layer. In addition, a Core Adapter layer handles the forwarding of the PCI Express Transaction Layer Packets (TLP) to the device Mbus.

[Figure 14](#) provides a high-level diagram of the PCI Express interface.

**Figure 14: High-level Block Diagram**

---



### 6.1.1 PHY Layer

On the Tx path, the PHY layer receives symbols from the MAC layer, converts them into a serialized format, and transmits them on the PCI Express port. On the Rx path, the PHY layer receives a serialized stream from the PCI Express port, and forwards them as parallel symbols to the MAC layer.

The PHY handles symbol-locking and 8b10b encoding/decoding.

The PHY layer is responsible also for the clock tolerance compensation. The received symbol stream is adapted to the local clock, by adding or deleting skip OSs (Ordered Sets).

### 6.1.2 MAC Layer

The MAC layer is responsible for establishment and maintenance of the PCI Express link, packet framing, and data packing. The PCI Express LTSSM (Link Training and Status State Machine) is located in this layer. It contains all the functionality for link configuration in terms of the lane polarity. Additionally, the MAC layer performs the scrambling and functions. The MAC layer controls the different link power-management modes, loopback mode, link disable mode and link hot-reset function. In addition, the MAC layer handles the generation and detection of the various TSs (Training Sequences) and OSs.

On the Tx path, the MAC layer receives packets (TLPs and DLLPs—Data Link Layer Packets) from the Data Link layer. The packets are framed, scrambled, and packed into the relevant link width and forwarded to the PHY.

On the Rx path, the MAC layer receives aligned symbols from the PHY. The symbols are unpacked according to the relevant link width and unframing is performed. The packets (DLLPs and TLPs) are then extracted from the frames and forwarded to the Data Link layer.

### 6.1.3 Data Link Layer

The Data Link layer provides a reliable TLP exchange between two components on the PCI Express link. This layer performs most of the data integrity functions as specified by the PCI Express 1.1 specification.

The Data Link layer controls the sequence number generation and detection. It also controls the LCRC generation and detection. Outgoing TLPs are temporarily stored in a replay buffer until an acknowledge is received from the far-end component. When a corrupted or missing TLP is detected, the replay mechanism is used to recover and maintain reliable Transaction layer to Transaction layer connection. The replay buffer holds transmitted packets and retransmits them when required.

The Data Link layer handles the generation and processing of DLLPs. DLLPs are used for conveying information such as flow control, TLP acknowledgment, and power management handshake.

#### 6.1.4 Transaction Layer

The Transaction layer primary responsibility is handling of TLPs. Outgoing TLPs are assembled and scheduled for transmission. Incoming TLPs are parsed and checked for various errors. In addition, the Transaction layer is responsible for handling the split transaction protocol—both towards the PCI Express port and the internal bus.

The Tx path accepts TLPs from the Mbus and schedules them for transmission, based on the flow-control credit availability and the relevant ordering rules. Non-Posted (NP) TLPs are assigned with a unique tag before they are scheduled for transmission. TLPs are then passed on to the Data Link layer for transmission.

The Rx path examines the incoming TLPs for a variety of packet formation errors. Incoming completions tags are checked for a valid NP request that was sent by the device. TLPs are then passed to the Mbus.

### 6.2 Link Initialization

Enable the PCI Express interface by setting the [<PEX0En>](#) field in the CPU Control and Status Register ([Table 116 p. 368](#)). This allows programming of link parameters before the start of link initialization.

Lane polarity inversion is supported. The lane differential couple can be routed on the board regardless of its polarity.

In case the initialization fails and no link is established, the PHY will keep on trying to initiate a link forever unless the port is disabled. As long as the port is enabled, the PHY will go on trying to establish a link; once the PHY identifies that a device is connected to it, a link will be established.

The link must be enabled within 100 ms after reset to allow link initialization (per *PCI Express Base Specification*, Revision 1.1).

It is recommended that from one second after reset, the software starts to check on DL Down status. If the link DL is not UP, the software should power down the link, by first disabling the link, and 100 ms later, setting the power down register in the CPU registers.



**Note** If the software accesses a unit when the unit is powered down, this may cause the device to hang.

### 6.3 Master Memory Transactions

Master memory transactions are memory space read and write requests (MRd and MWr TLPs) that are generated and sent over the PCI Express link, and the respective completion TLPs that are received in return.

The following features are supported as a master memory requester:

- In Root Complex mode, a eight outstanding non-posted (NP) request (memory read request) and a four posted (P) request (memory write request)
- Maximum memory read request of 128 bytes
- Maximum memory write request of 128 bytes
- 64-bit addressing

## 6.4

## Master I/O Transactions

Master I/O transactions are I/O space read and write requests (IORd and IOWr TLPs) that are generated and sent over the PCI Express link, and the respective completion TLPs that are received in return.

The following features are supported as a master I/O requester:

- Eight outstanding NP requests (I/O read or write)
- Maximum I/O read request of 4 bytes
- Maximum I/O write request of 4 bytes
- 32-bit addressing



### Note

- The user must not initiate I/O requests that are larger than 4 bytes and cross the 4 byte address boundary. These requests are illegal according to the *PCI Express Base Specification Revision 1.1*.
- Only partial I/O transactions are supported.
- In Endpoint mode, only memory request generation is allowed by the *PCI Express Base Specification Revision 1.1*. When working in Endpoint mode, do not initiate I/O requests.

## 6.5

## Master Configuration Transactions

Master Configuration transactions are configuration space read and write requests (CfgRd0, CfgWr0, CfgRd1 and CfgWr1 TLPs) that are generated and sent over the PCI Express link, and the respective completion TLPs that are received in return.

The following features are supported as a master configuration requester:

- Eight outstanding NP requests (I/O read or write)
- Maximum Configuration read request of 4 bytes
- Maximum Configuration write request of 4 bytes
- Extended register number support (4 KB extended PCI Express configuration header space)



### Note

In Endpoint mode, only memory request generation is allowed by the *PCI Express Base Specification Revision 1.1*. When working in Endpoint mode, do not initiate configuration requests.

### 6.5.1

### Generation of Configuration Requests

As a Root Complex port, the CPU may generate Type0 or Type1 configuration cycles to the PCI Express Endpoints, via indirect access using the PCI Express Configuration Address Register



(Table 269 p. 450) and PCI Express Configuration Data Register (Table 270 p. 450) registers. The following procedure is used for generating configuration cycles:

1. **PCI Express Configuration Address Register**—Write the Target Bus, Device, Function, Register and Extended Register Numbers fields, using the <ConfigEn> field to enable this mechanism.
2. **PCI Express Configuration Data Register**—Read or write to generate a respective read or write configuration request. The type of the request (type 0 or type 1) is set according to the following rules:
  - *Type1 request*: generated if Target Bus Number is different from the internal Bus Number.
  - *Type0 request*: generated if Target Bus Number is same as the internal Bus Number, and the Target Device Number is different from the internal Device Number.

The transmitted Configuration TLP includes the Target Bus, Device, Function and Register Numbers as written to the **PCI Express Configuration Address Register**.

The Configuration request generation is only enabled when the <ConfigEn> bit is set.

## 6.6 Target Memory Transactions

Target Memory transactions are memory space read and write requests (MRd, MWr TLPs) that are received over the PCI Express link, and the respective completion TLPs that are generated and transmitted in return.

The following features are supported as a target memory completer:

- Reception of up to eight Memory read requests
- Reception of up to four Memory write requests
- Maximum received read request size of 4 KB.
- Maximum received write request of 128 bytes
- Support PCI Express access to all of the device's internal registers
- 64-bit addressing
- Three Memory BARs (64-bit), BAR0 is dedicated to internal register access
- In Endpoint mode: Expansion ROM support

## 6.7 Target I/O Transactions

Target I/O transactions are I/O space read and write requests (IORd, IOWr TLPs) that are received over the PCI Express link, and the respective completion TLPs that are generated and transmitted in return.

Target I/O transactions are not supported by the device and should not be generated by the downstream device.

## 6.8 Target Configuration Transactions

Target Configuration transactions are Configuration space read and write requests (CfgRd0, CfgWr0, CfgRd1 and CfgWr1 TLPs) that are received over the PCI Express link, and the respective completion TLPs that are generated and transmitted in return.

When configured as Root Complex port, target Configuration transactions are not supported and should not be generated by downstream device. In Endpoint mode, Target Type0 Configuration transactions are supported.

The following features are supported as a target Configuration completer:

- Reception of up to eight NP requests (configuration read or write).
- Maximum received Configuration read request size of 4 bytes.
- Maximum received Configuration write request of 4 bytes

The device as an Endpoint supports responding with Configuration Request Retry Status (CRS) to a configuration read/write access. This is useful for postponing the Root Complex access until the local CPU finish initialization of the Endpoint port.

To enable this feature, set the [\*\*<Crs\\_Enable>\*\*](#) field in the PCI Express Control Register ([Table 307 p. 476](#)) to 1, prior to enable link training (meaning, prior to setting the [\*\*<PEX0En>\*\*](#) field in the CPU Control and Status Register ([Table 116 p. 368](#))). When the local CPU finish Endpoint initialization, clear the [\*\*<Crs\\_Enable>\*\*](#) field.

## 6.9 Target Special Cases

- Access attempts that fail address decoding (e.g., do not hit a memory BAR) are completed as Unsupported Requests.
- MemWr accesses to reserved, or not implemented registers, are completed normally on the PCI Express port, and the data is discarded.
- MemRd accesses to reserved, or not implemented registers, are completed normally on the PCI Express port, and a CplD TLP with data value of 0 and SC (*Successful Completion* status) is returned.

## 6.10 Messages

PCI Express defines a new message space. Messages are used to replace legacy PCI side-band signals such as interrupts, error signals, hot-plug signals etc. Messages are also used to enable new capabilities such as active power management, Slot Power Limit and others.

[Table 11](#) lists the message groups supported as a Root Complex port, and if the group is supported by the device.

**Table 11: Supported Message Groups—Root Complex Mode**

Message Group	Supported	Action
Interrupt Signaling	Yes	A received Assert_INTx message is forwarded as an interrupt to the CPU. Reception of a Deassert_INTx message clears the relevant interrupt. <b>NOTE:</b> Both INTA, INTB, INTC and INTD are supported. (x = A, B, C or D)
Power Management Event (PME)	Yes	A received PM_PME message is forwarded as an interrupt to the CPU. The log of the message is registered in the PCI Express Root Complex Power Management Event Register ( <a href="#">Table 312 p. 478</a> ). While software is handling one PME message, a new PME message may be received and registered in this register.
Error Signaling	Yes	A received Error message is forwarded as an interrupt to the CPU. Both Correctable, Non-fatal and Fatal error messages are supported.
Hot Plug Signaling	No	
Locked Transaction Support	No	
Slot Power Limit Support	No	To enable sending SSPL, set the <a href="#"><b>&lt;SsplMsgEnable&gt;</b></a> field in the PCI Express Root Complex Set Slot Power Limit Register ( <a href="#">Table 310 p. 478</a> ) to 1 before the link is up. If enabled, upon reaching link up and whenever there is a change in the <a href="#"><b>&lt;SlotPowerLimitValue&gt;</b></a> field or the <a href="#"><b>&lt;SlotPowerLimitValue&gt;</b></a> field in the PCI Express Root Complex Set Slot Power Limit Register ( <a href="#">Table 310 p. 477</a> ) a message is sent.

**Table 11: Supported Message Groups—Root Complex Mode (Continued)**

Message Group	Supported	Action
PME_TurnOff	Yes	<p>To send a Turn Off message, the <a href="#"><code>&lt;SendTurnOffMsg&gt;</code></a> field in the PCI Express Power Management Extended Register (<a href="#">Table 313 p. 479</a>) to 1. When the Endpoint reaches <i>L2</i> state, <a href="#"><code>&lt;EPReady4TurnOff&gt;</code></a> is asserted (see <a href="#"><code>&lt;RcvTurnOff&gt;</code></a> field in the PCI Express Interrupt Cause Register (<a href="#">Table 320 p. 487</a>)), indicating to the host that it may turn off the primary power of the Endpoint.</p> <p>The interrupt is set when the handshake is done (indicated by a timer or a Link state of <i>L2</i>).</p> <p>To exit this state, software must initiate a soft reset and clear the <a href="#"><code>&lt;SendTurnOffMsg&gt;</code></a> bit once the link is up again.</p>
Vendor Specific Messages	No	

[Table 11](#) lists the message groups that are supported in Endpoint mode:

**Table 12: Supported Message Groups—Endpoint Mode**

Message Group	Supported	Action
Interrupt Signaling	Yes	<p>Interrupt assertion on the internal interface is forwarded as an Interrupt Assert message to the PCI Express port.</p> <p>Interrupt de-assertion on the internal interface is forwarded as an Interrupt De-assert message to the PCI Express port.</p> <p>INTA, INTB, INTC, and INTD are supported.</p>
Power Management	Yes	<p>Receipt of a PME_Turn_Off Message triggers a maskable interrupt to the device CPU. The device CPU must prepare the device for this low power state, and acknowledge this message by setting the <a href="#"><code>&lt;SendTurnOff AckMsg&gt;</code></a> field in the PCI Express Power Management Extended Register (<a href="#">Table 313 p. 479</a>) within 1 ms. Following this, the link will start transition to L2/3 Ready. It is possible to set <a href="#"><code>&lt;SendTurnOff AckMsg&gt;</code></a> in advance to automatically activate this process.</p> <p>Generation of an PME Message is possible, by the device CPU setting the <a href="#"><code>&lt;PMESTAT&gt;</code></a> field in the PCI Express Power Management Control and Status Register (<a href="#">Table 286 p. 460</a>).</p>
Error Signaling	Yes	<p>Error in the PCI Express port is forwarded as an Error message to the PCI Express port.</p> <p>Correctable, Non-fatal, and Fatal error messages are supported.</p>
Hot Plug Signaling	No	
Locked Transaction Support	No	
Slot Power Limit Support	Yes	When Set_Slot_Power_Limit message is received, the Slot Power Limit Configuration registers are updated accordingly.
Vendor Specific Messages	No	

## 6.11

# Message Signaled Interrupts (MSI)

Message Signaled Interrupts (MSI) are supported in both Root Complex and Endpoint modes.

**Root Complex mode:** The Host sets the PCI Express MSI Message Address Register ([Table 288 p. 461](#)) to the same value that it has set the Endpoint device. A memory write received, with the same address, is handled as an MSI.

Upon receipt of MSI, an interrupt is set in the `<RcvMsi>` field in the PCI Express Interrupt Cause Register ([Table 320 p. 487](#)). Interrupt data is saved in the PCI Express MSI Message Data Register ([Table 290 p. 461](#)).

**Endpoint mode:** MSI support is required for PCI Express devices. MSI is driven by performing memory write TLP. When enabled through the PCI Express MSI Message Control Register ([Table 287 p. 460](#)), the Endpoint generates an MSI Write, for any edge interrupt assertion. The write address is set, according to the [PCI Express MSI Message Address Register](#), and for a 64-bit address, according to the PCI Express MSI Message Address (High) Register ([Table 289 p. 461](#)). The data content is as configured in the register [PCI Express MSI Message Data Register](#).

## 6.12

# Locked Transactions

Locked transaction semantics are not supported. This includes MRdLk, CplLk, and Unlock messages.

## 6.13

# Arbitration and Ordering

The arbitration scheme on both Tx and Rx directions are following the PCI Express ordering rules. For each direction there are separate queues for posted, non-posted, and completion TLPs. So TLPs can be forwarded according the ordering rules. A simple round-robin arbitration is performed on TLPs.

### 6.13.1

## Tx Ordering Rules

- All TLPs from same type (P, NP, C) are forwarded in order.
- Non-Posted transactions push posted transactions (unless the `<TxNpPushDis>` field in the PCI Express TL Control Register ([Table 318 p. 483](#)) is set to 1).
- Completions push posted transactions (unless the `<TxCmplPushDis>` field in the PCI Express TL Control Register ([Table 318 p. 483](#)) is set to 1).
- For other transaction couples, reordering may occur.
- Relaxed-ordering is not used for ordering in internal queues.
- Simple round-robin arbitration is performed on all transactions that may be transmitted to the PCI Express fabric according to those rules.

### 6.13.2

## Rx Ordering Rules

- All TLPs from same type (P, NP, C) are forwarded in order.
- Non-Posted transactions push posted transactions (unless `<RxNpPushDis>` field in the PCI Express TL Control Register ([Table 318 p. 483](#)) is set to 1).
- Completions push posted transactions (unless the `<RxCmplPushDis>` field in the PCI Express TL Control Register ([Table 318 p. 483](#)) is set to 1).
- For other transaction couples, reordering may occur.

- Relaxed-ordering is not used for ordering in internal queues.
- Simple round-robin arbitration is performed on all transactions.

## 6.14

## PCI Express Register Access

The PCI Express registers can be accessed from an external PCI Express device, or from the CPU.

The Read Only (RO) registers have the following access permissions:

- An external PCI Express device can only have read access to these registers.
- If not hardwired or set/clear by the hardware, these registers are read/write (RW) from the CPU.

If the device is configured as an endpoint, the configuration header registers can be accessed from an external Root Complex port via type0 configuration transactions.

The configuration header registers are also mapped to the chip internal address space as follows:

- All configuration header registers are mapped to the internal memory space.
- Direct memory access is enabled via the [\*\*<CfgMapTo MemEn>\*\*](#) field in the PCI Express Control Register ([Table 307 p. 475](#)). When enabled, a direct memory access can be performed from the CPU or from an external PCI Express device, even if device is configured as Root Complex port.
- If not hardwired or set/clear by the hardware, all RO configuration header registers are RW, if accessed through the direct memory mapping.

### 6.14.1

### D3<sub>hot</sub> to D0 Transition—Endpoint Mode

Switching from *D3<sub>hot</sub>* state to *D0* state is done by writing to [\*\*<PMState>\*\*](#) field in the PCI Express Power Management Control and Status Register ([Table 286 p. 459](#)).

When such transition occurs, the configuration headers are reset to default values, except the following registers:

- PCI Express Device and Vendor ID Register ([Table 271 p. 451](#))
- PCI Express Class Code and Revision ID Register ([Table 273 p. 453](#))
- PCI Express Subsystem Device and Vendor ID Register ([Table 281 p. 456](#))

### 6.14.2

### PHY Registers Access

The PCI Express PHY has its own register file. The software can access the PHY registers via the PCI Express PHY Indirect Access Register ([Table 319 p. 484](#)).



**Note** The PHY registers are for Marvell internal use (debug purposes). Do not access these registers unless explicitly directed to by a 88F6180/88F619x/88F6281 related document.

To write to a PHY register, write to the following fields in the PCI Express PHY Indirect Access Register:

- [\*\*<PhyAddr>\*\*](#) field to point to the required register offset.
- [\*\*<PhyData>\*\*](#) field to set the desired data.
- [\*\*<PhyAccessMd>\*\*](#) field set to 0.

To read from a PHY register:

- Write to PCI Express PHY Indirect Access Register with the [\*\*<PhyAddr>\*\*](#) field pointing to the required register offset, and with the [\*\*<PhyAccessMd>\*\*](#) filed set to 1.
- Read the PCI Express PHY Indirect Access Register; the read data is available in the [\*\*<PhyData>\*\*](#) field.

## 6.15 Hot Reset

Hot Reset is an in-band reset indication that can be sent from the root-complex downstream and reset the PCI Express hierarchy. Use the following procedure to generate a hot reset:

1. Write to the [`<ConfMstrHot Reset>`](#) field in the PCI Express Control Register ([Table 307 p. 475](#)).
2. To check that Hot Reset has been completed, poll the [`<DLDown>`](#) field in the PCI Express Status Register ([Table 308 p. 476](#)). When this bit is set, DL is down and Hot Reset has been completed.
3. Clear the [`<ConfMstrHot Reset>`](#) field.



Root Complex registers are not reset by Hot Reset.

**Note**

## 6.16 Link Disable

According to the *PCI Express Base Specification*, Revision 1.1, as a Root Complex, the host may disable the device connected to the PCI Express Link. To disable the link, set [`<LnkDis>`](#) field in the PCI Express Link Control Status Register ([Table 295 p. 467](#)).

## 6.17 Power Management

This section describes the PCI Express power management functions.

### 6.17.1 Software Power Management

The device supports all software Power Management options D1, D2, D3 as described in the *PCI Express Base Specification*, Revision 1.1 and also supports the Turnoff process.

**Root Complex mode:** As a Root Complex it sets the device to Turnoff state, by sending a Turnoff message, as described under *Power Management Event* in [Table 11, Supported Message Groups—Root Complex Mode, on page 79](#). When the Turnoff process is completed, a maskable interrupt is set in the PCI Express Interrupt Cause Register ([Table 320 p. 484](#)).

As a Root complex, it responds to a PME event, generated by the device. A maskable interrupt is set, upon receiving a PME message. The PME Data is saved in PCI Express Root Complex Power Management Event Register ([Table 312 p. 478](#)).

**Endpoint mode:** Upon receiving a turnoff message from the Root Complex, a maskable interrupt is set.

The device CPU acknowledges the turnoff, as described under *Power Management Event* in [Table 12, Supported Message Groups—Endpoint Mode, on page 80](#).

### 6.17.2 Active State Power Management (ASPM)

Active Power Management event is a mechanism, defined by the *PCI Express Base Specification*, Revision 1.1, that allows the hardware to lower the power state of the link. This device support all Active State Power Management (ASPM) options—both as a Root Complex and as an Endpoint.



## L1 ASPM

PCI Express driver should enable this feature as described in the PCI Express Specification.

<b>L1 ASPM Endpoint mode:</b>	When the Endpoint is enabled, the device starts the L1 ASPM event when the conditions defined in the <i>PCI Express Base Specification</i> for this event are met and when the <a href="#"><code>&lt;L1_aspm_en&gt;</code></a> field in the PCI Express Power Management Extended Register ( <a href="#">Table 313 p. 479</a> ).
<b>L1 ASPM Root Complex mode:</b>	As a root complex, the device acknowledges an L1 ASPM event only when the <a href="#"><code>&lt;L1AspmAck&gt;</code></a> field in the PCI Express Power Management Extended Register ( <a href="#">Table 313 p. 479</a> ) is set. If this bit is not set, a L1 ASPM <i>nack</i> respond is sent upon an L1 ASPM request.

## L0 ASPM

The device supports L0s event on both Rx and Tx.

## 6.18 Error Handling

This section details the error handling features.

### 6.18.1 Physical Layer Errors

[Table 13](#) lists the conditions that may cause a PHY layer Receive error.

**Table 13: Physical Layer Error List**

Error Name	Conditions
Receiver Error	<ul style="list-style-type: none"><li>• PHY Overflow</li><li>• PHY Underrun</li><li>• PHY 8B/10B decode error</li><li>• PHY Disparity error</li></ul> Severity: Correctable.

### 6.18.2 Data Link Layer Errors

[Table 14](#) lists the Data Link layer errors.

**Table 14: Data Link Layer Error List**

Error Name	Conditions
Bad TLP	<ul style="list-style-type: none"><li>• LCRC Error detected in received TLP.</li><li>• Sequence number error detected in received TLP.</li></ul> Severity: Correctable.
Bad DLLP	CRC Error detected in received DLLP. Severity: Correctable.
Replay Timeout Error	Replay timer expired. Severity: Correctable.
REPLAY_NUM Rollover Error	REPLAY_NUM rolled over. Four consecutive replays were transmitted. Severity: Correctable.
Data Link Layer Protocol Error	Reception of an Ack with out-of-range ackNac_Seq_Num. Severity: Fatal.

### 6.18.3 Transaction Layer Errors

[Table 15](#) lists the Transaction layer errors.

**Table 15: Transaction Layer Error List**

Error Name	Description
Flow Control Protocol Error	<ul style="list-style-type: none"> <li>DLLP receive timer expiration.</li> <li>Received FC initial credit values that are less than the minimum advertisement according to the spec.</li> <li>Received update FC message for a credit type that was advertised as infinite on initialization.</li> </ul> Default severity: Fatal.
Malformed TLP	<ul style="list-style-type: none"> <li>Received TLP with data payload size larger than the Maximum Payload Size.</li> <li>Received TLP with undefined <i>Type</i> and <i>Fmt</i> fields value.</li> <li>Received TLP with length different than expected according to the length, type, and TD (TLP Digest) field.</li> <li>Received request with Address/Length combination crossing the 4-KB boundary.</li> <li>Received Power Management Set_Slot_Power, Unlock, INTx, and error message with TC field not equal to 0 (TC0).</li> </ul> Default severity: Fatal.
Poisoned TLP Received.	Poisoned TLP received. Default severity: Non-fatal.
Unsupported Request	<ul style="list-style-type: none"> <li>Received unsupported TLP type (CfgWr1, CfgRd1, MrdLk).</li> <li>Received unsupported message codes.</li> <li>Failed address decoding on received TLP.</li> <li>Received CfgWr0 or CfgRd0 with function_number different than 0.</li> <li>Received poisoned write request to internal register space.</li> </ul> Default severity: Non-fatal. <b>NOTE:</b> Reception of Vendor_Defined_Type_1 message is discarded silently. It is not an error state.
Received UR Completion	<ul style="list-style-type: none"> <li>Received Cpl TLP with UR completion status.</li> <li>Received CplLk or CplID with UR completion status.</li> </ul> Not a PCI Express error. Mapped to PCI status.
Completion Timeout	Outstanding Non Posted request to PCI Express timeout has expired. Default severity: Non-fatal.
Completer Abort	Received read requests to the internal address space, with the Length field different than 1 DWORD. Default severity: Non-fatal.
Received CA completion	Received a Cpl with CA completion status Not a PCI Express error. Mapped to PCI status.
Unexpected Completion	<ul style="list-style-type: none"> <li>Received unexpected completion TLP (Cpl or CplID). Completion does not correspond to one of the outstanding NP requests.</li> <li>Received CplLk or CplDLk TLPs.</li> </ul> Default severity: Non-fatal.

### 6.18.4 Error Propagation

The PCI Express specification defines a mechanism for propagation of erroneous TLPs (erroneous data payload) via an EP (Error Poisoning) bit in the packet header.

## Receive

Upon receiving a poisoned write TLP:

- If it is not an access to the chip internal registers and if the [`<RxDPPropEn>`](#) field in the PCI Express Mbus Adapter Control Register ([Table 322 p. 488](#)) is set, an erroneous data indication is forwarded along with the data. Regardless of [`<RxDPPropEn>`](#) field setting, Error status bits in the PCI Express configuration header registers are set (if enabled by the relevant configuration registers control bits) and Error messages are sent (if enabled).
- If it is an access to the chip internal registers (whether PCI Express register file or another register file), the transaction is discarded (not written to registers). An unsupported requests completion message is sent if needed, and the relevant error status bits are set.

Upon receiving a poisoned completion TLP, if the [`<RxDPPropEn>`](#) field is set, an erroneous data indication is forwarded along with the data. Regardless of [`<RxDPPropEn>`](#) field setting, error status field are set, if enabled by the relevant control bits.

## Transmit

Tx error forwarding is controlled by the [`<TxDPPropEn>`](#) field in the PCI Express Mbus Adapter Control Register ([Table 322 p. 488](#)). For either requests or responses, the corresponding TLP is poisoned (EP bit is set) if:

- Data received from the Mbus is marked as erroneous.
- Forwarding is enabled by the [`<TxDPPropEn>`](#) field.

## 6.18.5 Completion Timeout

The Completion timeout (Cpl TO) mechanism is defined in the PCI Express specification. When a device issues a NP request on the PCI Express port and does not receive all the related completions of the request after a predefined period of time, the device must indicate a Completion Timeout error status.

The Cpl TO period is set by the [`<ConfCmpToThrshld>`](#) field in the PCI Express Completion Timeout Register ([Table 311 p. 478](#)). The Cpl TO mechanism can also be disabled through this register field.

If the Cpl TO expires before a Tx NP request is completed (not all completion fragments arrived):

- The relevant error status bits are set.
- An error message is transmitted, if not masked.
- The timed-out requests are completed on the Mbus with dummy read data, and with an erroneous data indication.

## 6.19 Loopback Modes

The following DFT features are supported by the PCI Express port:

- Master Loopback
- Internal Loopback
- Slave Loopback
- Pseudo-Random Bit Sequence (PRBS) generation and checking

### 6.19.1 Master Loopback

Master Loopback mode forces the device on the other side of the PCI Express link to enter a Loopback mode and mirror all the received traffic back to its Tx side. This mode enables a self-test procedure for the PCI Express port and link.

Entering master loopback must be done before the PCI Express link is enabled. Use the following procedure:

1. Set the [`<ConfMstrLb>`](#) field in the PCI Express Control Register ([Table 307 p. 475](#)).

2. Set the <PEX0En> field in the CPU Control and Status Register ([Table 116 p. 368](#)) to enable the PCI Express port.
3. Check that the PCI Express link is ready for a loopback test by polling the <DLDown> field in the PCI Express Status Register ([Table 308 p. 476](#)). When this bit is cleared, the link is in Master Loopback mode and the loopback test can start.
4. Set the relevant registers to control the inbound traffic (BAR, address space control and the address decoding windows).
5. Generate the loopback test traffic and check that it is received correctly.

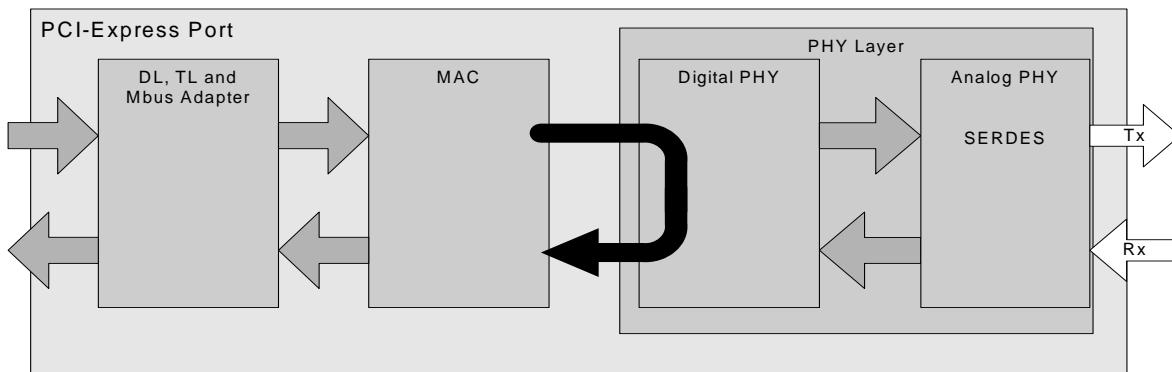
To exit Master Loopback mode, generate hot reset or reset the entire chip.

## 6.19.2 Internal Loopback

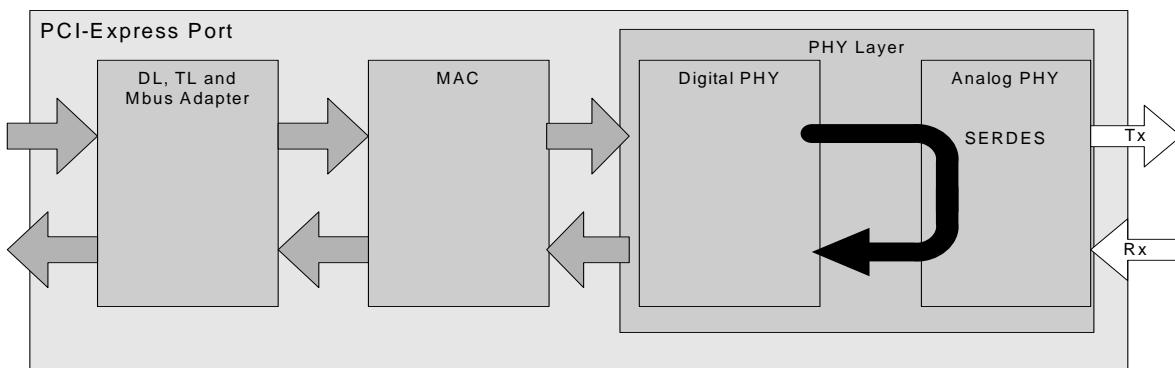
When working in Internal Loopback mode, the PCI Express port mirrors all the traffic received from the Mbus back to its Rx side. This mode enables self-test procedures for the PCI Express port, even when no external device is attached.

The PCI Express PHY supports Shallow and Deep Loopback modes as shown in [Figure 15](#) and [Figure 16](#).

**Figure 15: Shallow Internal Loopback**



**Figure 16: Deep Internal Loopback**



Use the following procedure to enter Internal Loopback mode:

1. For Shallow Loopback mode, set the PCI Express PHY Register 0x0 bit[0] to 1, using the PHY registers indirect access as explained in [Section 6.14.2, PHY Registers Access, on page 82](#). For Deep Loopback mode set PHY register 0x0 bit[1] to 1.

2. Set PCI Express PHY register 0x44 bits[9:8] to 0x3, and register 0x81 bits [11:10] to 0x2.
3. Set the [\*\*<PEX0En>\*\*](#) field in the CPU Control and Status Register ([Table 116 p. 368](#)) to enable the PCI Express port.
4. Check that the PCI Express Link is ready for the loopback test by polling the [\*\*<DLDown>\*\*](#) field in the PCI Express Status Register ([Table 308 p. 476](#)). When this bit is cleared, the loopback test can start.
5. Set the relevant registers to control the inbound traffic (BAR, address space control and the address decoding windows).
6. Generate the loopback test traffic and check that it is received correctly.

To exit Internal Loopback mode, generate hot reset or reset the entire chip.

### 6.19.3

### Pseudo-Random Bit Sequence (PRBS)

When using an internal loopback, it is possible to use PHY PRBS generation and checking, rather than activate the entire chip in order to generate traffic.

To enable PRBS generation, set PHY register 0x40 bit[2] to 1. PRBS errors are counted in PHY registers 0x50 and 0x51. To reset the PRBS error counter, set PHY register 0x40 bit[3] to 1.

### 6.19.4

### Slave Loopback

Slave loopback is the opposite case of Master loopback. This procedure is initiated and controlled by the external PCI Express device.

## 6.20

### Peer-to-Peer Traffic

The device supports the following PCI Express peer-to-peer traffic:

- Memory and I/O transactions from the PCI interface to PCI Express ports.
- Memory transactions from the PCI Express ports to PCI interface.

The device does not comply with PCI transparent bridge specification:

- It does not implement a PCI bridge configuration header.
- It does not support type1 configuration cycles forwarding.
- It does not comply with the errors forwarding specification.

However, it is still very useful as a non-transparent bridge:

- It supports forwarding of memory and I/O transactions.
- It supports forwarding of PCI interrupts to PCI Express interrupt messages.
- It supports configuration cycle forwarding via indirect access, using the PCI Express Configuration Address Register ([Table 269 p. 450](#)) and PCI Express Configuration Data Register ([Table 270 p. 450](#)).



#### Note

The PCI bus supports nonconsecutive byte enable within a burst write. The PCI Express does not support nonconsecutive byte enable within a burst write. If the PCI bus write traffic is subject to nonconsecutive byte enable, PCI-to-PCI Express traffic is not supported.

# 7

## **Serial-ATA (SATA) II Interface (88F619x and 88F6281 Only)**

The 88F6192 and 88F6281 devices integrate two Serial-ATA (SATA) II compliant ports.

The 88F6190 device integrates one Serial-ATA (SATA) II compliant port.

Unless specifically noted, the interface information in this section refers to a single port. Both ports are identical and have the same features.

Based on the Marvell® SATA host controllers (SATAHC) and SATA proven technology. The 88F619x and 88F6281 are fully compatible with SATA II phase 1.0 specification (Extension to SATA I specification).

The 88F619x and 88F6281 employ the latest SATA II PHY technology, with 3.0 Gbps (Gen2i) and backwards compatible with 1.5 Gbps (Gen1i) SATA I. The Marvell 88F619x and 88F6281 SATA II PHY accommodates the following features:

- SATA II 3 Gb/s speed
- Backwards compatible with SATA I PHYs and devices<sup>1</sup>
- Support Spread Spectrum Clocking (SSC)
- Programmable PHY for industry leading backplane drive capability
- SATA II power management compliant
- SATA II Device Hot-Swap compliant
- Low power consumption — Less than 200 mW per SATA II PHY
- PHY isolation Debug mode

The SATA II interface supports the following protocols:

- Non Data type command
- PIO read command
- PIO write command
- DMA read command
- DMA write command
- Queued DMA read command
- Queued DMA write command
- Read FPDMAQueued command
- Write FPDMAQueued command

The SATA II interface does not support the following protocols:

- ATAPI (Packet) command
- CFA commands

### 7.1

#### **Serial ATA II Host Controller (SATAHC)**

The 88F619x and 88F6281 incorporate a Serial-ATA (SATA) host controller (SATAHC). The SATAHC consists of the SATA ports with an Enhanced DMA (EDMA) that controls each port.

The sub-sections below provide detailed information about the SATAHC.

---

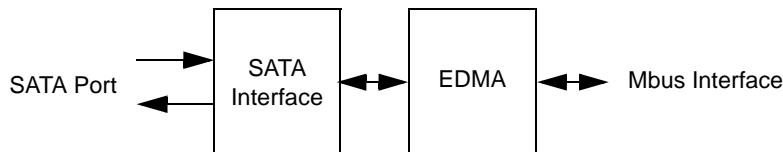
1. AC coupling is still required while working with Gen1 devices.

## 7.2

## SATAHC Block Diagram

Figure 17 provides a SATAHC block diagram, showing the flow to/from each SATA port, SATA interface, EDMA, and the Mbus Interface.

Figure 17: SATAHC Block Diagram



### 7.2.1

### SATAHC EDMA

The SATAHC EDMA:

- Controls the ATA transactions associated with its port
- Contains a 0.5-KB buffer for posted write and prefetch read transactions
- Contains the registers that control the EDMA operation

### 7.2.2

### SATA Interface

The SATA interface is compliant with the Serial-ATA II Phase 1.0 specification (Extension to SATA I specification). SATA interface features are listed above.

## 7.3

## SATAHC Initialization

### 7.3.1

### Interrupt Coalescing

The command execution can be accomplished with or without using the coalescing mechanism:

- If the interrupt coalescing mechanism is used, initialize the following registers:
  - a) SATAHC Interrupt Coalescing Threshold Register ([Table 351 p. 511](#))
  - b) SATAHC Interrupt Time Threshold Register ([Table 352 p. 511](#))
- If Interrupt coalescing mechanism is not used, the <SataCoalDone> field in the SATAHC Main Interrupt Mask Register ([Table 355 p. 514](#)) should be masked.

### 7.3.2

### Unused SATA Port

The unused SATA port should be shut down to save power by clearing the appropriate <PhyShutdown> field in the Serial-ATA Interface Configuration Register ([Table 365 p. 519](#)).

## 7.4

## Host Direct Control Over the Hard Disk Drive

- When the EDMA is disabled, the <eEnEDMA> field in the EDMA Command Register ([Table 342 p. 505](#)) is cleared.
  - The host has direct control over the device through the ATA task registers (see [Table 399, Shadow Register Block Registers Map, on page 549](#)).
- When the EDMA is enabled, the <eEnEDMA> is set.
  - The EDMA has full control over the hard disk drive (HDD).
  - If any of the ATA task registers are written, a write transaction results in unpredictable behavior.

## 7.5 LED Indications

For each SATA port, there are two LED indications:

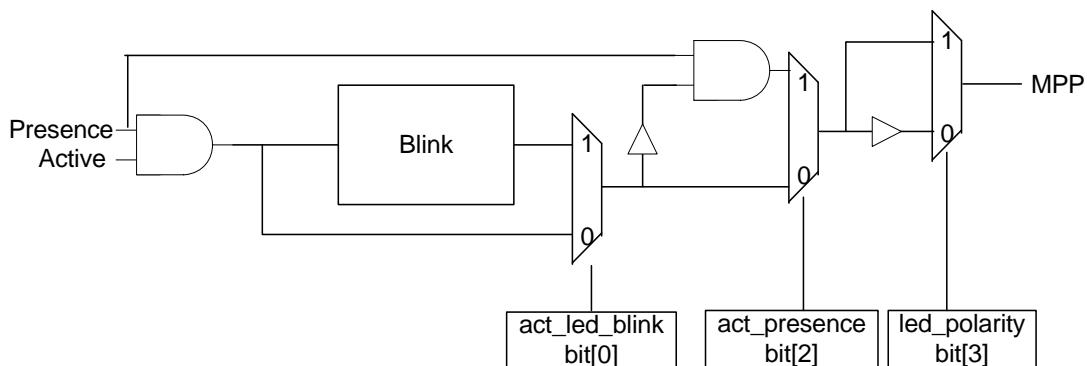
- Disk present indication
- Disk active indication

These LED indications have to be selected through the MPP interface.

Optionally by setting the GPIO Blink Enable Register ([Table 769 p. 763](#)), the LED indication for both the SATA and GPIO LEDs may blink.

The Disc Active or Presence LED indication is determined by the SATAHC LED Configuration Register ([Table 356 p. 515](#)). [Figure 18](#) shows the flow that sets the LED state. [Table 16, Disc Status LED State Settings, on page 91](#) explains the function of the bits shown in [Figure 18](#).

**Figure 18: Disc Status LED Indication Diagram**



**Table 16: Disc Status LED State Settings**

Bit Number	Bit Name	Bit Function
0	<a href="#"><code>&lt;act_led_blink&gt;</code></a>	0 = Use indication as is. 1 = Set indication to blinking.
2	<a href="#"><code>&lt;act_presence&gt;</code></a>	0 = Use active indication only. 1 = Multiplex active and presence indication on the same LED.
3	<a href="#"><code>&lt;led_polarity&gt;</code></a>	0 = Invert the active indication. 1 = Do not change the active indication.

## 7.6 EDMA Operation

The interface between host CPU and the EDMA consists of two queues: the request queue and the response queue. The request queue is the interface that the host CPU uses to queue ATA DMA commands as a request between the system memory and the device. The response queue is the interface that the EDMA uses to notify the host CPU that a data transaction between the system memory and the device was completed. Each entry in the request queue consists of an ATA DMA command and the EDMA parameters and descriptors to initiate the device and to perform the data transaction.

The EDMA is further responsible for parsing the commands, initializing the device, controlling the data transactions, verifying the device status, and updating the response queue when the command is completed. This all occurs without CPU intervention. Direct access to the device is also supported for device initialization and error handling.

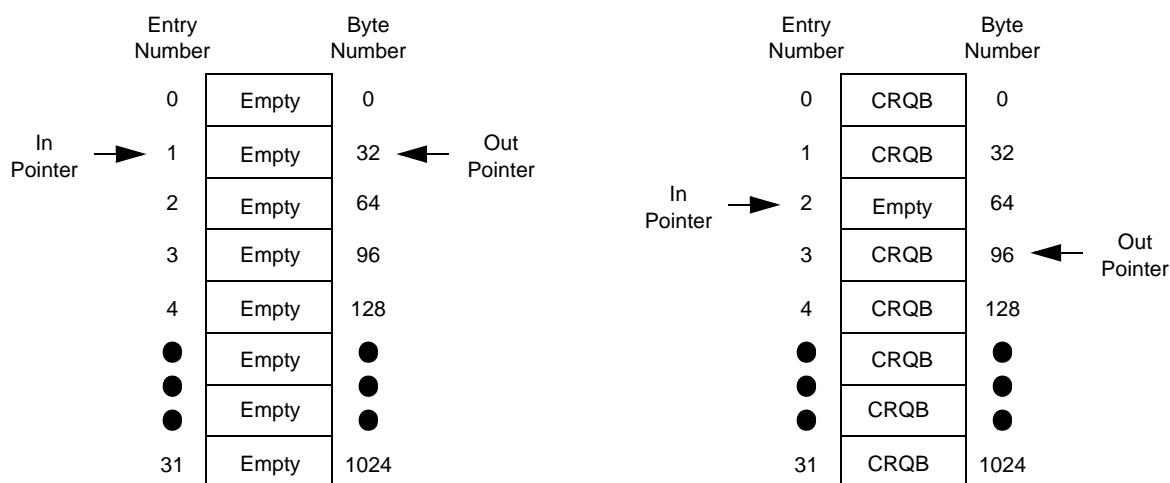
### 7.6.1 EDMA Request and Response Queues

The request queue and the response queue are each located in CPU memory and organized as a length of 32 entries, circular queues (FIFO) whose location is configured by the Queue In-Pointer and the Queue Out-Pointer entries. Since these pointers are implemented as indexes and each entry in the queue is a fixed length, the pointer can be converted to an address using the formula:  
 Entry address = Queue Base address + (entry length \* pointer value).

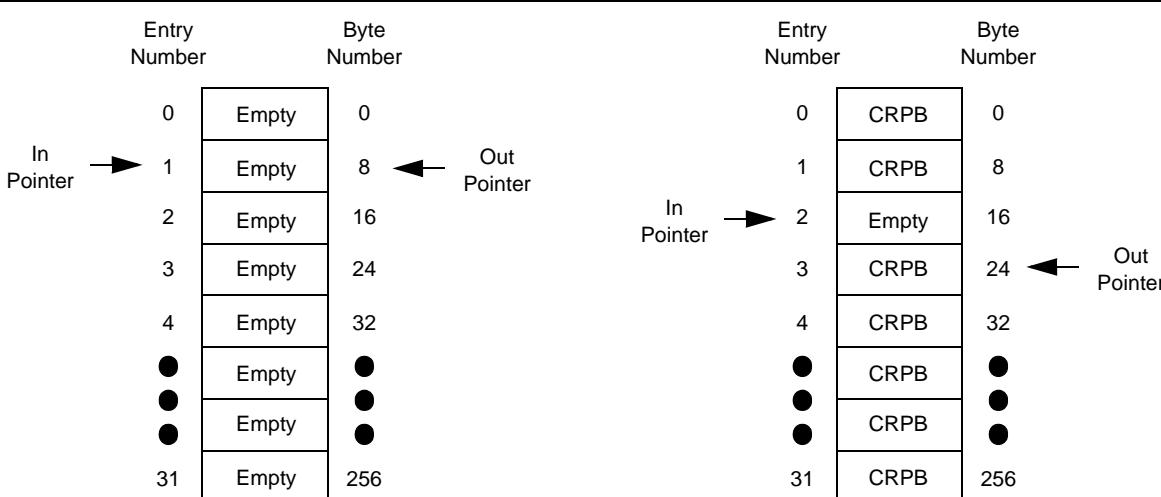
The request queue is the interface that the CPU software uses to queue ATA DMA commands as a request for a data transaction between the system memory and the device. Each entry in the request queue is 32 bytes in length, consisting of a command tag, the EDMA parameters, and the ATA device command to initiate the device and to perform the data transaction.

The response queue is the interface that the EDMA uses to notify the CPU software that a data transaction between the system memory and the device has completed. Each entry in the response queue is 8 bytes in length, consisting of the command tag and the response flags.

**Figure 19: Command Request Queue—32 Entries**



**Figure 20: Command Response Queue—32 Entries**



## 7.6.2 EDMA Configuration

The EDMA configuration is determined according by the EDMA Command Register ([Table 342 p. 505](#)). The registers listed below may be changed only when `<eEnEDMA>` in that register, is cleared, and the EDMA is disabled. The following registers must not be changed when `<eEnEDMA>` is set.

- SATAHC Configuration Register ([Table 348 p. 509](#))
- EDMA Configuration Register ([Table 333 p. 498](#))
- EDMA Command Delay Threshold Register ([Table 345 p. 508](#))
- All registers in [Table 399, Shadow Register Block Registers Map, on page 549](#), except that the host is allowed to change the `<HOB>` field (bit [7]) in the ATA Device Control register (offset 0x82120) while the EDMA is active.
- All [Basic DMA Registers \(page 494\)](#)
- All [Serial-ATA Interface Registers \(page 518\)](#)
  - FIS Interrupt Cause Register ([Table 384 p. 541](#))
  - FIS Interrupt Mask Register ([Table 385 p. 543](#))

## 7.6.3 EDMA Mode of Operation

### 7.6.3.1 Basic DMA Operation

When the `<eEnEDMA>` field in the EDMA Command Register ([Table 342 p. 505](#)) is cleared to 0, the EDMA is disabled, therefore, the request queue and the response queue are not in use. The Basic DMA may be controlled directly using the following registers:

- Basic DMA Command Register ([Table 327 p. 494](#))
- Basic DMA Status Register ([Table 328 p. 495](#))
- Descriptor Table Low Base Address Register ([Table 329 p. 496](#))
- Descriptor Table High Base Address Register ([Table 330 p. 497](#))
- SATAHC Interrupt Cause Register ([Table 353 p. 512](#))

The DMA is used to perform only DMA data transactions. The hard drive must be programmed by writing to the ATA task registers before activating basic DMA.



When in Basic DMA Operation mode, the commands are processed one by one: the host configures the device, configures the DMA and starts it. The DMA indicates completion of the data transaction by setting [`<SaCrbp0Done/DMA0Done>`](#) field in the SATAHC Interrupt Cause Register ([Table 353 p. 512](#)) and an interrupt is generated. If an error occurs during execution of the data transfer, the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)) is updated with the error cause, the Basic DMA Status Register ([Table 328 p. 495](#)) is updated with the completion error, and the host is further responsible for error handling.

### Host Initialization of Basic DMA Operation

The host initializes the DMA Read/Write operation as follows:

1. Initializes the device with the data transfer command.
2. Initializes the Physical Region Descriptor [PRD] in memory.
3. Initializes the Descriptor Table Low Base Address Register ([Table 329 p. 496](#)).
4. Initializes the Descriptor Table High Base Address Register ([Table 330 p. 497](#)).
5. Confirms that the [`<eEarlyCompletionEn>`](#) field in the EDMA Configuration Register ([Table 333 p. 499](#)) is clear to 0.
6. If Port Multiplier is used, initializes the [`<PMportTx>`](#) field in the Serial-ATA Interface Control Register ([Table 379 p. 535](#)).
7. Activates the Basic DMA by setting the control bits in the [Basic DMA Command Register](#).

### Basic DMA Read/Write Operation

The Basic DMA performs only the data transaction.

1. The Basic DMA performs the data transaction.
2. It sets [`<SaCrbp0Done/DMA0Done>`](#) and a maskable interrupt is generated.
3. The host is further responsible for the device completion status.

### Stop Basic DMA

The host may stop the Basic DMA operation before the commands are completed.

1. The host clears the [`<Start>`](#) field in the Basic DMA Command Register ([Table 327 p. 494](#)).
2. If the [`<Start>`](#) field is cleared while the Basic DMA is still active, as indicated by the active bit in the [Basic DMA Status Register](#), the Basic DMA command is aborted, and the data transferred may be discarded before reaching its destination.

#### 7.6.3.2 Target Mode Operation

When [`<eEnEDMA>`](#) field in the EDMA Command Register ([Table 342 p. 505](#)) is cleared to 0 and the [`<ComChannel>`](#) field in the Serial-ATA Interface Configuration Register ([Table 365 p. 519](#)) is set to 1, a communication channel is opened with Serial-ATA port of another 88F619x and 88F6281 (or any other Marvell® devices that support target mode). The communication channel is not symmetric: one side should be configured as an initiator ([`<TargetMode>`](#) field in the Serial-ATA Interface Configuration Register ([Table 365 p. 519](#)) is set to 0) while the other side is configured as a target ([`<TargetMode>`](#) is set to 1).

### Full Communication

The two channel should be used as follows:

1. In channel A—The SATA port in the device is configured as the initiator, while the companion SATA port in the other device is configured as the target.
2. In channel B—The SATA port in the device is configured as the target, while the companion SATA port in the other device is configured as the initiator.

## Initiate Basic DMA Read Operation

A Basic DMA read operation is implemented as follows:

1. Initiator Host—Activate the Initiator Basic DMA.
2. Initiator Host—Send the Register Device to the Host FIS (Frame Information Structure) using the Vendor Unique interface with 10-byte command (see [Section 7.8, Vendor Unique, on page 112](#)).
3. Target Transport—Update ATA task registers, set the port's `<SaDevInterrupt0>` field in the SATAHC Interrupt Cause Register ([Table 353 p. 513](#)), and generate the interrupt.
4. Target Host—Send the Register Device to the Host FIS using the Vendor Unique interface with acknowledge.
5. Initiator Transport—Update the ATA task registers, optionally set the port's `<SaDevInterrupt0>` field, and generate the interrupt if specified in the Register Device to the Host FIS.
6. Target Host—Activate Basic DMA, set `<eDMAActivate>` field in the Serial-ATA Interface Control Register ([Table 379 p. 536](#)).
7. Target Transport—Send the data as configured in the target Basic DMA.
8. Initiator Basic DMA—Set the port's `<SaCrb0Done/DMA0Done>` field and generate the interrupt to the initiator host when data transfer completes.
9. Target Basic DMA—Set the port's `<SaCrb0Done/DMA0Done>` field and generate the interrupt to the target host when data transfer completes.

## Initiate Basic DMA Write Operation

A Basic DMA write operation is implemented as follows:

1. Initiator Host—Activate Initiator Basic DMA.
2. Initiator Host—Send Register Device to Host FIS using the Vendor Unique interface with 10-byte command (see [Section 7.8, Vendor Unique, on page 112](#)).
3. Target Transport—Update ATA task registers and set the port's `<SaDevInterrupt0>` field in the SATAHC Interrupt Cause Register ([Table 353 p. 513](#)) and generate the interrupt.
4. Target Host—Send Register Device to Host FIS using the Vendor Unique interface with acknowledge.
5. Initiator Transport—Update the ATA task registers and optionally set the port's `<SaDevInterrupt0>` field and generate the interrupt if specified in the Register Device to Host FIS.
6. Target Host—Activate the Basic DMA, send DMA Activate frame using the Vendor Unique interface.
7. Initiator Transport—Set the `<eDMAActivate>` field in the Serial-ATA Interface Control Register ([Table 379 p. 536](#)).
8. Initiator Transport—Send the data as configured in the initiator Basic DMA.
9. Initiator Basic DMA—Set the port's `<SaCrb0Done/DMA0Done>` field and generate the interrupt to initiator host when the data transfer completes.
10. Target Basic DMA—Set the port's `<SaCrb0Done/DMA0Done>` field and generate the interrupt to target host when the data transfer completes.

**Note**

- In this mode, the ATA task registers are updated when Register Device to Host FIS is received regardless to the value of <BSY> bit in the ATA Status register (see [Table 399, Shadow Register Block Registers Map, on page 549](#)).
- Link Errors while transmitting Vendor Unique FIS are also reported in the <LinkCtlTxErr> field in the EDMA Interrupt Error Cause Register ([Table 334 p. 502](#)).
- For testing, this mode can be used to generate an External Loopback between the Serial-ATA ports of the same device.

### 7.6.3.3

#### Non-queued DMA Commands

When the <eEnEDMA> field in the EDMA Command Register ([Table 342 p. 505](#)) is set to 1, the <eSATANatvCmdQue> field in the EDMA Configuration Register ([Table 333 p. 498](#)) is cleared to 0, and the <eQue> is clear to 0, the EDMA is in Non-queued mode. In this mode, the EDMA supports only ATA DMA commands. It performs the commands that reside in the CRQB one by one. The next command is issued to the device only when the previous command has completed and the CRPB is updated. In this mode, the EDMA uses the following commands.

- Read DMA
- Read DMA EXT
- Write DMA
- Write DMA EXT
- Read STREAM DMA
- Write DMA FUA EXT
- Write STREAM DMA

### 7.6.3.4

#### Queued DMA Commands

When the <eEnEDMA> field in the EDMA Command Register ([Table 342 p. 505](#)) is set to 1, the <eSATANatvCmdQue> field in the EDMA Configuration Register ([Table 333 p. 498](#)) is clear to 0, and the <eQue> is set to 1, ATA QDMA commands are performed. These commands allows the CPU to issue concurrent commands to the same device. Along with the command, the EDMA provides the <cDeviceQueTag> to the device to uniquely identify the command. When the device restores register parameters during the execution of the SERVICE command, this tag is restored. The EDMA identify the command according to the <cDeviceQueTag> and the incoming PM port and restores the command parameters to execute the data transaction. The ATA devices support up to 32 concurrent queued commands, and these commands may perform out of order.

In this modes the EDMA uses the following commands:

- Read DMA Queued
- Read DMA Queued EXT
- Write DMA Queued
- Write DMA Queued EXT
- Write DMA Queued FUA EXT

### 7.6.3.5

#### SATA Native Command Queuing

When the <eEnEDMA> field in the EDMA Command Register ([Table 342 p. 505](#)) is set to 1 and the <eSATANatvCmdQue> field in the EDMA Configuration Register ([Table 333 p. 498](#)) is set to 1, a streamlined command queuing model for SATA (SATA native command queuing) is supported. This model minimizes the required number of protocol round trips and reduces the incurred overhead.

These commands allows the CPU to issue concurrent commands to the same device. Along with the command, the EDMA provides the <cDeviceQueTag> as the tag of the command to uniquely identify

the command. When the device restores register parameters, this tag is restored. The EDMA identify the command according to the [`<cDeviceQueTag>`](#) and the incoming PM port and restores the command parameters to execute the data transaction. The SATA devices support up to 32 concurrent queued commands, and these commands may perform out of order.

In this mode the EDMA uses the following commands:

- Read FPDMA Queued
- Write FPDMA Queued

## 7.6.4

### EDMA Activation

The CPU activates the EDMA according to the following flow:

1. Verifies that the device is ready to receive data commands, the [`<DET>`](#) field in the SStatus Register ([Table 367 p. 522](#)) equals 3, and the fields [`<Busy>`](#) and [`<DRQ>`](#) in the device Status register are cleared.
2. Clears the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)) and clears the appropriate [`<SaCrbp0Done/DMA0Done>`](#) field in the SATAHC Interrupt Cause Register ([Table 353 p. 512](#)).
3. Initializes the EDMA Configuration Register ([Table 333 p. 498](#)).
4. Clears the FIS Interrupt Cause Register ([Table 384 p. 541](#)).
5. Initialized the FIS Configuration Register ([Table 383 p. 540](#)).
6. Initializes the EDMA Request Queue In-Pointer Register ([Table 337 p. 504](#)).
7. Initializes the EDMA Request Queue Out-Pointer Register ([Table 338 p. 504](#)).
8. Initializes the EDMA Response Queue In-Pointer Register ([Table 340 p. 504](#)).
9. Initializes the EDMA Response Queue Out-Pointer Register ([Table 341 p. 505](#)).
10. Activates the EDMA by writing 1 to [`<eEnEDMA>`](#) field in the EDMA Command Register ([Table 342 p. 505](#)).

While the EDMA is enabled, the host should not access the registers listed in [Section 7.6.2, EDMA Configuration, on page 93](#). The CPU accesses these registers for direct access to the device when the EDMA is disabled. Accessing the above registers while EDMA is enabled—see the [`<eEnEDMA>`](#) field—will result in unpredictable behavior.

## 7.6.5

### Commands Hot Insertion to EDMA Queue

Hot insertion of commands into the EDMA queue follows these steps:

1. Sets the valid Physical Region Descriptors [PRD] for the new commands.
2. Initializes the new commands in the request queue.
3. Updates the EDMA Request Queue In-Pointer Register ([Table 337 p. 504](#)), to enable EDMA access to the new CRQBs in the request queue.

## 7.6.6

### Stop EDMA

To stop the EDMA operation, the CPU should sets the [`<eDsEDMA>`](#) field in the EDMA Command Register ([Table 342 p. 506](#)) to 1. The EDMA stops queue processing, aborts the current command and clears [`<eEnEDMA>`](#).

If EDMA is aborted during commands processing, the host must set the [`<eAtaRst>`](#) field in the EDMA Command Register ([Table 342 p. 506](#)) to recover.

## 7.6.7

### Restart EDMA

To restart the queue, the CPU must follow the EDMA activation flow (see [Section 7.6.4, EDMA Activation, on page 97](#)).

## 7.6.8 Device Link Disconnect

See [Device Disconnect on page 101](#) for the disconnect procedure.

Since loss of the link may occur at any time during EDMA programming, when the CPU receives a link-down interrupt, it must wait for re-establishment of the link (see [Section 7.6.9, Device Link Connect, on page 98](#)).

## 7.6.9 Device Link Connect

When the link to the device is renewed, the EDMA sets the `<eDevCon>` field in the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)).

Device hard reset (setting the `<eAtaRst>` field in the EDMA Command Register ([Table 342 p. 506](#)) and device initialization are required before any attempt to access the device.

## 7.6.10 EDMA Read Burst Limit

The `<eRdBSz>` field in the EDMA Configuration Register ([Table 333 p. 498](#)) defines the maximum burst read transactions SATAHC initiates towards the Mbus. The EDMA supports a maximum read burst size of 128B.

## 7.6.11 EDMA Write Burst Limit

The EDMA support a maximum write burst size of 128B.

## 7.6.12 Port Multiplier Support

The 88F619x and 88F6281 support the Port Multiplier (PM) ingredient in the following modes:

### 7.6.12.1 Port Multiplier—Command Based Switching

When the `<eEDMAFBS>` field in the EDMA Configuration Register ([Table 333 p. 499](#)) is cleared to 0, the EDMA Performs Command Based Switching as defined in SATA working group PM definition. Field `<cPMport>` in each command in the request queue (CRQB) is used to define the specific port in the Port Multiplier (PM) ingredient that belongs to this command. The EDMA is further responsible for forwarding the commands to the correct target device. In this mode, Non Queued DMA commands are supported (see [Section 7.6.3.3, Non-queued DMA Commands, on page 96](#)).

### 7.6.12.2 Port Multiplier—FIS-Based Switching

The EDMA performs FIS-based switching, as defined in SATA working group PM definition. In this mode, the EDMA issues multiple outstanding commands across multiple devices at the same time. The overall system performance increases significantly with this type of switching.

The following commands are supported in this mode:

- Non Queued DMA commands (see [Section 7.6.3.3, Non-queued DMA Commands, on page 96](#))
- Tag Command Queuing (TCQ) commands (see [Section 7.6.3.4, Queued DMA Commands, on page 96](#))
- Native Command Queuing commands (see [Section 7.6.3.5, SATA Native Command Queuing, on page 96](#))



The mode is selected before EDMA is enabled. It must not be changed when the `<eEnEDMA>` field in the EDMA Command Register ([Table 342 p. 505](#)) is set to 1.

## 7.6.13 Asynchronous Device Notification

When Set Device Bits (SDB) FIS is received with N bit set to 1, the following occurs:

In the FIS Configuration Register ([Table 383 p. 540](#)):

If [`<FISWait4RdyEn>`](#) is cleared to 0:

the device ignores the FIS.

If bit [`<FISWait4RdyEn>\[1\]`](#) is set to 1:

- Bit [1] of the [`<FISWait4HostRdyEn>`](#) field in the FIS Configuration Register ([Table 383 p. 540](#)) is set.
- The [`<eTransInt>`](#) field in the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)) is also set if the corresponding bit in the FIS Interrupt Mask Register ([Table 385 p. 543](#)) is set to 1.

## 7.6.14 EDMA Interrupts

### 7.6.14.1 Error indication

The EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)), provides the various error indications that may occur during DMA operation. For more information (see [Section 7.6.15, Error Handling, on page 100](#)).

In addition, the DMA contains a EDMA Interrupt Error Mask Register ([Table 335 p. 503](#)). This register may be used to mask the error bits in the [EDMA Interrupt Error Cause Register](#). If one (or more) of the unmasked bits in the [EDMA Interrupt Error Cause Register](#) is set, an error indication is propagated to the SATAHC Main Interrupt Cause Register ([Table 354 p. 513](#)).

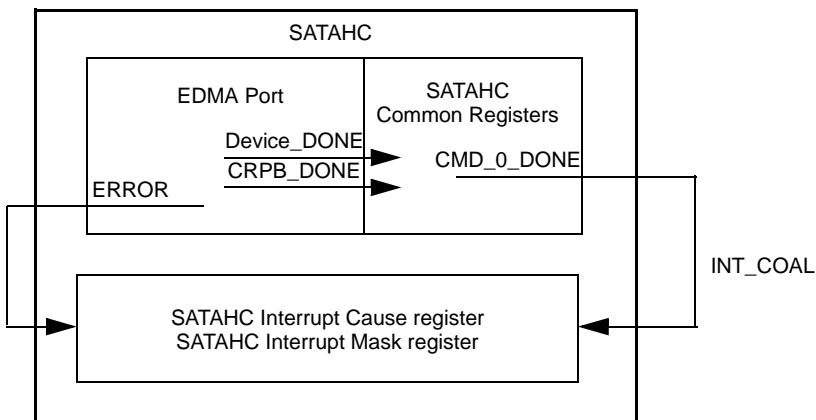
### 7.6.14.2 Command Completion Indication

The SATAHC Interrupt Cause Register ([Table 353 p. 512](#)), resides in the SATAHC arbiter. The command completion indications are propagated from the EDMAs to the appropriate bit in this register. The indications from the register are further propagated to the [SATAHC Main Interrupt Cause Register](#).

When the EDMA completes an ATA transaction:

- The last data leaves the device.
- The CRPB is updated.
- The EDMA indicates the appropriate bit in the [SATAHC Interrupt Cause Register](#), and
- An interrupt indication is propagated to [SATAHC Main Interrupt Cause Register](#).

Figure 21: EDMA Interrupt Hierarchy<sup>1</sup>



#### 7.6.14.3 Interrupt Coalescing

Since the SATA ports provide a high data rate, it is important to reduce the number of interrupts that the SATA EDMAs may generate. The device provides an interrupt coalescing mechanism that sets the interrupt coalescing bit in the SATAHC Interrupt Cause Register (Table 353 p. 512), and propagates an interrupt indication if one of the following is true:

- The number of EDMA commands reached the SATAHC interrupt coalescing threshold value (see the SATAHC Interrupt Coalescing Threshold Register (Table 351 p. 511)).
- At least one EDMA commands has completed and the time that passed since its completion reached the SATAHC interrupt time threshold value (see the SATAHC Interrupt Time Threshold Register (Table 352 p. 511)).

#### 7.6.14.4 Device Interrupt

When the EDMA is active, the device interrupt request is masked. When the EDMA is disabled and the device interrupt request is active, a separate bit is set in the [SATAHC Interrupt Cause Register](#) and a command completion indication is propagated to the [SATAHC Main Interrupt Cause Register](#).

#### 7.6.15 Error Handling

Error indications from all layers are gathered in the EDMA Interrupt Error Cause Register (Table 334 p. 501).

##### 7.6.15.1 List Of Unrecoverable Errors

The following bits in the [EDMA Interrupt Error Cause Register](#) list the unrecoverable errors:

- <eDevDis>
- <eIORdyErr>
- <LinkCtlRxErr>
- <LinkDataRxErr>
- <LinkDataTxErr>
- <TransProtErr>

When an unrecoverable error indication is set from the list above, the EDMA is self disabled and the host must set bit <eAtaRst> field in the EDMA Command Register (Table 342 p. 506) to recover.

1. All interrupt indications from the SATAHC are propagated to the SATAHC Main Interrupt Cause Register (Table 354 p. 513).

### 7.6.15.2 PHY Layer Errors

#### SError Register Errors

For PHY layer errors, see the SError Register ([Table 368 p. 523](#)).

The `<SerrInt>` field in the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)) is set when at least one bit in `SError Register` is set to 1, and the corresponding bit in the `SError Interrupt Mask Register` is enabled.

#### Device Disconnect

When the device is disconnected, the EDMA halts and:

- Sets the `<eDevDis>` field in the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)).
- Disables the EDMA operation by clearing bit `<eEnEDMA>` field in the EDMA Command Register ([Table 342 p. 505](#)).
- Sets the `<eSelfDis>` field in the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)).

Host must set the `<eAtaRst>` field in the EDMA Command Register ([Table 342 p. 506](#)) to recover.

The CPU is responsible for error recovery.

### 7.6.15.3 Link Layer Errors

#### Serial-ATA II Link Layer Error During Reception of a Control Frame

- **Transient Errors:** When the following errors occur during control FIS reception. The link layer responds with R\_ERR to the received frame. The transport layer drop this frame and waits for re-transmission of the frame. This may be a transient error. The EDMA ignore these type of errors and proceeds with normal operation.
  - Serial-ATA CRC error occurs. `<LinkCtlRxErr>` field in the EDMA Interrupt Error Cause Register ([Table 334 p. 502](#)) bit [0] is set.
  - Internal FIFO error occurs. `<LinkCtlRxErr>` bit [1] is set to 1.
  - Link state errors, coding errors, or running disparity errors occur. Bit [3] of the `<LinkCtlRxErr>` field is set.
- **Non Transient Errors:** When the following error occurs during control FIS reception. The transport layer goes to protocol error state. The host must sets bit `<eAtaRst>` to recover.
  - The Link Layer is reset (to Idle state) by the reception of SYNC primitives from the device. Bit [2] in the `<LinkCtlRxErr>` field and the `<TransProtErr>` field are set.

#### Serial-ATA II Link Layer Error During Reception of a Data Frame

When the Link Layer is reset (to Idle state) by the reception of SYNC primitives from the device, the transport layer goes to protocol error state, bit [2] in the `<LinkDataRxErr>` field and the `<TransProtErr>` field are set. The host must set bit `<eAtaRst>` to recover.

When the following errors occur during data FIS reception, the link layer responds with R\_ERR to the received frame. The transport layer ignores the error but the EDMA is self disabled.

- Serial-ATA CRC error occurs. Bit [0] of the `<LinkDataRxErr>` field is set.
- Internal FIFO error occurs. Bit [1] in the `<LinkDataRxErr>` field is set.
- Link state errors, coding errors, or running disparity errors occur. Bit [3] in the `<LinkDataRxErr>` field is set.

#### Serial-ATA II Link Layer Error During Transmission of a Control Frame

When the following errors occur during control FIS transmission, the transport layer re-transmits the frame. This may be a transient error.

- Serial-ATA CRC error occurs. Bit [0] in the `<LinkCtlRxErr>` field is set.

- Internal FIFO error occurs. Bit [1] in the [`<LinkCtlRxErr>`](#) field is set.
- The Link Layer is reset (to Idle state) by the reception of SYNC primitives from the device. Bit [2] in the [`<LinkCtlTxErr>`](#) field is set.
- Link layer accepts DMAT primitive from the device. Bit [3] in the [`<LinkCtlTxErr>`](#) field is set.
- FIS transmission is aborted due to collision with received traffic. Bit [4] in the [`<LinkCtlTxErr>`](#) field is set.

### Serial-ATA II Link Layer Error During Transmission of a Data Frame

When the following errors occur during data FIS transmission, the transport layer ignores the error, but the EDMA is self disabled.

- Serial-ATA CRC error occurs. Bit [0] in the [`<LinkDataTxErr>`](#) field is set.
- Internal FIFO error occurs. Bit [1] in the [`<LinkDataTxErr>`](#) field is set.
- The Link Layer is reset (to Idle state) by the reception of SYNC primitives from the device. Bit [2] in the [`<LinkDataTxErr>`](#) field is set.
- Link layer accepts DMAT primitive from the device. Bit [3] in the [`<LinkDataTxErr>`](#) field is set.
- FIS transmission is aborted due to collision with received traffic. Bit [4] in the [`<LinkDataTxErr>`](#) field is set.

#### 7.6.15.4 Transport Layer Errors

### Serial-ATA II Transport Layer Protocol Non Transient Errors

When a violation of the Serial-ATA protocol was detected, the transport layer goes to protocol error state and sets the [`<TransProtErr>`](#) field in the EDMA Interrupt Error Cause Register ([Table 334 p. 503](#)) and the EDMA is self disabled. The Host must set the [`<eAtaRst>`](#) field in the EDMA Command Register ([Table 342 p. 506](#)) to recover. This error state can arise from invalid or poorly formed FISs being received, from invalid state transitions, or from other causes.

The host must set [`<eAtaRst>`](#) to recover.

The CPU is responsible for error recovery.

### Device Error Indications

#### Device Errors in Non Queued or Queued DMA Commands—FIS-Based Switching Mode Disabled

FIS-Based Switching is disabled when the [`<eEDMAFBS>`](#) field in the EDMA Configuration Register ([Table 333 p. 499](#)) is cleared.



See [Section 7.6.3.3, Non-queued DMA Commands, on page 96](#) and [Section 7.6.3.4, Queued DMA Commands, on page 96](#).

Bit [2] in the [`<eHaltMask>`](#) field in the EDMA Halt Conditions Register ([Table 346 p. 508](#)) should be set to 1:

When bit [`<Error>`](#) in the ATA status register is set to 1:

- The following registers are updated with the command information:
  - Shadow Register Block Registers Map ([Table 399 p. 549](#))
  - Serial-ATA Interface Status Register ([Table 381 p. 538](#))
  - EDMA Status Register ([Table 343 p. 506](#))
  - EDMA Interrupt Error Cause Register ([Table 334 p. 501](#))

- <eDevErr> field in the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)) is set.
- The EDMA halts.

Device Error Indication in Serial-ATA Native Command Queuing

See [Section 7.6.3.5, SATA Native Command Queuing, on page 96](#) and [Section 7.6.12.2, Port Multiplier—FIS-Based Switching, on page 98](#).

Bit [2] in the <eHaltMask> field in the EDMA Halt Conditions Register ([Table 346 p. 508](#)) should be set to 1:

When bit Error in the ATA status register is set to 1, the following registers are updated with the command information:

- [Serial-ATA Interface Status Register](#)
- [EDMA Status Register](#)
- [EDMA Interrupt Error Cause Register](#)
- Host identifies which drive caused the error via the <PortNumDevErr> field in the Serial-ATA Interface Test Control Register ([Table 380 p. 537](#))

EDMA does NOT update CRPB with the error indication.

The host needs to:

- Waits for completion of all outstanding commands associate to other devices (that did not experience a device error).
- Check the <eCacheEmpty> field in the EDMA Status Register ([Table 343 p. 507](#)). If cleared, then wait for another Device error interrupt.
- Wait for <EDMAIdle> field in the EDMA Status Register ([Table 343 p. 507](#)) to clear. If set, then wait for another Device error interrupt.
  - Good CRPBs may be received.
- Set the <eDsEDMA> field in the EDMA Command Register ([Table 342 p. 506](#)) to disable EDMA operation.
- Wait for clearing of the <eEnEDMA> field.
- Issue a read log command to the drive and perform error handling accordingly.

### Device Errors in Non Queued or Queued DMA Commands in Port Multiplier—FIS-Based Switching Mode Enabled

See [Section 7.6.3.3, Non-queued DMA Commands, on page 96](#), [Section 7.6.3.4, Queued DMA Commands, on page 96](#) and [Section 7.6.12.2, Port Multiplier—FIS-Based Switching, on page 98](#)

Bit [2] in the <eHaltMask> field in the EDMA Halt Conditions Register ([Table 346 p. 508](#)) should be cleared to 0:

Bit [0] of the <FISWait4RdyEn> field in the FIS Configuration Register ([Table 383 p. 540](#)) should be set to 1:

When bit <Error> in the ATA status register is set to 1 via the Register-Device to Host FIS, the following registers are updated with the command information:

- Bit [0] of the <FISWait4HostRdy> field in the FIS Interrupt Cause Register ([Table 384 p. 542](#)) is set to 1, and the transport layer blocks reception of any new FIS until the host clears this bit.
- EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)).
- The EDMA updates the CRPB with the error indication.

The host must:

- Get detailed error information from the Shadow Register Block (see [Shadow Register Block Registers Map \(Table 399 p. 549\)](#)).
- Detect the aborted commands for the disk that experience error according to the EDMA NCQ0 Done/TCQ0 Outstanding Status Register ([Table 347 p. 509](#)).
- Clear the <eDevErr> field in the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)).



- Clear bit [0] in the [`<FISWait4HostRdy>`](#) field in the FIS Interrupt Cause Register ([Table 384 p. 542](#)).
- Wait for completion of all outstanding commands (that is, any commands to the EDMA that did not complete successfully and were not aborted or failed).
- Set bit [`<eDsEDMA>`](#) field in the EDMA Command Register ([Table 342 p. 506](#)) to disable the EDMA operation.
- Wait for clear of bit [`<eEnEDMA>`](#).

### 7.6.15.5 DMA Errors

#### Internal Parity Error

The device SATAHC is parity protected on internal memories.

The internal SRAMs contain a parity bit per entry (minimal transaction width). This bit is calculated and inserted on every write to the internal SRAMs. This bit is verified against the data when reading from the internal SRAMs.

The parity bit also indicates if errors occurred during the PCI transaction.

### 7.6.16 EDMA Data Structures

#### 7.6.16.1 Command Request Queue

The request queue is the interface that the CPU software uses to request data transactions between the system memory and the device. The request queue has a length of 32 entries. The request queue is a circular queue (FIFO) whose location is configured by the EDMA Request Queue In-Pointer Register ([Table 337 p. 504](#)), and the EDMA Request Queue Out-Pointer Register ([Table 338 p. 504](#)).

- A queue is empty when Request Queue Out-pointer reaches to the Request Queue In-pointer.
- A queue is full when Request Queue In-pointer is written with the same value as the Request Queue Out-pointer. A full queue contains 32 entries.
- A queue contains N entries when the Request Queue Out-pointer is N less than the Request Queue In-pointer, taking into account the wraparound condition.

See [Figure 19, Command Request Queue—32 Entries, on page 92](#).

Each 32-byte EDMA Command Request Block (CRQB) entry consists of EDMA parameters and commands for the ATA device. The CRQB data structure is written by the CPU. [Table 17](#) provides a map of the CRQB data structure registers.

#### 7.6.16.2 EDMA Command Request Block (CRQB) Data

**Table 17: EDMA CRQB Data Structure Map**

Register	Offset	Page
CRQB DW0—cPRD Descriptor Table Base Low Address	Offset: 0x00	<a href="#">Table 18, p. 105</a>
CRQB DW1—cPRD Descriptor Table Base High Address	Offset: 0x04	<a href="#">Table 19, p. 105</a>
CRQB DW2—Control Flags	Offset: 0x08	<a href="#">Table 20, p. 105</a>
CRQB DW3—Data Region Byte Count	Offset: 0x0C	<a href="#">Table 21, p. 106</a>
CRQB DW4—ATA Command	Offset: 0x10	<a href="#">Table 22, p. 106</a>

**Table 17: EDMA CRQB Data Structure Map (Continued)**

Register	Offset	Page
CRQB DW5—ATA Command	Offset: 0x14	<a href="#">Table 23, p. 106</a>
CRQB DW6—ATA Command	Offset: 0x18	<a href="#">Table 24, p. 107</a>
CRQB DW7—ATA Command	Offset: 0x1C	<a href="#">Table 25, p. 107</a>

**Table 18: CRQB DW0—cPRD Descriptor Table Base Low Address**

Offset: 0x00

Bits	Field	Description
31:0	cPRD[31:0]	<p>CRQB ePRD. When <code>&lt;cPRDMode&gt;</code> is cleared to 0: The CPU at initialization should construct a ePRD table in memory. This table contains consecutive descriptors that describe the data buffers allocated in memory for this command. This DWORD contains bit [31:4] of the physical starting address of this table. Bits [3:0] must be 0x0.</p> <p>When <code>&lt;cPRDMode&gt;</code> is set to 1: This DWORD contains bits [31:1] of the physical starting address of a data region in system memory. Bit [0] must be 0.</p>

**Table 19: CRQB DW1—cPRD Descriptor Table Base High Address**

Offset: 0x04

Bits	Field	Description
31:0	cPRD[63:32]	Reserved Must be set to 0x0.

**Table 20: CRQB DW2—Control Flags**

Offset: 0x08

Bits	Field	Description
0	cDIR	CRQB Direction of Data Transaction 0 = System memory to Device 1 = Device to system memory
5:1	cDeviceQueTag	CRQB Device Queue Tag This field contains the Queued commands used as tags attached to the command provided to the drive.
11:6	Reserved	Reserved Must be 0.
15:12	cPMport	PM Port Transmit This field specifies the Port Multiplier (PM) port (bits [11:8] in DW0 of the FIS header) inserted into the FISs transmission associate to this command.

**Table 20: CRQB DW2—Control Flags (Continued)**  
Offset: 0x08

Bits	Field	Description
16	cPRDMode	CRQB PRD Mode This bit defines how the physical data that resides in the system memory is described. 0 = PRD tables are being used. <cPRD[31:0]> and <cPRD[63:32]> provide the ePRD table starting address. 1 = Single data region, <cPRD[31:0]> and <cPRD[63:32]> provide its starting address. <cDataRegionByteCount> provides its length.
21:17	cHostQueTag	CRQB Host Queue Tag This 5-bit field contains the host identification of the command.
31:22	Reserved	Reserved

**Table 21: CRQB DW3—Data Region Byte Count**  
Offset: 0x0C

Bits	Field	Description
15:0	cDataRegionByteCount	Data Region Byte Count When <cPRDMode> is cleared to 0: This field is reserved. When <cPRDMode> is set to 1: This field contains the count of the region in bytes. Bit [0] is forced to 0. There is a 64 KB maximum. A value of 0 indicates 64 KB. The data in the buffer must not cross the boundary of the 32-bit address space; that is, the 32-bit high address of all data in the buffer must be identical.
31:16	Reserved	Reserved

The naming of the fields in the next four tables complies with the Serial-ATA convention. The corresponding name according to the ATA convention appears in parentheses.

**Table 22: CRQB DW4—ATA Command**  
Offset: 0x10

Bits	Field	Description
15:0	Reserved	Reserved
23:16	Command	This field contains the contents of the Command register of the Shadow Register Block (see <a href="#">Table 399 on page 549</a> ).
31:24	Features	This field contains the contents of the Features (Features Current) register of the Shadow Register Block.

**Table 23: CRQB DW5—ATA Command**  
Offset: 0x14

Bits	Field	Description
7:0	Sector Number	This field contains the contents of the Sector Number (LBA Low Current) register of the Shadow Register Block (see <a href="#">Table 399 on page 549</a> ).

**Table 23: CRQB DW5—ATA Command (Continued)**  
Offset: 0x14

Bits	Field	Description
15:8	Cylinder Low	This field contains the contents of the Cylinder Low (LBA Mid Current) register of the Shadow Register Block.
23:16	Cylinder High	This field contains the contents of the Cylinder High (LBA High Current) register of the Shadow Register Block.
31:24	Device/Head	This field contains the contents of the Device/Head (Device) register of the Shadow Register Block.

**Table 24: CRQB DW6—ATA Command**  
Offset: 0x18

Bits	Field	Description
7:0	Sector Number (Exp)	This field contains the contents of the Sector Number (Exp) (LBA Low Previous) register of the Shadow Register Block (see <a href="#">Table 399 on page 549</a> ).
15:8	Cylinder Low (Exp)	This field contains the contents of the Cylinder Low (Exp) (LBA Mid Previous) register of the Shadow Register Block
23:16	Cylinder High (Exp)	This field contains the contents of the Cylinder High (Exp) (LBA High Previous) register of the Shadow Register Block.
31:24	Features (Exp)	This field contains the contents of the Features (Exp) (Features Previous) register of the Shadow Register Block.

**Table 25: CRQB DW7—ATA Command**  
Offset: 0x1C

Bits	Field	Description
7:0	Sector Count	This field contains the contents of the Sector Count (Sector Count Current) register of the Shadow Register Block (see <a href="#">Table 399 on page 549</a> ).
15:8	Sector Count (Exp)	This field contains the contents of the Sector Count (exp) (Sector Count Previous) register of the Shadow Register Block
31:16	Reserved	Reserved

#### When the EDMA is in Non-Queued mode:

The following commands are supported.

- READ DMA
- READ DMA EXT
- READ STREAM DMA
- WRITE DMA
- WRITE DMA EXT
- WRITE DMA FUA EXT
- WRITE STREAM DMA

### When the EDMA is in Queued mode:

The following commands are supported.

- READ DMA QUEUED
- READ DMA QUEUED EXT
- WRITE DMA QUEUED
- WRITE DMA QUEUED EXT
- WRITE DMA QUEUED FUA EXT

### When the EDMA is in Native Command Queuing mode:

The following commands are supported.

- Read FPDMA Queued
- Write FPDMA Queued



Other commands cause unpredictable results.

**Note**

#### 7.6.16.3

### EDMA Physical Region Descriptors (ePRD) Table Data Structure

The physical memory region to be transferred is described by the EDMA Physical Region Descriptor [ePRD] for DWORDs 0–3. The data transfer proceeds until all regions described by the ePRDs in the table have been transferred. The starting address of this table must be 16B aligned, i.e., bits [3:0] of the table base address must be 0x0.



The total number of bytes in the PRD table (total byte count in DMA command) must be 4-byte aligned.

**Note**

**Table 26: ePRD Table Data Structure Map**

Descriptor	Table, Page
ePRD DWORD 0	<a href="#">Table 27, p. 108</a>
ePRD DWORD 1	<a href="#">Table 28, p. 109</a>
ePRD DWORD 2	<a href="#">Table 29, p. 109</a>
ePRD DWORD 3	<a href="#">Table 30, p. 109</a>

**Table 27: ePRD DWORD 0**

Bits	Field	Description
0	Reserved	Reserved
31:1	PRDBA[31:1]	The byte address of a physical memory region corresponds to address bits [31:1].

**Table 28: ePRD DWORD 1**

Bits	Field	Description
15:0	ByteCount	Byte Count The count of the region in bytes. Bit 0 is force to 0. There is a 64-KB maximum. A value of 0 indicates 64 KB. The data in the buffer must not cross the boundary of the 32-bit address space, that is the 32-bit high address of all data in the buffer must be identical.
30:16	Reserved	Reserved
31	EOT	End Of Table The data transfer operation terminates when the last descriptor has been retired. 0 = Not end of table 1 = End of table <b>NOTE:</b> The total number of bytes in the PRD table (total byte count in DMA command) must be 4-byte aligned.

**Table 29: ePRD DWORD 2**

Bits	Field	Description
31:0	PRDBA[63:32]	The byte address of a physical memory region corresponds to bits [64:32]. Must be set to 0x0.

**Table 30: ePRD DWORD 3**

Bits	Field	Description
31:0	Reserved	Reserved

#### 7.6.16.4 Command Response Queue

The response queue is the interface that the EDMA uses to notify the CPU software that a data transaction between the system memory and the device was completed. The response queue is a 32 entry, circular queue (FIFO) whose location is configured by the EDMA Response Queue In-Pointer Register ([Table 340 p. 504](#)) and the EDMA Response Queue Out-Pointer Register ([Table 341 p. 505](#)).

The queue status is determined by comparing the two pointers:

- A queue is empty when the Response Queue Out-pointer reaches the Response Queue In-pointer.
- A queue is full when Response Queue In-pointer is written with same value as a Response Queue Out-pointer. A full queue contains 32 entries.
- A queue contains N entries when the Response Queue Out-pointer is N less than the Response Queue In-pointer, taking into account the wraparound condition.



The EDMA may write over existing entries when the queue is full.

**Note**



See [Figure 20, Command Response Queue—32 Entries, on page 93](#).

Each 8-byte command response entry consists of command ID, response flags, and a timestamp (see [Table 31, “EDMA CRPB Data Structure Map,” on page 110](#)). The CRPB data structure, described in [Table 17, “EDMA CRQB Data Structure Map,” on page 104](#), is written by the EDMA.

### 7.6.16.5 EDMA Command Response Block (CRPB) Data

[Table 31](#) provides a map of the EDMA command response block data structure tables.

**Table 31: EDMA CRPB Data Structure Map**

Register	Offset	Table, Page
CRPB ID Register	Offset: 0x00	<a href="#">Table 32, p. 110</a>
CRPB Response Flags Register	Offset: 0x02	<a href="#">Table 33, p. 110</a>
CRPB Time Stamp Register	Offset: 0x04	<a href="#">Table 34, p. 111</a>

**Table 32: CRPB ID Register**

Offset: 0x00

Bits	Field	Description
4:0	cHostQueTag	CRPB ID In queued DMA commands, these bits are used as a tag. This field contains the host identification of the command. These bits are copied from field <cHostQueTag> of <a href="#">Table 20, CRQB DW2—Control Flags, on page 105</a> .
15:5	Reserved	Reserved

**Table 33: CRPB Response Flags Register**

Offset: 0x02

Bits	Field	Description
6:0	cEdmaSts	CRPB EDMA Status This field contains a copy of the EDMA Interrupt Error Cause Register ( <a href="#">Table 334 p. 501</a> ) bits [6:0] accepted in the last command. <b>NOTE:</b> When the EDMA is in NCQ mode, ignore this field since the value of this field may reflect the status of other commands.
7	Reserved	Reserved This bit is always 0.
15:8	cDevSts	CRPB Device Status This field contains a copy of the device status register accepted in the last read of the register from the device.

**Table 34: CRPB Time Stamp Register**

0Offset: x04

Bits	Field	Description
31:0	Reserved	—

## 7.7 BIST

### 7.7.1 Far-End Loopback

This mode is performed according to *SATA 1.0 specification*, section 8.5.7. *BIST activate FIS*.

The supported BIST patterns are:

- L: Far-end Retimed Loopback
- TS: Transmit Only and Scrambling Bypass
- TSA: Transmit Only, Scrambling Bypass, and Align Bypass

### 7.7.2 BIST as the Initiator Side

The following flow should be performed by the host CPU:

- Send BIST Activate FIS using vendor unique interface to initiate BIST mode over the SATA link (see [Section 7.8, Vendor Unique, on page 112](#)).
- Set the [`<BISTMode>`](#) field in the BIST Control Register ([Table 375 p. 534](#)) to 1 to determine FIS direction.
- Initiate [`<BISTPattern>`](#) according to the transmitted BIST Activate FIS.
- Initiate the [`<BistDw1>`](#) field in the BIST-Dword1 Register ([Table 376 p. 534](#)) according to the transmitted BIST Activate FIS.
- Initiate the [`<BistDw2>`](#) field in the BIST-Dword2 Register ([Table 377 p. 535](#)) according to the transmitted BIST Activate FIS.
- Set the [`<BISTEn>`](#) field in the BIST Control Register ([Table 375 p. 534](#)) to activate the pattern comparator operation.
- Read [`<BISTResult>`](#) status to determine BIST test passes or not.
- Set the [`<eAtaRst>`](#) field in the EDMA Command Register ([Table 342 p. 506](#)) to exit both sides of the link from BIST mode.

### 7.7.3 BIST as the Target Side

The following flow should be performed, when the Serial-ATA port receives BIST Activate FIS:

1. Host CPU must set bit [1] in the [`<FISWait4Rdy>`](#) field in the FIS Interrupt Cause Register ([Table 384 p. 541](#)) to 1.
2. The FIS content is updated in the FIS Dword0 Register ([Table 386 p. 543](#)) through FIS Dword6 Register ([Table 392 p. 544](#)).
3. Bit [3] in field [`<FISWait4HostRdy>`](#) field in the FIS Interrupt Cause Register ([Table 384 p. 542](#)) is set.
4. The [`<eTransInt>`](#) field in the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)) is also set if the corresponding bit in the FIS Interrupt Mask Register ([Table 385 p. 543](#)) is set to 1.
5. Host CPU must clear bit [1] in the [`<FISWait4Rdy>`](#) field in the FIS Interrupt Cause Register ([Table 384 p. 541](#)).
6. Host CPU must clear the [`<eTransInt>`](#) field in the EDMA Interrupt Error Cause Register ([Table 334 p. 501](#)).



7. Host CPU sets the <BISTMode> field in the BIST Control Register ([Table 375 p. 534](#)) to 0 to determine FIS direction.
8. Host CPU initiates <BISTPattern> according to the received BIST Activate FIS.
9. Host CPU initiates the <BistDw1> field in the BIST-Dword1 Register ([Table 376 p. 534](#)) with the contents matching the received BIST Activate FIS.
10. Host CPU initiates the <BistDw2> field in the BIST-Dword2 Register ([Table 377 p. 535](#)) with the contents matching the received BIST Activate FIS.
11. Host CPU sets the <BISTEn> field in the BIST Control Register ([Table 375 p. 534](#)) to activate the internal pattern generators to send the data stream onto the Serial-ATA link.

## 7.8 Vendor Unique

### 7.8.1 Vendor Unique Frames

The following flow should be performed to activate transmission of Vendor Unique FIS.

1. Wait until all pending commands in the EDMA are completed.
2. Disable the EDMA, set the <eDsEDMA> field in the EDMA Command Register ([Table 342 p. 506](#)).
3. Verify the EDMA is disabled, the <eEnEDMA> field is cleared.
4. Verify the Transport Layer is in idle, the <TransFsmSts> field in the Serial-ATA Interface Status Register ([Table 381 p. 539](#)) is cleared.
5. Set Vendor Unique Mode. Write 1 to the <VendorUqDn>.
6. Insert data into the Vendor Unique Register ([Table 382 p. 540](#)).
7. Repeat steps 6 until all data except the last DWORD in the vendor unique FIS is transferred. Note that according to the Serial-ATA protocol the FIS length is limited to 8 KB.
8. Write 1 to bit <VendorUqSend> field in the Serial-ATA Interface Control Register ([Table 379 p. 536](#)).
9. Write last DWORD in the FIS to Complete FIS transmission.
10. Wait for transmission completion. The <VendorUqDn> field or the <VendorUqErr> field in the Serial-ATA Interface Status Register ([Table 381 p. 538](#)) is set to 1.
11. Verify successful transmission of the FIS. Bit <VendorUqErr> is cleared.
12. Clear Vendor Unique Mode. Write 0 to the <VendorUqMd> field in the Serial-ATA Interface Control Register ([Table 379 p. 535](#)).

## 7.9 Protocol Based Port Select

The EDMA supports the Port Selector (PS) protocol based ingredient—When the host CPU sets the <PortSelector> field in the PHY Mode 4 Register ([Table 372 p. 530](#)), the Serial-ATA II PHY issues the protocol based OOB sequence to select the active host port.

# 8

## Gigabit Ethernet Controller

The 88F6180 implements a single Gigabit Ethernet (GbE) controller that operates in RGMII/MII/MMII mode.

The 88F6190 implements two Ethernet controllers. One is an interface to a Gbe port. The other is an interface to a Fast Ethernet port. These ports support RGMII/GMII/MII/MMII modes, in the configurations indicated below. The configuration is according to [<RGMIIEn>](#) field in the Port Serial Control1 (PSC1) Register ([Table 430 p. 571](#)).

### Set the <RGMIIEn> Field

Configuration for the 88F6190	Port0	Port1
■ Port0 RGMII, Port1 MII/MMII	0x1	0x0
■ Port0 GMII, Port1 N/A	0x0	0x0

The 88F6192 and 88F6281 implement two GbE controllers that support the following modes. The configuration is according to [<RGMIIEn>](#) field in the Port Serial Control1 (PSC1) Register ([Table 430 p. 571](#)) of port 0 and port 1:

### Set the <RGMIIEn> Field

Configuration for the 88F6192, 88F6281	Port0	Port1
■ Port0 RGMII, Port1 RGMII	0x1	0x1
■ Port0 RGMII, Port1 MII/MMII	0x1	0x0
■ Port0 MII/MMII, Port1 RGMII	0x0	0x1
■ Port0 GMII, Port1 N/A	0x0	0x0



GMII can operate at 1 Gb only, and cannot move to 10/100 Mbps and change to MII.

#### Note

### 8.1

## Port Features

The 10/100/1000 Mbps GbE port provides the following features:

- IEEE 802.3 compliant MAC layer function
- IEEE 802.3 compliant MII interface
- 1000 Mbps operation—full duplex
- 10/100 Mbps operation—half and full duplex
- GMII symmetric Flow Control: IEEE 802.3 flow-control for full-duplex operation mode
- MII symmetric Flow Control: Backpressure for half-duplex operation mode
- RGMII mode
- Transmit functions:
  - Short frame (less than 64 bytes) zero padding
  - Long frames transmission (limited only by external memory size)
  - Checksum on transmit frames for standard Ethernet packet size

- Programmable values for Inter Packet Gap
- CRC generation
- Backoff algorithm execution
- Error reporting
- Receive functions:
  - Address filtering modes:
    - 16 Unicast
    - 256 IP Multicast
    - 256 Multicast
    - Unicast promiscuous mode (reception of Unicast frames, even those not matched in the DA filter)
    - Broadcast
    - Broadcast reject mode
  - Automatic discard of error frames, smaller than the programmable minimum frame size
  - Reception of long frames (Programmable legal frame size is up to 9700 bytes)  
**NOTE:** Frames larger than the limit are actually received, however, they are marked in the descriptor as Oversize errors.
  - CRC checking
  - Error report
- Precise Timing Protocol (PTP) with:
  - Precise time stamping for packets, as defined in IEEE 1588 PTP v1 and v2 and IEEE 802.1AS draft standards
  - Flexible Time Application interface to distribute PTP clock and time to other devices in the system
  - Optionally accepts an external clock input for time stamping
- Audio Video Bridging (AVB) Networks including:
  - IEEE 802.1Qav pre-draft Audio Video Bridging networks
  - Time- and priority-aware egress pacing algorithm to prevent bunching and bursting effects—suitable for audio/video applications
  - Egress Jitter Pacer for AVB-Class A and AVB-Class B traffic and strict priority for legacy traffic queues

## 8.2

## Functional Overview

Each port includes an IEEE 802.3 compliant 10/100/1000 Mbps MAC. The port speed, duplex and IEEE 802.3 Flow Control can be auto-negotiated, according to IEEE standard 802.3. Backpressure is supported for half-duplex mode when operating at 10/100 Mbps speeds. Each port supports MIB counters.

Each receive port includes a Layer-2 Destination Address (MAC-DA) recognition and filtering mechanism of up to 16-Unicast MAC addresses, 256 IP Multicast addresses, and 256 Multicast/Broadcast address. The receive ports may also detect Layer-2 frame-type encapsulation, as well as common Layer 3 and Layer 4 protocols.

Layer-2 CRC, IP checksum, TCP checksum, and UDP checksum are always checked on received packets, and may be generated for transmit. This capability increases performance significantly by off-loading these operations from the software. Checksum is checked for Jumbo frames on received packets. Checksum generation for transmitted Jumbo frames is not supported.

Each port includes eight dedicated receive DMA queues and eight dedicated transmit DMA queues, plus two dedicated DMA engines (one for receive and one for transmit) that operate concurrently.

Each queue is managed by a chain of buffers-descriptors that are managed by the software. A transmit buffer of any byte alignment and any size, above 8 bytes, is supported. The receive buffers must be 64-bit aligned.

Queue classification on received traffic is assigned to the DMA queue based upon a configurable analysis, which evaluates the DA-MAC, IP, TOS (Type of Service), IEEE 802.1p priority tag, and protocol (ARP, TCP, or UDP). An example for use of this feature is the implementation of differentiated services or real-time, jitter-sensitive voice/video traffic intermixed with data traffic. As each queue has its own buffering, blocking is avoided and latency is reduced for CPU service.

Detailed status is given for each receive frame in the packet descriptors, while statistics are accumulated for received and transmitted traffic in the MIB counters, on a per port basis.

The 10/100/1000 Mbps GbE unit handles all functionality associated with moving packet data between local memory and the Ethernet ports.

The port's speed (10, 100, or 1000 Mbps) is auto-negotiated through the PHY and does not require user intervention. Auto-Negotiation is according to IEEE standard 802.3. The 1000 Mbps speed operates only in full-duplex mode. The 100 Mbps and 10 Mbps speeds operate either in half- or full-duplex mode, with the selection of the duplex mode auto-negotiated through the PHY without user intervention. With the RGMII/GMII interface, the port only supports symmetric Flow Control.



**Note**

Auto-Negotiation also can be disabled. When Auto-Negotiation is disabled, the link parameters (link speed and Duplex mode) are forced by the software. The link must be forced down when changing the port speed.

---

Frame type/encapsulation detection is available on:

- Layer 2 for:
  - Bridge Protocol Data Unit (BPDU)
  - VLAN (programmable VLAN-ethertype)
  - Ethernet v2, LLC/SNAP
- Layer 3 for:
  - IPv4 (according to Ethertype)
- Layer 4 (only over IPv4) for:
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)

Frame enqueueing is in accordance with DA, VLAN-802.1p, IP-TOS using the *highest* priority counts. Frame enqueueing is captured according to protocol type for TCP, UDP, ARP, or BPDU. The port also supports enqueueing based on the proprietary Marvell® Header or the DSA Tag. These are useful when interfacing Marvell FE or GE switches that utilize these features, resulting in improved routing performance, and support for cascade and stack of switches.

Received frames smaller than the programmable minimum frame size are automatically discarded. Reception and transmission of long frames, up to 9700 bytes, are supported. The frame type, encapsulation method, errors, and checksums are reported in the buffer descriptor. Automatic IP header 32-bit alignment is done in memory by adding 2 bytes at the beginning of each frame. The TCP and UDP checksum calculations are put into the receive descriptor (and are compared with the frame checksum for non-IP fragmented frames), even for frames over 9 KB.



**Note**

The GbE port is IPv6 ready. Even though it does not recognize IPv6 encapsulation, IPv6 packets will be accepted (as any other un-recognized L3 type) and enqueued based on L2 address and VLAN tag.

Each Ethernet port provides a great amount of flexibility with many programmable features. The TCP, UDP, and IP checksums are generated for transmitted frames. This is programmable on a per frame basis in the first descriptor. There are separate, programmable transmit and receive interrupt coalescing mechanisms to aggregate several interrupts (on a time-based masking window) before sending an indication to the CPU. The port provides programmable zero padding of short frames (frames less than 64 bytes).

Byte based band-width distribution among transmit queues by a weighted-round-robin arbitration mechanism is programmable. This includes programming of hybrid fixed and round-robin priorities. The maximum byte based band-width allocation per transmit queue is also programmable. An Egress Jitter Pacing (EJP) arbitration mechanism is supported, to comply with IEEE 802.1Qav pre-draft. A transmit buffer of any byte alignment and any buffer size (greater than 8 bytes) is supported; the minimum packet size is 32 bytes.

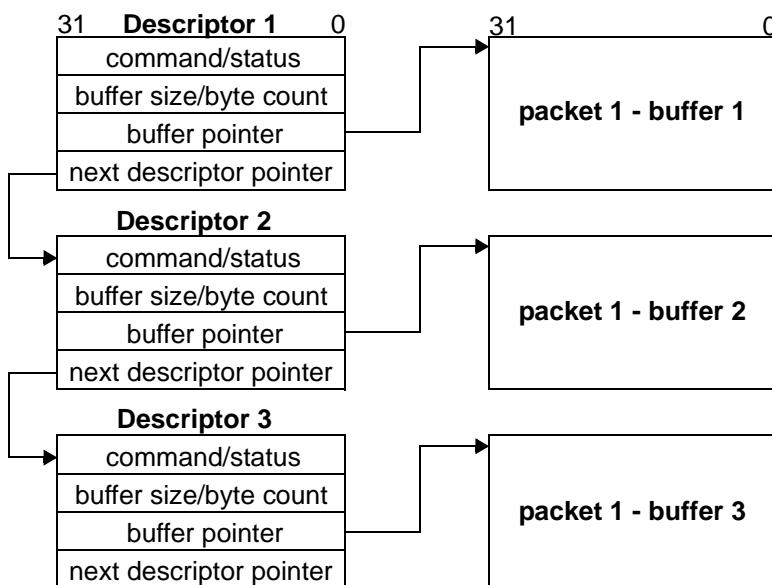
## 8.3 DMA Functionality

The GbE unit provides Ethernet ports functionality, with the port(s) capable of running at either 10 or 100 Mbps (half- or full duplex), or 1000Mbps (full duplex only). Each port manages packet data transfer between memory and PHY. The data is stored in memory buffers, with any single packet spanning multiple buffers if necessary. Upon completion of packet transmission or reception, a status report, which includes error indications, is (optionally) written by the Ethernet unit to the first descriptor (for receive ports) or to the last descriptor (for transmit ports) associated with this packet.

The buffers are allocated by the CPU and are managed through chained descriptor lists. Each descriptor points to a single memory buffer and contains all the relevant information relating to that buffer (that is buffer size, buffer pointer, etc.) and a pointer to the next descriptor. Whenever a new buffer is needed (end of buffer or end of packet), a new descriptor is automatically fetched, and the data movement operation is continued using the new buffer. These descriptors appear in [Table 35 on page 123](#) through [Table 42 on page 134](#).

Figure 22 shows an example of memory arrangement for a single packet using three buffers.

**Figure 22: Ethernet Descriptors and Buffers**



The following sections provide detailed information about the operation and user interface of the Ethernet unit and its logic subsections.

Tx and Rx buffers are managed via link list of descriptors placed in DRAM. Buffers and descriptors are being read/write from/to memory by the port Rx and Tx DMAs.

### 8.3.1

## Address Decoding

Each GbE MAC has six address windows. Each address window can be individually configured.

With the port DMA transaction (buffer read/write, descriptor read/write), the address is compared against the address decoding registers. Address comparison is done to select the correct target interface and attributes.

Four of the address windows have an upper 32-bit address register. These are used for accessing interfaces that support more than 4-GB address space. The address generated on the interface is composed of the 32-bit address issued by the SDMA (if it hits the relevant address window) concatenated with the High Address Remap register content.

For the port DMA to avoid accessing a forbidden address space (due to a programming bug), each port uses access protection logic that prevents it from read/write accesses to specific address windows.

If the address does not match any of the address windows, or if it violates the access protection settings, an interrupt is generated. The transaction is executed but not to the original address. Instead, the transaction is executed to a default address and target as specified in the Default Address and ID registers.

### 8.3.2

## Endianess and Swap Modes

Each DMA channel has configurable behavior on Little or Big Endian support, per DMA channel data receive and data transmission (see the [BLMR](#) field and the [BLMT](#) field in the SDMA Configuration (SDC) Register ([Table 422 p. 564](#)) field).

For every DMA channel, descriptor accesses may be swapped (see the [SwapMode](#) field in the SDMA Configuration (SDC) Register ([Table 422 p. 564](#))).

### 8.3.3

## Transmit DMA Descriptors

#### 8.3.3.1

### Transmit Operation

To initialize a transmit operation, the CPU must perform the following:

1. Prepare a chained list of descriptors and packet's data buffers.  
**NOTE:** The TxDMA supports several priority transmit queues with programmable fixed or weighted priority with optional bandwidth limiting on the port or queue (see [Section 8.8.1, Priority Modes, on page 145](#)). If the user wants to take advantage of this capability, a separate list of descriptors and buffers must be prepared for each of the priority queues.
2. Write the pointer of the first descriptor to the DMA's Transmit Current Queue Descriptor Pointer (TCQDP) Register (n=0–7) ([Table 446 p. 586](#)) associated with the priority queue to be started. If more than one of the priority queues are needed, initialize the [Transmit Current Queue Descriptor Pointer \(TCQDP\) Register \(n=0–7\)](#) for each queue.
3. Initialize and enable the Ethernet port by writing to the port's configuration and command registers.
4. Initialize and enable the DMA by writing to the DMA's configuration and command registers (Triggering the DMA is accomplished by setting the [ENQ](#) bit in the Tx Command register).

After completing these steps, the DMA starts and performs arbitration between the transmit queues according to the configuration, on a packet by packet basis, as explained in [Section 8.8.1, Priority](#)



**Modes.** The DMA fetches the first descriptor from the specific queue it decided to serve, and starts transferring data from the memory buffer to the Tx-FIFO.

When the entire packet is in the FIFO, it may potentially calculate and update the IP checksum, TCP, or UDP checksum. The port then initiates transmission of the packet across the external PHY interface. The port also calculates Layer-2 CRC and append to the packet tail. While data is read from the FIFO, new data is written into the FIFO by the DMA.

For packets that span more than one buffer in memory, the DMA will fetch new descriptors and buffers as necessary.

When transmission is completed, status is (optionally) written to the first long word of the last descriptor. The Next Descriptor's address, which belongs to the next packet in the queue, is written to the current descriptor pointer register.

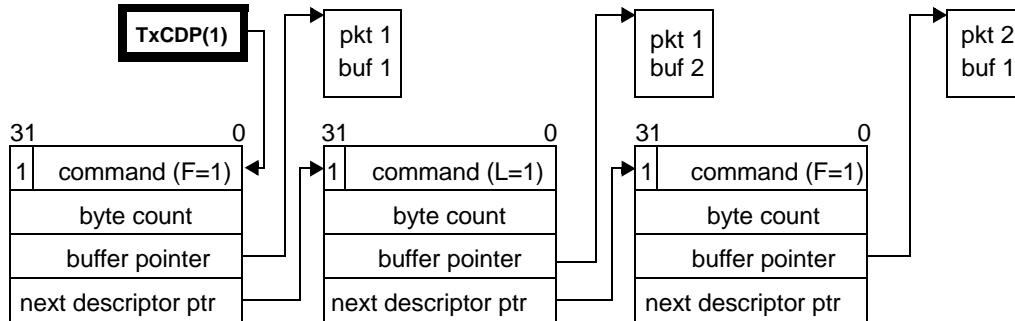
This process (starting with DMA arbitration) is repeated as long as there are packets pending in the transmit queues. When no packets are pending in a transmit queue, the DMA resets the <ENQ> bit in the Tx command register (one bit per queue) and reports the queue end via a TxEnd maskable interrupt in the ICE register (for each queue).

[Figure 23](#) shows how the transmit descriptors are managed when a two buffers packet is transmitted.

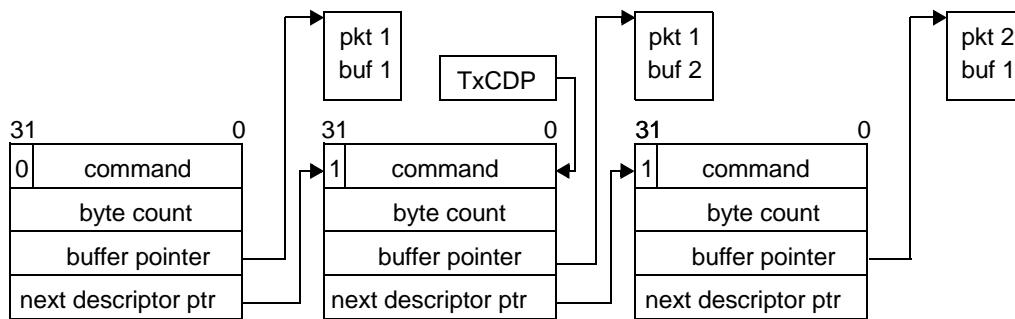
**Figure 23: Ethernet Packet Transmission Example**

---

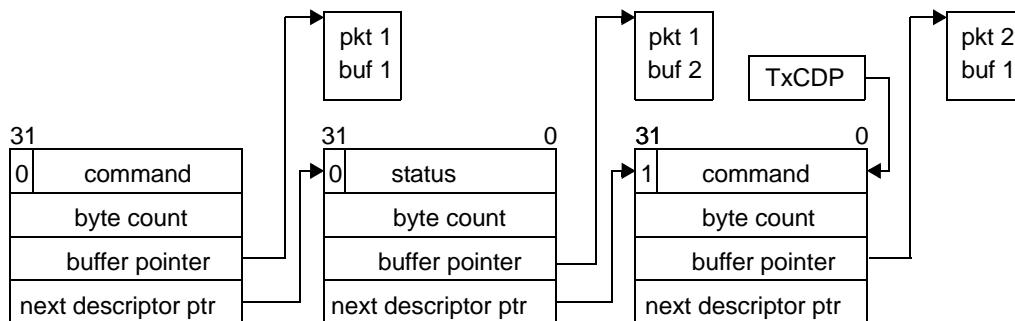
1. Packet 1 - Transmitting 1st buffer



2. Packet 1 - transmitting 2nd buffer



3. Packet 2 - transmitting 1st buffer



1. TxCMP = Transmit Current Descriptor Pointer.  
Key: pkt = packet, buf = buffer, ptr = pointer.

Ownership of any descriptor other than the last is returned to the CPU upon completion of data transfer from the buffer pointed by that descriptor. The last descriptor, however, is returned to CPU ownership only after the actual transmission of the packet is completed. While changing the ownership bit of the Last descriptor, the DMA also writes status information that indicates any errors that might have happened during transmission of this packet. There are two relevant modes:

- AM (Auto Mode): When this mode is set, the DMA will not close descriptors that are not last descriptors (since the only change in non-last descriptors is their ownership).
- <AMNoTxES> field in the Port Configuration (PxC) Register ([Table 416 p. 561](#)): When this mode is set, the last descriptors are also not closed.

The transmit buffer supports any byte alignment at any size (> 8 bytes) with a minimum packet size of 32 bytes.

If generation of IP or Layer4 checksum is required, then the maximum packet size is 1.6 KB.

### 8.3.3.2 Retransmission (Collision)

Collision support is integrated into the Ethernet port for half-duplex operation mode. Half-duplex mode is supported in 10 and 100Mbps speeds only.

A collision event is indicated each time receive and transmit are active simultaneously. When that happens, active transmission is stopped, the jam pattern is transmitted and the collision count for the packet increments. The packet is retransmitted after a waiting period that conforms to the binary backoff algorithm specified in the IEEE 802.3 standard. The retransmit process continues for multiple collision events as long as a limit is not reached. This retransmit limit, which sets the maximum number of transmit retries for a single packet, is defined by the IEEE 802.3 standard as 16.

As long as a packet is being retransmitted, its last descriptor is kept under port ownership. When a successful transmission takes place (i.e. no collision), a status word containing collision information is written to the last descriptor and ownership is returned to the CPU.

If a retransmit limit is reached with no successful transmission, a status word with error indication is written to the packet's last descriptor, and the transmit process continues with the next packet.

It is important to note that collision is considered legal only if it happens before transmitting the 65<sup>th</sup> byte of a packet. Any collision event that happens after sending the first 64 byte window is known as a *late collision*, and is considered a fatal network error. Late collision is reported to the CPU through the packet status, and no retransmission is done.



**Note** Any collision occurring during the transmission of the transmit packet's last four bytes is not detected.

### 8.3.3.3 Zero Padding of Short Frames

The Ethernet port offers a per frame padding enable bit in the transmit descriptor. This causes the port logic to enlarge frames shorter than 64 bytes by appending zero bytes. When this feature is used, only frames equal or larger than 64 bytes are transmitted as is. Frames smaller than 64 bytes are zero padded and transmitted as 64-byte packets.



**Note** When using the Marvell Header mode, DSA tag, or Extended DSA Tag, the port performs zero padding to packet sizes of 66, 68, or 72 bytes, respectively. This guarantees that the packet is still legal after the destination MAC strips the extra bytes of Marvell Header or DSA tag.

#### **8.3.3.4**

#### **CRC Generation**

Ethernet CRC denotes four bytes of Frame-Check-Sequence appended to each packet.

The port can generate and append CRC to a transmitted packet. CRC generation can be disabled via the [\*\*<TxCRCDis>\*\*](#) field in the Port Configuration Extend (PxCX) Register ([Table 417 p. 562](#)). If disabled, it is the software's responsibility to place the 32-bit CRC at the end of the packet.

If CRC generation is enabled, and the data fetched from memory is known to be erroneous, The packet is sent, and CRC is not generated (causing bad CRC detection at the receiving side).

#### **8.3.3.5**

#### **IP Checksum Generation**

IPv4 checksum may be calculated during the packet DMA from memory, and it is replaced in the checksum field for IPv4 packets encapsulated in Ethernet-v2 format, with or without a VLAN tag (this must be specified in the descriptor). IPv4 checksum is similarly supported for LLC/SNAP packets, including Jumbo frames (the CPU must set the LLC/SNAP-bit in the descriptor for such packets).

One bit in the transmit descriptor is used for specifying if the IPv4 checksum generation is required for a specific packet.

#### **8.3.3.6**

#### **TCP Checksum Generation**

The TCP checksum may be enabled per frame. When TCP checksum is enabled, it is calculated during the packet read from memory by the DMA and the calculated value is written in the checksum field before transmission begins.

This is supported for TCP over IPv4 over Ethernet-v2, with or without VLAN tag (this must be specified in the descriptor). It is similarly supported for LLC/SNAP packets, (not including Jumbo frames). The LLC/SNAP-bit must be set by the CPU, in the descriptor for such packets.

Since a TCP segment may be transmitted over several Ethernet packets, and since the checksum in the next packets continue the checksum calculation of previous packets, there are two types of checksum generation commands (depending on bit 10 in the Tx descriptor):

- Calculate the checksum on the *first* packet in the segment:  
In this case, the 16-bit checksum field in the descriptor must be zero. The checksum is done fully by the device, and will include parsing the header according to the descriptor fields and calculating the checksum on pseudo-header. The checksum continues with full checksum calculation on the TCP data, and finally it is placed in the packet before transmission.
- Calculating checksum on *non-first* packets in the segment:  
The CPU is required to calculate the initial checksum, including the pseudo-header checksum in the [\*\*<L4iChk>\*\*](#) field in the Tx descriptor. The DMA uses this initial checksum value in calculating the TCP checksum over the TCP payload in the packet and place it in the TCP checksum field of the packet before transmission.

The CPU may choose to always calculate the checksum over the pseudo-header, and let the hardware take care of the payload checksum.

#### **8.3.3.7**

#### **UDP Checksum Generation**

The UDP checksum generation is the same as the TCP checksum generation support, with both first and non-first modes (see above).

#### **8.3.3.8**

#### **VLAN Bit**

The CPU is required to specify whether the packet is VLAN tagged or not. This is needed to facilitate the packet parsing during fetching from memory. It is used to correctly locate the IP header when the IP checksum, TCP checksum, or UDP checksum generation is required.

### 8.3.3.9 LLC/SNAP Bit

The Layer 3 and Layer 4 checksum generation is supported for Ethernet-v2 frames, or for LLC/SNAP frames. This bit must be set in case the checksum generation features is required for LLC/SNP frames.

### 8.3.3.10 Transmit Descriptor Structure

- Descriptor length is four long words (4LW), and it must be 4LW aligned (that is, Descriptor\_Address[3:0]==0000).
- Descriptors may reside anywhere in the device address space except for a null address (0x00000000), which is used to indicate the end of the descriptor chain. Descriptor may not be placed on a NAN Flash device. Descriptors are fetched always in burst of 4LW.
- The last descriptor in the linked chain must have a null value in the Transmit Descriptor - Next Descriptor's <NextDescriptorPointer> bits[31:0] ([Table 38 on page 125](#)). Alternatively, the last descriptor may be not owned. Having a not owned descriptor is useful for performance optimization, by using a dummy pointer for adding descriptors to a chain without reprogramming the First Descriptor Pointer (FDP) register (see [Section 10.5.4.2, Chain Mode, on page 202](#) and [Section 10.5.4.5, Descriptor Ownership, on page 204](#)).
- For packets that span multiple descriptors, the CPU must provide ownership on all the packet's descriptors before giving ownership on the first descriptor of the packet, to avoid underrun situations.
- TX buffers associated with TX descriptors are limited to 64-1 KB and can reside anywhere in memory. However, buffers with a payload of one to eight bytes must be aligned to a 64-bit boundary. Zero size buffers are illegal.

**Figure 24: Transmit Descriptor Description**

3	3	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	Offset																							
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0																										
<b>Byte 3</b>							<b>Byte2</b>							<b>Byte1</b>							<b>Byte0</b>																																				
Command / Status																													+0																												
Byte Count[15:0]															<L4iChk>/Reserved														+4																												
Buffer Pointer [31:0]																													+8																												
Next Descriptor Pointer [31:4]																													+C																												

### 8.3.3.11 Tx Descriptor Command/Status

**Table 35: Transmit Descriptor Command/Status**

Bits	Field	Description
0	ES	<p>Error Summary of MAC level errors on frame transmission.            0 = No Error            1 = Error occurred (Late Collision - LC, or Retransmit Limit - RL, or Underrun Error - UR)</p> <p><b>NOTE:</b> This field is only valid only if &lt;L&gt; bit[20] is set.</p> <p>If &lt;AM&gt; bit[30] is set and the <a href="#">&lt;AMNoTxES&gt;</a> field in the Port Configuration (PxC) Register (<a href="#">Table 416 p. 561</a>) is set, this field, as well as &lt;EC&gt; bits[2:1], are not updated.</p>
2:1	EC	<p>Error Coding            00 = LC            01 = UR            10 = RL reached (excessive collision)            11 = Reserved</p> <p><b>NOTE:</b> Valid only if &lt;L&gt; bit[20] is set and &lt;ES&gt; bit[0] is set.</p>
8:3	Reserved	Reserved
9	LLC/SNAP	<p>When set, this bit signifies that the packet has an LLC/SNAP format.            0 = Not LLC/SNAP            1 = LLC/SNAP</p> <p><b>NOTE:</b> Valid only if F is set, and if GL4chk or GIPchk is set.</p> <p>IP and TCP/UDP checksum is supported for LLC/SNAP frames or for Ethernetv2 frames</p>
10	L4Chk_Mode	<p>Provides the TCP/UDP frame type for checksum calculation mode when GL4chk=1.            0 = Frame is IP fragmented. (The CPU must provide the initial checksum value calculated over the pseudo-header in the &lt;L4iChk&gt; field.)            1 = Frame is <i>not</i> IP fragmented. (The CPU must provide zero value in the &lt;L4iChk&gt; field.)</p> <p><b>NOTE:</b> The payload length over which the checksum is calculated is determined by the Layer4 LENGTH field in the packet, and therefore it should NOT include any pad bytes.</p> <p>Valid only if F is set, and if GL4chk=1.</p>
14:11	IPv4HdLen	<p>Provides the length in long words (4 bytes) of the IPv4 header.</p> <p><b>NOTE:</b> This is only valid if F bit [21] is set, and either GL4chk bit [17] or GIPchk bit [18] is set.</p>
15	VLAN	<p>VLAN signifies if the Ethernet-v2 frame is VLAN tagged or not.            Only if &lt;GIPchk&gt; bit[18] or &lt;GL4chk&gt; bit[17] are set, this field must have a correct value.            0 = Frame is <i>not</i> VLAN tagged.            1 = Frame is VLAN tagged.</p> <p><b>NOTE:</b> This is only valid if &lt;F&gt; bit[21] is set.</p>
16	L4type	<p>When GL4chk is set, signifies which Layer4 protocol is carried in the frame.            0 = TCP            1 = UDP</p> <p><b>NOTE:</b> This is only valid if the &lt;F&gt; bit[21] is set.</p>

**Table 35: Transmit Descriptor Command/Status (Continued)**

Bits	Field	Description
17	GL4chk	<p>Generate TCP/UDP Checksum 0 = No operation 1 = Generate TCP/UDP checksum. <b>NOTE:</b> This may only be set to TCP or to UDP over IPv4 over Ethernetsv2 frames (tagged or untagged).</p> <p>The CPU must provide the initial checksum value calculated over the pseudo-header in the Transmit Descriptor Register &lt;L4iChk&gt; bits[15:0].</p> <p>The payload length over which the checksum is calculated is determined by the Layer4 Length field in the packet, and therefore, it should NOT include any pad bytes.</p> <p>This is only valid if &lt;F&gt; bit[21] is set.</p>
18	GIPchk	<p>Generate IPv4 checksum. This is supported for Ethernetsv2 and LLC/SNAP frames (tagged or untagged), with a valid IPv4 Header (<math>\text{IPHL} \geq 5</math>, <math>\text{IPHL} * 4 \leq \text{IPTL}</math>). <b>NOTE:</b> This is only valid if the &lt;F&gt; bit[21] is set.</p>
19	P	<p>Padding When this bit is set and the packet is smaller than 60 bytes, zero-value bytes are appended to the packet. Use this feature to prevent transmission of fragments. <b>NOTE:</b> This is only valid if &lt;L&gt; bit[20] is set.</p> <p>If disabling HW CRC generation (CRC is generated by SW), zero padding must not be enabled.</p>
20	L	<p>Last Indicates the last buffer of frame.</p>
21	F	<p>First Indicates the first buffer of a frame.</p>
22	Reserved	
23	EI	<p>Enable Interrupt When set, a maskable interrupt will be generated upon the closing descriptor. <b>NOTE:</b> To limit the number of interrupts and prevent an interrupt per buffer situation, set this bit only in descriptors associated with Last buffers. This way the TxBuffer interrupt is only set when transmission of a frame is completed.</p> <p>Interrupts may be further delayed by the Interrupt coalescing mechanism (see <a href="#">Section 8.7.1, Interrupt Coalescing, on page 144</a>).</p>
29:24	Reserved	Reserved.
30	AM	<p>Auto Mode When set, the DMA will not clear the &lt;Ownership&gt; bit at the end of the buffer process. If the &lt;AMNoTxES&gt; field in the Port Configuration (PxC) Register (<a href="#">Table 416 p. 561</a>) is set, no status is reported in the last descriptor (See &lt;ES&gt; bit[0] and &lt;EC&gt; [2:1], fields above).</p>
31	O	<p>Ownership Bit 0 = Buffer owned by the CPU. 1 = Buffer owned by the DMA.</p>

**Table 36: Transmit Descriptor Byte Count**

Bits	Name	Description
15:0	L4iChk	The CPU provides the initial checksum value calculated on the pseudo-header when: The Transmit Descriptor's <GL4chk> bit[17] is set The Transmit Descriptor's <L4Chk_Mode> bit[10] is cleared. Otherwise these bits are reserved. <b>NOTE:</b> Only valid if the <F> bit[21] is set.
31:16	Byte Count	Number of bytes to be transmitted from the associated buffer. This is the payload size in bytes.

**Table 37: Transmit Descriptor Buffer Pointer**

Bits	Name	Description
31:0	Buffer Pointer	A 32-bit pointer to the beginning of the buffer associated with this descriptor. <b>NOTE:</b> There is a 64-bit alignment requirement for buffers that have a setting in the Transmit Descriptor Register Byte Count bits[31:16] of 1–8 bytes.

**Table 38: Transmit Descriptor Next Descriptor Pointer**

Bits	Name	Description
31:0	NextDescriptor Pointer	A 32-bit pointer that points to the beginning of the next descriptor. <b>NOTE:</b> bits[3:0] must be set to 0. A DMA operation is stopped when a null (all zeros) value is encountered in this field.

### 8.3.3.12 Transmit DMA Pointer Registers

The Tx DMA employs a single 32-bit pointer register per queue, the TX DMA Current Descriptor Pointer (TxCDP) register.

The TxCDP register is a 32-bit register used to point to the current descriptor of a transmit packet. The CPU must initialize this register before enabling DMA operation. The value used for initialization should be the address of the first descriptor to use.

### 8.3.3.13 Transmit DMA Notes

The transmit DMA process is packet oriented. The transmit DMA does not close the last descriptor of a packet, until the packet has been fully transmitted. When closing the last descriptor, the DMA writes packet transmission status to the Command/Status word and resets the ownership bit. A TxBuffer maskable interrupt is generated in the ICRE register for each queue, if the <EI> bit in the last descriptor is set.

Updating the status in the descriptor is programmable per the <AM> bit in the Tx descriptor. When set, the DMA will not clear the Ownership bit at the end of buffer process. If, in addition AMNoTxES bit is set in the Port Configuration register, no status will be reported in last descriptor. The advantage of this is that it reduces memory write access per descriptor. This versus the trade-off of not getting error indications per packet, like late collisions, and not relying on the ownership bit for each descriptor.

Transmit DMA stops processing a Tx queue whenever a descriptor with a null value in the Next Descriptor Pointer field is reached or when a CPU owned descriptor is fetched. When that happens, a TxEnd maskable interrupt is generated in the ICRE register (per queue) and the <ENQ> bit is reset. To restart the queue, the CPU should issue a Enable-queue command by writing 1 to the ENQ bit in the Tx command register.<sup>1</sup>

The transmit DMA does not expect a null Next Descriptor Pointer or a CPU owned descriptor in the middle of a packet. Additionally, the transmit DMA does not expect a data integrity error on descriptors. When any of these events occurs, the DMA aborts transmission and stops queue processing (that is, it resets the <ENQ> bit). A TxError maskable interrupt is generated. To restart the queue, the CPU should issue an Enable\_Queue command.

A transmit underrun occurs when the DMA cannot access the memory fast enough and packet data is not transferred to the FIFO before the FIFO becomes empty. In this case, the DMA aborts transmission and closes the last descriptor with a UR bit set in the status word. Additionally, a Tx\_Underrun maskable interrupt is generated. The transmit process continues with the next packet. In the device Tx DMA, transmitting packets less than 1600 bytes long, such an error *cannot* happen, as the packet is fully buffered in the FIFO before transmission begins.

To stop DMA operation before the DMA reaches the end of descriptor chain, the CPU should issue a Disable-Queue command by writing 1 to the <DISQ> bit in the DMA command register. The DMA stops processing the queue as soon as the current packet transmission is completed, and its last descriptor is returned to CPU ownership. Then, the DMA resets the ENQ bit. To ensure that the DMA is actually disabled, the CPU should poll the <DISQ> bit to confirm that it is set to 1, before trying to modify any of the DMA parameters relevant to this queue.

In addition, a TxEnd maskable interrupt is generated. To restart this queue, the CPU must issue a Enable-Queue command.

When the Ethernet link was lost during normal operation, the DMA will disable all the queues by resetting the <ENQ> bits. Since losing a link can happen anytime during DMA programming by the CPU (for example, a disconnected cable or a far end disconnect) the following precaution must be taken: If the CPU gets a link-down interrupt, then it must wait for the DMA to reset the <ENQ> bits of the DMA channels for Tx, after the link down event, before re-enabling the DMA channels.

The CPU must never modify the DMA configuration register or the TxCDP register while the DMA ENQ bit is set. Modifying the TxCDP registers is allowed only when the respective DMA ENQ bit is reset. Modifying the DMA configuration registers may be done only when *all* the DMA channels ENQ bits are reset.

The DMA ENQ bit cannot be reset by the CPU. Only the hardware resets it as a response to the DISQ command, or an end-condition, error condition, or link down.

**Note**

Most of the terms used to denote either DMA commands (Enable\_Queue and Disable\_Queue) or interrupts (TxBuffer, TxEnd, and TxError) actually reflect multiple terms (one per queue). For example, the device provides eight Enable\_Queue commands. The same applies to the other commands and interrupts listed above.

## 8.3.4

## Receive DMA Descriptors

### 8.3.4.1

### Receive Operation

To initialize a receive operation, the CPU must perform the following:

1. Prepare a chained list of descriptors and packet buffers.

**NOTE:** The RxDMA supports eight priority queues. If the user wants to take advantage of this capability, a separate list of descriptors and buffers should be prepared for each of the priority queues.

2. Write the pointer to the first descriptor to the DMA's current receive descriptor registers (RxC DP) associated with the priority queue to be started. If multiple priority queues are needed, the user has to initialize RxC DP for each queue.

- 
1. When the DMA stops due to a null descriptor pointer, the CPU has to write TxC DP before issuing an Enable\_Queue command. Otherwise, TxC DP remains null and the DMA cannot restart the queue processing.

3. Initialize and enable the DMA channel by writing to the DMA's configuration and command registers.
4. Initialize the Ethernet port by writing to the port's configuration registers (among them PSCR, Address Filter Tables, MII/GMII Serial Parameter registers, if necessary) for the desired operational modes. Enable the port by writing to the <PortEn> bit in the Port Serial Control0 register.

After completing these steps, the port starts waiting for a receive frame to arrive to the PHY interface. When this occurs, receive data is packed and transferred to the RxFIFO. At the same time, an address filtering check is performed to decide if the packet is destined to this port. If the packet passes the address filtering check, a decision is made regarding the destination queue to which this packet should be transferred. When this is done, the actual data transfer to memory takes place. For detailed address filtering and priority queue assignment decisions, refer to [Section 8.4, Receive Frame Processing, on page 134](#).



**Note**

- Packets that fail address filtering are dropped and are not transferred to memory.
- The received packet is padded with two null bytes (the received data starts on the third byte of the buffer). This is useful for the IP header to be placed in a 32-bit aligned address in memory (as expected by SW IP stack).

For packets that span more than one buffer in memory, the DMA fetches new descriptors as necessary. However, the first descriptor pointer will not be changed until packet reception is completed.

When reception is completed, the status is written to the first descriptor, and the Next Descriptor's address is written to the current descriptor pointer register. This process is repeated for each received packet.

Only after the packet had been fully received and status information was written to the first LW of the first descriptor, will the ownership bit be reset (that is, the descriptor is returned to CPU ownership).

Ownership of any descriptor, other than the first, is returned to the CPU upon completion of the data transfer to the buffer pointed by that descriptor. This means that, for each packet, the first descriptor of a packet is the last descriptor to return to CPU ownership.

#### **8.3.4.2 Receive DMA Pointer Register**

The Rx DMA employs one 32-bit pointer register per queue: RxCDP.

RxCDP is a 32-bit register used to point to the first descriptor of a receive packet. The CPU must initialize this register before enabling DMA operation. The value used for initialization should be the address of the first descriptor to use. The CPU must not write to this register while the DMA is enabled. Reading from this register could be used to assess the DMA progress, as well as to monitor the queue status.

#### **8.3.4.3 Receive DMA Notes**

The Receive DMA process is packet oriented. The DMA does not close the first descriptor of a packet, until the last descriptor of the packet is closed. When closing the first descriptor, the DMA writes the status to the Command/Status word and resets the ownership bit. A RxBuffer maskable interrupt is generated if the <EI> bit in the first descriptor is set.

When the DMA encounters a null next descriptor pointer or a CPU owned descriptor during normal operation (both are the only legal queue end conditions), the current received frame may be closed with error status in the descriptor, if there is insufficient space to store it in memory. The RxDMA engine will assert a maskable RxErrorQueue interrupt.

If the end-condition was a null next descriptor pointer, the DMA disables the queue by resetting the <ENQ> bit once it tries to prefetch the next descriptor. If the RxDMA requests a new descriptor before the CPU re-enables the queue, the DMA increments the Discarded Frames Counter (DFC). Any new frame to this queue will be discarded.

If the ending condition was a unowned descriptor, then the DMA does not disable itself, but rather continues to try to read the descriptor, every time a new frame arrives to this queue.

The latter case optimizes for high speed descriptor-buffer receive allocation, as it allows the CPU to avoid re-enabling the queue, every time it adds new descriptors to the queue.

Before the CPU may enable the queue again, it must write the correct descriptor pointer to the RxCDP register. Alternatively, in case the queue end was a result of an unowned descriptor, the CPU may simply provide ownership of it to the DMA and re-enable it.

When a frame is received while the Ethernet link is lost (link down), the last frame received is cut off and closed as a bad CRC in the first descriptor.

The CPU must never modify the DMA configuration register or the RxCDP register while the DMA <ENQ> bit is set. Modifying the RxCDP registers is allowed only when the respective DMA <ENQ> bit is reset.

DMA <ENQ> bits are reset after the CPU writes to the <DISQ> bits, and the DMA completes the current transaction on the disabled Queue (if working with the specific disabled Queue). If the CPU gets a NULL of not owned descriptor in the middle of a packet and the CPU does not solve the problem in time, the frame will be discarded, The last closed descriptor will be reclosed as a last descriptor, and the first descriptor will be closed with a resource error.

When disabling a receive queue with the Disable-Queue command, it is necessary to ensure that the DMA is actually disabled. Poll for the <DISQ> bit to be set to 1 before trying to modify any of the DMA parameters relevant to this queue.



**Note** The RX DMA does not reset the enable bits under link down. To re-program, disable the queue by writing to the <DISQ> field in the Receive Queue Command (RQC) Register ([Table 444 p. 585](#)).

#### 8.3.4.4

#### Frame Type Indications

The receive processing of the frame (see [Section 8.4, Receive Frame Processing, on page 134](#)) allows passing various useful indications about each packet in the Rx descriptor. These indicators include MAC level errors (like Ethernet CRC check fail), to facilitate CPU processing overhead in packet header processing, and in Layer 3 and Layer 4 checksum calculations.

See the descriptor description for details. For a definition of the indications, see [Section 8.4, Receive Frame Processing](#).

#### 8.3.4.5

#### TCP Checksum Checking

TCP frames include a 16-bit checksum that protects the entire segment payload (that usually spans over a number of packets) as well as the TCP header and some of the IPv4 fields.

Frames may be received in an interleaved fashion from different TCP connections, and also out of order, within any TCP connection.

The Rx frame parsing allows off loading most of the overhead from the software. The Rx descriptor below, provides frame type indications such as: IPv4, validity of IP header with correct IPHL, IPTL, and IP checksum checked OK, Layer 2 encapsulation information (VLAN, Ethernet2 or LLC/SNAP), and TCP or UDP type detection.

TCP checksum check results are generated in the Rx descriptor in the following way, where two cases are identified - fragmented or non-fragmented IP packets:

1. If it is a non-fragmented IP packet (the packet's IP Header Flag **<MF>** = 0 and **<Offset>** = 0x0), meaning, the packet includes both TCP header and the full payload, (and therefore TCP checksum can be calculated), the descriptor will be closed with an indication that the frame is not fragmented (descriptor's **<IPv4Frg>** bit set to 0). The TCP checksum calculation also takes into account the pseudo-header if the **<RxCS>** field in the Port Configuration (PxC) Register ([Table 416 p. 562](#)) is set to 1. If **<RxCS>** is set, the L4 checksum compare result will be valid (descriptor's **<L4ChkOK>** bit is valid).



**Note**

The length field for the pseudo-header is taken from the following operation:  $\text{IPTL} - \text{IPHL} * 4$ .

- 
- For the checksum calculation, the value 16'h00 is used instead of the checksum field in the received frame as required by the standard. In addition, the checksum calculation for each frame always starts with the initial value of 16'h00.
2. If it is a fragmented IP packet (either **<MF>** != 0 or **<Offset>** != 0) or if **<RxCS>** is set to 0 (calculation without pseudo-header), the pseudo-header is not calculated in the checksum. The checksum is calculated only on the TCP segment (header + data) and places the result in the first descriptor of each frame. Therefore, the checksum compare bit (L4ChkOK bit) is not valid.



**Note**

For fragmented IP packets, the checksum calculation does not put a zero in the checksum field, and therefore, in a frame that is the first fragment of IP packet (**<Offset>** = 0x0 and **<MF>** != 0), the checksum result can be wrong (since it includes the checksum field of the TCP header). This should be corrected by the software.

The port calculates the checksum per each packet. The software should sum all the checksum calculations for the complete IP packet and compare it to the checksum field in the TCP header.

#### 8.3.4.6

#### UDP Checksum Checking

UDP frames include a 16-bit checksum that protects the entire segment payload (that usually spans over a number of packets) as well as the UDP header and some of the IPv4 fields.

Frames may be received in an interleaved fashion from different UDP streams, and also out of order, within any UDP stream.

The Rx frame parsing allows off loading most of the overhead from the software. The Rx descriptor below, provides frame type indications such as: IPv4, validity of IP header with correct IPHL, IPTL, and IP checksum checked OK, Layer 2 encapsulation info (VLAN, Ethernetv2 or LLC/SNAP), and TCP or UDP type detection.

UDP checksum check results are generated in the Rx descriptor in the following way, where two cases are identified - fragmented or non-fragmented IP packets:

1. If it is a non-fragmented IP packet (the packet's IP Header Flag **<MF>** = 0 and **<Offset>** = 0x0), meaning, the packet includes both the UDP header and the full payload, (and therefore UDP checksum can be calculated), the descriptor will be closed with an indication that the frame is not fragmented (descriptor's **<IPv4Frg>** bit set to 0), and the L4 checksum compare result will be valid. UDP checksum calculation will also take into account the pseudo-header if the

<RxCS> field in the Port Configuration (PxC) Register ([Table 416 p. 562](#)) is set to 1.



The length field for the pseudo-header is taken from the following operation:

IPTL - IPHL \* 4.

For the checksum calculation, the value 16'h00 is used instead of the checksum field in the received frame as required by the standard. In addition, the checksum calculation for each frame always starts with the initial value of 16'h00

If the frame checksum is 0x0, then the checksum function does not compare and close the descriptor as checksum OK, since this is the indication that checksum check was disabled, according to the standard.

2. If it is a fragmented IP packet (either <MF> != 0 or <Offset> != 0) or if <RxCS> is set to 0 (calculation without pseudo-header), the pseudo-header is not calculated in the checksum. The checksum is calculated only on the UDP segment (header + data) and places the result in the first descriptor of each frame. Therefore, the checksum compare bit (L4ChkOK bit) is not valid.



Note

For fragmented IP packets, the checksum calculation does *not* put zero in the checksum field, and therefore, in a frame that is the first fragment of IP packet (<Offset> =0x0 and <MF>!= 0), the checksum result might be wrong (since it includes the checksum field of the UDP header). This should be corrected by the software.

The port calculates the checksum per each packet. The software should sum all the checksum calculations for the complete IP packet and compare to the checksum field in the UDP header.

#### 8.3.4.7 BPDU Indication

If a frame is detected as BPDU, and BPDU detection is enabled, the BPDU bit is set (see [Section 8.4, Receive Frame Processing, on page 134](#)). The rest of the L3/4 fields are still provided, but the user may want to ignore them, as they are likely not to be relevant for most BPDU protocols.

#### 8.3.4.8 Receive Descriptor Structure

- The descriptor length is 4LW, and it must be 4LW aligned (i.e. Descriptor\_Address[3:0]==0000).
- Descriptors may reside anywhere in the address space except for the null address (0x00000000), that is used to indicate the end of a descriptor chain. Descriptors cannot be placed on a Device-bus as they are fetched always in a burst of 4LW.
- The last descriptor in the linked chain must have a null value in the Receive Descriptor - Next Descriptor's <NextDescriptorPointer> bits[31:0] ([Table 42 on page 134](#)). Alternatively, the last descriptor may be not owned. The latter option is useful for performance optimization, by using a dummy pointer for adding descriptors to a chain without reprogramming the RxCDP register (see [Section 10.5.4.2, Chain Mode, on page 202](#) and [Section 10.5.4.5, Descriptor Ownership, on page 204](#)).
- Receive buffers associated with receive descriptors are limited to 64-1 KB and must be 64-bit aligned (i.e. Buffer\_Address[2:0]==000).
- The minimum buffer size for the receive buffer is eight bytes.

**Figure 25: Receive Descriptor Description**

---

3	3	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	Offset																																								
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0																																									
<b>Byte 3</b>		<b>Byte2</b>		<b>Byte1</b>		<b>Byte0</b>																																																																		
Command / Status																												+0																																												
Byte Count[15:0]															Buffer Size[15:3]															I	0	0	+4																																							
Buffer Pointer [31:3]																														+8																																										
Next Descriptor Pointer [31:4]																																0	0	0	+C																																					

### 8.3.4.9 Receive Descriptor Command/Status

**Table 39: Receive Descriptor Command/Status**

Bits	Name	Description
0	ES	Error Summary 0 = No Error 1 = Error Occurred (RE or MF or OR or CE), <b>NOTE:</b> This is only valid if <F> bit[27] is set.
2:1	EC	MAC Error Coding 00 = CE - CRC Error 01 = OR - Overrun Error 10 = MF - Maximum Frame Length Error. Frame is longer than the MAX_FRAME_SIZE. 11 = RE - Resource Error (No descriptors in the middle of the frame) <b>NOTE:</b> This is only valid if the <F> bit[27] and the <ES> bit[0] are set.  If multiple errors occurred, then the reporting priority is Resource Error, Maximum Frame Length Error, Overrun Error, and CRC Error.
18:3	L4Chk	Calculated TCP/UDP Checksum <b>NOTE:</b> This is only valid if Layer4 bits[22:21] are set to 00 or 01.  This is only valid if the <F> bit[27] is set and no MAC errors occurred.  This is only valid if the <IPHeadOK> bit[25] and the <L3IP> bit[24] are set.  The calculation does not include the pseudo-header if the Receive Descriptor -Byte Count Register <IPv4Frg> bit[2] is set or the <RxCS> field in the Port Configuration (PxC) Register ( <a href="#">Table 416 p. 562</a> ) is set to 0 (calculation without the pseudo-header).
19	VLAN	VLAN Frame is VLAN tagged (according to programmed VLAN-Ethertype). <b>NOTE:</b> This is only valid if the <F> bit[27] is set, and the <ES> bit[0] is set to 0.
20	BPDU	Bridge Protocol Data Unit Set when the frame is BPDU. <b>NOTE:</b> Only valid if the <F> bit[27] is set and the <ES> bit[0] is set to 0.
22:21	Layer4	Frame encapsulation and protocol. 00 = Frame is TCP over IPv4 over Ethernets2 or LLC/Snap (with or without VLAN tag). The Checksum result is provided in the <L4Chk> bits[18:3]. 01 = Frame is UDP over IPv4 over Ethernets2 or LLC/Snap (with or without VLAN tag). The Checksum result is provided in the <L4Chk> bits[18:3]. 10 = Other Frame type. 11 = Reserved <b>NOTE:</b> This is only valid if the <F> bit[27] is set, and the <ES> bit[0] is set to 0.  This is only valid if the <IPHeadOK> bit[25] and the <L3IP> bit[24] are set.
23	Layer2Ev2	Set if Layer 2 is Ethernets2.
24	L3IP	Frame type is IPv4. This is only set if EtherType-0x800 over Ethernets2, or over LLC/SNAP (with or without VLAN tag). Otherwise, reset. <b>NOTE:</b> This is only valid if the <F> bit[27] is set, and the <ES> bit[0] is set to 0.

**Table 39: Receive Descriptor Command/Status (Continued)**

Bits	Name	Description
25	IPHeadOK	<p>IP header is “ok” if:</p> <ul style="list-style-type: none"> <li>• Frame type is IPv4</li> <li>• IPhL <math>\geq 5</math></li> <li>• IPhL <math>\ast 4 \leq</math> IPTL</li> <li>• IPheader Checksum is OK.</li> </ul> <p>0 = Check failed 1 = Check passed</p> <p><b>NOTE:</b> This is only valid if &lt;L3IP&gt; bit[24] is set.  This is only valid if the &lt;F&gt; bit[27] is set, and the &lt;ES&gt; bit[0] is set to 0.</p>
26	L	Last Indicates the last buffer of a frame.
27	F	First Indicates the first buffer of a frame.
28	U	<p>Unknown Destination Address The frame is Unicast and was not matched to the MAC Address Base (DA[47:6]).</p> <p><b>NOTE:</b> This is only set if working in promiscuous mode. See &lt;UPM&gt; field in the Port Configuration (PxC) Register (<a href="#">Table 416 p. 561</a>).</p> <p>This is only valid if the &lt;F&gt; bit[27] is set, and the &lt;ES&gt; bit[0] is set to 0.</p>
29	EI	<p>Enable Interrupt When set, a maskable interrupt is generated upon the closing descriptor.</p> <p><b>NOTE:</b> To limit the number of interrupts and prevent an interrupt per buffer situation, set the &lt;RIFB&gt; field in the SDMA Configuration (SDC) Register (<a href="#">Table 422 p. 564</a>).</p> <p>Interrupts may be further delayed by the Interrupt coalescing mechanism (see <a href="#">Section 8.7.1, Interrupt Coalescing, on page 144</a>).</p>
30	L4ChkOK	<p>Layer4 Checksum OK 1 = OK (passed) 0 = Check failed</p> <p><b>NOTE:</b> If &lt;Layer4&gt; bits[22:21] is 01 and the received frame checksum field was 16'h00, then the bit will indicate passed.</p> <p>This is only valid if the &lt;IPHeadOK&gt; bit[25] and the &lt;L3IP&gt; bit[24] are set.</p> <p>This is only valid if Layer 4 is 00 or 01.</p> <p>This is only valid if the &lt;F&gt; bit[27] is set, and the &lt;ES&gt; bit[0] is set to 0.</p> <p>This is only valid if the Receive Descriptor - Byte Count Register IPv4Frg bit[2] is cleared and the &lt;RxCS&gt; field in the Port Configuration (PxC) Register (<a href="#">Table 416 p. 562</a>) is set to 1.</p>
31	O	Ownership 0 = Buffer owned by the CPU. 1 = Buffer owned by the DMA.

**Table 40: Receive Descriptor Byte Count**

Bits	Name	Description
1:0	Reserved	Reserved.
2	IPv4Frg	<p>IPv4 is fragmented.            1 = Fragmented            0 = Not fragmented</p> <p><b>NOTE:</b> This is only valid if the &lt;IPHeadOK&gt; bit[25] and the &lt;L3IP&gt; bit[24] are set.</p> <p>This is only valid if the Receive Descriptor - Command/Status Register &lt;F&gt; bit[27] is set, and &lt;ES&gt; bit[0] is set to 0.</p>
15:3	Buffer Size	<p>Buffer Size in Bytes</p> <p>When the number of bytes written to this buffer is equal to the field's value, the DMA closes the descriptor and moves to the next descriptor.</p> <p><b>NOTE:</b> The buffer size is represented by 16-bits, where the lower 3 bits are fixed to zero.            Buffer Size = Bits[15:3]    '000'.</p>
31:16	Byte Count	<p>When a descriptor is closed, this field is written by the device with a value indicating the number of bytes actually written by the DMA into the buffer (or buffers, where the packet occupies more than one buffer).</p> <p><b>NOTE:</b> This is only valid if the &lt;F&gt; bit[27] is set.</p>

**Table 41: Receive Descriptor Buffer Pointer**

Bits	Name	Description
31:0	Buffer Pointer	<p>A 32-bit pointer to the beginning of the buffer associated with this descriptor.</p> <p><b>NOTE:</b> This field must be 64-bit aligned; therefore, bits[2:0] must be set to 0.</p>

**Table 42: Receive Descriptor Next Descriptor Pointer**

Bits	Name	Description
31:0	Next Descriptor Pointer	<p>A 32-bit pointer that points to the beginning of the next descriptor.</p> <p><b>NOTE:</b> This field must be 4LW aligned; therefore, bits[3:0] must be set to 0.            The DMA operation is stopped when a null (all zeros) value in the Next Descriptor Pointer field is encountered.</p>

## 8.4 Receive Frame Processing

Once a frame is received by the port, the frame is parsed through the following processing:

- MAC errors checking
- Accept or reject decision
- Select the receive queue (0 through 7)
- MIB counter increments
- Extract Layer 2/3/4 protocols and perform IP and/or TCP/UDP checksum

Some MAC level errors, like fragments, are normally filtered from reception of frames and are only counted in MIB counters. Other MAC level errors (like CRC error frames and frames beyond the maximum allowed size) are reported in the first descriptor and in the MIB counters.

The frames maximum size is defined in the [\*\*<MRU>\*\*](#) field in the Port Serial Control0 (PSC0) Register ([Table 427 p. 568](#)). The receiver can also accept Jumbo frames, where the IEEE 802.3 Type/Length field is set to 0x8870 (with or without IEEE 802.1Q VLAN tag).



**Note**

In this datasheet, the MAC Destination address bit[47] is the Multicast/Unicast bit. The first DA byte received on the GMII RXD[7:0] pins is DA[40:47]. The last byte GMII received on the RXD[7:0] pins is DA[0:7].

## 8.4.1 Parsing the Frames

### 8.4.1.1 Filtering

The frame goes through the following stages that determine if it is accepted:

1. If the frame is in the Bridge Protocol Data Unit (BPDU) format (DA is equal to 01-80-C2-00-00-00 through 01-80-C2-00-00-FF, except for the Flow-Control Pause packets), frame is accepted/rejected according to the [\*\*<Span>\*\*](#) field in the Port Configuration Extend (PxCX) Register ([Table 417 p. 562](#)).
2. If the frame is Unicast then the MAC DA bits[47:4] are compared with MAC[47:4] (see the MAC Address Low (MACAL) Register ([Table 420 p. 563](#)) and MAC Address High (MACAH) Register ([Table 421 p. 563](#))).
  - If they do not match, the frame is accepted/rejected according to [\*\*<UPM>\*\*](#) field in the Port Configuration (PxC) Register ([Table 416 p. 561](#)) (Unicast Promiscuous Mode).
  - If they match, then the MAC DA[3:0] bits are used as a pointer to the Unicast Table entries in the DA-Filter table. Frame is accepted/rejected according to the appropriate [\*\*<Pass>\*\*](#) bit in the Destination Address Filter Special Multicast Table (DFSMT) Register (n=0–63) ([Table 447 p. 586](#)).
3. If DA=0xFFFFFFFF and the protocol is 0x806 (an ARP broadcast) in Ethernet-v2 (tagged or not), frame is accepted/rejected according to the [\*\*<RBArp>\*\*](#) field in the Port Configuration (PxC) Register ([Table 416 p. 561](#)).
4. If DA=0xFFFFFFFF and the protocol is 0x800 (an IP broadcast), in Ethernet-v2 or LLC/SNAP (tagged or not), the frame is accepted/rejected according to the [\*\*<RBIP>\*\*](#) in the same register.
5. If DA=0xFFFFFFFF and it is not an ARP nor an IP packet (another type of broadcast), the frame is accepted/rejected according to the [\*\*<RB>\*\*](#) in the same register.
6. If DA=0x01-00-5E-00-00-XX (an IP multicast; XX is between 0x00 and 0xFF) the MAC DA[7:0] bits are used as a pointer to the Special Multicast Table entries in the DA-Filter table. Frame is accepted/rejected according to the [\*\*<Pass>\*\*](#) bit.
7. If DA bit[0] is 1 and it is not a broadcast nor an IP multicast (the frame is a Multicast of another type), a 8-bit polynomial CRC is calculated ( $x^8+x^2+x^1+1$ ) and the result is used as an index to the Multicast Broadcast Table entries in the DA-Filter table. Frame is accepted/rejected according to the [\*\*<Pass>\*\*](#) bit.

### 8.4.1.2 Enqueuing

If the frame is accepted, the receive queue is determined according to the following flow:

1. If it is a BPDU packet, the Rx queue is determined by [\*\*<BPDUQ>\*\*](#) field in the Port Configuration (PxC) Register ([Table 416 p. 562](#)) (by default, it is the highest priority queue - 7).
2. If it is an ARP broadcast packet, the Rx queue is determined by the [\*\*<RXQArp>\*\*](#) field in the same register.
3. If it is a Unicast packet that failed Unicast address compare (but the packet is still accepted because of Promiscuous Mode), the Rx queue is determined by the [\*\*<RXQ>\*\*](#) field in the same register (that is the default receive queue).

If none of the above three conditions is met, Rx queue selection proceeds as follows:

1. If it is a TCP packet, the Rx queue is determined according to [\*\*<TCPQ>\*\*](#).
2. If it is a UDP packet, the Rx queue is determined according to [\*\*<UDPQ>\*\*](#).

If it is not either a TCP or a UDP packet, the Rx queue selection proceeds as follows:

1. Calculate the Rx queue as follows:
  - If it is a Unicast packet that passed address compare, MAC DA[3:0] bits are used as a pointer to the Unicast Table entries in the DA-Filter table. The table's **<Queue>** bits determines Rx queue number.
  - If it is an IP Multicast packet, MAC DA[7:0] bits are used as a pointer to the Special Multicast Table entries in the DA-Filter table. The table's **<Queue>** bits determine the Rx queue number.
  - If it is another type of Multicast packet, an 8-bit CRC value is used as an index to the Multicast Broadcast Table entries in the DA-Filter table. The table's **<Queue>** bits determine the Rx queue number.
  - If it is a Broadcast packet (IP or other), the Rx queue is determined by [\*\*<RXQ>\*\*](#) field in the Port Configuration (PxC) Register ([Table 416 p. 561](#)).
2. If it is a VLAN tagged packet, the Rx queue is determined by VLAN Priority Tag to Priority (VPT2P) Register ([Table 428 p. 569](#)).
3. If it is an IPv4 packet, the Rx queue is determined by IP Differentiated Services CodePoint 0 to Priority (DSCP0) Register ([Table 423 p. 565](#)).
4. Since a packet can match any of the above three conditions (Steps 1, 2, and 3 of this list)(e.g. a Unicast VLAN tagged IPv4 packet), the Rx queue number is determined as the *highest* queue from the one determined by the above three conditions.

**Note**

- To not use VLAN or DSCP mapping, program those registers as all zero values, which are their default values.
- If the packet has an unknown L3 type (e.g. IPv6), it is still accepted and queued based on L2 MAC address (and VLAN priority if it is also VLAN tagged).
- Before enabling the port, the DA-Filter tables must be programmed in full, as their initial values is undefined.
- The receive queue can be configured to be according to Marvell Header or DSA tag. In tat configuration, the receive queue defined by the Marvell Header/DSA tag overides the above enqueueing decision.

## 8.5

## Marvell® Header Support

When interfacing a Marvell SOHO switch, Marvell recommends to enable the Marvell Header mode in the switch and the GbE port. The Marvell Header consists of two octets placed ahead of the DA. The Marvell Header contains information used by the software to determine from which of the switch ports the packet arrived and to maintain QoS.

To enable Marvell Header mode, set the [\*\*<MHEn>\*\*](#) field in the Marvell Header Register ([Table 432 p. 575](#)) to 1.

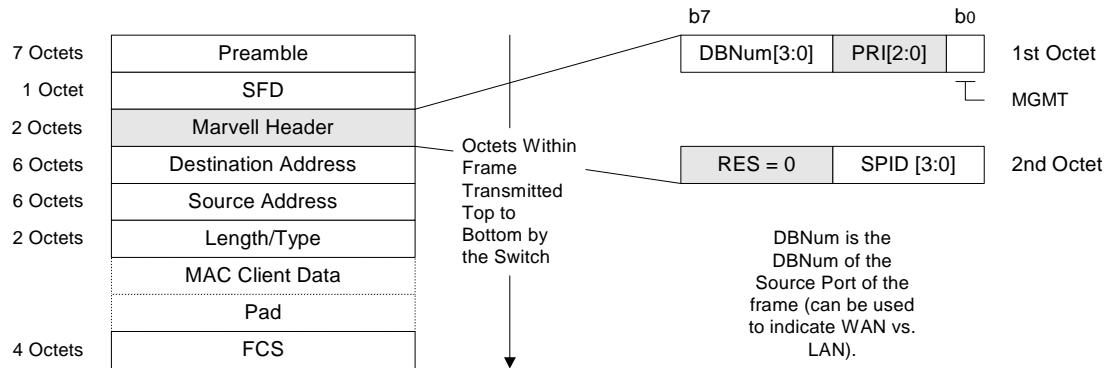
**Note**

- See the Marvell SOHO switches specification for additional information about the Marvell Header mode.
- When the Marvell header is enabled, the GbE port does not add two bytes when an IP header alignment is received. The packet received from the switch is padded with the Marvell Header two octets.

### 8.5.1 Receive Operation

When the Marvell Header mode enabled in the switch, it pads each packet with two octets just before the DA as shown in [Figure 26](#).

**Figure 26: Rx Packet Marvell Header Example**



[Table 43](#) details Marvell header fields.

**Table 43: Marvell Header Fields**

Name	Description
DBNum[3:0]	Database Number Represents the address database assigned to this frame, when it ingresses into the switch (assigned by the switch source port). Used to indicate the VLAN number of the source port.
PRI[2:0]	Frame priority The priority assignment details are in the switch specification.
RES	Reserved for future use
SPID[3:0]	Source Port ID Indicates the physical port on the frame that the switch entered.
MGMT	Management For details on the MGMT bit assignment, see the switch specification.



- The Marvell Header is transferred along with the rest of the packet to memory. Its fields are not updated in the Rx descriptor status.
- Packet CRC generation/checking includes the two Marvell Header octets.

The GbE port also supports Rx queuing based on Marvell Header fields. This is an alternative to the regular queuing described in [Section 8.4.1, Parsing the Frames, on page 135](#).

The <DA\_Prefix> field in the Marvell Header Register ([Table 432 p. 576](#)) selects the queuing policy:

- 0x0: The regular priority queuing is working.
- 0x1: The Rx queue is determined according to PRI[2:0] bits.
- 0x2: The Rx queue is determined according to DBNum[0] and PRI[2:1].
- 0x3: The Rx queue is determined according to SPID[3:0] and PRI[2:1].

For example, if the application requires two priority queues (Low and High) for Switch Port2, and two priority queues (Low and High) for the rest of the switch ports, set the following fields in the Marvell Header Register ([Table 432 p. 575](#)):

- <DA\_PREFIX> to 0x3
- <SPID> to 0x2
- <MH\_Mask> to 0x1 (indicating to use queues 0–3)

With this configuration, packets received from Switch Port2 are placed in queues 2 (L) and 3 (H), and packets received from the other switch ports are placed in queues 0 (L) and 1 (H).



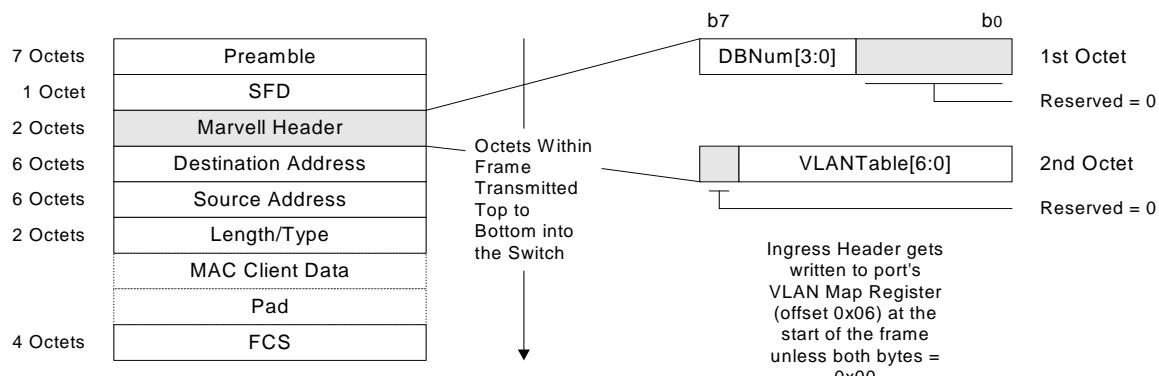
**Note**

- Packets accept/reject is handled the same way as in none Marvell Header mode as described in [Section 8.4.1, Parsing the Frames, on page 135](#) with one exception. If the packet is marked as MGMT, it is always accepted.
- Receive MIB counters do not take the Marvell Header into account. For example, if receiving a packet that has a 128 byte size (including the Marvell Header), frames 128–255 Octets counter are incremented, instead of frames 64–127 Octets counter.

## 8.5.2 Transmit Operation

With the Marvell header enabled, the software must add two octets of the Marvell header to the top of the packet, as shown in [Figure 27](#).

**Figure 27: Tx Packet with Marvell Header Example**



The Marvell Header gives the software the ability to control which VLAN and address database to use for the frame.



**Note**

See the specific SOHO switch specification for details on the Marvell Header format and usage.

The GbE port generates CRC for the entire packet, including the Marvell Header. When the switch receives the packet, it strips the Marvell Header, and recalculates the new CRC before forwarding the frame to the network.

## 8.6

## Distributed Switching Architecture (DSA) Tag Support

Most Marvell GbE switches support the Distributed Switching Architecture (DSA) feature. This is useful for cascading and stacking switch devices. Even when using a single GbE switch device, the DSA tag is useful for the software to determine from which of the switch ports the packet arrived and to maintain QoS. This is similar to using the Marvell Header mode, but DSA provides additional information on the received packet.

When interfacing one of Marvell GbE switches, Marvell recommends to enable the Marvell DSA Tag mode, both in the switch and the GbE port.

To enable Marvell the DSA feature, set the <DSAE<sub>n</sub>> field in the Marvell Header Register ([Table 432 p. 577](#)) to 1 or 2.



**Note**

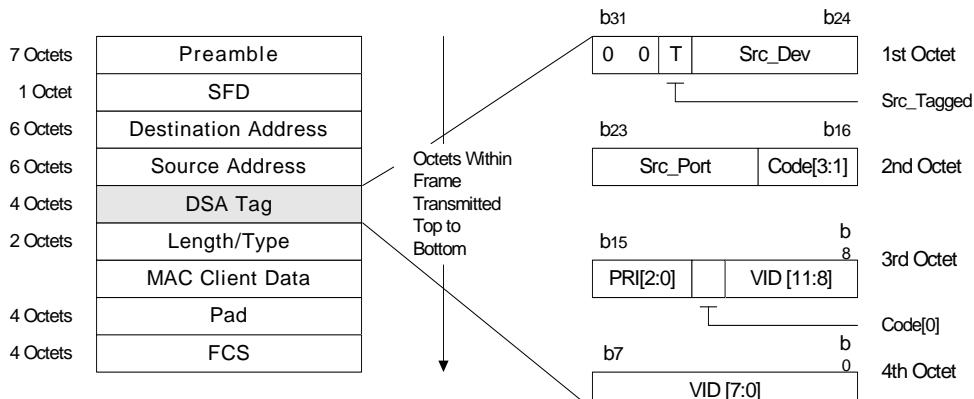
- See the Marvell GbE switch specification for additional information on DSA tag.
- The GbE port supports both extended (8 bytes) and non-extended (4 bytes) DSA tag formats.
- If using the DSA tag, disable the Marvell Header mode by setting the <MHEn> field in the Marvell Header Register ([Table 432 p. 575](#)) to 0. The DSA feature cannot be used simultaneously with the Marvell Header.

### 8.6.1

### Receive Operation

When the DSA tag is enabled in the switch, each packet is padded with four or eight octets after the SA, see [Figure 28](#).

**Figure 28: Rx Packet with DSA Tag Example (4 bytes tag, TO\_CPU Format)**



The GbE port can receive one of the two DSA tag formats: TO\_CPU format or FORWARD format. [Table 44](#) and [Table 45](#) summarize the DSA tag fields (for full details, see the GbE switch specification).



Table 44: DSA Tag Fields (TO\_CPU Format)

Bits	Name	Description
<b>Word0</b>		
31:30	TagCommand	00 = TO_CPU
29	SrcTagged/ TrgTagged	0 = Packet was received/transmitted from/to a network port untagged. 1 = Packet was received/transmitted from/to a network port tagged.
28:24	SrcDev/ TrgDev	Source/Destination device number on which the packet was received/transmitted.
23:19	SrcPort[4:0]/ TrgPort[4:0]	Source/Destination port number on which the packet was received/transmitted.
18:16	CPU_Code[3:1]	CPU_Code[3:1] when using none extended mode. CPU_Code[3:0] must be set to 0xF to indicate an Extended DSA tag.
15:13	UP	The IEEE 802.1p User Priority field assigned to the packet.
12	CPU_Code[0]	CPU_Code[0] when using none extended mode. CPU_Code[3:0] must be set to 0xF to indicate an Extended DSA tag.
11:0	VID	The packet's incoming/outgoing VID
<b>Word1</b>		
31	Extend	Word1 is the last extension. Must be set to 0.
30	CFI	When SrcTagged = 1, this is the VLAN Tag CFI bit with which the packet was received from the network port.
29:27	Reserved	
26	Truncated	Packet sent to CPU is truncated. Indicates that only the first 128 bytes of the packet are sent to the CPU. The packet's original byte count is forwarded to the CPU in <PktOrigBC> field.
25:12	PktOrigBC	The packet's original byte count.
11	Reserved	
10	SrcPort[5]/ TrgPort[5]	Bit 5 (MSB) of the SrcPort[5:0]/TrgPort[5:0] field.
9	Reserved	
8	SrcTrg	SrcTrg indicates the type of data forwarded to the CPU. 0 = The packet was forwarded to the CPU by the Ingress pipe and this tag contains the packet's source information. 1 = The packet was forwarded to the CPU by the Egress pipe and this tag contains the packet's destination information.
7:0	LongCPUCode	8 bit of CPU code

**Table 45: DSA Tag Fields (FORWARD Format)**

<b>Bits</b>	<b>Name</b>	<b>Description</b>
<b>Word0</b>		
31:30	TagCommand	11 = FORWARD
29	SrcTagged	0 = Packet was received from a network port untagged. 1 = Packet was received from a network port tagged.
28:24	SrcDev	Source device number on which the packet was received.
23:19	SrcPort[4:0]/ SrcTrunk[4:0]	If SrcIsTrunk = 0, indicates source port number on which the packet was received. If SrcIsTrunk = 1, indicates source trunk number on which the packet was received.
18	SrcIsTrunk	If the packet was received from a network port that is part of a trunk, this bit is set to 1
17	Reserved	
16	CFI	When SrcTagged = 1, this is the VLAN Tag CFI bit with which the packet was received from the network port
15:13	UP	The IEEE 802.1p User Priority field assigned to the packet.
12	Extend	1 = There is one more DSA tag word.
11:0	VID	The packet's incoming/outgoing VID
<b>Word1</b>		
31	Extend	Word1 is the last extension. Must be set to 0.
30	SrcTrunk[6]	When SrcIsTrunk = 1, this is the VLAN Tag CFI bit with which the packet was received from the network port (if SrcIsTrunk = 0, this bit is reserved)
29	SrcPort[5]/ SrcTrunk[5]	If SrcIsTrunk = 0, indicates source port number on which the packet was received. If SrcIsTrunk = 1, indicates source trunk number on which the packet was received.
28	EgressFilter Registered	If set to 1, indicates that the packet is Egress filtered as a Registered packet (when this field is 0, the type of the packet—Multicast or Unicast—is set according to the packet's MAC DA[40]).
27:26	Reserved	
25	Routed	If set to 1, indicates that the Packet has been Layer 3 routed.
24:20	SrcID	Packet's Source ID.
19:13	QoSProfile	Packet's QoS profile.
12	use_vidx	0 = Unicast packet forwarded to the Target port specified in this tag. 1 = Multicast packet forwarded to the Multicast group specified in this tag.
11	VIDX[11]	When use_vidx = 1, indicates the Multicast group to which the packet is transmitted (if use_vidx = 0, reserved)
10:5	VIDX[10:5]/ TrgPort	When use_vidx = 1, indicates the Multicast group to which the packet is transmitted. When use_vidx = 0, specifies the target port to which the packet is forwarded.

**Table 45: DSA Tag Fields (FORWARD Format) (Continued)**

Bits	Name	Description
4:0	VIDX[4:0]/TrgDev	When use_vidx = 1, indicates the Multicast group to which the packet is transmitted. When use_fivx = 0, specifies the target device to which the packet is forwarded.


**Note**

- The DSA tag is transferred along with the rest of the packet to memory. None of its fields are updated in the Rx descriptor status.
- Packet CRC generation/checking includes the four/eight octets of the Marvell Header.

The GbE port also supports Rx queuing based on DSA tag fields. This is an alternative to the regular queuing, as described in [Section 8.4.1, Parsing the Frames, on page 135](#).

The `<DAPrefix>` field in the Marvell Header Register ([Table 432 p. 576](#)) selects the queuing policy:

- If set to 0x0, the regular priority queuing is working.
- If set to 0x1, Rx queue is determined according to UP field.
- If set to 0x2, needs to distinguish between two cases:
  - If it is a FORWARD format, the Rx queue is determined according to UP field (as if `<DA_PREFIX>` is set to 1).
  - If it is a TO\_CPU format, the Rx queue is determined according to the 4-bit/8-bit CPU\_Code field. When using `<DAPrefix>` is set to 2, the Destination Address Filter Special Multicast Table (DFSMT) is no longer used for address filtering. Instead, it is used for mapping from CPU\_Code to Rx queue.
- If set to 0x3, the Rx queue is determined according to SrcPort, SrcDev, and UP[2:1].

For example, if the application requires four priority queues (Low and High) for Switch Device Number3 Port2, and four priority queues (Low and High) for the rest of the switches ports, set the following fields in the Marvell Header Register ([Table 432 p. 575](#)):

- `<DAPrefix>` to 0x3
- `<SPID>` field to 0x2
- `<SDID>` field to 0x3
- `<MHMask>` to 0x0 (indicating to use all 8 queues)

This way, packets received from Switch Number 3 Port2 are placed in queues 4–7, and packets received from the other ports are placed in queues 0–3.

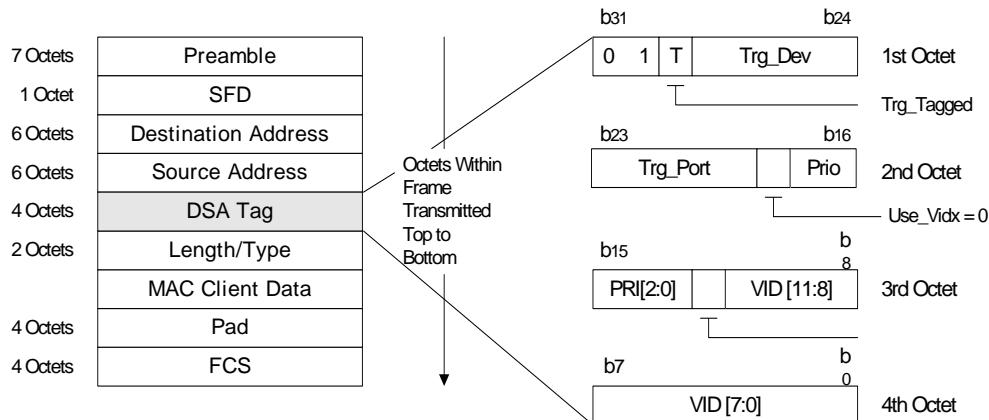

**Note**

- Packet accept/reject is handled in the same way as described for the Marvell Header mode, see [Section 8.4.1, Parsing the Frames, on page 135](#).
- When `<DAPrefix>` is set to 0x3, and the `<SDIDEn>` bit is set to 0, the `<SrcDev>` field are not taken into account in the queuing decision. Only the SrcPort is used, as in the Marvell Header mode.
- When `<DAPrefix>` is set to 0x3, and there is no SrcPort (FORWARD format, with `SrcIsTrunk` bit set), the Rx queue is determined according to UP[2:0] field only.
- If the received packet is not set to TO\_CPU or FORWARD format, the packet is queued as if `<DAPrefix>` is set to 0x0 (regular queuing).

## 8.6.2 Transmit Operation

With the DSA tag is enabled, the software must add four/eight octets of DSA tag after DA, as shown in [Figure 29](#).

**Figure 29: Tx Packet with a DSA Tag Example (FROM\_CPU format, use\_vidx = 0)**



See the specific GbE switch specification for exact details on the DSA tag format and usage.

The GbE port generates CRC for the entire packet, including the DSA tag. When the switch receives the packet, it strips the DSA tag, and recalculates the new CRC before forwarding the frame to the network.

## 8.7 Ethernet Interrupts

The GbE port supports many interrupt events, registered and controlled in the following registers:

- Ethernet Unit Interrupt Cause (EUIC) Register ([Table 405 p. 556](#)) and Ethernet Unit Interrupt Mask (EUIM) Register ([Table 406 p. 557](#))
- Port Interrupt Cause (IC) Register ([Table 433 p. 577](#)) and Port Interrupt Mask (PIM) Register ([Table 435 p. 582](#))
- Port Interrupt Cause Extend (ICE) Register ([Table 434 p. 580](#)) and the Port Extend Interrupt Mask (PEIM) Register ([Table 436 p. 582](#))

Each GbE port provides four interrupt bits to the device interrupt controller. These bits are:

- Summary bit: IC bit[31]
- Rx events: Covers bits IC [18:2]; ICE bit [18]
- Tx events: Covers bits IC [26:19]; ICE bits [15:0], and [19]
- Misc: Covers bits ICE [16], [27], and [23]

Separation of receive and transmit interrupt enables faster service for packet receive and transmit. If this separation is not required, the software driver can choose to use the summary bit instead.

In addition to the per port interrupt bits, the summary of all ports EUIC registers is reported as a single bit (OR of all ports) in the device Interrupt Controller.

## 8.7.1

### Interrupt Coalescing

Since the GbE line rate provides a high packet rate, it is important to reduce the amount of interrupts that the Ethernet DMA may generate.

For this purpose, the DMA receive and transmit have several modes that provide the option of choosing the type of events that initiate issuing interrupts. (See also Port Interrupt Cause Extend (ICE) Register ([Table 434 p. 580](#)), Ethernet Unit Interrupt Cause (EUIC) Register ([Table 405 p. 556](#)).)

The most intensive interrupts are the packet-level interrupts on receive and transmit. In addition to the CPU's ability to specify, in the receive and transmit descriptor, the descriptor close that may cause an interrupt, the device provides a programmable mechanism that allows coalescing these types of interrupts.

On a per port basis, and for Rx and Tx transactions, the device has a programmable timer in the `<IPGIntRx>` field in the SDMA Configuration (SDC) Register ([Table 422 p. 564](#)) for receive and `IPG_Int_Tx` (transmit) to force a minimum time between interrupts associated with the `<RxBuffer>` field in the Port Interrupt Cause (IC) Register ([Table 433 p. 577](#)) and the `<IPGIntTx>` field in the Port Tx FIFO Urgent Threshold (PxTFUT) Register ([Table 437 p. 583](#)). This minimum time is programmable and may be changed dynamically during normal operation.

The flow for packet-level interrupts on receive and transmit is as follows:



#### Note

The following example describes the receive flow, however, the transmit flow is implemented in an identical fashion.

1. A non-masked `<RxBufferQueue>` interrupt cause is asserted. As a result, an interrupt is raised (propagated in the interrupt hierarchy etc.) and the relevant interrupt coalescing counter begins to count. From this point (after CPU read from the interrupt register) until the count-down finishes, *no new interrupts can be raised* due to new packet reception (transmission) from any of the eight queues.
2. During the countdown time, and before the next CPU read of the PICR register (or PICER for Tx), the `<RxBufferQueue>` events would still cause loading 1 to the appropriate cause bit, but would not cause raising an interrupt at the unit, port, or port-Rx (Tx) level. Before reading the register, it is assumed that the CPU does not reset the `<RxBufferQueue>` cause bits.
3. Once the CPU reads the PICR (or PICER for Tx), the value of all `<RxBufferQueue>` interrupts that are recorded later on, accumulate in a shadow register, actually two separate registers—one for PICR `<RxBufferQueue>` and one for PICER TxBuffer additional events, both of which are invisible to the software.
4. When the countdown timer expires, the `<RxBufferQueue>` in the shadow register is loaded into the PICR (or PICER for Tx). This may cause raising an interrupt again.

This mechanism prevents loss of interrupt indications in the time frame between the time that the CPU reads the interrupt register and the time that it starts switching off interrupt bits. During this interval, new interrupts that arrive for the same receive or transmit queue would have been lost and buffers might have gotten stuck indefinitely. The shadow registers, just described, prevents this from happening.

## 8.8

### Transmit Weighted Round-Robin Arbitration

The device transmit port includes flexible bandwidth control distribution among eight transmit queues. For a transmit queue to be selected to transmit the next frame, the queue must be enabled

by setting the corresponding bit to 1 in the [\*\*<ENQ>\*\*](#) field in the Transmit Queue Command (TQC) Register ([Table 450 p. 590](#)), and the queue should have a frame ready for transmit.

### 8.8.1

## Priority Modes

Each transmit queue can be configured in two modes:

- Fixed priority mode
- Weighted-Round-Robin (WRR) priority mode

The priority mode is configured by the [\*\*<FIXPR>\*\*](#) field in the Transmit Queue Fixed Priority Configuration (TQFPC) Register ([Table 451 p. 591](#)). Setting these bits to 1 means the transmit queue is set to the Fixed priority mode. Setting these bits to 0 means the queue is configured to a WRR priority mode. Transmit Queue Fixed Priority Configuration register bit[7] is assigned to Transmit Queue 7 and bit[0] is assigned to Transmit Queue 0.

The transmit port arbitrates between the queues in two modes in the following fashion: While there is an enabled non-empty Fixed priority queue, select to transmit the next frame from the Fixed priority queue or else select to transmit from the WRR priority queue(s).

### 8.8.2

## Fixed Priority Mode

Select the Fixed Priority mode to transmit by selecting the non-empty, enabled and non-bandwidth limited queue with the highest queue number first (for example, select queue 7, then 6, etc.). A queue can be excluded from the arbitration, if it passed an optional per-queue programmable bandwidth-limitation based on the token-bucket mechanism (see [Section 8.9, Token Rate Configuration, on page 146](#)).

### 8.8.3

## Weighted Round-Robin Priority Mode

The port would service one of the Weighted Round Robin (WRR) queues only when the fixed priority queues have nothing to transmit.

The WRR priority mode is utilized to distribute bandwidth among transmit queues in a round-robin fashion when each WRR queue gets bandwidth portion relative to its configured weight. This is called below “WRR arbitration”.

A WRR queue may be excluded from the WRR arbitration, if it passed an optional per-queue programmable bandwidth-limitation based on token-bucket mechanism (see [Section 8.8.4, Transmit Queue Bandwidth Limitation, on page 146](#)). Therefore, only the queues that are below the per-queue bandwidth limitation would be considered in any WRR arbitration.

Configure the queue weight (0–255) by writing to the corresponding Transmit Queue Arbiter Configuration (TQxAC) Register (n=0–7) ([Table 458 p. 594](#)) (one of eight) and setting the [\*\*<WRRWGT>\*\*](#) field to the desired weight value (measure transmit ports in 256 byte units).

As used here, the “WRR bandwidth” is the available transmit bandwidth (that is not consumed by fixed-priority ports traffic).

The WRR arbitration end result is calculated by dividing the WRR bandwidth between the WRR-queues according to each queue’s WRRWGT/ (Sum of all WRRGTs of the WRR-queues that are not bandwidth limited by the token bucket mechanism).

When several [\*\*<WRRWGT>\*\*](#) combinations yield the same bandwidth distribution, the user must use a combination with the smallest [\*\*<WRRWGT>\*\*](#) value closest to the Maximum Transmit Unit (see [Section 8.8.6, Maximum Transmit Unit, on page 146](#)).

The WRR bandwidth distribution is determined by counting the transmitted bytes from each WRR queue and limiting the schedule of queues that used up their bandwidth portion (WRRWGT) until all other queues finish transmission.

## 8.8.4

### Transmit Queue Bandwidth Limitation

To implement a bandwidth limitation on transmit queues, specify the maximum available bandwidth for each queue in approximate ranges (2 Mbs–1 Gbs) in 1024 step resolution.

The bandwidth limitation is implemented by the Token Bucket per queue. The tokens are added to the 'Bucket' in a constant configurable rate and drained from the Bucket when the queue transmits. A queue is only allowed to transmit when the number of 'Tokens' is larger than Maximum Transmit Unit (MTU). It is possible to configure the queue Token-Bucket size that gives control over the amount of credit (silent time) that the queue is allowed to accumulate.

The queue 'Token-Rate' is programmed by setting the corresponding queues [\*\*<QTKNRT>\*\*](#) field in the Transmit Queue Token Bucket Configuration (TQxTBC) Register (n=0–7) ([Table 457 p. 593](#)). This field is filled to the desired value in 1/64 bit per clock cycle units.

To disable a bandwidth limitation, set the [\*\*<QTKNRT>\*\*](#) field to its maximum value.

The queue Bucket size is programmed by writing to the corresponding queue's [\*\*<QMTBS>\*\*](#) field in the Transmit Queue Token Bucket Configuration (TQxTBC) Register (n=0–7) ([Table 457 p. 594](#)) in a value of 256-byte units. The tokens accumulate in the bucket until the amount equals the [\*\*<QMTBS>\*\*](#) setting.

The user may examine or modify the current value of a queue's token bucket by read/write to the corresponding queue's [\*\*<QTKNBKT>\*\*](#) field in the Queue Transmit Token-Bucket Counter (QxTTBC) Register (n=0–7) ([Table 456 p. 593](#)) with a value in 1/64-byte units.

## 8.8.5

### Transmit Port Bandwidth Limitation

The transmit port implements a bandwidth limitation on all outgoing traffic. This applies to Fixed priority and WRR priority queues.

It is possible to specify the maximum available port bandwidth in approximate ranges (2 Mbs–1 Gbs) in 1024 step resolution. The bandwidth limitation is implemented by a port Token-Bucket. The tokens are added to the Bucket in a constant configurable rate and drained from the Bucket when the port transmits. The port is allowed to transmit only when the number of Tokens is bigger than the MTU. It is also possible to configure the port Token-Bucket size for the amount of credit (silent time) the queue is allowed to accumulate.

The port 'Token-Rate' is programmed by setting the [\*\*<PTKNRT>\*\*](#) field in the Port Transmit Token-Bucket Rate Configuration (PTTBRC) Register ([Table 452 p. 591](#)) to the desired value in 1/64-bit per clock cycle units. To disable bandwidth limitation, set [\*\*<PTKNRT>\*\*](#) to its maximum value.

The port 'Bucket' size is set by writing to the [\*\*<PMTBS>\*\*](#) field in the Port Maximum Token Bucket Size (PMTBS) Register ([Table 455 p. 592](#)) in a value of 256-byte units. The tokens accumulate in the port bucket until the setting for [\*\*<PMTBS>\*\*](#) is met.

The user may examine or modify the current value of port Token-Bucket by reading/writing to the [\*\*<PTKNBKT>\*\*](#) field in the Port Transmit Token-Bucket Counter (PTTBC) Register ([Table 459 p. 595](#)) with a value in 1/64-byte units.

## 8.8.6

### Maximum Transmit Unit

The MTU is a configurable value common to all transmit ports and all WRR queues. It is used for bandwidth limitation implementation.

The MTU is programmed by setting the [\*\*<PMTU>\*\*](#) field in the Port Maximum Transmit Unit (PMTU) Register ([Table 454 p. 592](#)) in 256-byte units.

## 8.9

### Token Rate Configuration

The relation of the desired bandwidth (BW) limitation and Token Rate programmed value is:

$$\text{TokenRate}[1/64 \text{ bit/cycle}] = \text{BW}[\text{Mb/sec}] * 64 / \text{TCLK}[\text{MHz}]$$

[Table 46](#) shows some examples of the Token Rate bandwidth configuration values.

**Table 46: Token Rate Configuration Examples**

Bandwidth [Mbps]	TCLK [MHz]	Token Rate [1/64 bit/cycle]
2.604	167	1
1000	167	384

As shown in [Table 46](#), 10 bits for the Token Rate span 1 Gbps–2.604 Mbps (when TCLK = 167 MHz).

## 8.10 Transmit Queues Egress Jitter Pacing (EJP) Arbitration

The EJP (Egress Jitter Pacing) mechanism is an enhanced arbitration mechanism, enabling low jitter. It complies with IEEE 802.1Qav pre-draft and enables implementing AVB technology.

EJP mechanism is an arbitration mechanism for four queues.

### 8.10.1 EJP Mechanism

When EJP mode is enabled, only enable queues 0–3, disable queues 4–7.

The EJP mechanism combines the token bucket mechanism with an advanced algorithm, for optimized jitter performance.



Utilize this mode of operation only for non-half duplex and non-10-Mbps operation where flow control is not expected to enabled.

Note

The EJP algorithm is based on a combination of:

- Sending packets from specific queue, when its bucket is full enough (“full enough” means that the number of tokens accumulated in the bucket is equal to or more than the Queue Maximum Transmit Unit).
- Forcing a configurable minimum of inter-packet gap (IPG) between packets from the same queue.
- According to the leaky bucket parameters, the transmission timing of packets from queue 3 (highest priority) must not be delayed, because a packet from lower priority queue is in transmission.  
Therefore, if there is a possibility that a transmission from a lower priority queue (0,1,or 2) may delay a transmission from queue 3, then transmit queue 3 first, even if its bucket is not full enough.
- According to the leaky bucket parameters, the transmission timing of packets from queue 2 (second highest priority) must not be delayed, because a packet from lower priority queue is in transmission.  
Therefore, if there is a possibility that a transmission from a lower priority queue (0 or 1) may delay a transmission from queue 2, then transmit queue 2 before transmitting queue 0 or 1, even if its bucket is not full enough.

## 8.10.2 Initialization Sequence

To initialize the EJP follow these steps:

1. Disable TX queues 0–7 (write the value 0x0F to the [\*\*<ENQ>\*\*](#) field in the Transmit Queue Command (TQC) Register ([Table 450 p. 590](#)).
2. Configure the following fields in the Transmit Queue Command1 (TQC1) Register ([Table 453 p. 591](#)):
  - Set the [\*\*<EJP\\_ENB>\*\*](#) field.
  - Clear Reserved bit[3].
  - Set the [\*\*<WRR\\_EJP\\_INIT>\*\*](#) field.
  - Configure the [\*\*<PTP\\_SYNC\\_ENB>\*\*](#).
3. Configure the [\*\*<FIXPR>\*\*](#) field in the Transmit Queue Fixed Priority Configuration (TQFPC) Register ([Table 451 p. 591](#)).
4. Clear the [\*\*<PTKNBKT>\*\*](#) field in the Port Transmit Token-Bucket Counter (PTTBC) Register ([Table 459 p. 595](#)).
5. Configure the port token bucket parameters:
  - Configure the [\*\*<PTKNRT>\*\*](#) field in the Port Transmit Token-Bucket Rate Configuration (PTTBRC) Register ([Table 452 p. 591](#)), for Port Token Rate.
  - Configure the [\*\*<PMTU>\*\*](#) field in the Port Maximum Transmit Unit (PMTU) Register ([Table 454 p. 592](#)). This defines the minimum tokens to be filled in the port bucket, before sending the next packet from one of the queues of the port.
  - Configure the [\*\*<PMTBS>\*\*](#) field in the Port Maximum Token Bucket Size (PMTBS) Register ([Table 455 p. 592](#)). This field defines the maximum accumulated transmit credit, in 256-byte units, used for the port Token-Bucket bandwidth limitation mechanism. The [\*\*<PTKNBKT>\*\*](#) field in the Port Transmit Token-Bucket Counter (PTTBC) Register ([Table 459 p. 595](#)) is incremented up to  $PMTBS \times 256[\text{byte}] \times 64[\text{credits}/\text{byte}]$ .
6. For each of queues 0–3, configure the following parameters:
  - Clear the [\*\*<QTKNBKT>\*\*](#) field in the Queue Transmit Token-Bucket Counter (QxTTBC) Register (n=0–7) ([Table 456 p. 593](#)).
  - Configure the [\*\*<QTKNRT>\*\*](#) field and the [\*\*<QMTBS>\*\*](#) field in the Transmit Queue Token Bucket Configuration (TQxTBC) Register (n=0–7) ([Table 457 p. 594](#)).
  - Configure the [\*\*<QMTU>\*\*](#) field, the [\*\*<WRR\\_BC\[17:0\]>\*\*](#) field, and the [\*\*<WRRWGT>\*\*](#) field in the Transmit Queue Arbiter Configuration (TQxAC) Register (n=0–7) ([Table 458 p. 594](#)).
  - Configure the [\*\*<IPG>\*\*](#) field in the Transmission Queue IPG (TQxIPG) Register (n=2–3) ([Table 460 p. 595](#)). This field defines the minimum inter-packet gap, to be used by the EJP mechanism.
7. Configure the [\*\*<TS>\*\*](#) field in the Transmission Speed (TS) Register ([Table 464 p. 596](#)).
8. Configure the [\*\*<HiTKNinLoPkt>\*\*](#) field in the High Token in Low Packet (HITKNinLOPkt) Register ([Table 461 p. 595](#)). This field defines the number of tokens that are accumulated in the IsoHi (queue 3) token bucket, during transmission of the longest IsoLo (queue 2) packet.
9. Configure the [\*\*<HITKNinAsyncPkt>\*\*](#) field in the High Token in Asynchronous Packet (HITKNinASYNCPKT) Register ([Table 462 p. 595](#)). This field defines the number of tokens that are accumulated in the IsoHi (queue 3) token bucket, during transmission of the longest asynchronous (queues 1 and 0) packet.
10. Configure the [\*\*<LoTKNinAsyncPkt>\*\*](#) field in the Low Token in Asynchronous Packet (LOTKNinASYNCPKT) Register ([Table 463 p. 596](#)). This field defines the number of tokens that are accumulated in the IsoLo (queue 2) token bucket, during transmission of the longest asynchronous (queue 1 and 0) packet.

11. To enable the EJP operation, clear the <WRR\_EJP\_INIT> field in the Transmit Queue Command1 (TQC1) Register ([Table 453 p. 591](#)).
12. Enable all TX queues.

### 8.10.3 EJP Algorithm

The following pseudo-code describes the egress jitter pacing algorithm.



**Note**

In the following pseudo code, the addition of  $\#n$  at the end of a register field, means the algorithm use this field from queue number  $n$  (e.g., QTKNBKT#3 means the QTKNBKT field of queue number 3).

```

function egress_jitter_pacer
begin
if { (QTKNBKT#3 >= QMTU#3) //enough tokens have accumulated in the
bucket of Q#3
    and (Nr of sys clk cycle passed since last packet transmition from
Q#3 > TQxIPG#3 )}
    send out Packet from Q#3
else
    if { (QTKNBKT#2 >= QMTU#2) //enough tokens have accumulated in the
bucket of Q#2
        and (Nr of sys clk cycle passed since last packet transmition from
Q#2 > TQxIPG#2 )}
    )}

    if { (Q3 is not empty and (QTKNBKT#3 +HiTKNinLoPkt>= QMTU#3)
// Will next Q#3 packet expected transmition timing would be delayed by
sending the Q#2 packet?
        send out Packet from Q#3
    else
        send out Packet from Q#2
    else
        if (QTKNBKT#1 >= QMTU#1) or QTKNBKT#0 >= QMTU#0
            if { (Q3 is not empty and QTKNBKT#3 +HiTKNinAsyncPkt>= QMTU#3)
// Will next Q#3 packet expected transmition timing would be delayed by
sending the Q#0 or Q#1packet??
                send out Packet from Q#3
            else if { (Q2 is not empty and ((QTKNBKT#2 +LoTKNinAsyncPkt>= QMTU#2)))
// Will next Q#2 packet expected transmition timing would be delayed by
sending the Q#0 or Q#1packet?
                send out Packet from Q#2
            else
                send out Async Packet(Q nr 1 or 0 - according to strict priority or
WRR )

```

end

## 8.11

## Network Interface (10/100/1000 Mbps)

The device can be connected to a GbE network using a RGMII PHY, GMII PHY (88F619x/88F6281 only), or MII PHY.

If Auto-Negotiation is enabled for GMII or MII interface mode (by the [<AN\\_Duplex>](#) field and the [<AN\\_FC>](#) field in the Port Serial Control0 (PSC0) Register ([Table 427 p. 567](#)), the MDC/MDIO Auto-Negotiation takes place using MDC/MDIO pins, as defined in IEEE 802.3 standard.

To support all speeds, the device includes several MAC blocks suited for 10, 100, and 1000 Mbps with the following considerations:

- Support for half-duplex (for 10 and 100 Mbps only) and full duplex (in all speeds)
- Backpressure option in half duplex (for 10 and 100 Mbps only)
- Flow-control option in full-duplex

### 8.11.1

### MII Interface

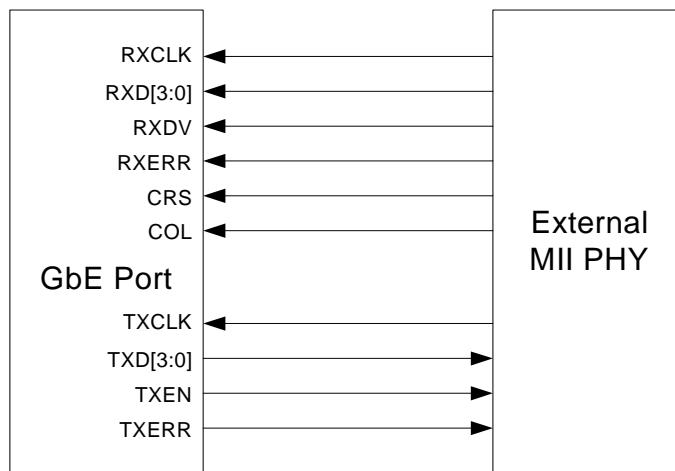
The device MAC allows it to be connected to a 10-Mbps or 100-Mbps network. The device interfaces with an IEEE 802.3 10/100 Mbps MII compatible PHY device. The data path consists of a separate nibble-wide stream for both transmit and receive activities.

These devices also support the Marvell® proprietary 200 Mbps MII (MMII) interface.

Depending on the speed of the network, the device can automatically switch between 10- or 100-Mbps operation. Data transfers are clocked by the 25-MHz transmit and receive clocks in 100-Mbps operation, or by 2.5-MHz transmit and receive clocks in 10-Mbps operation. The clock inputs are driven by the PHY. The PHY controls the clock rate based on its configuration, or on the Auto-Negotiation function.

In MII mode, the GbE port receives both RXCLK and TXCLK from the external PHY, as shown in [Figure 30](#).

**Figure 30: MII Connection**



In MII mode, the port operates at 10/100 Mbps (clock frequencies of 2.5/25 MHz respectively). In MMII mode, the port operates at 200 Mbps (clock frequency of 50 MHz).



**Note**

- Only four data bits (RXD[3:0], TXD[3:0]) are used.
- The port TXCLK\_OUT output is not used (left NC).

## 8.11.2

### GMII Interface (88F619x/88F6281)

The port Gigabit MAC supports the following:

- Connection to GMII PHY
- 1000 Mbps full duplex
- Standard IEEE 802.3 Flow Control in full duplex

The port MAC performs all of the functions of the IEEE 802.3 standard, such as frame formatting, frame stripping, collision handling, deferral to link traffic. The port ensures that any outgoing packet complies with the IEEE 802.3 specification in terms of preamble structure. The port transmits 56 preamble bits before the Start-of-Frame Delimiter.

The transmit and receive operations are done in full duplex and implement the standard.

When the port has a frame ready for transmission and the IPG counter has expired, the frame transmission begins. The data is transmitted via pins GE\_TXD[7:0] of the transmitting port and clocked on the rising edge of GE\_TXCLK\_OUT. At the same time, signal the GE\_TXEN is asserted. Since there is no carrier-extension required, the GE\_RXER signal is always driven LOW.



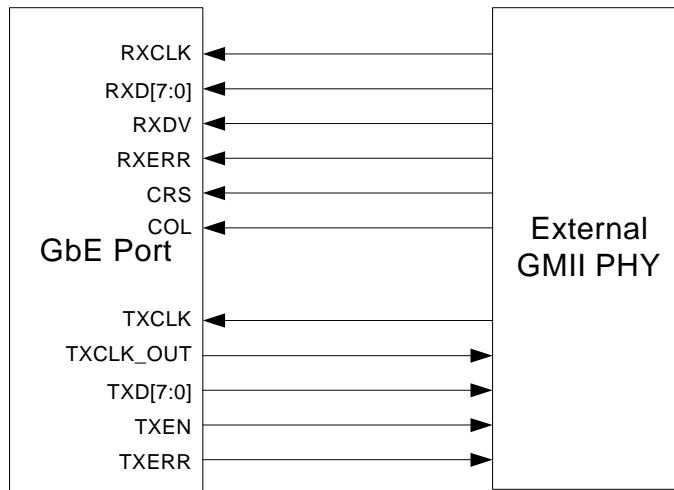
**Note**

The Carrier Sense (CRS) and Collision Detect (COL) input pins are ignored in this mode.

Frame reception starts with the PHY assertion of GE\_RXDV or GE\_RXER (while the port is not transmitting). Once GE\_RXDV or GE\_RXER are asserted, the port begins sampling incoming data on pins GE\_RXD[7:0], on the rising edge of the GE\_RXCLK.

The GE\_RXDV signal is high during reception of packet-data.

**Figure 31: GMII Connection**

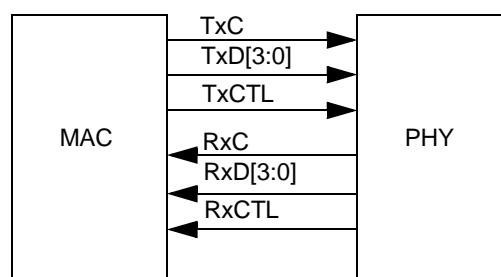


When the port is running 10/100 Mbps, TXCLK input from the external PHY is used as the reference for the transmit signals. When running at 1000 Mbps, TXCLK\_OUT is the reference clock. The TXCLK input is not used.

### 8.11.3 RGMII Interface

The RGMII specification reduces the number of pins required to interconnect the MAC and the PHY to 12 pins, in a cost effective and technology-independent manner. To accomplish this objective, the data paths and all associated control signals are reduced, control signals are multiplexed together, and both edges of the clock are used (see [Figure 32](#)). For Gigabit operation, the clocks operate at 125 MHz. For 10/100 operation, the clocks operate at 2.5 MHz or 25 MHz, respectively. The transmit and receive operations are done in full duplex and implement the standard.

**Figure 32: RGMII Pin Interconnection Between MAC and PHY**



The RGMII interface uses a 125 MHz DDR clock with 4-bits wide data path. All signals shall be conveyed with positive logic, except where explicitly defined differently. For descriptive purposes, a signal shall be at a logic "high" when it is at a valid voltage level greater than VOH\_MIN, and logic "low" when it is at a valid voltage level less than VOL\_MAX.

### 8.11.3.1

#### RGMII 10/100 Mbps Functionality—Modified MII

This interface can be used to implement 10/100 Mbps Ethernet MII by reducing the clock rate to 25 MHz for 100 Mbps operation, and 2.5 MHz for 10 Mbps. The MAC always generates the SYS\_CLK signal and the PHY always generates the Px\_RXCLK signal.

During packet reception, Px\_RXCLK can be stretched on either the positive or negative pulse to accommodate the transition from the free running clock to a data synchronous clock domain. When the speed of the PHY changes, a similar stretching of the positive or negative pulse is allowed. No glitching of the clocks is allowed during speed transitions.

The MAC must hold Px\_TXCTL (TX\_CTL) low until the MAC has ensured that Px\_TXCTL (TX\_CTL) is operating at the same speed as the PHY.

### 8.11.3.2

#### Signals Encoding

The RGMII interface is basically a GMII interface running at double data rate:

- GMII\_TXD[3:0] is driven on RGMII\_TXD[3:0] on the rising edge of RGMII\_TXCLKOUT; GMII\_TXD[7:4] is driven on RGMII\_TXD[3:0] on the falling edge of RGMII\_TXCLKOUT.
- GMII\_TXEN is driven on RGMII\_TXCTL on the rising edge of RGMII\_TXCLKOUT
- A logical value of GMII\_TXEN XOR GMII\_TXERR is driven on RGMII\_TXCTL on the falling edge of RGMII\_TXCLKOUT.
- GMII\_RXD[3:0] is sampled on RGMII\_RXD[3:0] on the rising edge of RGMII\_RXCLK; GMII\_RXD[7:4] is sampled on RGMII\_RXD[3:0] on the falling edge of RGMII\_RXCLK.
- GMII\_RXDV is sampled on RGMII\_RXCTL on the rising edge of RGMII\_RXCLK.
- A logical value of GMII\_RXDV XOR GMII\_RXERR is sampled on RGMII\_RXCTL on the falling edge of RGMII\_RXCLK.

### 8.11.3.3

#### In-Band Status

To ease detection of the link status, speed, and duplex mode of the PHY, inter-frame signals are placed onto the Px\_RXD[3:0] signals. CRS is indicated when, simultaneously, RX\_DV = True or RX\_ER = False, RX\_ER = True, and a value of FF binary exists on the Px\_RXD[3:0] bits.

Collision is determined at the MAC when TX\_EN = True, while either Px\_CRS or P0\_RXDV are true. The PHY does not assert Px\_CRS as a result of Px\_TXEN being true.

## 8.12

### Auto-Negotiation

#### 8.12.1

#### Auto-Negotiation in MII/GMII/RGMII Modes

The device implements the standard IEEE Auto-Negotiation, using the Serial Management Interface (SMI), for the following:

- Detect Link status
- Duplex: half- and full-duplex operation
- Flow-control for full-duplex
- Speed

To implement speed Auto-Negotiation, set the <ANSpeed> field in the Port Serial Control0 (PSC0) Register ([Table 427 p. 568](#)) to 0.

To switch between GMII and MII modes, see Port Serial Control1 (PSC1) Register ([Table 430 p. 571](#)).



The registers and bits referred to in this section (for example, registers 4, 5, and 15 and bit 1.8) are PHY device registers.

The device continuously reads the PHY register 1 to determine the link status, and also to determine whether or not bit 1.8 in PHY register 1 is set.

When exiting from reset, or when the link changes from up to down, the device advertises its flow control ability (if Auto-Negotiation for flow control is enabled by the [AN\\_FC](#) field in the Port Serial Control0 (PSC0) Register ([Table 427 p. 567](#))).

The device reads register bit 1.8. If this bit is reset, then the PHY does not support 1000 Mbps.

If bit 1.8 is set, the PHY supports 1000 Mbps (but the speed may still resolve to 10 or 100 Mbps at the end). The device then continues to read register 15 to determine whether the PHY is 1000baseX-capable or 1000baseT capable. If it is 1000baseX, then the device regards the multiplexed speed as 1000 Mbps only, and follows the IEEE 802.3 rules for duplex and flow-control Auto-Negotiation. If it is 1000baseT capable, then the device follows the IEEE 802.3 rules to resolve the speed, the duplex mode and the flow control.

After Auto-Negotiation is complete, the device resolves negotiated modes of operation. These values update the Port Status register fields and affect the Network port operation.

## 8.13 Data Blinder

The port data blinder is the time period in which the port does not look at the wire to determine if it is necessary to defer a pending transmission, due to receive activity.

The port data blinder is 32 bit time.

## 8.14 Inter-packet Gap

The Inter-packet Gap (IPG) is the idle time between two successive packets from the same port. The default (from the standard) is 96 ns.



Marvell does not recommend reducing the IPG setting in violation of the IEEE standards. Reducing the IPG can improve test scores but can create Ethernet compatibility problems.

Use the MII Serial Parameters Register ([Table 418 p. 562](#)) to set the IPG size.

## 8.15 Illegal Frames

For undersized frames (with or without good CRC), the port discards all illegal frames. These frames are not passed to the CPU, regardless of address filtering, and the appropriate error MIB counters are incremented. An undersized frame is determined by the Minimum Frame Size.

Oversized frames (greater than the Maximum Receive Unit—MRU) with or without bad CRC (bad checksum) are forwarded to the DMA queue with an error summary report in the Rx descriptor.



**Note**

An oversized packet is chopped according to the MRU setting, and only MRU bytes are transferred to memory. If the Marvell Header mode, DSA Tag, or Extended DSA Tag is used, an oversized packet is calculated based on MRU+2, MRU+4, or MRU+8 bytes respectively.

---

## 8.16

## Backpressure Mode

Back pressure is supported only when working in 10/100 Mbps speed and in half duplex mode.

The Backpressure algorithm is enabled by setting the [`<ForceBPMMode>`](#) field in the Port Serial Control0 (PSC0) Register ([Table 427 p. 567](#)).

For a port in Backpressure mode, the port waits until the medium is idle and then transmits a JAM pattern of 1536 bytes. The IPG between two consecutive JAM patterns is 4 bytes, and between last transmitted packet to first JAM is 12 bytes.

When a port in Backpressure mode has a pending packet for transmission, it halts the transmission of the JAM pattern. After an IPG is completed, the port transmits the packet. If the port remains in Backpressure mode, it resumes the JAM pattern transmission, following the packet transmission.

## 8.17

## Flow Control

The device implements the IEEE 802.3 flow control in full-duplex mode, including full Auto-Negotiation.

Auto-Negotiation for flow control is enabled for:

- The multiplexed interface
- PHYs that have SMI interface (MII or GMII PHYs)

The behavior of the device is determined by the value in Port Status register `<En_Fc>` bit, Port Serial Control register (PSCR) `<Force_FC_Mode>` bit.

The CPU may write to the PSCR `<Force_FC_Mode>` bit when Flow Control operation is enabled in Port Status register `<En_Fc>` bit (which may be result either of Auto-Negotiation resolution for flow control, or manual setting by the CPU to enable flow-control operation, which is then reflected in the Port Status register `<En_Fc>` bit).

The CPU must trigger the initiation of pause disable transmission when detecting that it cannot keep up with the received traffic (this is typically done by monitoring the queue filling process).

When the CPU suspects that it may not be able to provide enough resources to the port, it must trigger the beginning of flow control packets by writing 01 value to `<Force_FC_Mode>`. When resources are made available, the CPU must again write a 00 value to `<Force_FC_Mode>` to trigger transmission of pause enable packet (see the more detailed description in [Section 8.17.2, Pause Transmit Operation, on page 156](#)). The CPU response time to congestion cases would determine if and how many packets may be lost on receive.

The value in Port Status register `<En_Fc>` bit can be set from the following:

- CPU programming
- Result of Auto-Negotiation for Flow Control according to IEEE 802.3 standard in all modes

When in MII or GMII modes, and Auto-Negotiation for Flow Control is enabled, the device writes to the relevant advertisement register in the PHY on the following events:

- Exiting from reset
- Upon link fail detection (link changed from up to down)

Auto-Negotiation for Flow Control for 1000BASE-X PHY advertises that the device supports Symmetric Flow Control only according to the IEEE 802.3 standard.

The advertised ability of Pause support depends on the setting of the [\*\*<Pause\\_Adv>\*\*](#) field in the Port Serial Control0 (PSC0) Register ([Table 427 p. 567](#)) as follows:

- When set, the device advertises symmetric capability for Pause.
- When reset, the device advertises No Pause capability.

### 8.17.1 Pause Receive Operation

When the device receives a Pause packet, it avoids transmitting a new packet for the period of time specified in the received Pause packet.

The pause quantum is 512 bits time according to the port speed.

A received packet is recognized as flow control if it was received without errors and it has the following characteristics:

DA = 01-80-C2-00-00-01 and type=88-08 and MAC\_Control\_Opcode=01.

A packet received by the device from the network port that is identified as a Pause packet is always discarded, even if the Pause function is disabled.

### 8.17.2 Pause Transmit Operation

For enabling a Pause Transmit operation, or either enabling or disabling, the [\*\*<EnFC>\*\*](#) field in the Ethernet Port Status 0 (PS0) Register ([Table 429 p. 570](#)) must be in the active state.

It is the CPU's responsibility to determine that packets are in danger of being dropped by the receive port, according to the dynamic availability of resources. One way of doing it is monitoring how much of the descriptor chain is filled up by the port and how much is left. Another aspect is memory bandwidth should be allocated to the port via the Mbus "pizza arbiter" to avoid bandwidth shortage for the gigabit port.



**Note** This mechanism does not provide hardware guarantee of zero frame-loss. This mechanism depends on CPU functionality in triggering it dynamically.

When the CPU suspects that it may not be able to provide enough resources to the port, it must trigger the beginning of Flow Control, pause-disable packets transmission by writing a 01 value to the [\*\*<ForceFCMode>\*\*](#) field in the Port Serial Control0 (PSC0) Register ([Table 427 p. 567](#)). The transmit port will schedule transmission of a pause-disable frame (timer=0xFFFF) at the next possible frame boundary and will automatically retransmit it at least every 4.2 msec (GMII/RGMII), 42 msec (MII at 100 MB), or 420 msec (MII at 10 MB) as long as the value in the [\*\*<ForceFCMode>\*\*](#) field remains 01.

The other link partner is expected to stop packet transmission upon receiving the Flow Control disable packets, and the retransmission of them guarantees refreshing that indication continuously.

When resources are made available, the CPU must write a 00 value to [\*\*<ForceFCMode>\*\*](#). This will trigger transmission of a single pause enable packet (timer = 0x0000) that would enable the other link partner to resume packet transmission.

---

When transmitting a pause packet, the port address is put into the source address field. The 48-bit port address is located in the MAC Address Low and the MAC Address High registers.



**Note**

When the link goes down, the [ForceFCMode](#) is always reset to 00 (no pause disable frames are sent).

## 8.18

## Serial Management Interface (SMI)

The port MAC contains a Serial Management Interface (SMI) for interfacing with two GbE PHYs.

The SMI allows control and status parameters to be passed between the device and the PHY (parameters specified by the CPU) using one serial pin (MDIO) and a clocking pin (MDC), reducing the number of control pins required for PHY mode control. Typically, the device continuously queries the PHY device for the link status, without CPU intervention. The PHY addresses for the link query are programmable in the PHY Address Register ([Table 401 p. 554](#)).

A CPU connected to the device can write/read to/from all PHY addresses/registers. The SMI allows the CPU to have direct control over an MII or GMII compatible PHY device via the SMI Register ([Table 402 p. 554](#)). This control allows the driver software to place the PHY in specific modes such as Full-Duplex, Power-Down, or 1000-speed selection. It also helps control the PHY device's Auto-Negotiation function, if it exists. The CPU writes commands to the SMI register and the device reads or writes control/status parameters to the PHY device via a serial, bi-directional data pin called MDIO. These serial data transfers are clocked by the device MDC clock output.

### 8.18.1

### SMI Cycles

The SMI protocol consists of a bit stream that is driven or sampled by the device on each rising edge of the MDC clock. The SMI frame, bit-stream format starts with PRE and ends with IDLE. Its various steps are described in [Table 47](#).

**Table 47: SMI Bit Stream Format**

	<b>PRE</b>	<b>ST</b>	<b>OP</b>	<b>PhyAd</b>	<b>RegAd</b>	<b>TA</b>	<b>Data</b>	<b>IDLE</b>
READ	1...1	01	10	AAAAA	RRRRR	Z0	D.D(16)	Z
WRITE	1...1	01	01	AAAAA	RRRRR	10	D.D(16)	Z

- PRE (Preamble): At the beginning of each transaction, the device sends a sequence of 32 contiguous logic 1 bits on the MDIO with 32 corresponding cycles on the MDC to provide the PHY with a pattern that it can use to establish synchronization.
- ST (Start of Frame): A Start-of-Frame pattern of 01.
- OP (Operation Code): 10 - Read; 01 - Write.
- PhyAd (PHY Address): A 5-bit address of the PHY device (32 possible addresses). The first PHY Address bit transmitted by the device is the MSB of the address.
- RegAd (Register Address): A 5-bit address of the PHY register (32 possible registers in the PHY). The first register address bit transmitted by the device is the MSB of the address. The device always queries the PHY device for status of the link by reading register 1, bit 2.
- TA (Turn Around): The turnaround time is a 2-bit time spacing between the <RegAd> field and the <Data> field of the SMI frame to avoid contention during a read transaction. During a read transaction the PHY must not drive MDIO in the first bit time and drive 0 in the second bit time. During a write transaction, the device drives a '10' pattern to fill the TA time.

- Data (Data): The data field is 16-bits long. The PHY drives the data field during read transactions. The device drives the data field during write transactions. The first data bit transmitted and received is bit 15 of the PHY register being addressed.
- IDLE (Idle): The IDLE condition on MDIO is a high impedance state. The MDIO driver is disabled and the PHY must pull-up the MDIO line to a logic 1.

## 8.19

## Link Detection and Link Detection Bypass (ForceLinkPass\*)

Typically, the device continuously queries the PHY device for its link status, without CPU intervention. The PHY address used for the link query is determined by the PHY Addresses register, and it is programmable, where the default value is 8 for Port0, 9 for Port1. The device reads register 1 from PHY and updates the internal link bits according to the value of bit 2 of register 1. In the case of “link is down” (bit 2 is 0), that port enters link test fail state. In this state, all of the port’s logic is reset. The port exits from link test fail state only when the “link is up”, bit 2 of register 1 is read from the port’s PHY as 1.

The device offers the option to disable the link detection mechanism by forcing the link state of the interface to the link test pass state. This is done by forcing the register bit, and then the link status of the port remains in the “link is up” state regardless of the Interface-PHY’s link bit value. The link status of the Interface-PHY can be read through the SMI from the PHY devices (register 1, bit 2).

### 8.19.1

#### Force\_Link\_Fail

The PSCR Register <Force\_Link\_Fail> bit (bit 10) has the default value of forcing the link detection on each port to link down. The user *must* set this bit, in order to get the true link status of the port, and in order to enable the port link indication to go up.

The user must not program the <Force\_Link\_Fail> bit and the <Force\_Link\_Pass> bit to be set at the same time.

## 8.20

## Precise Time Protocol (PTP)

This section summarizes the functionality for Precise Time Protocol (PTP) Core hardware.

The Precise Timing Protocol defines a method to transport time-of-day across a network of devices supporting this protocol. There are several industrial, consumer, and enterprise applications for PTP. For example, in consumer space, IEEE 802.1 AVB (Audio Video Bridging) defines the usage of PTP for transporting reliable audio video content across compliant equipment. PTP adds time information to nodes connected via Ethernet. This is in contrast to IEEE 802 standard Ethernet that is an asynchronous interface where end stations do not operate using a common time base and neither do they have a concept of time.

The PTP allows multiple nodes within a given network to have a common notion of the time-of-day and also to be able to compute frequency offsets with respect to a master clock in the network.

There are three types of PTP frames that are supported in the device:

1. IEEE 802.1AS frames on Layer 2 Ethernet
2. IEEE 1588 frames on Layer 2 Ethernet
3. IEEE 1588 frames on Layer 4 UDP

For IEEE 802.1AS and IEEE 1588 over Layer 2, the detection mechanism uses a special EtherType. For IEEE 1588 over Layer 4 UDP, the detection mechanism is by checking the UDP destination port value. Once a PTP frame has been detected, event messages need to be time stamped in hardware. However, all PTP frames need to be forwarded to the CPU port for further processing. Once the ingressing PTP frames have been processed by the CPU, certain types of PTP frames get forwarded to other ports that again need to be detected and time stamped in hardware.

The VLAN tag of the incoming PTP frame is detected by hardware.

Using a clock selection algorithm, the PTP firmware computes the frequency and phase offset values, with respect to the best clock available in the network (labeled as the *grand master*). The computed offsets can optionally be used to control the hardware clock and time-of-day counter.

The PTP message types of frames that required time stamping by the hardware is configurable.

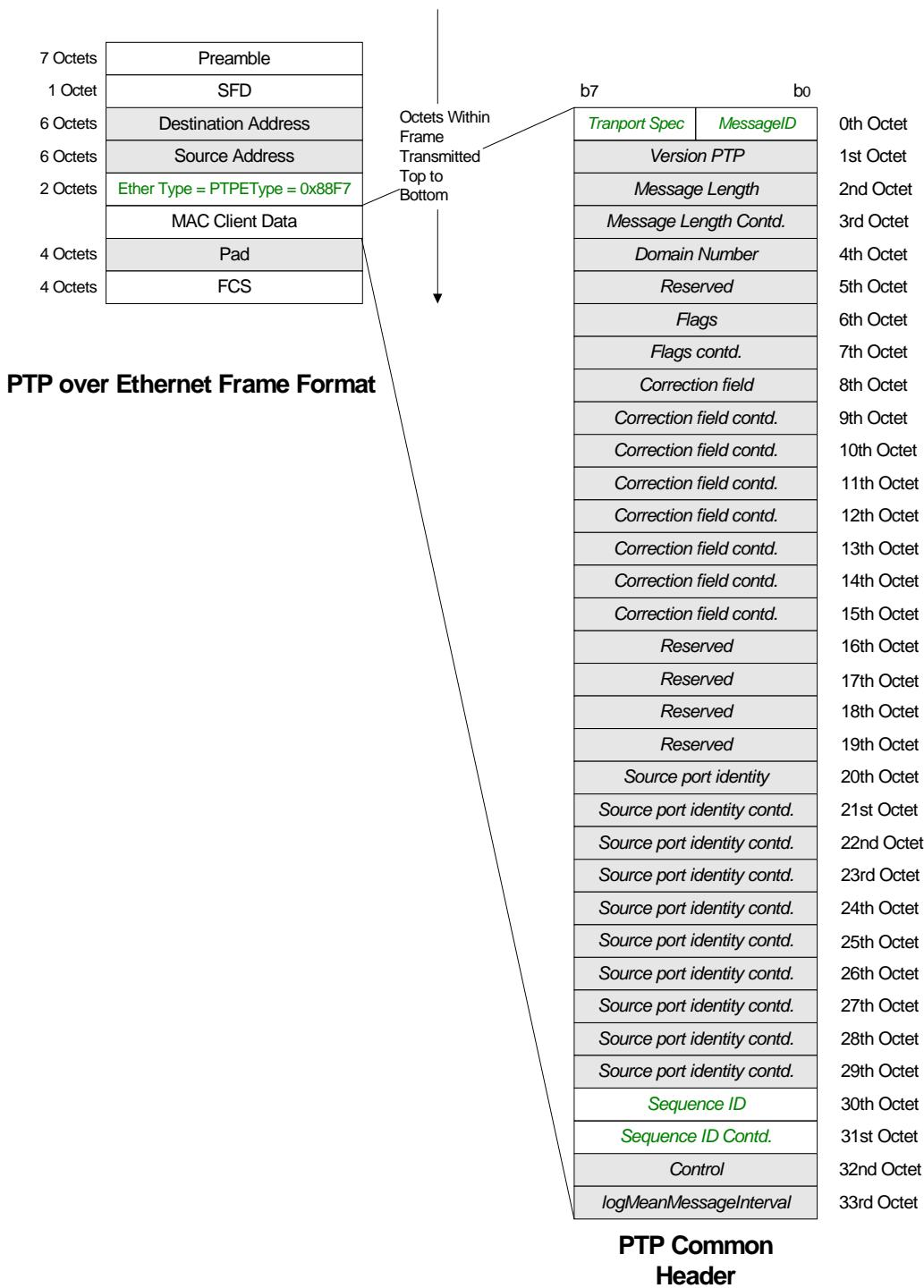
For every time-stamped frame, the sequence number from the PTP common header is captured along with the time stamp, in hardware for easier protocol software correlation.

IPv4/IPv6 are supported, however, IPv4/IPv6 options are not supported.

## 8.20.1 Frame Format

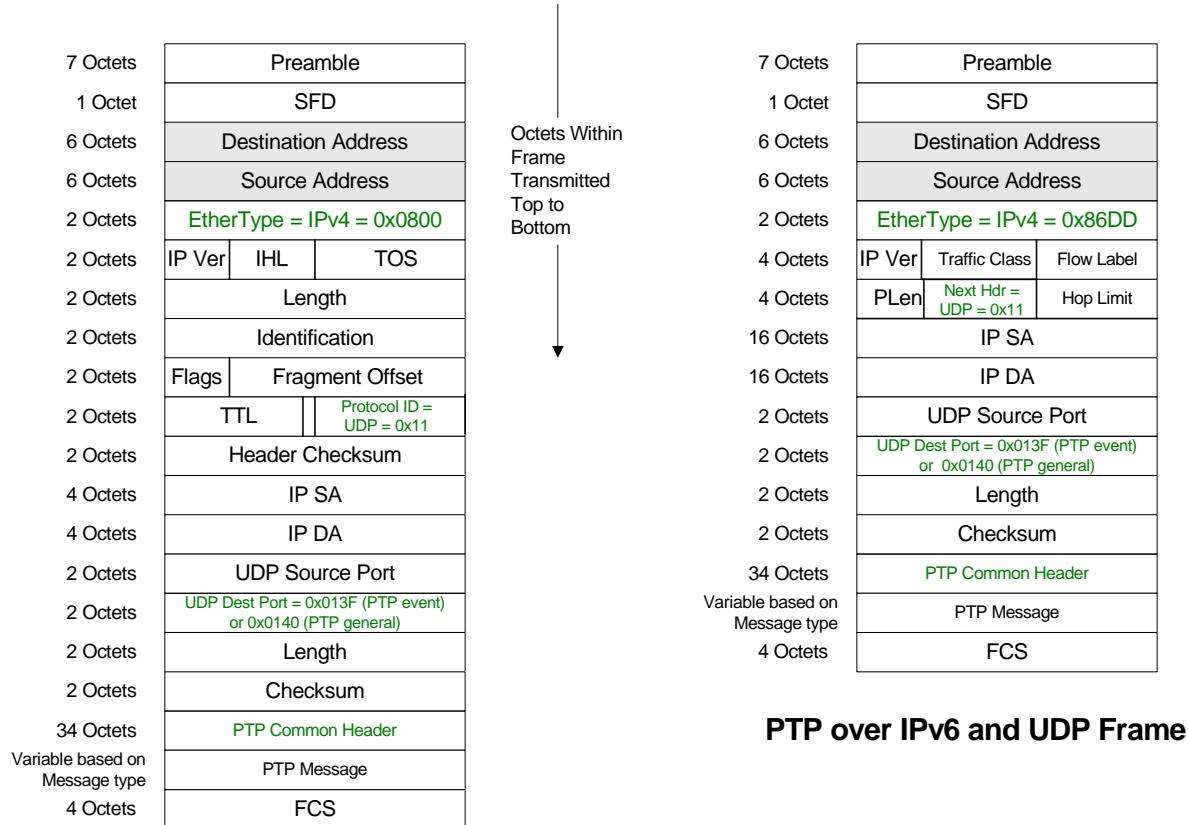
The PTP frame format is shown in [Figure 33](#).

**Figure 33: PTP Common Header Format**



The PTP over UDP frame format, for both IPv4 and IPv6, is shown in [Figure 34](#).

**Figure 34: PTP over UDP Frame**



### PTP over IPv4 and UDP Frame

#### 8.20.2 PTP Frame Time Stamping Clock

The PTP core logic receives a reference clock input (from the PTP time stamp clock) for time stamping purposes. This clock can be a free running clock available in the chip or it can be a PTP synchronized clock (PTP\_CLK), as defined by the `<PTP_SEL>` field in the PTP Clock Configuration Register ([Table 814 p. 784](#)) and the `<PTPClkSelect>` field in the PTP Clock Select Register ([Table 469 p. 600](#)). The benefits of synchronization are more prominent for endpoint devices than for standard AVB bridges. The definition of synchronized clock is when a free running clock's frequency is adjusted based on the frequency offset computed by the PTP software (this is the frequency difference between the PTP Grand Master node and the PTP slave node).

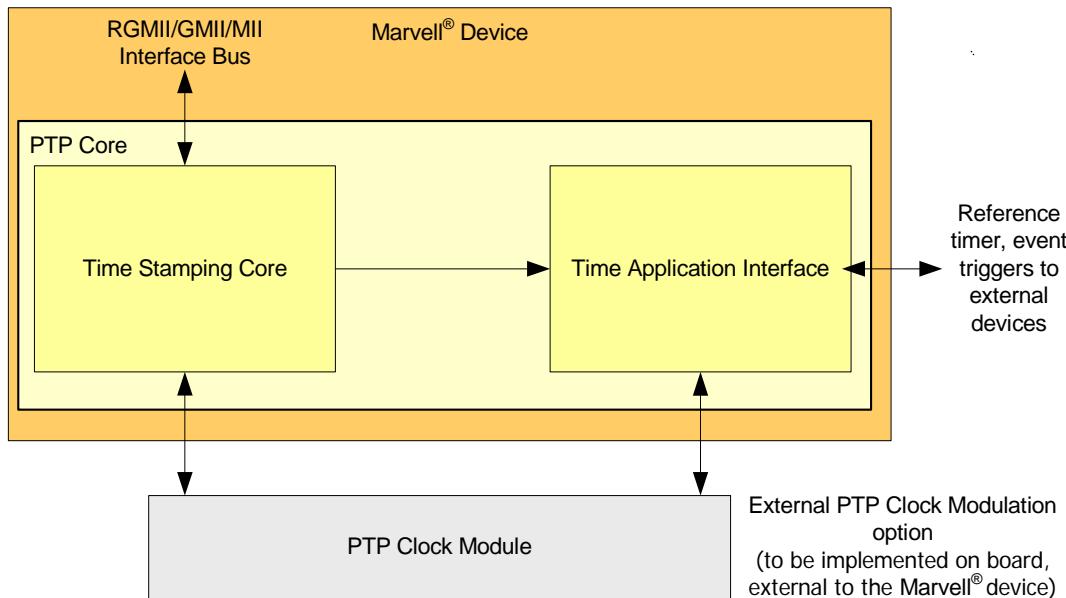
The clock frequency and/or phase adjustment is expected to be handled in an external device.

#### 8.20.3 Pipeline Stages

The PTP core is divided into two main blocks (see [Figure 35](#)):

- Time Stamping Core -see [Section 8.20.4](#)
- Time Application Interface—see [Section 8.20.5](#)

**Figure 35: PTP 2.1 Pipe Block Diagram**



**Note** The PTP Clock Module, which is a DSP circuit to adjust the frequency and phase offset of the clock with respect to the grand master clock, is not part of this Marvell® device.

#### 8.20.4

#### Time Stamping Block

The time stamping block is a plug-in module snooping the MII/GMII/RGMII interface between the MAC (within the device) and the PHY (external to the Marvell device) as shown in the [Figure 36](#). All incoming frames go through the PTP Frame Detection flow to determine if the frame is a PTP frame or not. Once the frame is determined to be a PTP frame, fields from the PTP common header (shown in [Figure 33, PTP Common Header Format, on page 160](#)) are extracted to determine if the PTP frame needs time stamping or not.

The PTP logic captures the time stamp for every incoming frame. Once the frame has been determined to be a PTP event messages (which need time stamping), the time stamp along with SequenceID (from the PTP common header) is captured in registers and optionally an interrupt is generated by hardware.

- For incoming frames on Arrival0, this capture takes place in the following registers:
  - [PTP Port Status0 Register](#)
  - [PTP Port Status1 Register](#)
  - [PTP Port Status2 Register](#)
- For incoming frames on Arrival1, this capture takes place in the following registers:
  - [PTP Port Status3 Register](#)
  - [PTP Port Status4 Register](#)
  - [PTP Port Status5 Register](#)

- For all outgoing frames, this capture takes place in the following registers:
  - PTP Port Status6 Register
  - PTP Port Status7 Register
  - PTP Port Status8 Register

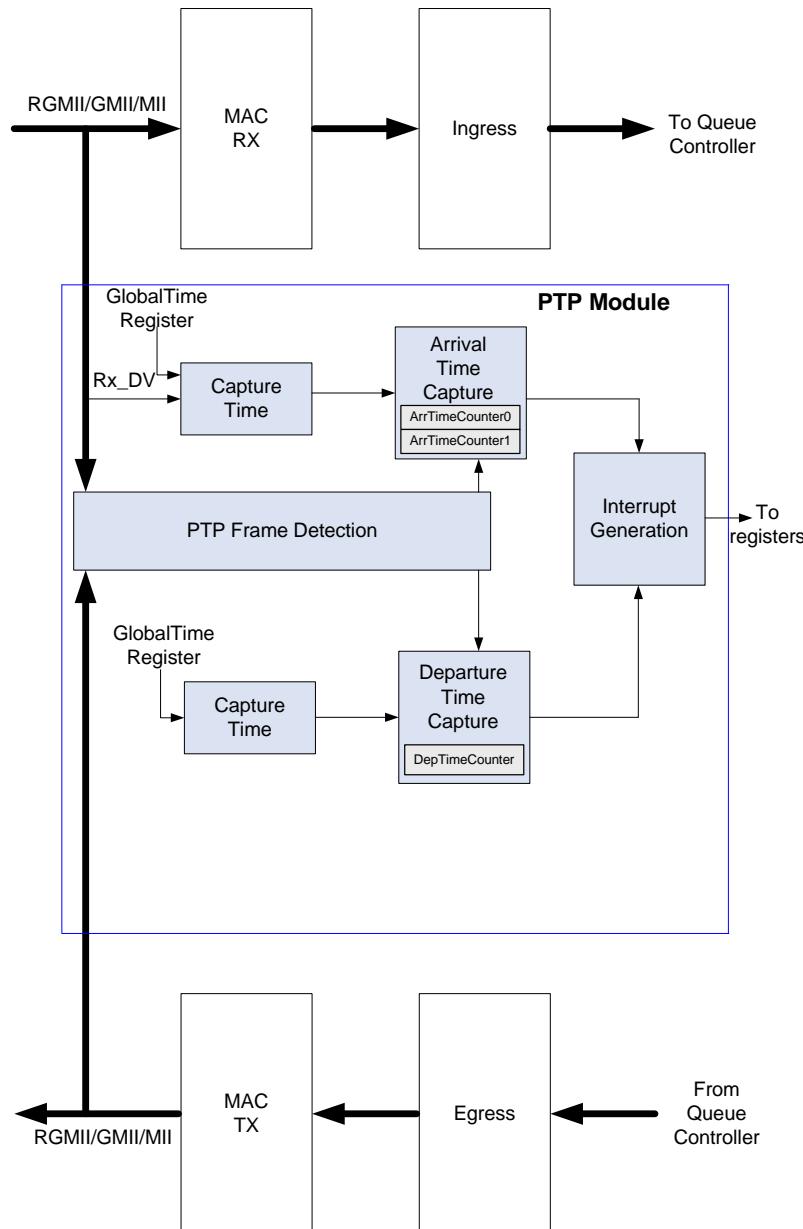
PTP frames go through normal MAC SA and DA processing and, since it is expected that a reserved MAC address is used in PTP frames, the MAC address lookup would result in a CPU destination port. There is no change in the logic for sending these frames to the CPU.

### Rationale for Two Arrival Counters and One Departure Counter

As stated earlier, there are two categories of event messages that need to be time stamped in hardware:

- Sync/Follow up
- PDelay\_Req/PDelay\_Response

The Sync/Follow up messages flow directly from the Grand Master to all the PTP slave nodes and PDelayReq/PDelayResponse messages are exchanged between link partners. The timing between these two sets of event messages is not controllable. Thus, the hardware logic needs to be designed for back-to-back PTP event messages that need time stamping. Each of these event message types can be configured to be captured in either Arrival Counter 0 or Arrival Counter 1, using the configuration parameter <TSArrPtr> field in the PTP Global Configuration2 Register ([Table 472 p. 602](#)).

**Figure 36: Time Stamping Pipeline Stages**


### 8.20.5

### Timing Application Interface (TAI) Block

The Precise Timing Protocol provides both frequency and time-of-day with respect to the PTP Grand Master for the entire PTP network. In a given endpoint device (media talker or a media listener), once the PTP Grand Master aware clock and time are available, this data must be transmitted over to the rest of the end user application, without loss of accuracy. For example, if a Digital Video Recorder (DVR) is the end device, the network clock and time need to be transported to the Video device and/or Storage device and the host processor seamlessly. This ensures that when the media is played out of the DVR, the presentation time of the content is required to be carried through the network between the media talker and the media listener.

There are three types of interfaces that this block supports:

- An event capture interface that captures the time of an event signaled by the requesting entity.
- A trigger-generate interface that causes an event to be signaled at a time specified by the requesting entity.
- A trigger-clock-generator interface that causes a periodic sequence of indications to be generated, with a rate specified by the requestor.

#### 8.20.5.1

#### Precise Timing Protocol—Time Application Interface (TAI)

The TAI Timing Interface Block supports features required for the above purpose. This block utilizes two MPP signals to offer various services:

- PTP\_EVENT\_REQ input signal (event request)
- PTP\_TRIG\_GEN output signal (trigger generate)

Using these signals, there are several functions that this block supports:

- Event pulse capture
- Multiple-event counter
- Trigger pulse generate, with pulse width control
- Trigger clock generate, with digital clock compensation
- PTP global time increment and/or decrement

#### 8.20.5.2

#### Event Capture Interface

When a PTP\_EVENT\_REQ signal is asserted, a snap shot of the PTP Global Timer is taken and stored in the <EventCap Register> field in the TAI Global Status2 Register ([Table 497 p. 621](#)). There is an option to overwrite the previous event time capture register by setting the <EventCapOv> field in the TAI Global Configuration, PTP Port = 0xE ([Table 488 p. 615](#)).



**Note**

If the hardware registers are being accessed by the CPU at the same time the hardware logic is trying to update an <EventCap Register> field in the TAI Global Status2 Register ([Table 497 p. 621](#)) field, an <EventCapErr> field in the TAI Global Status1 Register ([Table 496 p. 621](#)) is set to 0x1.

#### Event Pulse Capture Interface

In many IEEE 1588 applications like industrial automation, it is important to precisely capture the time at which a particular event occurred. The event is defined by a low-to-high transition on an external input signal called PTP\_EVENT\_REQ. The event time is captured in <EventCap Register>. This field is validated by <EventCapValid> field in the TAI Global Status1 Register ([Table 496 p. 621](#)).

The captured event time register, TAI Global Status1 Register ([Table 496 p. 621](#)), must be read by software and the <EventCapValid> field must be cleared before the hardware captures another event. If two back-to-back events occurred before the software read the results of the first event, an error indication is set in <EventCapErr>. To have the hardware overwrite the Event capture register rather than generate an interrupt, then set <EventCapOv> field in the TAI Global Configuration, PTP Port = 0xE ([Table 488 p. 615](#)).

Once an event has been captured, the software can generate an interrupt, if both the <EventCapIntEn> field in the TAI Global Configuration, PTP Port = 0xE ([Table 488 p. 616](#)) and the <EventInt> field in the TAI Global Status1 Register ([Table 496 p. 621](#)) are set.

The maximum jitter associated with capturing the PTP\_EVENT\_REQ signal pulse is the value set in the <TSClkPer> field in the TAI Global Configuration Register, PTP Port = 0xE and 0xF ([Table 489 p. 617](#)). The minimum pulse width of the PTP\_EVENT\_REQ signal must be 1.5 times the value set

in the [\*\*<TSClkPer>\*\*](#). For the hardware logic to detect distinct events on the PTP\_EVENT\_REQ signal, the minimum gap between two events needs to be 125 ns.

### Multiple Event Counter Function

Similar to the Event Pulse capture interface described in [Event Pulse Capture Interface](#), if multiple events need to be captured, for an application to detect how many times a particular event occurred on the PTP\_EVENT\_REQ input signal, the [\*\*<EventCtrStart>\*\*](#) field in the TAI Global Configuration, PTP Port = 0xE ([Table 488 p. 616](#)) must be set to a 0x1 and [\*\*<EventCapOv>\*\*](#) field in the TAI Global Configuration, PTP Port = 0xE ([Table 488 p. 615](#)) must set to a 0x1.

The current PTP core is capable of capturing up to 255 events in the [\*\*<EventCapCtr>\*\*](#) field in the TAI Global Status1 Register ([Table 496 p. 621](#)).

The maximum jitter associated with capturing the PTP\_EVENT\_REQ signal pulse is the value set in the [\*\*<TSClkPer>\*\*](#) field in the TAI Global Configuration Register, PTP Port = 0xE and 0xF ([Table 489 p. 617](#)). The minimum pulse width of the PTP\_EVENT\_REQ signal must be 1.5 times the value set in the [\*\*<TSClkPer>\*\*](#). For the hardware logic to detect distinct events on the PTP\_EVENT\_REQ signal, the minimum gap between two events needs to be 125 ns.



In the multiple event counter mode, the EventCapRegister (TAI Global Status, Offset 0xA & 0xB) indicate the time stamp value for the last captured event register.

**Note**

### Trigger Pulse Generate Function

In many PTP applications, the time of day computed in PTP needs to be distributed in some form to the rest of the node. One commonly used method generating a pulse whenever the PTP Global Time matches a certain configured value. The pulse gets output on a PTP\_TRIG\_GEN output signal.

In the PTP core, configure the Trigger Pulse Generate function:

1. Set the [\*\*<TrigGenReq>\*\*](#) field in the TAI Global Configuration, PTP Port = 0xE ([Table 488 p. 617](#)) to 0x1.
2. Set the [\*\*<TrigMode>\*\*](#) field in the TAI Global Configuration, PTP Port = 0xE ([Table 488 p. 617](#)) to 0x1.
3. Configuring the [\*\*<TrigGenAmt>\*\*](#) field in the TAI Global Configuration0 Register ([Table 490 p. 618](#)) with the length of time amount after which the pulse needs to be generated.

The PTP Global Timer gets compared to the value in the [\*\*<TrigGenAmt>\*\*](#) field, and upon a match, a pulse signal is generated on the PTP\_TRIG\_GEN output signal.

Optionally, after generating the PTP\_TRIG\_GEN pulse, the CPU can be notified by setting the [\*\*<TrigGenIntEn>\*\*](#) field in the TAI Global Configuration, PTP Port = 0xE ([Table 488 p. 616](#)), and along with the pulse output, the [\*\*<TrigGenInt>\*\*](#) field in the TAI Global Status0 Register ([Table 495 p. 620](#)) is set. Upon receiving the interrupt, the CPU must clear the interrupt bit.

In Pulse mode The pulse width of the output signal can be controlled by the [\*\*<PulseWidth>\*\*](#) field in the TAI Global Configuration2 Register ([Table 492 p. 618](#)).



Do not set the [\*\*<PulseWidth>\*\*](#) to 0x0.

**Note**

### Trigger Clock Generate Function

Similar to the trigger pulse generation function described [Trigger Pulse Generate Function](#), the same set of registers can be used to generate a periodic clock. The value specified in [`<TrigGenAmt>`](#) field is used to generate the base period of the clock output. For this functional mode, the [`<TrigMode>`](#) field in the TAI Global Configuration, PTP Port = 0xE ([Table 488 p. 617](#)) field must be set to a 0x0, and the [`<TrigGenReq>`](#) must be set to a 0x1.

The output clock can be compensated by configuring the field [`<TrigClkComp>`](#) field in the TAI Global Configuration1 Register ([Table 491 p. 618](#)). This field specifies the remainder amount for the clock that is being generated with the period specified by the [`<TrigGenAmt>`](#) field. The [`<TrigClkComp>`](#) amount is constantly accumulated. When this accumulated amount exceeds the value specified in [`<TSClkPer>`](#) field in the TAI Global Configuration Register, PTP Port = 0xE and 0xF ([Table 489 p. 617](#)), the [`<TSClkPer>`](#) value is added to the output clock momentarily to compensate for the remainder accumulated overtime.



The [`<TrigGenAmt>`](#) field should be set to no less than two times the [`<TSClkPer>`](#) value.

**Note**

### PTP Global Time Increment/Decrement Function

The 32-bit nanosecond segment of the Time of Day counter for the PTP is stored in hardware and the 64-bit seconds segment is stored in software. Since every PTP slave node constantly computes the time of day, it needs to adjust its hardware with the updated phase offset information. The PTP core assists by providing a PTP Global Timer adjustment functions.

For an increment operation:

- Set the [`<TimeIncDecEn>`](#) field in the TAI Global Configuration, PTP Port = 0xE ([Table 488 p. 616](#)) to 0x1.
- Set [`<TimeIncDecOp>`](#) field in the TAI Global Configuration2 Register ([Table 492 p. 619](#)) to 0x0.
- Configure the [`<TimeIncDecAmt>`](#) field to the value that needs to be added to the current PTP Global Time value.

For a decrement operation:

- Set the [`<TimeIncDecEn>`](#) field to 0x1.
- Set [`<TimeIncDecOp>`](#) field to 0x1.
- Configure the [`<TimeIncDecAmt>`](#) field to the value that needs to be subtracted from the current PTP Global Time value.

Given that the [`<TimeIncDecAmt>`](#) field is only an 11-bit parameter, and the PTP Global Timer is 31-bits wide, multiple iterations of the operation need to be performed to get to the adjusted time of day value.

## 8.20.6

### Software Initialization Procedure

The following are the steps required to initialize the PTP core:

1. Set the corresponding per-port [`<DisPTP>`](#) field in the PTP Port Configuration0 Register ([Table 475 p. 604](#)) to a 0x1 (or set all bits, if being done after the power cycle).
2. Configure the [`<PTPType>`](#) field in the PTP Global Configuration0 Register ([Table 470 p. 601](#)) to the EtherType value on which the PTP frames are expected to arrive.
3. Configure [`<MsgIDTSEN>`](#) field in the PTP Global Configuration1 Register ([Table 471 p. 601](#)), specifying the PTP message types that need to be time stamped by hardware.

4. Configure <TSArrPtr> field in the PTP Global Configuration2 Register ([Table 472 p. 602](#)), specifying which of the 16 message types identified by <MsgIDTSEn> field need to occupy Arrival counter 0 and/or 1.
5. Configure <TransSpec> field in the PTP Port Configuration0 Register ([Table 475 p. 604](#)) with the value defined in the IEEE standard. For IEEE802.1AS, this needs to be configured to 0x1, and for IEEE1588, this needs to be configured to 0x0.
6. Configure the <DisTSOverwrite> field in the PTP Port Configuration0 Register ([Table 475 p. 604](#)), <PTPDepIntEn> field in the PTP Port Configuration2 Register ([Table 477 p. 605](#)), the <PTPArrIntEn> field in the PTP Port Configuration2 Register ([Table 477 p. 606](#)), and the <DisTSOverwrite> field in the PTP Port Configuration0 Register ([Table 475 p. 604](#)), as required by the application.
7. Reset the per-port <DisPTP> field to a 0x0.

## 8.21

## Network Management Interface Counters

The device incorporates a set of management counters. For a complete description refer to [Table 499, MAC MIB Counters, on page 623](#).

## 8.22

## Port MIB Counters

The MAC MIB Counters provide the necessary counters that support MAU, IEEE 802.3 and EtherLike MIB. Each port has a set of counters that reside in consecutive address space. Some counters are 64-bits wide.

The counters are meant to provide management software to support:

- IEEE 802.3 DTE Management objects
- Ethernet-like interface MIB: RFC 2665
- Interface MIB: RFC 2863
- Remote Network Monitoring (RMON) groups 1-4: RFC 2819



The MAC MIB counters are not intended to be used for Bridge MIB nor for SMON MIB.

**Note**

[Table 48](#) summarizes the terms used in the definition of the counters.

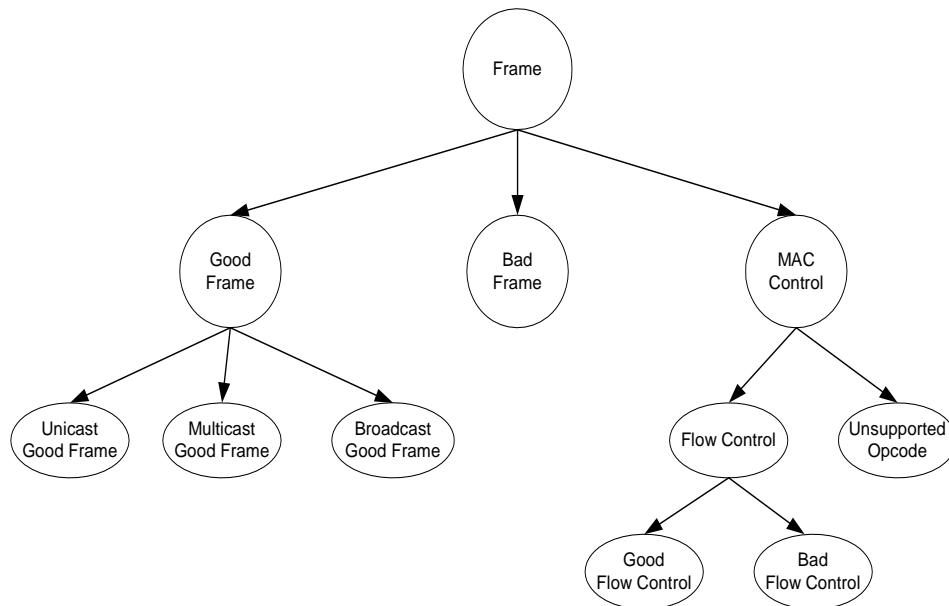
**Table 48: Definitions for MAC MIB Counters**

Term	Definition
Collision Event	A collision has been detected before 576-bit times into the transmitted packet after Px_TXEN is asserted. Relevant to 10 Mbps and 100 Mbps speeds in half-duplex mode only.
Late Collision Event	A collision has been detected after 576-bit times into the transmitted packet after Px_TXEN. Relevant to 10 Mbps and 100 Mbps speeds in half-duplex mode only.
Excessive Collision Event	When a packet to be transmitted suffers from 15 consecutive collision events, therefore, it ought to be dropped according to the IEEE 802.3 standard. Relevant to 10-Mbps and 100-Mbps speeds in half-duplex mode only.
MRU	Maximal Receive Unit: A programmable parameter that sets the maximal length of a valid received packet.
Rx Error Event	The Receive Error signal/symbol was asserted while a frame is received.

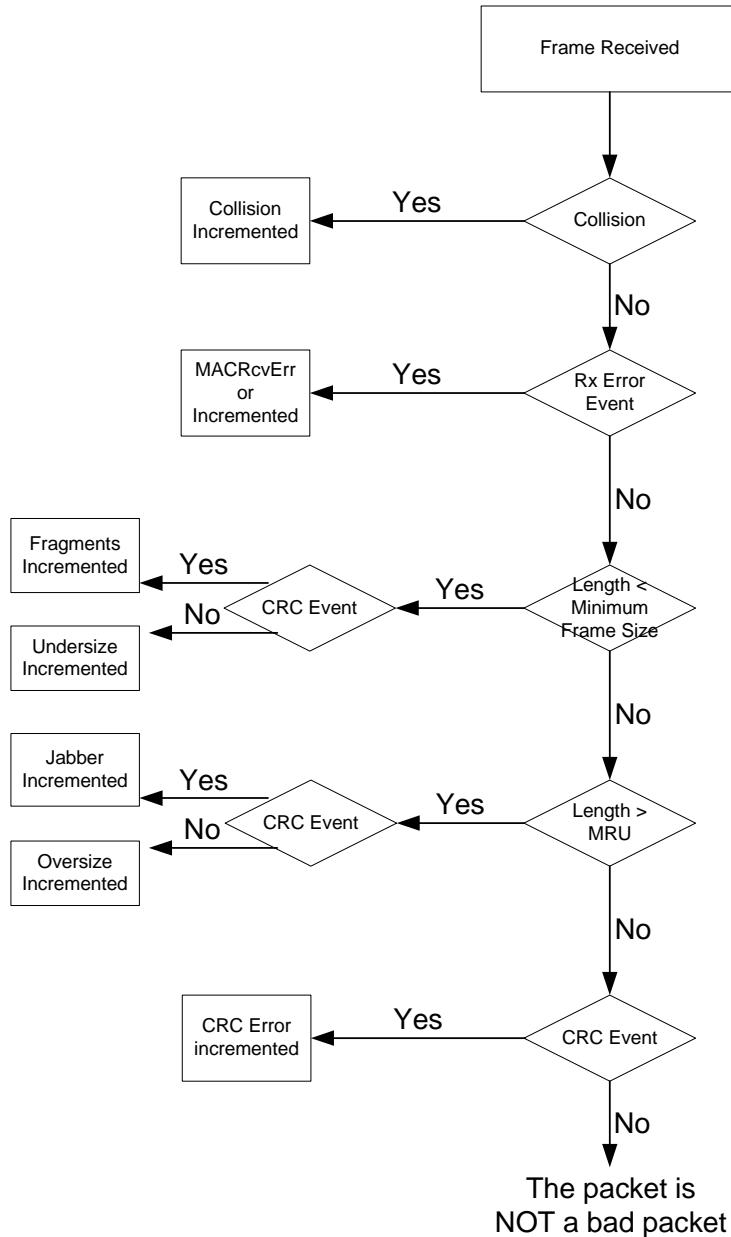
**Table 48: Definitions for MAC MIB Counters (Continued)**

Term	Definition
CRC Error Event	This event occurs whenever an Ethernet frame is received and the following conditions are satisfied: 1. Packet data length is between the Minimum Frame Size - and the MRU byte size inclusive (that is, it is a valid packet data length according to the IEEE standard). 2. Packet has an invalid CRC. 3. Collision Event has not been detected. 4. Late Collision Event has not been detected. 5. Rx Error Event has not been detected.
Undersize packet	An Ethernet frame satisfying <i>all</i> of the following conditions: 1. Packet length is less than Minimum Frame Size bytes. 2. Collision Event has not been detected. 3. Rx Error Event has not been detected. 4. Packet has a valid CRC.
Fragment	An Ethernet frame satisfying <i>all</i> of the following conditions: 1. Packet data length is less than 64 bytes, OR a packet without a Start Frame Delimiter (SFD) and the packet is less than 64 bytes in length. 2. Collision Event has not been detected. 3. Rx Error Event has not been detected. 4. Packet has an invalid CRC.
Oversize packet	An Ethernet frame satisfying <i>all</i> of the following conditions: 1. Packet length is more than the MRU byte size. 2. Collision Event has not been detected. 3. Late Collision Event has not been detected. 4. Rx Error Event has not been detected. 5. Packet has a valid CRC.
Jabber	An Ethernet frame satisfying <i>all</i> of the following conditions: 1. Packet data length is greater than the MRU. 2. Packet has an invalid CRC. 3. Rx Error Event has not been detected.
Tx Error Event	An internal error event in the transmit MAC. This is a very rare situation and when it happens, it means that there the system is misconfigured.
Bad frame	An Ethernet frame that has one of the following conditions met: CRC Error Event, Undersize, Oversize, Fragments, Jabber, Rx Error event and Tx Error Event.
MAC Control Frame	An Ethernet frame that is not a bad frame and has a value of 88-08 in the EtherType/Length field.
Flow Control Frame	A MAC Control Frame with an opcode equal to 00-01.
Good Flow Control Frame	A flow control frame with: 1. MAC Destination equal to 01-80-C2-00-00-01 2. 64-byte length
Bad Flow Control Frame	All flow control frames that are not good flow control frames
Good frame	An Ethernet frame that is not a bad frame NOR a MAC Control frame

Figure 37 and Figure 38 illustrate the terms defined above:

**Figure 37: Ethernet Frame Classification**

**Figure 38: Bad Frame Procedure**



The counters initialize to 0 immediately after reset. Most counters are 32-bits, the Good Bytes received and Bytes Sent are 64-bit counters that should be read in two separate cycles.

The 64-bit counters should be read from the low address and the next read from the MIB counters must be from the high address of the same counter. The MIB interface will always return the high counter data on a read from the MIBs following a read from the low address of a 64-bit counter.

Refer to PHY Address Register ([Table 401 p. 554](#)) for the address offsets of all the counters for the port. Within the counters block, the counter offset is in the address bits[6:0].



## 88F6180/88F619x/88F6281 Functional Specifications

In addition to the per port counters, in [Section A.8.4, MAC MIB Counters](#), there are some additional counters that count filtered frames for reasons like MAC address lookup results, called Port Overrun Frame Counter (PxOFC) Register ([Table 440 p. 583](#)) and Port Rx Discard Frame Counter (PxDFC) Register ([Table 439 p. 583](#)). In conjunction with the counters block they provide total frames received information.

# 9

# Universal Serial Bus (USB 2.0) Interface

The device contains a Universal Serial Bus (USB 2.0) port that includes an embedded USB 2.0 PHY.

The USB 2.0 interface contains a single dual-role controller that can act as a host or a peripheral controller (USB controller). A bridge connects the controller to the internal Mbus interface (USB bridge).

Embedded USB 2.0 PHY features include:

- 480 Mbps High Speed (HS)/ 12 Mbps Full Speed (FS) and 1.5 Mbps Low Speed (LS) serial data transmission rates
- Synchronization/End-of-Packet (SYNC/EOP) generation and checking
- Data and clock recovery from serial stream on the USB
- Non Return to Zero Invert (NRZI) encoding/decoding with bit stuffing/unstuffing
- Bit stuff error detections
- Bit stuffing/unstuffing; bit stuff error detection
- Holding registers to stage transmit and receive data
- Supports USB 2.0 Test Modes
- Ability to switch between FS and HS terminations/signaling

**Note**

For more details, refer to the controller specification document *USB-HS High-Speed Controller Core Reference*.

---

# 10

## Cryptographic Engines and Security Accelerator (CESA)

The device integrates hardware-based cryptographic engines and security accelerator (CESA). The cryptographic engines and the security accelerator reduce CPU packet processing overhead by performing time consuming cryptographic operations such as Advanced Encryption Standard (AES), Data Encryption Standard (DES), and Triple Data Encryption Standard (3DES) encryption and Message Digest 5 (MD5) and Secure Hash Algorithm 1 (SHA1) authentication.

The acronyms, abbreviations and definitions shown in [Table 49](#) are used in this section.

**Table 49: Acronyms, Abbreviations, and Definitions**

Acronym	Definition
AES	Advanced Encryption Standard
AES128/128	128 data bits AES with 128-bit key width
AES128/192	128 data bits AES with 192-bit key width
AES128/256	128 data bits AES with 256-bit key width
Block/data	Block of 512 bits in the Authentication engine
CBC	Cipher Block Chain
CFB	Cipher Feedback
DES	Data Encryption Standard
3DES	Triple Data Encryption Standard
ECB	Electronic Code Book
EDE	Encryption Decryption Encryption
EEE	Encryption Encryption Encryption
IV	Initial Vector / Initial Value
MD5	Message Digest 5
OFB	Output Feedback
SHA-1	Secure Hash Algorithm 1
W0...W15	Designates the 16 words in an authentication input data block; W0 is the first word and W15 is the last word.
word	32-bit

## 10.1 Cryptographic Engine Features

### Engine Features

There are four cryptographic engines that operate independently, one at a time.

- DES encryption/decryption engine
- AES128 encryption engine
- AES128 decryption engine
- Authentication MD5/SHA engine

The user can implement the following algorithms, using the above hardware engines:

- Encryption: DES (ECB and CBC modes) and Triple DES (ECB, CBC EDE and EEE modes)<sup>1</sup>
- Encryption: AES128/128, AES128/192, and AES128/256<sup>2</sup>
- Authentication: SHA-1 and MD5

The cryptographic engines implement the following features:

- Authentication in the MD5 or SHA algorithm, selectable by the user
- Authentication Continue mode, enables chaining between blocks
- Authentication Automatic Padding mode
- Encryption and Decryption in DES, Single (ECB) or Block (CBC) mode, or 3DES, EEE or EDE mode, selectable by the user
- DES write pipeline
- Optimal external update of Authentication and Encryption (in CBC) initial values, enabling flexibility of use—multi-packet calculation, sharing between resources
- Byte Swap support for DES/3DES and AES data input/output.
- Automatic Engine activation when the required data block is loaded, (Saves write cycles)
- Authentication and encryption termination interrupts
- Supports DES, OFB, and CFB modes with additional software
- AES encryption and decryption—completely separate engines that can work simultaneously when TDMA<sup>3</sup> is not used

## 10.2 Security Accelerator Features

There is one security accelerator which implements the following features.

- Performs a complete over-the-packet operation with no software intervention
- Supports four types of operation:
  - Authentication only (MD5 / SHA-1 / HMAC-MD5 / HMAC-SHA1)
  - Encryption/Decryption only (DES / 3DES / AES—both ECB and CBC)
  - Authentication followed by Decryption/Encryption
  - Decryption/Encryption followed by Authentication

## 10.3 Cryptographic Engines Operational Description

The unit combines four separate engines:

- Authentication MD5/SHA engine (see [Section 10.3.2, Authentication, on page 177](#))
- DES encryption/decryption engine (see [Section 10.3.3, DES Encryption/Decryption, on page 180](#))

---

1. ECB = Electronic Code Book, CBC = Cipher Block Chain, EDE = Encryption Decryption Encryption, and EEE = Encryption Encryption Encryption.

2. AES128/128 = 128 data bits AES with 128-bit key width, AES128/192 = 128 data bits AES with 192-bit key width, and AES128/256 = 128 data bits AES with 256-bit key width.

3. TDMA is the Cryptographic engine's Direct Memory Access (DMA).



- AES128 encryption engine (see [Section 10.3.4, AES128 Encryption, on page 184](#))
- AES128 decryption engine (see [Section 10.3.5, AES128 Decryption, on page 187](#))

Each of these four independent cryptographic engines has separate registers for data, control, and operation modes. The address allocation is specified in [Table 516, Register Map Table for the Cryptographic Engine and Security Accelerator \(CESA\) Registers, on page 634](#).

### 10.3.1 Using the Cryptographic Engines

**Access** The cryptographic engines can be accessed by the security accelerator or by the host<sup>1</sup>. The access is performed by writing and reading to specified addresses in the engine.

**Commands and Control** The engines' modes of operation (AES, DES, 3DES, and SHA) and the endianess of the input data are controlled by the host. Control is performed via four specified command registers:

- SHA-1/MD5 Authentication Command Register ([Table 567 p. 650](#))
- DES Command Register ([Table 549 p. 643](#))
- AES Decryption Command Register ([Table 529 p. 638](#))
- AES Encryption Command Register ([Table 542 p. 641](#))

These registers also contain flags that are used as status indicators to the host.

The engines also provide interrupts that are set when an operation is completed (see [Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register \(Table 556 p. 645\)](#)).

**Input Data** The encryption engines operate on data blocks of 64 bits or 128 bits, while the authentication engine requires a block of 16 words (512 bits) as input.

The input data is loaded by writing to data registers.

The engines, excluding DES, do not support multi-tasking. When a data block is written to one of the engines, the host must wait until this engine finishes the calculation before it writes the next input data block.

The engines also require cryptographic parameters, such as keys and initial values, which vary according to the mode of operation used. Setting these parameters is performed to by writing to the specific Key and Initial Value (IV)/Digest registers.

Both the encryption engine and the authentication engine support byte swap of input data.

**Output Data** The encryption engines return a cipher/decipher data block of 64 or 128 bits. The engines support byte swap of their output data.

The authentication engine returns four or five words that are the hash signature of the input data block.

The output data is accessed through specific Data In/Out registers in the engines.

**Principle of Operation** When a host wants to perform cryptographic operations, it writes the cryptographic parameters (for example, IV and keys) and the command to be performed in registers. Then the host writes the data to be processed (in the data registers). This triggers the engine, which starts the processing automatically after the required amount of data was written. When the engine finishes the cryptographic calculation, it sets a termination bit in one of the command registers and activates an interrupt to notify the host that the operation is finished. After the operation is finished, the host can read the result of the operation from the engines' registers.

1. The word host is used as a generic term representing the agent that controls the operation of the cryptographic engines (either the CPU or the security accelerator). While the accelerator is working the host cannot work with the cryptographic engines and vice versa.

## 10.3.2 Authentication

To activate the authentication process, the host should perform the following:

1. Verify the [\*\*<Termination>\*\*](#) field in the SHA-1/MD5 Authentication Command Register ([Table 567 p. 651](#)).

The host must read the register to verify that the engine is not busy. Writing in the middle of a calculation results in erroneous data.

At this stage the host must read the register to verify that the engine is not in the middle of a calculation process. Writing in the middle of a calculation results in erroneous data.

After reset the [\*\*<Termination>\*\*](#) is set. Any write that the host performs to the Authentication engine resets the termination bit.

2. Write to the [\*\*<Mode>\*\*](#) field and [\*\*<DataByteSwap>\*\*](#) field in the SHA-1/MD5 Authentication Command Register ([Table 567 p. 651](#)) (This is optional if no change is needed.)

3. Write initial values:

This step is only required if multiple packets are processed simultaneously.

Externally written Initial values are valid only if the [\*\*<Mode>\*\*](#) is set to 1 (Continue mode).

In SHA, the initial value is five words long and in MD5 it is four words long. The addresses of the IV/digest registers are specified in the SHA-1/MD5 Initial Value/Digest A Register ([Table 562 p. 649](#)) through SHA-1 Initial Value/Digest E Register ([Table 566 p. 650](#)).

The host may write to any of these registers. Initial values registers that are not written contain the digest from the previous calculation.

**NOTE:** The host may change the value of the command register during the process of writing the data when it is necessary to swap only part of the data.

4. Write data words.

SHA/MD5 algorithms work in 512-bit chunks, which are equal to 16 words. There are two ways to write the data words to the engine:

- Write cycles to Data In register.
- Write cycles to the Data In register and to Byte Count registers.

### 10.3.2.1 Write Cycles to Data In Register

The host must perform 16 write cycles, first to W0, then to W1 through W15.

### 10.3.2.2 Write Cycles to the Data In Register and to Byte Count Registers

This type of access is preferred where a packet is small (i.e., less than 14 words) or for the last chunk of a packet whose size is less than 14 words. In these cases, the algorithm requires a zero padding to 14 words in addition to the 2-word padding of the bit count needed in single/last chunks. This requires successive writes of full zero words.

In this type of access, the writing of these zero padding is skipped, and thus less than 16 write accesses activate the engine.

This access is performed as follows:

1. The host writes between 0 to 13 words to the Data In register. The write to the Data In register is performed until the word that contains bit N+1 of the data, where N is the place of the last bit of data in the chunk. The last word written must be padded with one bit of 1 and then zeros until the completion of the 32-bit word.
2. Then, the host writes the lower part (MD5) or the higher part (SHA) of the bit count value to the SHA-1/MD5 Bit Count Low Register ([Table 568 p. 651](#))—word 14 (see [Figure 39](#)).
3. Then, the host writes the higher part (MD5) or the lower part (SHA) of the bit count value to the SHA-1/MD5 Bit Count High Register ([Table 569 p. 651](#))—word 15 (see [Figure 39](#)). After this

write, the engine starts working automatically, and all words of the data chunk from the last word written to the Data In register until word 14 are considered as zeros.

**Figure 39: Authentication of a Data Chunk**

Less than 16-Word	16-Word
word 0	word 0
word 1	word 1
:	
:	
:	
word 13 or less	word 13
bit count low	word 14
bit count high	word 15

4. Poll the SHA-1/MD5 Authentication Command Register or wait for the interrupt:

After the engine is loaded with the data chunk or after a write to the two Bit Count registers, the engine starts working automatically.

When the <Termination> is 1, it is an indication for the host that the engine finished the calculation process, and the digest is ready.

The host can poll the <Termination> bit or wait for the ZInt0 interrupt—bit 0 in the Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register ([Table 556 p. 645](#)). This interrupt is activated on the rising edge of the <Termination> bit. To clear the interrupt, the host must write a 0 to it. Writing a 1 has no effect.

5. Read result:

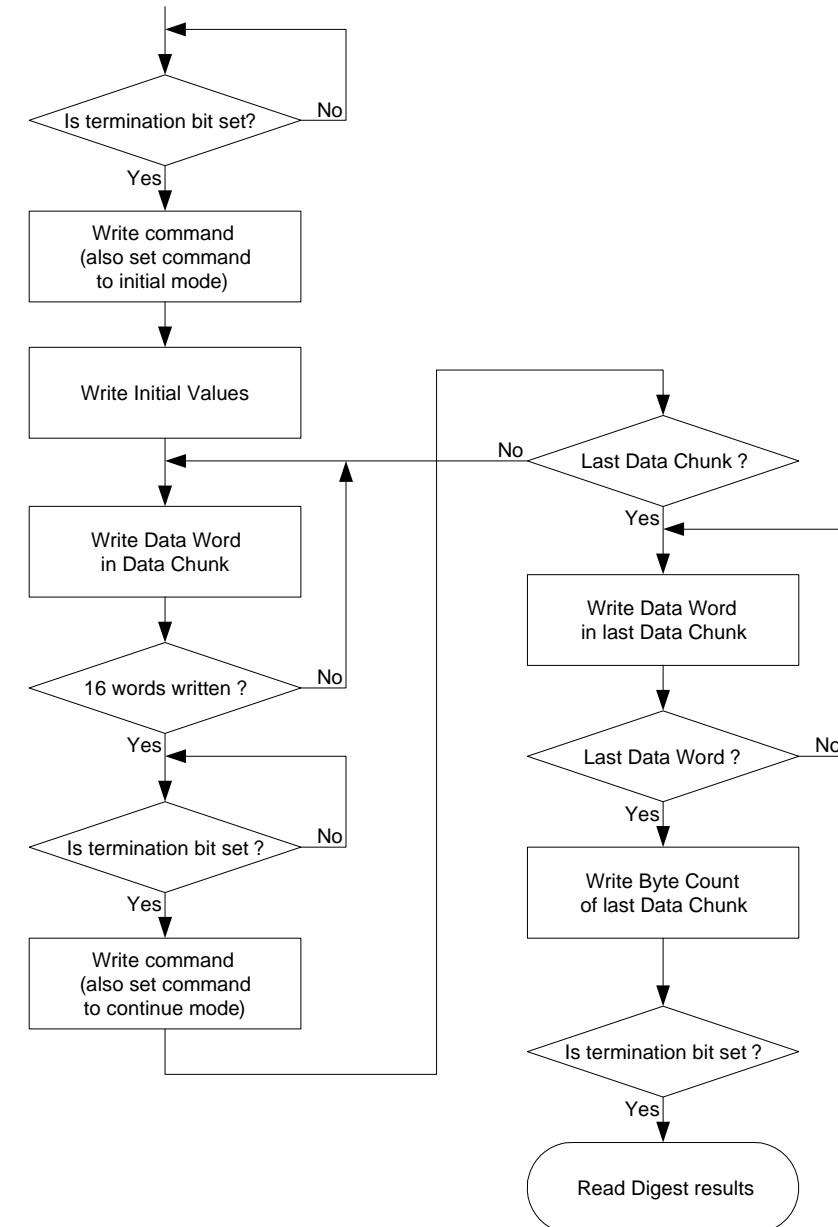
Once the <Termination> bit was asserted, the host can read the digest. The digest length is five words for SHA-1 or four words for MD5. These words are stored in the IV/Digest registers (see [Table 562, p. 649](#) through [Table 570, p. 652](#)).

After reading the result, the host can immediately start the write command again for initial values and data, or just data.

[Figure 40](#) shows a typical authentication flow for a packet.

**Figure 40: Typical Authentication Flow for a Packet**

---



### 10.3.3 DES Encryption/Decryption

The DES encryption algorithm complies with the DES standard as described in FIPS PUB 46-2.

The engine implements two different modes:

**ECB** A direct application of the DES standard to encrypt and decrypt data

**CBC** An enhanced mode of ECB, which chains together blocks of cipher text. The chain's glue is the DES IV (Initial Value) register (see [Table 543 on page 642](#) and [Table 544 on page 642](#)).

Two other modes, CFB and OFB, may be implemented by the engine, however, additional software is required.

The engine implements two triple DES (3DES) modes, as described in RFC 1851:

- EEE
- EDE

The 3DES modes can work with three different keys for high security and can be combined with ECB or CBC modes.

All the modes are reciprocal, that is, they decipher or cipher data.

Encryption/Decryption calculation time is 9 cycles in DES mode and 25 cycles in 3DES mode. This is without taking into consideration the read and write cycles, associated with writing the input data/key and reading the result.

To activate the Encryption engine, the following steps are required:

1. Verify termination bits in the DES Command Register ([Table 549 p. 643](#)).

At this stage the host must read the register, to verify that the engine is not in the middle of a calculation process and that the engine's parameters (i.e., DES operation modes and either the DES key or the 3DES keys) can be updated.

In the initial operation, it is always necessary to write the DES key or the 3DES keys and possibly to write an operational mode other than the default operational mode. In that case, before the engine's parameters are written, the [`<AllTermination>`](#) field in the DES Command Register ([Table 549 p. 644](#)) (bit [30]) must be set.

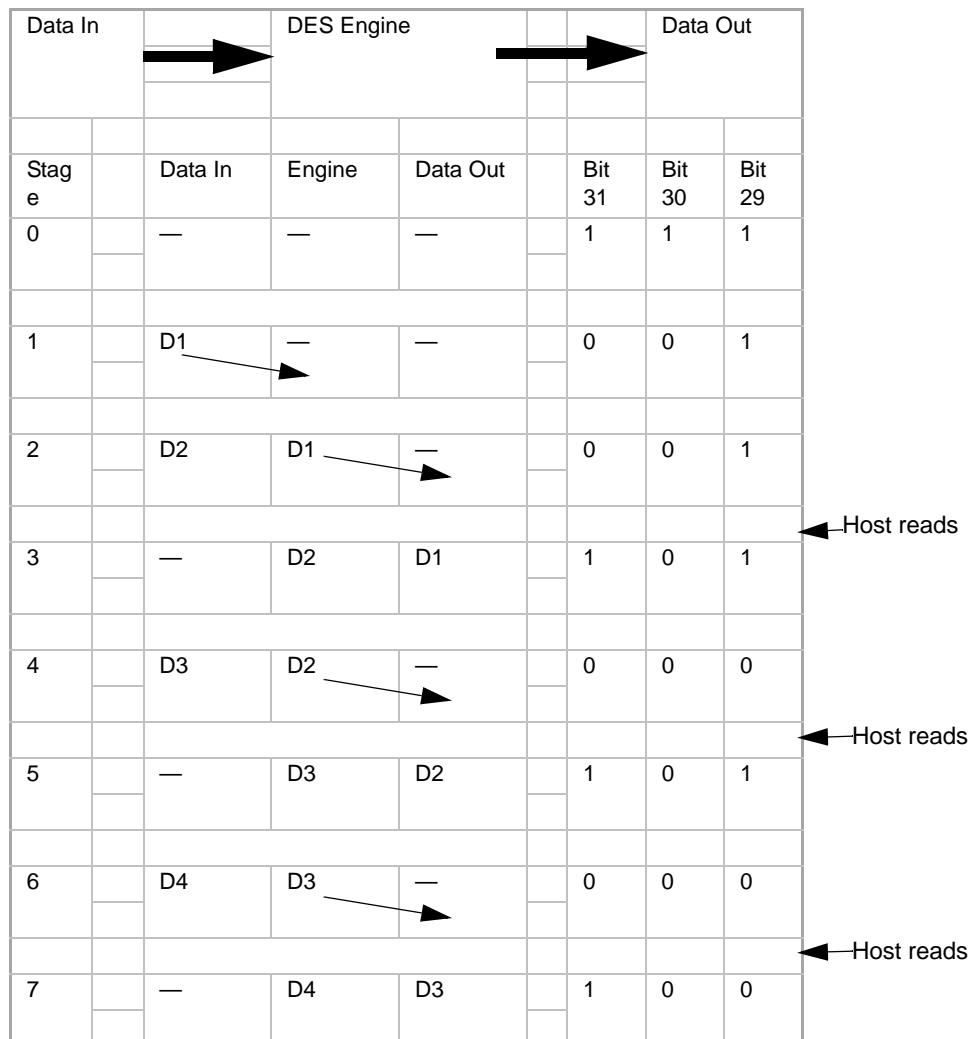
When the [`<Termination>`](#) field (bit [31]) in that register is set, a 64-bit DES data block can be written to the engine, but this does not necessarily mean that parameters can be updated. Writing to the engine de-asserts this bit.

However, when the [`<AllTermination>`](#) is set, a DOUBLE 64-bit DES data block can be written to the engine. In addition, when this bit is set the engine's parameters can be updated.

The DES engine operates on a pipeline principle (see [Figure 41](#)).

The host writes to the engine when bit [`<WriteAllow>`](#) field in the DES Command Register ([Table 549 p. 644](#)). When a result is ready, the host must read it, to enable the engine to process the next data in the pipeline. When bit [`<WriteAllow>`](#) (bit [29]) is set, the host can write one data block to the engine.

**Figure 41:DES Engine Pipeline**



**Note:** A read of Dataout Low resets bit[29].

Meaning of bits [31:30] and their possible states:

Bit [31]	Bit [30]	Description	Number of Data 64-Bit Blocks That Can Be Written	Can Engine Parameters Be Updated?
0	0	Engine is busy.	1—only if bit [29] is set	No
0	1	Engine is ready.	2	Yes
1	0	First data block waiting for read; pipeline is full.	1—only if bit [29] is set	No
1	1	Engine completed calculations; pipeline is empty.	2	Yes

2. Write operational mode and endianess for fields in the DES Command register:

<b>Direction</b>	Encryption or decryption
<b>Algorithm</b>	Single DES or 3DES
<b>Triple DES mode</b>	EEE or EDE
<b>Chain</b>	ECB or CBC
<b>Data byte swap</b>	Similar to swapping performed in the Authentication engine. See <a href="#">Section 10.2</a> .
<b>IV byte swap</b>	Swapping of data written to the DES IV register. Similar to data swap.
<b>Data out byte swap</b>	These bits control byte swap of the cipher/decipher output result.

3. Write the keys.

Each key is a 64-bit block. Writing a single key requires two write operations.

**Single DES mode** A write to a single key must be performed. The key used in DES mode is the KEY0 register.

**3DES mode** It is recommended to write to three different keys—KEY0, KEY1, and KEY2 registers.

Since keys are changed only occasionally, this step is not always required.

4. Write the initial value (IV).

This step is necessary only in the DES/3DES CBC modes. A 64-bit block must be written to the IV register. In CBC, the IV value is XORed with the Data\_In. The result is used as the input to the cipher/decipher machine.

Writing the IV register requires two write operations, one to the DES Initial Value High Register ([Table 544 p. 642](#)) and the other to the DES Initial Value Low Register ([Table 543 p. 642](#)). However, in 64-bit mode, a single write operation is required.

5. Write blocks for the 64-bit data.

DES encryption requires a 64-bit block of input data, which is loaded by writing to one of the DES Data Buffer registers (see [Table 552, p. 644](#) or [Table 553, p. 645](#)). One or two data blocks can be loaded according to the engine status (see the example below).

If a data block is shorter than 64-bit length, the specification requires zero padding to 64 bits.

**NOTE:** If the next block to be processed uses the same cryptographic parameters (i.e., the same keys and modes), and if the IV is the output data of the previous block, the host writes only the DES Data Buffer registers (see [Table 552, p. 644](#) or [Table 553, p. 645](#)). This is useful when it is necessary to encrypt a message that consists of multiple 64-bit blocks. In this case, it is efficient to use the DES pipeline option and write two blocks at once, as specified in the example below.

The DES machine starts working automatically when a 64-bit data block is written to it, or when there is data in the DES machine pipeline, and the previous result has been read.

Operation starts after the host writes to the DES data buffer addresses. The host must first write to the DES data in/out low address and then to the DES data buffer high address, as shown in the following example.

The data block to encrypt is 0x1122334455667788.

The host writes 0x55667788 to address 0xDD70.

The host writes 0x11223344 to address 0xDD74.

Results:

Byte Swap	DES Data Buffer Register Actual Data That Will Be Encrypted
0	1122334455667788
1	4433221188776655

**NOTE:** Other writes to addresses 0xDD70/0xDD74 will cause unexpected results.

6. Poll the DES Command Register ([Table 549 p. 643](#)) or wait for the interrupt.  
 After the engine is loaded with one 64-bit blocks, it starts working automatically. The host must not write anything to the engine until the [`<WriteAllow>`](#) field (bit[29]) is set.  
 At this stage, the host must poll the [`<Termination>`](#) field (bit[31]). When the bit is 1, it is an indication to the host that the engine finished the calculation process and the result is ready.  
 The host may write one data block each time, or it may perform consecutive writes of two data blocks:

**The host writes one data block each time:**

The result of the data block is ready and no more data is in the engine pipeline. Bit [30], [`<AllTermination>`](#), is set and engine parameters can be updated.

**The host performs consecutive writes of two data blocks:**

**First data block** When the first data block result is ready, the [`<Termination>`](#) field and the [`<WriteAllow>`](#) field are set. At this stage, the engine has the second data in the pipeline. However, the calculation cannot start until the host has read the result of the first data via the DES Data Out registers (read the Data High before the Data Low). Here the [`<AllTermination>`](#) field is not set.

Once the first data result is read, the [`<WriteAllow>`](#) field is reset, and the engine starts the calculation of the second data. At this stage, the host can write one data block (the third block) to the engine. This is indicated by the [`<WriteAllow>`](#) field (see [Figure 41 on page 181](#)).

**Second data block** When the second data block result is ready, the [`<Termination>`](#) field is set. At this stage, the engine has completed the second data calculation, the pipeline is empty, and the [`<AllTermination>`](#) field is also set. Once the data result is read, the host can alter the DES parameter and then write two data blocks.

The ZInt1 interrupt is set every time the [`<AllTermination>`](#) field in the [DES Command Register](#)—changes from 0 to 1.

The ZInt4 interrupt is set every time the [`<Termination>`](#) field in the [DES Command Register](#)—changes from 0 to 1.

After the interrupt occurs, the host writes 0 to the relevant interrupt bit in the Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register ([Table 556 p. 645](#)) to reset it. Writing 1 has no effect.

7. Read DES result.  
 Once a termination bit has been asserted, the host may read the result. The result of the encryption (or decryption) is stored in the DES Data In/Out registers.  
 Two read operations are required to read the result—the first read from address 0xDD7C and the second read from address 0xDD78.

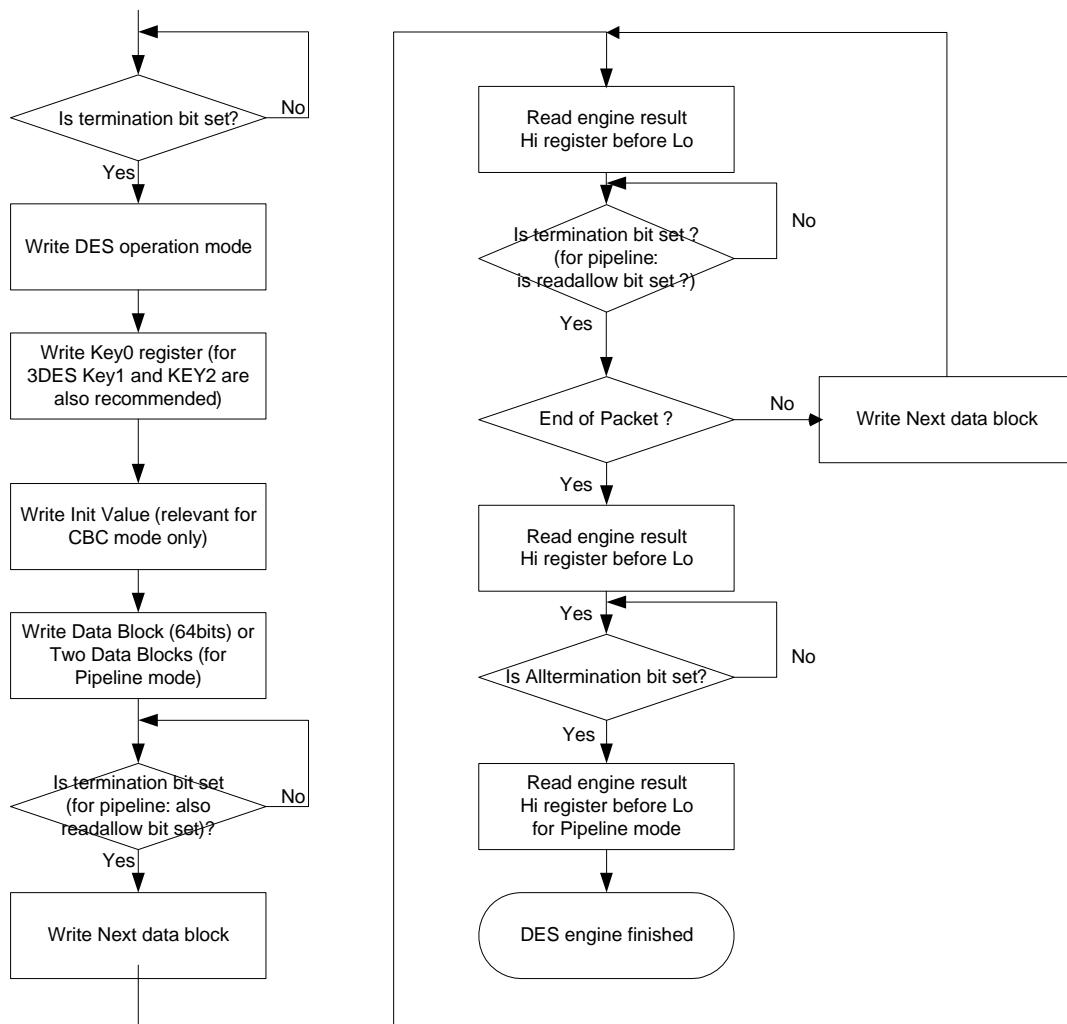
Byte swap bits have no effect on reads, i.e. the engine does not perform any endianess change on the result. To perform an endianess change use the <OutByteSwap> field in the DES Command Register ([Table 549 p. 644](#)).

#### 8. Read keys and IV.

It is possible to read the key values and IV value at anytime by accessing the appropriate registers. In Chain modes (CBC), the IV register is changed by the machine at the end of every DES calculation cycle. In this situation, the IV register is loaded with the calculation result, so it should have the same value as the DES Data Out register (see [Table 554, p. 645](#) and [Table 555, p. 645](#)).

[Figure 42](#) illustrates the typical DES/3DES packet encryption flow.

**Figure 42: Typical DES/3DES Encryption Flow for Packet**



#### 10.3.4 AES128 Encryption

The AES128 Encryption engine complies with the AES standard as described in the FIPS standard.

This engine performs encryption only. The decryption is performed by the AES128 decryption engine.

The engine implements the AES algorithm on a 128-bit data block on three possible Key Length modes:

- 128-bit mode
- 192-bit mode
- 256-bit mode

Encryption calculation time is 20 cycles, without taking into consideration the read and write cycles associated with writing the input data/key and reading the result.

To activate the AES Encryption engine, the following steps are required.

1. Verify the <Termination> field in the AES Encryption Command Register ([Table 542 p. 642](#)). At this stage the host must read the register, to verify that the engine is not in the middle of a calculation process. Writing in the middle of a calculation will result in erroneous data. Unless the host made a write to the engine before, the <Termination> field must be set (The host must set this bit after reset.).

Any write that the host performs to the AES Encryption engine resets the <Termination> field.

2. Write operational mode and endianess for fields in the AES Encryption Command Register:
  - Key Length mode—128, 192 or 256 bit
  - Data byte swap: This is similar to the swapping performed in the Authentication engine. Refer to [Section 10.3.2](#) for more details.
  - Data out byte swap: These bits control byte swap of the cipher output result.
3. Write key.

The AES key length can be changed in accordance with the Key Length mode selected. It may be a 128-, 192- or 256-bit block. Writing a single key requires four, six, or eight write operations.

With the key maximum length 256-bit block, the block is structured from 8 words, each word is a column in the AES Cipher Key block:

AES key column 0	AES key column 1	AES key column 2	AES key column 3
------------------	------------------	------------------	------------------

Thus, there are eight AES encryption key registers, each of them containing a column of the AES cipher key block.

Commonly a 4-word key is used. In this case, the host must write to the Key Column 0, 1, 2, and 3 registers.

Since keys are only changed occasionally, this step is not always required.

4. Write block for the 128-bit data.

The AES encryption requires a 128-bit block of input data. This block is loaded by writing to the AES Encryption Data In/Out register.

If a data block is shorter than 128-bits, the specification requires zero padding to 128 bits.

**NOTE:** If the next block to be processed uses the same key; the host should write only the AES Data In/out encryption registers (see [Table 538 on page 640](#) through [Table 541 on page 641](#)). This is useful for encrypting a message consisting of multiple 128-bit blocks.

The AES machine starts working automatically when a 128-bit data block is written to it (i.e., operation starts after the host writes to all the AES data in/out addresses).

This data is a 128-bit block. This block is structured from four words. Each word is a column in the AES Cipher Data block:

AES Data column 0 (0xDDAC)	AES Data column 1 (0xDDA8)	AES Data column 2 (0-DDA4)	AES Data column 3 (0-DDA0)
-------------------------------	-------------------------------	-------------------------------	-------------------------------

Thus, there are four AES Encryption Data In/Out registers. Each of them containing a column of the AES cipher block.

Upon writing to all these registers, the AES cipher machine will automatically start working as shown in the following example. The order in which these registers is written is not significant.

The data block to encrypt is 0x9900AABBCCDDEEFF1122334455667788.

This data is a 128 bit block, which is structured from four words.

Each word is a column in the AES Cipher data block.

The data must be loaded to the cipher machine as follows:

Write 0x55667788 to address 0xDDA0.

Write 0x11223344 to address 0xDDA4.

Write 0xCCDDEEFF to address 0xDDA8.

Write 0x9900AABB to address 0xDDAC.

There is full support for data byte swap to the AES data blocks, similar to the DES engine, but in the AES cipher engine all data column registers are one word in width.

Results of this write operation:

<b>Byte Swap</b>	<b>AES Data In/out</b>
0	9900AABBCCDDEEFF1122334455667788
1	BBAA0099FFEEEDDC4433221188776655

**NOTE:** Other writes to addresses 0xDDA0, 0xDDA4, 0xDDA8, or 0xDDAC cause unexpected results.

5. Poll the AES Command register or wait for the interrupt:

After the engine is loaded with the 128-bit block, it starts working automatically. The host must not write anything to the engine until it finishes the calculation.

At this stage, the host must poll the [Termination](#) field in the AES Encryption Command Register ([Table 542 p. 642](#)). When the value is 1, it is an indication to the host that the engine finished the calculation process, and the result is ready.

Authentication calculation termination will activate the ZInt2 interrupt (see the Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register ([Table 556 p. 645](#))), which can serve as an alternative to host polling. This interrupt is set every time the [Termination](#) changes from 0 to 1.

After the interrupt occurs, the host should write 0 to the [ZInt1](#) field in the Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register ([Table 556 p. 645](#)) to reset it. Writing 1 has no effect.

6. Read AES result.

Once the termination bit has been asserted, the host may read the result. The result of the encryption is stored in the AES Data In/Out registers (see [Table 538 on page 640](#) through [Table 541 on page 641](#)).

Four read operations are required to read the result, i.e., the host must read addresses 0xDDA0, 0xDDA4, 0xDDA8, and 0xDDAC.

Byte swap bits have no effect on reads, i.e., the engine does not perform any endianess change on the result. To perform an endianess change, use the [OutByteSwap](#) field in the AES Encryption Command Register ([Table 542 p. 642](#)).

7. Read keys.

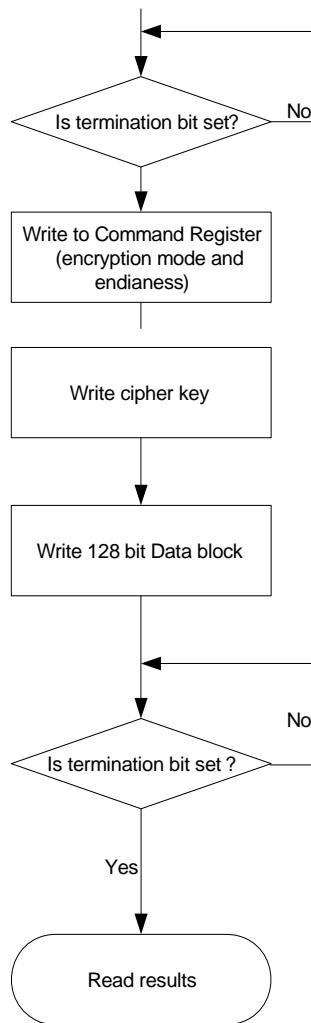
It is possible to read the key values at anytime, by accessing the appropriate registers. To perform an endianess change on key reads, use the [OutByteSwap](#).

The AES Encryption Command Register controls the AES encryption modes. It also flags when the processing is complete.

**NOTE:** The AES encryption key reads are necessary for the AES decryption engine (see [Section 10.3.5](#)).

[Figure 43](#) shows a typical AES encryption flow for a data block.

**Figure 43: Typical AES Encryption Flow for a Data Block**



### 10.3.5

### AES128 Decryption

The AES128 Decryption engine complies the AES standard as described in the FIPS draft. This engine only performs decryption. The modes of operation and activation are similar to the AES encryption. The difference is that before loading the key, the host must calculate a decryption key. This process is performed by loading a key into the AES encryption unit, performing a “dummy” encryption cycle, and then reading the resolved key from that engine. That result is the decryption key.

A decryption key is the last 128/192/256 bits of the key block created by the key expansion algorithm.



As in the encryption engine, the decryption engine implements AES algorithm on a 128-bit data block size in three possible mode sizes:

- 128-bit mode
- 192-bit mode
- 256-bit mode

Decryption calculation time is 20 cycles, without taking into consideration decryption key calculations and read and write cycles, associated with writing the input data/key and reading the result.

To activate the AES Decryption engine, the following steps are required:

1. Calculate the decryption key.

This step is unique to the AES Decryption engine, and is only necessary when a new key, which was never used before, is loaded. The AES algorithm uses a complex key schedule. Thus, at the end of the encryption operation the key is changed. To perform AES decryption the engine must actually start from the key at the end of the encryption key schedule. To decrypt a data block with a given key, the host must first load this key into the decryption engine, then start the key generation process setting [`<AesDecMakeKey>`](#) field in the AES Decryption Command Register ([Table 529 p. 638](#)) bit to 1. At the end of the key generation process, the host reads the key registers from the Encryption engine. This decryption key is loaded by the host into the decryption key registers, to start the required decryption process.

To read the decryption key from the encryption engine, the host must set the [`<AesDecKeyReady>`](#) field in the AES Decryption Command Register ([Table 529 p. 639](#)) to 1 prior to the reading of the AES encryption key registers. Setting this bit enables reading of the internal key in the AES Encryption engine, which at the end of an encryption process, is the key for the decryption start point.

The host may store the decryption key in memory, so that the decryption key calculation may be skipped next time, and the same key used.

2. Verify the termination bit in the AES Decryption/Encryption Command register.<sup>1</sup>
3. Write operational mode and endianess for fields in the [`AES Encryption Command Register`](#)—This is the same as for the AES encryption.
4. Write decryption key.
5. Write block for the 128-bit data.
6. Poll the AES Decryption/Encryption Command register or wait for the interrupt<sup>1</sup>.
7. Read AES result<sup>1</sup>.
8. Read keys.

It is possible to read the key values at anytime by accessing the appropriate registers. To perform an endianess change on key reads, use the [`<OutByteSwap>`](#) bit of the AES Encryption Command Register ([Table 542 p. 641](#)) or the AES Decryption Command Register ([Table 529 p. 638](#)).

The AES Decryption Key registers (see [Table 517 on page 636](#) through [Table 524 on page 637](#)) contain the AES key block for the Decryption engine. A read to these registers returns the last key written there by the host. The host must load a pre-calculated decryption key to these registers (as described previously, see “[Calculate the decryption key.” on page 188](#)).

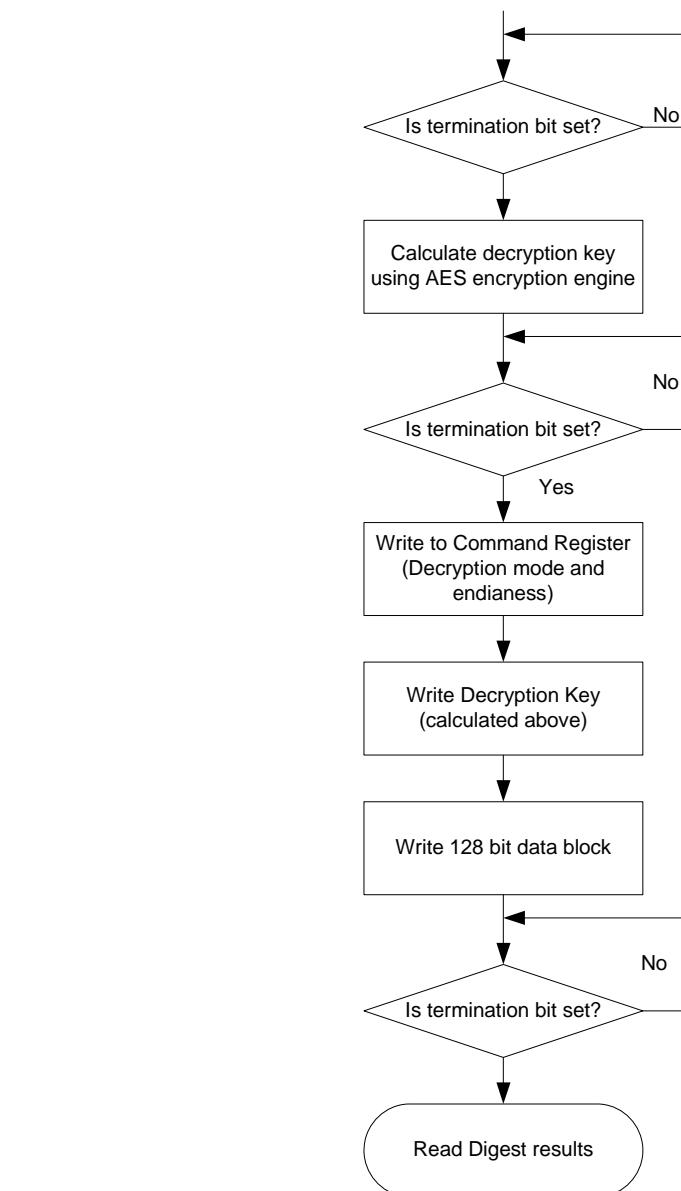
**NOTE:** Directly loading an encryption key to these registers returns incorrect results!

The AES Decryption Command Register ([Table 529 p. 638](#)) controls the AES decryption operation. It also flags when the processing is complete.

1. This is the same as for the AES encryption (see [Section 10.3.4](#)).

[Figure 44](#) shows a typical decryption flow for a data block.

**Figure 44: Typical AES Decryption Flow for a Data Block**



## 10.4 Security Accelerator Operational Description

### 10.4.1 Using the Security Accelerator

The accelerator is activated by the CPU. The entire operation of the accelerator is performed in the security accelerator local SRAM as follows:

- The CPU performs the following:

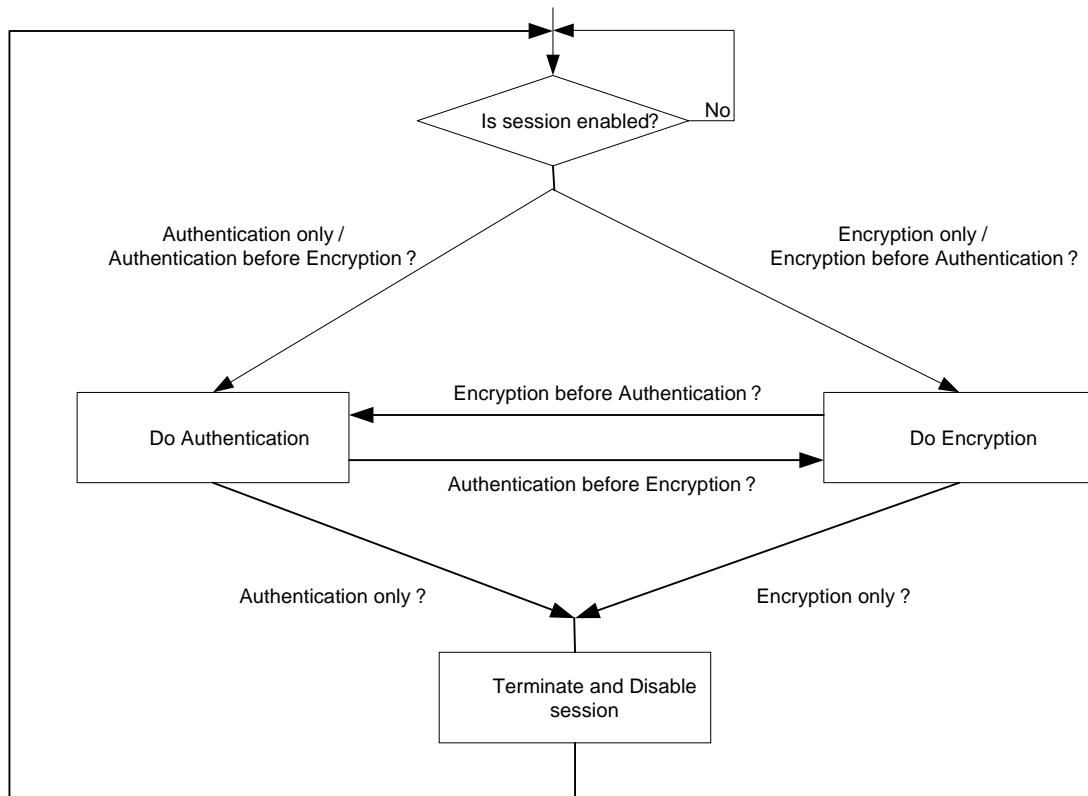
1. Copies the packet to be processed into the security accelerator local SRAM or the uses the TDMA engine instead for this process.
  2. Prepares a descriptor, stating the required operation (see [Section 10.4.4 "Security Accelerator Descriptor Data Structure"](#))
  3. Activates the accelerator
- The accelerator performs the following:
    1. Reads the descriptor
    2. Sets up the engines
    3. Starts feeding the packet data into the engine one data block at a time
    4. Waits for completion,
    5. Reads the data from the engine
    6. Stores the data in the security accelerator local SRAM.

This process repeats until the entire packet is processed.

## 10.4.2 Hardware Flow Chart

[Figure 45](#) shows the main accelerator decision flow.

**Figure 45: Security Accelerator Main Decision Flow**



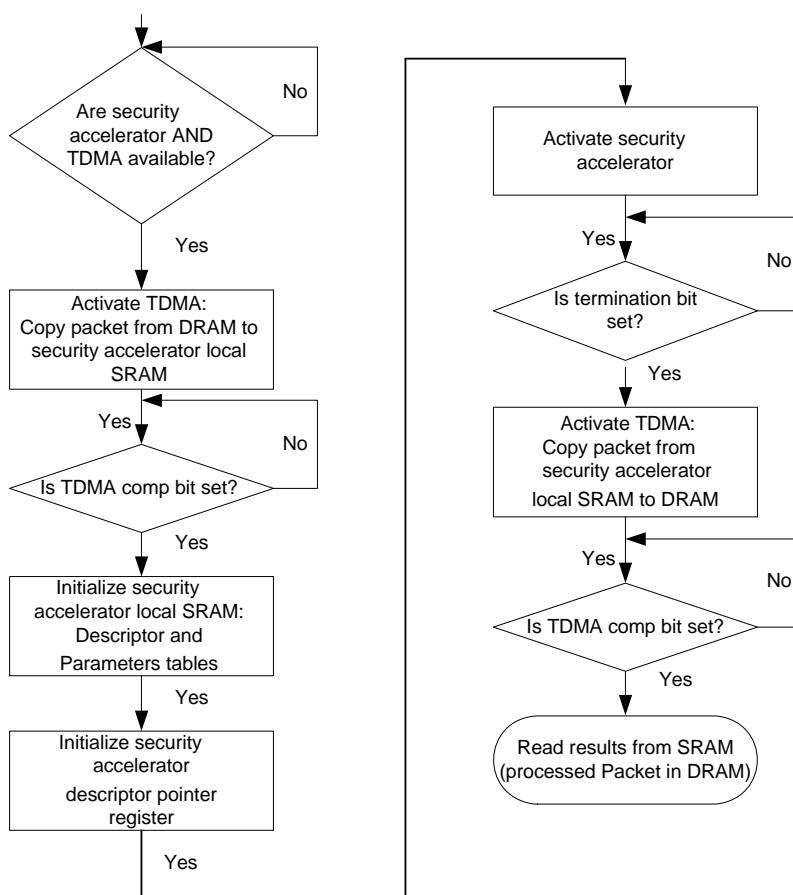
### 10.4.3 Software Flow Chart

The managing software performs the following:

1. Uses the TDMA to copy the packet from main memory into the local SRAM of the security accelerator. Since the local SRAM of the security accelerator is only 2 KB, it cannot contain large packets.
2. Stores the cryptographic parameters relevant for the security accelerator operation to be undergone by the packet.
3. Prepares a descriptor in the local SRAM of the security accelerator, stating the required operation and parameters for the accelerator. The descriptor, which is 8 Dwords long, is described in [Section 10.4.4](#).
4. Writes the pointer to this descriptor into the selected session, the <SecurityAcclDescPtr0> field in the Security Accelerator Descriptor Pointer Register ([Table 559 p. 647](#)).
5. Activates the programmed session by setting the appropriate bit in the Security Accelerator Command Register ([Table 558 p. 647](#)).
6. Waits for the session completion indication either by polling the Security Accelerator Status Register ([Table 561 p. 648](#)) or by interrupt.
7. Uses the TDMA to copy the processed packet back into main memory.

[Figure 46](#) illustrates the security acceleration flow for packet processing and [Figure 47](#) illustrates the enhanced mode.

**Figure 46: Security Acceleration Flow for Packet Processing**



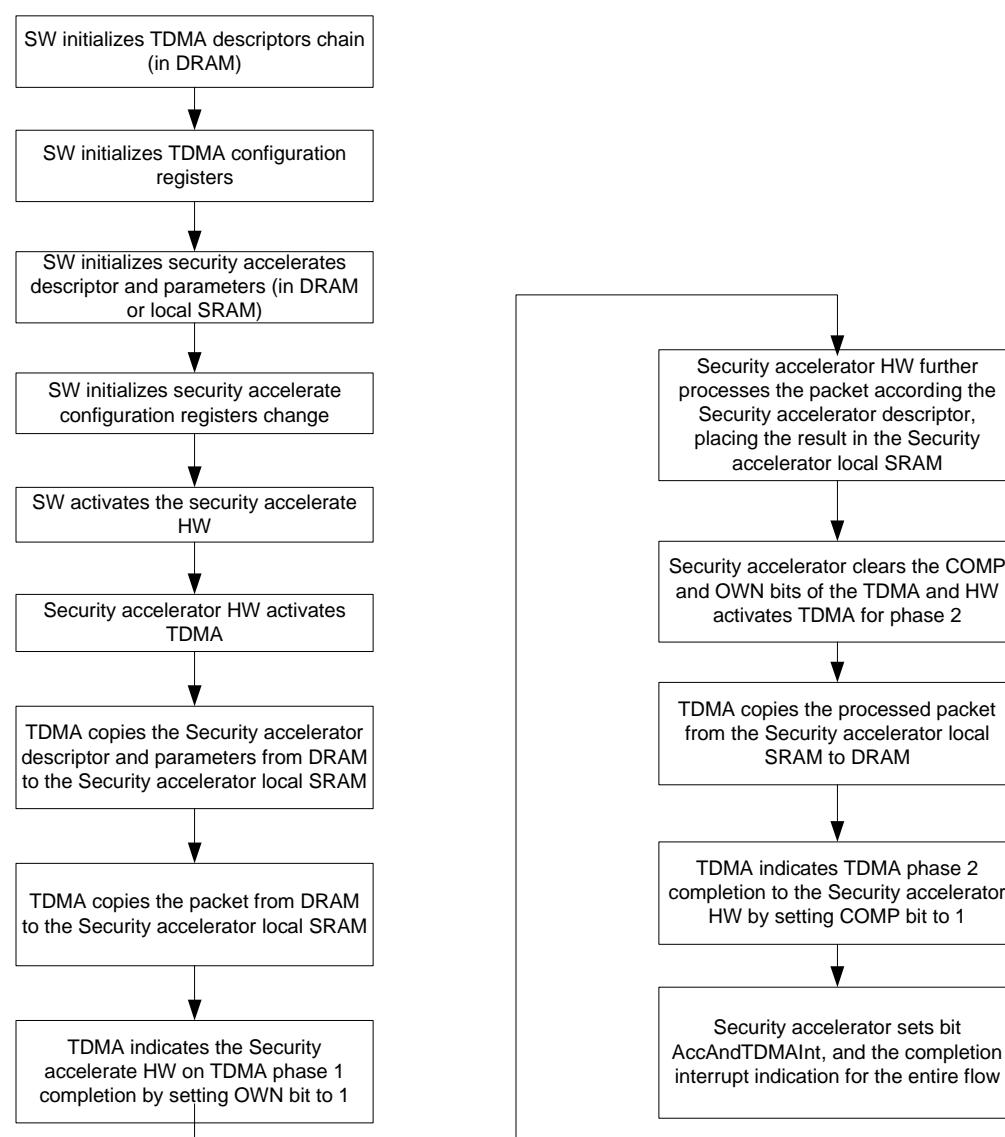
#### 10.4.3.1

#### Attaching TDMA to Security Accelerator for Enhanced Software Flow

To perform the following flow, set the [`<WaitForTDMA>`](#) field and the [`<ActivateTDMA>`](#) field in the Security Accelerator Configuration Register ([Table 560 p. 648](#)).

When the security accelerator operates in conjunction with the TDMA, the security accelerator can operate solely with external DRAM. As indicated by the flow shown in [Figure 47](#), the security accelerator has the capability to activate the TDMA, determine the status of the TDMA, and provide a single completion interrupt. The first five steps in the flow are performed by the software (SW) and the remaining steps are preformed by the hardware (HW).

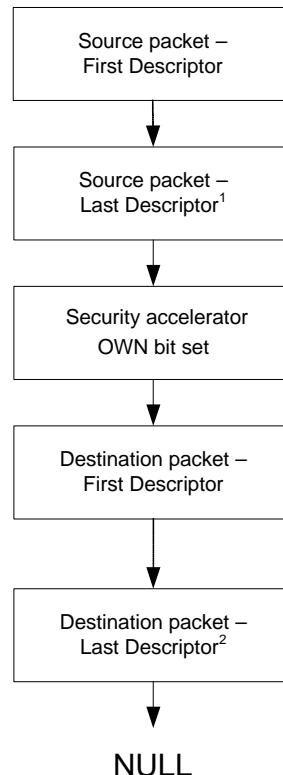
**Figure 47: Security Acceleration Flow for Packet Processing—Enhanced Mode**



[Figure 48](#) depicts the TDMA descriptors structure for security accelerator packet processing when in the security accelerator is operating in enhanced mode.

**Figure 48: TDMA Descriptors Structure for Security Accelerator Packet Processing in Enhanced Mode**

---



<sup>1</sup>This step only applies if there is more than one source packet descriptor.

<sup>2</sup>This step applies if there is more than one destination packet descriptor.

#### 10.4.3.2 Multi-Packet Chain Mode

The device security accelerator also supports Multi-Packet Chain mode. In this mode, multiple packets can be chained and processed by the hardware without software interference. To enable this mode, set the <MultiPacketChainMode> field in the Security Accelerator Configuration Register ([Table 560 p. 648](#)) to 1.

In Multi-Packet Chain mode, the software prepares multiple packets in memory. For each packet that needs to be processed, the software also prepares a set of descriptors, as described in [Figure 48, TDMA Descriptors Structure for Security Accelerator Packet Processing in Enhanced Mode, on page 193](#) except for a small modification. Instead of a NULL pointer at the last descriptor of the destination packet, the software includes pointer to the next source packet's first descriptor.

If the Multi-Packet Chain mode is enabled, after writing the encrypted packet N back to memory:

1. The TDMA fetches the first descriptor of source packet N+1.
2. The security accelerator sets the <AccAndTDMAInt\_CM> field in the Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register ([Table 556 p. 646](#)).



3. When there are no more packets to process (meaning, the destination packet last descriptor points to a NULL pointer), the security accelerator halts, and the Cryptographic Engine Interrupt Cause register's <AccAndTDMAInt> bit[7] is set.

### 10.4.3.3 Encryption Operation

#### Initialization

To initialize the encryption operation follow these steps:

1. Reads Encryption mode from the Security Accelerator Data Structure Dword 0—Configuration ([Table 50 p. 196](#)) and the configuration register of the selected cryptographic engine is written.
2. Reads the source and destination pointers from the Security Accelerator Data Structure Dword 1—Encryption Pointers ([Table 51 p. 197](#)).
3. Reads the number of bytes to be encrypted from the Security Accelerator Data Structure Dword 2—Encryption Data Length ([Table 52 p. 198](#)).
4. Reads the keys for encryption from the pointer specified in the Security Accelerator Data Structure Dword 3—Encryption Keys Pointer ([Table 53 p. 198](#)) and writes to the selected cryptographic engine.
5. When the mode selected is CBC, reads initial values from the pointer specified in the Security Accelerator Data Structure Dword 4—Encryption Initial Values Pointer ([Table 54 p. 198](#)). For DES/3DES, writes initial values to the engine.

#### Data Processing

Data processing must implement the following steps:

1. Reads data from the source pointer block by block for the specified data size (8 bytes for DES/3DES, 16 bytes for AES). If the size is not a multiple of block size, the last block is padded with zeros.
2. Feeds each block to the engine. When the engine finishes processing a block, the result is read and written to the location specified by the destination pointer.
3. When using AES CBC encryption, XORs the first block data with the initial values before the writing data to the engine. Each of the following blocks is first XORed with the result of the previous block processing before it is written to the engine.
4. When using AES CBC decryption, XORs the result of the first block processing with the initial values before writing it to the destination. Then, each processed block is XORed with the previous block of data before it is written to destination.

#### Termination

Termination is carried out as follows:

1. When using CBC encryption, writes the last block of the destination data to memory, according to the pointer specified in the Security Accelerator Data Structure Dword 4—Encryption Initial Values Pointer, overwriting the previous data.
2. When using CBC decryption, writes the last block of source data to memory, according to the pointer specified in the Security Accelerator Data Structure Dword 4—Encryption Initial Values Pointer.

#### 10.4.3.4 Authentication Operation

##### Initialization

To initialize the authentication operation follow these steps:

1. Reads Authentication mode from the Security Accelerator Data Structure Dword 0—Configuration ([Table 50 p. 196](#)) and writes the configuration register of the Authentication engine.
2. Reads the source pointer from the Security Accelerator Data Structure Dword 5—MAC Source Pointer ([Table 55 p. 199](#)).
3. Reads the Digest location pointer and the number of bytes in the message from the Security Accelerator Data Structure Dword 6—MAC Digest ([Table 56 p. 199](#)).
4. When the direction is “Decode”, reads the original digest first and stores it internally, then overwrites it with zeros.
5. When using the Hash-based Message Authentication Code (HMAC) operation, reads the inner initial values pointer from the Security Accelerator Data Structure Dword 7—MAC Initial Values Pointers ([Table 57 p. 199](#)), and writes the values to the IV registers of the authentication engine.

##### Data Processing

Data processing is carried out as follows:

1. Reads data from the source pointer block-by-block (64 bytes per block).
2. Pads the last chunk of data with a single 1 bit, as many zeros as needed, and the original message length. For HMAC modes, packet length is increased by 64 bytes to reflect the ipad string length.
3. For HMAC modes, reads the outer initial values pointer from the Security Accelerator Data Structure Dword 7—MAC Initial Values Pointers ([Table 57 p. 199](#)), and writes the values to the IV registers of the Authentication engine. The digest from the previous section is now used as the input data to the engine, padded again as specified, with the packet length now equal to the key length plus 64 bytes (which now reflects the opad string length).

##### Termination

Termination is carried out as follows:

1. Reads the resulting digest from the engine, and stores it at the location that has been read from the Security Accelerator Data Structure Dword 6—MAC Digest.
2. When the direction is “Decode”, compares the digest to the copy extracted from the original message.
  - If the digests are not identical, an indication bit is set in the Security Accelerator Status Register ([Table 561 p. 648](#)).
  - If the <StopOnDecodeDigestErr> field in the Security Accelerator Configuration Register ([Table 560 p. 648](#)) was set, the session is immediately aborted.

### 10.4.3.5 Security Accelerator in Fragmented and Non-Fragmented Modes

- Non-Fragmented mode** The Security Accelerator supports Non-Fragmented (multi-packet) mode. To enable this mode, set the [`<MultiPacketChainMode>`](#) field in the Security Accelerator Configuration Register ([Table 560 p. 648](#)) to 1. In this mode the Security accelerator continues to process packets until TDMA reach the NULL pointer (no more packets to process).
- Fragmented mode** The Security Accelerator also supports Fragmented (single packet) mode. To enable this mode, set the [`<MultiPacketChainMode>`](#) field in the Security Accelerator Configuration Register ([Table 560 p. 648](#)) to 0.
- In this mode, the Security Accelerator can process fragmented packets one at a time. Only single packets need to reside in the Security Accelerator local SRAM, while the total size of the packet is limited to 64 KB.
- Field Fragmentation mode is used to indicate if the current fragment is the first, middle, or last in the packet.
- The fragments must be inserted in order.

#### Processing the First Fragment

The first fragment is process as follows:

1. The operation starts in the same manner as in Non-fragmented mode. Finalization does not performed.
2. In encryption, IV is not stored.
3. In authentication, data is not padded, the outer operation in HMAC does not occur, and the digest is not written or compared.

#### Processing All of the Middle Fragments

The middle fragments are process as follows:

1. Initializations are not processed.
2. In encryption, keys and IVs are not loaded to the engines. Finalization is not performed, as for the first fragment.
3. In authentication, the digest is not read or cleared. Inner operation of HMAC does not occur.

#### Processing the Last fragment

The last fragment is process as follows:

1. Initializations are not processed, as for a middle fragment.
2. Finalization is processed as in Non-fragmented mode.

### 10.4.4 Security Accelerator Descriptor Data Structure

The descriptor data structure is described in [Table 50](#) through [Table 57](#).

**Table 50: Security Accelerator Data Structure Dword 0—Configuration**

Bits	Field	Function
1:0	Operation	00 = MAC only 01 = Cryptographic only 10 = MAC then cryptographic 11 = Cryptographic then MAC
3:2	Reserved	Reserved

**Table 50: Security Accelerator Data Structure Dword 0—Configuration (Continued)**

<b>Bits</b>	<b>Field</b>	<b>Function</b>
6:4	MacMode	100 = MD5 101 = SHA1 110 = HMAC-MD5 111 = HMAC-SHA1 All other combinations are reserved (no operation).
7	AuthResultLen	Authentication result length 0 = Full size (128 bit in MD-5, 160b in SHA-1) 1 = 96b
9:8	EncryptMode	00 = Reserved (no operation) 01 = DES 10 = 3DES 11 = AES
11:10	Reserved	Reserved
12	Direction	0 = Encode 1 = Decode
15:13	Reserved	Reserved
16	EncryptConfidentialityMode	0 = ECB 1 = CBC
19:17	Reserved	Reserved
20	3DESMODE	0 = EEE 1 = EDE Relevant only in 3DES encryption mode.
31:30	FragMode	Fragmentation mode 00 = Not fragmented 01 = First Fragment in packet 10 = Last Fragment in packet 11 = Middle Fragment in packet

**Table 51: Security Accelerator Data Structure Dword 1—Encryption Pointers**

<b>Bits</b>	<b>Field</b>	<b>Function</b>
10:0	EncrypSourceDataPtr	Pointer to the first Dword of data to encrypt (Dword aligned) Bits [2:0] must be 0.
15:11	Reserved	Reserved
26:16	EncrypDesDataPtr	Pointer to the first Dword of encrypted data (Dword aligned) Bits [18:16] must be 0.
31:27	Reserved	Reserved

**Table 52: Security Accelerator Data Structure Dword 2—Encryption Data Length**

Bits	Field	Function
10:0	EncrypDataLen	Numbers of bytes to encrypt The length should be a multiple of encryption block size (8 bytes for DES/3DES, 16 bytes for AES). Bits [2:0] must be 0. Bit [3] (in AES only) is reserved and assumed to be 0 regardless of programming.
31:11	Reserved	Reserved

**Table 53: Security Accelerator Data Structure Dword 3—Encryption Keys Pointer**

Bits	Field	Function
10:0	EncrypKeyPointer	Pointer to an array (EKey) of Dwords that contains the encryption key (Dword aligned): <ul style="list-style-type: none"> <li>EKey[0] = Key 0 low of DES/3DES / Key column 0 of AES 128/192/256</li> <li>EKey[1] = Key 0 high of DES/3DES / Key column 1 of AES 128/192/256</li> <li>EKey[2] = Key 1 low of 3DES / Key column 2 of AES 128/192/256</li> <li>EKey[3] = Key 1 high of 3DES / Key column 3 of AES 128/192/256</li> <li>EKey[4] = Key 2 low of 3DES / Key column 4 of AES 192/256</li> <li>EKey[5] = Key 2 high of 3DES / Key column 5 of AES 192/256</li> <li>EKey[6] = Key column 6 of AES 256</li> <li>EKey[7] = Key column 7 of AES 256</li> </ul> Bits [2:0] must be 0.
31:11	Reserved	Reserved

**Table 54: Security Accelerator Data Structure Dword 4—Encryption Initial Values Pointer**

Bits	Field	Function
10:0	EncryptIVPointer	Pointer to an array (EIV) of Dwords that contains the encryption initial values (Dword aligned): <ul style="list-style-type: none"> <li>EIV[0] = IV low of DES/3DES / IV 0 of AES</li> <li>EIV[1] = IV high of DES/3DES / IV 1 of AES</li> <li>EIV[2] = IV 2 of AES</li> <li>EIV[3] = IV 3 of AES</li> </ul>
15:11	Reserved	Reserved
26:16	EncryptIVBufPointer	<ul style="list-style-type: none"> <li>In Encryption mode, in the encryption direction:            Before encryption starts, the security accelerator copies the contents of &lt;EncryptIVPointer&gt; (bit[15:0] in same register) to &lt;EncryptIVBufPointer&gt;.</li> <li>In Encryption mode, in the decryption direction:            Before decryption starts, the security accelerator copies the contents of &lt;EncryptIVBufPointer&gt; to &lt;EncryptIVPointer&gt;.</li> </ul> Bits [18:16] must be 0.

**Table 54: Security Accelerator Data Structure Dword 4—Encryption Initial Values Pointer (Continued)**

Bits	Field	Function
31:27	Reserved	Reserved

**Table 55: Security Accelerator Data Structure Dword 5—MAC Source Pointer**

Bits	Field	Function
10:0	MACSourceDataPointer	Pointer to the first Dword of data to MAC (Dword aligned) Bits [2:0] must be 0.
15:11	Reserved	Reserved
31:16	TotalMacDataLength	In MAC Non Fragment mode, this field should be equal to MacDataLength (see <a href="#">Table 56</a> ). In the last MAC fragment, it should be equal to the total data lengths of all the packet fragments.

**Table 56: Security Accelerator Data Structure Dword 6—MAC Digest**

Bits	Field	Function
10:0	MACDigestPointer	Byte location in which digest is stored during encoding or should be stored during decoding Bits [2:0] must be 0.
15:11	Reserved	Reserved
26:16	MACDataLength	Numbers of bytes to MAC
31:27	Reserved	Reserved

**Table 57: Security Accelerator Data Structure Dword 7—MAC Initial Values Pointers**

Bits	Field	Function
10:0	MACInnerIVPointer	Pointer to an array (MIIV) of Dwords that contains the MAC inner initial values (Dword aligned) These values are the outcome of the hash function operation over the 64-byte string that equals the bitwise XOR between the key padded with zeros and the ipad string. <ul style="list-style-type: none"> <li>• MIIV[0] = Inner IV 0 of HMAC-MD5/HMAC-SHA1</li> <li>• MIIV[1] = Inner IV 1 of HMAC-MD5/HMAC-SHA1</li> <li>• MIIV[2] = Inner IV 2 of HMAC-MD5/HMAC-SHA1</li> <li>• MIIV[3] = Inner IV 3 of HMAC-MD5/HMAC-SHA1</li> <li>• MIIV[4] = Inner IV 4 of HMAC-SHA1</li> </ul> Bits [2:0] must be 0.
15:13	Reserved	Reserved

**Table 57: Security Accelerator Data Structure Dword 7—MAC Initial Values Pointers (Continued)**

Bits	Field	Function
26:16	MACOuterIVPointer	<p>Pointer to an array (MOIV) of Dwords that contains the MAC outer initial values (Dword aligned). These values are the outcome of the hash function operation over the 64-byte string that equals the bitwise XOR between the key padded with zeros and the opad string.</p> <ul style="list-style-type: none"> <li>• MOIV[0] = Outer IV 0 of HMAC-MD5/HMAC-SHA1</li> <li>• MOIV[1] = Outer IV 1 of HMAC-MD5/HMAC-SHA1</li> <li>• MOIV[2] = Outer IV 2 of HMAC-MD5/HMAC-SHA1</li> <li>• MOIV[3] = Outer IV 3 of HMAC-MD5/HMAC-SHA1</li> <li>• MOIV[4] = Outer IV 4 of HMAC-SHA1</li> </ul> <p>Bits [18:16] must be 0.</p>
31:27	Reserved	Reserved

## 10.5 TDMA Controller

The device has one independent TDMA engine. The TDMA engine optimizes system performance by moving large amounts of data without significant CPU intervention.

The TDMA engine can move data between the DDR memory and the internal SRAM, and from the internal SRAM to the DDR memory. It can transfer a single data buffer of up to 2 KB. It can also run in Chain mode. That mode assigns a unique descriptor to each buffer.

As long as TDMA is active, any software read access to the local SRAM of the security accelerator is forbidden. A software read access is not expected. See the software flow description in [Section 10.4.3](#).

### 10.5.1 Functional Description

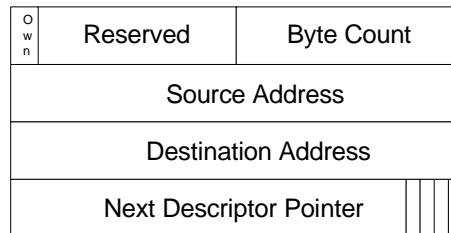
When fetching data into the SRAM, the data is read from the source through the Mbus and written directly into the internal SRAM.

When storing data from the SRAM, the data is read from the SRAM and written to the DDR memory through the Mbus.

### 10.5.2 TDMA Descriptors

The TDMA Descriptor consists of four 32-bit registers.

**Figure 49: TDMA Descriptors**



**Table 58: TDMA Descriptor Definitions**

TDMA Descriptor	Definition
Byte Count	Number of bytes of data to transfer. The maximum number of bytes to which the TDMA controller can be configured to transfer is 64 KB-1 (16-bit register). This register decrements at the end of every burst of transmitted data from the source to the destination. When the byte count register is 0, the TDMA transaction is finished or terminated.
Source Address	Bits [31:0] of the TDMA source address.
Destination Address	Bits [31:0] of the TDMA destination address.
Pointer to the Next Descriptor	Bits [31:0] of the TDMA Next Descriptor address for chained operation. The descriptor must be 16 sequential bytes located at a 16-byte aligned address (bits [3:0] are 0). <b>NOTE:</b> This descriptor is used only used when the TDMA is configured to Chained mode.
Own	The <Own> field in the TDMA Byte Count Register ( <a href="#">Table 574 p. 655</a> ) acts as an ownership bit. <ul style="list-style-type: none"> <li>• If set to 1, the descriptor is owned by the device TDMA.</li> <li>• If set to 0, the descriptor is owned by the CPU. Once the CPU prepares a buffer to be transferred, it sets the ownership bit. This indicates that the buffer is owned by the TDMA.</li> </ul>



The source or destination address must be configured to the internal SRAM address space.

### 10.5.3 TDMA Address Decoding

The TDMA shares four address windows. Each address window can be individually configured.

For each TDMA transaction, the TDMA engine first compares the address (source, destination, or descriptor) against its address decoding registers. Each window can be configured to a different target interface. Address comparison is completed to select the correct target interface.

## 10.5.4 TDMA Control

The TDMA has its own unique control register, where certain TDMA modes are programmed. The following sections describe the bits for each field in the control registers.

### 10.5.4.1 Burst Limit

The TDMA byte count is divided into small bursts.

The burst limit can be 32 or 128 bytes. The limit determines the burst length of the TDMA transaction against the source and destination. There are separate Burst Limit parameters for source and destination.

The burst limit setting is affected by the source and destination characteristics, as well as by system bandwidth allocation considerations.

**Note**

Regardless of the burst limit setting, the fetch of a new descriptor is always a 16-byte burst. Therefore, descriptors cannot be located in devices that do not support such bursts.

### 10.5.4.2 Chain Mode

When the [\*\*<ChainMode>\*\*](#) field in the Control Register ([Table 573 p. 654](#)) is set to 0, Chained mode is enabled.

In Chain mode, at the completion of one buffer transfer, the Pointer to Next Descriptor provides the address of the next TDMA descriptor. If it is a NULL pointer (value of 0), it indicates that this is the last descriptor in the chain. If not, the TDMA engine fetches the new descriptor, and starts transferring the new buffer.

Fetching of the next descriptor can be forced by bit [13] the [\*\*<FetchND>\*\*](#) field in the Control Register ([Table 573 p. 654](#)).

Setting the [\*\*<FetchND>\*\*](#) to 1 forces a fetch of the next descriptor based on the value in the Pointer to Next Descriptor register. This bit is reset to 0 after the fetch of the new descriptor is complete. Setting [\*\*<FetchND>\*\*](#) is not allowed if the next descriptor pointer equals NULL.

The first descriptor of a chain can be set directly by programming the TDMA registers, or can be fetched from memory, using the [\*\*<FetchND>\*\*](#) bit. If fetched from memory, the next descriptor address must be first written to the Next Descriptor Pointer register of the TDMA. The TDMA then must be enabled by setting bit [12] the [\*\*<TDMAEn>\*\*](#) field in the Control Register ([Table 573 p. 654](#)) to 1 (see [Section 10.5.4.3, TDMA Activation, on page 203](#)) and by setting the [\*\*<FetchND>\*\*](#) field to 1.

When the TDMA transfer is completed, a TDMA completion interrupt is set. When running in Chain mode, an interrupt is asserted only upon the completion of the last descriptor byte count.

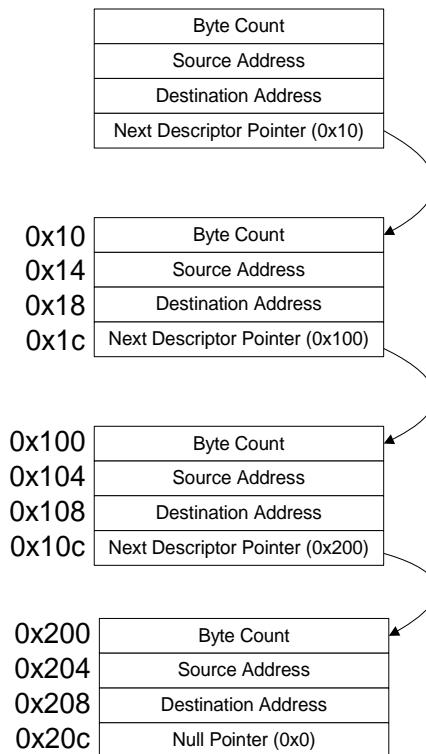
If the [\*\*<ChainMode>\*\*](#) field in the Control Register ([Table 573 p. 654](#)) is set to 1, Chained mode is disabled and the Pointer to Next Descriptor register is not loaded at the completion of the TDMA transaction.

**Note**

In Non-chained mode, the Byte Count, Source, and Destination registers must be initialized prior to enabling the TDMA.

Figure 50 shows an example of a TDMA descriptor chain.

**Figure 50: Chained Mode TDMA**



#### 10.5.4.3 TDMA Activation

Software TDMA activation is performed using the <TDMAEn> field in the Control Register (Table 573 p. 654) as follows:

- When set to 0, the TDMA is disabled.
- When set to 1, the TDMA is initiated based on the current setting loaded in the TDMA descriptor (i.e., byte count, source address, and destination address).

An active TDMA can be temporarily stopped by clearing the <TDMAEn> bit. Then the active TDMA can be continued from the point where it stopped by setting the <TDMAEn> bit back to 1.

Clearing the <TDMAEn> bit during a TDMA operation does not guarantee an immediate TDMA pause. The TDMA engine must complete transferring the last burst it was processing before it pauses. Software can monitor the TDMA status by reading the <TDMAAct> field (bit [14]).

The <TDMAAct> bit is read only.

- If set to 0, the TDMA is not active.
- If set to 1, the TDMA is active.

In Non-chain mode, this bit is de-asserted when the byte count reaches zero.

In Chain mode, this bit is de-asserted when the pointer to the next descriptor is NULL and the byte count reaches zero.

#### 10.5.4.4 Source and Destination Addresses Alignment

The TDMA implementation maintains aligned accesses to both source and destination.



If source and destination addresses have different alignments, the TDMA performs multiple reads from the source to execute a write of full burst limit to the destination. For example, if the source address is 0x64 and the destination address is the internal SRAM:

1. The Burst limit of the source is set to 32 bytes.
2. The byte count is 64.
3. The TDMA perform three reads from the source:
  - a) 28 bytes from address 0x64 (due to the Mbus transfer limit).
  - b) 32 bytes from address 0x80 (due to the source burst limit).
  - c) 4 bytes from address 0xA0.

This implementation guarantees that all reads from the source and all writes to the destination have all byte enables asserted (except for the buffer start/end, in case they are not aligned). This is especially important when the source device does not tolerate reads of extra data (destructive reads) or when the destination device does not support write byte enables.

#### 10.5.4.5

#### Descriptor Ownership

A typical application of Chain mode TDMA involves the CPU preparing a chain of descriptors in memory and then preparing buffers to move the descriptors from source to destination.

The [\*\*<Own>\*\*](#) field in the TDMA Byte Count Register ([Table 574 p. 655](#)) (bit [31]) is assigned as an ownership bit.

- If set to 1, the descriptor is owned by the device TDMA.
- If set to 0, it is owned by the CPU. Once the CPU prepares a buffer to be transferred, it sets the ownership bit. This indicates that the buffer is owned by the TDMA.

An attempt by the TDMA to fetch a descriptor that is owned by the CPU (which means the CPU did not prepare a new buffer yet) results in an interrupt assertion, and the TDMA stops.

#### 10.5.5

#### TDMA Interrupts

The TDMA interrupts are registered in the Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register ([Table 556 p. 645](#)). Upon an interrupt event, the corresponding cause bit is set to 1. It is cleared upon a software write of 0.

The following interrupt events are supported:

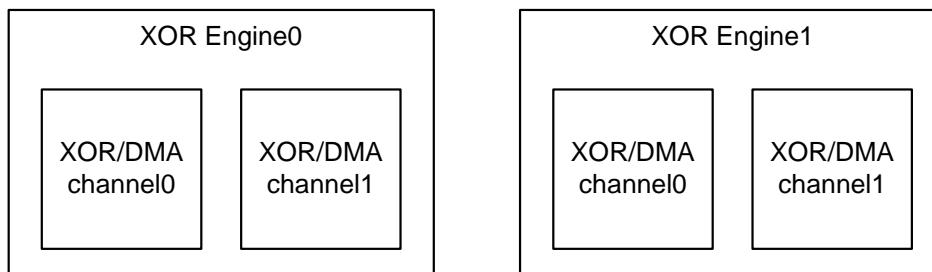
- TDMA completion
- TDMA descriptor ownership violation

# 11 XOR Engine

This device integrates two XOR engines. Each one contains two XOR/DMA channels (for a total of four XOR/DMA channels), as shown in [Figure 51](#).

**Figure 51: Schematic Diagram of the Two XOR Engines**

---



The section describes a single XOR engine.

The XOR engine is a generic acceleration engine for storage applications that provides a low latency, high throughput XOR calculation capabilities, enabling CPU XOR calculation off-loading in various RAID implementations. In addition, the engine provides iSCSI CRC32C calculation, DMA operation, memory initialization, support.

The XOR engine enables PC/Server manufacturers (ROM), Internal RAID Controllers and External RAID systems to speed up overall system performance.

XOR engine features:

- Two separate channels for enabling concurrent operation (for example, concurrent XOR and iSCSI CRC32C calculations)
- 1 KB temporary result store queue per channel. Arranged as 128 X 8B buffer
- Support packing/unpacking of unaligned data transfers
- XOR calculation for up to eight data block sources
- Data block size up to 16 MB
- Programmable maximum burst size on read and write
- Descriptor chain mechanism
- Hot insertion of new descriptors to chain
- iSCSI CRC32C calculation that is compliant with IPS iSCSI version 13 draft
- DMA operation
- Memory initialization support
- Write access protection of configuration registers

## 11.1 Theory of Operation

XOR engine has four main operation modes:

- XOR calculation Mode (XOR)
- iSCSI CRC32C Calculation Mode (CRC)

- DMA Operation Mode (DMA)
- Memory initialization Mode (MemInit)

The engine has two independent channels. Each channel can be configured to one of the operation modes at a time. The operation mode is defined through the <OperationMode> field in the XOR Engine [0..1] Configuration (XExCR) Register (n=0–1) ([Table 588 p. 664](#)). In the XOR, CRC and DMA operation modes, the XOR engine is controlled by chain descriptors and responds to similar activation scheme. These modes differ only in the interpretation of the chain descriptor fields. In Memory Initialization (MemInit) mode, the XOR engine responds to different activation schemes. It is controlled by programming internal registers directly. On all operation modes, XOR engine uses the same address decoding scheme.

Upon startup, the two XOR channels are in an inactive state and can be configured to any operation mode (XOR, CRC, DMA, or MemInit). After being configured, the XOR engine channel can be activated. It can be stopped or paused by software at any time. After stopped by software, the engine re-enters inactive state and can be configured to another operation mode and re-activated. This also applies if the XOR engine channel finished the operation (reached End Of Chain) without being stopped by the software. Again, the engine re-enters an inactive state and can be configured to another operation mode, and re-activated. After paused by software, the XOR engine channel suspends the current operation at the earliest opportunity. Upon activating the channel again, it resumes executing the same operation.

The two XOR engine channels are independent in their operation modes. The only exception is that both engines must not be configured to MemInit operation modes. These modes share hardware resources.

**Note**

Attempting to change the channels operation mode during a pause will result in unexpected behavior.

### 11.1.1

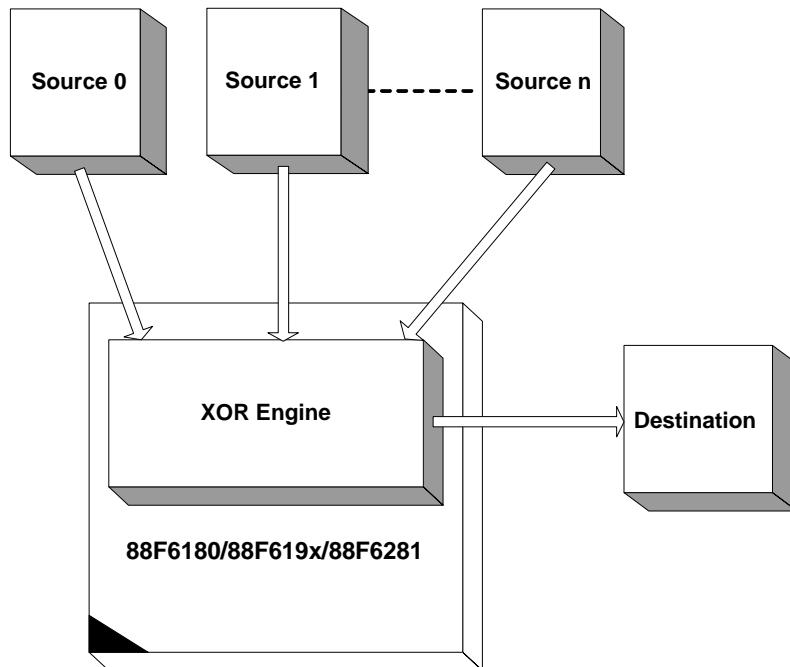
### XOR Operation

The XOR engine enables block XOR calculation in hardware. It performs the XOR operation on multiple blocks of source (incoming) data and stores the result back in a destination block. The source and destination addresses are specified through a chain descriptor.

[Figure 52](#) shows how the XOR operation works with multiple blocks of source (incoming) data and stores the result back in a destination block.

**Figure 52: XOR Operation with Multiple Incoming Data Blocks**

---



**destination = (source 0) xor (source 1) ... xor (source n)**  
**number of sources can be up to 8 (n = 7)**

The parameters of the XOR operation are configured by writing the relevant information to a chain descriptor. The relevant parameters consist of source addresses, destination addresses, the number of bytes to transfer, and various control information.

When activated in XOR mode, the XOR engine fetches the first descriptor and starts performing the XOR operation according to its parameters. After finishing the operation, the XOR engine closes the descriptor by writing back the status word to the descriptor and returning the ownership of it to the CPU. The XOR engine checks whether it reached the end of the descriptor chain. If it is the end, the engine enters an inactive state and waits to be re-activated by software. If it did not reach the end of the chain, it progresses to the next descriptor, and so on.

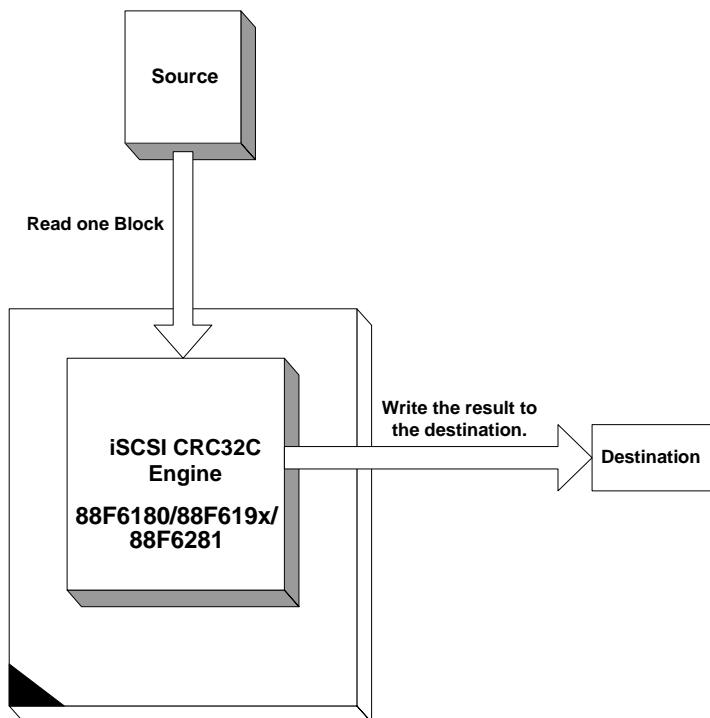
The basic XOR operation algorithm is as follows:

1. Read data from the first enabled source block to the internal buffer.
2. Read from the second enabled source block and calculate XOR with the data from the internal buffer. The intermediate result is stored in the internal buffer.
3. Step 2 is repeated for the rest of the source buffers until all enabled source buffers are handled (up to 8 sources).
4. Write the internal buffer to the destination buffer.
5. Repeat stages 1-4 until the descriptor byte count is reached.

## 11.1.2 iSCSI CRC32C Calculation

In addition to the XOR operation, the XOR engine also provides iSCSI CRC32C calculation capabilities. It performs iSCSI CRC32C calculation on a source block and writes the result back to a descriptor, as shown in [Figure 53](#).

**Figure 53: XOR iSCSI CRC32C Operation**



The source blocks are specified through a chain of descriptors. Parameters of the iSCSI CRC32C operation are configured in the same way as in XOR operation - writing the relevant information to a chain descriptor. The relevant parameters consist of source addresses, destination address, size of source block, and 'last block in CRC source chain' indication.

The CRC source block in CRC mode can be scattered over a few target blocks. It can be in non-consecutive memory spaces and even in different interfaces. The CRC Source block is represented by a source block chain of descriptors. Every descriptor represents one consecutive section of the CRC source block.

When activated in CRC mode (see the [`<OperationMode>`](#) field in the XOR Engine [0..1] Configuration (XExCR) Register ( $n=0\text{--}1$ ) ([Table 588 p. 664](#)) is set to CRC), XOR engine executes iSCSI CRC32C calculation according to the source block chain. The last descriptor in a source block chain is marked as 'last'. After the calculation is finished, the 32-bit result is written to the CRC source block chain's last descriptor.

The chain descriptor operation is slightly different in CRC-32 mode than in XOR mode. In XOR mode, every descriptor stands for a self contained XOR operation. In CRC mode, one CRC operation (one CRC source block) can be represented by a number of chain descriptors, thus enabling concatenation of a few data sources to one block, for CRC calculation. The descriptor chain is constructed of several source block chains.

The Basic iSCSI CRC32C operation is as follows:

1. Read data from the source block to internal buffer.
2. Calculate iSCSI CRC32C on the internal buffer and store intermediate result.
3. Repeat step 1-2 for the rest of the source block, until all of the blocks are processed.
4. Repeat steps 1-3 for the rest of the blocks in the source block chain.
5. Write result to the last descriptor.

### 11.1.3

### DMA Operation

The XOR engine also provides generic DMA capabilities—copying of a source block to a destination block. The source blocks are specified through a chain of descriptors. Parameters of the DMA operation are configured in the same way as in XOR operation - by writing the relevant information to a chain descriptor. The relevant parameters consist of source address, destination address, and size of source block.

When activated in DMA mode, the XOR engine fetches the first descriptor and starts performing the DMA operation according to its parameters. After finishing the operation, the XOR engine closes the descriptor by writing back status word to the descriptor and returning the ownership of it to the CPU. The XOR engine checks whether it reached the end of the descriptor chain. If the end has been reached, the engine enters an inactive state and waits to be re-activated by the software. If it did not reach the end of the chain, it progresses to the next descriptor, and so on.

### 11.1.4

### Memory Initialization

The XOR engine provides memory value initialization capabilities. It performs writes of pre-defined values to a destination memory block. The destination address and block size are specified directly by internal registers. The relevant parameters consist of a destination address, an initial memory value, and a size of the destination block.



**Note**

Only one channel may be configured to MemInit mode at a time. If both channels are configured to MemInit mode, engine behavior is unpredictable.

---

When activated in MemInit mode, the XOR engine executes a memory initialization operation according to the relevant internal registers. It will write the 64-bit initial value, specified by the XOR Engine Initial Value Low (XEIVRL) Register ([Table 599 p. 670](#)) and XOR Engine Initial Value High (XEIVRH) Register ([Table 600 p. 670](#)), in a cyclical method to the destination block. Upon completion of the memory initialization operation, the XOR engine channel asserts the EOC interrupt.

## 11.2

## Descriptor Chain

### 11.2.1

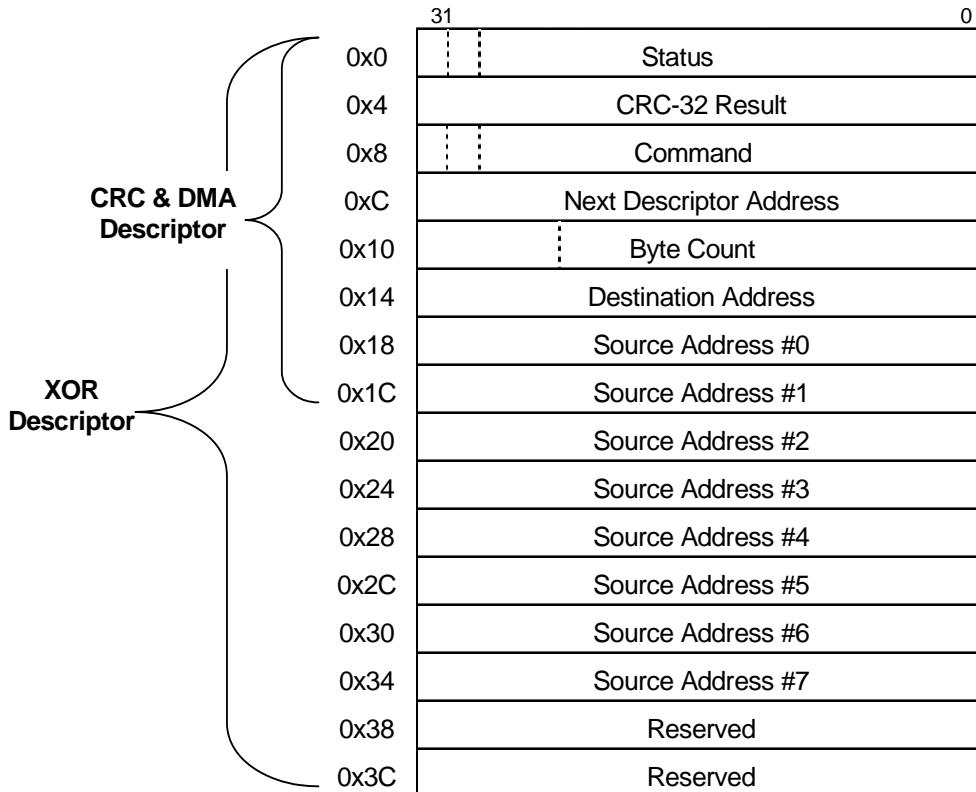
### Descriptor Format

The XOR engine descriptor format supports 32-bit addressing. In XOR mode, the descriptor consists of sixteen 32-bit words which totals the 64B size of each descriptor. In CRC and DMA modes, only the upper 32B of the descriptor is needed. Therefore, the descriptor consists of eight 32-bit words, totalling to a 32B size for each descriptor (see [Figure 54](#)).

By fetching a descriptor from memory, the XOR engine gets all the information about the next operation to be performed. When the XOR engine finishes the operation associated with a descriptor, it closes the descriptor by updating the status word. This means the operation completed successfully and returns the ownership of the descriptor to the CPU.

**Note**

The chain descriptor operation is valid only in XOR, CRC and DMA operation modes. In MemInit mode, the XOR engine receives the operation data directly from its internal registers.

**Figure 54: XOR Descriptor Format**

The XOR descriptor must be 64 Bytes aligned (Address[5:0]=0). The CRC and DMA descriptors must be 32 Bytes aligned (Address[4:0]=0). There are no restrictions on source or destination data block alignment. Source and destination blocks can have different alignments. Different source blocks can have different alignments as well.

**Table 59: Descriptor Status Word Definition**

Bit	Field	Description
29:0	Reserved	Reserved.
30	Success	Successful descriptor execution indication. Indicates whether the operation completed successfully. 0 = Completed unsuccessfully - Transfer terminated before the whole byte count was transferred. 1 = Completed successfully - The whole byte count transferred. That field is updated upon closing the descriptor
31	Own	Ownership Bit Indicates whether the descriptor is owned by the CPU or the XOR engine. 0 = CPU owned. 1 = XOR engine owned. That field is updated upon closing a descriptor - XOR engine gives back ownership to the CPU by clearing the own bit.

**Table 60: Descriptor CRC-32 Result Word Definition**

Bit	Field	Description
31:0	CRCresult	Result of CRC-32 calculation Valid only in the last descriptor of a CRC source block chain, after it was closed by the XOR engine. <b>NOTE:</b> Valid only in CRC mode.

**Table 61: Descriptor Command Word Definition**

Bit	Field	Description
0	Src0Cmd	Specifies the type of operation to be carried out on the data pointed by SA#0 (Source Address 0 word of the descriptor). 0x0 = Null Command - Data from Source will be disregarded in the current descriptor operation. 0x1 = XOR Command - Data from source will be transferred and will be significant in the XOR calculation. <b>NOTE:</b> Relevant only on XOR operation mode. Disregarded in all other operation modes.
1	Src1Cmd	Specifies the type of operation to be carried out on the data pointed by SA#1 (Source Address #1 word of the descriptor). <b>NOTE:</b> Relevant only on XOR operation mode. Disregard in all other operation modes.
2	Src2Cmd	Specifies the type of operation to be carried out on the data pointed by SA#2 (Source Address #2 word of the descriptor). <b>NOTE:</b> Relevant only on XOR operation mode. Disregard in all other operation modes.
3	Src3Cmd	Specifies the type of operation to be carried out on the data pointed by SA#3 (Source Address #3 word of the descriptor). <b>NOTE:</b> Relevant only on XOR operation mode. Disregard in all other operation modes.
4	Src4Cmd	Specifies the type of operation to be carried out on the data pointed by SA#4 (Source Address #4 word of the descriptor). <b>NOTE:</b> Relevant only on XOR operation mode. Disregard in all other operation modes.

**Table 61: Descriptor Command Word Definition (Continued)**

<b>Bit</b>	<b>Field</b>	<b>Description</b>
5	Src5Cmd	Specifies the type of operation to be carried out on the data pointed by SA#5 (Source Address #5 word of the descriptor). <b>NOTE:</b> Relevant only on XOR operation mode. Disregard in all other operation modes.
6	Src6Cmd	Specifies the type of operation to be carried out on the data pointed by SA#6 (Source Address #6 word of the descriptor). <b>NOTE:</b> Relevant only on XOR operation mode. disregarded in all other operation modes.
7	Src7Cmd	Specifies the type of operation to be carried out on the data pointed by SA#7 (Source Address #7 word of the descriptor). <b>NOTE:</b> Relevant only on XOR operation mode. Disregard in all other operation modes.
29:8	Reserved	Reserved
30	CRCLast	Indicated last descriptor in a CRC-32 calculation chain. 0 = Not last descriptor in a CRC calculation chain. 1 = Last descriptor in a CRC calculation chain. When closing the descriptor, the XOR engine writes the CRC result to its CRC-32 Result word. The next descriptor in the descriptor chain initiates a new CRC calculation. If the source block is represented by one descriptor only, it should be marked as last. <b>NOTE:</b> Relevant only in CRC operation mode.
31	EODIntEn	End Of Descriptor Interrupt Enable. Specifies if the EOD interrupt is asserted upon closure of that descriptor. 1 - EOD Enabled. 0 - EOD Disabled.

**Table 62: Descriptor Next Descriptor Address Word**

<b>Bits</b>	<b>Field</b>	<b>Description</b>
31:0	NDA	Next descriptor address pointer XOR Mode: NDA must be 64-byte aligned (bits[5:0] must be 0x0). CRC/DMA Mode: NDA must be 32-byte aligned (bits[4:0] must be 0x0). NDA field of the last descriptor of a descriptor chain must be NULL.

**Table 63: Descriptor Byte Count Word**

<b>Bit</b>	<b>Field</b>	<b>Description</b>
23:0	ByteCount	XOR mode: Size of source and destination blocks in bytes. CRC mode: Size of source block part represented by the descriptor. DMA mode: Size of source and destination block in bytes. Minimum blocks' size: 16B. Maximum blocks' size: 16MB-1
31:24	Reserved	Reserved.

**Table 64: Descriptor Destination Address Word**

<b>Bits</b>	<b>Field</b>	<b>Description</b>
31:0	DA	Destination Block address pointer XOR Mode: Destination Block address pointer. CRC mode: Not used. DMA mode: Destination Block address pointer.

**Table 65: Descriptor Source Address #N Words**

Bits	Field	Description
31:0	SA#0 Source Address #0	source block #0 address pointer. XOR Mode: Source Block #0 address pointer. CRC mode: Address pointer to part of source block represented by the descriptor. DMA Mode: Source Block address pointer.
31:0	SA#N [N=1..7] Source Address #N	source block #N address pointer. XOR mode: Source Block #N address pointer. CRC mode: Not used. DMA mode: Not used.

## 11.3 Address Decoding

The XOR engine has eight address windows that can be individually configured. With each transaction, the XOR engine first compares the address (source, destination, or descriptor) against the address decoding registers. Each window can be configured to a different target interface. Address comparison is done to select the correct target interface. If the address does not match any of the address windows (no hit), an interrupt is generated and the XOR engine is stopped. If the address matches more than one address window (multiple hit), an interrupt is generated and the XOR engine is stopped.

For the XOR engine to avoid accessing forbidden address space (due to a programming bug), each channel uses access protection logic that prevents it from read/write access to specific address windows. In case of access violation, the operation is stopped, the channel becomes inactive, and an interrupt is asserted.

### 11.3.1 Target Interface

Source data blocks, destination data block, and descriptors can be targeted to any of the chip Interfaces. The unique attributes of each interface are configured per address window through the XOR engine BARs (Base Address Registers) (see [Appendix A.11.1, XOR Engine Address Decoding Registers, on page 659](#)).

### 11.3.2 64-bit Addressing

Four of the eight address windows have an upper 32-bit address register. These are used for accessing interfaces that support more than 4 GB of address space. The address generated on the interface is composed of the 32-bit address issued by the XOR engine, if it hits the relevant address window, concatenated with the High Remap register.

The XOR engine address decoder can map a total of up to a 4 GB address space.

### 11.3.3 Address Override

The XOR engine also supports an address override feature. Each of the sources, destination, or next descriptor addresses of each channel, can be configured to use the override feature by using the XOR Engine [0..1] Address Override Control (XEAOCR) Register (n=0–1) ([Table 586 p. 661](#)). When override is enabled and the respective pointer field is set to 0x0, the transaction target interface, and attributes, are taken from the Base Address register 0 (XEBA0) and the upper 32 bits of the 64 bit address are taken from High Address Remap Register 0 (XEHARR0). When set to 0x1, these items are taken from XEBAR1 and XEHARR1 respectively, and so on for pointer values of 0x2 and 0x3.

This address override feature, enables additional address de-coupling. For example, it allows the use of the same source and destination addresses, while the source is targeted to one interface and destination to a different interface.

**Note**

When using the address override option, no window access control is performed. For example if override is set to SA#5 of channel 1, any address that is specified in the descriptor as SA#5 is directly accessed without any window access control check.

## 11.4 Arbitration

The two XOR engines share the same Mbus port.

The arbitration is performed in two stages.

1. Arbitration between the two XOR channels within each engine.
2. Arbitration between the chosen XOR channel of XOR engine0, and the chosen XOR channel of XOR engine1.

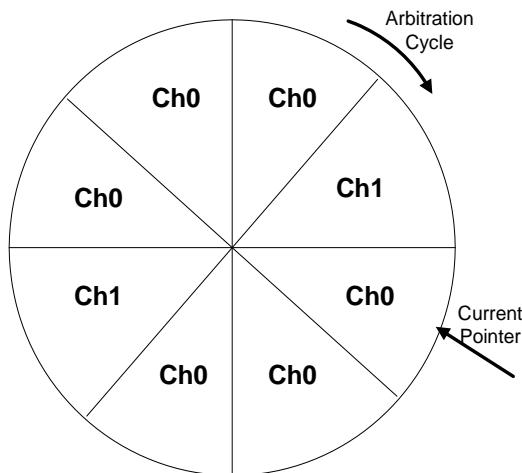
### 11.4.1 Arbitration Between XOR Engines 0 and 1

A fixed round robin arbitration is performed between the two XOR engines. If both are active, the effective bandwidth allocation in congestion conditions is approximately 50% for each unit.

### 11.4.2 Arbitration Between XOR Engine Channels

The two XOR engine channel use the same Mbus port. A programmable weighted round robin arbiter controls the bandwidth allocation for each channel on the Mbus port. Each channel can be configured to have a different bandwidth allocation. [Figure 55](#) shows an example of the arbitration cycle.

**Figure 55: Programmable Channel Pizza Arbiter**



The pizza arbiter has eight slices, each slice can be configured to serve a different channel. In [Figure 55](#), channel0 gets 75% of the bandwidth, and channel1 25% each. At each clock cycle, the arbiter samples all channels requests and gives the bus to the next channel according to the "pizza" setting.

The bandwidth allocation is flexible. The arbiter influences the bandwidth allocation only when two ports demand Mbus port service at the same time (congestion conditions). If only one channel demands Mbus bandwidth, the channel receives 100% of the bandwidth. For example, in [Figure 55](#), only Channel0 is active and so it gets 100% of the Mbus port bandwidth.

## 11.5 XOR Engine Programming

### 11.5.1 Programming in XOR, CRC, and DMA modes

The XOR engine operation is similar in XOR, CRC, and DMA modes. All of these modes use descriptor chains. The modes differ in their configuration parameters and in their chain descriptor size and field interpretation.

#### 11.5.1.1 Activation on Startup

To activate an XOR engine for the first time after Reset de-assertion, the software must perform the following sequence:

1. Confirm that the relevant XOR engine channel is inactive (the `<XEstatus>` field in the XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) ([Table 589 p. 666](#)) is set to 0).
2. Initialize the relevant XOR engine channel configuration through the XOR Engine [0..1] Configuration (XExCR) Register (n=0–1) ([Table 588 p. 664](#)).
3. Prepare the descriptor (or chain of descriptors) in memory.
4. Update the relevant XOR Engine [0..1] Next Descriptor Pointer (XExNDPR) Register (n=0–1) ([Table 590 p. 666](#)) register.
5. Set the `<XEStart>` field in the XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) ([Table 589 p. 665](#)).
6. When the XOR engine activates the relevant channel (`<XEstatus>` field in XE0ACTR or XE1ACTR is set).

When activated (`<XEstatus>` is set), the XOR engine fetches the descriptor pointed by the XOR Engine [0..1] Next Descriptor Pointer (XExNDPR) Register (n=0–1) ([Table 590 p. 666](#)), and starts performing the operation on it. Upon completion of the operation it progresses to the next descriptor. It continues this operation until it reaches the end of the descriptor chain (Next descriptor Address field of current descriptor = NULL). When it reaches the end of the chain, the XOR engine asserts an interrupt (EOC - End Of Chain interrupt), clears the `<XEstatus>` bit and enters an inactive state. This inactive state is equal to the initial state of XOR engine upon startup.

#### 11.5.1.2 Update Descriptor Chain

A new descriptor can be added to the chain even when the XOR engine is active (`<XEstatus>=1`).

The software adds new descriptors to the descriptor chain by performing the following:

1. Prepares new descriptors (or chain of descriptors) in memory.
2. Updates the next descriptor address field in the former last descriptor.



If the ownership mechanism is violated, the CPU will write to an XOR engine owned descriptor (the former last one). This does not affect the XOR engine operation.

#### 11.5.1.3 Pause Operation

The pause operation enables a temporary halt of the current descriptor chain processing and then a continuation of it without any impact on the execution, except the delay caused by the pause period.

When paused the XOR engine channel does not initiate any requests to the Mbus, releasing its resources to other units.

The pause operation can be used for boosting performance of a mission critical process for a specific time period. After the critical time period is over, the software signals the XOR engine channel to continue processing the current descriptor chain from the point at which it was paused. The software can pause the XOR engine channel operation during an active phase by performing the following:

1. Confirm that the relevant XOR engine channel is active ([\*\*<XEpause>\*\*](#) field in the XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) ([Table 589 p. 665](#)) is set). If it is not active, the pause operation is not necessary.
2. Set the relevant [\*\*<XEpause>\*\*](#).
3. Check the relevant [\*\*<XEstatus>\*\*](#) field. When it is cleared, the pause operation completed.

When paused ([\*\*<XEpause>\*\*](#) is set), the XOR engine channel suspends the current operation at the earliest opportunity, and enters a pause state. Upon entering a pause state, the XOR engine channel signals the software by clearing the [\*\*<XEstatus>\*\*](#) bit in the activation register and asserting the paused interrupt.

Receipt of an EOC interrupt before a paused interrupt, after initiation of a pause operation, implies that:

- The channel completed the current descriptor chain before the pause operation.
- The channel is in stop mode and not in paused mode.

The software must act accordingly and reactivate the channel according to the [Re-Activation After Stop](#) information.

### Re-Activation After Pause

To re-activate the channel, the software must set the [\*\*<XErestart>\*\*](#) field in the relevant activation register (XE0ACTR or XE1ACTR). When [\*\*<XEstatus>\*\*](#) field is set, the XOR engine has resumed operation.

After pausing a channel, it is not allowed to stop it. To stop the channel, the software must first perform a re-activation after pause operation. Only after the channel becomes active can it be stopped.

#### 11.5.1.4 Stop Operation

The stop operation terminates processing of an XOR engine channel's current operation. After stop, the current operation cannot be resumed and a new operation must be loaded to the XOR engine channel. The software can stop the XOR engine channel operation while active, by performing the following:

1. Check that the relevant XOR engine channel is active ([\*\*<XEstatus>\*\*](#) field in the XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) ([Table 589 p. 666](#)) is set). If it is not active, the stop operation is not necessary.
2. Set the relevant [\*\*<XEstop>\*\*](#) field.
3. Check the relevant [\*\*<XEstatus>\*\*](#) field. When it is cleared, the stop operation is completed.

When stopped ([\*\*<XEstop>\*\*](#) is set), the XOR engine stops performing the operation at the earliest opportunity and enters an inactive state. Upon entering an inactive state, the XOR engine closes the current descriptor and signals the software by clearing the [\*\*<XEstatus>\*\*](#) field in the activation register and asserting the stopped interrupt. Inactive state is similar to Initial state of XOR engine on startup.

### Re-Activation After Stop

Re-activation is similar to activation on startup (see [Section 11.5.1.1, Activation on Startup, on page 215](#)).

The software must perform the following steps:

1. Confirm that the relevant XOR engine channel is inactive. The [\*\*<XEstatus>\*\*](#) field in the XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) ([Table 589 p. 666](#)) is set to 0.
2. Initialize the relevant XOR engine channel through the XOR Engine [0..1] Configuration (XExCR) Register (n=0–1) ([Table 588 p. 664](#)).
3. Prepare a descriptor (or chain of descriptors) in memory.
4. Update the XOR Engine [0..1] Next Descriptor Pointer (XExNDPR) Register (n=0–1) ([Table 590 p. 666](#)).
5. Set the [\*\*<XERestart>\*\*](#) field.
6. When the XOR engine activates the relevant channel (the [\*\*<XEstatus>\*\*](#) field in the XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) ([Table 589 p. 666](#)) is set), the activation sequence is complete.

When activated ([\*\*<XEstatus>\*\*](#) is set), the XOR engine fetches the descriptor pointed by the XOR Engine [0..1] Next Descriptor Pointer (XExNDPR) Register (n=0–1) ([Table 590 p. 666](#)), and starts performing the operation on it. Upon completion, it progresses to the next descriptor and continues until the end of the descriptor chain is reached, the EOC - Next descriptor Address field of the current descriptor equals NULL. Upon reaching the end of the chain, the XOR engine clears the [\*\*<XEstatus>\*\*](#) bit and enters inactive state. This state is equal to the initial state of XOR engine on startup.

#### 11.5.1.5 Reaching End of Descriptor Chain

Upon reaching the end of the descriptor chain, the XOR engine asserts an EOC (End Of Chain) interrupt and enters an inactive state. It waits to be re-activated by the software (setting [\*\*<XEStart>\*\*](#) field in the XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) ([Table 589 p. 665](#))).

Upon receiving an EOC interrupt, two options must be examined by the software:

- True EOC: The XOR engine reaches the end of a descriptor chain.
- False EOC: The chain was updated and the XOR engine is not in the current EOC. This can occur when software updates the descriptor chain while the XOR engine is processing the former last descriptor in the chain. In this case, the XOR Engine [0..1] Next Descriptor Pointer (XExNDPR) Register (n=0–1) ([Table 590 p. 666](#)) has a NULL value. Although it did not reach a true EOC, the XOR engine enters an inactive state.

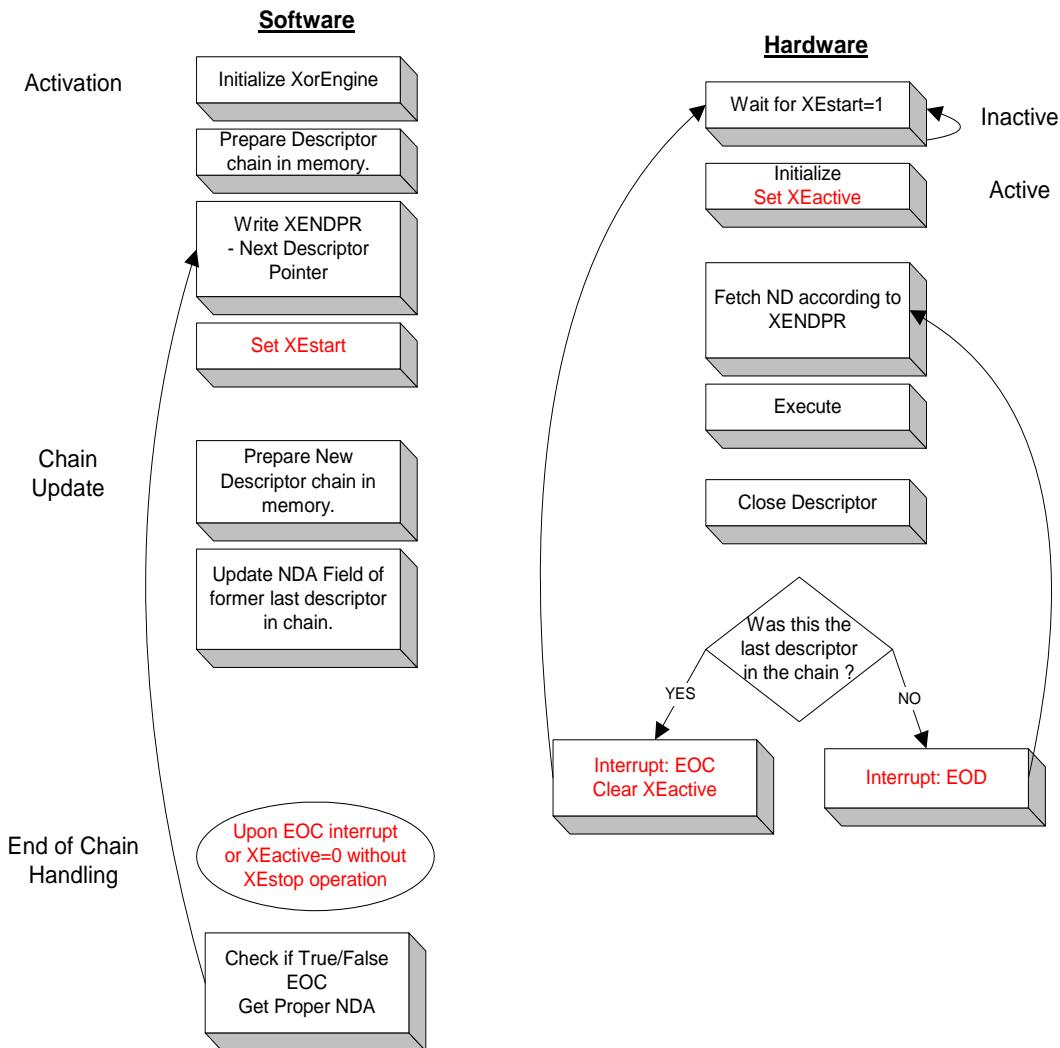
To determine which option is valid, and to act accordingly, the software must check if the XOR engine current descriptor is currently the last descriptor in the chain. For example, read the XOR Engine [0..1] Current Descriptor Pointer (XExCDPR) Register (n=0–1) ([Table 591 p. 666](#)) and match it with the software's current descriptor parameter.

If it is true, EOC acts according to activation after stop.

If it is false EOC forces the XOR engine to re-read the current descriptor. That is done by writing to the current descriptor pointer, that was read from XECDPR, to the Next descriptor Pointer Register (XENDPR), and performing an activation after stop, set the [\*\*<XEstart>\*\*](#) bit in the XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) ([Table 589 p. 665](#)).

### 11.5.1.6 Synchronizing Software and Hardware

**Figure 56: Software and Hardware Synchronization**



### 11.5.2 Programming in MemInit modes

MemInit mode is programmed and controlled directly through internal registers (without using descriptor chains).

### 11.5.2.1

#### Activation

To activate the XOR engine, the software must perform the following sequence:

1. Confirm that XOR engine relevant channel is inactive. The [\*\*<XEstatus>\*\*](#) field in the XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) ([Table 589 p. 666](#)) is set to 0.
2. Initialize the relevant XOR engine channel configuration through the XOR Engine [0..1] Configuration (XExCR) Register (n=0–1) ([Table 588 p. 664](#)).
3. Program the relevant internal registers (XOR engine MemInit Registers).
4. Set the [\*\*<XErestart>\*\*](#) field.

### 11.5.2.2

#### Stop Operation

The stop operation terminates a XOR engine channel's processing of the current operation. After stop, the current operation cannot be resumed. A new operation must be loaded to the XOR engine channel.

To stop the XOR engine channel operation while active, performing the following:

1. Check that the relevant XOR engine channel is active. The [\*\*<XEstatus>\*\*](#) field in the XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) ([Table 589 p. 666](#)) Register must be set. If it is not active, the stop operation is not necessary.
2. Set the [\*\*<XEstop>\*\*](#) field in the relevant activation register.
3. Check the relevant [\*\*<XEstatus>\*\*](#) field. When it is cleared, the stop operation is completed.

When stopped ([\*\*<XEstop>\*\*](#) is set), the XOR engine stops performing the operation at the earliest opportunity and enters an inactive state. Upon entering inactive state, the XOR engine signals the software by clearing the [\*\*<XEstatus>\*\*](#) field in the activation register and asserting the stopped interrupt. The inactive state is similar to initial state of the XOR engine on startup.

### 11.5.3

#### Internal Registers Write Access Protection

When an XOR engine channel is active, all the registers that are related to that channel, the shared address decoding registers, the shared channel arbitration registers and the shared memory initialization initial value registers, are write access protected. Every write request to those internal registers when the channel is active ([\*\*<XEstatus>\*\*](#) of the relevant channel is set) is silently disregarded. The only channel related registers that can be write accessed during the channel active period are the activation registers, the shared interrupt cause and mask registers, and the debug register.

This design prevents configuration changes during channel operation. Changes during a channel's operation can cause unpredictable results. The register access protection can be de-activated per channel through the relevant [\*\*<RegAccProtect>\*\*](#) field in the XOR Engine [0..1] Configuration (XExCR) Register (n=0–1) ([Table 588 p. 665](#)).

If at least one of the channels enables register write access protection, write accesses to internal registers shared by channels (for example, address decoding, channel arbitration, and memory initialization initial value registers) are disregarded.

Read requests for all internal registers are enabled at all times, regardless of the channel activation status (except for WO - Write Only registers).

## 11.6

### Burst Limit

The maximum burst sizes of different transaction types on the Mbus can be configured through the XOR Engine [0..1] Configuration (XExCR) Register (n=0–1) ([Table 588 p. 664](#)).

- Data read (reading a source buffer) maximum burst size: 32B / 64B / 128B.
- Data write (writing to destination buffer) maximum burst size: 32B / 64B / 128B.

A descriptor read, fetching a descriptor, is always a 32B burst size. In XOR mode, almost all the 64 bytes of the descriptor are relevant and two 32B read requests are required. In CRC mode, only the upper 32B of the descriptor are relevant and one 32B read request is sufficient.

Descriptor write, closing a descriptor, is always 8B burst size (Status and iSCSI CRC32C Result words).

## 11.7

## Errors and Interrupts

The XOR engine interrupts are registered in the XOR Engine Interrupt Cause (XEICR1) Register ([Table 593 p. 667](#)). Upon an interrupt event, the corresponding cause bit is set to 1. It is cleared upon a software write of 0.

The XOR Engine Interrupt Mask (XEIMR) Register ([Table 594 p. 668](#)) controls whether an interrupt event causes an interrupt assertion. The setting of the mask register only affects the interrupt assertion. This setting has no effect on the cause register bits setting.

The XOR engine interrupts can be divided to two groups:

- Error Interrupts: Descriptor ownership violation, address miss, multiple hit, window access violation, write protect violation, or parity error.
- Operation Completion Interrupts: EOD (End of Descriptor), EOC (End of Chain), pause, or stop by software.

[Table 66](#) summarizes the interpretation of EOD and EOC interrupts for each operation mode.

**Table 66: EOC/EOD interpretation**

Operation Mode	Operation Related Interrupt Description
XOR, CRC, DMA	<ul style="list-style-type: none"> <li>• The EOD interrupt is asserted upon closing each descriptor. If the &lt;EODIntEn&gt; bit of the descriptor is cleared, EOD interrupt is not asserted when it is closed.</li> <li>• The EOC interrupt is asserted upon reaching end of descriptor chain or upon end of chain processing due to error condition.</li> </ul>
MemInit	<ul style="list-style-type: none"> <li>• The EOC interrupt is asserted upon completing the MemInit operation.</li> </ul>

The following error interrupts are supported:

- Parity error: Internal data path parity error.
- Ownership error: Fetching descriptor that is owned by the CPU (software error).
- Address Miss Error: Accessing an address that is not in one of the address windows, or an address which matches more than one address window.
- Access Protect Error: Accessing an address which is access protected.
- Write Protect Error: Writing to a write protected address.

In all error conditions, the XOR engine halts, as if it is stopped by the software. Also, in all case of an error address, the address is latched in the XOR Engine Error Address (XEEAR) Register ([Table 596 p. 669](#)). Once an address is latched, no new addresses (due to additional errors) can be latched until the current address being read.

# 12

## Two-Wire Serial Interface (TWSI)

The device integrates a single, general-purpose TWSI port that can act as a TWSI master or as a slave.

If enabled via reset strap, the TWSI interface also supports initialization from external TWSI serial ROM.



**Note**

Refer to *AN-179 TWSI Software Guidelines for Discovery™, Horizon™, and Feroceon® Devices* for the specific sequences required for each CPU write or read access to the TWSI interface. If these are not implemented, the TWSI may not work well.

### 12.1

#### TWSI Bus Operation

The TWSI port consists of two open drain signals:

- SCL (Serial Clock)
- SDA (Serial Data/Address)

The TWSI master starts a transaction by driving a start condition followed by a 7- or 10-bit slave address and a read/write bit indication. The target TWSI slave responds with acknowledge.

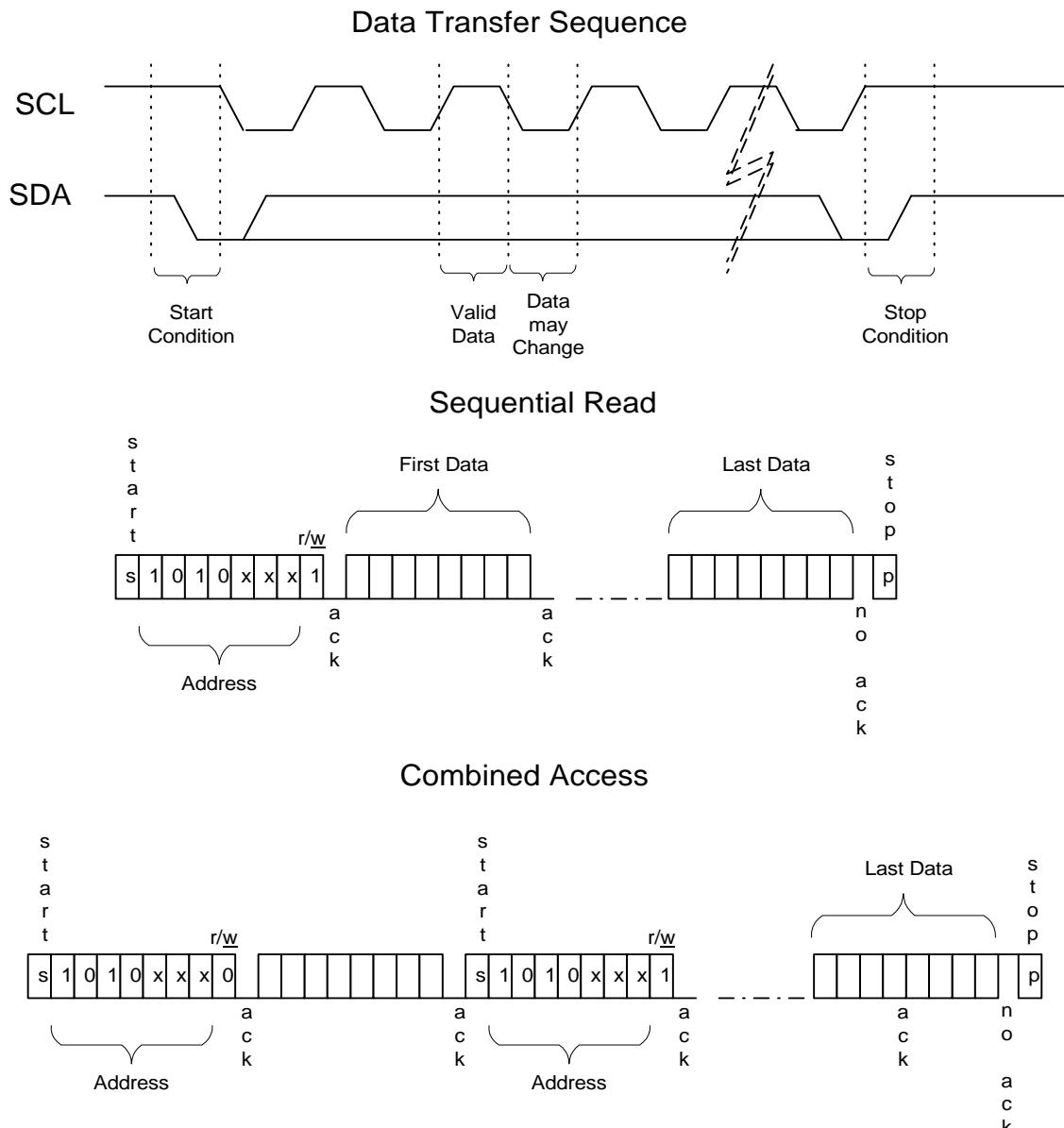
In case of a write access (R/W bit is 0 following the TWSI slave acknowledge), the master drives eight bits of data, and the slave responds with acknowledge. This write access (8-bit data followed by acknowledge) continues until the TWSI master ends the transaction with a stop condition.

In case of a read access following the TWSI slave address acknowledge, the TWSI slave drives eight bits of data and the master responds with acknowledge. This read access (8-bit data followed by acknowledge) continues until the TWSI master ends the transaction by responding with no acknowledge to the last 8-bit data, followed by a stop condition.

A target slave that cannot drive valid read data right after it received the address, can insert “wait states” by forcing SCL low until it has valid data to drive on the SDA line.

A master is allowed to combine two transactions. After the last data transfer, it can drive a new start condition followed by new slave address, rather than drive stop condition. Combining transactions guarantees that the master does not lose arbitration to some other TWSI master.

TWSI examples are shown in [Figure 57](#).

**Figure 57: TWSI Examples**


## 12.2

## TWSI Port Operation

The port can act as master, generating read/write requests, and as a slave, responding to read/write requests from an external master. It can be used for various applications, and can control other TWSI on-board devices such as temp sensors, to read DIMM SPD ROM. It is also used for serial ROM initialization (see the Serial ROM Initialization section in the device *Hardware Specifications*)

The TWSI interface master and slave activities are handled by a simple CPU access to internal registers, plus an interrupt interface. The protocol is byte oriented—the CPU is required to handle each transmitted/received byte. The following sections describe TWSI registers and receive/transmit operation.

## 12.2.1

### TWSI Slave Address Registers

The TWSI slave interface supports both 7-bit and 10-bit addressing. The slave address is programmed by the TWSI Slave Address Register ([Table 602 p. 671](#)) and TWSI Extended Slave Address Register ([Table 607 p. 674](#)).

When the TWSI receives a 7-bit address after a start condition, it compares that address against the value programmed in the Slave Address register. If the address matches, it responds with acknowledge.

If the received 7 address bits are ‘11110xx’, meaning that it is an 10-bit slave address, the TWSI compares the received 10-bit address with the 10-bit value programed in the Slave Address and Extended Slave Address registers. If the address matches, it responds with acknowledge.

The TWSI interface also supports slave response to general call transactions. If GCE bit in the Slave Address register is set to 1, the TWSI also responds to general call address (0x0).

## 12.2.2

### TWSI Data Register

The 8-bit data register is used both in master and slave modes.

In master mode, the CPU must place the slave address or write data to be transmitted. In case of read access, it contains received data (need to be read by CPU).

In slave mode, the Data register contains data received from master on write access, or data to be transmitted (written by CPU) on read access.



The data register Most Significant bit (MSb) contains the first bit to be transmitted or being received.

## Note

## 12.2.3

### TWSI Control Register

This 8-bit register contains the following bits:

**Table 67: TWSI Control Register Bits**

Bit	Function	Description
1:0	Reserved	Read only 0.
2	Acknowledge Bit	When set to 1, the TWSI drives an acknowledge bit on the bus in response to a received address (slave mode), or in response to a data received (read data in master mod, write data in slave mode). For a master to signal a TWSI target a read of last data, the CPU must clear this bit (generating no acknowledge bit on the bus). For the slave to respond, this bit must always be set back to 1.
3	Interrupt Flag	If any of the interrupt events occur, set to 1 by TWSI hardware If set to 1 and TWSI interrupts are enabled through bit[7], an interrupt is asserted.
4	Stop Bit	When set to 1, the TWSI master initiates a stop condition on the bus. The bit is set only. It is cleared by TWSI hardware after a stop condition is driven on the bus.

**Table 67: TWSI Control Register Bits (Continued)**

Bit	Function	Description
5	Start Bit	When set to 1, the TWSI master initiates a start condition on the bus, when the bus is free, or a repeated start condition, if the master already drives the bus. The bit is set only. It is cleared by TWSI hardware after a start condition is driven on the bus.
6	TWSI Enable	If set to 1, the TWSI slave responds to calls to its slave address, and to general calls if enabled. If set to 0, SDA and SCL inputs are ignored. The TWSI slave does not respond to any address on the bus.
7	Interrupt Enable	If set to 1, TWSI interrupts are enabled. Marvell recommends to use the TWSI interrupt to interface the TWSI module, rather than using the register polling method.
31:8	Reserved	Reserved

## 12.2.4 TWSI Status Register

This 8-bit register contains the current status of the TWSI interface. Bits[7:3] are the status code, bits[2:0] are Reserved (read only 0). [Table 68](#) summarizes all possible status codes.

**Table 68: TWSI Status Codes**

Code	Status
0x00	Bus error. <b>NOTE:</b> A bus error occurs if the TWSI slave interface drives the bus (Data or Clock) when it should not. To recover from this error, set the <a href="#"><b>&lt;Stop&gt;</b></a> field in the TWSI Control Register ( <a href="#">Table 604 p. 672</a> ) and clear the interrupt.
0x08	Start condition transmitted.
0x10	Repeated start condition transmitted.
0x18	Address + write bit transmitted, acknowledge received.
0x20	Address + write bit transmitted, acknowledge not received.
0x28	Master transmitted data byte, acknowledge received.
0x30	Master transmitted data byte, acknowledge not received.
0x38	Master lost arbitration during address or data transfer.
0x40	Address + read bit transmitted, acknowledge received.
0x48	Address + read bit transmitted, acknowledge not received.
0x50	Master received read data, acknowledge transmitted.
0x58	Master received read data, acknowledge not transmitted.
0x60	Slave received slave address, acknowledge transmitted.
0x68	Master lost arbitration during address transmit, address is targeted to the slave (write access), acknowledge transmitted.

**Table 68: TWSI Status Codes (Continued)**

Code	Status
0x70	General call received, acknowledge transmitted.
0x78	Master lost arbitration during address transmit, general call address received, acknowledge transmitted.
0x80	Slave received write data after receiving slave address, acknowledge transmitted.
0x88	Slave received write data after receiving slave address, acknowledge not transmitted.
0x90	Slave received write data after receiving general call, acknowledge transmitted.
0x98	Slave received write data after receiving general call, acknowledge not transmitted.
0xA0	Slave received stop or repeated start condition.
0xA8	Slave received address + read bit, acknowledge transmitted.
0xB0	Master lost arbitration during address transmit, address is targeted to the slave (read access), acknowledge transmitted.
0xB8	Slave transmitted read data, acknowledge received.
0xC0	Slave transmitted read data, acknowledge not received.
0xC8	Slave transmitted last read byte, acknowledge received.
0xD0	Second address + write bit transmitted, acknowledge received.
0xD8	Second address + write bit transmitted, acknowledge not received.
0xE0	Second address + read bit transmitted, acknowledge received.
0xE8	Second address + read bit transmitted, acknowledge not received.
0xF8	No relevant status. Interrupt flag is kept 0.

## 12.2.5 Baud Rate Register

The TWSI spec defines SCL frequency of 100 kHz (400 kHz in fast mode). The TWSI module contains a clock divider to generate the SCL clock. Setting bits[6:0] (fields **<M>** and **<N>**) of the TWSI Baud Rate Register ([Table 606 p. 674](#)) defines SCL frequency as follows:

$$F_{SCL} = \frac{F_{TClk}}{10 \blacktriangle (M+1) \blacktriangle 2^{(N+1)}}$$



**Note**

Where M is the value represented by bits[6:3] and N the value represented by bits[2:0]. If for example  $M=N=4$  (which are the default values), running  $TCLK$  at 200 MHz results in SCL frequency of 125 KHz.

As defined in the TWSI spec, the maximum supported SCL frequency is 100 kHz. Fast mode (where SCL frequency is 400 kHz) is not supported.

The Baud Rate register must be set properly, even when using the TWSI port as a slave only. It should be set such that  $F_{SCL}$  will be in the range of x1 to x2 of the TWSI bus frequency.

## 12.2.6

## TWSI Port Master Operation

A master write access consists of the following steps:

1. The CPU sets the [<Start>](#) field in the TWSI Control Register ([Table 604 p. 672](#)) to 1. The TWSI master then generates a start condition as soon as the bus is free, sets an Interrupt flag, and sets the Status register to 0x8.
2. The CPU writes a 7-bit address plus a write bit to the Data register and clears the Interrupt flag for the TWSI master interface to drive the slave address on the bus. The target slave responds with acknowledge. This causes an Interrupt flag to be set and a status code of 0x18 is registered in the Status register.

If the target TWSI device has an 10-bit address, the CPU needs to write the remainder 8-bit address bits to the Data register. The CPU then clears the Interrupt flag for the master to drive this address on the bus. The target device responds with acknowledge, causing an Interrupt flag to be set and status code of 0xD0 be registered in the Status register.

3. The CPU writes a data byte to the Data register, and then clears the Interrupt flag for the TWSI master interface to drive the data on the bus. The target slave responds with acknowledge, causing the Interrupt flag to be set, and status code of 0x28 be registered in the Status register. The CPU continues this loop of writing new data to the Data register and clearing the Interrupt flag as long as it needs to transmit write data to the target.
4. After the last data transmit, the CPU may terminate the transaction or restart a new transaction. To terminate the transaction, the CPU sets the Control Register [<Stop>](#) bit and then clears the Interrupt flag. This causes the TWSI master to generate a stop condition on the bus and to return to idle state. To restart a new transaction, the CPU sets the TWSI Control Register [<Start>](#) bit and clears the Interrupt flag, thus causing the TWSI master to generate a new start condition.

**Note**

The above sequence describes a normal operation. There are also abnormal cases, such as a slave not responding with acknowledge or arbitration loss. Each of these cases is reported in the Status register and needs to be processed by the CPU.

A master read access consists of the following steps:

1. Generating start condition, exactly the same as in the case of write access (see [Section 12.2.1, TWSI Slave Address Registers](#) ).
2. Drive 7- or 10-bit slave address, exactly the same as in the case of write access, with the exception that the status code after the first address byte transmit is 0x40, and after 2nd address byte transmit (in case of 10-bit address) is 0xE0.
3. Read data being received from the target device is placed in the data register and acknowledge is driven on the bus. Also interrupt flag is set, and status code of 0x50 is registered in the Status register. The CPU reads data from Data register and clears the Interrupt flag to continue receiving next read data byte. This loop is continued as long as the CPU wishes to read data from the target device.
4. To terminate, the read access needs to respond with no acknowledge to the last data. It then generates a stop condition or generates a new start condition to restart a new transaction. With last data, the CPU clears the TWSI Control Register [<Acknowledge>](#) bit (when clearing the Interrupt bit), causing the TWSI master interface to respond with no acknowledge to last received read data. In this case, the Interrupt flag is set with status code of 0x58. Now, the CPU can issue a stop condition or a new start condition.



**Note**

This sequence describes a normal operation. There are also abnormal cases, such as the slave not responding with acknowledge, or arbitration loss. Each of these cases is reported in the Status register and needs to be handled by CPU.

### 12.2.7 TWSI Port Slave Operation

The TWSI slave interface can respond to a read access, driving read data back to the master that initiated the transaction, or respond to write access, receiving write data from the master.

Upon detecting a new address driven on the bus with a read bit indication, the TWSI slave interface compares the address against the address programmed in the Slave Address register. If it matches, the slave responds with acknowledge. It also sets the Interrupt flag, and sets status code to 0xA8.



**Note**

If the TWSI slave address is 10-bit, the interrupt flag is set, and the status code changes only after receiving and identifying an address match on the second address byte.

The CPU now must write new read data to the Data register and clears the Interrupt flag, causing TWSI slave interface to drive the data on the bus. The master responds with acknowledge causing an Interrupt flag to be set, and status code of 0xB8 to be registered in the Status register.

If the master does not respond with acknowledge, the Interrupt flag is set, status code of 0xC0 is registered, and TWSI slave interface returns back to idle state.

If the master generates a stop condition after driving an acknowledge bit, the TWSI slave interface returns back to idle state.

Upon detecting a new address driven on the bus with write bit indication, the TWSI slave interface compares the address against the address programmed in the Slave Address register. If the address matches, it responds with acknowledge.matches, responds with acknowledge. It also sets an Interrupt flag, and sets the status code to 0x60 (0x70 in case of general call address, if general call is enabled).

Following each write byte received, the TWSI slave interface responds with acknowledge, sets an Interrupt flag, and sets status code to 0x80 (0x90 in case of general call access). The CPU then reads the received data from Data register and clears Interrupt flag to allow transfer to continue.

If a stop condition or a start condition of a new access is detected after driving the acknowledge bit, an Interrupt flag is set and a status code of 0xA0 is registered.

## 12.3 TWSI Serial ROM Initialization

The TWSI port can be set at reset to perform serial ROM initialization (to act as a TWSI master, reading data from an external TWSI ROM, and writing this data to the device registers). See the Reset Configuration section of the device *Hardware Specifications*.

# 13 UART Interface

The device supports a two-port Universal Asynchronous Receiver/Transmitter (UART) interface.

Each UART port performs the following data conversions:

- Serial-to-parallel conversion on data characters received from a peripheral device or a modem
- Parallel-to-serial conversion on data characters received from the device

The device can read the complete UART status for the Line Status (LSR) Register ([Table 624 p. 683](#)). Status information includes the type and condition of transfer operations and error conditions (parity, overrun, framing, or break interrupt) associated with the UART.

Each serial port operates in either FIFO or non-FIFO mode. In FIFO mode, a 16-byte transmit FIFO holds data from the device until it is transmitted on the serial link; a 16-byte receive FIFO buffers data from the serial link until it is read by the device.

For complete information regarding the UART, refer to the Synopsys DW\_16550 specification.

## 13.1 Features

The UART interface incorporates the following features:

- Ability to add or delete standard asynchronous communications bits (start, stop, and parity) in the serial data
- Independently controlled transmit, receive, line\_status, and data-set interrupts
- Programmable baud-rate generator (see [Section 13.4, Programmable Baud-Rate Generator](#))
- Modem control functions (nCTS and nRTS).
- Programmable serial interface:
  - 5-, 6-, 7- or 8-bit characters
  - Even, odd, or no parity detection
  - 1, or 2 stop-bit generation
- 16-byte transmit FIFO
- 16-byte receive FIFO
- Complete status-reporting capability
- Ability to generate and detect line breaks
- Internal diagnostic capabilities that include:
  - Loopback controls for communication link fault isolation
  - Break, parity, and framing-error simulation
- Fully prioritized interrupt system controls

## 13.2 UART Interface Pin Assignment

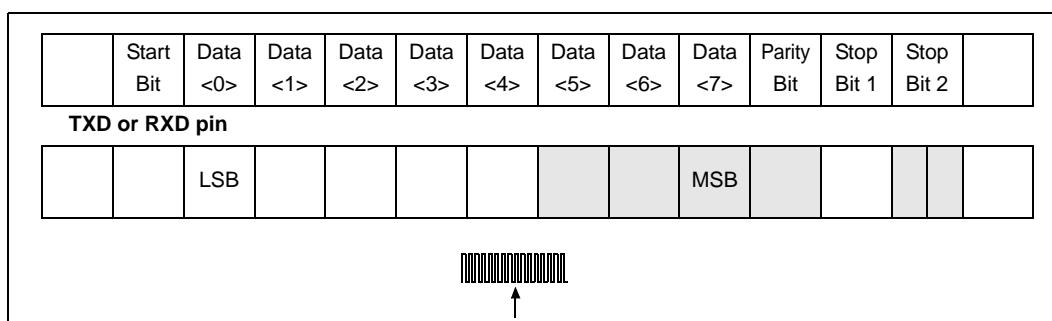
The device supports the UART interface through the UA0/1\_TXD and UA0/1\_RXD pins and provides modem control functions through the UA0/1\_CTSn and UA0/1\_RTSn pins, multiplexed on the MPP.

For a description of the UART signals, refer to the *Hardware Specifications* for the device.

## 13.3 Operation

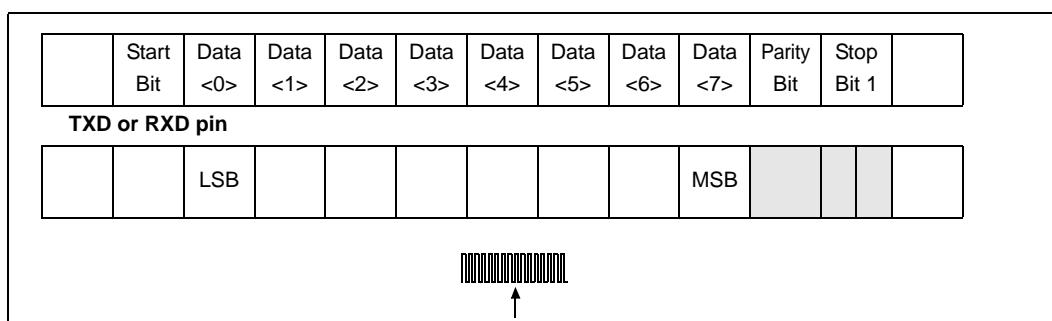
[Figure 58](#) and [Figure 59](#) show the format of a UART data frame, respectively, with two stop bits and with one stop bit.

**Figure 58: Example UART Data Frame (Two Stop Bits)**



To ensure bus stability, the receiver samples the serial input data at approximately the mid-point of the bit sequence, once the start bit has been detected.

**Figure 59: Example UART Data Frame (One Stop Bit)**



The receive-data sample-counter frequency is 16 times the value of the bit frequency. The 16x clock is created by the baud-rate generator. Each bit is sampled three times in the middle of the bit sequence. Shaded bits are optional and can be programmed by software.

The data frame is between 7 and 12 bits long, depending on the size of the data programmed, parity status (enabled/disabled), and the number of stop bits. A data frame begins by transmitting a start bit that is represented by a high to low transition. The start bit is followed by five to eight bits of data that begin with the least significant bit (LSB).

The data bits are followed by an optional parity bit. The parity bit is set if even parity is enabled and the data byte has an odd number of ones, or if odd parity is enabled and the data byte has an even number of ones. The data frame ends with one or two stop bits, as programmed by software. The stop bits are represented by one or two successive bit periods of logic one.

Each UART port has a transmit and a receive FIFO.

- The transmit FIFO is 16 bytes deep.
- The receive FIFO is 16 bytes deep.

## 13.4

## Programmable Baud-Rate Generator

Each UART port includes a programmable baud-rate generator that can take a fixed-input clock and divide it to generate the preferred baud rate. The baud rate is calculated by taking the TCLK frequency and dividing it by a value between 1 and  $(2^{16} - 1)$ , to produce a 16x clock that is used to drive the internal transmit and receive logic. Each UART operates in an environment that is either controlled by software and can be polled, or is interrupt driven.



The baud-rate generator output frequency is 16 times the baud rate. The [<DivLatchLow>](#) field in the Divisor Latch Low (DLL) Register ([Table 616 p. 679](#)) and the [<DivLatchHigh>](#) field in the Divisor Latch High (DLH) Register ([Table 619 p. 680](#)) make up the two 8-bit divisor latch fields that store the divisor in a 16-bit binary format. Load these divisor latches during initialization to ensure that the baud-rate generator operates properly. The 16x clock stops both fields are loaded with 0x0.

The baud rate of the data shifted into or out of a UART is given by the formula:

$$\text{BaudRate} = \frac{\text{TCLK freq}}{(16 \times \text{Divisor})}$$

**Table 69: Typical Baud Rates where TCLK = 166 MHz**

Required Baud	Divisor (decimal)	Actual Baud Rate	Error (%)
1200	8681	1200	0
2400	4340	2400	0
4800	2170	4800	0
9600	1085	9600	0
19,200	543	19,184	0.1
38,400	271	38,438	0.1
57,600	181	57,551	0.1
76,800	136	76,593	0.3
115,200	90	115,741	0.5
230,400	45	231,481	0.5
460,800	23	452,899	1.7
500,000	21	496,032	0.8
921,600	11	946,970	2.8

# 14 8-bit NAND Flash Interface

The device integrates a NAND Flash interface that incorporates the following features:

- Glueless interface to CE don't care NAND flash
- Glueless interface to CE care NAND flash
- Boot from NAND flash (see [Section 14.6, Boot from NAND Flash, on page 234](#))
- Read bursts of up to 32 bytes, splits the transaction to multiple 8 bytes bursts

## 14.1 NAND Flash Interface Pin Assignment

[Table 70](#) provides a list of the NAND Flash interface pins.

**Table 70: Device Controller Pin Assignments**

Pin Name	Type	Description
NF_CEn	O	NAND Flash Chip Enable (CE)
NF_REn	O	NAND Flash Read Enable
NF_WEn	O	NAND Flash Write Enable
NF_ALE	O	NAND Flash Address Latch Enable
NF_CLE	O	NAND Flash Command Latch Enable
NF_D[7:0]	t/s I/O	NAND Flash Data Bus <b>NOTE:</b> NF_D[7:0] do not have dedicated pins; these signals are multiplexed on the MPP interface.

## 14.2 NAND Flash Types

The device bus supports both chip enable (CE) care NAND flash and CE don't care NAND flash.

The difference between CE care NAND flash and CE don't care NAND flash is that for CE don't care NAND flash the chip enable does not need to be continuously asserted low during the read busy period.

## 14.3 Software Responsibilities

The software is responsible for the following:

- Enabling access to NAND flash, following the appropriate guidelines, as specified in [Section 14.3.1, Guidelines for Access to NAND Flash](#)
- Generating ECC
- Checking ECC
- Error recognition
- Error correction
- Following the timing constraints of the NAND flash device

### 14.3.1 Guidelines for Access to NAND Flash

Follow the guidelines listed below for access to NAND flash.



These guidelines apply to both CE care NAND flash and CE don't care NAND flash.

**Note**

**For a CE care NAND Flash device:** Make sure the [<NFActCEnBoot>](#) field in the NAND Flash Control Register ([Table 613 p. 676](#)) is set to 1. This bit may be cleared to 0 after the NAND flash transaction is completed.

**Address phase:** Write single-byte data to the NAND Flash with A[1:0] = 10 and with the data containing the required address. Repeat this phase as required by the NAND flash device.

**Command phase:** Write single-byte data to the NAND Flash with A[1:0] = 01 and with the data containing the required command.

**Read Data phase:** Single or burst read transactions are allowed. Perform as many read transactions as required with A[1:0] = any value. Up to 32 bytes per burst (see [Section 14.4, NAND Flash Interface Read Timing Parameters, on page 232](#)).

**Write Data phase:** Single-byte write transaction with A[1:0] = 00 and the data including the required data to be written to the NAND Flash. Repeat this stage as many times as required. Bursts are not allowed (see [Section 14.5, NAND Flash Interface Write Timing Parameters, on page 234](#)).

**Polling:** Poll (read) the NAND Flash device status via the Status Register in the NAND Flash device, which may be read to determine whether the program or erase operation is completed, and whether the operation has completed successfully. Alternatively, the ready busy (R/B) signal can be connected to the device MPP interface.



Only the CPU can access a NAND Flash device on the device bus interface.

**Note**

## 14.4 NAND Flash Interface Read Timing Parameters

To allow flexible interfacing to slow and fast devices, the interface read timing can be programmed with different timing parameters according to the NAND Read Parameters Register ([Table 611 p. 675](#)) and the NAND Flash Control Register ([Table 613 p. 676](#)). The following parameters affect the read timing:

**TurnOff:** Used to define the number of cycles in a read access between the negation of NF\_CEn to following assertion of NF\_CEn.  
Number of cycles = TurnOff + 4

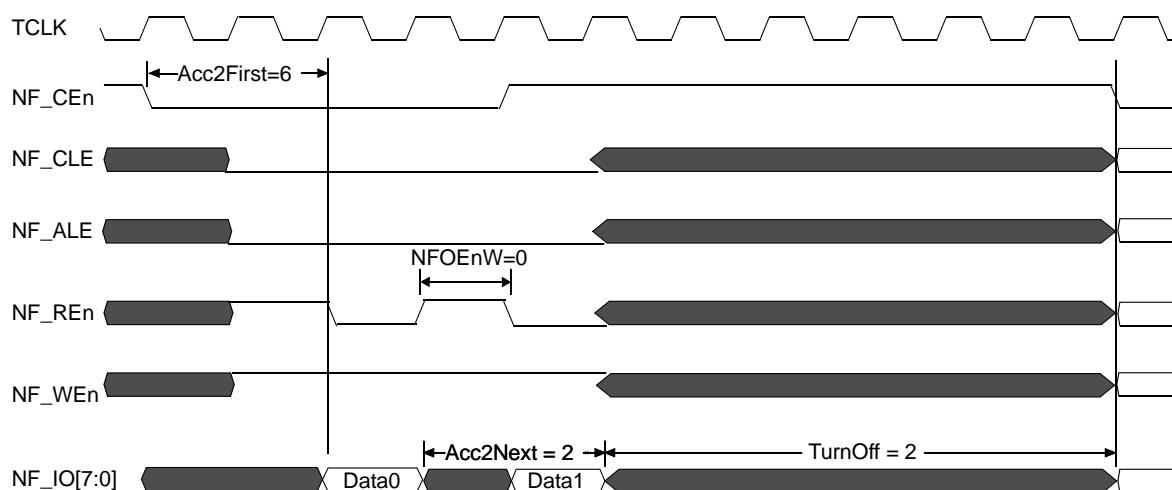
**Acc2First:** Used to define the number of TCLK cycles from the assertion of NF\_CEn to the cycle where the first read data is sampled by the device.  
Number of cycles = Acc2First - 4

- Acc2Next:** This parameter defines the number of TCLK cycles between the cycle that samples data N to the cycle that samples data N+1 (in burst accesses). The minimum setting of this parameter is 0x2.
- NFOEnW:** Used to define the number of TCLK cycles from the negation of NF\_REn to its next assertion during a burst read.  
Number of cycles = NFOEnW + 1

All output signals are driven with the rising edge of TCLK, and all inputs are sampled with the rising edge of TCLK.

Figure 60 provides a NAND Flash read timing parameters example.

**Figure 60: 8-bit NAND Flash Read Parameters Example**



#### 14.4.1 Read Burst Support

The NAND Flash controller supports a write of single byte in each transaction and a read of up to 32-byte burst. The NAND Flash controller incorporates a single 32-byte read buffer. Multiple 8-byte read bursts are performed towards the NAND Flash device until the data transfer is completed.

The NAND Flash controller needs to pack read data from the 8-bit wide NAND Flash to the device internal 64-bit data path. The NAND Flash controller drives the read data to the initiator only when the transaction on the NAND Flash interface completes and all data resides on its internal buffer.

## 14.5

## NAND Flash Interface Write Timing Parameters

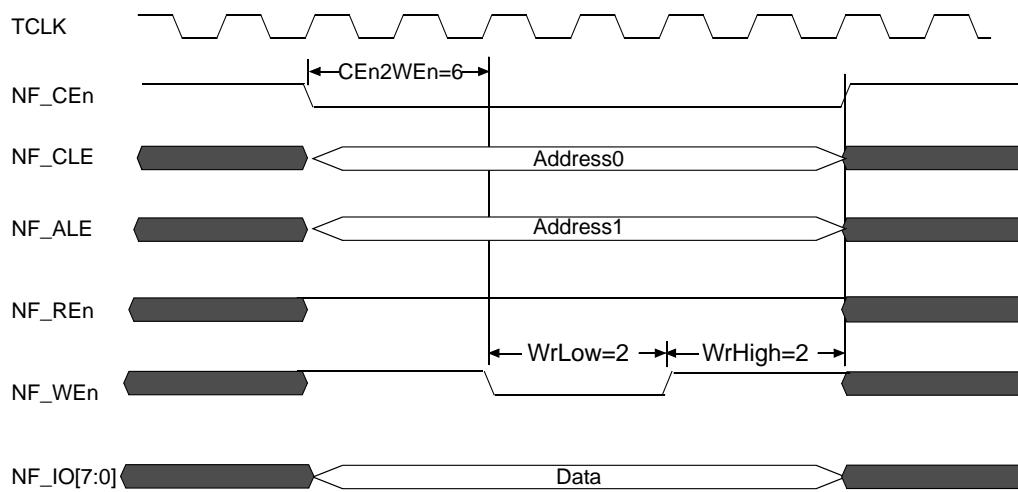
To allow flexible interfacing to slow and fast devices, the interface write timing can be programmed with different timing parameters according to NAND Write Parameters Register ([Table 612 p. 676](#)). The following parameters affect the write timing:

- CEn2WEn:** Used to define the number of TCLK cycles from NF\_CEn assertion to NF\_WEn assertion.  
 Number of cycles = CEn2WEn - 4
- WrLow:** This parameter defines the number of TCLK cycles that NF\_WEn is active (low), which is the setup time of data to NF\_WEn rise. The NF\_D bus is valid as long as NF\_WEn is active.
- WrHigh:** This parameter defines the number of cycles in a write access from NF\_WEn de-assertion to NF\_CEn de-assertion, which is the hold time of the NF\_D bus after NF\_WEn rise. Data is kept valid as long as NF\_CEn is active. The minimum setting of this parameter is 0x1.

All output signals are driven with the rising edge of TCLK, and all inputs are sampled with the rising edge of TCLK.

[Figure 61](#) provides a NAND Flash write timing parameters example.

**Figure 61: 8-bit NAND Flash Write Parameters Example**



## 14.6

## Boot from NAND Flash

The device supports booting directly from NAND Flash, when the first block—placed on 00h block address—is guaranteed to be a valid block with no errors.

### 14.6.1

### Boot Sequence

If the **<NFISD>** field in the NAND Flash Control Register ([Table 613 p. 677](#)) is cleared to 0, the following internal process is performed immediately after reset to issue the read command to the

NAND Flash device. This process allows the CPU to perform sequential reads to the NAND Flash immediately after reset, starting from address 0x0.

1. Set CPU internal reset.
2. Write 8 bits to the NAND Flash to Address = 0x1 with Data = 0x0. Set the NAND Flash command to Read.
3. Write 8 bits to the NAND Flash to Address = 0x2 with Data = 0x0. Set Address to 0x0.  
This step is repeated 5 times.
4. Write 8 bits to the NAND Flash to Address = 0x1 with Data = 0x30.
5. Wait until NAND Flash is ready with valid data, as defined by the [\*\*<NFTr>\*\*](#) field.
6. Clear CPU internal reset.

# 15

## Serial Peripheral Interface (SPI)

The device integrates a general-purpose Serial Peripheral Interface (SPI) port. It supports two modes:

**Indirect mode** Byte transmit and receive operations, using register access.

**Direct mode** This mode only supports read operations. It enables the CPU to directly access the memory space of the SPI. This mode support bursts of up to 128B.

The SPI is a synchronous serial data protocol used for transferring data simply and quickly from one device to another. With an SPI connection, there is always one Master device that controls the peripheral devices (Slaves).

The main features of the SPI are:

- Synchronous protocol:
  - The data is clocked along with a clock signal.
  - The clock signal controls when data is changed and when it should be read.
  - The clock rate can vary.
- Master-Slave protocol:
  - The Master device controls the clock.
  - No data is transferred unless a clock signal is present.
  - The slaves are controlled by the master clock, and may not manipulate the clock.
- Data Exchange protocol:
  - As data is being clocked out, new data is clocked in.
  - The Master controls the exchange by manipulating the clock line.

### 15.1

### SPI Interface Signals

Table 71 provides a list of the SPI interface signals.

**Table 71: SPI Interface Signals**

Signal Name	Type	Description
SPI_MOSI	O	SPI data output. Data is output from the master and input to the slave.
SPI_MISO	I	SPI data input. Data is input to the master and output from the slave.
SPI_SCK	O	SPI clock
SPI_CSn	O	SPI chip select

SPI signals are multiplex on the Multi-Purpose Pins (MPPs). For a description of the SPI signals, refer to the *Hardware Specifications* for the device.

## 15.2 Indirect Mode

### 15.2.1 SPI Input/Output

The serial data clock is generated out of the device core clock (TCLK) with a programmable baud generator that has a prescaler values of 4, 6, 8,...30. At power up, the prescaler value is 18.

The SPI Chip-Select signal (SPI\_CS<sub>n</sub>) is asserted whenever the [\*\*<CSnAct>\*\*](#) field in the Serial Memory Interface Control Register ([Table 628 p. 685](#)) is set. The Serial data clock output SPI\_SCK is driven high and low only during data I/O.

### 15.2.2 Output One Byte to SPI

To output one byte, the CPU writes to the Serial Memory Data Out Register ([Table 630 p. 687](#)).

This register's bits [7:0] are shifted out on the SPI\_MOSI pin and on the falling edge of the SPI\_SCK. There will be eight clock pulses output for eight data bits. Bit [7] is output first. After all eight bits are shifted out, the [\*\*<SMemRdy>\*\*](#) field in the Serial Memory Interface Control Register ([Table 628 p. 685](#)) is set. When the CPU again writes to the [Serial Memory Data Out Register](#), the Serial-Memory-Data-Ready status ([\*\*<SMemRdy>\*\*](#) field) is cleared.

### 15.2.3 Input One Byte from SPI

While SPI\_SCK is toggling, the SPI\_MISO pin is clocked into the Serial Memory Data In Register ([Table 631 p. 687](#)). The data is clocked into bit 0 as the register is shifted LEFT.

Therefore, to shift in one byte from the external SPI device, the CPU can write a dummy data into the [Serial Memory Data Out Register](#). As dummy data is shifted out on SPI\_MOSI, data is shifted in from the SPI\_MISO pin. When the [\*\*<SMemRdy>\*\*](#) field is set, the CPU can read the [Serial Memory Data In Register](#) to retrieve the input data byte.

### 15.2.4 Output or Input Two Bytes

To simplify firmware and improve the read/write throughput, the SPI interface logic can shift two bytes in and out for each access. When the [\*\*<BYTE\\_LEN>\*\*](#) field in the Serial Memory Interface Configuration Register ([Table 629 p. 686](#)) is set to 0x1, two byte I/O mode is enable. When the CPU writes to the [Serial Memory Data Out Register](#), bits [7:0] are shifted out first, then bits {15:8} are shifted out.

For data in, the SPI\_MISO pin is shifted first into the [Serial Memory Data In Register](#) bits [7:0], and then bits [15:8].

When both bytes are shifted out (and in), the [\*\*<SMemRdy>\*\*](#) field is set.

## 15.3 Direct Mode

The device SPI controller also supports direct read and writes from/to external SPI devices, without the software overhead of reading and writing from/to the Serial Memory Interface Control Register ([Table 628 p. 685](#)), Serial Memory Data Out Register ([Table 630 p. 687](#)), and Serial Memory Data In Register ([Table 631 p. 687](#)).

### 15.3.1

### Direct Read from SPI

Two read modes are supported, as defined by the [\*\*<DirectRdCommand>\*\*](#) field in the Serial Memory Interface Configuration Register ([Table 629 p. 686](#)).

- **Read** A read command is used (0x03), and there is no dummy (Read Memory at up to TCLK/8): writer after the address phase.
- **Fast\_Read** A higher-speed read command is used (0x0B), and a (also known as high-speed read) one-byte dummy write is added after the address phase. (Read Memory at up to TCLK/4):

The length of the address can be set to be between 1 and 4 bytes according to the [\*\*<DirectAddrLen>\*\*](#) field of that register.

If using SPI flash, set the [\*\*<Attr>\*\*](#) field in the CPU address decoding windows to match the SPI interface (see [Section 2, Address Map, on page 34](#) for more details). Any CPU read to this address space is converted by the SPI controller to a SPI flash read transaction, composed of an address phase, followed by a data phase.

The actual sequence that is driven on the SPI interface is:

1. Assert SPI\_CS<sub>n</sub>. Write command to the SPI. The command is either Read or Fast\_Read, based on the configuration of the [\*\*<DirectRdCommand>\*\*](#) field.
2. Write the address. The Address is driven in 1 to 4 phases based on the configuration of the [\*\*<DirectAddrLen>\*\*](#) field. The order of the bytes is MSB to LSB.
3. In Fast\_Read mode, add a one-byte dummy write.
4. Read the data, according to the length given by the request.
5. De-assert the SPI\_CS<sub>n</sub> signal.



The maximum supported burst read is 32B.

**Note**

### 15.3.2

### Boot from SPI Flash

Direct Read from SPI can be used to boot from SPI.

If configured to boot from SPI at reset, the [\*\*<Attr>\*\*](#) field in the Window7 Control Register ([Table 112 p. 365](#)) is set to SPI interface, which results in redirecting the CPU boot reads to the SPI interface.

This causes the first instruction fetches to be directed to the SPI with an attribute that causes the transaction to be an SPI Direct mode transaction (see [Section 15.3](#)).

For boot from SPI flash using the BootROM code, see [Section 24.2, BootROM Firmware, on page 290](#).

### 15.3.3

### Direct Write to SPI

The device SPI Controller supports direct writes that will be transmitted on the SPI. The action and data to be transmitted consists of the following:

1. Assert CS.  
This stage can be omitted by clearing the [\*\*<Direct Wr Deassert Cs>\*\*](#) field in the Serial Memory Direct Write Configuration Register ([Table 634 p. 688](#)).
2. CS hold Gap.  
All signals are steady. The length of the gap will be at least [\*\*<Direct Wr Cs Hold>\*\*](#) core clock cycles. This gap will not exist when [\*\*<Direct Wr Deassert Cs>\*\*](#) is cleared.

3. Constant Header:  
This is a 1-4 byte field that is taken from the <SpiDirectHdr> field in the Serial Memory Direct Write Header Register ([Table 635 p. 688](#)).  
The size of the header is configured on <Direct Wr Hdr Size> field in the Serial Memory Direct Write Configuration Register ([Table 634 p. 688](#)) and can be omitted by clearing <Direct Write Hdr Enable> of that same register. A possible usage of this header is setting a Write command, that will be served as a prefix to any command.
4. Address Phase.  
This is a 1-4 byte field that is taken from the address of the request.  
The size of the address is configured via the <DirectAddrLen> field in the Serial Memory Interface Configuration Register ([Table 629 p. 686](#)).  
The entire Address phase is omitted when <Direct Wr Addr Enable> field in the Serial Memory Direct Write Configuration Register ([Table 634 p. 688](#)) is cleared.
5. Data Phase.  
A 1-32 byte transition of the data provided by the write request.
6. CS Preservation.  
The CS will be return to the state that it was prior to the Write command. If it was asserted before the action took place, then this stage is meaningless. If it was not asserted, then at this stage the CS will be de-asserted.  
This stage will be omitted if <Direct Wr Deassert Cs> is cleared.



The maximum supported burst read is 32B.

**Note**

#### 15.3.4 DMA Based SPI

Direct Write to SPI can be accomplished by the DMA function in the XOR engine, to transmit large buffers of data without the software overhead.

This is accomplished by completing the following steps:

1. Configure DMA registers so that the burst length is at most 32B.
2. Set the <Target> field in the XOR Engine Base Address (XEBA Rx) Register (n=0-7) ([Table 583 p. 660](#)) to 0x1.
3. Set in the <Attr> field of that register to 0x1E (Boot from SPI).



Direct Write to SPI cannot be used for flash devices.

**Note**

# 16 Audio ( $\text{I}^2\text{S}$ / S/PDIF) Interface (88F6180, 88F6192, and 88F6281 Only)

The 88F6180, 88F6192, and 88F6281 integrate an Audio Unit (ADU) that supports recording and playback for the following audio protocols:

- $\text{I}^2\text{S}$  with three different protocols:
  - Plain  $\text{I}^2\text{S}$  is the original Philips protocol
  - Right justified
  - Left justified
- Sony/Philips Digital Interconnect Format (S/PDIF) Consumer Mode IEC 60958-3
- IEC 61937 Non-PCM protocols above the S/PDIF layer

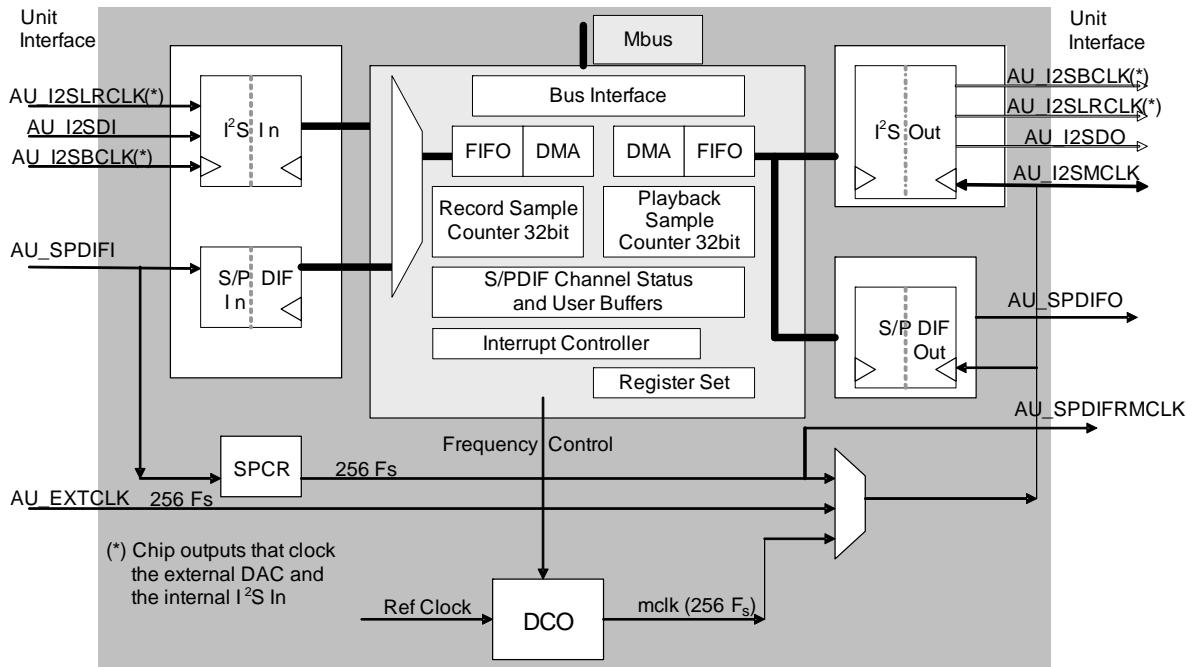
**Record:** The ADU can record audio using either the  $\text{I}^2\text{S}$  or S/PDIF input protocols. They cannot be used simultaneously.

**Play:** It can play audio using either the  $\text{I}^2\text{S}$  or S/PDIF output protocols, or using both protocols simultaneously.

By using two separate DMA engines, the ADU supports concurrent recording and playback.

Figure 62 provides a block diagram of the ADU.

**Figure 62: Audio Unit Block Diagram**



### General Features

The general features include:

- If Pulse Code Modulation (PCM) data is being played, the  $I^2S$  and S/PDIF outputs can be active simultaneously. In this scenario, the same audio data is passed to the  $I^2S$  and S/PDIF outputs. If non-PCM data is being played over the S/PDIF output, the  $I^2S$  output must be muted or disabled.
- Only one of the  $I^2S$  or S/PDIF inputs can be active at one time.
- The S/PDIF Clock Recovery (SPCR) is used to recover the S/PDIF clock (SPDIFRMCLK) and the clock frequency.
- The Digital Controlled Oscillator (DCO) is used for generating the  $I^2S$  playback and recording clock. It can also be used for the S/PDIF playback clock. The DCO takes its reference clock from the internal core clock. It can be configured to work at each of the supported frequencies with a configurable runtime offset of ppm<sup>1</sup> resolution adjustments.
- $I^2S$  or S/PDIF outputs can work with the SPCR, an external clock, or the programmable DCO. It is possible to move between the SPCR clock and the DCO clock in run time. Only switch to the AU\_EXTCLK or from the AU\_EXTCLK to the other clock sources when playback is inactive.
- The ADU supports stereo and mono playback and recording. During mono recording, only one of the channels is recorded. During mono playback, both channels may be used to concurrently play the same data, or the either of the channels can be muted resulting in the data playing in the single un-muted channel.
- Recording and playback may also be muted. Playback mute is possible on each of the interfaces separately.
- The ADU utilizes two DMA engines:
  - One recording DMA that is used to transfer data from the  $I^2S$  or S/PDIF interface to memory.
  - One playback DMA that is used to transfer data from memory to either the  $I^2S$  interface, the S/PDIF interface, or both interfaces.
- The DMA engines operate at the core clock speed, and can be configured to have a burst transfer of 32 bytes or 128 bytes during memory access.
- When recording or playback is activated, the DMA engines work independently using the configured start address and the size of a cyclic buffer in memory.
- The DMA operation continues without the need for software intervention until playback or recording is stopped or paused. However, software intervention is required for moving data from the recording cyclic buffer to another memory location during recording, and to write new data in the playback cyclic buffer during playback.

### $I^2S$ Supported Features

The  $I^2S$  protocol supports:

- An audio sample rate ( $F_s$ ) of 44.1/48/96 kHz.
- The  $I^2S$  input and  $I^2S$  output can only operate at the same sample rate.
- The available sample sizes are 16-bit, 20-bit, 24-bit, and 32-bit.
- Regardless of the sample size, 32-bits are always used per channel.
- Two channels are supported.
- Sample sizes do not have to be the same for input and output. Unused data bits are padded with zeros. For example, 8-bit padding is used for 24-bit sample sizes and 16-bit padding for 16-bit sample sizes.
- For the receiver and the transmitter side, the ADU outputs AU\_I2SMCLK (master clock = 256xF<sub>s</sub>) and functions as a master over ADU\_I2SLRCLK (1xF<sub>s</sub>) and over AU\_I2SBCLK (64xF<sub>s</sub>).
- Works in plain  $I^2S$ , right justified, and left justified formats.

---

1. ppm = Parts Per Million

### S/PDIF Supported Features

The S/PDIF protocol supports:

- The IEC 60958-1, IEC 60958-3, and IEC 61937 specifications.
- An audio sample rate ( $F_s$ ) of 44.1/48/96 kHz.
- Sample sizes of 16-bit, 20-bit, and 24-bit
- Outputs S/PDIFRMCLK = Recovered S/PDIF master clock (master clock =  $256xF_s$ )
- Signals lock status change to the CPU (loss of lock/return to lock). In the event of loss of S/PDIF input lock, the data stream is automatically paused. New samples are not written to the recording FIFO.
- Double buffering for status and user bits at both recording and playback.
  - For recording, an interrupt can be generated at the end of the recorded block if there was a change of any of the recorded channel status/user bits. An interrupt can also be generated at the end of each recorded block. This allows the user approximately a one block recording period to retrieve the new status or user bits.
  - For playback, an interrupt can be generated when a new block playback begins. This allows the user approximately a one block playback period to configure the new channel status and user bits of the next block before these bits are sent.
  - All interrupts can be masked. S/PDIF control bit related interrupts can be masked per channel.

## 16.1 Recording Data Flow

Follow these steps to record data:

1. Configure the <Recording Buffer Start Address> field in the Recording Start Address Register (Table 666 p. 708) and the <Recording Buffer Size> field in the Recording Buffer Size Register (Table 667 p. 708).
2. Configure the:
  - <Recording Base> field in the Recording Window Base Address Register (Table 644 p. 694)
  - Recording Window Control Register (Table 645 p. 694)
  - <Recording DMA Burst Size> field in the Recording Control Register (Table 665 p. 707) and the required audio configuration modes using:
    - the Recording Control Register (Table 665 p. 706)  
and
    - the I2S Recording Control Register (Table 670 p. 709)  
or
    - the SPDIF Recording General Register (Table 671 p. 710).
3. Configure the I<sup>2</sup>S clock source by selecting one of the following clock sources through the <MCLK source> field in the Clocks Control Register (Table 642 p. 693):
  - SPCR: The spcr\_lock should read high before playback is enabled.
  - Configurable DCO clock: The dco\_lock should be read high before playback is enabled.
  - External clock



**Note**

- Each of these clocks has a frequency of  $256xF_s$ .
- When I<sup>2</sup>S data is recorded, the ADU generates the I<sup>2</sup>S clocks from the configured clock source.

4. Decide which buffer management method to use. Enable the relevant mask bits and optionally set a proper value in the Recorded Byte Counter for Interrupt Register ([Table 669 p. 709](#)). The recording buffer management methods are described in section [Section 16.1.1, Recorded Data Buffer Management Methods, on page 243](#).
5. Enable recording by asserting the <I<sup>2</sup>S Recording Enable> or the <SPDIF Recording Enable> field in the Recording Control Register ([Table 665 p. 708](#)).

Based on the protocol being used, the DMA transfers data to the cyclic memory buffer when:

- The I<sup>2</sup>S interface is enabled.
- The S/PDIF interface is enabled and synchronized to the incoming signal.

The size of each DMA write burst access is determined by the recording DMA burst size configuration.

### 16.1.1

### Recorded Data Buffer Management Methods

There are two methods used by the software to move recorded data from the cyclic buffer to other memory locations before the recording DMA overwrites the buffer.

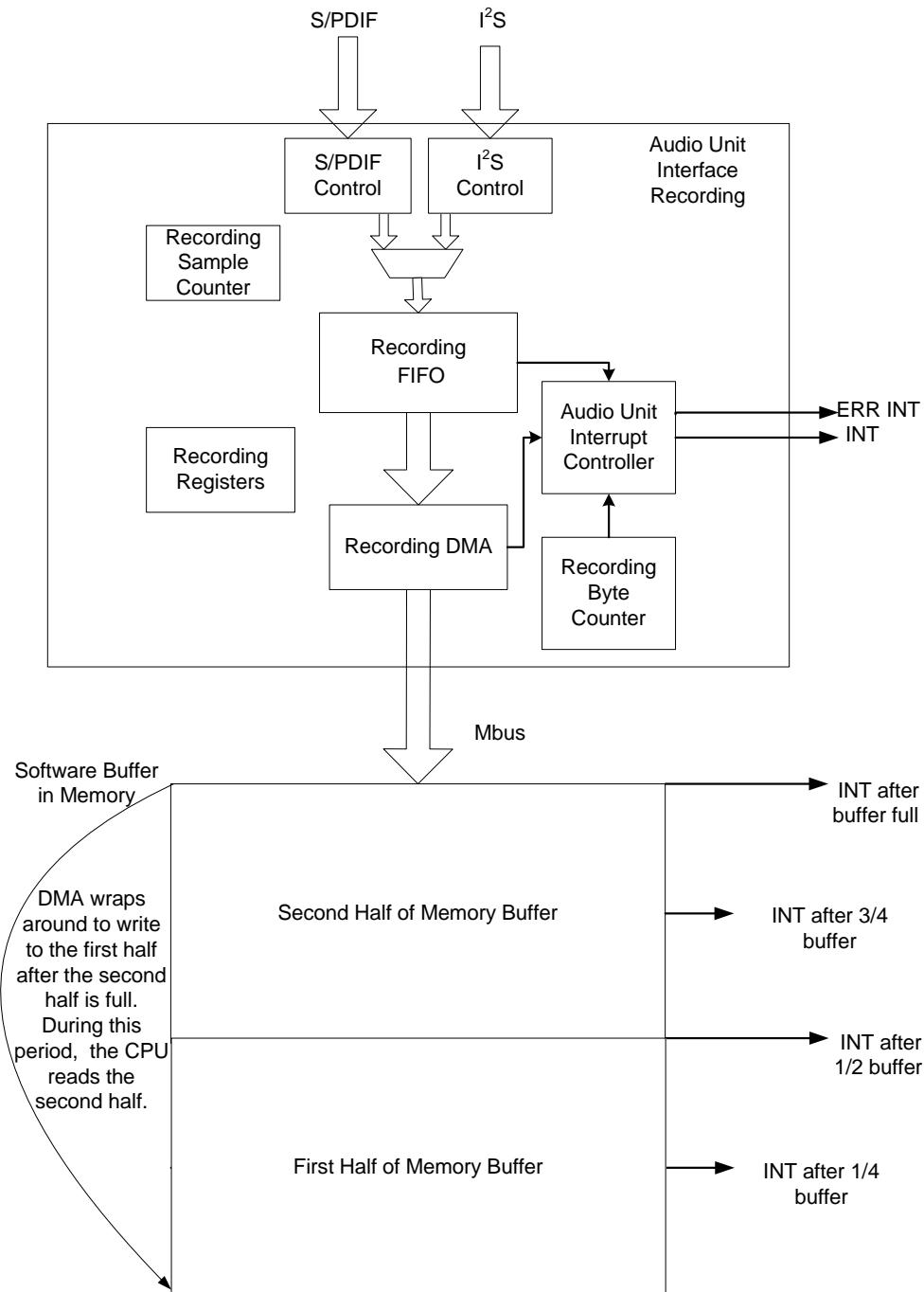
**Divided Buffer Method** As shown in [Figure 63](#), the buffer is divided into quarters or halves. After each quarter or half of the buffer is written by the recording DMA, the software is interrupted. An interrupt handler can then move the respective buffer segment to another memory location before it is overwritten, when the DMA address counter wraps around.

In this method, if an interrupt handler operation is delayed for a period of less than a full buffer recording time, the cause register still holds information of which quarters of the buffer were recorded. This provides the software an opportunity to recover from an unexpected interrupt handler delay and prevents data being dropped.

<b>Configured Recorded Byte Counter Method</b>	<p>It is possible to configure the recorded byte counter for an interrupt.</p> <p>Each time the dedicated counter reaches a pre-configured value in the Recorded Byte Counter for Interrupt Register (<a href="#">Table 669 p. 709</a>), an interrupt is asserted in the Audio Interrupt Cause Register (<a href="#">Table 650 p. 696</a>), and the counter is reset. This allows for user-defined interrupt resolutions.</p>
--	---

The interrupt cause bits can be read from the Audio Interrupt Cause Register ([Table 650 p. 696](#)). The respective mask bits can be found at the Audio Interrupt Mask Register ([Table 651 p. 698](#)).

**Figure 63: Recording Flow**



### 16.1.2

## Recovering Residual Data

After recording data is disabled, regardless of which buffer management method is used, residual recorded data remains in the memory buffer. This residual data enters the memory in the period between the assertion of a recording data interrupt and the disabling of the interface.

With the divided buffer method, it is possible to determine how many recorded bytes remain by reading the [<Recording Buf Byte Count>](#) field in the Recording Buffer Byte Counter Register ([Table 668 p. 709](#)). This field holds the number of bytes written by the recording DMA, since the last recording byte counter wrap around.

With the byte counter method, the [<Recorded Byte Count for Interrupt>](#) field in the Recorded Byte Count for Interrupt Register ([Table 652 p. 698](#)) can be used for the same purpose.

### 16.1.3

## I<sup>2</sup>S Recording

The ADU is always master over the I<sup>2</sup>S clocks. When I<sup>2</sup>S recording is enabled, the ADU activates the I<sup>2</sup>S clocks for both the external transmitter and the ADU recording sub-unit.

If a DCO is used to generate I<sup>2</sup>S clocks, check that the [<dco\\_lock>](#) field in the SPCR and DCO Status Register ([Table 638 p. 692](#)) is asserted before enabling I<sup>2</sup>S recording. This ensures that the I<sup>2</sup>S clocks operate as required.

### 16.1.4

## S/PDIF Specific Recording Flow

### Retrieving S/PDIF Status/User Bits from ADU Internal Buffers

When the recorded status/user bits are used from the internal ADU buffers, channel specific interrupts can be enabled through the SPDIF Recording Interrupt Cause and Mask Register ([Table 672 p. 711](#)). These interrupts are asserted after each recorded block that contains new channel status/user information is compared to the previous block. The comparison is performed at the end of the recorded block.

An interrupt can also be enabled after each recorded block, using the [<spdif\\_rc\\_block\\_end>](#) field in the Audio Interrupt Cause Register ([Table 650 p. 697](#)).

The status/user bits are double buffered. After the interrupt is asserted, software can read the status/user bits of the previous block while the recording of the current block is taking place. The interrupt handler has only one recording block time period to retrieve the user/status bits from the internal buffers before the bits may be changed at the beginning of the next recorded block.

### S/PDIF Recording Flow

Follow this procedure to perform S/PDIF recording:

1. Before enabling S/PDIF recording, mask all the S/PDIF interrupts cause bits in the SPDIF Recording Interrupt Cause and Mask Register ([Table 672 p. 711](#)).
2. After S/PDIF recording is enabled, wait for the ADU to lock on the incoming S/PDIF signal. A dedicated interrupt in the [<spdif\\_rc\\_locked>](#) field in the Audio Interrupt Cause Register ([Table 650 p. 698](#)) indicates that the ADU has locked. This register also reflects the lock status in the [<spdif\\_rc\\_lock\\_status>](#) field at the time the interrupt handler reads the cause register. After a lock is achieved, the SPCR registers the recovered S/PDIF frequency in the [<spcr\\_ctrlfs>](#) field in the SPCR and DCO Status Register ([Table 638 p. 691](#)). Then, the SPCR outputs the S/PDIF recovered clock.  
The data, status, and user bits only begin to be recorded after the lock is established.
3. Wait for the first interrupt assertion of the [<spdif\\_rc\\_block\\_end>](#) field in the Audio Interrupt Cause Register ([Table 650 p. 697](#)). Clear this interrupt and all additional interrupts in the [SPDIF Recording Interrupt Cause and Mask Register](#). Then, enable the required interrupts in the [SPDIF Recording Interrupt Cause and Mask Register](#).

Explanation: Recording may start after the beginning of the current S/PDIF block. This means that the status and user bits of the first recorded block stored in the ADU internal buffers cannot be used.

The reason is that it is impossible to know the location of the first recorded status and user bits inside the block at the time that the first block is being recorded.

4. The second time that any of the interrupts in the [SPDIF Recording Interrupt Cause and Mask Register](#) is asserted, the software reads the status information stored in the SPDIF Recording Channel Status Left n Register (n=0–5) ([Table 673 p. 712](#)) and SPDIF Recording Channel Status Right n Register (n=0–5) ([Table 674 p. 713](#)). The software then checks if PCM or non-PCM data is recorded and determines the sample size of the recorded S/PDIF.
5. Write the sample size that was retrieved from the status bits to the [<Recording Sample Size>](#) field in the Recording Control Register ([Table 665 p. 706](#)). After the sample size is written, future recorded data is organized to reflect the new sample size configuration (see [Section 16.4, Audio Unit Memory Structure, on page 252](#)).

**Note**

When following steps 1-5, up to three of the first S/PDIF blocks in memory can contain data that is not organized according to the [Audio Unit Memory Structure](#). This occurs when the recorded sample size that was retrieved from the status buffers is different than the initially configured sample size. Therefore, the user should mute or pause the recording from the time when recording is enabled until the actual sample size is configured.

## 16.1.5 Recording Mute

When recording mute is enabled by the [<Recording Mute>](#) field in the Recording Control Register ([Table 665 p. 707](#)), the I<sup>2</sup>S or S/PDIF recorded data value is replaced by zeros. The S/PDIF control bits remain untouched, except for the required S/PDIF parity update.

The recording DMA and the I<sup>2</sup>S or S/PDIF interfaces continue working normally.

## 16.1.6 Recording Pause

When recording pause is enabled in the [<Recording Pause>](#) field in the Recording Control Register ([Table 665 p. 707](#)), the I<sup>2</sup>S interface continues to work normally but the recording DMA halts. Unlike the I<sup>2</sup>S recording disable (STOP), the I<sup>2</sup>S clocks remains active as long as I<sup>2</sup>S recording is enabled. This allows the I<sup>2</sup>S transmitter to continue playing data even though the ADU discards it.

**Note**

If I<sup>2</sup>S recording is disabled by setting the [<I2S Recording Enable>](#) field in the Recording Control Register Recording Control Register ([Table 428 p. 386](#)) ([Table 428 p. 389](#)) to 0, the I<sup>2</sup>S clocks remains active if the I<sup>2</sup>S playback is enabled, because the I<sup>2</sup>S master mode is used.

## 16.2 Playback Flow

To playback data, use the following procedure:

1. Select one of the following clock sources through the [<MCLK source>](#) field in the Clocks Control Register ([Table 642 p. 693](#)). Each of these clocks has the frequency of 256xF<sub>S</sub>:
  - SPCR recovered S/PDIF recording clock: The spcr\_lock should be read asserted before playback is enabled.
  - Configurable DCO clock: The [<dco\\_lock>](#) field in the SPCR and DCO Status Register ([Table 638 p. 692](#)) should be read asserted before playback is enabled. The frequency of the DCO must be configured by writing to the [<dco\\_ctrlfs>](#) field in the DCO Control Register ([Table 637 p. 691](#)).
  - External clock

2. Configure the <Playback Buffer Start Address> field in the Playback Start Address Register ([Table 655 p. 702](#)) and the <Playback Buffer Size> field in the Playback Buffer Size Register ([Table 656 p. 702](#)).
3. Write the playback data to the entire cyclic buffer.
4. Configure the following:
  - <Playback Base> field in the Playback Window Base Address Register ([Table 646 p. 695](#))
  - Playback Window Control Register ([Table 647 p. 695](#))
  - <Playback DMA Burst Size> field in the Playback Control Register ([Table 654 p. 701](#))
  - Required audio configuration modes in the:
    - Playback Control Register ([Table 654 p. 699](#))
    - S/PDIF Playback Registers (see [Table 660](#) through [Table 664](#))
    - I2S Playback Control Register ([Table 659 p. 703](#))
5. Decide which buffer management method to use. Enable the relevant mask bits and optionally set a proper value in the Playback Byte Counter for Interrupt Register ([Table 658 p. 703](#)). The playback buffer management methods are described in section [Section 16.2.1, Playback Data Buffer Management Methods, on page 248](#).
6. Enable playback with either the <I2S Playback Enable> or the <SPDIF Playback Enable> field in the Playback Control Register ([Table 654 p. 700](#)).

It is possible to enable the I<sup>2</sup>S interface, the S/PDIF interface, or both interfaces simultaneously. If both interfaces are enabled, they should be enabled and disabled at the same time.

During simultaneous playback, the DMA sends the same data to both I<sup>2</sup>S and S/PDIF interfaces. S/PDIF control bits are sent only to the S/PDIF interface when a playback sample size of 24-bits or less is selected. Non-PCM data cannot be used during simultaneous playback.



When playing a 32-bit sample, disable or mute the S/PDIF output interface.

**Note**

---

As long as playback is activated, playback-DMA transfers data independently from the cyclic memory buffer to the ADU outputs. The size of each DMA read burst access is determined by the <Playback DMA Burst Size> field in the Playback Control Register ([Table 654 p. 701](#)).

### 16.2.1

## Playback Data Buffer Management Methods

The software must continuously fill the cyclic memory buffer using one of the following interrupt methods. Interrupt cause bits can be found at the Audio Interrupt Cause Register ([Table 650 p. 696](#)) and can be masked by using the Audio Interrupt Mask Register ([Table 651 p. 698](#)).

**Divided Buffer Method** As shown in [Figure 64](#), the first method divides the buffer into quarters or halves. After each quarter or half of the buffer is read by the playback DMA, the software is interrupted. An interrupt handler can overwrite the already played buffer segment with new playback data, while playback continues from the following buffer segments.

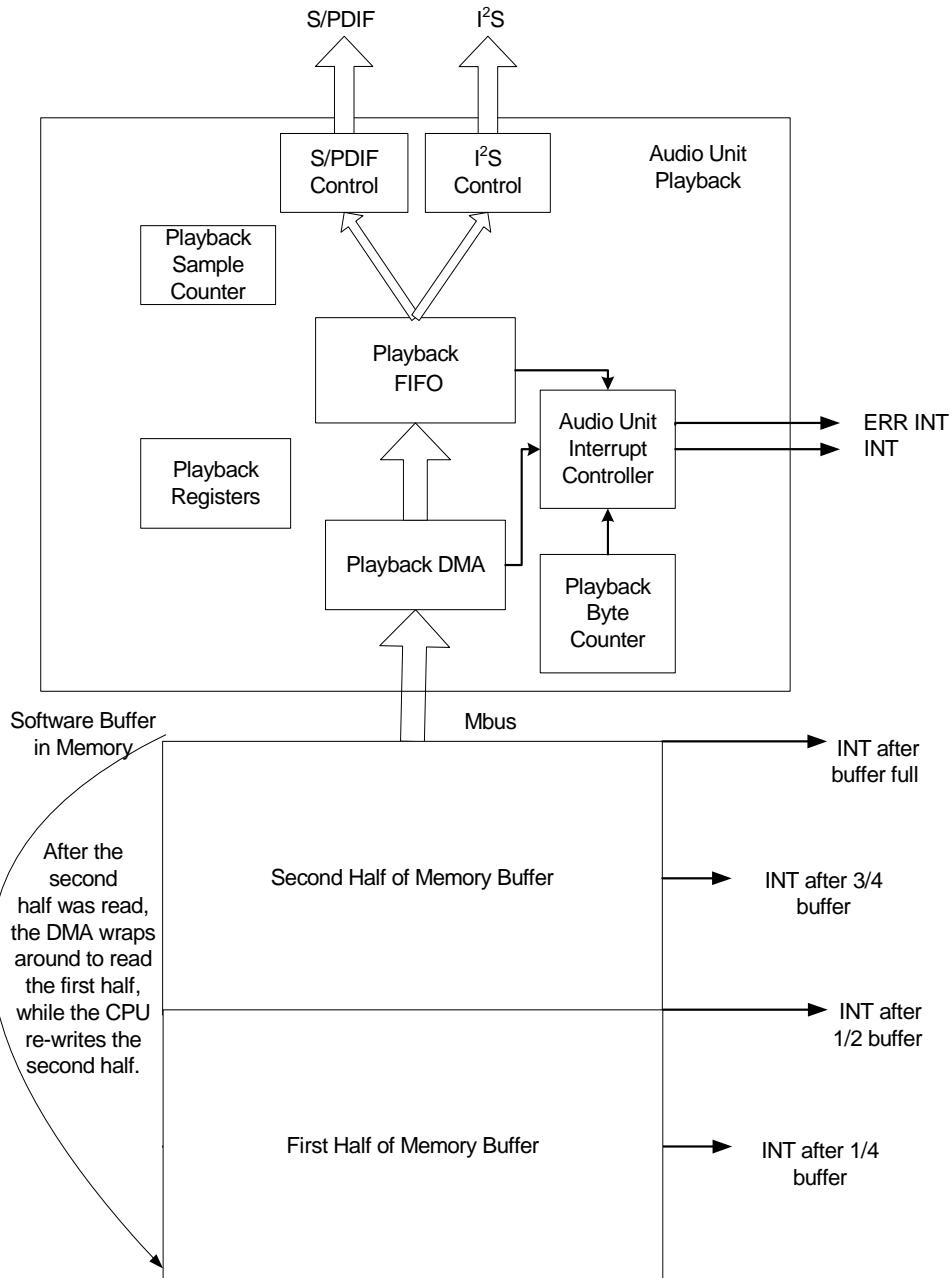
In this method, if an interrupt handler operation is delayed for a period of less than a full buffer playback cycle, the cause register still holds the information of which quarters of the buffer were played. This gives the software more time to recover from an unexpected interrupt handler delay.

**Configured Playback Byte Counter Method** The second method enables the user to configure the playback byte count for an interrupt.

Each time the dedicated counter reaches a pre-configured value in the Recorded Byte Counter for Interrupt Register ([Table 669 p. 709](#)), an interrupt is asserted in the Audio Interrupt Cause Register ([Table 650 p. 696](#)), and the counter is reset.

The advantage of this method is that it allows other user-defined interrupt resolutions.

**Figure 64: Playback Flow**



### 16.2.2 S/PDIF Specific Playback Flow

User bits can be sent either from the ADU internal buffers at:

- SPDIF Playback User Bits Left n Register (n=0–5) ([Table 663 p. 705](#))  
and
- SPDIF Playback User Bits Right n Register (n=0–5) ([Table 664 p. 706](#))

or from memory. When status and user bits are sent from the ADU internal buffers, they are updated during playback in parallel to the data. Before the start of each new played block, an S/PDIF start of block interrupt can be enabled. The software uses this interrupt to update the new ADU internal buffers status and/or user bits. The new bits are played together with the block following the one that has now started.

Status and user bits are double buffered, allowing the software approximately one block of playback time to update the new S/PDIF user and/or status bits.

### 16.2.3 Playback Mute

It is possible to mute either the I<sup>2</sup>S or the S/PDIF interface during simultaneous playback, or when only one of the interfaces is active.

During simultaneous playback with both interfaces enabled, the interfaces can be muted together or separately at any time. Muting is performed using the [<I<sup>2</sup>S Playback Mute>](#) field or the [<SPDIF Playback Mute>](#) field in the Playback Control Register ([Table 654 p. 701](#)).

During mute, the Playback-DMA and I<sup>2</sup>S clocks remain active.

During non-simultaneous I<sup>2</sup>S and S/PDIF playback control, when mute is activated, the S/PDIF user and status bits are automatically taken from the internal buffers. This means that the internal buffers should hold their required mute values before mute can be asserted.

### 16.2.4 Playback Pause

If the [<Playback Pause>](#) field in the Playback Control Register ([Table 654 p. 701](#)) is enabled, the playback DMA halts, but the I<sup>2</sup>S and the S/PDIF interfaces remain active and playing “silence”.

During pause, the S/PDIF user and status bits are automatically played from the internal buffers. This means that the internal buffers should hold their required pause values before pause can be asserted.

### 16.2.5 Sample Counters and DCO Run-Time Operation

The ADU includes a [<Playback Sample Counter>](#) field in the Playback Sample Counter Register ([Table 640 p. 693](#)) and a [<Recorded Sample Counter>](#) field in the Recording Sample Counter Register ([Table 641 p. 693](#)). These counters are incremented by each incoming and outgoing audio sample from the ADU I/O interface.

When S/PDIF recorded data is being played concurrently to being recorded and the DCO clock is used to clock the playback circuits, sample counters can be used to adjust the DCO playback frequency, in ppm resolution, to balance the playback rate with the recording rate. The recording rate is clocked by the external transmitter clock. This external clock frequency can differ slightly from the DCO clock frequency. Therefore, adjust the DCO playback frequency to avoid a long-term memory buffer overflow.

## 16.3 Error Handling

The ADU error handling of recording parity errors, recording overrun, and playback underrun is described in the following sections.

**Recording Parity Error** When an S/PDIF parity error is encountered during recording, the data is stored normally, recorded validity is asserted by the ADU, and an S/PDIF Recording Parity Error interrupt can be enabled in bit [1] of the Audio Error Mask Register ([Table 649 p. 696](#)).

**Recording Overrun** When the internal recording FIFO is full, new samples are dropped. The [\*\*<Recording Overrun Err>\*\*](#) field in the Audio Error Cause Register ([Table 648 p. 696](#)) is asserted. This event may occur, for example, when the Audio unit write accesses to the DDR are slower than the recording speed.

The field [\*\*<Recording Overrun Err>\*\*](#) can be masked by de-asserting bit [4] in the Audio Error Mask Register ([Table 649 p. 696](#)).

**Playback Underrun** If Playback underrun occurs, a Playback I<sup>2</sup>S Underrun Error interrupt or a Playback S/PDIF Underrun Error interrupt in the Audio Error Cause Register ([Table 648 p. 695](#)) can be enabled via bit [5] and bit [6], respectively, in the Audio Error Mask Register ([Table 649 p. 696](#)).

## 16.4

## Audio Unit Memory Structure

The memory structure for I<sup>2</sup>S and S/PDIF data transmit and receive can use 16-, 20-, or 24-bit sample sizes. I<sup>2</sup>S data transmit and receive may also use a 32-bit sample size.

For 16-bit playback and recording, two memory structure options exist:

**16-Bit Compact Mode** Data is stored in memory sequentially and there are no gaps, see Example A (stereo) and Example C (mono) in [Figure 65](#).

In this option, the 192 status and user bits per channel are double buffered in the ADU internal buffers. Validity, parity, and block start bits are not recorded.

On the playback path, the user and channel status bits are taken from the internal buffers that hold the 192 status bits and the 192 user bits per channel. These registers are configured per channel, and are sent automatically and repeatedly for every block.

For normal operations, the validity bit should be configured in the register file as:

- 0 for PCM traffic
- 1 for non-PCM traffic

### Full 16-Bit Mode

This option is depicted in Example B (stereo) and Example D (mono) in [Figure 65](#).

This option uses twice as much memory space and bandwidth versus the 16-bit Compact Mode. It allows the user to store status, validity, parity, block start and parity error indication bits values in memory, along with the data.

During playback the user bit, validity bit, and start of block can be played from memory on a sub-frame basis.



Non-linear PCM traffic always uses an S/PDIF 16-bit sample size.

### Note

To determine the sample size mode, configure the <Playback Sample Size> field in the Playback Control Register ([Table 654 p. 699](#)) and the <Recording Sample Size> field in the Recording Control Register ([Table 665 p. 706](#)). For I<sup>2</sup>S only, also configure the <I<sup>2</sup>S PB sample size> field in the I<sup>2</sup>S Playback Control Register ([Table 659 p. 704](#)), and the <I<sup>2</sup>S Recording Sample Size> field in the I<sup>2</sup>S Recording Control Register ([Table 670 p. 710](#)).

[Figure 65](#) displays the memory structure for data transmit and receive in 16-/20-/24-bit sample size for I<sup>2</sup>S and S/PDIF data.

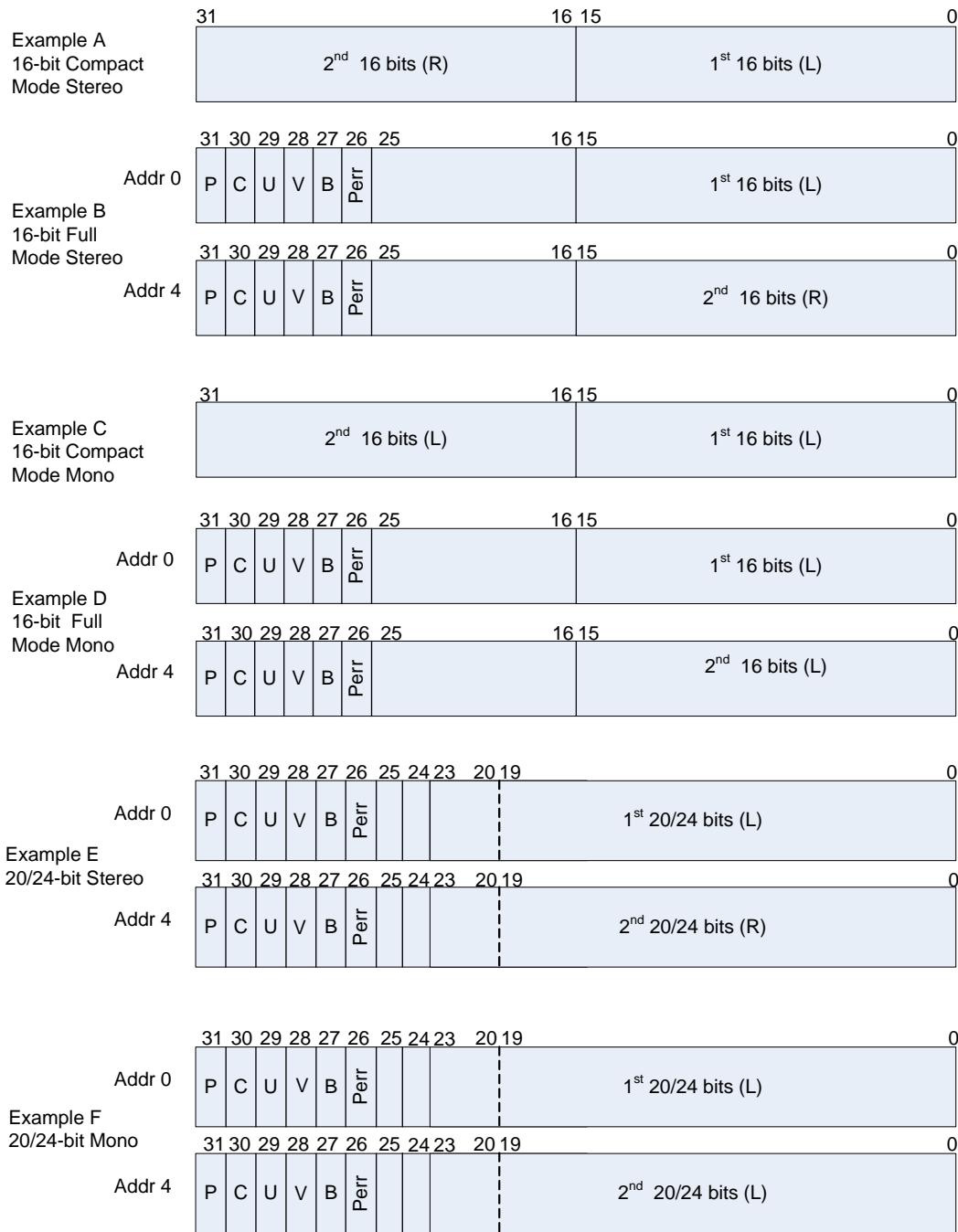


**Note**

Before using the Audio unit to play a PCM file, the user must first ensure that the PCM file memory structure is compatible with the ADU memory structure. If not, the PCM file must be converted to the Audio unit memory structure.

For example, an audio .wav file may use the following memory structure per sample size:

- **16-bit:** Same as the Audio unit 16-bit Compact Mode.
  - **20-bit:** Each 20-bit sample is stored in a 24-bit word, where bits[19:0] are stored in bits[23:4] of the 24-bits word. The four bits [3:0] are padded with zeros. The 24-bit word is stored in memory as described below.
  - **24-bit:** Each of the 24-bits is stored sequentially in memory without padding.
-

**Figure 65: Memory Structure for Transmit and Receive**


**Table 72** describes the meaning of each memory structure bit for all sample sizes, except for 16-bit Compact mode.

**Table 72: Audio Unit Memory Bit Description**

Bit	Symbol	Name	Description
31	P	S/PDIF parity bit	This bit is recorded but not used during playback.
30	C	S/PDIF channel status	This bit is recorded but not used during playback. Channel status bits are used from the ADU internal buffers.
29	U	S/PDIF user bit	User bit is recorded and can be used during playback from the ADU internal buffers or from memory.
28	V	S/PDIF validity bit	When the validity bit is recorded or played from memory, it is always asserted by the ADU when the Perr bit is asserted, or in the case of recording/playback mute or pause. Validity can be used during playback from the ADU register file or from memory, as configured in the SPDIF Playback Control Register ( <a href="#">Table 660 p. 704</a> ). When playing non-PCM audio, This bit should be asserted.
27	B	S/PDIF block start trigger	This bit is recorded and can be used during playback when the <a href="#"><b>&lt;SPDIF PB Block Start&gt;</b></a> field in the SPDIF Playback Control Register ( <a href="#">Table 660 p. 704</a> ) is asserted. When asserted, the B-bit indicates to the ADU to send the data sample with the start of block preamble. Block start should be computed internally, and not taken from the B-bit during playback when both user and status bits are taken from the ADU internal buffers. Otherwise, the block start should be taken from memory. During underrun or pause, or during mute in non-simultaneous S/PDIF and I <sup>2</sup> S playback, this bit is automatically computed by ADU and the B-bit is ignored.
26	Perr	S/PDIF parity error indication	This bit is used to indicate that recorded data has a parity error. Used in playback to assert validity in the event of a parity error indication. Perr can be used in playback to force a parity error in the outgoing sub-frame. If the <a href="#"><b>&lt;Force Parity Error&gt;</b></a> field in the SPDIF Playback Control Register ( <a href="#">Table 660 p. 704</a> ) is asserted and the <a href="#"><b>&lt;Perr&gt;</b></a> bit is asserted, the ADU computes an erroneous parity. This can be used to maintain the parity error that was detected in the recording. <b>NOTE:</b> When Perr is asserted and PCM data was recorded, the recorded Perr bit may equal the calculated parity since the ADU asserts the recorded validity when a parity error is detected.

### 16.4.1 Stereo Mode Alignment

In Stereo mode recording and playback, the left channel data is stored in memory locations with address2=0, and the right channel is stored in memory locations with address2=1.

When using the 16-bit Compact mode in Stereo mode, the left channel is stored in memory locations with address1=0, and the right channel is stored in memory locations with address1=1.

# 17

## Secure Digital Input/Output (SDIO) Interface

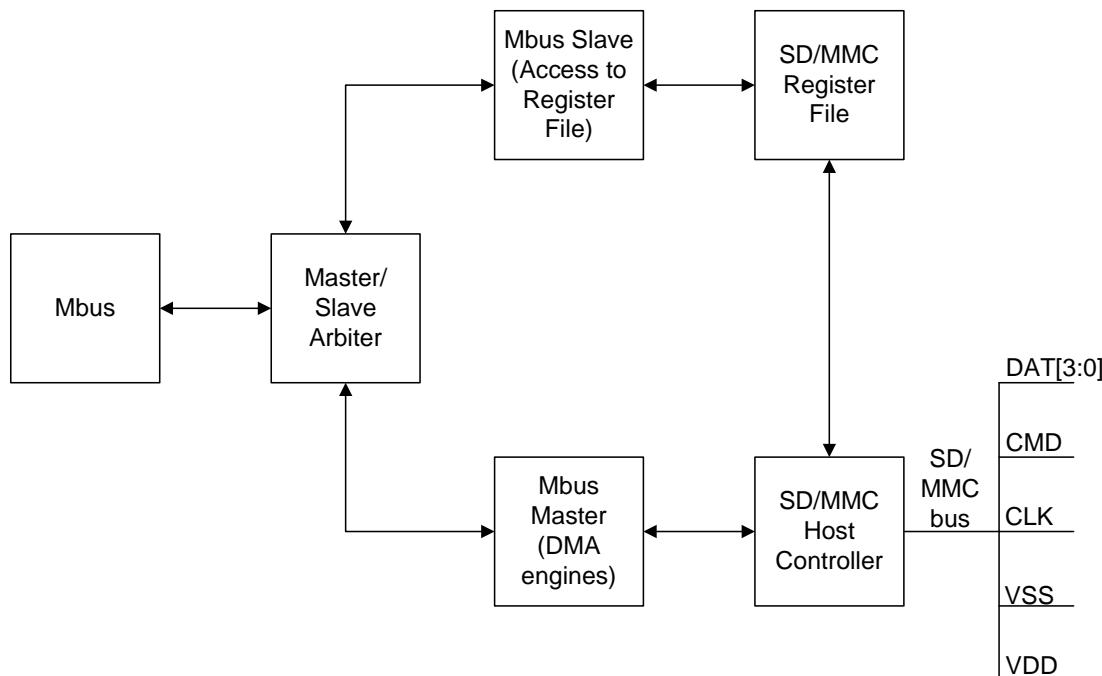
The device integrates a Secure Digital (SD) and MultiMedia card (MMC) host controller unit. This unit functions as a host for the SD/MMC bus to transfer data through the Mbus between SDMem, SDIO, and MMC cards on one side and system buffers (see [Figure 66](#)).

One side of the unit interfaces with a standard SD/MMC host bus.

The other side is programmable to interface either:

- Directly with the CPU—Mbus Slave
- From the DMA engines to the DRAM—Mbus Master

**Figure 66: SD\_MMIC Host Controller Hardware Block Diagram**



Communication over the SD/MMC bus is based on commands and data bit streams. They are initiated by a start bit and terminated by a stop bit.

<b>Command</b>	A command is a token that starts an operation. A command is sent from the host to the card(s). The command is transferred serially on the CMD line and multiplexed on the MPP as SD_CMD. The SD_CMD is bidirectional.
<b>Command response</b>	A command response is a token that is sent from an addressed card to the host as an answer to a previously received host command. The response is transferred serially on the CMD line and multiplexed on the MPP as SD_CMD.

<b>Data</b>	There are four bidirectional data transfer lines used to transfer data from the card to the host or vice versa. They are multiplexed on the MPP as SD_D[3:0].
<b>SD_CLK</b>	This clock synchronizes the SDIO bus of the device.



For more information, refer to the *SD Memory Card Specifications*.

**Note**

## 17.1 Features

The SDIO features supported by the device, and those features that are not supported are:

**Features supported:**

- 1-bit/4-bit SDMem, SDIO, and MMC cards
- Up to 50 MHz for SD and MMC
- Multiple MMC cards sharing a common bus
- Interrupts for information exchange between host and cards
- Read wait commands in SD cards
- Hardware generate/check CRC on all command and data transaction on card bus
- Suspend/resume in SDIO cards
- Stream read/write in MMC cards

**Features not supported:**

- Interrupts mode in the MMC card
- SPI mode

## 17.2 SDMem, MMC, and SDIO Arbitration Scheme

Host and SD/MMC cards go through two phases following each power on reset, software reset, or addition of a new card (see [Figure 67](#)):

**Card identification phase**

The host looks for new cards on the bus. While in this phase, the host resets all the cards that are in Card Identification mode. Any card that is already identified will not be reset. Then the host sends the command to validate the operation voltage range, identify the card, and request the card's Relative Card Address (RCA).

- In SDmem, this operation is done to each card separately on its own CMD line.
- In the MMC system, it is done on a shared CMD line.

All data communication in the Card Identification phase uses the CMD line only. The host starts the card identification process with an identification clock rate fOD that is slow (see *SD Memory Card Specifications*).

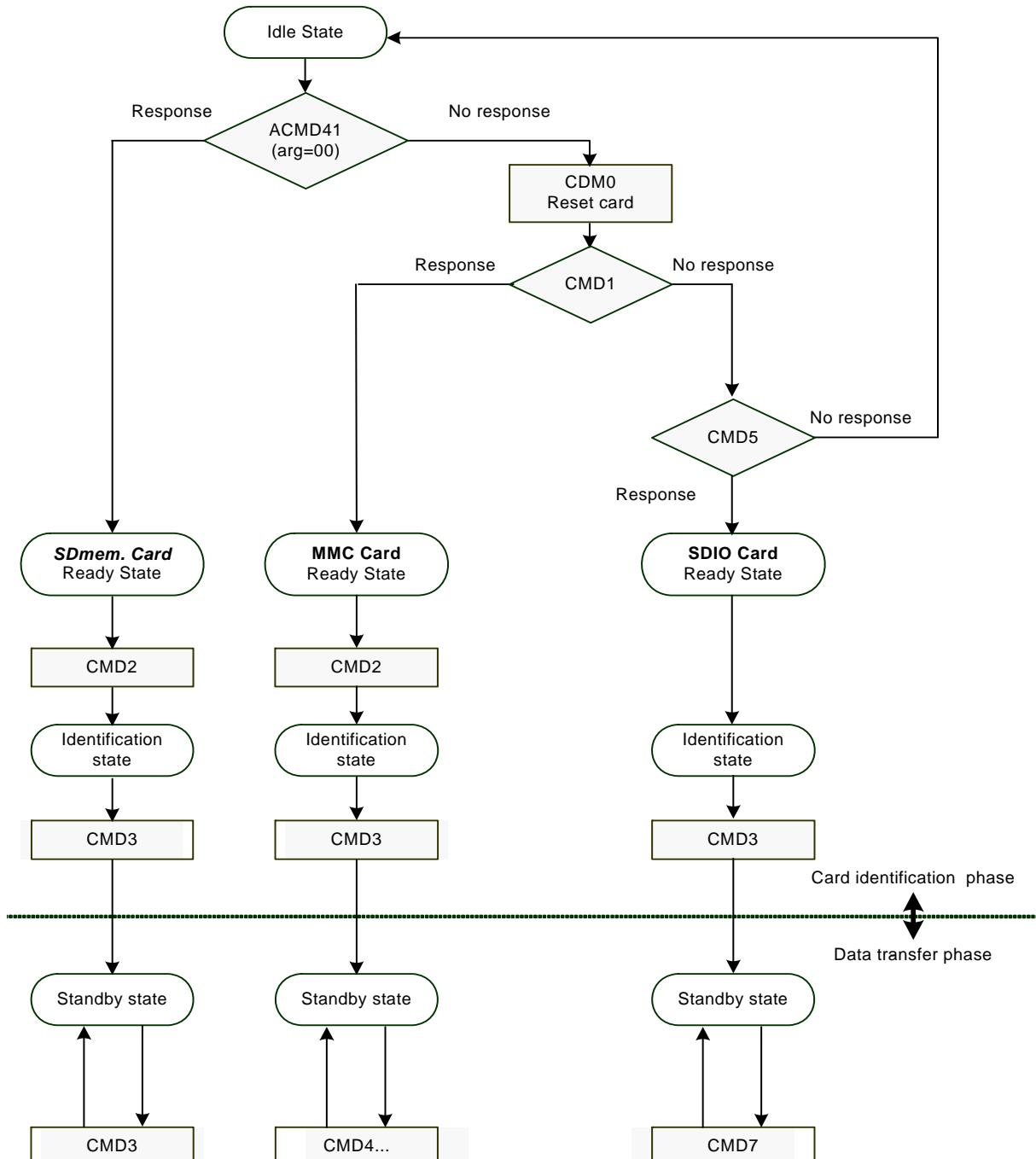
**Data transfer phase**

The host enters the data transfer phase after identifying all the cards on the bus. In this phase, the host is ready to transfer data.

After the host driver completes the setup of the host controller, it starts data transfer by writing to the Command Register ([Table 685 p. 719](#)). Therefore, the Command register has to be written last.

Figure 67 shows the Host initialization flow.

**Figure 67: Host Initialization Flow**



## 17.3

## Difference Between SD Cards and MMC Cards

The SD card protocol is designed to be a super-set of the MultiMedia card (MMC) protocol. However, besides some different commands in the initialization protocol, the only difference between the SD and MMC cards is the bus topology.

<b>SD card</b>	Each SD card has a dedicated, independent point-to-point connection—containing its own CLK, CMD, DATA lines—to the host. The I/O pads of the SD card are a push-pull type.
<b>MultiMedia card</b>	The MMC card bus is a shared bus, between multiple MMC cards with the MMC cards identified serially, one at a time. Therefore, during initialization, MMC cards use open-drain I/O pad types. In the data transfer phase, MMC cards switch to push-pull I/O type just like the SD cards.  To the host, this shared bus makes no difference. However, each of the MMC cards has to monitor the bus to determine whether or not it has won the bus.

To support both bus topologies, the host controller functions as an open-drain during the initialization phase, and functions as a push-pull type during the data transfer phase. For the CMD and DATA lines, for both SD cards and MMC cards, the type needs to be pull-up.

## 17.4

## SDIO / SDMem / MMC Host Controller Initialization

If the SD/MMC host controller requires a configuration different from the default, the software should configure the following registers:

- Mbus Control Low Register ([Table 716 p. 739](#)) (offset: 0x80100)
- Mbus Control High Register ([Table 717 p. 739](#)) (offset: 0x80104)
- Window0 Control Register ([Table 718 p. 740](#)) (offset: 0x80108)—size, target, and attributes
- Window0 Base Register ([Table 719 p. 740](#)) (offset: 0x8010C)
- Window1 Control Register ([Table 720 p. 741](#)) (offset: 0x80110)—size, target, and attributes
- Window1 Base Register ([Table 721 p. 741](#)) (offset: 0x80114)
- Window2 Control Register ([Table 722 p. 741](#)) (offset: 0x80118)—size, target, and attributes
- Window2 Base Register ([Table 723 p. 742](#)) (offset: 0x8011C)
- Window3 Control Register ([Table 724 p. 742](#)) (offset: 0x80120)—size, target, and attributes
- Window3 Base Register ([Table 725 p. 743](#)) (offset: 0x80124)

## 17.5

## SDIO / SDMem / MMC Command Execution

The SD/MMC command is activated by configuration of the relevant SD registers (see [Table 677, Register Map Table for the SDIO Registers, on page 714](#)). Some commands (e.g., Auto Cmd12) require the setting of dedicated registers before executing the command. Software will set appropriate registers before writing Command to the Command Register ([Table 685 p. 719](#)). Upon writing Command to the Command register, the hardware automatically starts command execution, data transfer, and CRC generation or checking. Therefore, the Command register is the last register to be written by software. The command completion could be detected by interrupt assertion or by polling Interrupt Status Registers (offsets: 0x80060–0x80074).



Table 73 shows an example of the software flow for the SDIO command execution. Configure the registers listed in this table:

**Table 73: Software Flow**

Register Name	Offset	Field
DMA Buffer Address 16 LSB Register ( <a href="#">Table 678 p. 715</a> )	0x80000	<DmaAddrLo>
DMA Buffer Address 16 MSB Register ( <a href="#">Table 679 p. 715</a> )	0x80004	<DmaAddrHi>
Data Block Size Register ( <a href="#">Table 680 p. 716</a> )	0x80008	<BlockSize>
Data Block Count Register ( <a href="#">Table 681 p. 716</a> )	0x8000C	<BlockCount>
Argument in Command 16 LSB Register ( <a href="#">Table 682 p. 716</a> )	0x80010	<ArgLow>
Argument in Command 16 MSB Register ( <a href="#">Table 683 p. 717</a> )	0x80014	<ArgHigh>
Host Control Register ( <a href="#">Table 697 p. 724</a> )	0x80050	<PushPullEn> <CardType> <BigEndian> <LsbFirst> <DataWidth> <HiSpeedEn> <TimeoutValue> <TimeoutEn>
Data Block Gap Control Register ( <a href="#">Table 698 p. 727</a> )	0x80054	<Resume>
Clock Control Register ( <a href="#">Table 699 p. 728</a> )	0x80058	<SclkMasterEn>
Argument in Auto Cmd12 Command 16 LSB Transferred Register ( <a href="#">Table 710 p. 737</a> )	0x80084	<AutoCmd12ArgLo>
Argument in Auto Cmd12 Command 16 MSB Transferred Register ( <a href="#">Table 711 p. 737</a> )	0x80088	<AutoCmd12ArgHi>
Index of Auto Cmd12 Commands Transferred Register ( <a href="#">Table 712 p. 738</a> )	0x8008C	<AutoCmd12BusyChkEn> <AutoCmd12IndexChkEn> <AutoCmd12Index>
Transfer Mode Register ( <a href="#">Table 684 p. 717</a> )	0x80018	<SwWrDataStart> <HwWrDataEn> <AutoCMD12En> <IntChkEn> <DataXferTowardHost> <StopClkEn> <HostXferMode>
Command Register ( <a href="#">Table 685 p. 719</a> )	0x8001C	<ResponseType> <DataCrc16ChkEn> <CmdCrcChkEn> <CmdIndexChkEn> <DataPresent> <UnexpectedRespEn> <CmdIndex>

**NOTE:** After this sequence, the Command Register ([Table 685 p. 719](#)) fields must be configured. Writing to the command register is the trigger for SDIO command execution.

## 17.6

## SDIO / SDMem / MMC Interrupts

The SD/MMC interrupts are registered in two Interrupt Cause registers:

- Normal Interrupt Status Register ([Table 701 p. 729](#)) (offset: 0x80060)
- Error Interrupt Status Register ([Table 702 p. 730](#)) (offset: 0x80064)

The interrupt going from the unit to the CPU is generated as a bitwise OR of all bits of these registers. Upon an interrupt event, the corresponding cause bit is set to 1. It is cleared upon a software write of 0.

Each bit in the [Normal Interrupt Status Register](#) can be enabled or disabled by the relevant bit in the Normal Interrupt Status Enable Register ([Table 703 p. 732](#)) (offset: 0x80068).

The impact of each bit on SD/MMC interrupt generation may be enabled or disabled by the relevant bit in Normal Interrupt Status Interrupt Enable Register ([Table 705 p. 734](#)) (offset: 0x80070).

Each bit in [Error Interrupt Status Register](#) can be enabled or disabled by the relevant bit in the Error Interrupt Status Enable Register ([Table 704 p. 733](#)) (offset: 0x8006C).

The impact of each bit on SD/MMC interrupt generation may be enabled or disabled by the relevant bit in Error Interrupt Status Interrupt Enable Register ([Table 706 p. 735](#)) (offset: 0x80074).

The following interrupt events are supported:

### Normal Interrupts:

- Command Complete
- Transfer Complete
- Block gap event
- DMA interrupt
- TX ready—the FIFO has room for the CPU to write 16 bits of data.
- RX ready—the FIFO contains at least 1 byte of data ready to be read by the CPU.
- Cards interrupt.
- Read Wait state is on.
- There are at least eight filled entries for data to be read from the FIFO.
- There are at least eight empty entries for data to be written in the FIFO.
- The hardware is suspended.
- Auto\_cmd12 is completed.
- Unexpected response from devices detected.

### Error Interrupts:

- Command timeout error
- Command CRC error
- Command end bit error
- Command start bit error
- Command index error
- Auto CMD12 error
- Data timeout error
- Read data CRC error
- Read data end bit error
- Transfer size mismatched error
- Response T bit error
- CRC end bit error
- CRC start bit error
- CRC status error

# 18

## Transport Stream (TS) Interface (88F6192 and 88F6281 Only)

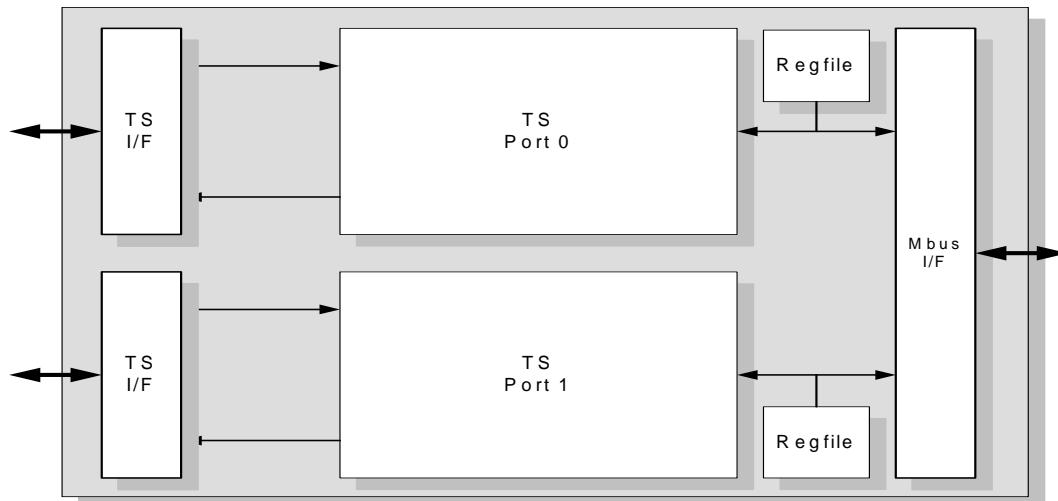
The 88F6192 and 88F6281 implement a Transport Stream Unit (TSU) that is responsible for handling MPEG-2 Transport Stream (TS) Interface format. This packet-based format transfers video, audio, and other information such as timestamps, as defined in ISO/IEC 13818-1.

The TSU includes two ports, supporting the following parallel and serial modes:

- Single parallel input (only Port 0 is active)
- Single parallel output (only Port 0 is active)
- Dual serial outputs (two ports are active)
- Dual serial inputs (two ports are active)
- Single serial input and single serial output (two ports are active)

[Figure 68](#) illustrates the flow to/from each TS port, TS interface, and Mbus internal interface.

**Figure 68: TSU Block Diagram**



The TSU implements the following features:

- ISO/IEC 13818-1 format compatibility
- Sampling/sending data with rising or falling edge of the clock
- Support of configurable control signals polarity

## 18.1

## TS Port Architecture

Unless specifically noted, the information in this section refers to a single TS port. The two ports are identical and have the same features.

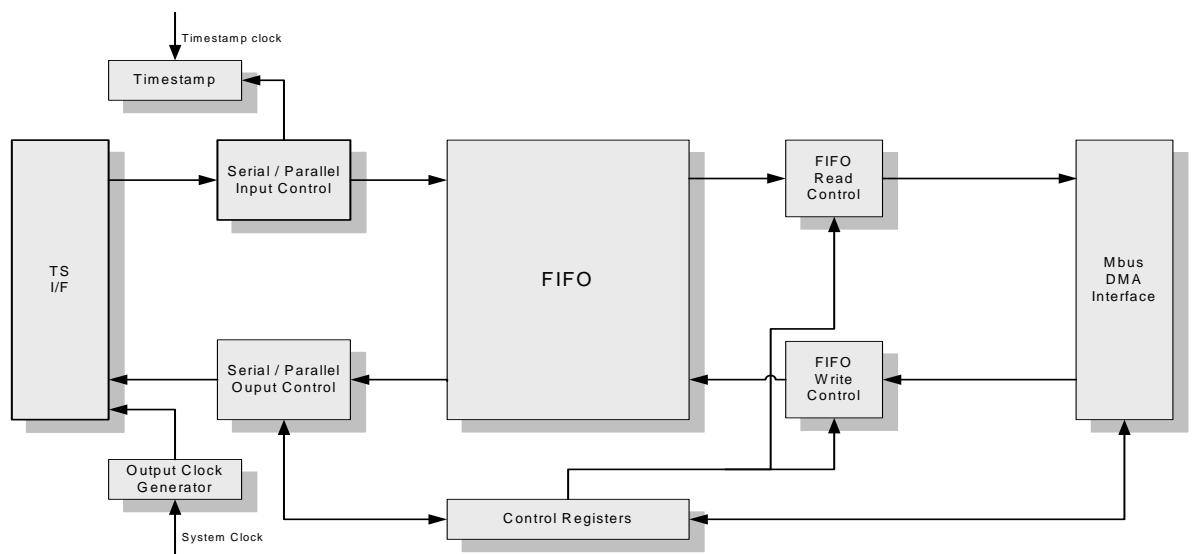
The TS interface can handle both input and output TS streams, when one direction is used at a time. The Input and Output Control Blocks are configurable for either serial or parallel mode. They can process data byte-wise or bit-wise and sample/send data with the rising or falling edge of the clock. Polarity of the control signals is also configurable. The Input Control Block samples the timestamp of the received TS packet. A FIFO is implemented, to compensate for bus latency on the input path and process time on the output path. A Control Registers block is implemented to configure the TS interface.

The Mbus Direct Memory Access (DMA) interface controls data transfer from memory to the TS interface and vice versa. It also handles register accesses from the CPU. The DMA interface processes a chain of descriptors. It is triggered to fetch the next descriptor, based on the TS Descriptor Write and Read Pointer scheme. Data transfer in the TS input direction starts every time TS data is available in the TS port FIFO. In the TS output direction, the data transfer from memory over the Mbus to the FIFO is triggered by a timer. This timer is controlled by the firmware, based on the timestamps of the received TS packets. Therefore, the firmware and the TS port can correct any network jitter.

After each data transfer, the TS port stores a status double-word in the SDRAM Done queue.

[Figure 69](#) provides a block diagram of the TS interface.

**Figure 69: TS Interface Block Diagram**



## 18.2

## TS Interface

The TS port includes a set of pins that can be configured to support input or output configurations. In addition, this set of pins can be configured in serial or parallel mode.

TS Serial mode is a subset of Synchronous Parallel Interface (SPI) mode, using the same control interface, but only one TS\_DATA[0].

The interface defined in [Table 74](#) fulfills the following requirements:

- Selection of interface mode (serial/parallel) is programmable via the [\*\*<TS Data Mode>\*\*](#) field in the TS Interface Configuration Register ([Table 741 p. 751](#)).
- Selection of interface direction (input/output) is programmable via [\*\*<TS Data Direction>\*\*](#).
- Supports of up to 256-byte TS packet size, programmable via [\*\*<TS Packet Size>\*\*](#).
- Polarity of TS\_SYNC is programmable via [\*\*<TS\\_SYNC Polarity>\*\*](#).
- Polarity of TS\_VAL is programmable via [\*\*<TS\\_VAL Polarity>\*\*](#).
- Polarity of TS\_ERR is programmable via [\*\*<TS\\_ERR Polarity>\*\*](#).
- A complete TS packet can be output on consecutive clock cycles.
- Signals TS\_DATA[7:0] remain low during inter-packet gaps (SPI).
- Signal TS\_ERR is active during the entire TS frame.
- Data transition is programmable to be either on the rising or falling edge of the TS\_CLK via [\*\*<TS Data Transmission Edge>\*\*](#).

For Serial mode the following additional requirements are fulfilled:

- Signals TS\_DATA[7:1] are held low.
- Serial data stream is programmable to output either in an MSB or LSB first direction via [\*\*<TS Serial Data Order>\*\*](#).
- Signal TS\_SYNC is programmable to be active during the entire first byte or the first bit only via [\*\*<Serial TS\\_SYNC Active>\*\*](#).
- TX\_CLK is programmable to either gapped or continuous clock mode via [\*\*<TS Serial Clock Mode>\*\*](#).
- In continuous mode the clock runs without regard to data being output. TS\_VAL is used as a data strobe.

**Table 74: Transport Stream (TS) Interface Signal Assignment**

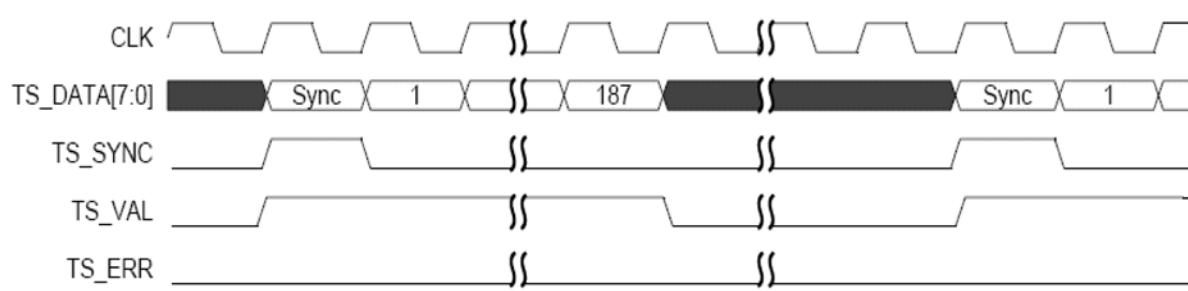
Pin Name	I/O	Description
TSMP[0]	I	EXT_CLK External clock that can be used to drive the TS0_CLK and TS1_CLK
TSMP[1]	I/O	TS0_CLK Port0 TS clock. • If TS0_VAL is used, the clock may be continuous. • If TS0_VAL is not used, the clock may toggle only when valid data is available on TS0_DATA.
TSMP[2]	I/O	TS0_SYNC Port0 Sync/Frame Start Indicator or Packet Clock. The TS0_SYNC in parallel mode is a pulse that is active during the first (Sync) byte of the TS packet. In serial mode, the TS0_SYNC pulse may be active for the entire byte or only for the first bit. The polarity is programmable to be either active high or active low.
TSMP[3]	I/O	TS0_VAL Port0 Valid Data Indicator When this signal is used and is valid, it indicates that valid data is present on TS0_DATA. TS0_VAL is active during the TS frame packet data and inactive when there is no TS synchronization. In output mode, the polarity of TS0_VAL is programmable to be either active high or active low.

**Table 74: Transport Stream (TS) Interface Signal Assignment (Continued)**

Pin Name	I/O	Description
TSMP[4]	I/O	<p>TS0_ERR Port0 Uncorrectable Packet Error When this signal is used, an error indicates that the packet contains an uncorrectable error, and therefore should not be used. In output mode, the TS0_ERR is active during the entire TS frame.</p>
TSMP[5]	I/O	<p>TS0_DATA[0] Port0 TS Data bit 0 in both parallel and serial modes. In Serial mode TS0_DATA[0] is used as data input or output.</p>
TSMP[6]	I/O	<ul style="list-style-type: none"> <li>Parallel Mode: TS0_DATA[1]: Port0 TS Data bit 1</li> <li>Serial Mode: TS1_CLK: Port1 TS clock. - If TS1_VAL is used, the clock may be continuous. - If TS1_VAL is not used, the clock may toggle only when valid data is available on TS1_DATA</li> </ul>
TSMP[7]	I/O	<ul style="list-style-type: none"> <li>Parallel Mode: TS0_DATA[2]: Port0 TS Data bit 2</li> <li>Serial Mode: TS1_SYNC: Port1 Sync/Frame Start Indicator or Packet Clock. The TS1_SYNC pulse may be active for the entire byte or only for the first bit. The polarity is programmable to be either active high or active low</li> </ul>
TSMP[8]	I/O	<ul style="list-style-type: none"> <li>Parallel Mode: TS0_DATA[3]: Port0 TS Data bit 3</li> <li>Serial Mode: TS1_VAL: Port1 Valid Data Indicator When this signal is used and is valid, it indicates that valid data is present on TS1_DATA[0]. TS1_VAL is active during the TS frame packet data and inactive when there is no TS synchronization. In output mode, the polarity of TS1_VAL is programmable to be either active high or active low.</li> </ul>
TSMP[9]	I/O	<ul style="list-style-type: none"> <li>Parallel Mode: TS0_DATA[4]: Port0 TS Data bit 4</li> <li>Serial Mode: TS1_ERR: Port1 Uncorrectable Packet Error When this signal is used, an error indicates that the packet contains an uncorrectable error, and, therefore, should not be used. In output mode the TS1_ERR is active during the entire TS frame.</li> </ul>
TSMP[10]	I/O	<ul style="list-style-type: none"> <li>Parallel Mode: TS0_DATA[5]: Port0 TS Data bit 5</li> <li>Serial Mode: TS1_DATA[0]: Port1 TS Data bit 0, used as data input or output.</li> </ul>
TSMP[11]	I/O	<p>TS0_DATA[6] Port0 TS Data bit 6 This pin is only valid in Parallel mode.</p>
TSMP[12]	I/O	<p>TS0_DATA[7] Port0 TS Data bit 7 This pin is only valid in Parallel mode.</p>

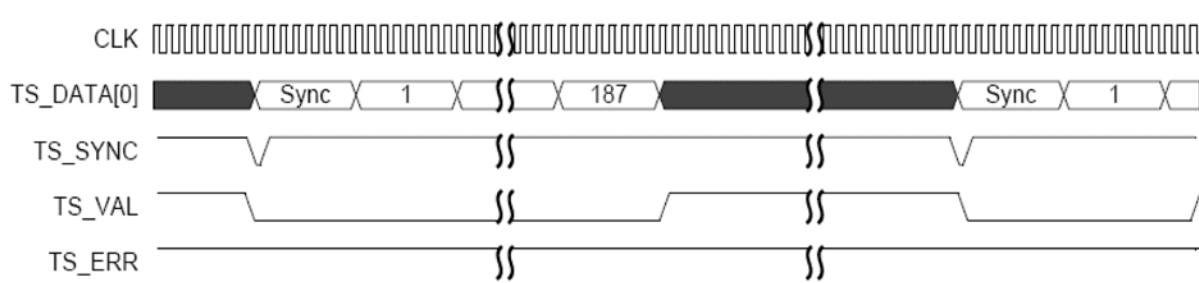
[Figure 70](#) is an example of the parallel TS data protocol, with active high control signals and data transition on the rising edge of the TS\_CLK.

**Figure 70: TS Parallel Protocol (Example)**



[Figure 71](#) is an example of continuous serial TS data protocol with active low control signals and data transition on the falling edge of TS\_CLK.

**Figure 71: TS Continuous Serial Data Protocol (Example)**



## 18.3 Clocks

The TS does not support any clock recovery mechanism for generating the output clock on the basis of packets received at any input interface.

Mode is selected by the firmware, which analyzes the timestamp of the TS packets and calculates the TS data rate of the sending device. Once the data rate of the sender has been determined, the firmware selects the output clock mode as follows:

Sender data rate  $\leq$  mode data rate

The TS port output clock cycles are controlled by the TSU source clock, as supplied to the TSU from the device and the [Output Clock Frequency](#) field in the TS Interface Configuration Register ([Table 741 p. 751](#)) (which supplies the three different options for serial and parallel interfaces).

The Output clock is configured to supply:

- The same data rate as supplied by the source clock
- Half of the source clock data rate
- A quarter of the source clock data rate

The TSU source clock from the device must be configured before the TSU exits from reset by programming (see the [<TSCK88>](#) field in the TSU Modes Register ([Table 730 p. 744](#))).

**Timestamp Counter Clocking:** The timestamp counter is clocked by an internally generated clock with a period of 192 ns.

## 18.4

## TS Input Data Flow

TS input data flow is as follows:

1. During initialization the firmware programs the Control registers to set up TS signals and set the Control Block to input mode.
2. Received TS packets (either in serial or parallel mode) are stored in the FIFO. A timestamp for each packet is calculated, by deriving a sampling instant from the clock, which is constantly incrementing.
3. TSU transfers data into the memory (The buffer address was previously supplied by the firmware.) and also writes the status double-word (timestamp, errors information) for each packet into the memory. Then it informs the firmware that a new packet is available.
4. After having updated the remaining header fields, the firmware writes the start address of the packet into the pointer area of the memory.

## 18.5

## TS Output Data Flow

TS output data flow is as follows:

1. During initialization the firmware programs the Control registers to set up TS signals and set the Control Block to output mode.
2. Firmware configures the TS port to load data from the memory into the TSU port FIFO. For this reason the firmware provides the start address of data, the number of bytes, and a timer value. Based on the timer value the TSU can build the inter-packet gaps of the TS input interface on the TS output interface and corrects network jitter and delay.
3. Data transfer of each TS packet from the memory is triggered by the TS port timer.
4. Inside the TSU, the data is transferred from the DMA interface into the FIFO.
5. The TS port writes a status double-word for each packet into the memory.

## 18.6

## DMA Engine

The TS port manages packet stream transfers between the memory and the TS interface. Data is stored in the memory buffers and managed by descriptors.

The TS port includes the DMA engine, which handles the packets and descriptor transfers from the memory to TS and vice versa.

The TS port works in half-duplex mode, i.e., TS packets are transferred in only one direction at a time.

The TS port manages the following data structures to handle data flow:

- TS port DMA Engine includes two descriptor registers—one for the descriptor of the current TS packet (Current Descriptor) and one for the descriptor of the next TS packet (Next Descriptor).
- TS Descriptor queue: Stores one double-word for each TS input direction packet and two double-words for each output direction packet in the memory. The TS Descriptor queue is defined by the following registers:
  - Queue Base Address (Current Descriptor Register ([Table 749 p. 754](#))),
  - Write Pointer (TSU Enable Access Register ([Table 752 p. 755](#)))
  - Read Pointer (TSU Timestamp Register ([Table 753 p. 756](#))).
- TS Done queue: Stores one double-word for each TS input and output packet in the memory. The TS Done queue is defined by the following registers:

- Queue Base Address (Descriptor Queue Read Pointer Register ([Table 748 p. 754](#))),
- Write Pointer (Current DMA Address Register ([Table 750 p. 755](#)))
- Read Pointer (Current DMA Length Register ([Table 751 p. 755](#))).

All descriptors in the TS Descriptor queue and in the TS Done queue start at a double-word boundary.

## 18.6.1 TS Input Direction Data Structures

### 18.6.1.1 TS Input Descriptor Structure

Each TS input descriptor consists of a 32-bit register containing the buffer address of the packet to be received.

### 18.6.1.2 TS Input Descriptor Queue Structure

The TS Input Descriptor queue is located in the CPU memory and has up to 1024 entries. It is organized as a circular queue (FIFO) whose location is configured by the following registers:

- Current Descriptor Register ([Table 749 p. 754](#))
- TSU Enable Access Register ([Table 752 p. 755](#))
- Descriptor Queue Read Pointer Register ([Table 748 p. 754](#)).

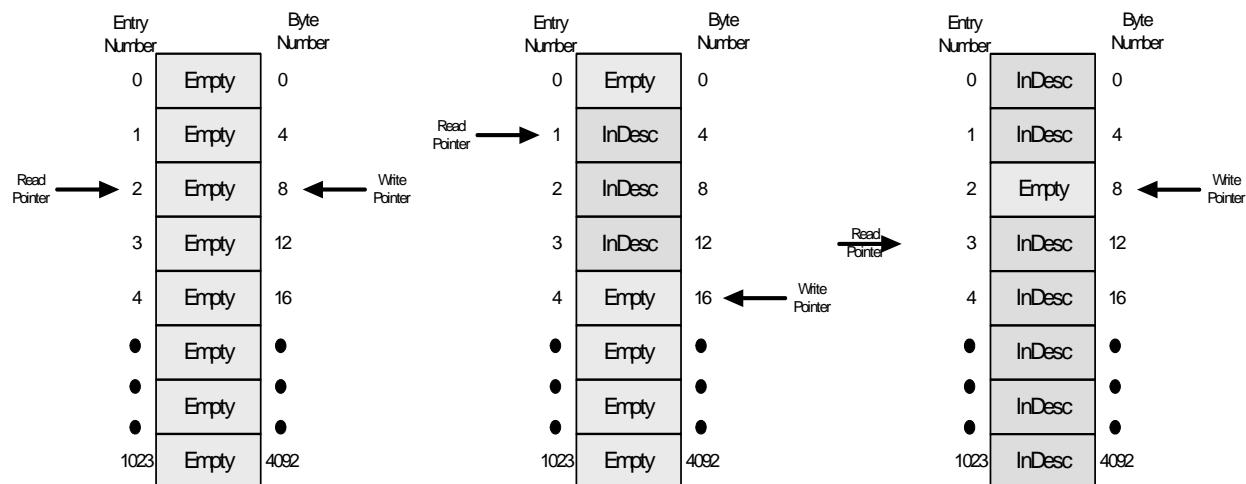
The queue Write and Read pointers are defined as byte addresses. Adding them to the Queue Base address results in the byte address of the queue entry.

The TS Descriptor queue size is determined by programming the [\*<TS Descriptor Queue Size>\*](#) field in the TS DMA Parameter Register ([Table 742 p. 752](#)), which defines the number of double-words the queue consists of, and as a result, the number of descriptors.

The circular queue structure is maintained using Read Pointer and Write Pointer rollover bits, whose locations are derived from the programmed queue size.

The TS Input Descriptor queue is the interface used by firmware to queue valid descriptors for managing incoming TS traffic through the device memory. The Write Pointer is controlled by TSU and is incremented whenever a valid descriptor is placed for use. The Read Pointer is controlled by the firmware and is incremented whenever a descriptor from the queue is used for pointing to a new received TS packet in the memory (see [Figure 72](#)).

**Figure 72: TS Input Descriptor Queue**



### 18.6.1.3 TS Input Done Queue Structure

The TS Done queue is used for storing completion status for each received TS packet. It is located in the CPU memory and is organized as a circular queue whose location is configured by:

- Descriptor Queue Read Pointer Register ([Table 748 p. 754](#)), Current DMA Address Register ([Table 750 p. 755](#))
- Current DMA Length Register ([Table 751 p. 755](#)).

Queue size is determined by programming the [`<TS Done Queue Size>`](#) field in the TS DMA Parameter Register ([Table 742 p. 752](#)).

The TS Input double-word Status consists of the following fields in TSU Status Register ([Table 754 p. 756](#)):

- [`<TS Connection Error>`](#)
- [`<TS FIFO Overflow Error>`](#)
- [`<TS IF Error>`](#)
- and the [`<Timestamp>`](#) field in the TSU Timestamp Register ([Table 753 p. 756](#)): Timestamp value

### 18.6.2 TS Input Direction DMA Flow

If the TSU is configured as an input device, received TS packets are handled on serial or parallel input interfaces and stored in the memory.

At initialization, the following steps are taken, to program the TSU to handle input:

1. Allocate buffers with predefined headers in the memory.
2. Configure the TSU to TS Input mode.
3. Set up descriptors for the TSU in the TS Descriptors queue. All the descriptors are one double-word per TS packet. The descriptors act as pointers to the start of the data payload area within the buffer, or in case of aggregation to the start offset for the next TS packet, within the buffer.
4. Update the TSU Enable Access Register ([Table 752 p. 755](#)).

Each time the next descriptor is not valid and the TS Descriptor Write Pointer differs from the TS Descriptor Read Pointer, the DMA engine fetches a descriptor in the Next Descriptor register from

the TS Descriptor queue in the memory. The descriptor is transferred from the Next Descriptor register to the Current Descriptor register at the beginning of each TS packet transfer.

As soon as data arrives in the TS interface, the FIFO of the DMA engine is filled. As soon as the watermark is reached or the last byte of the TS packet arrives in the FIFO, a DMA transfer is triggered, with the size of one watermark or the number of bytes up to End of Packet (EOP), whichever is lower. After the DMA has been performed, the Current DMA Address is incremented, and the Length Pointer is decremented by the number of bytes transferred. The TSU generates an interrupt. While the CPU manages the received interrupt, the TSU can receive and transfer more packets but cannot interrupt the CPU unless the interrupt is cleared. For handling the received packets, the firmware can determine from the TS Descriptor queue Read Pointer, how many buffers have been used to store received packets. The status of the received packets is stored in the TS Done queue. Once the received packets have been processed, the TS Descriptor queue and TS Descriptor Write Pointer must be updated, to re-enable TS input interrupt. Another option is that the TSU generates an interrupt only if the difference between the TS Descriptor Write Pointer and the TS Descriptor Read Pointer is less than the programmable value <Descriptor IRQ Threshold> field in the IRQ Parameter Register ([Table 759 p. 759](#)). The interrupt triggers the firmware to process new TS packets and generate new descriptors.

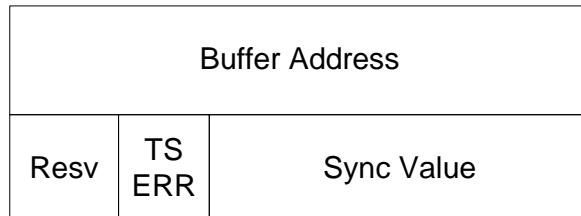
## 18.6.3 TS Output Direction Data Structures

### 18.6.3.1 TS Output Descriptor Structure

When Type 1 Aggregation or Type 2 Aggregation is used, each output descriptor consists of one 32-bit register that holds the buffer address (see [Section 18.8, TS Packet Aggregation, on page 273](#)).

When aggregated modes are not used, each output descriptor consists of two 32-bit registers, as depicted in [Figure 73](#).

**Figure 73: TS Output Descriptor Structure—No Packet Aggregation**



The TS Output Descriptor includes the following fields:

<b>Buffer Address</b>	Bits[31:0] of the first register	Buffer address of the packet to be transmitted.
<b>Sync Value</b>	Bits[27:0] of the second register	Timestamp timer value for this packet. Used for packet transmission scheduling.
<b>TS ERR</b>	Bit[28] of the second register	If set, the TSU asserts a TS_ERR signal on the TS interface when sending this packet.

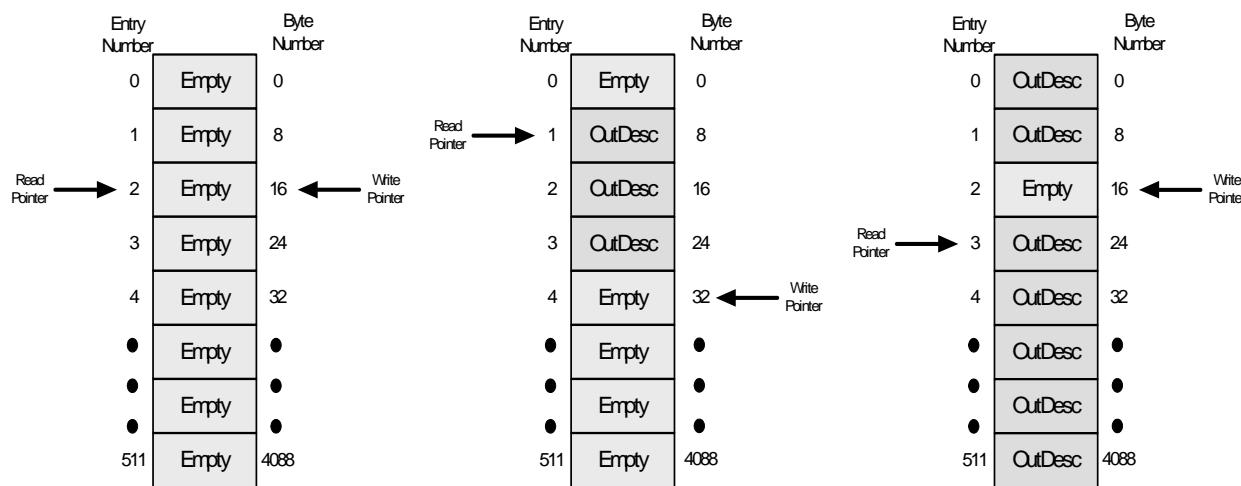
### 18.6.3.2 TS Output Descriptor Queue Structure

When packet aggregation is used, the TS output queue structure is the same as the TS input descriptor queue structure (see [Section 18.6.1.2, TS Input Descriptor Queue Structure, on page 268](#)).

The remainder of this section describes the TS output queue structure when packet aggregation is not used.

The TS Output Descriptor queue is located in the CPU memory and is organized as a circular queue (FIFO) of up to 512 entries. The queue is controlled by the same pointers as those used for managing the TS Input Descriptor queue, with the exception that each entry is two double-words.

**Figure 74: TS Output Descriptor Queue—No Packet Aggregation**



### 18.6.3.3 TS Output Done Queue Structure

The TS Output Done queue is used for storing completion status for each transmitted TS packet. The TS Output Done queue is located in the CPU memory and is organized as a circular queue whose location is configured by:

- Descriptor Queue Read Pointer Register ([Table 748 p. 754](#))
- Current DMA Address Register ([Table 750 p. 755](#))
- Current DMA Length Register ([Table 751 p. 755](#)).

Queue size is determined by programming the <TS Done Queue Size> field in the TS DMA Parameter Register ([Table 742 p. 752](#)).

The TS Output Status consists of the following field:

TS FIFO Underflow Error - Bit[28]. TS FIFO Underflow Error signaled by the interface

### 18.6.4 TS Output Direction DMA Flow

At initialization the firmware configures the TSU to TS Output mode. It creates the address pointers of the descriptors pointing to the start of the TS packets within the received packets. In addition, the firmware generates the status fields of the descriptors, writes the descriptors to the TS Descriptor queue, and updates the TS Descriptor queue Write Pointer. Each time the Next Descriptor is not valid and the TS Descriptor Write Pointer differs from the TS Descriptor Read Pointer, the DMA engine fetches a descriptor in the Next Descriptor register from the TS Descriptor queue in the memory. The descriptor is transferred from the Next Descriptor register to the Current Descriptor register at the beginning of each TS packet transfer.

The data DMA for each packet is timer-triggered (see [Section 18.7, TS Timestamp Mechanism](#)). All data DMAs within the packet start as soon as enough space is available in the FIFO (watermark size). DMA size in one watermark or the number of bytes up to EOP, whichever is lower. After the

DMA has been performed, the Current DMA Address is incremented and the Length Counter is decremented by the number of bytes transferred. If the Length Counter is 0, the status is written to the TS Done queue, and the TS Done queue Write Pointer and TS Descriptor Read Pointer are incremented.

The TS port only generates an interrupt if the difference between the TS Descriptor Write Pointer and the TS Descriptor Read Pointer is smaller than the programmable value [Descriptor IRQ Threshold](#) field in the IRQ Parameter Register ([Table 759 p. 759](#)). The interrupt triggers the firmware to process new TS packets and generate new descriptors. If the firmware has generated the last descriptor (for the last TS packet) of the session, it writes to the Interrupt Mask register in the TS port to disable further TS port interrupts.

## 18.7

## TS Timestamp Mechanism

The average data rate on the TS output interface must be constant, i.e., the TS packet gaps on the TS input have to be rebuilt on the TS output.

At the beginning of the TS packet transfer on the TS input interface, a timestamp is sampled for each received packet on the TS port input interface. The sampled timestamp for each packet is an absolute time value, based on a free-running timer in the TS port. After the DMA transfer of the TS packet, the related timestamp is stored as part of the descriptor in the TS Done queue.

Synchronization on the TS output interface is also based on a timer scheme. For each packet an absolute time value is provided in the TS Descriptor queue. At the beginning of a Transport Stream session, the firmware pre-loads the timer with the absolute time value of the first packet (see TSU Timestamp Control Register ([Table 755 p. 757](#))), generates the first descriptor, and updates the TS Descriptor Write Pointer. This triggers the TS port to start the data DMA of the first packet. Then the firmware starts the timer (see TSU Timestamp Control Register), which is incremented with each clock edge. After the transfer of Packet x from the memory to the FIFO, the actual timer value is compared with the time value of Packet x+1. This value is pre-fetched with the descriptor x+1, during the data transfer of Packet x. If the actual timer value is equal to or greater than that of x+1, the TS port starts the data transfer of Packet x+1 and the pre-fetch of the descriptor x+2.

The TSU Timestamp mechanism is fully controllable by programming the [TSU Timestamp Control Register](#). The timer value is readable by reading the TSU Timestamp Register ([Table 753 p. 756](#)).

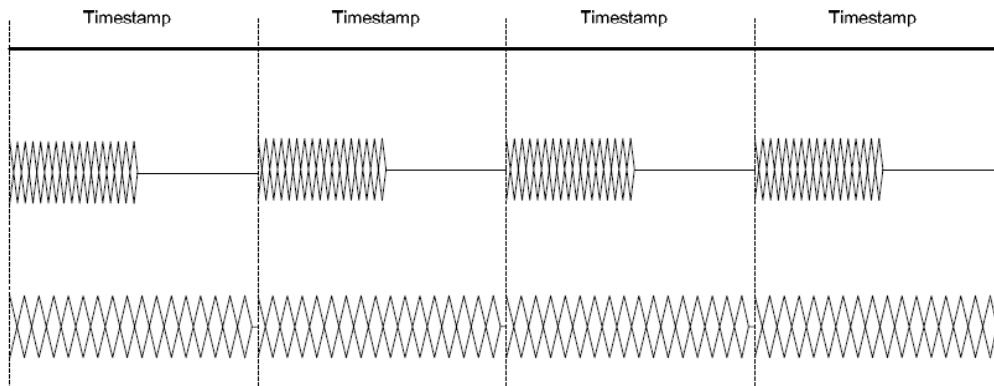
When the timestamp mechanism is disabled (see the TSU Timestamp Control Register), the TS packets are only transmitted on the TS output back to back, based on the TS output frequencies.

Every TS packet can receive a timestamp based on a software calculation. It is not only input stream dependent. This timestamp can define the TS output data rate. Furthermore, an existing timestamp can be changed via a software algorithm, to redefine the TS output data rate.

For example, packets come in bursts of 20 packets every 100us. Setting the timestamps of the packets to intervals of 5us configures the TS port to send them as a stream of one packet every 5us.

[Figure 75](#) illustrates a TS output data rate that is twice the input rate, but data is only transmitted within the timestamp range.

**Figure 75: Impact of Timestamp on the Average TS Data Output Data Rate**



## 18.8 TS Packet Aggregation

The TSU supports TS packet aggregation in both input and output directions. Aggregation is enabled by setting the [Aggregation on](#) field in the TSU Aggregation Control Register ([Table 764 p. 760](#)). Aggregation mode parameters are also fully controlled by this register.

### 18.8.1 TS Input Mode

The following actions are performed for hardware aggregation of the TS packets in TS Input mode:

#### Type 1 Aggregation

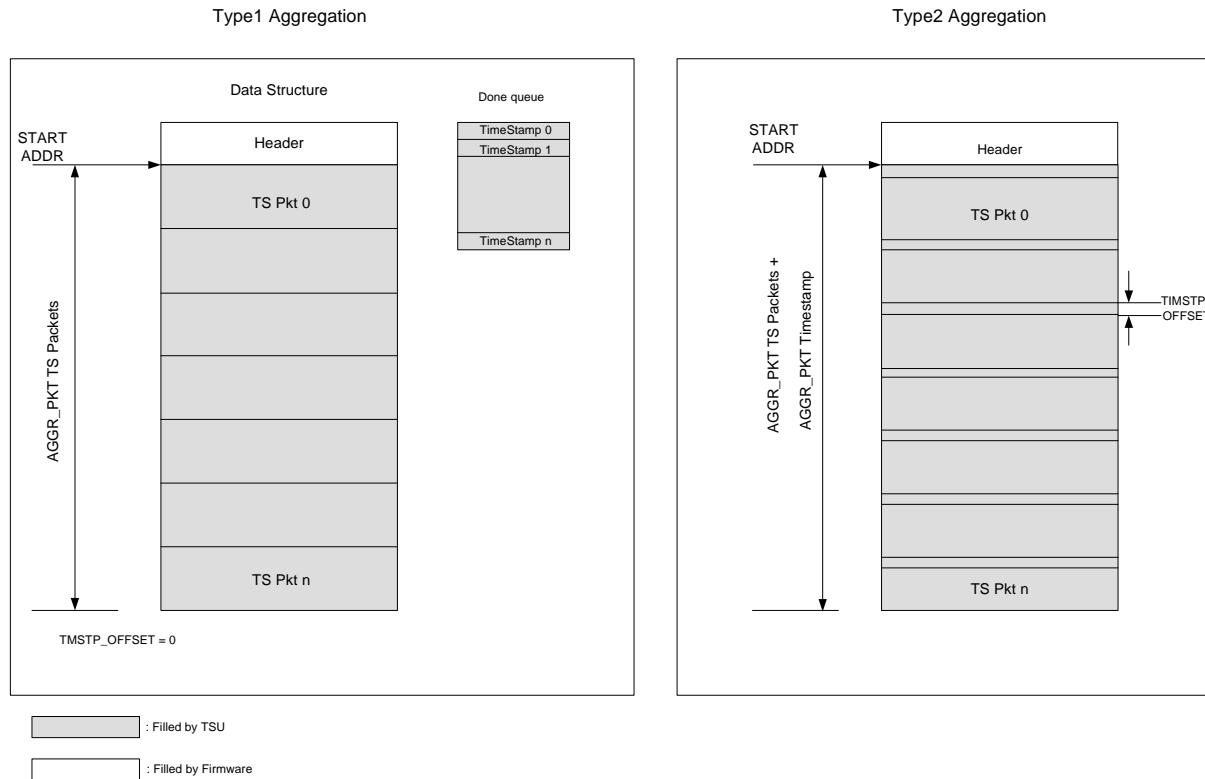
1. The TSU loads the Start Address provided by the CPU via the descriptor.  
**NOTE:** Type 1 Aggregation must only be used when the packet size is a multiple of 32 bytes. The start address must be aligned to 32 bytes.
2. The TSU writes the TS packets continuously into memory from the Start Address.
3. At the end of each TS packet, the TS port writes the timestamp (see the TSU Timestamp Register ([Table 753 p. 756](#))) into the Done queue and calculates the time interval. (Later, firmware will use only one of these timestamps.)
4. It iterates according to the number defined in the [Aggregated Packets](#) field and an interrupt is asserted after the last TS packet.

#### Type 2 Aggregation

1. The TSU loads the Start Address provided by the CPU via the descriptor.  
**NOTE:** Aggregation mode 2 must only be used when the packet size is *not* a multiple of 32 bytes. The start address and the Time stamp offset (the [Timestamp Offset](#) field in the TSU Aggregation Control Register ([Table 764 p. 761](#))) must be such that the sum of (start address + Time stamp offset) is aligned to 32 bytes.
2. The TS port writes each TS packet from the Start Address, but packets are separated by the [Timestamp Offset](#). The offset space remains empty at this time.
3. At the end of each TS packet, the TS port writes the timestamp into the offset space.
4. It iterates according to the number defined in the [Aggregated Packets](#) field, and an interrupt is asserted after the last TS packet.

Figure 76 depicts the aggregate TS Input mode.

**Figure 76: Aggregated TS Input Mode**



## 18.8.2 TS Output Mode

The following actions are performed for hardware aggregation of the TS packets in TS Output mode:

### Type 1 Aggregation

1. The TS port loads the Start Address provided by the CPU via the descriptor.  
**Note:** Type 1 Aggregation must only be used when the packet size is a multiple of 32 bytes. The start of packet, i.e., the sum of (start address+ Timestamp interval size + initial timestamp size) must be aligned to 32 bytes.
2. The TS port loads the timestamp interval and the initial timestamp. It uses the initial timestamp for the first TS packet and the interval value for remaining packets.
3. This procedure occurs <Aggregated Packets> times.
4. After the last Status has been written to the Done queue, an interrupt is asserted.

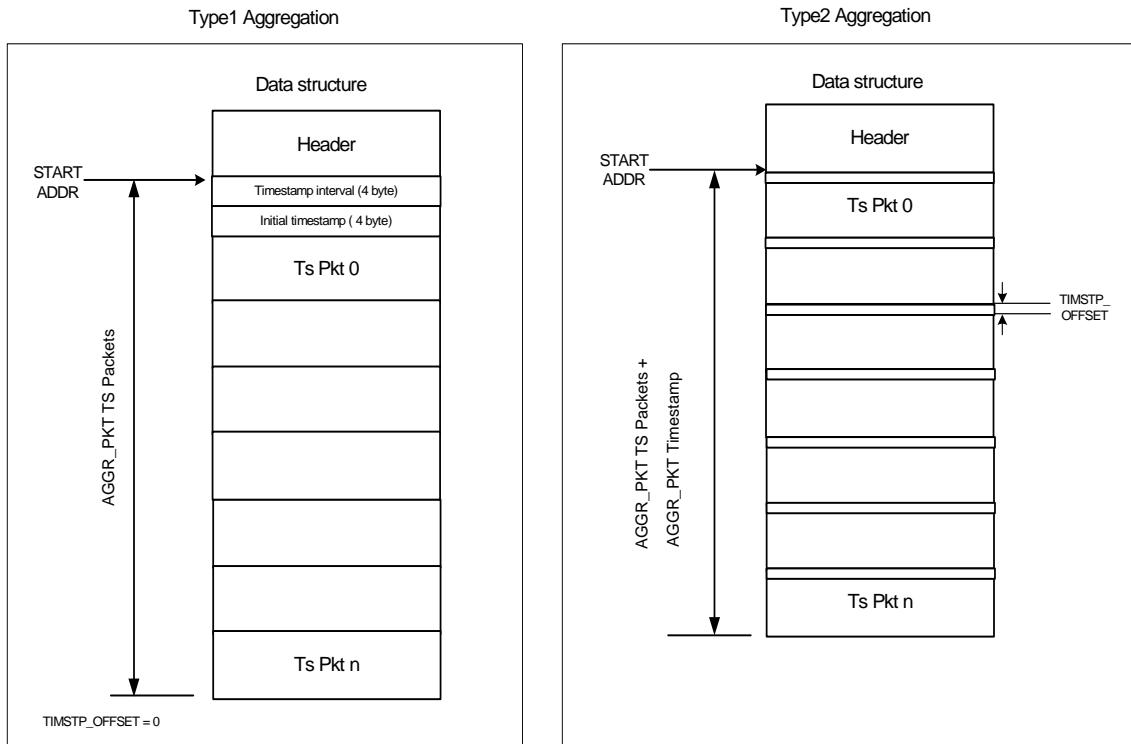
### Type 2 Aggregation

1. The TS port loads the Start Address provided by the CPU via the descriptor.  
**Note:** Aggregation mode 2 must only be used when the packet size is *not* a multiple of 32 bytes. The start address and the Time stamp offset (field <Timestamp Offset>) must be such that the sum of (start address + Time stamp offset) is aligned to 32 bytes.
2. The TS port reads the first timestamp with the first TS packet, then the second timestamp and second TS packet, etc.

3. This procedure occurs <Aggregated Packets> times.
  4. After the last Status has been written to the Done queue, an interrupt is asserted.
- Figure 77 depicts the aggregate TS Input mode.

**Figure 77: Aggregated TS Output Mode**

---



## 18.9 TS Port Interrupts

The TS port supports different interrupt events, which are registered and controlled in the following registers:

- TS port interrupts are registered in the TSU Interrupt Source Register ([Table 757 p. 758](#)). Upon an interrupt event, the corresponding cause bit is set to 1. It is cleared upon software read.
- TSU interrupts are masked in the TSU Interrupt Mask Register ([Table 758 p. 758](#)).

The TS port supports the following interrupt events:

- Interrupt Descriptor Threshold: Asserted when the difference between the Descriptor Read Pointer and Descriptor Write Pointer is that defined in the <Descriptor IRQ Threshold> field in the IRQ Parameter Register ([Table 759 p. 759](#)).
- Interrupt Done Put Descriptor: Asserted after the descriptor has been inserted in the TS Done queue.
- Interrupt Done Aggregated Packet: Asserted after the last Status/Timestamp of the aggregated packet has been put into the TS Done queue.
- Interrupt on TS Interface Error: Asserted if a TS\_ERR signal has been asserted for a packet.

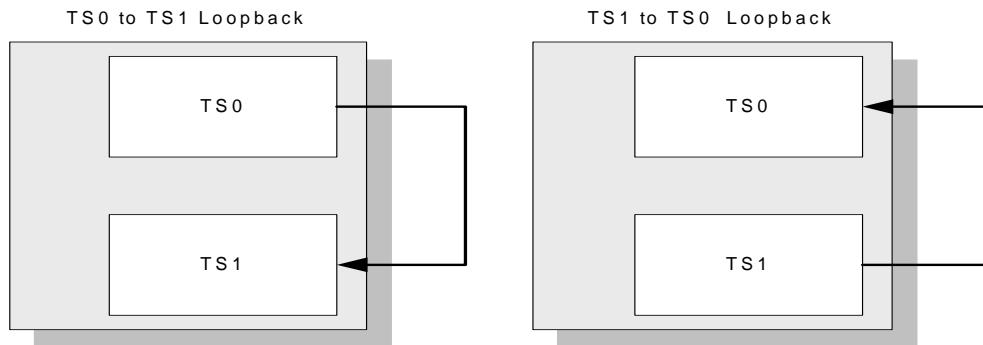
- Interrupt on FIFO Overflow Error: Asserted if a FIFO Overflow error has occurred on a packet.
- Interrupt on TS connection Error: Asserted if a TS Connection Error has occurred on a packet.
- Interrupt on CLK Sync Timer: Asserted after the Clock Synchronization timer expires.

## 18.10

### Loopback Mode

The device supports Loopback mode between the two integrated TS ports. Loopback is supported in both parallel and serial modes. Both TS0-to-TS1 and TS1-to-TS0 loopback directions are supported. The loopback operation is enabled by setting the [<Enable Loop TS1 to TS0>](#) or [<Enable Loop TS0 to TS1>](#) field in the TSU Test Register ([Table 756 p. 757](#)) of each TS port.

**Figure 78: TS Loopback Modes**



# 19

# General-Purpose I/O (GPIO) Port Interface

The device contains a General-Purpose Input/Output (GPIO) port that varies in size for each device:

- 88F6180: 30-bit GPIO port
- 88F619x: 36-bit GPIO port
- 88F6281: 50-bit GPIO port

Some of the GPIO pins can be assigned to act as a general-purpose input or output pins and can be used to register external interrupts (when assigned as input pins).

Some of the GPIO pins are used for reset sampling. These pins cannot be used as input GPIO pins, but are output GPIO pins only. For details, see the MPP Functional Summary table in the *Hardware Specifications* for each device.

## 19.1

## GPIO Control Registers

Depending on the setting of the GPIO Data In Polarity Register ([Table 770 p. 763](#)), the GPIO Data In Register ([Table 771 p. 763](#)) samples either the data sampled on the pins or the inverted data.

Each GPIO can act also as an output. Setting the GPIO High Data Out Enable Control Register ([Table 776 p. 764](#)), enables driving data on the corresponding GPIO output pin. The data driven on the pin is the data configured into GPIO Data Out Register ([Table 767 p. 762](#)).

The combination of the [GPIO Data Out Register](#) and [GPIO Data Out Enable Control Register](#), also allows open drain/open source output implementation. For example, to implement an open drain output, set the `<GPIODataOut>` field in the GPIO Data Out Register ([Table 767 p. 762](#)) to 0, and toggle the output by setting/clearing the corresponding field in the [GPIO Data Out Enable Control Register](#).

## 19.2

## GPIO Blink Enable Register

When GPIO Blink Enable Register ([Table 769 p. 763](#)) is set and the corresponding bit in GPIO Data Out Enable Control Register ([Table 768 p. 762](#)) is enabled, the GPIO pin blinks every  $2^{24}$  TCLK clocks.

## 19.3

## GPIO Interrupts

When configured to act as GPIO inputs, the GPIOs can be used for registration of external devices interrupts into the CPU interrupt controller. The device supports both edge and level sensitive external interrupts.

- When the external device uses edge sensitive interrupts, each toggle of `<GPIODataIn>` field in the GPIO Data In Register ([Table 771 p. 763](#)) from 0 to 1 is registered in the GPIO Interrupt Cause Register ([Table 772 p. 763](#)). If the interrupt is not masked by the GPIO Interrupt Mask Register ([Table 773 p. 764](#)), the interrupt is routed to the CPU Main Interrupt Cause register.  
To clear an edge sensitive interrupt, the software must clear the corresponding bit in the GPIO Edge Sensitive Interrupt Cause register.
- When the external device uses level sensitive interrupts, the data is registered in the [GPIO Data In Polarity Register](#). If not masked by the GPIO High Interrupt Level Mask Register ([Table 782 p. 766](#)), the interrupt is routed to the CPU Main Interrupt Cause register.  
To clear a level sensitive interrupt, the software clears the interrupt directly on the external device.

# 20

## Real-Time Clock (RTC) Unit

The device integrates a real-time clock (RTC)/calendar that provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap years up to the end-of-year 2099. The clock operates in either the 24-hour or 12-hour format with an AM/PM indicator. The RTC also provides an alarm function.

The RTC unit operates with an external 32.768 kHz crystal, a dedicated power supply of 1.5V–1.8V, which can be supplied by a battery when the device power is down.

### 20.1

#### Features

RTC features include:

- Real-time fields: second, minute, hour, date, day, month, and year
- Leap-year compensation
- Independent power pin (RTC\_AVDD) for power by battery
- Total current consumption for typical case: 3 uA
- Alarm-time fields: second, minute, hour, date, month, and year
- Real-time field setting by CPU
- Valid until the end of the century (i.e., 23:59:59, December 31, 2099)

### 20.2

#### Functionality

To set the RTC date and time, write the desired time and date to the respective time and date registers:

- RTC Time Register ([Table 784 p. 767](#))
- RTC Date Register ([Table 785 p. 768](#))

The values written to the fields in these registers are in BCD (Binary-Coded Decimal) format.

RTC date and time are known by reading the [RTC Time Register](#) and [RTC Date Register](#).

#### 20.2.1

##### Setting the Time

To configure the day of the week and the time:

1. Set the [`<WeekDay>`](#) field in the RTC Time Register ([Table 784 p. 768](#)) field to indicate the day of the week.
2. Set the [`<HourFormat>`](#) to indicate if the time value is a 24-hour or a 12-hour presentation.
  - When this field is set to a 12-hour representation, set bit [21] of the [`<Hour>`](#) field to indicate whether the time is AM or PM, and set bits [20:16] to set the hour.
  - When this field is set to a 24-hour representation, set the [`<Hour>`](#) field to indicate the hour.
3. Set the [`<Minute>`](#) field to indicate the minutes within the hour.
4. Set the [`<Second>`](#) field to indicate the seconds within the minutes.

All the above fields are located in the same register, and can be set using a single write transaction.

## 20.2.2 Setting the Date

To set the date:

1. Set the [\*\*<Day>\*\*](#) field in the RTC Date Register ([Table 785 p. 768](#)) to indicate the day of the month.
2. Set the [\*\*<Month>\*\*](#) field to indicate the month.
3. Set the [\*\*<Year>\*\*](#) field to indicate the year.

All the above fields are located in the same register, and can be set using a single write transaction.

## 20.2.3 Setting Both the Time and Date

When setting both the time and the date, Marvell® recommends setting the [\*\*RTC Time Register\*\*](#) prior to setting the [\*\*RTC Date Register\*\*](#). This avoids the occurrence of a new date increment due to an existing (invalid) time, which may occur if the recommended order is not followed. For example, if the updated date is set first, while the current time is set to 23:59:59, when the time is updated (1 second or more later), the date advances again when the clock reaches 0:0:0, resulting in a date advance of one additional day.

## 20.2.4 RTC Alarm Operation

The RTC alarm is not powered by the battery power pin. It can be used when the device power is on (see [Section 22.3, RTC Alarm, on page 283](#)).

# 21 Interrupt Controller

The device includes an interrupt controller that routes internal interrupt requests and external interrupt requests (GPIOs) to the CPU.

The device interrupt controller drives two interrupt signals to the CPU—FIQ (high priority) and IRQ (regular priority). All interrupts are level-sensitive. The interrupt is kept active as long as there is at least one non-masked cause bit set in the Interrupt Cause registers.

The device can also be used as the interrupt controller for external devices generating interrupts to the CPU via GPIO inputs. The interrupt controller can also receive interrupt messages from an external PCI Express device.

In addition, the device can act as a PCI Express Endpoint. As such, it can generate the PCI Express INTA emulation message or the INTAn signal.

The device handles interrupts in two stages.

- In the first stage, the specific unit cause register that distinguish between specific interrupt events within the unit is set, if it is not masked in the unit mask register (see [Section 21.1, Local Interrupt Cause and Mask Registers](#) ).
- The second stage includes the main interrupt mask registers that summarize the interrupts generated by each unit (see [Section 21.2, Main Interrupt Cause and Mask Registers](#) ). The interrupt handler:
  - First reads the Main Interrupt Cause Low Register ([Table 151 p. 383](#)).
  - If <MainHighSum> field in the Main Interrupt Cause Low Register ([Table 151 p. 383](#)) is set, it also reads the Main Interrupt Cause High Register ([Table 155 p. 386](#)).
  - Then reads the specific unit cause register.

## 21.1 Local Interrupt Cause and Mask Registers

Once an interrupt event occurs, its corresponding bit in the unit cause register is set to 1. If the interrupt is not masked by the unit mask register, it is asserted in the Main Interrupt Cause Low Register ([Table 151 p. 383](#)) or Main Interrupt Cause High Register ([Table 155 p. 386](#)). The unit local mask register has no effect on the setting of interrupt bits in the unit local cause register. It only affects the setting of the interrupt bit in the [Main Interrupt Cause Low Register](#) or [Main Interrupt Cause High Register](#).

When working in level mode, the GPIO Data In Register ([Table 771 p. 763](#)) must be used. Do not use the GPIO Interrupt Cause Register ([Table 772 p. 763](#)).

The different units cause registers are:

- Mbus-L to Mbus Bridge Interrupt Cause Register
- TDM Interrupt Cause Register (88F6192 and 88F6281 only)
- PCI Express Port Interrupt Cause Registers
- SATAHC Main Interrupt Cause register and SATAHC Interrupt Cause Register (88F619x and 88F6281 only)
- GbE Port 0/1 Interrupt Cause Register (Port 1 is implemented only in the 88F6192 and 88F6281)
- USB 2.0 Port Interrupt Cause Register
- Cryptographic Engine Security Accelerator Cause Register
- XOR Engine Interrupt Cause Register

- TWSI Interrupt Cause Register
- UART0/1 Interrupt Identity (IIR) Register
- SPI Cause Register
- Audio Interrupt Cause Register (88F6180, 88F6192 and 88F6281 only)
- SDIO Error Interrupt Status Register
- MPEG-TS 0/1 Interrupt Cause Register (88F6192 and 88F6281 only)
- GPIO Interrupt Cause Register

## 21.2 Main Interrupt Cause and Mask Registers

---



Note

The Main Interrupt Cause register bits are Read Only. To clear an interrupt cause, the software needs to clear the active bit(s) in the specific unit cause register.

There are two mask register sets corresponding to the two CPU interrupt lines—IRQ and FIQ. Setting these registers allows the reporting of different interrupt events on the different interrupt lines. If a bit in the mask register is set to 1, the corresponding interrupt event is enabled. The setting of the mask bits has no effect on the value registered in the [Main Interrupt Cause Low Register](#) or [Main Interrupt Cause High Register](#), it only affects the assertion of the interrupt line. An interrupt is asserted if at least one of the non-masked bits in the cause register is set to 1.

When the device functions as Endpoint, a third mask register corresponding to PCI Express interrupt is used to generate an interrupt towards the host. The INTA interrupt or MSI is routed to host according to this bit value.



Note

- See the [Appendix A.3.6, Main Interrupt Controller Registers, on page 383](#).
- See [Section 21.4, Device Interrupt Controller Scheme, on page 282](#).

## 21.3 Doorbell Interrupt

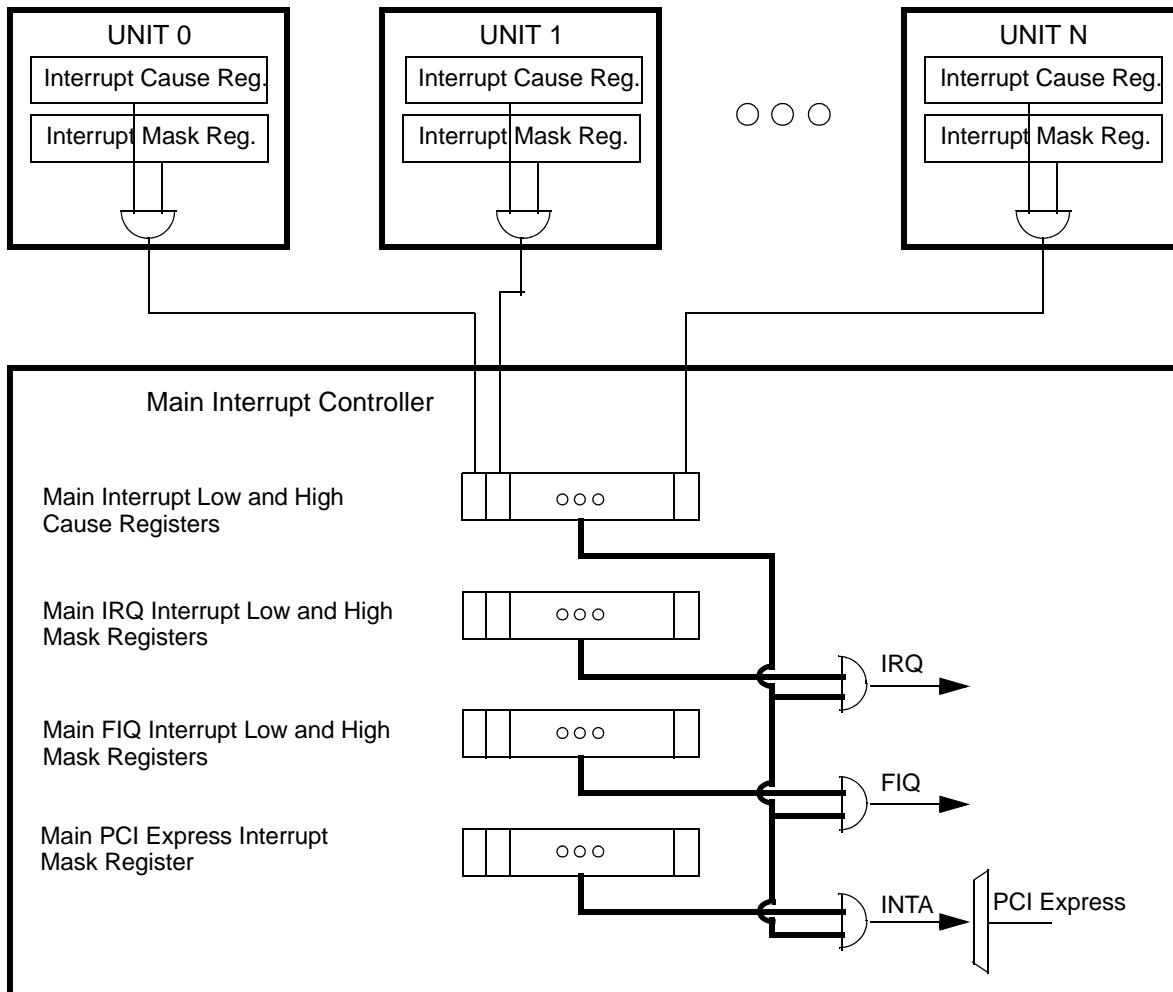
When device functions as an Endpoint device, a doorbell mechanism is provided to communicate between CPU and the external host.

The device supports a 32-bit doorbell interrupt register from the host to the CPU. See the Host-to-CPU Doorbell Register ([Table 132 p. 377](#)) and the Host-to-CPU Doorbell Mask Register ([Table 133 p. 378](#)).

## 21.4 Device Interrupt Controller Scheme

Figure 79 depicts the interaction between the individual unit interrupt and mask registers and the Main Interrupt controller.

**Figure 79: Device Interrupt Controller Scheme**



# 22 Timers and Counters

The device provides two general-purpose timers and one watchdog timer.

## 22.1 32-bit General-Purpose Timers

The device provides two 32-bit general-purpose timers. Each timer decrements with every TCLK rising edge, if the corresponding enable bit is enabled. Reads and writes from/to the timer are performed to the counter itself.

The timers provide Auto mode:

- When Auto mode is disabled, and the timer reaches 0, the timer stops counting.
- When Auto mode is enabled, and the timer reaches 0, the timer reloads and continues counting.

Regardless of whether Auto mode is enabled or disabled, when the timers reach 0, a maskable interrupt is generated.



See Appendix A.3.4, CPU Timers Registers, on page 378.

Note

## 22.2 Watchdog Timer

The device internal watchdog timer is a 32-bit count down counter that can be used to generate a maskable interrupt or reset the system in the event of unpredictable software behavior.

After the watchdog is enabled, it is a free running counter that needs to be serviced periodically to prevent its expiration.

When the watchdog timer expires and the <WDRstOutEn> field in the RSTOUTn Mask Register (Table 117 p. 368) is set to 1, the SYSRST\_OUTn output signal is set.



See Appendix A.3.4, CPU Timers Registers, on page 378.

Note

## 22.3 RTC Alarm

This section describes the RTC alarm function.

To set the RTC Alarm date and time, write the desired alarm time and date and alarm valid data to the respective alarm time and date registers:

- RTC Alarm Time Configuration Register (Table 786 p. 768)
- RTC Alarm Date Configuration Register (Table 787 p. 769)

The values written to the fields in these registers are in BCD (Binary-Coded Decimal) format.

RTC alarm date and time are known by reading the [RTC Alarm Time Configuration Register](#) and [RTC Alarm Date Configuration Register](#).

These two registers contain the six alarm fields Real-time Clock (RTC), corresponding to the six data and time fields (day in week is excluded). For each such field, a valid bit specifies whether an alarm event does or does not include the matching between the alarm field and the corresponding real-time field.

The match is tested on a once-per-second update of the time and date registers.

Upon a tested match the [`<AlarmInterrupt>`](#) field in the RTC Interrupt Cause Register ([Table 789 p. 770](#)) is set resulting in an RTC Alarm interrupt.

To clarify the Alarm Interrupt valid bit usage, [Table 75](#) provides a number of examples of configurations and the corresponding alarm behavior.

**Table 75: Alarm Interrupt Valid Bit Usage**

RTC Alarm Date Configuration Register			RTC Alarm Time Configuration Register			Alarm Rate
Alarm Year Valid	Alarm Month Valid	Alarm Day Valid	Alarm Hour Valid	Alarm Minute Valid	Alarm Second Valid	
0	0	0	0	0	0	Alarm once per second.
0	0	0	0	0	1	Alarm once per minute, when seconds match.
0	0	0	0	1	1	Alarm once per hour, when minutes and seconds match.
0	0	0	1	1	1	Alarm once per day, when hour, minute, and seconds match.
1	1	1	0	0	0	Alarm once per second, in a specific date.
1	1	1	1	0	0	Alarm every second in a specified date and hour.
1	1	1	1	1	0	Alarm every second in a specified date, hour, and minute.
1	1	1	1	1	1	Alarm once, when all fields match.

The alarm registers are initialized at reset, with all Alarm Valid bits being in *not set* state. As shown in the first row of [Table 75](#), this mode will set the alarm upon every second. Without setting the [RTC Alarm Time Configuration Register](#) and [RTC Alarm Date Configuration Register](#), the user should not expect a deterministic value at the [`<AlarmInterrupt>`](#) field, since its value depends on the reset de-assertion and the register read of a real-time-second change.

To set the alarm registers, the following sequence must be applied:

1. Disable interrupts by writing 0 to the [`<AlarmInterruptEnable>`](#) field in the RTC Interrupt Mask Register ([Table 788 p. 770](#)).
2. Set the fields in the [RTC Alarm Time Configuration Register](#) and [RTC Alarm Date Configuration Register](#).
3. Clear the [`<AlarmInterrupt>`](#) field.
4. Enable the interrupt by writing 1 to [`<AlarmInterruptEnable>`](#) field.

## 22.4

## SYSRST<sub>n</sub> Duration Counter

The devices implement a hardware-based SYSRST<sub>n</sub> duration counter. When SYSRST<sub>n</sub> is asserted low, a SYSRST<sub>n</sub> duration counter is running. The SYSRST<sub>n</sub> duration counter is useful for implementing a manufacturer/factory reset. Upon a long reset assertion that is greater than a pre-configured threshold, the host software may reset all settings to the manufacturer/factory default values. The counter value is stored in the SYSRST<sub>n</sub> Length Counter Register ([Table 811 p. 783](#)).

- The counter is based on the 25-MHz reference clock (40ns)
- It is a 29-bit counter, yielding a maximum counting duration of  $2^{29}/25$  MHz (21.4 seconds).

The host software can read the counter value through the [`<Count>`](#) field and reset the counter through the [`<Clr>`](#) field.

When the counter reaches its maximum value, it remains at this value until counter reset is triggered by software.

# 23 eFuse

The device integrates two eFuse blocks: eFuse0 and eFuse1. Each eFuse block has 64 programmable data bits that provide a total of 128 bits ( $2 * 64$ ). The eFuse is a one-time electrical programmable element, where the data bit values can be programmed.

The device supports:

- eFuse programming—updating the data bits in the eFuse
- eFuse locking—disabling the programming operation
- eFuse reading—reading the data bits and the lock bit.

The default value of the eFuse data bits and the lock bit is 0x0.

## 23.1 Typical eFuse Applications

- Specific private/public key for security and prevention of hacking.
- Operations information, such as system serial number/production number.
- Time of the initial operation of system by the end user. (For example, software may burn date/time after the initial power up.)
- Software upgrade limiter. (For example, only three upgrades are allowed by the system vendor, where each upgrade burns specific eFuse bits).
- Device unique name/ID (for example, MAC address).

## 23.2 eFuse Power Supply

When programming the eFuse, a 2.5V power supply is required for the VHV power pin.

When reading from the eFuse, a 1.0V power supply is required for the VHV power pin.

For more details, see the Electrical Specifications section in the *Hardware Specifications*.

## 23.3 eFuse Program and Lock

eFuse0 and eFuse1 can be programmed and /or locked by the CPU. Each data bit can be set to 1, one time only. After programming a bit in the eFuse to 1, it can no longer be changed to 0. When a data bit has a value of 0, and it is programmed to 0, it is as if it has not been programmed, since it remains unchanged. In such a situation, the bit can still be programmed to the value 1.



**Note** Programming and/or locking the eFuse without using the required power supply results in unpredictable data in the eFuse.

Each of the eFuse0 and eFuse1 blocks can be locked independently. When a eFuse block is locked, the eFuse programming operation is disabled.

The eFuse lock operation can be performed one time only. After locking the eFuse, it cannot be unlocked.



Note

Refer to the device *Hardware Specifications* for information regarding the VHV power pins and burn and read voltage requirements.

Follow the below sequence to program and/or lock the eFuse:

1. Set the [\*\*<Burn Mode>\*\*](#) field in the eFuse Control Register ([Table 807 p. 781](#)).
  2. Set the required data bits (the bits that should be burned to 1) by writing to the [\*\*<eFuse0 Low>\*\*](#) field in the eFuse0 Low Register ([Table 803 p. 780](#)) and the [\*\*<eFuse0 High>\*\*](#) field in the eFuse0 High Register ([Table 804 p. 780](#)).
  3. To lock the eFuse, set the security bit by writing to [\*\*<FSB0>\*\*](#) field in the eFuse Protection Register ([Table 802 p. 779](#)).
- NOTE:** Where further programming to the eFuse block is required, do not lock the eFuse.
4. Set the [\*\*<eFuse0 Write Trigger>\*\*](#) field in the eFuse Control Register ([Table 807 p. 781](#)) to trigger the burning of eFuse0.
  5. Poll the [\*\*<eFuse Burn Done>\*\*](#) field until it set, for an indication that the burn process completed.
  6. To program eFuse1, repeat steps 2–5, using the fields related to eFuse1 instead of eFuse0.
  7. Clear the [\*\*<Burn Mode>\*\*](#) field.



Note

- The sequence above demonstrates a write to both eFuses. The user can optionally write to only one of the eFuses. For example, to program eFuse0 only, skip step 6, in the programming sequence.
- eFuse0 and eFuse1 must be programmed separately. To prevent simultaneously programming of both eFuse0 and eFuse1, do not set eFuse0 Write Trigger and eFuse1 Write Trigger simultaneously.

## 23.4 eFuse Read

For eFuse value reading, read:

- eFuse0 Low Register ([Table 803 p. 780](#))
- eFuse0 High Register ([Table 804 p. 780](#))
- eFuse1 Low Register ([Table 805 p. 780](#))
- eFuse1 High Register ([Table 806 p. 781](#))

# 24 System Considerations

This section describes the Endianess, bootROM, power management, and error handling for the device.

## 24.1 Big and Little Endian Support

The device supports both Big Endian and Little Endian byte ordering, as defined in the *ARM Architecture Reference Manual*, Second Edition. It also supports hardware features for performing data conversion on some of its interfaces.

### 24.1.1 CPU Core Byte Ordering

The Endian mode is set by <B> bit [7] in register r1, within the CPU-CP15 registers. The value of <B> bit is reflected by the [<BigEndian>](#) field in the CPU Control and Status Register ([Table 116 p. 368](#)).

The initial value of <B> bit is set by the [<EndianInit>](#) field in the CPU Configuration Register ([Table 115 p. 366](#)). The initial value of bit [<EndianInit>](#) is defined as Little Endian mode (0).

The CPU performs the proper data swapping, according to this setting. For full details, see *AN-183, 88F5181 and 88F5281 Big Endian and Little Endian Support*.

**Note**

Regardless of the endianess mode, the device internal registers and DMAs descriptors always operate in Little Endian mode. When working in Big Endian mode, use a software macro to perform byte swapping when an internal register or a descriptor is written, or read.

### 24.1.2 PCI Express Space

The PCI Express specification defines that a TLP data payload be presented as Big Endian, and the packet header (address, command) be presented as Little Endian. The PCI Express interface meets these requirements. Depending on the chip endianess configuration, the PCI Express interface either performs or does not perform byte swapping.

The least significant byte is always the one to be transmitted first. For example, if a CPU is configured to Big Endian mode and there is a CPU write of 4 bytes to address 0x0 (which appears on the CPU bus as 64'hAABBCCDD.XXXXXXXXXX), the device drives 0xAA as the first byte on the PCI Express link.

**Note**

The PCI Express interface does not support different byte swapping on a per master and slave operation, nor on a per address-window basis.

## 24.1.3 DMA Data Swapping

The device DMAs (XOR DMA, GbE SDMA, USB DMA, SATAHC DMA, and encryption DMA) support the required mechanisms for proper data transfer both in Big Endian and Little Endian environments.

### 24.1.3.1 XOR DMA Data Swapping

By default, the XOR DMA does not need to perform any data byte swapping when transferring data buffers over the Mbus. However, the XOR DMA supports byte swapping on a 64-bit Qword basis upon a read from source buffers or a write to a destination buffer via the [`<DrdResSwp>`](#) field and the [`<DwrReqSwp>`](#) field in the XOR Engine [0..1] Configuration (XExCR) Register (n=0–1) ([Table 588 p. 664](#)). This can be useful for endianess conversion.

The XOR DMA descriptor is fetched from memory as either a 32-byte or 64-byte burst, depending on the XOR mode of operation. The descriptor is loaded into the XOR DMA register file, which maintains the Little Endian convention. Thus, the descriptor is expected to be prepared in memory accordingly.

If the [`<DesSwp>`](#) field in the XOR Engine [0..1] Configuration (XExCR) Register (n=0–1) ([Table 588 p. 665](#)) is set to 1, the DMA performs a byte swap on a 64-bit Qword basis when fetching (and closing) descriptors.

### 24.1.3.2 GbE SDMA Data Swapping

When transferring data between memory and MAC, the GbE SDMA can swap the bytes on 64-bit Qword basis. Set the [`<BLMR>`](#) field in the SDMA Configuration (SDC) Register ([Table 422 p. 564](#)) to 0, for byte swap of Rx buffers when written to memory, and set field [`<BLMT>`](#) to 0, for byte swap of Tx buffers when being read from memory.

When reading Tx data from memory, the GbE SDMA always first transmits byte[7:0], followed by byte[15:8], and so on. In Big Endian convention, byte[63:56] stands for the least significant byte. This means that it is expected to be transmitted first. To achieve this behavior, set [`<BLMR>`](#) and [`<BLMT>`](#) fields to 0.

The GbE SDMA descriptor is being fetched from memory as a burst of 16 Bytes. The descriptor is loaded into the SDMA register file, which maintains Little Endian convention. Thus, the descriptor is expected to be prepared in memory accordingly.

If the [`<SwapMode>`](#) is set to 1, the SDMA performs byte swap on a 64-bit qword basis when fetching (and closing) descriptors.

### 24.1.3.3 USB Data Swapping

The device supports byte swap (within 8 byte qword) upon read/write access to memory. To enable byte swap, clear the [`<BS>`](#) field in the USB 2.0 Bridge Control Register ([Table 509 p. 629](#)).



The data swapping logic cannot distinguish between descriptors to raw data. It is a static setting.

**Note**

### 24.1.3.4 SATAHC Data Swapping

The device SATA controller supports the following byte swap mechanisms, via the SATAHC Configuration Register ([Table 348 p. 509](#)):

- Byte swap (within a 8 byte Qword) upon Basic DMA read/write access to memory. Clear the [`<DmaBS>`](#) to enable byte swapping.

- Byte swap (within a 8 byte Qword) upon EDMA read/write access to memory. Clear the [`<EDmaBS>`](#) to enable byte swapping.
- Byte swap (within a 8 byte Qword) upon PRDP read/write access to memory. Clear the [`<PrdpBS>`](#) to enable byte swapping.

**Note**

The data swapping logic cannot distinguish between descriptors to raw data. It is a static setting.

#### 24.1.3.5

### Cryptographic Engine Data Swapping

The device Cryptographic Engine and Security Accelerator (CESA) supports the following byte swap mechanisms:

- Byte swap and word swap (within an 8-byte Qword) upon read/write access to the CESA integrated SRAM. Set the Target Attributes field in the CPU address decoding registers to achieve the required data swapping (see [Section 2.1, Sheeva™ CPU Core Address Decoding, on page 34](#)).
- Byte swap (within an 8-byte Qword) upon a TDMA read/write access to memory. Clear the [`<BS>`](#) field in the Control Register ([Table 573 p. 654](#)) to enable byte swapping.

**Note**

The data swapping logic cannot distinguish between descriptor and packet data. It is a static setting.

## 24.2

### BootROM Firmware

#### 24.2.1

### Functional Description

The bootROM firmware—on the device's bootROM—is executed according to the sample at reset configuration bits in the Sample at Reset Register ([Table 800 p. 778](#)).

The bootROM firmware performs basic initialization of the device and loads and executes code that is on one of the following boot devices:

- Serial (SPI) flash
- NAND flash
- SATA interface (88F619x and 88F6281 only)
- PCI Express interface
- UART interface (using the Xmodem protocol)

**Note**

Boot from the UART interface is not a configurable reset strap option, but it is performed before every boot process, of the other boot types.

The BootROM firmware senses the UART0 interface (Rx side) looking for a unique stream of data. Detecting the data pattern identifies the multiplexed pins (MPP) assigned for the UART0 interface. This boot option is useful during system debugging and manufacturing.

## 24.2.2

### General Considerations

BootROM firmware code was written and compiled to take into account the following:

**Endianess**

BootROM firmware always executes in Little Endian mode. However there are no restrictions on the Endianess of the image booted from the device's interfaces. If the image was compiled to Big Endian mode, it is the responsibility of the image to switch back to Big Endian mode.

**ARM/Thumb Mode**

Most of the bootROM firmware code is compiled to run in Thumb mode, due to code size considerations. However, bootROM firmware switches back to ARM mode before booting an image from one of the device's interfaces.

**BootROM Firmware Image Format** The bootROM firmware image size is 12 KB, with the last 4 bytes consisting of a CRC-32 field.

## 24.2.3

### Address Decoding and Memory Management Unit (MMU) Operations

For performance enhancement purposes, the bootROM performs the following:

- Enables the L2 Cache
- Enables the Instruction Cache
- Enables the MMU (using a reduced translation table resident in the Cryptographic engine SRAM)
- Enables the Data Cache

Since the MMU is enabled, access to the 4 GB memory space is done using virtual addresses.

[Table 76](#) provides the virtual-to-physical address translation table.

**Table 76: MMU Virtual-to-Physical Address Translation Table**

Device	Physical Address	Virtual Address	Size	Caching
BootROM	0xFFFF00000	0xFFFF00000	1 MB	Cacheable
NAND registers	0xE80000000	0xFFE000000	1 MB	Non-Cacheable
Internal register	0xD00000000	0xFFD000000	1 MB	Non-Cacheable
SPI	0xD80000000	0xE80000000	125 MB	Non-Cacheable
SDRAM	0x0	0x0	384 MB	Non-Cacheable



**Note**

Since the MMU translation table covers 512 MB out of the 4 GB address space, make certain to consider the following points:

- The image can be copied up to offset 0x1FFFFFFF (384 MB) in the DDR.
- The SPI space is 125 MB instead of 128 MB.
- The extended header should include register configurations in the virtual space. For example, if internal register 0x1480 should be set to 0x1, then the extension header should include an Address/Value couple in the form 0xFFD01480/0x00000001.

## 24.2.4

### Boot Image Format

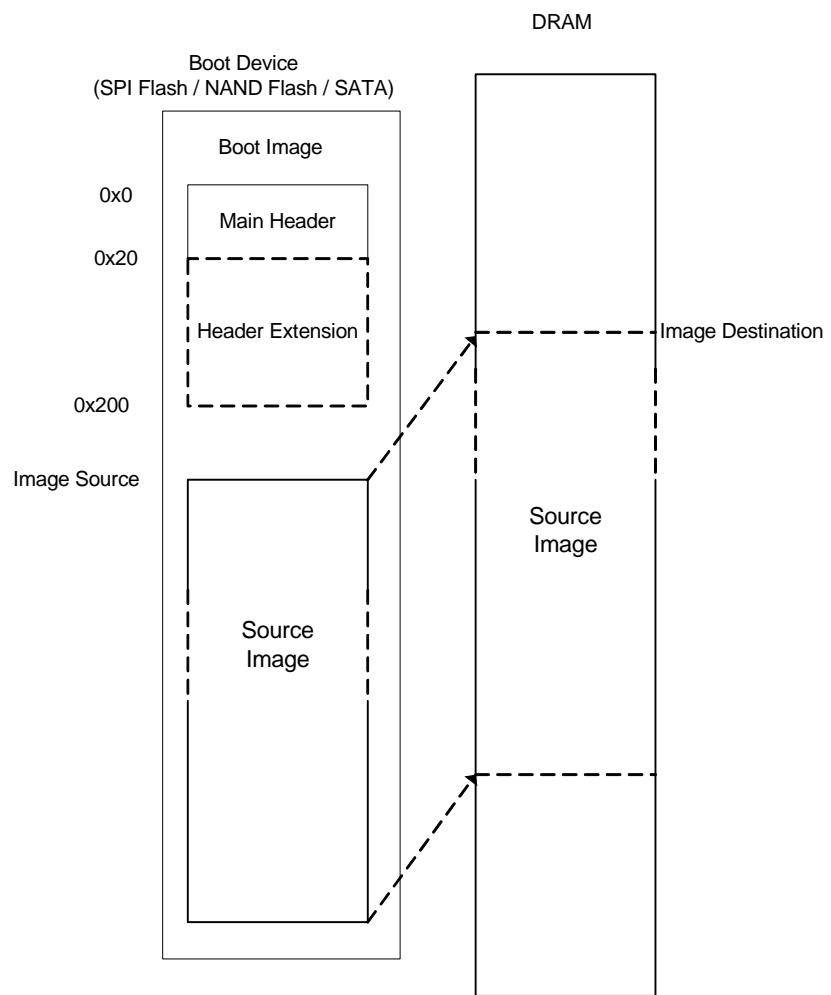
The boot image is the binary image residing on the specific boot device. It must contain a valid main image header at offset 0x0 and may include a header extension, according to the header extension flag in the main header. In addition, it includes the binary image that should be copied and executed

in the SDRAM (excluding boot from SPI when the image is executed directly from the SPI memory space).

In most cases, the header extension is used to provide the bootROM firmware with the DDR configurations. It must be concatenated to the main header, and the total size of the main header and the header extension must be exactly 512 bytes. The header extension may be omitted if the boot is to be performed directly from the boot device, without copying the user image to the DDR (valid for SPI boot device only).

The source image can immediately follow the main header or header extension, or it can reside at a more distant offset (this option is necessary in boot from SATA) (see [Figure 80](#)).

**Figure 80: Binary Image Layout in the Boot Device**



#### 24.2.4.1 Main Header Format

The main header is 32 bytes long and is in Little Endian mode. Its content differs according to the desired boot method and the device where the header is located.

**Table 77: Main Header Format**

Byte	Field	Description / Usage
0x0	Identifier	0x5A = Boot from Serial (SPI) flash 0x8B = Boot from NAND flash 0x78 = Boot from SATA device (88F619x and 88F6281 only) 0x9C = Boot from PEX interface 0x69 = Boot from UART0
0x1	NAND ECC mode	Relevant to boot from NAND flash only. This field can be used to change/disable the default ECC algorithm, used in reading the image from the NAND flash device. The following are the four options setting this field: 0x0 = Default ECC algorithm, based on page size (Hamming from 512B page devices and Reed-Solomon for large page devices). 0x1 = Force Hamming ECC algorithm. 0x2 = Force RS algorithm (for large page devices only). 0x3 = Disable ECC calculation.
0x2–0x3	NAND flash page size	Relevant for boot from NAND flash only. Provides the bootROM firmware with the ability to boot from different NAND flash devices with different page sizes. If the Boot device is NAND flash, this field must contain the page size of the NAND flash used (for example, 0x800 for 2-KB page devices). 0x0 = Boot device is not NAND flash (page size is not relevant), this field is ignored.
0x4–0x7	Block size	Image size in bytes to be downloaded into DDR.
0x8–0xB	Reserved	Should be 0x0.
0xC–0xF	Source address	Boot from serial (SPI) flash and NAND flash: Image offset in bytes from the beginning of the flash device used for boot. For NAND flash devices, this address must be aligned to the boundaries of 512 byte. Necessary for the ECC calculation. Boot from SATA (88F619x and 88F6281 only): Image LBA location offset (in sectors) in the hard drive used for boot. Boot from UART0: Image location offset in bytes from the file transferred by the Xmodem. Boot from PCI Express: This parameter is not used, since the Root Complex device initiates the transfer of the image to the DDR. Set to 0xFFFFFFFF.
0x10–0x13	Destination address	Destination address in DDR where the image will be copied. If this address equals 0xFFFFFFFF, the image is not downloaded to DDR (relevant only for boot from serial (SPI) flash only). For Boot from PCI Express, this field indicates the start address of the image (Starting from this address, the 32-bit checksum is calculated and verified).

**Table 77: Main Header Format (Continued)**

<b>Byte</b>	<b>Field</b>	<b>Description / Usage</b>
0x14–0x17	Execution address	Address from which to start executing the image. This address is usually on the DDR, unless the destination address was set to 0xFFFFFFFF, where the execution address must be on the boot device (relevant only for boot from SPI flash).
0x18	Reserved	Must be 0x0.
0x19–0x1D	Reserved	Must be 0x0.
0x1E	Header extension	0x1 = An extra header of 480 bytes is appended directly after the main header. 0x0 = No extra header exists (relevant for SPI boot device only).
0x1F	Checksum	8-bit checksum of the main header The checksum is calculated by adding together each 8 bits of data. Each time the sum exceeds 0xFF, it restarts at zero (e.g., 0xF0 + 0x12 = 0x02).

#### 24.2.4.2 Header Extension Format

The header extension starts right after the main header, existing only if byte 0x1E of the main header does not equal 0x0. It is in Little Endian mode and is 480 bytes (480 = 512-32). The extension header is used for register configurations (usually DDR registers) before downloading the image to the DDR. Its format is as follows:

**Table 78: Header Extension Format**

<b>Byte</b>	<b>Field</b>	<b>Description / Usage</b>
0x0–0x3	Offset of Register configurations	The offset of register configuration information in this header extension. The offset is from the main header start. When this field is set to 0x0, there is no valid register configuration in the header extension.
0x4–0x1F	Reserved	Must be 0x0.
Offset that appears at 0x0	Register configuration	Registers configuration. The format is: <ul style="list-style-type: none"> <li>• Offset n is the address of the register.</li> <li>• Offset n+4 is the value of the register.</li> </ul> The actual registers configuration address space must end with zero terminators (the last 8 bytes must be zero). This field is used to modify any 32-bit address with any 32-bit value. This is useful for performing basic configuration before starting to execute the user image.
0x1FF	Checksum	8-bit checksum The checksum is calculated by adding each 8 bits of data together. Each time the sum exceeds 0xFF, it restarts at zero (e.g., 0xF0 + 0x12 = 0x2).

#### **24.2.4.3**

#### **Source Image Considerations**

BootROM firmware assumes the following for the images that can be booted from the device's interfaces:

- Image base address at the specific device interface is aligned to 32 bits (for NAND flash boot—88F619x and 88F6281, the image must be aligned to the boundaries of 512 bytes).
- Image size including the checksum is aligned to 32 bits.
- If the image will be downloaded to the DDR, the destination offset on the DDR is aligned to 32 bits.
- The image includes a simple 32-bit checksum in the last 4 bytes.
- The first instruction of the image is in Little Endian mode. If the image will be executed in Big Endian mode, the image must include the appropriate code to switch back to Big Endian mode.
- The first instruction is in ARM mode.

#### **24.2.5**

#### **BootROM Firmware Boot Sequence**

The bootROM firmware boot sequence can be divided into the following phases, which are described in the following sub-sections:

1. Initialization
2. Boot mode selection
3. Sensing the UART0 interface to detect a user request for entering the bootROM command line debug mode or for booting from UART0.
4. Load, check, and update the device's main header information.
5. If an extended header exists:
  - a) Load and check the device's extended header.
  - b) Execute all register configurations indicated in the extended header.
6. Load and check the device image (unless it is to be executed from the SPI).
7. Execute the device image code.

In addition, bootROM firmware contains the following functionality:

- Error reporting and handling
- Exception handling

##### **24.2.5.1**

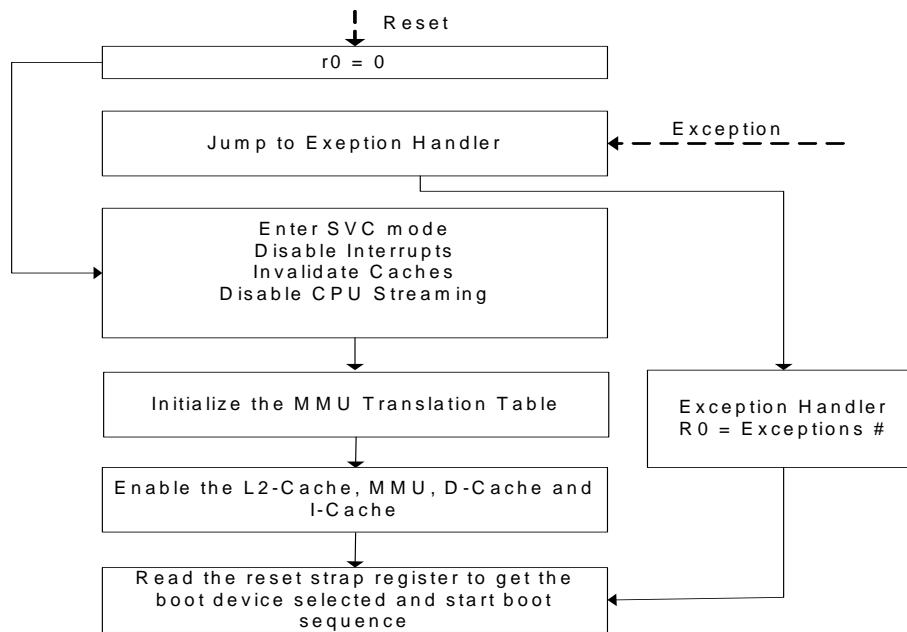
##### **Initialization**

After reset, the CPU starts running at 0xFFFFF0000, where the bootROM firmware code is located. It performs the following operations (in Assembly Language):

1. Sets r0 = 0.
2. Enters CPU supervisor mode and disables interrupts.
3. Invalidates caches and flushes TLB.
4. Disables CPU streaming.
5. Initializes the Translation table.
6. Enables L2-cache, MMU, D-Cache, and I-Cache.
7. Starts the boot sequence, by detecting the selected boot device.

This process is illustrated in [Figure 81](#).

**Figure 81: Initialization and Boot Method Selection Flow**



#### 24.2.5.2 Boot Device Selection

The Boot device is selected in the main routine, executed directly after initialization of the MMU and caches. The following operations are executed in the sequence listed:

1. Calls the UART Initialization routine, which initializes the UART0 to the 115,200-8N1 baud rate (8N1—8-bits of Data, No parity, 1 stop bit) according to the detected value of the TCLK.
2. If the main routine was executed as the result of an exception, then the Error Handler is called.
3. Call the Execution Handler.

#### Execution Handler

This routine reads the Sample at Reset Register (Table 800 p. 778) and calls the appropriate routine, according to the value of bits[13:12]. Also refer to *Boot Device* in the Reset Configuration table of the *Hardware Specifications*.

If the Execution Handler returns an error code, the error is registered in the Boot ROM Routine and Error Code Register (Table 791 p. 771).

#### 24.2.5.3 UART0 Sensing

Before starting a specific boot sequence indicated by the bootstrap, the bootROM tries to sense the UART0 interface, to detect a user request for entering bootROM command line debug mode or a request to boot from UART, using the Xmodem protocol.

Since UART0 pins are multiplexed pins, the bootROM attempts to detect the location of the Rx data signal, by first configuring MPP[4] as UA0\_RXD and trying to read from there with timeout. If the read process times out twice, the bootROM configures MPP[4] back to its default value, configures MPP[11] as UA0\_RXD, and tries again to read with timeout. If the second option fails, the bootROM configures MPP[11] back to its default value and moves back to the normal boot flow.

There are two modes of operating the UART interface based on the detected pattern on the RX side:

### Entering command line debug mode:

During power up, the user can transmit a special 64-bit debug pattern (0xDD 0x11 0x22 0x33 0x44 0x55 0x66 0x77) in a loop, to indicate a request for debug. When the bootROM begins execution, it attempts to read data from the UART0 interface (as explained above). If it successfully reads the debug pattern, it enters the command line debug mode. It configures the appropriate MPP pin to operate as a UA0\_TXD signal. (If a pattern is detected on MPP[4], then it configures MPP[5] as UA0\_TXD, otherwise it configures MPP[10] as UA0\_TXD.) Then it prints the bootROM version, and waits on the debug prompt for user input.

### Boot from UART:

If the bootROM detects the 64-bit UART boot pattern (0xBB 0x11 0x22 0x33 0x44 0x55 0x66 0x77) injected by the user, as an indication to start the Xmodem protocol and the UART boot process; it configures the appropriate MPPs to operate as UA0\_RXD and UA0\_TXD, as in the previous sub-section and starts the Xmodem protocol to load the image from UART to the DDR.

## 24.2.5.4

### Header Decoding, DDR Initialization, and Image Execution

Each of the boot methods is described in detail in [Section 24.2.6, BootROM Firmware Boot Options](#). However, all boot methods use the same logic for reading/decoding the header, and execute the loaded image. The following describes the flow (also illustrated in [Figure 82](#)).

#### 1. Main Header decoding:

- a) Read the main header (first 32 bytes) from the boot device into the stack (for NAND flash boot, 512 bytes of data are loaded, since ECC is calculated over chunks of 512 bytes).
- b) Checksum is calculated and verified on the main header.
- c) If the checksum is valid, verify the header ID. If either is invalid, report an error.

#### 2. Header Extension decoding:

- a) If byte 0x1E of the main header is 0x0, no extension header exists, therefore no register configuration occurs. Proceed to step 3.  
If byte 0x1E of the main header is not 0x0, an extension header exists.
- b) Load the extension header from offset 0x20 into the stack (for NAND flash boot this is not necessary, since the extension header was already loaded, together with the main header).
- c) Verify the checksum. If the checksum is invalid, report an error.
- d) If offset 0x0 in the extension header is not 0, there are valid register configurations in the header extension.  
The bootROM parses the Address/Value list, writing each value in the appropriate Address until it reaches the first 0x0/0x0 couple.  
If offset 0x0 in the extension header is 0, no register configuration occurs.

#### 3. Image Loading and verification:

For boot from SATA, NAND flash, or SPI (with image copy to DDR), the image must be loaded to the DDR and executed from there. The following sequence is performed:

- a) Copy the image from the Source Address indicated in bytes 0xC–0xF of the main header to the Destination Address indicated in bytes 0x10–0x13. The size of the image to be copied is indicated in bytes 0x4–0x7. (This size includes the extra 4 bytes of the checksum-32.)  
**NOTE:** For NAND flash and SPI flash, the Source Address is the offset in bytes from the start of the device (where the main header resides).  
For SATA, the Source Address is the LBA address (i.e., the number of the first sector of the partition containing the image).
- b) Calculate the checksum-32 on the entire image (in the DDR), without the last 4 bytes, and compare the result to the last 4 bytes of the image.
- c) If the calculated checksum-32 and the recorded checksum-32 (following the image) differ; report the error; else proceed to the next step.

For boot from SPI, the image can optionally be executed from the same location, without copying it to the RAM. The following sequence is performed:

- a) The destination must be 0xFFFFFFFF, as an indication that no copy is necessary and the image must be executed in place (i.e., executed directly from the SPI flash, using the directly mapped memory space).
- b) The Source Address indicates the offset in bytes (from the beginning of the SPI flash, where the main header resides) at which the image starts.
- c) Calculate the checksum-32 on the entire image (on the SPI flash) without the last 4 bytes, and compare the result to the last 4 bytes of the image.
- d) If the calculated checksum-32 and the recorded checksum-32 (following the image) differ; report the error; otherwise proceed to the next step.

For boot from PCI Express, the image is loaded by the Root Complex and no copying is necessary. The following sequence is performed:

- a) The bootROM starts by initializing the PCI Express Boot Address Register ([Table 309 p. 477](#)) with 0xFFFFFFFF.  
It then performs PCI Express interface initialization, setting it to Endpoint mode.  
Next it enters an infinite loop, waiting for the Root Complex to change the [PCI Express Boot Address Register](#) and update it with the address holding the image header. This is a handshake, indicating that the Root Complex has completed the necessary DDR configuration and downloaded the image to the execution location.
- b) When the [PCI Express Boot Address Register](#) has been updated with the location of the main header address, the bootROM verifies the main header.
- c) The Source Address in the header is **not used** if the boot device is PCI Express.
- d) The Destination Address specifies the location of the image.
- e) The bootROM calculates the checksum-32 on the image in the location specified in the Destination Address.
- f) If the calculated checksum-32 and the recorded checksum-32 (following the image) differ; report the error; otherwise proceed to the next step.

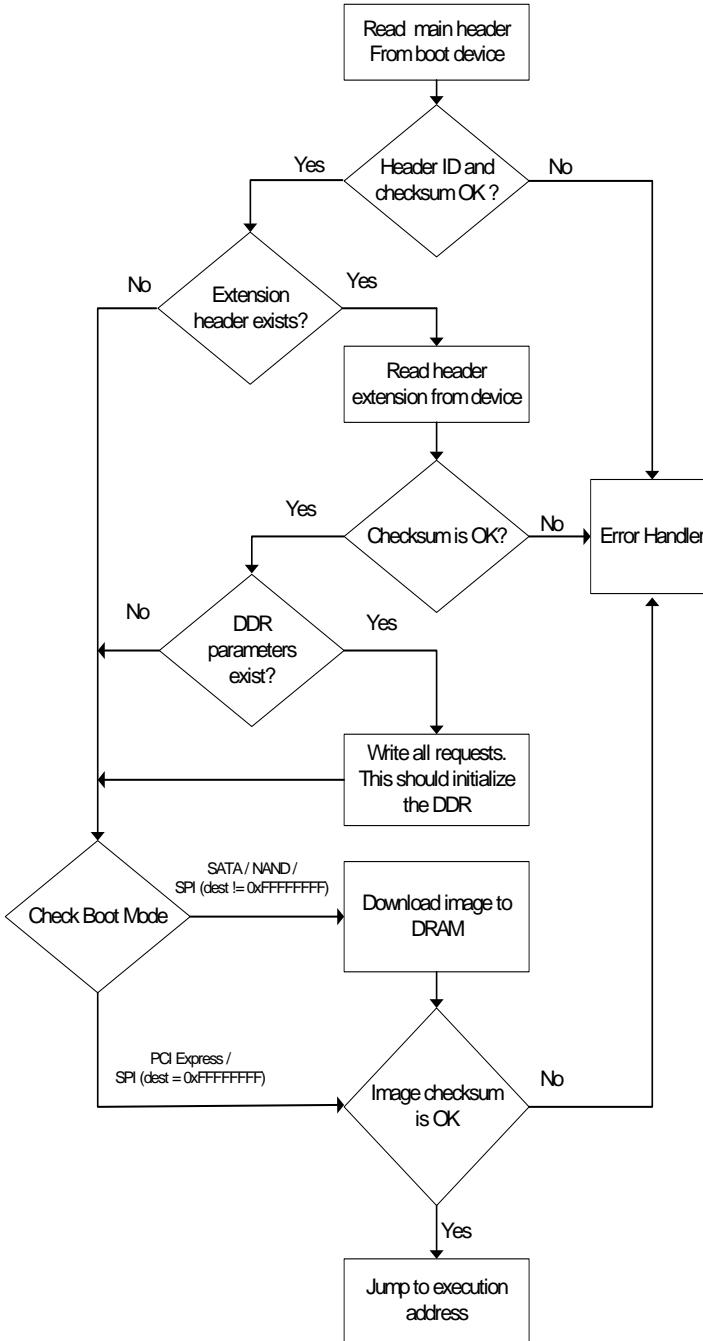
#### 4. Image execution:

- a) Disable MMU, I-Cache, and D-Cache.
- a) Disable L2 cache.
- b) Flush L2-Cache, I-Cache, and D-Cache
- c) Jump to the execution address that was extracted from the main header.

**Note**

If any error occurs during the entire process, it is registered in the [<Error Code>](#) field in the Boot ROM Routine and Error Code Register ([Table 791 p. 771](#)), and the entire process is restarted from the early boot stages, where the sample at reset configuration is read from the bootstrap register.

**Figure 82: Header Decoding, DDR Initialization, and Image Execution Flowchart**



## 24.2.5.5 Debug and Error Handling

### BootROM Firmware Error Registers

All boot methods use the same Error Handling mechanism (see [Section 24.4, Error Handling Functional Description, on page 309](#)). This mechanism takes advantage of the Boot ROM Routine and Error Code Register ([Table 791 p. 771](#)).

### BootROM Firmware Error Handling Description

Error Handling is performed as follows:

1. On error, the specific boot method calls the Error Handler with the appropriate <Error Code> and <Error Location> field in the Boot ROM Routine and Error Code Register ([Table 791 p. 771](#)) representing the current execution method.
2. The Error Handler automatically increments the <Retry\_Count> field
3. If this is the first call to the Error Handler (checked by examining the <Retry\_Count> field), it updates the <Error Code> and <Error Location> fields.
4. The bootROM tries to perform the boot method again by jumping to the main routine.
5. If the retry count exceeds 16, the bootROM stops the retry mechanism and tries to sense the UART0 port, looking for one of the two UART patterns.

## 24.2.6 BootROM Firmware Boot Options

### 24.2.6.1 Boot from UART0

As previously indicated, boot from UART0 is not a sample at reset configuration option, but is performed before every type of boot through sensing the Rx side of the UART0 interface on both MPP options.

If the debug pattern (0xDD 0x11 0x22 0x33 0x44 0x55 0x66 0x77) was received, the bootROM enters the command line debug mode. Otherwise, if the boot sequence (0xBB 0x11 0x22 0x33 0x44 0x55 0x66 0x77) was received, the bootROM enters the Boot from UART routine, which starts receiving the boot image using the Xmodem protocol (i.e., it starts receiving chunks of 128 bytes, each with an 8-bit checksum).

The boot image must be a continuous image, which includes a main header and an extension header (extended header is mandatory in this case, since the DDR must be initialized).

The main header, extension header, and source image must contain valid checksum values.

According to the boot sequence described above, this boot method:

1. Executes the registers configuration in the extended header.
2. Downloads the source image to DDR.
3. Executes it in the DDR.

### 24.2.6.2 Boot from Serial (SPI) Flash

In this boot method, a boot image must be located on the external serial (SPI) Flash. The main header must exist at offset 0 of the External SPI flash. An extension header may exist, if the extended header bit is set in the main header.

The main header, extension header, and source image must contain valid checksum values.

The boot from serial (SPI) flash can be performed in two ways, based on the value of the Destination Address specified in the main header.

- If the destination field holds a value different from 0xFFFFFFFF, it downloads the source image to the DDR address specified in the destination field and executes it from the address indicated in the execution address field. In this case, an extension header is mandatory, to perform the necessary DDR initialization (since the image must be copied to the DDR).

- If the destination field equals 0xFFFFFFFF, it simply executes the source image directly from the serial (SPI) flash, using SPI directly mapped memory space.

#### 24.2.6.3 Boot from NAND Flash

In this boot method, a boot image must be located on the first page of the NAND flash. The main header must exist at offset 0, and an extension header is mandatory for this boot device, to perform DDR initialization (since the image must be copied to the DDR to be executed, and it cannot be executed directly from NAND flash).



**Note**

In this boot mode, <NFActCEnBoot> field in the NAND Flash Control Register ([Table 613 p. 676](#)) is set to 1, which enables the NAND flash controller to hold the CEn signal asserted throughout the entire read phase.

The source image must be located at the offset specified by the main header.

The main header, extension header, and source image must contain valid checksum values.

The source image is downloaded to the DDR byte-by-byte, using a NAND flash software protocol.

Since there are different types of NAND flash devices, with different read command sequences, the bootROM implements a detection mechanism, to support most of the NAND flash types. The bootROM tries to read the first 512 bytes in four different ways, and uses the main header checksum check as a success indication.

Following are the four types of NAND flash read commands used, listed in the order they are tried by the bootROM:

**Table 79: Types of NAND Flash Read Commands Supported**

	NAND Flash Type	Read Command Sequence
1	Large pages 5 address cycles with 0x30 command trailer	Command 0x00, Address 0–7, Address 8–11, Address 12–19, Address 20–27, Address 28 and above, Command 0x30 <b>NOTE:</b> For pages larger than 2K, the size should be indicated in the main header of the BootROM.
2	512 byte pages 3 address cycles	Plane A (Address bit-8 = 0): Command 0x00, Address 0–7, Address 9–16, Address 17–24 Plane B (Address bit-8 = 1): Command 0x01, Address 0–7, Address 9–16, Address 17–24
3	Large pages 4 address cycles with 0x30 command trailer	Command 0x00, Address 0–7, Address 8–11, Address 12–19, Address 20–27, Command 0x30 <b>NOTE:</b> For pages larger than 2K, the size should be indicated in the main header of the BootROM.
4	512 byte pages 4 address cycles	Plane A (Address bit-8 = 0): Command 0x00, Address 0–7, Address 9–16, Address 17–24, Address 25–26 Plane B (Address bit-8 = 1): Command 0x01, Address 0–7, Address 9–16, Address 17–24, Address 25–26

The bootROM tries to boot from four types of NAND flash devices four times (a total of 16 times). The first eight trials are performed with ECC calculation and the last eight trials are performed skipping ECC correction. A NAND flash reset (using the command 0xFF) is performed before each boot trial.

On each trial, the bootROM checks the 32-bit checksum on the image copied to the RAM. If the checksum fails, the bootROM registers the error, increments the retry count, and restarts the boot process. The bootROM supports both 4-bit RS ECC (Reed-Solomon Error Correcting Code) and 1-bit Hamming ECC (Error Correcting Code). For Large Page NAND flash devices, RS ECC is used, while for small page devices, Hamming ECC is used.

The following are the ECC algorithms supported per NAND flash types.

**Table 80: Types of ECC Protocols Supported per Flash Type**

	<b>NAND Flash Type</b>	<b>Read Command Sequence</b>
1	Large pages 4 or 5 address cycles with 0x30 command trailer	RS-ECC with 4-bit detection/correction per 512B of data.
2	512 byte page 3 and 4 address cycles	Hamming with 2-bit detection, with 1-bit correction per 256B of data (23-bit ECC).

### **Bad Block Management**

The BootROM supports bad block skipping. Before reading from a block, it is verified to be a good block, by checking the appropriate OOB byte (or bytes) in the Spare area to be 0xFF.

For the 512B page devices, the number of pages per block is fixed at 32 (and thus block size is 16 KB).

For Large page devices (2 KB and over), the block size, pages per block number, and the technology used (cell type is MLC or SLC) are read at runtime using the READID command (in bytes 3 and 4).

The following are the OOB locations checked by the bootROM based on the type of NAND flash selected through the reset strap.

**Table 81: Bad Block Indicators per NAND Flash Cell Type**

	<b>NAND Flash Type</b>	<b>Read Command Sequence</b>
1	Large page MLC devices	Byte[0] of the spare area in the last page of the block (For a good block, the byte should be equal to 0xFF).
2	Large page SLC devices	Byte[0] and Byte[5] of the spare area in the first and second pages of the block (For a good block, both bytes should be equal to 0xFF).
3	512B page SLC devices	Byte[5] of the spare area in the first and second pages of the block (For a good block, the bytes should be equal to 0xFF).

#### **24.2.6.4 Boot from a SATA Device**

In this boot device, the main header must be located in sector number 1 of the hard disk (since sector 0 holds the partition table). An extension header is mandatory, to perform DDR initialization (since the image must be copied to the DDR to be executed from there and cannot be executed in place).

The source image must be located at the beginning of the sector specified by the main header. (Usually the first partition starts in sector 63.)

The main header, extension header, and source image must contain valid checksum values.

This boot method downloads the source image to the DDR and executes from there. The source image is downloaded to DDR using the DMA.

The first 2 KB of SDRAM are used for the SATA descriptors.

#### **24.2.6.5**

#### **Boot from PCI Express Interface**

This boot device differs from the other four boot devices. The device functions as a PCI Express Endpoint, with a Root Complex, which is responsible for performing the basic steps of the boot process.

In this boot mode, the first task the bootROM performs is to set the PCI Express Boot Address Register ([Table 309 p. 477](#)) with the value 0xFFFFFFFF.

Then the bootROM initializes the PCI Express interface and configures the PCI Express controller to function as an Endpoint. By default, the device configures its BARs to allow access to its internal registers. This allows the Root Complex to use BAR 0 to configure the device, initialize the DDR (if it exists) and set up the DDR BAR 1 and windows, if necessary.

After performing the interface initialization, the bootROM enters an infinite loop, waiting for the Root Complex to change the [PCI Express Boot Address Register](#) and update it with the address (locally on the DDR) holding the image header. This is a handshake mechanism, which indicates that the Root Complex is ready, having performed the necessary configurations and downloaded the image to the execution location on the DDR.

After detecting the change in the [PCI Express Boot Address Register](#), the bootROM uses the value passed as the location of the main header and verifies the header checksum to make sure that it is valid.

The bootROM uses the Destination Address as the absolute location of the image to check the image checksum-32.

Finally, if the checksum is valid, the bootROM jumps to the Execution Address specified in the main header and starts running from there.

### **24.3**

## **Power Management**

#### **24.3.1**

### **Functional Description**

The device includes various power management (PM) features that enable fine-tuning of the device's power consumption, according to the desired usage scheme.

For a detailed description of power management in this device, refer to *AN-260 System Power-Saving Methods for 88F6180, 88F6190, 88F6192, and 88F6281*.

For the exact power consumption values for each device, refer to the respective device *Hardware Specifications*.

#### **24.3.2**

### **CPU Power Saving**

This section describes the CPU power saving options.

#### **24.3.2.1**

### **Wait for Interrupt CP15 Mode**

The CPU Wait for Interrupt CP15 mode disables all the CPU Core clocks, including logic, caches, MMU and tables. It also disables the L2 controller and array clocks. The CPU preserves the latest data and caches. Once the CPU wakes up, it proceeds from the last executed instruction.

To enter this mode, execute the Wait For Interrupt CP15 instruction.

The transition from this mode to active mode is caused by one of the following:

- Assertion of an interrupt (nIRQ) or fast interrupt (nFIQ), whether masked on unmasked.
- Assertion of reset.

For additional information, refer to the *Sheeva™ 88SV131 ARM v5TE Processor Core with MMU and L1/L2 Cache Datasheet*.

#### 24.3.2.2

#### Dynamic Frequency Scaling

The CPU Subsystem dynamic frequency scaling enables a fast transition between Fast Clock mode and Slow Clock mode per application demand.

- In Fast Clock mode, the CPU Subsystem (CPU and L2 cache) operate at fast frequencies, as determined at reset.
- In Slow Clock mode, the CPU Subsystem clock is lowered to the DDR frequency.

To enable the Slow Clock mode, perform the following sequence:

1. Set the <CPU\_SW\_Int\_Blk> field in the CPU Control and Status Register ([Table 116 p. 368](#)) to 0x1. This setting disables interrupts to the CPU.
2. To enable the Power Saving mode, set the <GotoPowerSave> field in the Clock Gating Control Register ([Table 122 p. 372](#)).
3. Use the MCR CP15 Wait-For-Interrupt command.
4. Clear the <CPU\_SW\_Int\_Blk>. This enables the interrupts to the CPU.

To return to Fast Clock mode, perform the following sequence:

1. Set the <CPU\_SW\_Int\_Blk> to 0x1. This setting disables interrupts to the CPU.
2. To disable the Power Saving mode, clear the <GotoPowerSave>.
3. Use MCR CP15 Wait-For-Interrupt command.
4. Clear the <CPU\_SW\_Int\_Blk>. This enables the interrupts to the CPU.



##### Note

When entering the Wait For Interrupt mode (step 3 in both procedures), there is no need to assert the interrupt for waking up. The Clock Generation logic takes care of this, and once the clocks adjust to the needed frequencies, they automatically wake the CPU Subsystem clocks (Core and L2 cache). During the transition period, the DDR Clock remains intact and keeps its frequency and phase throughout the procedure, making it transparent to the system.

#### 24.3.3

#### SDRAM Power Saving

The SDRAM power consumption depends on SDRAM technology (DDR2), density, operation frequency, and operating mode. [Table 82](#) list the typical values for a single 512 Mbit (DDR400) SDRAM device.

**Table 82: 512 Mb—SDRAM IDD Values**

Operating Condition	Symbol	DDR2
Full Operating Power Four bank interleaving read access.	IDD7	230 mA
Idle Current No activity and all pages are closed.	IDD2	40 mA
Self Refresh Current	IDD6	5 mA

In Self Refresh mode, the SDRAM current is far less than with other operating conditions. This mode is useful when the system is in Standby mode, and no activity is expected in the next few milliseconds.

To set the SDRAM to Self Refresh mode, set the [<Cmd>](#) field in the SDRAM Operation Register ([Table 174 p. 400](#)) to 0x7. After 256 cycles from when the [<Cmd>](#) field is set, the SDRAM controller waits for an idle state and then sets the SDRAM to Self Refresh mode.

When in Self Refresh mode:

- The SDRAM controller does not generate refresh cycles. The SDRAM has an internal refresh counter that manages the SDRAM refresh.
- Excluding the M\_CLKOUT, M\_CLKOUTn, M\_CKE, and M\_STARTBURST signals, all of the other SDRAM interface signals are floated, resulting in additional power-saving.

When new pending transactions are targeted to the SDRAM, the SDRAM controller restores the SDRAM interface to normal operation.

Resuming normal operation requires 200 SYS\_CLK cycles. Due to this requirement, Marvell® recommends using the SDRAM Self Refresh mode only when the system is really in a Standby mode. This prevents frequent Self Refresh enter/exit operations that can result in performance degradation.



**Note**

Do not use Self Refresh mode if an external PLL-based clock buffer is used for clock fan out toward the SDRAM.

---

#### 24.3.4 PCI Express Power Saving

The device supports PCI Express power management as a root complex (controlling external PCI Express endpoints), or as an endpoint.

When there is no device connected to the PCI Express interface, it is possible to power down the PCI Express PHY. Shutting down the PCI Express PHY reduces the device power consumption.

To power down the PCI Express PHY, write the value 0x20800087 to the PCI Express PHY Indirect Access Register ([Table 319 p. 484](#)).

It is also possible to reduce the device power consumption by disabling the PCI Express clock. To accomplish this, set the [<PEX0\\_Mem\\_PD>](#) field in the Memory Power Management Control Register ([Table 121 p. 370](#)).

##### 24.3.4.1 Device as a PCI Express Root Complex

When the device is in Root Complex mode, the operating system should perform the following sequence, to set the external PCI Express endpoint into one of the power-saving modes:

1. Requests the endpoint software driver to complete all of its tasks, and sets the device into a power-down state.
2. Sets the endpoint [<PMState>](#) field in the PCI Express Power Management Control and Status Register ([Table 286 p. 459](#)) to the required power state.

As a result, the endpoint initiates a link power-down procedure. This turns the device PCI Express PHY to power-saving mode.



**Note**

When in D3 power state, the external endpoint is only permitted to respond to configuration cycles. The software must not access the external device with memory or I/O transactions. These types of accesses result in a master abort condition.

If the external PCI Express endpoint requires a wake up (for example, a NIC device that received a wake-up packet), it sends a PME message. As a result, the [<RcvPmPme>](#) field in the PCI Express

Interrupt Cause Register ([Table 320 p. 487](#)) is set and a CPU interrupt is asserted, if not masked. In response, the operating system:

1. Sets the endpoint back to *D0* state for normal operation, and clears the device's PMEn interrupt by writing 1 to the [`<PMEnStat>`](#) field in the PCI Express Power Management Control and Status Register ([Table 286 p. 460](#)).
2. Signals the endpoint software driver to resume normal operation.

Any access to the PCI Express endpoint (for example, a CPU write to the endpoint's [`<PMState>`](#)) results in the link resuming its full power state.

#### 24.3.4.2 Device as a PCI Express Endpoint

As a PCI Express endpoint, the device implements the required power management configuration registers, including PME message and PHY power down.

For the external system host to set the device into one of the power-saving modes, the external host must perform the following steps:

1. Request the device software driver to complete all of its tasks.
2. Set the [`<PMState>`](#) to the required power state.

As a result, the PCI Express interface initiates a link power-down process, that leads to a PHY power down.

Due to the [`<PMState>`](#) field change, the device sets the [`<DstateChange>`](#) field in the PCI Express Interrupt Cause Register ([Table 320 p. 485](#)) and asserts a CPU interrupt, if not masked. The local CPU can use this interrupt to set the board components to power-saving mode.



When in *D3* power state, the device resets its PCI Express configuration registers to their default values, as required by the PCI Express specification.

For the external system host to return the device to normal operation, it must perform the following steps:

1. Clear the [`<PMState>`](#) field to 0x0 (*D0* state).
2. Signal to the device software driver to resume normal operation.

As a result of a [`<PMState>`](#) field change, the device sets the [`<DstateChange>`](#) and asserts a CPU interrupt, if not masked. The local CPU can use this interrupt to return the board components to normal operation.

The device also supports PMEn message generation. Upon a wake-up event, the local CPU attached to the device is expected to trigger a PMEn message by writing 1 to the [`<PMEnStat>`](#) field in the PCI Express Power Management Control and Status Register ([Table 286 p. 460](#)). In response, the system host sets the device back to the *D0* state, and clears the PMEn interrupt by writing 1 to the [`<PMEnStat>`](#).

#### 24.3.4.3 L0 Power Saving

The device also support PCI Express L0 Rx and Tx power-saving states, as defined in the PCI Express specification. If there is no traffic on the link for seven microseconds, the PCI Express PHY initiates power down on its Tx side.

Similarly, the external PCI Express device may initiate L0 power down, causing an L0 power down on the device PCI Express Rx side.

#### 24.3.4.4

#### Endpoint Power Down (Extended D3 Power Saving)

If it is necessary to extend the power-saving methods beyond the *D3* state, as described in [Section 24.3.4.1, Device as a PCI Express Root Complex, on page 305](#), there is an option to power down the endpoint device by using the following sequence:

1. The operating system requests the endpoint software driver to complete all of its tasks, and then, it sets the device to a power-down state.
2. The operating system sets the endpoint [`<PMState>`](#) field in the PCI Express Power Management Control and Status Register ([Table 286 p. 459](#)) to 0x3, *D3* power-saving state.
3. Read the D-state register from the endpoint device and verify that the *D3* is written in the [`<PMState>`](#) field.
4. Clear the [`<PexLinkdownResetCpu>`](#) field in the CPU Configuration Register ([Table 115 p. 367](#)).
5. Wait 20 ms.
6. Turn off the power to the endpoint device, this is system board implementation dependent. For example, a board designer can use an MPP output to disable/enable the dedicated endpoint power supplies, see [Figure 83, "Endpoint Power Supply Control](#).

To power up the endpoint device, use the following sequence:

1. Turn on power to the endpoint device. The period required to power up the endpoint device depends on the reset sequence.
2. Clear the [`<PexLinkdownResetCpu>`](#) to 0.
3. Wait for the [`<DLDown>`](#) field in the PCI Express Status Register ([Table 308 p. 476](#)) to be cleared to 0 (the DL link is active).

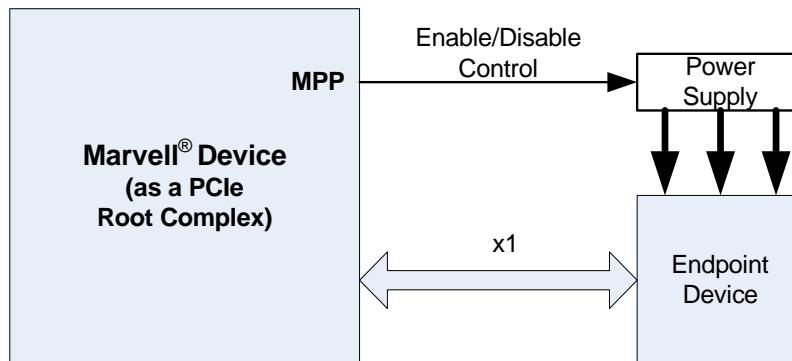
The endpoint device is ready for enumeration and software initialization.



**Note**

When designing the software, take into account the fact that the endpoint device initialization is software dependent.

**Figure 83: Endpoint Power Supply Control**



#### 24.3.5

#### Ethernet Power Saving

It is possible to reduce the device power consumption by disabling the Ethernet port and its clock.

To disable the port, proceed as follows:

1. Set the active queues in the [\*\*<DISQ>\*\*](#) field in the Receive Queue Command (RQC) Register ([Table 444 p. 585](#)) and in the [\*\*<DISQ>\*\*](#) field in the Transmit Queue Command (TQC) Register ([Table 450 p. 590](#)). There is one bit per queue. This action stops all active Rx and Tx queues.
2. Verify that all of the bits in the [\*\*<ENQ>\*\*](#) field in the Receive Queue Command (RQC) Register ([Table 444 p. 585](#)) and in the [\*\*<ENQ>\*\*](#) field in the Transmit Queue Command (TQC) Register ([Table 450 p. 590](#)) are 0. When these bits are set to 0, it indicates that all of the queues have stopped.
3. Read the Ethernet Port Status 0 (PS0) Register ([Table 429 p. 569](#)) register to confirm that the [\*\*<TxFIFOEmpl>\*\*](#) and [\*\*<TxInProg>\*\*](#) fields each have a value of 1. This value means that the Tx FIFO is empty and that transmit activity has stopped.
4. Clear the [\*\*<ForceLinkFail>\*\*](#) field in the Port Serial Control0 (PSC0) Register ([Table 427 p. 568](#)) to force the link down.
5. Clear the [\*\*<PortEn>\*\*](#) field in the Port Serial Control0 (PSC0) Register ([Table 427 p. 566](#)) to disable the port.

To disable the Ethernet port clock, set the [\*\*<GE0\\_Mem\\_PD>\*\*](#) field and the [\*\*<GE1\\_Mem\\_PD>\*\*](#) field in the Memory Power Management Control Register ([Table 121 p. 371](#)).

#### 24.3.6

### USB Power Saving

The device supports power down of any attached USB device, as specified in the EHCI specification.



**Note** The bits mentioned in this section are in the PORTSC register (offset 0x50184). Refer to the *ARC USB-HS OTG High-Speed USB On-The-Go Controller Core V 4.0.1 Reference*.

To configure the USB port to Suspend mode, set the PORTSC register Suspend field (SUSP bit[7]) to 1. The USB port stops all bus activity, resulting in power down of the attached USB devices.

To resume normal operation, set the PORTSC register Force Port Resume field (FPR bit[6]) to 1. After resuming normal operation, the USB MAC sets the PORTSC register Force Port Resume field back to 0.

The USB port may be in an inactive state (no external device connected). Upon connection of an external device, the PORTSC register Connect Status Change field (CSC bit[1]) is set, and an interrupt is asserted, if not masked. The software then wakes up the bus. Similarly, upon a disconnect event, the software turns the USB port off.

#### 24.3.7

### SATA Power Saving

When using the SATA interface, it is possible to reduce the device power consumption by disabling the SATA PHY. If the SATA interface is not used, it is still recommended to disable the SATA PHY to ensure the minimal device power consumption.

To disable the PHY, shut down the link and then disable the port PHY.

#### Shutting down the port link

To disable the PHY, shut down the link, and then, disable the port PHY.

To shut down the link:

1. Set the [\*\*<SPM>\*\*](#) field in the SControl Register ([Table 369 p. 525](#)) to:
  - 0x1 = Initiate a Partial power management mode, a faster, but less effective power management state.  
OR
  - 0x2 = Initiate Slumber power management mode.

2. Ensure that the <IPM> field represents the enabled interface power management states that can be invoked via the Serial ATA interface power management capabilities.

### Disabling the port PHY

After the <SPD> field is set, disable the port PHY by using the following sequence:

1. In the PHY Mode 2 Register ([Table 374 p. 532](#)), clear the <FORCE\_PU\_TX>, <FORCE\_PU\_RX>, <PU\_PLL>, and <PU\_IVREF> fields.
2. Set the <PhyShutdown> field in the Serial-ATA Interface Configuration Register ([Table 365 p. 519](#)) to 0x1. This setting places the PHY in Shutdown mode.

### Reactivating the SATA port

To reactivate the SATA port, perform the following sequence:

1. Clear the <PhyShutdown>. This setting places the PHY in Operational mode.
2. Set the <FORCE\_PU\_TX>, <FORCE\_PU\_RX>, <PU\_PLL>, and <PU\_IVREF> to 0x1.
3. Set the <SPD> field in the SControl Register ([Table 369 p. 525](#)) to 0x3. This setting places the PHY in Active mode.

## 24.3.8

### Cryptographic Engines and Security Accelerator (CESA)

The CESA unit does not have any external interfaces (no PHY is involved), which simplifies power management. In addition, the CESA unit was designed with an orientation to power-saving modes.

Currently, the only sequence that can be performed is to enable the automatic power-save mode. This mode manages enabling/disabling the clock for the entire unit. The unit disables the clock whenever it is idle and enables it with the first access to the unit.

To enable the automatic power saving mode, set the <Crypto\_Mem\_PD> field in the Memory Power Management Control Register ([Table 121 p. 371](#)).

## 24.3.9

### Core Clock Power Saving

It is possible to enhance power saving by shutting down the core clock (TCLK) supply to inactive interfaces. If an interface is not used, it is possible to disable the core clock input to that interface. Use the Memory Power Management Control Register ([Table 121 p. 370](#)) to control the clock input for the different interfaces.

## 24.4

### Error Handling Functional Description

The device provides error handling for the following types of errors:

- CPU address decoding errors
- PCI Express errors
- USB errors



Note

As the DDR controller does not support ECC, no errors are generated by the DDR controller.

---

## 24.4.1 CPU Address Decoding Errors

[Table 83](#) lists the CPU address decoding errors and describes how they are handled.

**Table 83: CPU Address Decoding Error Handling**

Error Type	Error Handling
Access to unmapped window	Accesses are completed according to the setting of bit <a href="#"><code>&lt;AHBErrorProp&gt;</code></a> field in the CPU Configuration Register ( <a href="#">Table 115 p. 366</a> ). <a href="#"><code>&lt;AHBErrorProp&gt;</code></a> = 0 Error indications are not propagate to Mbus-L. The transactions are completed normally. <a href="#"><code>&lt;AHBErrorProp&gt;</code></a> = 1 Error indications are propagate to Mbus-L.
Write Access to write protected window	
Other errors	Unpredictable behavior.

## 24.4.2 PCI Express Errors

[Table 84](#) lists the PCI Express errors and describes how they are handled.

**Table 84: PCI Express Error Handling**

Flow	Error Type	Error Handling
PCI Express Master Write	Error indication from initiator unit	Forward the transactions with data poisoning indications to the PCI Express interface.
PCI Express Master Read completion	1. Data Poisoning from PCI Express interface 2. Completion timeout 3. Received completion with unsuccessful completion status	Forward the transactions with error indications to the initiator. <sup>1</sup>
PCI Express Slave Write	Data Poisoning from PCI Express interface	Drop the data. Close the transactions normally.
PCI Express Slave Read	Error indication from target unit	Forward the transactions with data poisoning indications to the PCI Express interface.
Link Fail		1. Reset link state machine. 2. Generate maskable interrupt. 3. Activate system reset if enabled.
Hot reset received (Endpoint mode)		
PHY/Link/Transport Unrecoverable error		Set maskable interrupt.
PHY/Link/Transport Recoverable error		
Address Decoding errors		Unpredictable behavior.

1. The CPU is the initiator. The transaction is propagated on the Mbus-L if the [`<AHBErrorProp>`](#) field in the CPU Configuration Register ([Table 115 p. 366](#)) is set to 1.

## 24.4.3 USB Errors

[Table 85](#) lists the USB errors and describes how they are handled.

**Table 85: USB Error Handling**

Flow	Error Type	Error Handling
Address Decoding errors	No hit or multiple hit on address windows	Transactions are forwarded according to window0. Generate maskable interrupts.
USB Controller errors		Handled by the USB controller core according to the USB 2.0 specification.

# 25 Internal Architecture

## 25.1 Mbus-L—Sheeva™ CPU Core Local Bus

The device CPU uses the Mbus-Light (Mbus-L) protocol as the internal interface to the rest of the device.

The Mbus-L protocol is targeted at a high-performance, high-frequency CPU. It is intended to maximize the device CPU Bus Interface Unit (BIU) throughput, and to minimize the bus read latency. The Mbus-L provides the following features:

- Separate address/control and data phases—burst-based transactions with only start addresses:
  - Enables multiple outstanding read transactions
  - Enables new read/write requests during the read response data phase
  - Enables new read transfer requests during the write data phase
- Separate write data and read data interfaces:
  - Enables write data transfer and read response data transfer to occur simultaneously
  - Guarantees streaming read response transfer, no acknowledgment needed
- Separate interfaces to DDR controller and to Mbus bridge:
  - Enables simultaneous transactions to DDR controller and to Mbus bridge
  - Guarantees streaming read response transfer, no acknowledgment needed
- Transaction ID is attached to the transaction command:
  - Enables out-of-order transaction completion

[Figure 84](#) provides a block diagram of the 88F6180 and 88F619x Bus Interface Unit Mbus-L.

**Figure 84: 88F6180 and 88F619x Bus Interface Unit Mbus-L Block Diagram**

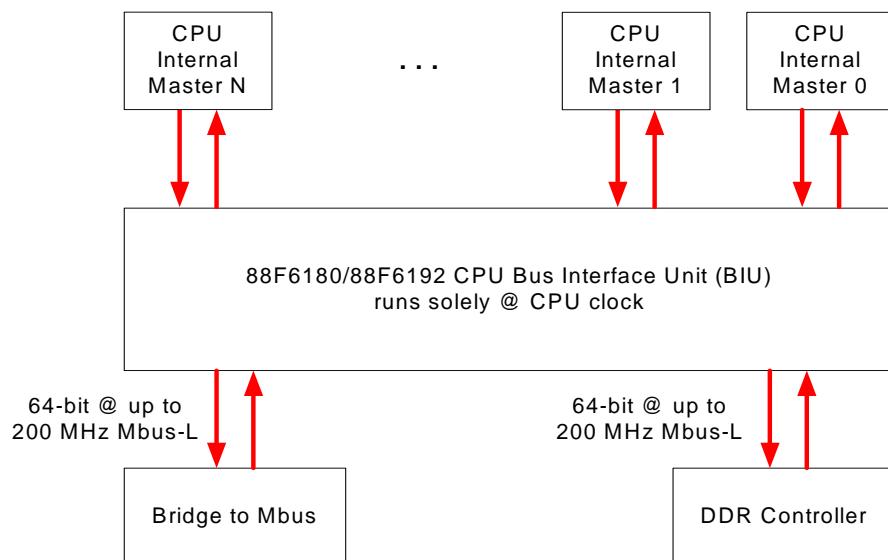
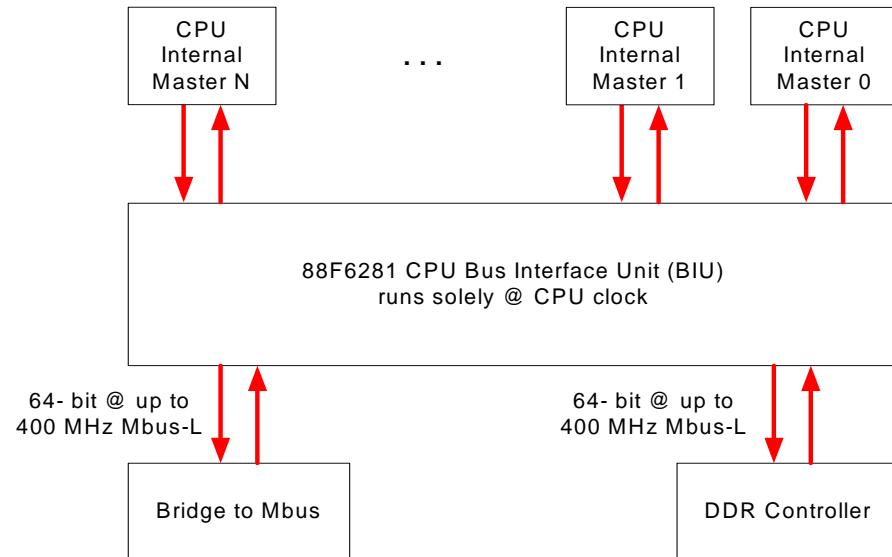


Figure 85 provides a block diagram of the 88F6281 Bus Interface Unit Mbus-L.

**Figure 85: 88F6281 Bus Interface Unit Mbus-L Block Diagram**



### 25.1.1 CPU Throughput

To further utilize Mbus-L capabilities:

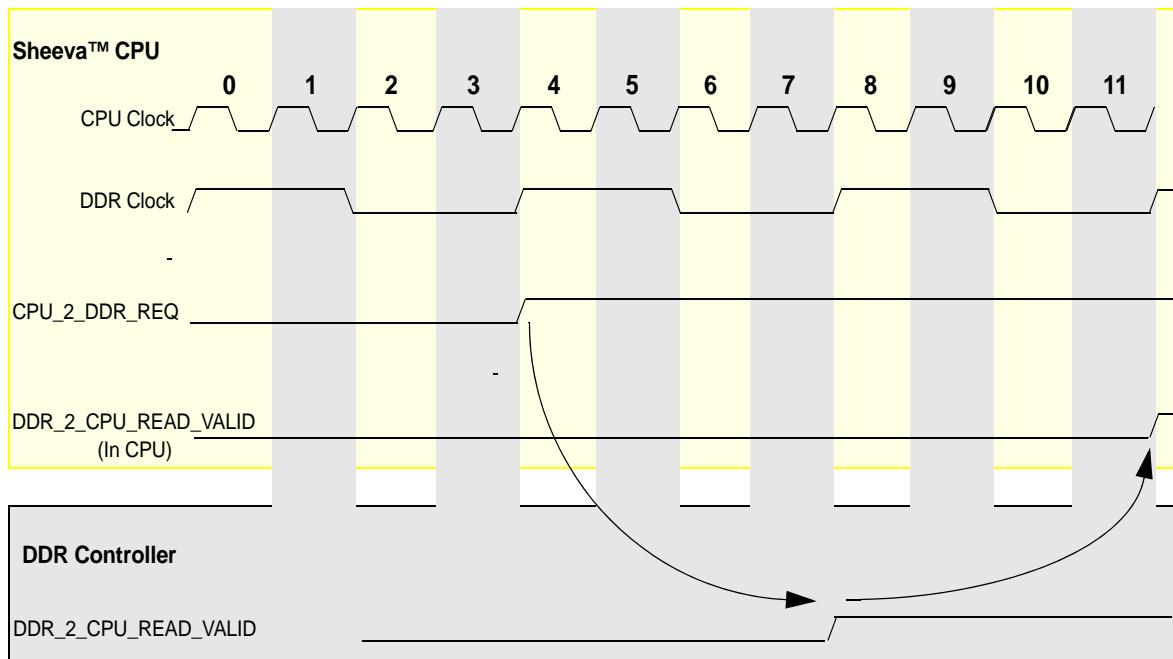
- DDR controller provides two read/write transaction buffers for Mbus-L transactions:
  - Supports multiple outstanding transactions
  - Enables back-to-back transactions on DDR memory
- Mbus-L to Mbus bridge provides two read/write transaction buffers:
  - Supports multiple outstanding transactions

### 25.1.2 DDR Latency

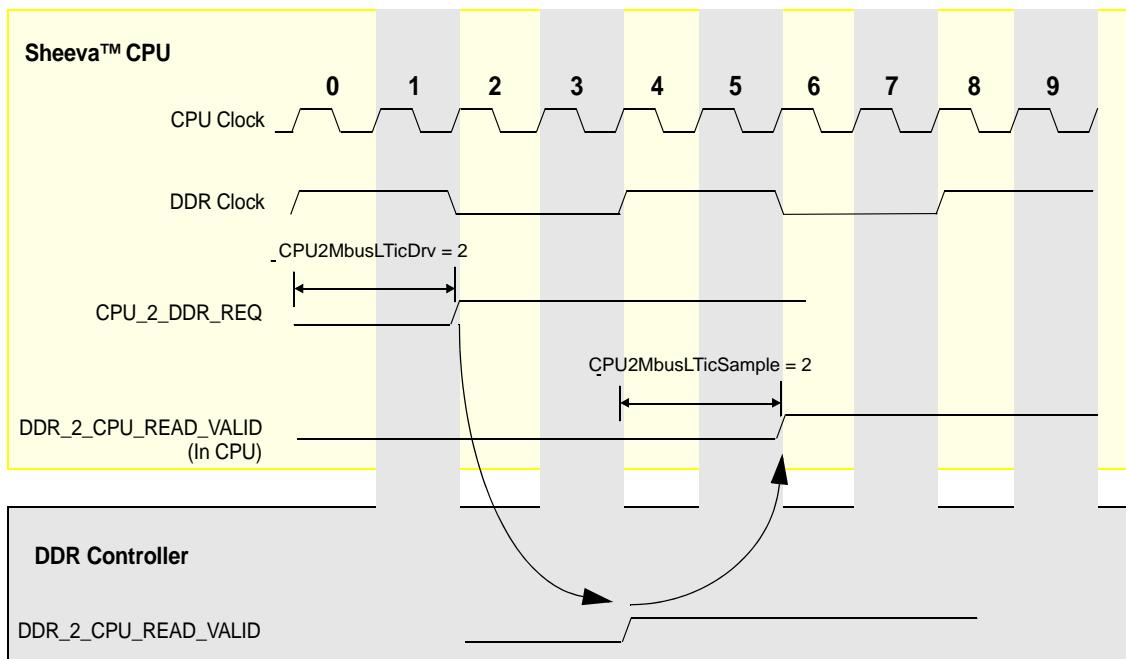
The CPU and DDR work in different clock domains. The Bus Interface Unit (BIU) executes in the same clock domain as the CPU. Since the DDR controller runs at the DDR clock frequency, the following mechanism is used to minimize the delay of the Mbus-L signals on the CPU–DDR interface:

- The CPU drives Mbus-L signals towards the DDR controller on a specific CPU clock rather than on the DDR clock. This specific CPU clock is selected from the <CPU2MbusLTickDrv> field in the CPU Configuration Register ([Table 115 p. 366](#)). For an example of this mechanism for the CPU-to-DDR clock period ratio of 1:4, see [Figure 86, CPU to DDR Mbus-L Timing Diagrams—CPU2MbusLTickDrv=0, CPU2MbusLTickSample=0, on page 314](#) and [Figure 87, CPU to DDR Mbus-L Timing Diagrams—CPU2MbusLTickDrv=2, CPU2MbusLTickSample=2, on page 314](#).
- The CPU samples Mbus-L signals from the DDR controller on a specific CPU clock rather than on the DDR clock. The specific CPU clock is selected by the <CPU2MbusLTickSample> field in the CPU Configuration Register ([Table 115 p. 367](#)). See [Figure 86](#) and [Figure 87](#) for an example of this mechanism for the CPU-to-DDR clock period ratio of 1:4.

**Figure 86: CPU to DDR Mbus-L Timing Diagrams—CPU2MbusLTickDrv=0,  
CPU2MbusLTickSample=0**



**Figure 87: CPU to DDR Mbus-L Timing Diagrams—CPU2MbusLTickDrv=2,  
CPU2MbusLTickSample=2**



## 25.2

## Mbus—Device Internal Bus

The Mbus is a 64-bit internal bus. It is used for data transfer between the different units (except for the CPU access to DDR SDRAM). The different units can act as masters on the bus generating requests, or as targets driving read responses. [Table 86](#) lists the units connected through the Mbus and indicates the functions implemented by each unit. The Mbus runs at TCLK.

**Table 86: Mbus Units**

Unit	Unit ID	Function
DDR SDRAM controller	0x0	Target
TWSI, UART, NAND flash, SPI, RTC, GPIO, and BootROM	0x1	Master/Target
Mbus-L to Mbus bridge	0x2	Master/Target
Cryptographic engines and Security accelerator	0x3	Master/Target
PCI Express port	0x4	Master/Target
USB 2.0 port	0x5	Master/Target
XOR and DMA unit	0x6	Master/Target
Gigabit Ethernet port(s)	0x7	Master/Target
SATA ports (88F619x and 88F6281 only)	0x8	Master/Target
SDIO port	0x9	Master/Target
Audio port (88F6180, 88F6192, and 88F6281 only)	0xA	Master/Target
MPEG Transport Stream port (88F6192 and 88F6281 only)	0xB	Master/Target
TDM ports(88F6192 and 88F6281 only)	0xD	Master/Target



**Note**

- The TWSI, UART, NAND flash, SPI, BootROM interfaces act as targets. The TWSI functions as a master only after reset, when using TWSI serial ROM initialization.
- Only the target interfaces PCI Express, NAND flash, SPI and DDR are units that are targeted for direct access to an external interface. The rest of the target units are accessed for their internal registers to configure and operate the unit.

The Mbus uses a proprietary protocol. All read transactions are split transactions. This ensures that the bus does not remain busy while a slow memory unit is accessing the requested data. The bus supports up to 128B transfer per transaction.

### 25.2.1

### Mbus Arbitration

The DDR SDRAM interface Mbus port implements a programmable arbitration scheme to optimize the device performance, according to the system requirements. The arbitration priorities for the initiators can be adjusted through the registers in [Section A.4, DDR SDRAM Controller Registers, on page 389](#).

The DDR SDRAM controller further arbitrates between the winning Mbus transaction and Mbus-L requests from the CPU. For more details, see [Section 4.1.3, Arbitration and Ordering, on page 45](#).



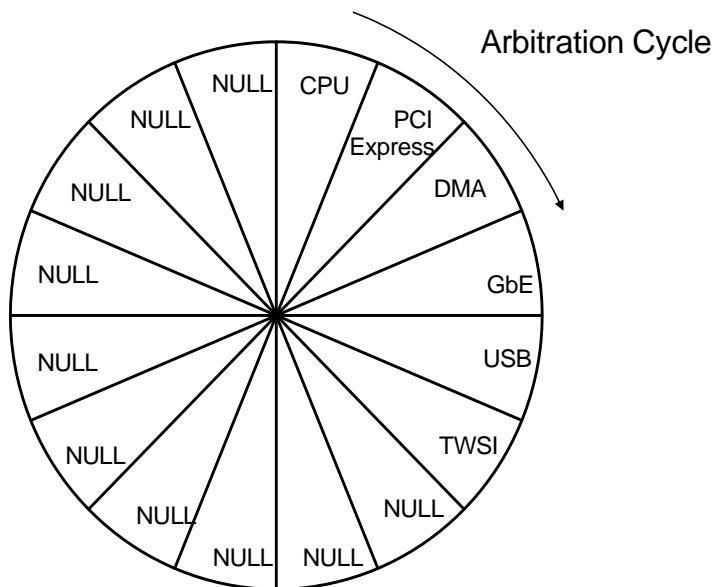
**Note**

Each of the other Mbus target units also has a dedicated arbiter. Those arbiters used a fixed arbitration scheme that cannot be programmed. A two-level arbitration scheme is used for those arbiters. All read responses target units are serviced first, followed by all initiator requests from master units. The arbitration between the initiator units operates in a fixed-round-robin fashion. The arbitration between target units also operates in a fixed-round-robin fashion. Each of these arbitration schemes operations in a fixed-round-robin fashion.

[Figure 88](#) depicts the arbitration in the form of a wheel with the wheel representing one clock cycle and each slice of the wheel representing a transaction on the Mbus. The arbitration scheme works as follows:

- The arbiter gives access to the initiator unit that has priority according to the current transaction slice.
- For each TCLK cycle, if the initiator unit that has priority in the current slice has not re-asserted a request, the arbiter will give the transaction to the initiator unit that has priority in the next slice. This will continue until an initiator unit is found that has asserted a request.

**Figure 88: Masters Request Default Arbitration Cycle**



The Mbus SDRAM arbiter priority scheme can be used to allocate a fair bandwidth to the different master units. The arbiter has 16 slices, each of which can be assigned to any unit. The arbiter works in a fixed-round-robin fashion, calculating at each available cycle which of the pending requests is the next to be served. The arbiter works on a transaction basis, meaning, it enters a new arbitration cycle, only when a transaction ends. This means that priority settings should be affected by the typical transaction size for each unit. If for example, the typical transaction size of unit A is twice the typical transaction size of unit B, unit B should get twice as many priority slices as unit A to have the same bandwidth allocation.

## 25.3 Mbus-L to Mbus Bridge

The CPU interfaces with the device units over the Mbus-L to Mbus bridge. This bridge forwards CPU transactions to the device units over the Mbus, and forwards read responses back from the units to the CPU. This bridge also integrates the device control and status registers related to the CPU.

### 25.3.1 Mbus-L to Mbus Bridge Features

- Unidirectional bridge—Only transactions from Mbus-L to Mbus are supported.
- Supports two outstanding read requests.
- Contains two read/write buffers of 32B.

The Mbus-L to Mbus bridge is configured using the registers listed in following sections:

- [Appendix A.3.1, CPU Address Map Registers, on page 357](#).
- [Appendix A.3.2, CPU Control and Status Registers, on page 365](#).
- [Appendix A.3.3, CPU Doorbell Registers, on page 377](#).
- [Appendix A.3.4, CPU Timers Registers, on page 378](#).
- [Appendix A.3.6, Main Interrupt Controller Registers, on page 383](#).

## 25.4 Transaction Ordering

The device supports the following ordering rules:

- CPU lock
- Read after write ordering
- Write after write ordering
- PCI Express bridge ordering rules
- Producer-consumer ordering

This section details these rules.

### 25.4.1 CPU Lock

The CPU supports Lock transactions. These transactions are useful for atomic read-modify-write commands.

When the CPU generates a lock read transaction to DRAM, no other DRAM transaction is served until the CPU performs an unlock transaction.

### 25.4.2 Read-after-Write and Write-after-Write Ordering

The implementation of the transaction queues, for each of the device master units, guarantees read-after-write and write-after-write ordering of transactions from the same originator to the same target.

However, the basic implementation of the transaction queues cannot guarantee ordering of transactions between different sources and destinations. For example, if the CPU generates two consecutive write transactions, the first one to the GbE MAC and the second one to the DRAM, there is no way for the hardware to guarantee that the write on the GbE MAC will be executed first.

## 25.4.3 PCI Express Bridge Ordering Rules

### PCI Express Host Hardware-Enforced Ordering



Note

The term “upstream” describes a data transfer from the PCI Express towards the CPU/DRAM.

The term “downstream” is used for data transfer from the CPU/DRAM to the PCI Express.

The device supports the PCI Express bridge transaction ordering rules. Especially, in cases of a CPU read from the PCI Express endpoint, the device drives the read response on the CPU bus only after flushing all upstream write data previously posted from the PCI Express into memory.

To enable PCI Express host hardware-enforced ordering, set both the [`<RxCmplPushDis>`](#) field and the [`<RxNpPushDis>`](#) field in the PCI Express TL Control Register ([Table 318 p. 483](#)) to 0 (the default settings).

### PCI Express Endpoint Hardware-Enforced Ordering

The device also supports the transactions ordering rules when it acts as an endpoint. Especially, when an external PCI Express host performs a read from the device local memory. The device only drives downstream read completion after all pending downstream posted write data is flushed.

To enable PCI Express endpoint hardware-enforced ordering, set the [`<TxCmplPushDis>`](#) field and the [`<TxNpPushDis>`](#) field in the PCI Express TL Control Register ([Table 318 p. 483](#)) to 0 (the default settings).

## 25.4.4 Producer-Consumer Ordering

With producer-consumer ordering, the producer can be the I/O device and the consumer can be the CPU, or the opposite, the producer is the CPU and the consumer is the I/O device).

The basic concept of a producer-consumer model operates as follows:

- The producer places some data in memory for the consumer to process.
- The producer notifies the consumer (via interrupt or any other means) that there is pending data in memory for the consumer to process.
- The consumer reads the data from memory and processes it.

The device implementation guarantees that this model works properly, meaning that it is guaranteed that when the consumer reads data from memory, it reads the valid data.

### Producer-Consumer Operation



Note

The following describes the producer-consumer operation between the CPU and the GbE SDMA. However, it also applies to all of the chip DMAs.

GbE MAC transmit operation consists of:

1. The CPU prepares transmit descriptors and buffers in memory.
2. The software flushes buffer descriptors from L1 and L2 caches to the DRAM, using cache operations. The software also performs drain write buffer operation, to guarantee that data is flushed all the way to memory.
3. Triggering the GbE SDMA (a write to GbE MAC register).
4. The GbE SDMA starts to fetch descriptors and buffers from the DRAM.

The DRAM controller implementation guarantees that the GbE SDMA reads the latest data from DRAM, and not some old invalid data (see DRAM Controller chapter for full details).

GbE MAC receive operation consists of:

1. The SDMA writes received packets to buffers in memory.
2. The SDMA writes the Rx descriptor status to memory. It then interrupts the CPU.
3. The CPU reads the interrupt cause register (identify interrupt cause).
4. The CPU reads the Rx data from memory.

The DRAM controller implementation guarantees that the CPU reads the latest data from DRAM, and not old invalid data (see DRAM Controller chapter for full details).



**88F6180/88F619x/88F6281**  
**Functional Specifications**

---

THIS PAGE IS INTENTIONALLY LEFT BLANK



# 88F6180/88F619x/88F6281

## Register Set



**88F6180/88F619x/88F6281**  
**Functional Specifications**

---

THIS PAGE IS INTENTIONALLY LEFT BLANK

# List of Registers

A.3 Mbus-L to Mbus Bridge Registers .....	355
Table 90: Window0 Control Register .....	357
Offset: 0x20000	
Table 91: Window0 Base Register .....	357
Offset: 0x20004	
Table 92: Window0 Remap Low Register .....	358
Offset: 0x20008	
Table 93: Window0 Remap High Register .....	358
Offset: 0x2000C	
Table 94: Window1 Control Register .....	358
Offset: 0x20010	
Table 95: Window1 Base Register .....	359
Offset: 0x20014	
Table 96: Window1 Remap Low Register .....	359
Offset: 0x20018	
Table 97: Window1 Remap High Register .....	359
Offset: 0x2001C	
Table 98: Window2 Control Register .....	359
Offset: 0x20020	
Table 99: Window2 Base Register .....	360
Offset: 0x20024	
Table 100: Window2 Remap Low Register .....	360
Offset: 0x20028	
Table 101: Window2 Remap High Register .....	360
Offset: 0x2002C	
Table 102: Window3 Control Register .....	361
Offset: 0x20030	
Table 103: Window3 Base Register .....	361
Offset: 0x20034	
Table 104: Window3 Remap Low Register .....	361
Offset: 0x20038	
Table 105: Window3 Remap High Register .....	362
Offset: 0x2003C	
Table 106: Window4 Control Register .....	362
Offset: 0x20040	
Table 107: Window4 Base Register .....	362
Offset: 0x20044	
Table 108: Window5 Control Register .....	363
Offset: 0x20050	
Table 109: Window5 Base Register .....	363
Offset: 0x20054	
Table 110: Window6 Control Register .....	363
Offset: 0x20060	
Table 111: Window6 Base Register .....	364
Offset: 0x20064	



Table 112: Window7 Control Register .....	364
Offset: 0x20070	
Table 113: Window7 Base Register .....	365
Offset: 0x20074	
Table 114: Device Internal Registers Base Address .....	365
Offset: 0x20080	
Table 115: CPU Configuration Register .....	365
Offset: 0x20100	
Table 116: CPU Control and Status Register .....	368
Offset: 0x20104	
Table 117: RSTOUTn Mask Register .....	368
Offset: 0x20108	
Table 118: System Soft Reset Register .....	369
Offset: 0x2010C	
Table 119: Mbus-L to Mbus Bridge Interrupt Cause Register .....	369
Offset: 0x20110	
Table 120: Mbus-L to Mbus Bridge Interrupt Mask Register .....	370
Offset: 0x20114	
Table 121: Memory Power Management Control Register .....	370
Offset: 0x20118	
Table 122: Clock Gating Control Register .....	371
Offset: 0x2011C	
Table 123: BIU Configuration Register .....	373
Offset: 0x20120	
Table 124: CPU L2 Configuration Register .....	374
Offset: 0x20128	
Table 125: L2 RAM Timing 0 Register .....	375
Offset: 0x20134	
Table 126: L2 RAM Timing 1 Register .....	375
Offset: 0x20138	
Table 127: L2 RAM Power Management Control Register .....	376
Offset: 0x20144	
Table 128: CPU RAM Management Control0 Register .....	376
Offset: 0x20148	
Table 129: CPU RAM Management Control1 Register .....	376
Offset: 0x2014C	
Table 130: CPU RAM Management Control2 Register .....	377
Offset: 0x20150	
Table 131: CPU RAM Management Control3 Register .....	377
Offset: 0x20154	
Table 132: Host-to-CPU Doorbell Register .....	377
Offset: 0x20400	
Table 133: Host-to-CPU Doorbell Mask Register .....	378
Offset: 0x20404	
Table 134: CPU-to-Host Doorbell Register .....	378
Offset: 0x20408	
Table 135: CPU-to-Host Doorbell Mask Register .....	378
Offset: 0x2040C	
Table 136: CPU Timers Control Register .....	378
Offset: 0x20300	
Table 137: CPU Timer0 Reload Register .....	379
Offset: 0x20310	

---

Table 138: CPU Timer 0 Register .....	380
Offset: 0x20314	
Table 139: CPU Timer1 Reload Register .....	380
Offset: 0x20318	
Table 140: CPU Timer 1 Register .....	380
Offset: 0x2031C	
Table 141: CPU Watchdog Timer Reload Register .....	380
Offset: 0x20320	
Table 142: CPU Watchdog Timer Register .....	381
Offset: 0x20324	
Table 143: Window0 Base Address Register .....	381
Offset: 0x20A00	
Table 144: Window0 Size Address Register .....	381
Offset: 0x20A04	
Table 145: Window1 Base Address Register .....	381
Offset: 0x20A08	
Table 146: Window1 Size Address Register .....	382
Offset: 0x20A0C	
Table 147: Window2 Base Address Register .....	382
Offset: 0x20A10	
Table 148: Window2 Size Address Register .....	382
Offset: 0x20A14	
Table 149: Window3 Base Address Register .....	383
Offset: 0x20A18	
Table 150: Window3 Size Address Register .....	383
Offset: 0x20A1C	
Table 151: Main Interrupt Cause Low Register .....	383
Offset: 0x20200	
Table 152: Main IRQ Interrupt Mask Low Register .....	385
Offset: 0x20204	
Table 153: Main FIQ Interrupt Mask Low Register .....	386
Offset: 0x20208	
Table 154: Endpoint Interrupt Mask Low Register .....	386
Offset: 0x2020C	
Table 155: Main Interrupt Cause High Register .....	386
Offset: 0x20210	
Table 156: Main IRQ Interrupt Mask High Register .....	387
Offset: 0x20214	
Table 157: Main FIQ Interrupt Mask High Register .....	388
Offset: 0x20218	
Table 158: Endpoint Interrupt Mask High Register .....	388
Offset: 0x2021C	

#### A.4 DDR SDRAM Controller Registers ..... 389

Table 160: CPU CS Window0 Base Address Register .....	390
Offset: 0x01500	
Table 161: CPU CS Window0 Size Register .....	390
Offset: 0x01504	
Table 162: CPU CS Window1 Base Address Register .....	390
Offset: 0x01508	
Table 163: CPU CS Window1 Size Register .....	391
Offset: 0x0150C	



Table 164: CPU CS Window2 Base Address Register .....	391
Offset: 0x01510	
Table 165: CPU CS Window2 Size Register .....	392
Offset: 0x01514	
Table 166: CPU CS Window3 Base Address Register .....	392
Offset: 0x01518	
Table 167: CPU CS Window3 Size Register .....	392
Offset: 0x0151C	
Table 168: SDRAM Configuration Register .....	393
Offset: 0x01400	
Table 169: DDR Controller Control (Low) Register .....	394
Offset: 0x01404	
Table 170: SDRAM Timing (Low) Register .....	397
Offset: 0x01408	
Table 171: SDRAM Timing (High) Register .....	397
Offset: 0x0140C	
Table 172: SDRAM Address Control Register .....	398
Offset: 0x01410	
Table 173: SDRAM Open Pages Control Register .....	400
Offset: 0x01414	
Table 174: SDRAM Operation Register .....	400
Offset: 0x01418	
Table 175: SDRAM Mode Register .....	401
Offset: 0x0141C	
Table 176: Extended DRAM Mode Register .....	402
Offset: 0x01420	
Table 177: DDR Controller Control (High) Register .....	403
Offset: 0x01424	
Table 178: DDR2 SDRAM Timing (Low) Register .....	404
Offset: 0x01428	
Table 179: SDRAM Operation Control Register .....	404
Offset: 0x0142C	
Table 180: SDRAM Interface Mbus Control (Low) Register .....	404
Offset: 0x01430	
Table 181: SDRAM Interface Mbus Control (High) Register .....	405
Offset: 0x01434	
Table 182: SDRAM Interface Mbus Timeout Register .....	406
Offset: 0x01438	
Table 183: DDR2 SDRAM Timing (High) Register .....	406
Offset: 0x0147C	
Table 184: SDRAM Initialization Control Register .....	407
Offset: 0x01480	
Table 185: Extended DRAM Mode 2 Register .....	407
Offset: 0x0148C	
Table 186: Extended DRAM Mode 3 Register .....	407
Offset: 0x01490	
Table 187: SDRAM ODT Control (Low) Register .....	407
Offset: 0x01494	
Table 188: SDRAM ODT Control (High) Register .....	408
Offset: 0x01498	
Table 189: DDR Controller ODT Control Register .....	409
Offset: 0x0149C	

---

Table 190: Read Buffer Select Register .....	410
Offset: 0x014A4	
Table 191: DDR SDRAM Address/Control Pads Calibration Register .....	410
Offset: 0x014C0	
Table 192: DDR SDRAM DQ Pads Calibration Register .....	411
Offset: 0x014C4	
Table 193: DDR SDRAM DQS Pads Calibration Register .....	412
Offset: 0x014C8	

## A.5 Time Division Multiplexing (TDM) Unit Registers..... 413

Table 195: CSU System Clock Prescaler Register .....	414
Offset: 0xD3100	
Table 196: CSU Global Control Register .....	415
Offset: 0xD3104	
Table 197: SPI Control Register .....	415
Offset: 0xD3108	
Table 198: Codec Access Command Low Register .....	415
Offset: 0xD3130	
Table 199: Codec Access Command High Register .....	416
Offset: 0xD3134	
Table 200: Codec Registers Access Control .....	416
Offset: 0xD3138	
Table 201: Codec Read Data Register .....	417
Offset: 0xD313C	
Table 202: Codec Registers Access Control1 .....	417
Offset: 0xD3140	
Table 203: PCM Control Register .....	418
Offset: 0xD0000	
Table 204: Channel Time Slot Control Register .....	420
Offset: 0xD0004	
Table 205: Channel 0 Delay Control Register .....	421
Offset: 0xD0008	
Table 206: Channel 1 Delay Control Register .....	422
Offset: 0xD000C	
Table 207: Channel 0/1 Enable and Disable Register (n=0–1) .....	422
Offset: Channel0: 0xD0010, Channel1: 0xD0020	
Table 208: Channel 0/1 Buffer Ownership Register (n=0–1) .....	423
Offset: Channel0: 0xD0014, Channel1: 0xD0024	
Table 209: Channel 0 Transmit Data Start Address Register .....	423
Offset: 0xD0018	
Table 210: Channel 0 Receive Data Start Address Register .....	424
Offset: 0xD001C	
Table 211: Channel 1 Transmit Data Start Address Register .....	424
Offset: 0xD0028	
Table 212: Channel 1 Receive Data Start Address Register .....	424
Offset: 0xD002C	
Table 213: Channel 0/1 Total Sample Count Register (n=0–1) .....	425
Offset: Channel0: 0xD0030, Channel1: 0xD0034	
Table 214: Number of Time Slots Register .....	425
Offset: 0xD0038	
Table 215: TDM PCM Clock Rate Divisor Register .....	426
Offset: 0xD003C	



Table 216: Interrupt Event Mask Register .....	426
Offset: 0xD0040	
Table 217: Interrupt Status Mask Register .....	426
Offset: 0xD0048	
Table 218: Interrupt Reset Selection Register .....	427
Offset: 0xD004C	
Table 219: Interrupt Status Register .....	427
Offset: 0xD0050	
Table 220: Dummy Data for Dummy RX Write Register .....	428
Offset: 0xD0054	
Table 221: Miscellaneous Control Register .....	429
Offset: 0xD0058	
Table 222: Channel 0/1 Transmit Data Current Address Register (n=0–1) .....	429
Offset: Channel0: 0xD0060, Channel1: 0xD0068	
Table 223: Channel 0/1 Receive Data Current Address Register (n=0–1) .....	429
Offset: Channel0: 0xD0064, Channel1: 0xD006C	
Table 224: Current Time Slot Register .....	429
Offset: 0xD0070	
Table 225: TDM Revision Register .....	429
Offset: 0xD0074	
Table 226: TDM Channel 0/1 Debug Register (n=0–1) .....	430
Offset: Channel0: 0xD0078, Channel1: 0xD007C	
Table 227: TDM DMA Abort Register 1 .....	431
Offset: 0xD0080	
Table 228: TDM DMA Abort Register 2 .....	431
Offset: 0xD0084	
Table 229: TDM Channel0 Wideband Delay Control Register .....	431
Offset: 0xD0088	
Table 230: TDM Channel 1 Wideband Delay Control Register .....	432
Offset: 0xD008C	
Table 231: SPI Interface Output Enable Control Register .....	432
Offset: 0xD4000	
Table 232: TDM-Mbus Configuration Register .....	433
Offset: 0xD4010	
Table 233: Window0 Control Register .....	433
Offset: 0xD4030	
Table 234: Window0 Base Register .....	433
Offset: 0xD4034	
Table 235: Window1 Control Register .....	434
Offset: 0xD4040	
Table 236: Window1 Base Register .....	434
Offset: 0xD4044	
Table 237: Window2 Control Register .....	434
Offset: 0xD4050	
Table 238: Window2 Base Register .....	435
Offset: 0xD4054	
Table 239: Window3 Control Register .....	435
Offset: 0xD4060	
Table 240: Window3 Base Register .....	436
Offset: 0xD4064	
Table 241: TDM-Mbus Configuration 1 Register .....	436
Offset: 0xD4070	

---

<b>A.6 PCI Express Interface Registers .....</b>	<b>437</b>
Table 243: PCI Express Window0 Control Register .....	439
Offset: 0x41820	
Table 244: PCI Express Window0 Base Register .....	440
Offset: 0x41824	
Table 245: PCI Express Window0 Remap Register .....	440
Offset: 0x4182C	
Table 246: PCI Express Window1 Control Register .....	441
Offset: 0x41830	
Table 247: PCI Express Window1 Base Register .....	441
Offset: 0x41834	
Table 248: PCI Express Window1 Remap Register .....	442
Offset: 0x4183C	
Table 249: PCI Express Window2 Control Register .....	442
Offset: 0x41840	
Table 250: PCI Express Window2 Base Register .....	443
Offset: 0x41844	
Table 251: PCI Express Window2 Remap Register .....	443
Offset: 0x4184C	
Table 252: PCI Express Window3 Control Register .....	443
Offset: 0x41850	
Table 253: PCI Express Window3 Base Register .....	444
Offset: 0x41854	
Table 254: PCI Express Window3 Remap Register .....	444
Offset: 0x4185C	
Table 255: PCI Express Window4 Control Register .....	444
Offset: 0x41860	
Table 256: PCI Express Window4 Base Register .....	445
Offset: 0x41864	
Table 257: PCI Express Window4 Remap Register .....	445
Offset: 0x4186C	
Table 258: PCI Express Window4 Remap (High) Register .....	446
Offset: 0x41870	
Table 259: PCI Express Window5 Control Register .....	446
Offset: 0x41880	
Table 260: PCI Express Window5 Base Register .....	447
Offset: 0x41884	
Table 261: PCI Express Window5 Remap Register .....	447
Offset: 0x4188C	
Table 262: PCI Express Window5 Remap (High) Register .....	447
Offset: 0x41890	
Table 263: PCI Express Default Window Control Register .....	447
Offset: 0x418B0	
Table 264: PCI Express Expansion ROM Window Control Register .....	448
Offset: 0x418C0	
Table 265: PCI Express Expansion ROM Window Remap Register .....	448
Offset: 0x418C4	
Table 266: PCI Express BAR1 Control Register .....	449
Offset: 0x41804	
Table 267: PCI Express BAR2 Control Register .....	449
Offset: 0x41808	



Table 268: PCI Express Expansion ROM BAR Control Register .....	449
Offset: 0x4180C	
Table 269: PCI Express Configuration Address Register .....	450
Offset: 0x418F8	
Table 270: PCI Express Configuration Data Register .....	450
Offset: 0x418FC	
Table 271: PCI Express Device and Vendor ID Register .....	451
Offset: 0x40000	
Table 272: PCI Express Command and Status Register .....	451
Offset: 0x40004	
Table 273: PCI Express Class Code and Revision ID Register .....	453
Offset: 0x40008	
Table 274: PCI Express BIST Header Type and Cache Line Size Register .....	454
Offset: 0x4000C	
Table 275: PCI Express BAR0 Internal Register .....	454
Offset: 0x40010	
Table 276: PCI Express BAR0 Internal (High) Register .....	455
Offset: 0x40014	
Table 277: PCI Express BAR1 Register .....	455
Offset: 0x40018	
Table 278: PCI Express BAR1 (High) Register .....	455
Offset: 0x4001C	
Table 279: PCI Express BAR2 Register .....	455
Offset: 0x40020	
Table 280: PCI Express BAR2 (High) Register .....	456
Offset: 0x40024	
Table 281: PCI Express Subsystem Device and Vendor ID Register .....	456
Offset: 0x4002C	
Table 282: PCI Express Expansion ROM BAR Register .....	457
Offset: 0x40030	
Table 283: PCI Express Capability List Pointer Register .....	457
Offset: 0x40034	
Table 284: PCI Express Interrupt Pin and Line Register .....	457
Offset: 0x4003C	
Table 285: PCI Express Power Management Capability Header Register .....	458
Offset: 0x40040	
Table 286: PCI Express Power Management Control and Status Register .....	459
Offset: 0x40044	
Table 287: PCI Express MSI Message Control Register .....	460
Offset: 0x40050	
Table 288: PCI Express MSI Message Address Register .....	461
Offset: 0x40054	
Table 289: PCI Express MSI Message Address (High) Register .....	461
Offset: 0x40058	
Table 290: PCI Express MSI Message Data Register .....	461
Offset: 0x4005C	
Table 291: PCI Express Capability Register .....	461
Offset: 0x40060	
Table 292: PCI Express Device Capabilities Register .....	462
Offset: 0x40064	
Table 293: PCI Express Device Control Status Register .....	463
Offset: 0x40068	

---

Table 294: PCI Express Link Capabilities Register .....	465
Offset: 0x4006C	
Table 295: PCI Express Link Control Status Register .....	466
Offset: 0x40070	
Table 296: PCI Express Advanced Error Report Header Register .....	468
Offset: 0x40100	
Table 297: PCI Express Uncorrectable Error Status Register .....	468
Offset: 0x40104	
Table 298: PCI Express Uncorrectable Error Mask Register .....	469
Offset: 0x40108	
Table 299: PCI Express Uncorrectable Error Severity Register .....	470
Offset: 0x4010C	
Table 300: PCI Express Correctable Error Status Register .....	471
Offset: 0x40110	
Table 301: PCI Express Correctable Error Mask Register .....	472
Offset: 0x40114	
Table 302: PCI Express Advanced Error Capability and Control Register .....	473
Offset: 0x40118	
Table 303: PCI Express Header Log First DWORD Register .....	473
Offset: 0x4011C	
Table 304: PCI Express Header Log Second DWORD Register .....	474
Offset: 0x40120	
Table 305: PCI Express Header Log Third DWORD Register .....	474
Offset: 0x40124	
Table 306: PCI Express Header Log Fourth DWORD Register .....	474
Offset: 0x40128	
Table 307: PCI Express Control Register .....	474
Offset: 0x41A00	
Table 308: PCI Express Status Register .....	476
Offset: 0x41A04	
Table 309: PCI Express Boot Address Register .....	477
Offset: 0x41A08	
Table 310: PCI Express Root Complex Set Slot Power Limit Register .....	477
Offset: 0x41A0C	
Table 311: PCI Express Completion Timeout Register .....	478
Offset: 0x41A10	
Table 312: PCI Express Root Complex Power Management Event Register .....	478
Offset: 0x41A14	
Table 313: PCI Express Power Management Extended Register .....	479
Offset: 0x41A18	
Table 314: PCI Express Flow Control Register .....	480
Offset: 0x41A20	
Table 315: PCI Express Acknowledge Timers (1X) Register .....	480
Offset: 0x41A40	
Table 316: PCI Express RAM Parity Protection Control Register .....	480
Offset: 0x41A50	
Table 317: PCI Express Debug Control Register .....	481
Offset: 0x41A60	
Table 318: PCI Express TL Control Register .....	483
Offset: 0x41AB0	
Table 319: PCI Express PHY Indirect Access Register .....	484
Offset: 0x41B00	



Table 320: PCI Express Interrupt Cause Register .....	484
Offset: 0x41900	
Table 321: PCI Express Interrupt Mask Register .....	487
Offset: 0x41910	
Table 322: PCI Express Mbus Adapter Control Register .....	488
Offset: 0x418D0	
Table 323: PCI Express Mbus Arbiter Control Register (Low) .....	489
Offset: 0x418E0	
Table 324: PCI Express Mbus Arbiter Control Register (High) .....	489
Offset: 0x418E4	
Table 325: PCI Express Mbus Arbiter Timeout Register .....	490
Offset: 0x418E8	

## A.7 Serial-ATA Host Controller (SATAHC) Registers ..... 491

Table 327: Basic DMA Command Register .....	494
Offset: Port0: 0x82224 Port1: 0x84224	
Table 328: Basic DMA Status Register .....	495
Offset: Port0: 0x82228 Port1: 0x84228	
Table 329: Descriptor Table Low Base Address Register .....	496
Offset: Port0: 0x8222C Port1: 0x8422C	
Table 330: Descriptor Table High Base Address Register .....	497
Offset: Port0: 0x82230 Port1: 0x84230	
Table 331: Data Region Low Address Register .....	497
Offset: Port0: 0x82234 Port1: 0x84234	
Table 332: Data Region High Address Register .....	497
Offset: Port0: 0x82238 Port1: 0x84238	
Table 333: EDMA Configuration Register .....	498
Offset: Port0: 0x82000 Port1: 0x84000	
Table 334: EDMA Interrupt Error Cause Register .....	501
Offset: Port0: 0x82008 Port1: 0x84008	
Table 335: EDMA Interrupt Error Mask Register .....	503
Offset: Port0: 0x8200C Port1: 0x8400C	
Table 336: EDMA Request Queue Base Address High Register .....	503
Offset: Port0: 0x82010 Port1: 0x84010	
Table 337: EDMA Request Queue In-Pointer Register .....	504
Offset: Port0: 0x82014 Port1: 0x84014	
Table 338: EDMA Request Queue Out-Pointer Register .....	504
Offset: Port0: 0x82018 Port1: 0x84018	
Table 339: EDMA Response Queue Base Address High Register .....	504
Offset: Port0: 0x8201C Port1: 0x8401C	
Table 340: EDMA Response Queue In-Pointer Register .....	504
Offset: Port0: 0x82020 Port1: 0x84020	
Table 341: EDMA Response Queue Out-Pointer Register .....	505
Offset: Port0: 0x82024 Port1: 0x84024	
Table 342: EDMA Command Register .....	505
Offset: Port0: 0x82028 Port1: 0x84028	
Table 343: EDMA Status Register .....	506
Offset: Port0: 0x82030 Port1: 0x84030	
Table 344: EDMA IORdy Timeout Register .....	508
Offset: Port0: 0x82034 Port1: 0x84034	
Table 345: EDMA Command Delay Threshold Register .....	508
Offset: Port0: 0x82040 Port1: 0x84040	

---

Table 346: EDMA Halt Conditions Register .....	508
Offset: Port0: 0x82060 Port1: 0x84060	
Table 347: EDMA NCQ0 Done/TCQ0 Outstanding Status Register .....	509
Offset: Port0: 0x82094 Port1: 0x84094	
Table 348: SATAHC Configuration Register .....	509
Offset: 0x80000	
Table 349: SATAHC Request Queue Out-Pointer Register .....	510
Offset: 0x80004	
Table 350: SATAHC Response Queue In-Pointer Register .....	510
Offset: 0x80008	
Table 351: SATAHC Interrupt Coalescing Threshold Register .....	511
Offset: 0x8000C	
Table 352: SATAHC Interrupt Time Threshold Register .....	511
Offset: 0x80010	
Table 353: SATAHC Interrupt Cause Register .....	512
Offset: 0x80014	
Table 354: SATAHC Main Interrupt Cause Register .....	513
Offset: 0x80020	
Table 355: SATAHC Main Interrupt Mask Register .....	514
Offset: 0x80024	
Table 356: SATAHC LED Configuration Register .....	515
Offset: 0x8002C	
Table 357: Window0 Control Register .....	515
Offset: 0x80030	
Table 358: Window0 Base Register .....	516
Offset: 0x80034	
Table 359: Window1 Control Register .....	516
Offset: 0x80040	
Table 360: Window1 Base Register .....	517
Offset: 0x80044	
Table 361: Window2 Control Register .....	517
Offset: 0x80050	
Table 362: Window2 Base Register .....	517
Offset: 0x80054	
Table 363: Window3 Control Register .....	518
Offset: 0x80060	
Table 364: Window3 Base Register .....	518
Offset: 0x80064	
Table 365: Serial-ATA Interface Configuration Register .....	518
Offset: Port0: 0x82050 Port1: 0x84050	
Table 366: Serial-ATA PLL Configuration Register .....	521
Offset: Port0: 0x82054 Port1: 0x84054	
Table 367: SStatus Register .....	522
Offset: Port0: 0x82300 Port1: 0x84300	
Table 368: SError Register .....	523
Offset: Port0: 0x82304 Port1: 0x84304	
Table 369: SControl Register .....	525
Offset: Port0: 0x82308 Port1: 0x84308	
Table 370: LTMode Register .....	526
Offset: Port0: 0x8230C Port1: 0x8430C	
Table 371: PHY Mode 3 Register .....	527
Offset: Port0: 0x82310 Port1: 0x84310	



Table 372: PHY Mode 4 Register .....	529
Offset: Port0: 0x82314 Port1: 0x84314	
Table 373: PHY Mode 1 Register .....	531
Offset: Port0: 0x8232C Port1: 0x8432C	
Table 374: PHY Mode 2 Register .....	532
Offset: Port0: 0x82330 Port1: 0x84330	
Table 375: BIST Control Register .....	534
Offset: Port0: 0x82334 Port1: 0x84334	
Table 376: BIST-Dword1 Register .....	534
Offset: Port0: 0x82338 Port1: 0x84338	
Table 377: BIST-Dword2 Register .....	535
Offset: Port0: 0x8233C Port1: 0x8433C	
Table 378: SError Interrupt Mask Register .....	535
Offset: Port0: 0x82340 Port1: 0x84340	
Table 379: Serial-ATA Interface Control Register .....	535
Offset: Port0: 0x82344 Port1: 0x84344	
Table 380: Serial-ATA Interface Test Control Register .....	537
Offset: Port0: 0x82348 Port1: 0x84348	
Table 381: Serial-ATA Interface Status Register .....	538
Offset: Port0: 0x8234C Port1: 0x8434C	
Table 382: Vendor Unique Register .....	540
Offset: Port0: 0x8235C Port1: 0x8435C	
Table 383: FIS Configuration Register .....	540
Offset: Port0: 0x82360 Port1: 0x84360	
Table 384: FIS Interrupt Cause Register .....	541
Offset: Port0: 0x82364 Port1: 0x84364	
Table 385: FIS Interrupt Mask Register .....	543
Offset: Port0: 0x82368 Port1: 0x84368	
Table 386: FIS Dword0 Register .....	543
Offset: Port0: 0x82370 Port1: 0x84370	
Table 387: FIS Dword1 Register .....	543
Offset: Port0: 0x82374 Port1: 0x84374	
Table 388: FIS Dword2 Register .....	543
Offset: Port0: 0x82378 Port1: 0x84378	
Table 389: FIS Dword3 Register .....	543
Offset: Port0: 0x8237C Port1: 0x8437C	
Table 390: FIS Dword4 Register .....	544
Offset: Port0: 0x82380 Port1: 0x84380	
Table 391: FIS Dword5 Register .....	544
Offset: Port0: 0x82384 Port1: 0x84384	
Table 392: FIS Dword6 Register .....	544
Offset: Port0: 0x82388 Port1: 0x84388	
Table 393: PHYMODE9_GEN2 Register .....	544
Offset: Port0: 0x82398 Port1: 0x84398	
Table 394: PHYMODE9_GEN1 Register .....	545
Offset: Port0: 0x8239C Port1: 0x8439C	
Table 395: PHY Configuration Register .....	546
Offset: Port0: 0x823A0 Port1: 0x843A0	
Table 396: PHYTCTL Register .....	547
Offset: Port0: 0x823A4 Port1: 0x843A4	
Table 397: PHY Mode 10 Register .....	547
Offset: Port0: 0x823A8 Port1: 0x843A8	

---

Table 398: PHY Mode 12 Register .....	548
Offset: Port0: 0x823B0 Port1: 0x843B0	

## A.8 Gigabit Ethernet Controller Registers ..... 550

Table 401: PHY Address Register .....	554
Offset: Port0: 0x72000 Port1: 0x76000	
Table 402: SMI Register .....	554
Offset: Port0: 0x72004 Port1: 0x76004	
Table 403: Ethernet Unit Default Address (EUDA) Register .....	555
Offset: Port0: 0x72008 Port1: 0x76008	
Table 404: Ethernet Unit Default ID (EUDID) Register .....	555
Offset: Port0: 0x7200C Port1: 0x7600C	
Table 405: Ethernet Unit Interrupt Cause (EUIC) Register .....	556
Offset: Port0: 0x72080 Port1: 0x76080	
Table 406: Ethernet Unit Interrupt Mask (EUIM) Register .....	557
Offset: Port0: 0x72084 Port1: 0x76084	
Table 407: Ethernet Unit Error Address (EUEA) Register .....	557
Offset: Port0: 0x72094 Port1: 0x76094	
Table 408: Ethernet Unit Internal Address Error (EUIAE) Register .....	557
Offset: Port0: 0x72098 Port1: 0x76098	
Table 409: Ethernet Unit Control (EUC) Register .....	557
Offset: Port0: 0x720B0 Port1: 0x760B0	
Table 410: Base Address Register (n=0–5) .....	558
Offset: Port0: BA0: 0x72200, BA1: 0x72208, BA2: 0x72210, BA3: 0x72218, BA4: 0x72220, BA5: 0x72228 Port1: BA0: 0x76200, BA1: 0x76208, BA2: 0x76210, BA3: 0x76218, BA4: 0x76220, BA5: 0x76228	
Table 411: Size (S) Register (n=0–5) .....	559
Offset: Port0: SR0: 0x72204, SR1: 0x7220C, SR2: 0x72214, SR3: 0x7221C, SR4: 0x72224, SR5: 0x7222C Port1: SR0: 0x76204, SR1: 0x7620C, SR2: 0x76214, SR3: 0x7621C, SR4: 0x76224, SR5: 0x7622C	
Table 412: High Address Remap (HA)1 Register (n=0–3) .....	559
Offset: Port0: HARR0: 0x72280, HARR1: 0x72284, HARR2: 0x72288, HARR3: 0x7228C Port1: HARR0: 0x76280, HARR1: 0x76284, HARR2: 0x76288, HARR3: 0x7628C	
Table 413: Base Address Enable (BARE) Register .....	559
Offset: Port0: 0x72290 Port1: 0x76290	
Table 414: Ethernet Port Access Protect (EPAP) Register .....	560
Offset: Port0: 0x72294 Port1: 0x76294	
Table 415: Port0/1 Mbus Top Arbiter Register .....	560
Offset: Port0: 0xE20C0 Port1: 0xE60C0	
Table 416: Port Configuration (PxC) Register .....	561
Offset: Port0: 0x72400 Port1: 0x76400	
Table 417: Port Configuration Extend (PxCX) Register .....	562
Offset: Port0: 0x72404 Port1: 0x76404	
Table 418: MII Serial Parameters Register .....	562
Offset: Port0: 0x72408 Port1: 0x76408	
Table 419: VLAN EtherType (EVLANE) Register .....	563
Offset: Port0: 0x72410 Port1: 0x76410	
Table 420: MAC Address Low (MACAL) Register .....	563
Offset: Port0: 0x72414 Port1: 0x76414	
Table 421: MAC Address High (MACAH) Register .....	563
Offset: Port0: 0x72418 Port1: 0x76418	
Table 422: SDMA Configuration (SDC) Register .....	564
Offset: Port0: 0x7241C Port1: 0x7641C	



Table 423: IP Differentiated Services CodePoint 0 to Priority (DSCP0) Register .....	565
Offset: Port0: 0x72420 Port1: 0x76420	
Table 424: IP Differentiated Services CodePoint 1 to Priority (DSCP1) Register .....	565
Offset: Port0: 0x72424 Port1: 0x76424	
Table 425: IP Differentiated Services CodePoint 2 to Priority (DSCP2/3/4/5) Register (n=0–3) .....	566
Offset: Port0: DSCP0: 0x72428, DSCP1: 0x7242C, DSCP2: 0x72430, DSCP3: 0x72434	
Port1: DSCP0: 0x76428, DSCP1: 0x7642C, DSCP2: 0x76430, DSCP3: 0x76434	
Table 426: IP Differentiated Services CodePoint 6 to Priority (DSCP6) Register .....	566
Offset: Port0: 0x72438 Port1: 0x76438	
Table 427: Port Serial Control0 (PSC0) Register .....	566
Offset: Port0: 0x7243C Port1: 0x7643C	
Table 428: VLAN Priority Tag to Priority (VPT2P) Register .....	569
Offset: Port0: 0x72440 Port1: 0x76440	
Table 429: Ethernet Port Status 0 (PS0) Register .....	569
Offset: Port0: 0x72444 Port1: 0x76444	
Table 430: Port Serial Control1 (PSC1) Register .....	571
Offset: Port0: 0x7244C Port1: 0x7644C	
Table 431: Ethernet Port Status1 (PS1) Register .....	573
Offset: Port0: 0x72450 Port1: 0x76450	
Table 432: Marvell Header Register .....	575
Offset: Port0: 0x72454 Port1: 0x76454	
Table 433: Port Interrupt Cause (IC) Register .....	577
Offset: Port0: 0x72460 Port1: 0x76460	
Table 434: Port Interrupt Cause Extend (ICE) Register .....	580
Offset: Port0: 0x72464 Port1: 0x76464	
Table 435: Port Interrupt Mask (PIM) Register .....	582
Offset: Port0: 0x72468 Port1: 0x76468	
Table 436: Port Extend Interrupt Mask (PEIM) Register .....	582
Offset: Port0: 0x7246C Port1: 0x7646C	
Table 437: Port Tx FIFO Urgent Threshold (PxTFUT) Register .....	582
Offset: Port0: 0x72474 Port1: 0x76474	
Table 438: Port Rx Minimal Frame Size (PxMFS) Register .....	583
Offset: Port0: 0x7247C Port1: 0x7647C	
Table 439: Port Rx Discard Frame Counter (PxDFC) Register .....	583
Offset: Port0: 0x72484 Port1: 0x76484	
Table 440: Port Overrun Frame Counter (PxOFC) Register .....	583
Offset: Port0: 0x72488 Port1: 0x76488	
Table 441: Port Internal Address Error (EUIAE) Register .....	584
Offset: Port0: 0x72494 Port1: 0x76494	
Table 442: Ethernet Type Priority Register .....	584
Offset: Port0: 0x724BC Port1: 0x764BC	
Table 443: Ethernet Current Receive Descriptor Pointers (CRDP) Register (n=0–7) .....	585
Offset: Port0: Q0: 0x7260C, Q1: 0x7261C, Q2: 0x7262C, Q3: 0x7263C, Q4: 0x7264C, Q5: 0x7265C,	
Q6: 0x7266C, Q7: 0x7267C	
Port1: Q0: 0x7660C, Q1: 0x7661C, Q2: 0x7662C, Q3: 0x7663C, Q4: 0x7664C, Q5: 0x7665C,	
Q6: 0x7666C, Q7: 0x7667C	
Table 444: Receive Queue Command (RQC) Register .....	585
Offset: Port0: 0x72680 Port1: 0x76680	
Table 445: Transmit Current Served Descriptor Pointer Register .....	586
Offset: Port0: 0x72684 Port1: 0x76684	

---

Table 446: Transmit Current Queue Descriptor Pointer (TCQDP) Register (n=0–7) .....	586
Offset: Port0: Q0: 0x726C0, Q1: 0x726C4, Q2: 0x726C8, Q3: 0x726CC, Q4: 0x726D0, Q5: 0x726D4, Q6: 0x726D8, Q7: 0x726DC	
Port1: Q0: 0x766C0, Q1: 0x766C4, Q2: 0x766C8, Q3: 0x766CC, Q4: 0x766D0, Q5: 0x766D4, Q6: 0x766D8, Q7: 0x766DC	
Table 447: Destination Address Filter Special Multicast Table (DFSMT) Register (n=0–63) .....	586
Offset: Port0: Register0: 0x73400, Register1: 0x73404...Register63: 0x734FC	
Port1: Register0: 0x77400, Register1: 0x77404...Register63: 0x774FC	
Table 448: Destination Address Filter Other Multicast Table (DFOMT) Register (n=0–63) .....	587
Offset: Port0: Register0: 0x73500, Register1: 0x73504...Register63: 0x735FC	
Port1: Register0: 0x77500, Register1: 0x77504...Register63: 0x775FC	
Table 449: Destination Address Filter Unicast Table (DFUT) Register (n=0–3) .....	589
Offset: Port0: Register0: 0x73600, Register1: 0x73604, Register2: 0x73608, Register3: 0x7360C	
Port1: Register0: 0x77600, Register1: 0x77604, Register2: 0x77608, Register3: 0x7760C	
Table 450: Transmit Queue Command (TQC) Register .....	590
Offset: Port0: 0x72448 Port1: 0x76448	
Table 451: Transmit Queue Fixed Priority Configuration (TQFPC) Register .....	591
Offset: Port0: 0x724DC Port1: 0x764DC	
Table 452: Port Transmit Token-Bucket Rate Configuration (PTTBRC) Register .....	591
Offset: Port0: 0x724E0 Port1: 0x764E0	
Table 453: Transmit Queue Command1 (TQC1) Register .....	591
Offset: Port0: 0x724E4 Port1: 0x764E4	
Table 454: Port Maximum Transmit Unit (PMTU) Register .....	592
Offset: Port0: 0x724E8 Port1: 0x764E8	
Table 455: Port Maximum Token Bucket Size (PMTBS) Register .....	592
Offset: Port0: 0x724EC Port1: 0x764EC	
Table 456: Queue Transmit Token-Bucket Counter (QxTTBC) Register (n=0–7) .....	593
Offset: Port0: Q0: 0x72700, Q1: 0x72710, Q2: 0x72720, Q3: 0x72730, Q4: 0x72740, Q5: 0x72750, Q6: 0x72760, Q7: 0x72770	
Port1: Q0: 0x76700, Q1: 0x76710, Q2: 0x76720, Q3: 0x76730, Q4: 0x76740, Q5: 0x76750, Q6: 0x76760, Q7: 0x76770	
Table 457: Transmit Queue Token Bucket Configuration (TQxTBC) Register (n=0–7) .....	593
Offset: Port0: Q0: 0x72704, Q1: 0x72714, Q2: 0x72724, Q3: 0x72734, Q4: 0x72744, Q5: 0x72754, Q6: 0x72764, Q7: 0x72774	
Port1: Q0: 0x76704, Q1: 0x76714, Q2: 0x76724, Q3: 0x76734, Q4: 0x76744, Q5: 0x76754, Q6: 0x76764, Q7: 0x76774	
Table 458: Transmit Queue Arbiter Configuration (TQxAC) Register (n=0–7) .....	594
Offset: Port0: Q0: 0x72708, Q1: 0x72718, Q2: 0x72728, Q3: 0x72738, Q4: 0x72748, Q5: 0x72758, Q6: 0x72768, Q7: 0x72778	
Port1: Q0: 0x76708, Q1: 0x76718, Q2: 0x76728, Q3: 0x76738, Q4: 0x76748, Q5: 0x76758, Q6: 0x76768, Q7: 0x76778	
Table 459: Port Transmit Token-Bucket Counter (PTTBC) Register .....	595
Offset: Port0: 0x72780 Port1: 0x76780	
Table 460: Transmission Queue IPG (TQxIPG) Register (n=2–3) .....	595
Offset: Port0: Q2: 0x727A8, Q3: 0x727B8 Port1: Q2: 0x767A8, Q3: 0x767B8	
Table 461: High Token in Low Packet (HITKNinLOPKT) Register .....	595
Offset: Port0: 0x727C0 Port1: 0x767C0	
Table 462: High Token in Asynchronous Packet (HITKNinASYNCPKT) Register .....	595
Offset: Port0: 0x727C4 Port1: 0x767C4	
Table 463: Low Token in Asynchronous Packet (LOTKNinASYNCPKT) Register .....	596
Offset: Port0: 0x727C8 Port1: 0x767C8	
Table 464: Transmission Speed (TS) Register .....	596
Offset: Port0: 0x727D0 Port1: 0x767D0	



Table 466: PTP Command Register.....	598
Offset: 0x7C000	
Table 467: PTP Data Register.....	599
Offset: 0x7C008	
Table 468: PTP Reset Register.....	599
Offset: 0x7C010	
Table 469: PTP Clock Select Register .....	600
Offset: 0x7C018	
Table 470: PTP Global Configuration0 Register .....	601
Offset: 0x0, or Decimal 0	
Table 471: PTP Global Configuration1 Register .....	601
Offset: 0x1, or Decimal 1	
Table 472: PTP Global Configuration2 Register .....	602
Offset: 0x2, or Decimal 2	
Table 473: PTP Global Configuration3 Register .....	602
Offset: 0x3, or Decimal 3	
Table 474: PTP Global Status0 Register.....	603
Offset: 0x8, or Decimal 8	
Table 475: PTP Port Configuration0 Register .....	604
Offset: 0x0, or Decimal 0	
Table 476: PTP Port Configuration1 Register .....	604
Offset: 0x1, or Decimal 1	
Table 477: PTP Port Configuration2 Register .....	605
Offset: 0x2, or Decimal 2	
Table 478: PTP Port Status0 Register .....	608
Offset: 0x8, or Decimal 8	
Table 479: PTP Port Status1 Register .....	609
Offset: 0x9 and 0xA, or Decimal 9 and Decimal 10	
Table 480: PTP Port Status2 Register .....	609
Offset: 0xB, or Decimal 11	
Table 481: PTP Port Status3 Register .....	609
Offset: 0xC, or Decimal 12	
Table 482: PTP Port Status4 Register .....	611
Offset: 0xD and 0xE, or Decimal 13 and 14	
Table 483: PTP Port Status5 Register .....	611
Offset: 0xF, or Decimal 15	
Table 484: PTP Port Status6 Register .....	611
Offset: 0x10, or Decimal 16	
Table 485: PTP Port Status7 Register .....	613
Offset: 0x11 and 0x12, or Decimal 17 and 18	
Table 486: PTP Port Status8 Register .....	613
Offset: 0x13, or Decimal 19	
Table 487: PTP Port Status9 Register .....	613
Offset: 0x15, or Decimal 21	
Table 488: TAI Global Configuration, PTP Port = 0xE .....	615
Offset: 0x0, or Decimal 0	
Table 489: TAI Global Configuration Register, PTP Port = 0xE and 0xF .....	617
Offset: 0x1, or Decimal 1	
Table 490: TAI Global Configuration0 Register.....	618
Offset: 0x2 and 0x3, or Decimal 2 and 3	
Table 491: TAI Global Configuration1 Register.....	618
Offset: 0x4, or Decimal 4	

---

Table 492: TAI Global Configuration2 Register .....	618
Offset: 0x5, or Decimal 5	
Table 493: TAI Global Configuration3 Register .....	619
Offset: 0x6, or Decimal 6	
Table 494: TAI Global Configuration4 Register .....	619
Offset: 0x7, or Decimal 7	
Table 495: TAI Global Status0 Register .....	620
Offset: 0x8, or Decimal 8	
Table 496: TAI Global Status1 Register .....	621
Offset: 0x9, or Decimal 9	
Table 497: TAI Global Status2 Register .....	621
Offset: 0xA and 0xB, or Decimal 10 and 11	
Table 498: PTP Global Status1 Register .....	622
Offset: 0xE and 0xF, or Decimal 14 and 15	
Table 499: MAC MIB Counters.....	623
Offset: Port0: 0x73000–0x7307C, Port1: 0x77000–0x7707C	

## A.9 USB 2.0 Registers..... **625**

Table 501: USB 2.0 Window0 Control Register .....	625
Offset: 0x50320	
Table 502: USB 2.0 Window0 Base Register .....	626
Offset: 0x50324	
Table 503: USB 2.0 Window1 Control Register .....	626
Offset: 0x50330	
Table 504: USB 2.0 Window1 Base Register .....	627
Offset: 0x50334	
Table 505: USB 2.0 Window2 Control Register .....	627
Offset: 0x50340	
Table 506: USB 2.0 Window2 Base Register .....	628
Offset: 0x50344	
Table 507: USB 2.0 Window3 Control Register .....	628
Offset: 0x50350	
Table 508: USB 2.0 Window3 Base Register .....	629
Offset: 0x50354	
Table 509: USB 2.0 Bridge Control Register .....	629
Offset: 0x50300	
Table 510: USB 2.0 Bridge Interrupt Cause Register .....	629
Offset: 0x50310	
Table 511: USB 2.0 Bridge Interrupt Mask Register .....	630
Offset: 0x50314	
Table 512: USB 2.0 Bridge Error Address Register .....	630
Offset: 0x5031C	
Table 513: USB 2.0 PHY Configuration0 Register .....	630
Offset: 0x50360	
Table 514: USB 2.0 Power Control Register .....	631
Offset: 0x50400	

## A.10 Cryptographic Engine and Security Accelerator (CESA) Registers ..... **634**

Table 517: AES Decryption Key Column 7 Register .....	636
Offset: 0x3DDC0	
Table 518: AES Decryption Key Column 6 Register .....	636
Offset: 0x3DDC4	



Table 519: AES Decryption Key Column 5 Register .....	636
Offset: 0x3DDC8	
Table 520: AES Decryption Key Column 4 Register .....	636
Offset: 0x3DDCC	
Table 521: AES Decryption Key Column 3 Register .....	636
Offset: 0x3DDD0	
Table 522: AES Decryption Key Column 2 Register .....	637
Offset: 0x3DDD4	
Table 523: AES Decryption Key Column 1 Register .....	637
Offset: 0x3DDD8	
Table 524: AES Decryption Key Column 0 Register .....	637
Offset: 0x3DDDC	
Table 525: AES Decryption Data In/Out Column 3 Register .....	637
Offset: 0x3DDE0	
Table 526: AES Decryption Data In/Out Column 2 Register .....	637
Offset: 0x3DDE4	
Table 527: AES Decryption Data In/Out Column 1 Register .....	638
Offset: 0x3DDE8	
Table 528: AES Decryption Data In/Out Column 0 Register .....	638
Offset: 0x3DDEC	
Table 529: AES Decryption Command Register .....	638
Offset: 0x3DDF0	
Table 530: AES Encryption Key Column 7 Register .....	639
Offset: 0x3DD80	
Table 531: AES Encryption Key Column 6 Register .....	639
Offset: 0x3DD84	
Table 532: AES Encryption Key Column 5 Register .....	639
Offset: 0x3DD88	
Table 533: AES Encryption Key Column 4 Register .....	639
Offset: 0x3DD8C	
Table 534: AES Encryption Key Column 3 Register .....	640
Offset: 0x3DD90	
Table 535: AES Encryption Key Column 2 Register .....	640
Offset: 0x3DD94	
Table 536: AES Encryption Key Column 1 Register .....	640
Offset: 0x3DD98	
Table 537: AES Encryption Key Column 0 Register .....	640
Offset: 0x3DD9C	
Table 538: AES Encryption Data In/Out Column 3 Register .....	640
Offset: 0x3DDA0	
Table 539: AES Encryption Data In/Out Column 2 Register .....	641
Offset: 0x3DDA4	
Table 540: AES Encryption Data In/Out Column 1 Register .....	641
Offset: 0x3DDA8	
Table 541: AES Encryption Data In/Out Column 0 Register .....	641
Offset: 0x3DDAC	
Table 542: AES Encryption Command Register .....	641
Offset: 0x3DBB0	
Table 543: DES Initial Value Low Register .....	642
Offset: 0x3DD40	
Table 544: DES Initial Value High Register .....	642
Offset: 0x3DD44	

---

Table 545: DES Key0 Low Register .....	642
Offset: 0x3DD48	
Table 546: DES Key0 High Register .....	642
Offset: 0x3DD4C	
Table 547: DES Key1 Low Register .....	643
Offset: 0x3DD50	
Table 548: DES Key1 High Register .....	643
Offset: 0x3DD54	
Table 549: DES Command Register .....	643
Offset: 0x3DD58	
Table 550: DES Key2 Low Register .....	644
Offset: 0x3DD60	
Table 551: DES Key2 High Register .....	644
Offset: 0x3DD64	
Table 552: DES Data Buffer Low Register .....	644
Offset: 0x3DD70	
Table 553: DES Data Buffer High Register .....	645
Offset: 0x3DD74	
Table 554: DES Data Out Low Register .....	645
Offset: 0x3DD78	
Table 555: DES Data Out High Register .....	645
Offset: 0x3DD7C	
Table 556: Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register .....	645
Offset: 0x3DE20	
Table 557: Cryptographic Engines and Security Accelerator Interrupt Mask Register .....	647
Offset: 0x3DE24	
Table 558: Security Accelerator Command Register .....	647
Offset: 0x3DE00	
Table 559: Security Accelerator Descriptor Pointer Register .....	647
Offset: 0x3DE04	
Table 560: Security Accelerator Configuration Register .....	648
Offset: 0x3DE08	
Table 561: Security Accelerator Status Register .....	648
Offset: 0x3DE0C	
Table 562: SHA-1/MD5 Initial Value/Digest A Register .....	649
Offset: 0x3DD00	
Table 563: SHA-1/MD5 Initial Value/Digest B Register .....	649
Offset: 0x3DD04	
Table 564: SHA-1/MD5 Initial Value/Digest C Register .....	649
Offset: 0x3DD08	
Table 565: SHA-1/MD5 Initial Value/Digest D Register .....	649
Offset: 0x3DD0C	
Table 566: SHA-1 Initial Value/Digest E Register .....	650
Offset: 0x3DD10	
Table 567: SHA-1/MD5 Authentication Command Register .....	650
Offset: 0x3DD18	
Table 568: SHA-1/MD5 Bit Count Low Register .....	651
Offset: 0x3DD20	
Table 569: SHA-1/MD5 Bit Count High Register .....	651
Offset: 0x3DD24	
Table 570: SHA-1/MD5 Data In Register .....	652
Offset: 0x3DD38	



Table 571: Base Address Register (n=0–3) .....	652
Offset: BAR0: 0x30A00, BAR1: 0x30A08, BAR2: 0x30A10, BAR3: 0x30A18	
Table 572: Window Control Register (n=0–3) .....	652
Offset: SR0: 0x30A04, SR1: 0x30A0C, SR2: 0x30A14, SR3: 0x30A1C	
Table 573: Control Register .....	653
Offset: 0x30840	
Table 574: TDMA Byte Count Register .....	654
Offset: 0x30800	
Table 575: TDMA Source Address Register .....	655
Offset: 0x30810	
Table 576: TDMA Destination Address Register .....	655
Offset: 0x30820	
Table 577: Next Descriptor Pointer Register .....	655
Offset: 0x30830	
Table 578: Current Descriptor Pointer Register .....	655
Offset: 0x30870	
Table 579: TDMA Error Cause Register .....	656
Offset: 0x308C8	
Table 580: TDMA Error Mask Register .....	656
Offset: 0x308CC	

## A.11 XOR Engine Registers ..... 657

Table 582: XOR Engine [0..1] Window Control (XExWCR) Register (n=0–1) .....	659
Offset: Port0: XOR0: 0x60A40, XOR1: 0x60A44	
Port1: XOR0: 0x60B40, XOR1: 0x60B44	
Table 583: XOR Engine Base Address (XEBarRx) Register (n=0–7) .....	660
Offset: Port0: XEBAR0: 0x60A50, XEBAR1: 0x60A54, XEBAR2: 0x60A58, XEBAR3: 0x60A5C,	
XEBAR4: 0x60A60, XEBAR5: 0x60A64, XEBAR6: 0x60A68, XEBAR7: 0x60A6C	
Port1: XEBAR0: 0x60B50, XEBAR1: 0x60B54, XEBAR2: 0x60B58, XEBAR3: 0x60B5C,	
XEBAR4: 0x60B60, XEBAR5: 0x60B64, XEBAR6: 0x60B68, XEBAR7: 0x60B6C	
Table 584: XOR Engine Size Mask (XESMRx) Register (n=0–7) .....	660
Offset: Port0: XESMR0: 0x60A70, XESMR1: 0x60A74, XESMR2: 0x60A78, XESMR3: 0x60A7C,	
XESMR4: 0x60A80, XESMR5: 0x60A84, XESMR6: 0x60A88, XESMR7: 0x60A8C	
Port1: XESMR0: 0x60B70, XESMR1: 0x60B74, XESMR2: 0x60B78, XESMR3: 0x60B7C,	
XESMR4: 0x60B80, XESMR5: 0x60B84, XESMR6: 0x60B88, XESMR7: 0x60B8C	
Table 585: XOR Engine High Address Remap (XEHARRx) Register (n=0–3) .....	661
Offset: Port0: XEHARR0: 0x60A90, XEHARR1: 0x60A94, XEHARR2: 0x60A98, XEHARR3: 0x60A9C	
Port1: XEHARR0: 0x60B90, XEHARR1: 0x60B94, XEHARR2: 0x60B98, XEHARR3: 0x60B9C	
Table 586: XOR Engine [0..1] Address Override Control (XEAOCR) Register (n=0–1) .....	661
Offset: Port0: XEAOCR0: 0x60AA0, XEAOCR1: 0x60AA4	
Port1: XEAOCR0: 0x60BA0, XEAOCR1: 0x60BA4	
Table 587: XOR Engine Channel Arbiter (XECHAR) Register .....	663
Offset: Port0: 0x60800 Port1: 0x60900	
Table 588: XOR Engine [0..1] Configuration (XExCR) Register (n=0–1) .....	664
Offset: Port0: XOR0: 0x60810, XOR1: 0x60814	
Port1: XOR0: 0x60910, XOR1: 0x60914	
Table 589: XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) .....	665
Offset: Port0: XOR0: 0x60820, XOR1: 0x60824	
Port1: XOR0: 0x60920, XOR1: 0x60924	
Table 590: XOR Engine [0..1] Next Descriptor Pointer (XExNDPR) Register (n=0–1) .....	666
Offset: Port0: XOR0: 0x60A00, XOR1: 0x60A04	
Port1: XOR0: 0x60B00, XOR1: 0x60B04	

Table 591: XOR Engine [0..1] Current Descriptor Pointer (XExCDPR) Register (n=0–1) .....	666
Offset: Port0: XOR0: 0x60A10, XOR1: 0x60A14	
Port1: XOR0: 0x60B10, XOR1: 0x60B14	
Table 592: XOR Engine [0..1] Byte Count (XExBCR) Register (n=0–1) .....	667
Offset: Port0: XOR0: 0x60A20, XOR1: 0x60A24	
Port1: XOR0: 0x60B20, XOR1: 0x60B24	
Table 593: XOR Engine Interrupt Cause (XEICR1) Register .....	667
Offset: Port0: 0x60830 Port1: 0x60930	
Table 594: XOR Engine Interrupt Mask (XEIMR) Register .....	668
Offset: Port0: 0x60840 Port1: 0x60940	
Table 595: XOR Engine Error Cause (XEECR) Register .....	669
Offset: Port0: 0x60850 Port1: 0x60950	
Table 596: XOR Engine Error Address (XEEAR) Register .....	669
Offset: Port0: 0x60860 Port1: 0x60960	
Table 597: XOR Engine 0 and 1 Destination Pointer (XExDPR0) Register (n=0–1) .....	670
Offset: Port0: XOR0: 0x60AB0, XOR1: 0x60AB4	
Port1: XOR0: 0x60BB0, XOR1: 0x60BB4	
Table 598: XOR Engine 0 and 1 Block Size (XExBSR) Register (n=0–1) .....	670
Offset: Port0: XOR0: 0x60AC0, XOR1: 0x60AC4	
Port1: XOR0: 0x60BC0, XOR1: 0x60BC4	
Table 599: XOR Engine Initial Value Low (XEIVRL) Register .....	670
Offset: Port0: 0x60AE0 Port1: 0x60BE0	
Table 600: XOR Engine Initial Value High (XEIVRH) Register .....	670
Offset: Port0: 0x60AE4 Port1: 0x60BE4	

## A.12 TWSI Registers ..... 671

Table 602: TWSI Slave Address Register .....	671
Offset: 0x11000	
Table 603: TWSI Data Register .....	671
Offset: 0x11004	
Table 604: TWSI Control Register .....	671
Offset: 0x11008	
Table 605: TWSI Status Register .....	673
Offset: 0x1100C	
Table 606: TWSI Baud Rate Register .....	674
Offset: 0x1100C	
Table 607: TWSI Extended Slave Address Register .....	674
Offset: 0x11010	
Table 608: TWSI Soft Reset Register .....	674
Offset: 0x1101C	
Table 609: TWSI Initialization Last Data Register .....	674
Offset: 0x11098	

## A.13 NAND Flash Registers ..... 675

Table 611: NAND Read Parameters Register .....	675
Offset: 0x10418	
Table 612: NAND Write Parameters Register .....	676
Offset: 0x1041C	
Table 613: NAND Flash Control Register .....	676
Offset: 0x10470	



---

## A.14 UART Registers ..... 678

Table 615: Transmit Holding (THR) Register .....	678
Offset: UART0: 0x12000 UART1: 0x12100	
Table 616: Divisor Latch Low (DLL) Register .....	679
Offset: UART0: 0x12000 UART1: 0x12100	
Table 617: Receive Buffer (RBR) Register .....	679
Offset: UART0: 0x12000 UART1: 0x12100	
Table 618: Interrupt Enable (IER) Register .....	679
Offset: UART0: 0x12004 UART1: 0x12104	
Table 619: Divisor Latch High (DLH) Register .....	680
Offset: UART0: 0x12004 UART1: 0x12104	
Table 620: Interrupt Identity (IIR) Register .....	680
Offset: UART0: 0x12008 UART1: 0x12108	
Table 621: FIFO Control (FCR) Register .....	681
Offset: UART0: 0x12008 UART1: 0x12108	
Table 622: Line Control (LCR) Register .....	681
Offset: UART0: 0x1200C UART1: 0x1210C	
Table 623: Modem Control (MCR) Register .....	682
Offset: UART0: 0x12010 UART1: 0x12110	
Table 624: Line Status (LSR) Register .....	683
Offset: UART0: 0x12014 UART1: 0x12114	
Table 625: Modem Status (MSR) Register .....	684
Offset: UART0: 0x12018 UART1: 0x12118	
Table 626: Scratch Pad (SCR) Register .....	684
Offset: UART0: 0x1201C UART1: 0x1211C	

## A.15 SPI Registers ..... 685

Table 628: Serial Memory Interface Control Register .....	685
Offset: 0x10600	
Table 629: Serial Memory Interface Configuration Register .....	686
Offset: 0x10604	
Table 630: Serial Memory Data Out Register .....	687
Offset: 0x10608	
Table 631: Serial Memory Data In Register .....	687
Offset: 0x1060C	
Table 632: Serial Memory Interface Interrupt Cause Register .....	687
Offset: 0x10610	
Table 633: Serial Memory Interface Interrupt Mask Register .....	687
Offset: 0x10614	
Table 634: Serial Memory Direct Write Configuration Register .....	688
Offset: 0x10620	
Table 635: Serial Memory Direct Write Header Register .....	688
Offset: 0x10624	

## A.16 Audio Interface Registers ..... 689

Table 637: DCO Control Register .....	691
Offset: 0xA1204	
Table 638: SPCR and DCO Status Register .....	691
Offset: 0xA120C	
Table 639: Sample Counters Control Register .....	692
Offset: 0xA1220	

---

Table 640: Playback Sample Counter Register .....	693
Offset: 0xA1224	
Table 641: Recording Sample Counter Register .....	693
Offset: 0xA1228	
Table 642: Clocks Control Register .....	693
Offset: 0xA1230	
Table 643: Reserved Register .....	694
Offset: 0xA1238	
Table 644: Recording Window Base Address Register .....	694
Offset: 0xA0A00	
Table 645: Recording Window Control Register .....	694
Offset: 0xA0A04	
Table 646: Playback Window Base Address Register .....	695
Offset: 0xA0A08	
Table 647: Playback Window Control Register .....	695
Offset: 0xA0A0C	
Table 648: Audio Error Cause Register .....	695
Offset: 0xA1300	
Table 649: Audio Error Mask Register .....	696
Offset: 0xA1304	
Table 650: Audio Interrupt Cause Register .....	696
Offset: 0xA1308	
Table 651: Audio Interrupt Mask Register .....	698
Offset: 0xA130C	
Table 652: Recorded Byte Count for Interrupt Register .....	698
Offset: 0xA1310	
Table 653: Playback Byte Count for Interrupt Register .....	699
Offset: 0xA1314	
Table 654: Playback Control Register .....	699
Offset: 0xA1100	
Table 655: Playback Start Address Register .....	702
Offset: 0xA1104	
Table 656: Playback Buffer Size Register .....	702
Offset: 0xA1108	
Table 657: Playback Buffer Byte Counter Register .....	703
Offset: 0xA110C	
Table 658: Playback Byte Counter for Interrupt Register .....	703
Offset: 0xA1318	
Table 659: I2S Playback Control Register .....	703
Offset: 0xA2508	
Table 660: SPDIF Playback Control Register .....	704
Offset: 0xA2204	
Table 661: SPDIF Playback Channel Status Left n Register (n=0–5) .....	705
Offset: status0: 0xA2280, status1: 0xA2284, status2: 0xA2288, status3: 0xA228C, status4: 0xA2290, status5: 0xA2294	
Table 662: SPDIF Playback Channel Status Right n Register (n=0–5) .....	705
Offset: status0: 0xA22A0, status1: 0xA22A4, status2: 0xA22A8, status3: 0xA22AC, status4: 0xA22B0, status5: 0xA22B4	
Table 663: SPDIF Playback User Bits Left n Register (n=0–5) .....	705
Offset: User0: 0xA22C0, User1: 0xA22C4, User2: 0xA22C8, User3: 0xA22CC, User4: 0xA22D0, User5: 0xA22D4	



Table 664: SPDIF Playback User Bits Right n Register (n=0–5) .....	706
Offset: User0: 0xA22E0, User1: 0xA22E4, User2: 0xA22E8, User3: 0xA22EC, User4: 0xA22F0, User5: 0xA22F4	
Table 665: Recording Control Register .....	706
Offset: 0xA1000	
Table 666: Recording Start Address Register .....	708
Offset: 0xA1004	
Table 667: Recording Buffer Size Register .....	708
Offset: 0xA1008	
Table 668: Recording Buffer Byte Counter Register .....	709
Offset: 0xA100C	
Table 669: Recorded Byte Counter for Interrupt Register .....	709
Offset: 0xA131C	
Table 670: I2S Recording Control Register .....	709
Offset: 0xA2408	
Table 671: SPDIF Recording General Register .....	710
Offset: 0xA2004	
Table 672: SPDIF Recording Interrupt Cause and Mask Register .....	711
Offset: 0xA2008	
Table 673: SPDIF Recording Channel Status Left n Register (n=0–5) .....	712
Offset: status0: 0xA2180, status1: 0xA2184, status2: 0xA2188, status3: 0xA218C, status4: 0xA2190, status5: 0xA2194	
Table 674: SPDIF Recording Channel Status Right n Register (n=0–5) .....	713
Offset: status0: 0xA21A0, status1: 0xA21A4, status2: 0xA21A8, status3: 0xA21AC, status4: 0xA21B0, status5: 0xA21B4	
Table 675: SPDIF Recording User Bits Left n Register (n=0–5) .....	713
Offset: User0: 0xA21C0, User1: 0xA21C4, User2: 0xA21C8, User3: 0xA21CC, User4: 0xA21D0, User5: 0xA21D4	
Table 676: SPDIF Recording User Bits Right n Register (n=0–5) .....	713
Offset: User0: 0xA21E0, User1: 0xA21E4, User2: 0xA21E8, User3: 0xA21EC, User4: 0xA21F0, User5: 0xA21F4	

## A.17 SDIO Registers ..... 714

Table 678: DMA Buffer Address 16 LSB Register .....	715
Offset: 0x90000	
Table 679: DMA Buffer Address 16 MSB Register .....	715
Offset: 0x90004	
Table 680: Data Block Size Register .....	716
Offset: 0x90008	
Table 681: Data Block Count Register .....	716
Offset: 0x9000C	
Table 682: Argument in Command 16 LSB Register .....	716
Offset: 0x90010	
Table 683: Argument in Command 16 MSB Register .....	717
Offset: 0x90014	
Table 684: Transfer Mode Register .....	717
Offset: 0x90018	
Table 685: Command Register .....	719
Offset: 0x9001C	
Table 686: Response Halfword 0 Register .....	720
Offset: 0x90020	
Table 687: Response Halfword 1 Register .....	720
Offset: 0x90024	

---

Table 688: Response Halfword 2 Register .....	720
Offset: 0x90028	
Table 689: Response Halfword 3 Register .....	721
Offset: 0x9002C	
Table 690: Response Halfword 4 Register .....	721
Offset: 0x90030	
Table 691: Response Halfword 5 Register .....	721
Offset: 0x90034	
Table 692: Response Halfword 6 Register .....	721
Offset: 0x90038	
Table 693: Response Halfword 7 Register .....	722
Offset: 0x9003C	
Table 694: 16-bit Data Word Accessed by CPU Register .....	722
Offset: 0x90040	
Table 695: CRC7 of I Response Register .....	722
Offset: 0x90044	
Table 696: Host Present State 16 LSB Register .....	723
Offset: 0x90048	
Table 697: Host Control Register .....	724
Offset: 0x90050	
Table 698: Data Block Gap Control Register .....	727
Offset: 0x90054	
Table 699: Clock Control Register .....	728
Offset: 0x90058	
Table 700: Software Reset Register .....	728
Offset: 0x9005C	
Table 701: Normal Interrupt Status Register .....	729
Offset: 0x90060	
Table 702: Error Interrupt Status Register .....	730
Offset: 0x90064	
Table 703: Normal Interrupt Status Enable Register .....	732
Offset: 0x90068	
Table 704: Error Interrupt Status Enable Register .....	733
Offset: 0x9006C	
Table 705: Normal Interrupt Status Interrupt Enable Register .....	734
Offset: 0x90070	
Table 706: Error Interrupt Status Interrupt Enable Register .....	735
Offset: 0x90074	
Table 707: Auto CMD12 Interrupt Status Register .....	736
Offset: 0x90078	
Table 708: Current Number of Bytes Remaining in Data Block Register .....	736
Offset: 0x9007C	
Table 709: Current Number of Data Blocks Left to Be Transferred Register .....	737
Offset: 0x90080	
Table 710: Argument in Auto Cmd12 Command 16 LSB Transferred Register .....	737
Offset: 0x90084	
Table 711: Argument in Auto Cmd12 Command 16 MSB Transferred Register .....	737
Offset: 0x90088	
Table 712: Index of Auto Cmd12 Commands Transferred Register .....	738
Offset: 0x9008C	
Table 713: Auto Cmd12 Response Halfword 0 Register .....	738
Offset: 0x90090	



Table 714: Auto Cmd12 Response Halfword 1 Register .....	738
Offset: 0x90094	
Table 715: Auto Cmd12 Response Halfword 2 Register .....	738
Offset: 0x90098	
Table 716: Mbus Control Low Register .....	739
Offset: 0x90100	
Table 717: Mbus Control High Register .....	739
Offset: 0x90104	
Table 718: Window0 Control Register .....	740
Offset: 0x90108	
Table 719: Window0 Base Register .....	740
Offset: 0x9010C	
Table 720: Window1 Control Register .....	741
Offset: 0x90110	
Table 721: Window1 Base Register .....	741
Offset: 0x90114	
Table 722: Window2 Control Register .....	741
Offset: 0x90118	
Table 723: Window2 Base Register .....	742
Offset: 0x9011C	
Table 724: Window3 Control Register .....	742
Offset: 0x90120	
Table 725: Window3 Base Register .....	743
Offset: 0x90124	
Table 726: Clock Divider Value Register .....	743
Offset: 0x90128	
Table 727: Address Decoder Error Register .....	743
Offset: 0x9012C	
Table 728: Address Decoder Error Mask Register .....	743
Offset: 0x90130	

## A.18 Transport Stream (TS) Registers ..... 744

Table 730: TSU Modes Register .....	744
Offset: 0xB4000	
Table 731: TSU-Mbus Configuration Register .....	745
Offset: 0xB4010	
Table 732: Window0 Control Register .....	745
Offset: 0xB4030	
Table 733: Window0 Base Register .....	746
Offset: 0xB4034	
Table 734: Window1 Control Register .....	746
Offset: 0xB4040	
Table 735: Window1 Base Register .....	746
Offset: 0xB4044	
Table 736: Window2 Control Register .....	747
Offset: 0xB4050	
Table 737: Window2 Base Register .....	747
Offset: 0xB4054	
Table 738: Window3 Control Register .....	747
Offset: 0xB4060	
Table 739: Window3 Base Register .....	748
Offset: 0xB4064	

---

Table 741: TS Interface Configuration Register .....	750
Offset: Port0: 0xB8000 Port1: 0xB8800	
Table 742: TS DMA Parameter Register .....	752
Offset: Port0: 0xB8004 Port1: 0xB8804	
Table 743: Done Queue Base Register.....	752
Offset: Port0: 0xB8008 Port1: 0xB8808	
Table 744: Descriptor Queue Base Register.....	752
Offset: Port0: 0xB800C Port1: 0xB880C	
Table 745: Done Queue Write Pointer Register .....	753
Offset: Port0: 0xB8010 Port1: 0xB8810	
Table 746: Done Queue Read Pointer Register.....	753
Offset: Port0: 0xB8014 Port1: 0xB8814	
Table 747: Descriptor Queue Write Pointer Register .....	754
Offset: Port0: 0xB8018 Port1: 0xB8818	
Table 748: Descriptor Queue Read Pointer Register .....	754
Offset: Port0: 0xB801C Port1: 0xB881C	
Table 749: Current Descriptor Register.....	754
Offset: Port0: 0xB8020 Port1: 0xB8820	
Table 750: Current DMA Address Register .....	755
Offset: Port0: 0xB8024 Port1: 0xB8824	
Table 751: Current DMA Length Register .....	755
Offset: Port0: 0xB8028 Port1: 0xB8828	
Table 752: TSU Enable Access Register .....	755
Offset: Port0: 0xB802C Port1: 0xB882C	
Table 753: TSU Timestamp Register .....	756
Offset: Port0: 0xB8030 Port1: 0xB8830	
Table 754: TSU Status Register.....	756
Offset: Port0: 0xB8034 Port1: 0xB8834	
Table 755: TSU Timestamp Control Register.....	757
Offset: Port0: 0xB8038 Port1: 0xB8838	
Table 756: TSU Test Register.....	757
Offset: Port0: 0xB803C Port1: 0xB883C	
Table 757: TSU Interrupt Source Register .....	758
Offset: Port0: 0xB8040 Port1: 0xB8840	
Table 758: TSU Interrupt Mask Register .....	758
Offset: Port0: 0xB8044 Port1: 0xB8844	
Table 759: IRQ Parameter Register.....	759
Offset: Port0: 0xB8048 Port1: 0xB8848	
Table 760: TSU Next Descriptor1 Register .....	759
Offset: Port0: 0xB8050 Port1: 0xB8850	
Table 761: TSU Next Descriptor2 Register .....	759
Offset: Port0: 0xB8054 Port1: 0xB8854	
Table 762: TSU SyncByte Detection Register.....	760
Offset: Port0: 0xB8058 Port1: 0xB8858	
Table 763: TSU Revision Register .....	760
Offset: Port0: 0xB805C Port1: 0xB885C	
Table 764: TSU Aggregation Control Register .....	760
Offset: Port0: 0xB8060 Port1: 0xB8860	
Table 765: TSU Timestamp Interval Register .....	761
Offset: Port0: 0xB8064 Port1: 0xB8864	



---

## A.19 General Purpose Port Registers ..... 762

Table 767: GPIO Data Out Register .....	762
Offset: 0x10100	
Table 768: GPIO Data Out Enable Control Register .....	762
Offset: 0x10104	
Table 769: GPIO Blink Enable Register .....	763
Offset: 0x10108	
Table 770: GPIO Data In Polarity Register .....	763
Offset: 0x1010C	
Table 771: GPIO Data In Register .....	763
Offset: 0x10110	
Table 772: GPIO Interrupt Cause Register .....	763
Offset: 0x10114	
Table 773: GPIO Interrupt Mask Register .....	764
Offset: 0x10118	
Table 774: GPIO Interrupt Level Mask Register .....	764
Offset: 0x1011C	
Table 775: GPIO High Data Out Register .....	764
Offset: 0x10140	
Table 776: GPIO High Data Out Enable Control Register .....	764
Offset: 0x10144	
Table 777: GPIO High Blink Enable Register .....	765
Offset: 0x10148	
Table 778: GPIO High Data In Polarity Register .....	765
Offset: 0x1014C	
Table 779: GPIO High Data In Register .....	765
Offset: 0x10150	
Table 780: GPIO High Interrupt Cause Register .....	765
Offset: 0x10154	
Table 781: GPIO High Interrupt Mask Register .....	766
Offset: 0x10158	
Table 782: GPIO High Interrupt Level Mask Register .....	766
Offset: 0x1015C	

## A.20 RTC Registers ..... 767

Table 784: RTC Time Register .....	767
Offset: 0x10300	
Table 785: RTC Date Register .....	768
Offset: 0x10304	
Table 786: RTC Alarm Time Configuration Register .....	768
Offset: 0x10308	
Table 787: RTC Alarm Date Configuration Register .....	769
Offset: 0x1030C	
Table 788: RTC Interrupt Mask Register .....	770
Offset: 0x10310	
Table 789: RTC Interrupt Cause Register .....	770
Offset: 0x10314	

## A.21 Boot ROM Registers ..... 771

Table 791: Boot ROM Routine and Error Code Register .....	771
Offset: 0x100D0	

---

<b>A.22 MPP Registers .....</b>	<b>773</b>
Table 793: MPP Control 0 Register .....	773
Offset: 0x10000	
Table 794: MPP Control 1 Register .....	774
Offset: 0x10004	
Table 795: MPP Control 2 Register .....	774
Offset: 0x10008	
Table 796: MPP Control 3 Register .....	775
Offset: 0x1000C	
Table 797: MPP Control 4 Register .....	776
Offset: 0x10010	
Table 798: MPP Control 5 Register .....	776
Offset: 0x10014	
Table 799: MPP Control 6 Register .....	777
Offset: 0x10018	
Table 800: Sample at Reset Register .....	778
Offset: 0x10030	
<b>A.23 eFuse Registers .....</b>	<b>779</b>
Table 802: eFuse Protection Register .....	779
Offset: 0x1008C	
Table 803: eFuse0 Low Register .....	780
Offset: 0x100A4	
Table 804: eFuse0 High Register .....	780
Offset: 0x100A8	
Table 805: eFuse1 Low Register .....	780
Offset: 0x100AC	
Table 806: eFuse1 High Register .....	781
Offset: 0x100B0	
Table 807: eFuse Control Register .....	781
Offset: 0x100B4	
<b>A.24 Miscellaneous Registers.....</b>	<b>782</b>
Table 809: Device ID Register .....	782
Offset: 0x10034	
Table 810: Clock Control Register .....	782
Offset: 0x1004C	
Table 811: SYSRSTn Length Counter Register .....	783
Offset: 0x10050	
Table 812: Analog Group Configuration Register .....	783
Offset: 0x1007C	
Table 813: SSCG Configuration Register .....	783
Offset: 0x100D8	
Table 814: PTP Clock Configuration Register .....	784
Offset: 0x100DC	
Table 815: IO Configuration 0 Register .....	784
Offset: 0x100E0	

# A

# 88F6180/88F619x/88F6281 Register Set

## A.1 Registers Overview

This section details the 88F6180/88F619x/88F6281 registers.

### A.1.1 Register Description

The device internal registers are mapped into a 1 MB address window. However, only part of this space is really populated with registers. The chip registers are distributed among the device's different units (distributed register file). Each unit has its own 64 KB register file space.

All registers are 32 bits wide [31:0]. The device registers use Little Endian byte orientation in which the Most Significant Byte (MSB) of a multi-byte expression is located in the highest address. So, for example, if a register value is 0xAABBCCDD (bits[31:24] are 0xAA), a write of single byte 0x77 to address 0x1, results in a new register value of 0xAABB77DD. The bits within a given byte are always ordered so that bit[7] is the Most Significant bit (MSb) and bit[0] is the Least Significant bit (LSb).

Unless otherwise noted, the registers support byte, half-word, and word accesses.

### A.1.2 Register Field Type Codes

The 88F6180/88F619x/88F6281 registers are made up of up to 32-bit fields, where each field is associated with one or more bits. Each of these register fields have a unique programming functionality and their operation is defined by the field's type. The following list describes the function of each type:

**Table 87: Register Field Type Codes**

Type	Description
EXEC_HP	Written to with 1 (high pulse) leads to the execution of an action (e.g., test step). Read value is always 0.
EXEC_RB	Special function pair of bits (radio button). Write values 0b01 or 0b10 are possible (other values are ignored).
RO	Read Only. Writing to this type of field may cause unpredictable results.
ROC	Read Only Clear. After read, register field is cleared to zero.
RSVD	Reserved for future use. All reserved bits are read as zero unless otherwise noted.
RW	Read and Write.
RW0C	Read-only status, Write-0 to clear status register. Register bits indicate status when read, a set bit indicates a status event may be cleared by writing a 0. Writing a 1 to RW0C bits have no effect.
RW1C	Read-only status, Write-1 to clear status register. Register bits indicate status when read, a set bit indicates a status event may be cleared by writing a 1. Writing a 0 to RW1C bits have no effect.
RWR	Read, Write, and Reset. All bits are readable and writable. After reset the register field is cleared to zero.
RWS	Read, Write, and Set. All bits are readable and writable. After reset the register field is set to a non-zero value, as specified in the bit description.
SAR	Sample at Reset
SC	Self-Clear. Writing a one to this register causes the desired function to be immediately executed, then the register field is cleared to zero when the function is complete.

**Table 87: Register Field Type Codes (Continued)**

Type	Description
TO	Writable for testing only.
WO	Write Only. A write to the register field will trigger an internal function and a read will return an undefined value.

## A.2 Internal Registers Address Map

The device Internal registers reside in a 1-MB address space. This address space is distributed among the various device modules in 64-KB segments, as indicated in [Table 88](#).

**Table 88: Device Internal Registers Address Map**

Unit ID	Unit Name	Address Space Size	Address Range	Address Range in Hexadecimal
0	DDR registers	64 KB	0–64K	0x0000–0x0FFF
1	TWSI, UART, SPI flash, NAND flash, RTC, GPIO, and MPP registers	64 KB	64K–128K	0x1000–0x1FFF
2	Mbus-L to Mbus Bridge registers	64 KB	128K–192K	0x2000–0x2FFF
3	Cryptographic Engine and Security Accelerator registers	64 KB	192K–256K	0x3000–0x3FFF
4	PCI Express registers	64 KB	256K–320K	0x4000–0x4FFF
5	USB registers	64 KB	320K–384K	0x5000–0x5FFF
6	XOR engine registers	64 KB	384K–448K	0x6000–0x6FFF
7	Gigabit Ethernet registers	64 KB	448K–512K	0x7000–0x7FFF
8	SATA registers (88F619x and 88F6281 only) <b>NOTE:</b> In the 88F6180, this address space is reserved.	64 KB	512K–576K	0x8000–0x8FFF
9	SDIO registers	64 KB	576K–640K	0x9000–0x9FFF
A	Audio registers (88F6180, 88F6192, and 88F6281 only)	64 KB	640K–704K	0xA000–0xAF
B	MPEG Transport Stream registers (88F6192 and 88F6281 only) <b>NOTE:</b> In the 88F6180, and 88F6190 this address space is reserved.	64 KB	704K–768K	0xB000–0xBFFF
C	Reserved	64 KB	768K–832K	0xC000–0xCFFF
D	TDM registers (88F619x/88F6281 only) <b>NOTE:</b> In the 88F6180, this address space is reserved.	64 KB	832K–896K	0xD000–0xDFFF
E	Reserved	64 KB	896K–960K	0xE000–0xEFFF
F	Reserved	64 KB	960K–1024K	0xF000–0xFFFF

## A.3

## Mbus-L to Mbus Bridge Registers

Refer to the "Mbus-L to Mbus Bridge" section of the "Internal Architecture" of this document.

The following table provides a summarized list of all of the Mbus-L to Mbus Bridge registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 89: Register Map Table for the Mbus-L to Mbus Bridge Registers**

Register Name	Offset	Table and Page
<b>CPU Address Map Registers</b>		
Window0 Control Register	0x20000	<a href="#">Table 90, p. 357</a>
Window0 Base Register	0x20004	<a href="#">Table 91, p. 357</a>
Window0 Remap Low Register	0x20008	<a href="#">Table 92, p. 358</a>
Window0 Remap High Register	0x2000C	<a href="#">Table 93, p. 358</a>
Window1 Control Register	0x20010	<a href="#">Table 94, p. 358</a>
Window1 Base Register	0x20014	<a href="#">Table 95, p. 359</a>
Window1 Remap Low Register	0x20018	<a href="#">Table 96, p. 359</a>
Window1 Remap High Register	0x2001C	<a href="#">Table 97, p. 359</a>
Window2 Control Register	0x20020	<a href="#">Table 98, p. 359</a>
Window2 Base Register	0x20024	<a href="#">Table 99, p. 360</a>
Window2 Remap Low Register	0x20028	<a href="#">Table 100, p. 360</a>
Window2 Remap High Register	0x2002C	<a href="#">Table 101, p. 360</a>
Window3 Control Register	0x20030	<a href="#">Table 102, p. 361</a>
Window3 Base Register	0x20034	<a href="#">Table 103, p. 361</a>
Window3 Remap Low Register	0x20038	<a href="#">Table 104, p. 361</a>
Window3 Remap High Register	0x2003C	<a href="#">Table 105, p. 362</a>
Window4 Control Register	0x20040	<a href="#">Table 106, p. 362</a>
Window4 Base Register	0x20044	<a href="#">Table 107, p. 362</a>
Window5 Control Register	0x20050	<a href="#">Table 108, p. 363</a>
Window5 Base Register	0x20054	<a href="#">Table 109, p. 363</a>
Window6 Control Register	0x20060	<a href="#">Table 110, p. 363</a>
Window6 Base Register	0x20064	<a href="#">Table 111, p. 364</a>
Window7 Control Register	0x20070	<a href="#">Table 112, p. 364</a>
Window7 Base Register	0x20074	<a href="#">Table 113, p. 365</a>
Device Internal Registers Base Address	0x20080	<a href="#">Table 114, p. 365</a>
<b>CPU Control and Status Registers</b>		
CPU Configuration Register	0x20100	<a href="#">Table 115, p. 365</a>
CPU Control and Status Register	0x20104	<a href="#">Table 116, p. 368</a>
RSTOUTn Mask Register	0x20108	<a href="#">Table 117, p. 368</a>
System Soft Reset Register	0x2010C	<a href="#">Table 118, p. 369</a>
Mbus-L to Mbus Bridge Interrupt Cause Register	0x20110	<a href="#">Table 119, p. 369</a>
Mbus-L to Mbus Bridge Interrupt Mask Register	0x20114	<a href="#">Table 120, p. 370</a>
Memory Power Management Control Register	0x20118	<a href="#">Table 121, p. 370</a>
Clock Gating Control Register	0x2011C	<a href="#">Table 122, p. 371</a>



Table 89: Register Map Table for the Mbus-L to Mbus Bridge Registers (Continued)

Register Name	Offset	Table and Page
BIU Configuration Register	0x20120	Table 123, p. 373
CPU L2 Configuration Register	0x20128	Table 124, p. 374
L2 RAM Timing 0 Register	0x20134	Table 125, p. 375
L2 RAM Timing 1 Register	0x20138	Table 126, p. 375
L2 RAM Power Management Control Register	0x20144	Table 127, p. 376
CPU RAM Management Control0 Register	0x20148	Table 128, p. 376
CPU RAM Management Control1 Register	0x2014C	Table 129, p. 376
CPU RAM Management Control2 Register	0x20150	Table 130, p. 377
CPU RAM Management Control3 Register	0x20154	Table 131, p. 377
<b>CPU Doorbell Registers</b>		
Host-to-CPU Doorbell Register	0x20400	Table 132, p. 377
Host-to-CPU Doorbell Mask Register	0x20404	Table 133, p. 378
CPU-to-Host Doorbell Register	0x20408	Table 134, p. 378
CPU-to-Host Doorbell Mask Register	0x2040C	Table 135, p. 378
<b>CPU Timers Registers</b>		
CPU Timers Control Register	0x20300	Table 136, p. 378
CPU Timer0 Reload Register	0x20310	Table 137, p. 379
CPU Timer 0 Register	0x20314	Table 138, p. 380
CPU Timer1 Reload Register	0x20318	Table 139, p. 380
CPU Timer 1 Register	0x2031C	Table 140, p. 380
CPU Watchdog Timer Reload Register	0x20320	Table 141, p. 380
CPU Watchdog Timer Register	0x20324	Table 142, p. 381
<b>L2 Non Cacheable Address</b>		
Window0 Base Address Register	0x20A00	Table 143, p. 381
Window0 Size Address Register	0x20A04	Table 144, p. 381
Window1 Base Address Register	0x20A08	Table 145, p. 381
Window1 Size Address Register	0x20A0C	Table 146, p. 382
Window2 Base Address Register	0x20A10	Table 147, p. 382
Window2 Size Address Register	0x20A14	Table 148, p. 382
Window3 Base Address Register	0x20A18	Table 149, p. 383
Window3 Size Address Register	0x20A1C	Table 150, p. 383
<b>Main Interrupt Controller Registers</b>		
Main Interrupt Cause Low Register	0x20200	Table 151, p. 383
Main IRQ Interrupt Mask Low Register	0x20204	Table 152, p. 385
Main FIQ Interrupt Mask Low Register	0x20208	Table 153, p. 386
Endpoint Interrupt Mask Low Register	0x2020C	Table 154, p. 386
Main Interrupt Cause High Register	0x20210	Table 155, p. 386
Main IRQ Interrupt Mask High Register	0x20214	Table 156, p. 387
Main FIQ Interrupt Mask High Register	0x20218	Table 157, p. 388
Endpoint Interrupt Mask High Register	0x2021C	Table 158, p. 388

### A.3.1 CPU Address Map Registers

**Table 90: Window0 Control Register**  
Offset: 0x20000

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x1	Window0 Enable 0 = Disable: Window is disabled. 1 = Enable: Window is enabled.
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x4	Specifies the target interface associated with this window. See the "Target Interface Configurations" section. <b>NOTE:</b> Do not configure this field to the SDRAM Controller.
15:8	Attr	RW 0xe8	Specifies the target interface attributes associated with this window. See the "Target Interface Configurations" section.
31:16	Size	RW 0x1FFF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as sequence of 1's followed by sequence of 0's. The number of 1's specifies the size of the window in 64 KByte granularity (e.g., a value of 0x00FF specifies 256 = 16 MByte). <b>NOTE:</b> A value of 0x0 specifies 64-KByte size.

**Table 91: Window0 Base Register**  
Offset: 0x20004

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> If the remap function for this register is not used, the <Remap> field in the Window0 Remap Low Register must be set to same value as the <Base> field in this register!			
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x8000	Base Address Used with the <Size> field in the Window0 Control Register to set the address window size and location. Corresponds to transaction address[31:16].



**Table 92: Window0 Remap Low Register**  
Offset: 0x20008

Bit	Field	Type/InitVal	Description
15:0	Reserved	RW 0x0	Reserved
31:16	Remap	RW 0x8000	Remap Address Used with the <Size> field in the Window0 Control Register to specifies address bits[31:0] to be driven to the target interface.

**Table 93: Window0 Remap High Register**  
Offset: 0x2000C

Bit	Field	Type/InitVal	Description
31:0	RemapHigh	RW 0x0	Specifies address bits[63:32] to be driven to the target interface.

**Table 94: Window1 Control Register**  
Offset: 0x20010

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x1	Window1 Enable See the Window0 Control Register 0 = Disable 1 = Enable
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x1	Specifies the unit ID (target interface) associated with this window. See the Window0 Control Register
15:8	Attr	RW 0x2f	Target specific attributes depending on the target interface. See the Window0 Control Register
31:16	Size	RW 0x7FF	Window Size See the Window0 Control Register

**Table 95: Window1 Base Register**  
**Offset: 0x20014**

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> If the remap function for this register is not used, the <Remap> field in the Window0 Remap Low Register must be set to same value as the <Base> field in this register.			
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0xD800	Base Address See the Window0 Base Register

**Table 96: Window1 Remap Low Register**  
**Offset: 0x20018**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RW 0x0	Reserved
31:16	Remap	RW 0xD800	Remap Address See the Window0 Remap Low Register

**Table 97: Window1 Remap High Register**  
**Offset: 0x2001C**

Bit	Field	Type/InitVal	Description
31:0	RemapHigh	RW 0x0	Remap Address See the Window0 Remap High Register.

**Table 98: Window2 Control Register**  
**Offset: 0x20020**

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x1	Window2 Enable See the Window0 Control Register 0 = Disable 1 = Enable
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x4	Specifies the unit ID (target interface) associated with this window. See the Window0 Control Register



**Table 98: Window2 Control Register (Continued)**  
Offset: 0x20020

Bit	Field	Type/InitVal	Description
15:8	Attr	RW 0xe0	Target specific attributes depending on the target interface. See the Window0 Control Register
31:16	Size	RW 0x0	Window Size See the Window0 Control Register

**Table 99: Window2 Base Register**  
Offset: 0x20024

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> If the remap function for this register is not used, the <Remap> field in the Window2 Remap Low Register must be set to same value as the <Base> field in this register.			
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0xC000	Base Address See the Window0 Base Register

**Table 100: Window2 Remap Low Register**  
Offset: 0x20028

Bit	Field	Type/InitVal	Description
15:0	Reserved	RW 0x0	Reserved
31:16	Remap	RW 0xC000	Remap Address See the Window0 Remap Low Register.

**Table 101: Window2 Remap High Register**  
Offset: 0x2002C

Bit	Field	Type/InitVal	Description
31:0	RemapHigh	RW 0x0	Remap Address See the Window0 Remap High Register.

**Table 102: Window3 Control Register**  
**Offset: 0x20030**

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x0	Window3 Enable See the Window0 Control Register 0 = Disable 1 = Enable
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See the Window0 Control Register
15:8	Attr	RW 0x0	Target specific attributes depending on the target interface. See the Window0 Control Register
31:16	Size	RW 0x0	Window Size See the Window0 Control Register

**Table 103: Window3 Base Register**  
**Offset: 0x20034**

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> If the remap function for this register is not used, the <Remap> field in the Window3 Remap Low Register must be set to same value as the <Base> field in this register.			
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x0	Base Address See the Window0 Base Register

**Table 104: Window3 Remap Low Register**  
**Offset: 0x20038**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RW 0x0	Reserved
31:16	Remap	RW 0x0	Remap Address See the Window0 Remap Low Register



**Table 105: Window3 Remap High Register**  
Offset: 0x2003C

Bit	Field	Type/InitVal	Description
31:0	RemapHigh	RW 0x0	Remap Address See the Window0 Remap High Register.

**Table 106: Window4 Control Register**  
Offset: 0x20040

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x1	Window4 Enable See the Window0 Control Register 0 = Disable 1 = Enable
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x3	Specifies the unit ID (target interface) associated with this window. See the Window0 Control Register
15:8	Attr	RW 0x1	Target specific attributes depending on the target interface. See the Window0 Control Register
31:16	Size	RW 0x0	Window Size See the Window0 Control Register

**Table 107: Window4 Base Register**  
Offset: 0x20044

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0xc801	Base Address See the Window0 Base Register.

**Table 108: Window5 Control Register**  
**Offset: 0x20050**

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x1	Window5 Enable See the Window0 Control Register 0 = Disable 1 = Enable
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x1	Specifies the unit ID (target interface) associated with this window. See the Window0 Control Register
15:8	Attr	RW 0x1E	Target specific attributes depending on the target interface. See the Window0 Control Register
31:16	Size	RW 0x7ff	Window Size See the Window0 Control Register

**Table 109: Window5 Base Register**  
**Offset: 0x20054**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0xe800	Base Address See the Window0 Base Register

**Table 110: Window6 Control Register**  
**Offset: 0x20060**

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x1	Window6 Enable See the Window0 Control Register 0 = Disable 1 = Enable
1	WinWrProt	RW 0x0	Window6 Write Protection. When this bit is set to 1, the window is write protected and a write transaction to this window responds to the CPU core with an indication. 0 = Disable: Not write protected 1 = Enable: Write protected
3:2	Reserved	RSVD 0x0	Reserved



**Table 110: Window6 Control Register (Continued)**  
Offset: 0x20060

Bit	Field	Type/InitVal	Description
7:4	Target	RW 0x1	Specifies the unit ID (target interface) associated with this window. See the Window0 Control Register
15:8	Attr	RW 0x1D	Target specific attributes depending on the target interface. See the Window0 Control Register
31:16	Size	RW 0x7ff	Window Size See the Window0 Control Register

**Table 111: Window6 Base Register**  
Offset: 0x20064

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0xf000	Base Address See the Window0 Base Register

**Table 112: Window7 Control Register**  
Offset: 0x20070

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x1	Window7 Enable See the Window0 Control Register 0 = Disable 1 = Enable
1	WinWrProt	RW 0x0	Window7 Write Protection When this bit is set to 1, the window is write protected and a write transaction to this window responds to the CPU core with an error indication. 0 = Disable: Not write protected 1 = Enable: Write protected
3:2	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x1	Specifies the unit ID (target interface) associated with this window. See the Window0 Control Register The initial value is sampled at reset and depend on Boot-ROM Enable: 0x1 = Boot from Device

**Table 112: Window7 Control Register (Continued)**  
Offset: 0x20070

Bit	Field	Type/InitVal	Description
15:8	Attr	RW 0x1B	<p>Target specific attributes depending on the target interface.  See the Window0 Control Register</p> <p>The initial value is sampled at reset and depend on Boot-Mode Device:  0x0 = Boot from PCI Express (If Boot Rom is enabled)  0x1E = Boot from SPI  0x2F = Boot from NAND Flash (CE care or CE don't care)</p>
31:16	Size	RW 0x07FF	<p>Window Size  See the Window0 Control Register</p>

**Table 113: Window7 Base Register**  
Offset: 0x20074

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0xf800	<p>Base Address  See the Window0 Base Register</p>

**Table 114: Device Internal Registers Base Address**  
Offset: 0x20080

Bit	Field	Type/InitVal	Description
19:0	Reserved	RSVD 0x0	Reserved
31:20	Base	RW 0xD00	Internal registers Base Address

### A.3.2 CPU Control and Status Registers

**Table 115: CPU Configuration Register**  
Offset: 0x20100

Bit	Field	Type/InitVal	Description
0	Reserved	RSVD 0x0	Reserved
1	VecInitLoc	RW 0x1	<p>Determines the reset location of the boot starting address.</p> <p>0 = MsbZeros: Boot starting address is 0x00000000.  1 = MsbOnes: Boot starting address is 0xFFFF0000.</p>



**Table 115: CPU Configuration Register (Continued)**  
Offset: 0x20100

Bit	Field	Type/InitVal	Description
2	AHBErrorProp	RW 0x1	Mbus-L Error propagation For a list of errors see "Error Handling". 0 = NoErrProp: Error indications are not propagated to Mbus-L. The transactions are completed normally. 1 = ErrProp: Error indications are propagated to Mbus-L.
3	EndianInit	RW 0x0	Endian Initialization Determines the endianness of the device CPU core operation after reset. This bit defines the initial value of bit [7] <B> of the Control Register--R1. 0 = Little: Little Endian mode 1 = Big: Big Endian mode
4	Reserved	RSVD 0x1	Reserved
5	CfgNcbBlocking	RW 0x1	Ncb blocking 0 = non blocking: Non blocking 1 = blocking: NC requests from the L2 write buffer (I2c_out) to bridge blocks any NC read request from the L2 cache.
6	CfgWaitForWbEmptyNcb	RW 0x0	Wait For Wb Empty Ncb 0 = NoWait 1 = Wait: If this bit is set or I2C exists, the D-cache state machine waits, for NCB read requests, until CPU WB is empty.
7	Reserved_7	RW 0x0	Reserved
11:8	CPU2MbusLTickDrv	RW 0x0	CPU to Mbus-L DDR Interface Tick Driver [3:0] Defines the timing when CPU drives the Mbus-L DDR interface signals. CPU drives the Mbus-L signals <CPU2MbusLTickDrv> cycles after each rising edge of the DDR Clock that unites with the rising edge of the CPU clock. In the CPU to DDR clock period ratio of 2:N: The CPU also drives the Mbus-L signals on the CPU clock rising edge that immediately follows the DDR clock rising edge that does not unite with the rising edge of the CPU clock.  <b>NOTE:</b> This field should be configured according to the CPU to DDR clock ratio. This field must be configured before access to DDR memory (as part of the DDR initialization sequence). The value of 0x0 always works; other values improve performance as follows: 1:1 = Must be set to 0x0. 1:2 = Must be set to 0x0. <b>NOTE:</b> When working in 2:n mode, set this field to 0x0. <b>NOTE:</b> This field has no effect on the interface between the CPU and the Mbus bridge.

**Table 115: CPU Configuration Register (Continued)**  
Offset: 0x20100

Bit	Field	Type/InitVal	Description
15:12	CPU2MbusLTickSa mple	RW 0x0	<p>CPU to Mbus-L Tick Sample [3:0]  Defines the timing when the CPU samples the Mbus-L DDR interface signals.  The CPU samples the Mbus-L signals &lt;CPU2MbusLTickSample&gt; CPU clock  cycles after the rising edge of each DDR Clock that unites with the rising edge  of the CPU clock.</p> <p>In the CPU to DDR clock period ratio of 2:N: The CPU also samples the  Mbus-L signals on the CPU rising edge that precedes each DDR clock rising  edge that does not unite with the rising edge of the CPU clock.</p> <p><b>NOTE:</b> This field should be configured according to the CPU to DDR clock  ratio. This field must be configured before access to DDR memory (as  part of the DDR initialization sequence). The value of 0x0 always  works; other values improve performance as follows:  1:1 = Must be set to 0x0.  1:2 = Must be set to 0x0.</p> <p><b>NOTE:</b> When working in 2:N mode, set this field to 0x0.</p> <p><b>NOTE:</b> This field has no effect on the interface between the CPU and the  Mbus bridge.</p>
16	Reserved	RW 0x0	Reserved Must write 0x0.
17	Reserved	RW 0x0	Reserved Must write 0x0.
18	Reserved_18	RW 0x0	Reserved
19	Reserved_19	RW 0x0	Reserved
20	DCachePriority	RW 0x1	D-cache Priority The D-cache has a higher priority in the BIU arbiter than the I-cache. 0 = Disable 1 = Enable
21	Reserved_21	RW 0x0	Reserved
22	Reserved_22	RW 0x0	Reserved
23	PexLinkdownReset Cpu	RW 0x0	PCI Express Link Down Reset of CPU 0 = Disable 1 = Enable
31:24	Reserved_24	RW 0x0	Reserved



**Table 116: CPU Control and Status Register**  
Offset: 0x20104

Bit	Field	Type/InitVal	Description
0	PEX0En	RW 0x0	PCI Express port0 Enable 0 = Disable 1 = Enable
1	CPURest	RW 0x0	CPU Reset When this bit is set to 1, the CPU core is reset. <b>NOTE:</b> From other interfaces, read/write.
2	SelfInt	SC 0x0	When set to 1, bit <CPUSelfInt> in Mbus-L to Mbus Bridge Interrupt Cause Register is also set to 1.
14:3	Reserved	RW 0x0	Reserved
15	BigEndian	RO 0x0	Big Endian Reflects the value of the Big Endian field of theCPU CP15 register. 0 = Little 1 = Big
27:16	Reserved	RW 0x0	Reserved
28	CPU_SW_Int_Blk	RW 0x0	When entering or exiting power saving mode, must set this bit to 1 to disable interrupts to CPU.
31:29	Reserved	RW 0x0	Reserved

**Table 117: RSTOUTn Mask Register**  
Offset: 0x20108

Bit	Field	Type/InitVal	Description
0	PexRstOutEn	RW 0x0	If set to 1, the device asserts RSTOUTn upon receiving a PCI Express reset indication from the PCI Express Endpoint interface, as configured in bit <EndPointIF> in the CPU Configuration Register.
1	WDRstOutEn	RW 0x0	If set to 1, the device asserts RSTOUTn upon watchdog timer expiration.
2	SoftRstOutEn	RW 0x0	If set to 1, the device asserts RSTOUTn upon software reset.
31:3	Reserved	RSVD 0x0	Reserved

**Table 118: System Soft Reset Register**  
**Offset: 0x2010C**

Bit	Field	Type/InitVal	Description
0	SystemSoftRst	RW 0x0	When software sets this bit to 1 and bit <SoftRstOutEn> is set to 1 in the RSTOUTn Mask Register, the device asserts the RSTOUTn pin.
31:1	Reserved	RSVD 0x0	Reserved

**Table 119: Mbus-L to Mbus Bridge Interrupt Cause Register**  
**Offset: 0x20110**

Bit	Field	Type/InitVal	Description
0	CPUSelfInt	RW0C 0x0	This bit is set when bit <SelfInt> is set to 1 in CPU Control and Status Register.
1	CPUTimer0IntReq	RW0C 0x0	CPU Timer 0 Interrupt This bit is set when field <CPUTimer0> in CPU Timer 0 Register reaches 0.
2	CPUTimer1IntReq	RW0C 0x0	CPU Timer 1 Interrupt This bit is set when field <CPUTimer1> in CPU Timer 1 Register reaches 0.
3	CPUWDTimerIntReq	RW0C 0x0	CPU Watchdog Timer Interrupt This bit is set when field <CPUWDTimer> in CPU Watchdog Timer Register reaches 0.
4	AccessErr	RW0C 0x0	Access Error On an erroneous read, this field is set when bit[2] <AHBErrorProp> in the CPU Configuration Register is cleared and the Mbus-L to Mbus bridge recognizes an error access (for example, burst to internal, write to wr_protected, unmapped address). On any erroneous write, this field is set (without regard to the setting of bit[2] <AHBErrorProp> in the CPU Configuration register).
5	Bit64Err	RW0C 0x0	This bit is set when bit[2] <AHBErrorProp> in the CPU Configuration Register is cleared and the Mbus-L to Mbus bridge recognizes an error indication from the Mbus target unit.
31:6	Reserved	RSVD 0x0	Reserved



**Table 120: Mbus-L to Mbus Bridge Interrupt Mask Register**  
**Offset: 0x20114**

Bit	Field	Type/InitVal	Description
5:0	Mask	RW 0x0	There is a mask bit per each cause bit. Mask only affects the assertion of interrupt pins. It does not affect the setting of bits in the Cause register.
31:6	Reserved	RW 0x0	Reserved

**Table 121: Memory Power Management Control Register**  
**Offset: 0x20118**

Bit	Field	Type/InitVal	Description
0	GE0_Mem_PD	RW 0x0	GbE0 Memory Power Down: 0 = Functional 1 = PowerDown
1	PEX0_Mem_PD	RW 0x0	PCI Express0 Memory Power Down; 0 = Functional 1 = PowerDown
2	USB0_Mem_PD	RW 0x0	USB0 Memory Power Down: 0 = Functional 1 = PowerDown
3	Dunit_Mem_PD	RW 0x0	Dunit Memory Power Down: 0 = Functional 1 = PowerDown
4	Runit_Mem_PD	RW 0x0	Runit Memory Power Down: 0 = Functional 1 = PowerDown
5	XOR0_Mem_PD	RW 0x0	XOR0 Memory Power Down: 0 = Functional 1 = PowerDown
6	SATA0_Mem_PD	RW 0x0	Sata0 Memory Power Down: 0 = Functional 1 = PowerDown
7	XOR1_Mem_PD	RW 0x0	XOR1 Memory Power Down: 0 = Functional 1 = PowerDown

**Table 121: Memory Power Management Control Register (Continued)**  
Offset: 0x20118

Bit	Field	Type/InitVal	Description
8	Crypto_Mem_PD	RW 0x0	Cryptographic HW accelerator Memory Power Down  0 = Functional 1 = PowerDown
9	AUDIO_Mem_PD	RW 0x0	Audio Memory Power Down  0 = Functional 1 = PowerDown
10	Reserved	RSVD 0x0	Reserved
11	SATA1_Mem_PD	RW 0x0	Sata1 Memory Power Down: 0 = Functional 1 = PowerDown
12	Reserved	RSVD 0x0	Reserved
13	GE1_Mem_PD	RW 0x0	GbE1 Memory Power Down: 0 = Functional 1 = PowerDown
31:14	Reserved	RSVD 0x0	Reserved

**Table 122: Clock Gating Control Register**  
Offset: 0x2011C

Bit	Field	Type/InitVal	Description
0	GE0_clock_enable	RW 0x1	GbE0 Clock Enable 0 = Clock Disable 1 = Clock Enable
1	Reserved	RW 0x0	Reserved.
2	Pex0_clock_enable	RW 0x1	PCI Express0 Clock Enable 0 = Clock Disable 1 = Clock Enable
3	USB0_clock_enable	RW 0x1	USB0 Clock Enable 0 = Clock Disable 1 = Clock Enable



**Table 122: Clock Gating Control Register (Continued)**  
Offset: 0x2011C

Bit	Field	Type/InitVal	Description
4	Sdio_clock_enable	RW 0x1	SDIO Clock Enable 0 = Clock Disable 1 = Clock Enable
5	TSU_clock_enable	RW 0x1	TSU Clock Enable 0 = Clock Disable 1 = Clock Enable
6	Dunit_clock_enable	RW 0x1	SDRAM Unit Clock Enable 0 = Clock Disable 1 = Clock Enable
7	Runit_clock_enable	RW 0x1	Runit Clock Enable 0 = Clock Disable 1 = Clock Enable
8	XOR0_clock_enable	RW 0x1	XOR0 Clock Enable <b>NOTE:</b> Set XOR to Clock Disable only if not using both XOR channels 0 = Clock Disable 1 = Clock Enable
9	AUDIO_clock_enable	RW 0x1	Audio Clock Enable 0 = Clock Disable 1 = Clock Enable
10	ClockRealign	RW 0x0	Configure Production Realignment Production patterns assist. If set, CPU clock to HCLK will re-align once CPU Reset is de-asserted, assuring a deterministic production tester patterns. 0 = Disable 1 = Enable
11	GotoPowerSave	RW 0x0	Configure Go To Power Save Software request to enter power save mode. 0 = Not_active 1 = Active
12	GotoPowerHalf	RW 0x0	Configure Go To Half Power Save Software request to enter power save mode qualifier (Works only if <GotoPowerSave> is set). If set, CPU clock will drop down to half of its operational frequency. If cleared, CPU clock will drop down to the bus clock frequency.  0 = Not_active 1 = Active
13	LoadNewRatio	RW 0x0	Configure Load New Ratio Software request to apply new bus to CPU core clock ratio. 0 = Not_active 1 = Active

**Table 122: Clock Gating Control Register (Continued)**  
Offset: 0x2011C

Bit	Field	Type/InitVal	Description
14	Sata0_clock_enable	RW 0x1	SATA Channel0 Clock Enable 0 = Clock Disable 1 = Clock Enable
15	Sata1_clock_enable	RW 0x1	SATA Channel1 Clock Enable 0 = Clock Disable 1 = Clock Enable
16	XOR1_clock_enable	RW 0x1	XOR1 Clock Enable <b>NOTE:</b> Set XOR to clock disable only if not using both XOR channels. 0 = Clock Disable 1 = Clock Enable
17	Crypto_clock_enable	RW 0x1	Cryptographic HW accelerator Clock Enable 0 = Clock Disable 1 = Clock Enable
18	Reserved	RSVD 0x1	Reserved
19	GE1_clock_enable	RW 0x1	GbE1 Clock Enable 0 = Clock Disable 1 = Clock Enable
20	V_clock_enable	RW 0x1	TDM clock enable 0 = Clock Disable 1 = Clock Enable
31:21	Reserved	RSVD 0x6	Reserved

**Table 123: BIU Configuration Register**  
Offset: 0x20120

Bit	Field	Type/InitVal	Description
0	Reserved	RW 0x1	Always write 1 to this bit.
1	Reserved	RW 0x0	Always write 0 to this field.
2	Reserved	RW 0x0	Always write 0 to this field.
3	Reserved	RW 0x0	Always write 0 to this field.
7:4	Reserved	RW 0x1	The value of this field must not be changed after reset.



**Table 123: BIU Configuration Register (Continued)**  
Offset: 0x20120

Bit	Field	Type/InitVal	Description
11:8	Reserved	RW 0x1	The value of this field must not be changed after reset.
15:12	Reserved	RW 0x1	The value of this field must not be changed after reset.
16	Reserved	RW 0x0	Sleep Exit Pending BIU-Is-Idle assertion
31:17	Reserved_31_17	RW 0x7f00	Reserved

**Table 124: CPU L2 Configuration Register**  
Offset: 0x20128

Bit	Field	Type/InitVal	Description
1:0	Reserved	RSVD 0x0	Reserved
2	L2EccEn	RW 0x0	L2 ECC Enable
3	L2Exist	RW 0x1	When this bit is set , L2 is enabled.
4	L2WtMode	RW 0x1	L2 Coherency mode 0 = WriteBack: L2 is working in Write Back (WB) mode. 1 = WriteThrough: L2 is working in Write Through (WT) mode.
31:5	Reserved	RSVD 0x8	Reserved

**Table 125: L2 RAM Timing 0 Register**  
Offset: 0x20134

Bit	Field	Type/InitVal	Description
31:0	L2RAMConfiguratio n0	RW 0xffffffff	<p>Tag RAM: conf_l2ram_ctrl0[1:0] RTC RAM0  Tag RAM: conf_l2ram_ctrl0[3:2] WTC RAM0  conf_l2ram_ctrl0[7:4] Reserved</p> <p>Valid RAM: conf_l2ram_ctrl0[9:8] RTC RAM  Valid RAM: conf_l2ram_ctrl0[11:10] WTC RAM</p> <p>Dirty RAM: conf_l2ram_ctrl0[13:12] RTC RAM  Dirty RAM: conf_l2ram_ctrl0[15:14] WTC RAM</p> <p>Data RAM: conf_l2ram_ctrl0[17:16] RTC RAM0  Data RAM: conf_l2ram_ctrl0[19:18] WTC RAM0  Data RAM: conf_l2ram_ctrl0[21:20] RTC RAM1  Data RAM: conf_l2ram_ctrl0[23:22] WTC RAM1  Data RAM: conf_l2ram_ctrl0[25:24] RTC RAM2  Data RAM: conf_l2ram_ctrl0[27:26] WTC RAM2  Data RAM: conf_l2ram_ctrl0[29:28] RTC RAM3  Data RAM: conf_l2ram_ctrl0[31:30] WTC RAM3</p> <p><b>NOTE:</b> Setting recommendations will be published after silicon qualifications.</p>

**Table 126: L2 RAM Timing 1 Register**  
Offset: 0x20138

Bit	Field	Type/InitVal	Description
19:0	L2RAMConfiguratio n1	RW 0xffff	<p>Data RAM: conf_l2ram_ctrl1[15:0] Reserved</p> <p>ECC RAM: conf_l2ram_ctrl1[17:16] RTC RAM0  ECC RAM: conf_l2ram_ctrl1[19:18] WTC RAM0</p> <p><b>NOTE:</b> Setting recommendations will be published after silicon qualifications</p>
31:20	Reserved_31_20	RW 0x9	<p>[31:20] Reserved</p> <p><b>NOTE:</b> Setting recommendations will be published after silicon qualifications</p>

**Table 127: L2 RAM Power Management Control Register**  
**Offset: 0x20144**

Bit	Field	Type/InitVal	Description
5:0	L2RAMConfiguratio n2	RW 0x0	Dirty Power Down: Tag Power Down: conf_l2ram_ctrl2[0]  Valid Power Down: conf_l2ram_ctrl2[1]  Dirty Power Down: conf_l2ram_ctrl2[2]  Data Power Down0: conf_l2ram_ctrl2[3] [4] Reserved  ECC Power Down: conf_l2ram_ctrl2[5]
31:6	Reserved_31_6	RW 0x0	Reserved

**Table 128: CPU RAM Management Control0 Register**  
**Offset: 0x20148**

Bit	Field	Type/InitVal	Description
19:0	ConfigClkTune	RW 0x1	[1:0] - addr clk tune [3:2] - reserved [5:4] - dtcmp clk tune [7:6] - reserved [9:8] - macdrv clk tune [11:10] - opmuxgm2 clk tune [15:14] - rf clk tune [17:16] - rfbypass clk tune [19:18] - pc dp clk tune
23:20	ictag_ct	RW 0x0	icache clk tune
27:24	dctag_ct	RW 0x0	dcache clk tune
31:28	rf_config	RW 0x0	regfile tuning

**Table 129: CPU RAM Management Control1 Register**  
**Offset: 0x2014C**

Bit	Field	Type/InitVal	Description
15:0	OpmuxTune	RW 0x0000	Opmux Tuning

**Table 129: CPU RAM Management Control1 Register (Continued)**  
Offset: 0x2014C

Bit	Field	Type/InitVal	Description
31:16	PcDpTune	RW 0x0	Pc Dp Tuning

**Table 130: CPU RAM Management Control2 Register**  
Offset: 0x20150

Bit	Field	Type/InitVal	Description
31:0	RtcWtcCfg	RW 0x99d00000	The bits in this field serve the following functions: [3:0] - RTC tuning for all D-cache SRAM-4K instances [7:4] - RTC tuning for all I-cache SRAM-4K instances [11:8] - RTC tuning for two instances of Physical Address SRAM [15:12] - RTC tuning for BPU SRAM [19:16] - RTC tuning for Dirty SRAM [21:20] - RTC tuning for TLB SRAM [23:22] - WTC tuning for TLB SRAM [27:24] - WTC/RTC tuning for DC Parity SRAM [31:28] - WTC/RTC tuning for IC Parity SRAM

**Table 131: CPU RAM Management Control3 Register**  
Offset: 0x20154

Bit	Field	Type/InitVal	Description
23:0	AddrConfig	RW 0x0	Addr Config tuning
31:24	Reserved_31_24	RW 0x0	Reserved

### A.3.3 CPU Doorbell Registers

**Table 132: Host-to-CPU Doorbell Register**  
Offset: 0x20400

Bit	Field	Type/InitVal	Description
31:0	HostIntCs	RW 0x0	Host Command Cause When this bit is not zero and the corresponding bit <HostIntCsMask> in the Host-to-CPU Doorbell Mask Register is set, bit <Host2CPUDoorbell> is set in the Main Interrupt Cause Register. A Host write of 1 set the bits in this field. A Host write of 0 has no effect. A CPU write of 0 clear the bits in this field. A CPU write of 1 has no effect.



**Table 133: Host-to-CPU Doorbell Mask Register**  
Offset: 0x20404

Bit	Field	Type/InitVal	Description
31:0	HostIntCsMask	RW 0x0	Host Interrupt Cause Mask Mask bit per each cause bit in the <HostIntCs> field in the Host-to-CPU Doorbell Register. Mask only affects the assertion of interrupt bit in Main Interrupt Cause Register. It does not affect the setting of bits in the Host-to-CPU Doorbell Register.

**Table 134: CPU-to-Host Doorbell Register**  
Offset: 0x20408

Bit	Field	Type/InitVal	Description
31:0	CPUIntCs	RW 0x0	CPU Interrupt Cause When a bit in this field is set and the corresponding bit in <CPUIntCsMask> in the CPU-to-Host Doorbell Mask Register is also set, bit <CPU2HostDoorbell> is set in the Main Interrupt Cause Register. A CPU write of 1 set the bits in this field. A CPU write of 0 has no effect. A Host write of 0 clear the bits in this field. A Host write of 1 has no effect.

**Table 135: CPU-to-Host Doorbell Mask Register**  
Offset: 0x2040C

Bit	Field	Type/InitVal	Description
31:0	CPUIntCsMask	RW 0x0	CPU Interrupt Cause Mask Mask bit per each cause bit in <CPUIntCs> field in the CPU-to-Host Doorbell Register. Mask only affects the assertion of the interrupt bit in the Main Interrupt Cause Register. It does not affect the setting of bits in the CPU-to-Host Doorbell Register.

### A.3.4 CPU Timers Registers

**Table 136: CPU Timers Control Register**  
Offset: 0x20300

Bit	Field	Type/InitVal	Description
0	CPUTimer0En	RW 0x0	CPU Timer 0 Enable

**Table 136: CPU Timers Control Register (Continued)**  
Offset: 0x20300

Bit	Field	Type/InitVal	Description
1	CPUTimer0Auto	RW 0x0	CPU Timer 0 Auto Mode When this bit is cleared to 0 and <CPUTimer0> in the CPU Timer0 Register has reached zero, <CPUTimer0> stops counting. When this bit is set to 1 and <CPUTimer0> has reached zero, <CPUTimer0Rel> in the CPU Timer0 Reload Register is reload to <CPUTimer0>, then it continues to count.
2	CPUTimer1En	RW 0x0	CPU Timer 1 Enable
3	CPUTimer1Auto	RW 0x0	CPU Timer 1 Auto Mode When this bit is clear to 0 and <CPUTimer1> in the CPU Timer1 Register has reached to zero, <CPUTimer1> stops counting. When this bit is set to 1 and <CPUTimer1> has reached zero, <CPUTimer1Rel> in the CPU Timer1 Reload Register is reload to <CPUTimer1>, then it continues to count.
4	CPUWDTimerEn	RW 0x0	CPU Watchdog Timer Enable 0 = WdDisabled 1 = WdEnabled
5	CPUWDTimerAuto	RW 0x0	CPU Watchdog Timer Auto Mode When this bit is clear to 0 and <CPUWDTimer> in the CPU Watchdog Timer Register has reached zero, <CPUWDTimer> stops counting. When this bit is set to 1 and <CPUWDTimer> has reached zero, <CPUTimer0Rel> in the CPU Timer0 Reload Register is reload to <CPUWDTimer>, then it continues to count.
31:6	Reserved	RW 0x0	Reserved

**Table 137: CPU Timer0 Reload Register**  
Offset: 0x20310

Bit	Field	Type/InitVal	Description
31:0	CPUTimer0Rel	RW 0x0	CPU Timer 0 Reload This field contains the reload value of timer 0, it is used as the reload value in Periodic mode.



**Table 138: CPU Timer 0 Register**  
Offset: 0x20314

Bit	Field	Type/InitVal	Description
31:0	CPUTimer0	RW 0x0	CPU Timer 0 This 32-bit counter is decremented every internal clock cycle of the device. When field <CPUTimer0En> in the CPU Timer Control Register = 0, <CPUTimer0> stops. When <CPUTimer0En> = 1 and <CPUTimer0Auto> in the CPU Timer Control Register = 0, <CPUTimer0> is decremented until it reaches to 0, then it stops. When <CPUTimer0En> = 1 and <CPUTimer0Auto> = 1, <CPUTimer0> is decremented until it reaches to 0, then the field <CPUTimer0Rel> in the CPU 0 Reload Register is reloaded to <CPUTimer0>, and then <CPUTimer0> continues to count. When <CPUTimer0> reaches 0, bit <CPUTimer0IntReq> in the Mbus-L to Mbus Bridge Interrupt Cause Register is set to 1.

**Table 139: CPU Timer1 Reload Register**  
Offset: 0x20318

Bit	Field	Type/InitVal	Description
31:0	CPUTimer1Rel	RW 0x0	CPU Timer 1 Reload See the CPU Timer0 Reload Register.

**Table 140: CPU Timer 1 Register**  
Offset: 0x2031C

Bit	Field	Type/InitVal	Description
31:0	CPUTimer1	RW 0x0	CPU Timer 0 See the CPU Timer 0 Register.

**Table 141: CPU Watchdog Timer Reload Register**  
Offset: 0x20320

Bit	Field	Type/InitVal	Description
31:0	CPUWDTimerLen	RW 0x0	CPU Timer 1 Length See the CPU Timer0 Reload Register.

**Table 142: CPU Watchdog Timer Register**  
**Offset: 0x20324**

Bit	Field	Type/InitVal	Description
31:0	CPUWDTimer	RW 0xFFFFFFFF	CPU Watchdog Timer See the CPU Timer 0 Register.

### A.3.5 L2 Non Cacheable Address

**Table 143: Window0 Base Address Register**  
**Offset: 0x20A00**

Bit	Field	Type/InitVal	Description
0	En	RW 0x0	Window Enable 0 = Disable 1 = Enable
15:1	Reserved_15_1	RO 0x0	Reserved
31:16	Base	RW 0x0	Base address. Used together with the size register to set the address window size and location within the range of the 4-GB space.

**Table 144: Window0 Size Address Register**  
**Offset: 0x20A04**

Bit	Field	Type/InitVal	Description
15:0	Reserved_15_0	RO 0x0	Reserved
31:16	Size	RW 0x0	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64-KB granularity. (A value of 0xFFFF specifies 256 MB.)

**Table 145: Window1 Base Address Register**  
**Offset: 0x20A08**

Bit	Field	Type/InitVal	Description
0	En	RW 0x0	Window Enable 0 = Disable 1 = Enable



**Table 145: Window1 Base Address Register (Continued)**  
Offset: 0x20A08

Bit	Field	Type/InitVal	Description
15:1	Reserved_15_1	RO 0x0	Reserved
31:16	Base	RW 0x0	Base address. Used together with the size register to set the address window size and location within the range of the 4-GB space.

**Table 146: Window1 Size Address Register**  
Offset: 0x20A0C

Bit	Field	Type/InitVal	Description
15:0	Reserved_15_0	RO 0x0	Reserved
31:16	Size	RW 0x0	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64-KB granularity. (A value of 0xFFFF specifies 256 MB.)

**Table 147: Window2 Base Address Register**  
Offset: 0x20A10

Bit	Field	Type/InitVal	Description
0	En	RW 0x0	Window Enable 0 = Disable 1 = Enable
15:1	Reserved_15_1	RO 0x0	Reserved
31:16	Base	RW 0x0	Base address. Used together with the size register to set the address window size and location within the range of the 4-GB space.

**Table 148: Window2 Size Address Register**  
Offset: 0x20A14

Bit	Field	Type/InitVal	Description
15:0	Reserved_15_0	RO 0x0	Reserved

**Table 148: Window2 Size Address Register (Continued)**  
Offset: 0x20A14

Bit	Field	Type/InitVal	Description
31:16	Size	RW 0x0	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64-KB granularity. (A value of 0xFFFF specifies 256 MB.)

**Table 149: Window3 Base Address Register**  
Offset: 0x20A18

Bit	Field	Type/InitVal	Description
0	En	RW 0x0	Window Enable 0 = Disable 1 = Enable
15:1	Reserved_15_1	RO 0x0	Reserved
31:16	Base	RW 0x0	Base address. Used together with the size register to set the address window size and location within the range of the 4-GB space.

**Table 150: Window3 Size Address Register**  
Offset: 0x20A1C

Bit	Field	Type/InitVal	Description
15:0	Reserved_15_0	RO 0x0	Reserved
31:16	Size	RW 0x0	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64-KB granularity. (A value of 0xFFFF specifies 256 MB.)

### A.3.6 Main Interrupt Controller Registers

**Table 151: Main Interrupt Cause Low Register**  
Offset: 0x20200

Bit	Field	Type/InitVal	Description
0	MainHighSum	RO 0x0	Summary of Main Interrupt High Cause register



**Table 151: Main Interrupt Cause Low Register (Continued)**  
Offset: 0x20200

Bit	Field	Type/InitVal	Description
1	Bridge	RO 0x0	Mbus-L to Mbus Bridge interrupt This bit is set when at least one bit is set in AHB to Mbus Bridge Interrupt Cause Register and is not masked by the corresponding bit in Mbus to Mbus Bridge Interrupt Mask Register.
2	Host2CPUDoorbell	RO 0x0	Doorbell interrupt This bit is set when at least one bit is set in Host-to-CPU Doorbell Register and is not masked by the corresponding bit in Host-to-CPU Doorbell Mask Register.
3	CPU2HostDoorbell	RO 0x0	Doorbell interrupt This bit is set when at least one bit is set in CPU-to-Host Doorbell Register and is not masked by the corresponding bit in CPU-to-Host Doorbell Mask Register.
4	Reserved_4	RO 0x0	Reserved
5	Xor0Chan0	RO 0x0	Xor 0 Channel0
6	Xor0Chan1	RO 0x0	Xor 0 Channel1
7	Xor1Chan0	RO 0x0	Xor 1 Channel0
8	Xor1Chan1	RO 0x0	Xor 1 Channel1
9	PEX0INT	RO 0x0	PCI Express port0 INT A, B, C, and D message
10	Reserved	RSVD 0x0	Reserved
11	GbE0Sum	RO 0x0	GbE0 summary
12	GbE0Rx	RO 0x0	GbE0 receive interrupt
13	GbE0Tx	RO 0x0	GbE0 transmit interrupt
14	GbE0Misc	RO 0x0	GbE0 miscellaneous interrupt
15	GbE1Sum	RO 0x0	GbE1 summary
16	GbE1Rx	RO 0x0	GbE1 receive interrupt
17	GbE1Tx	RO 0x0	GbE1 transmit interrupt
18	GbE1Misc	RO 0x0	GbE1 miscellaneous interrupt

**Table 151: Main Interrupt Cause Low Register (Continued)**  
Offset: 0x20200

Bit	Field	Type/InitVal	Description
19	USB0Cnt	RO 0x0	USB0 controller interrupt
20	Reserved	RSVD 0x0	Reserved
21	Sata	RO 0x0	Sata ports interrupt summary of 2 ports
22	SecurityInt	RO 0x0	Security engine completion Interrupt
23	SPIInt	RO 0x0	SPI Interrupt
24	AudioINT	RO 0x0	Audio interrupt
25	Reserved	RSVD 0x0	Reserved
26	TS0Int	RO 0x0	TS0 Interrupt
27	Reserved	RO 0x0	Reserved
28	SDIOInt	RO 0x0	SDIO Interrupt
29	TWSI	RO 0x0	TWSI interrupt
30	AVBInt	RO 0x0	AVB Interrupt
31	TDMInt	RO 0x0	TDM Interrupt

**Table 152: Main IRQ Interrupt Mask Low Register**  
Offset: 0x20204

Bit	Field	Type/InitVal	Description
31:0	Mask	RW 0x0	Mask bit per each cause bit Mask only effects the assertion of the CPU core IRQ interrupt line. It does not effect the setting of bits in the Cause register.



**Table 153: Main FIQ Interrupt Mask Low Register**  
Offset: 0x20208

Bit	Field	Type/InitVal	Description
31:0	Mask	RW 0x0	Mask bit per each cause bit Mask only effects the assertion of CPU core FIQ interrupt line. It does not effect the setting of bits in the Cause register.

**Table 154: Endpoint Interrupt Mask Low Register**  
Offset: 0x2020C

Bit	Field	Type/InitVal	Description
31:0	Mask	RW 0x0	Mask bit per each cause bit Mask only affects the assertion of the interrupt pin. It does not affect the setting of bits in the Cause register. The interrupt pin is asserted in the appropriate interface as defined by bit <EndPointIF> in the CPU Configuration Register.

**Table 155: Main Interrupt Cause High Register**  
Offset: 0x20210

Bit	Field	Type/InitVal	Description
0	Reserved	RSVD 0x0	Reserved
1	Uart0Int	RO 0x0	UART0
2	Uart1Int	RO 0x0	UART1
3	GPIOLo7_0	RO 0x0	GPIO Low[7:0]
4	GPIOLo8_15	RO 0x0	GPIO Low[15:8]
5	GPIOLo16_23	RO 0x0	GPIO Low[23:16]
6	GPIOLo24_31	RO 0x0	GPIO Low[31:24]
7	GPIOHi7_0	RO 0x0	GPIO High[7:0]
8	GPIOHi8_15	RO 0x0	GPIO High[15:8]
9	GPIOHi16_23	RO 0x0	GPIO High[23:16]
10	XOR0Err	RO 0x0	XOR0 error

**Table 155: Main Interrupt Cause High Register (Continued)**  
Offset: 0x20210

Bit	Field	Type/InitVal	Description
11	XOR1Err	RO 0x0	XOR1error
12	PEX0Err	RO 0x0	PCI Express0 error
13	Reserved	RSVD 0x0	Reserved
14	GbE0Err	RO 0x0	GbE port0 error
15	GbE1Err	RO 0x0	GbE port1 error
16	USBErr	RO 0x0	USB error
17	SecurityErr	RO 0x0	Cryptographic engine error
18	AudioErr	RO 0x0	Audio error
19	Reserved	RO 0x0	Reserved
20	Reserved	RO 0x0	Reserved
21	RTCInt	RO 0x0	Real time clock interrupt
22	Reserved	RSVD 0x0	Reserved
31:23	Reserved	RO 0x0	Reserved

**Table 156: Main IRQ Interrupt Mask High Register**  
Offset: 0x20214

Bit	Field	Type/InitVal	Description
31:0	Mask	RW 0x0	Mask bit per each cause bit Mask only effects the assertion of CPU core IRQ interrupt line. It does not effect the setting of bits in the Cause register.



**Table 157: Main FIQ Interrupt Mask High Register**

Offset: 0x20218

Bit	Field	Type/InitVal	Description
31:0	Mask	RW 0x0	Mask bit per each cause bit Mask only effects the assertion of CPU core FIQ interrupt line. It does not effect the setting of bits in the Cause register.

**Table 158: Endpoint Interrupt Mask High Register**

Offset: 0x2021C

Bit	Field	Type/InitVal	Description
31:0	Mask	RW 0x0	Mask bit per each cause bit Mask only affects the assertion of the interrupt pin. It does not affect the setting of bits in the Cause register. The interrupt pin is asserted in the appropriate interface as defined by bit <EndPointIF> in the CPU Configuration Register.

## A.4

## DDR SDRAM Controller Registers

The following table provides a summarized list of all of the DDR SDRAM Controller registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 159: Register Map Table for the DDR SDRAM Controller Registers**

Register Name	Offset	Table and Page
<b>SDRAM Address Decode</b>		
CPU CS Window0 Base Address Register	0x01500	<a href="#">Table 160, p. 390</a>
CPU CS Window0 Size Register	0x01504	<a href="#">Table 161, p. 390</a>
CPU CS Window1 Base Address Register	0x01508	<a href="#">Table 162, p. 390</a>
CPU CS Window1 Size Register	0x0150C	<a href="#">Table 163, p. 391</a>
CPU CS Window2 Base Address Register	0x01510	<a href="#">Table 164, p. 391</a>
CPU CS Window2 Size Register	0x01514	<a href="#">Table 165, p. 392</a>
CPU CS Window3 Base Address Register	0x01518	<a href="#">Table 166, p. 392</a>
CPU CS Window3 Size Register	0x0151C	<a href="#">Table 167, p. 392</a>
<b>SDRAM Control Registers</b>		
SDRAM Configuration Register	0x01400	<a href="#">Table 168, p. 393</a>
DDR Controller Control (Low) Register	0x01404	<a href="#">Table 169, p. 394</a>
SDRAM Timing (Low) Register	0x01408	<a href="#">Table 170, p. 397</a>
SDRAM Timing (High) Register	0x0140C	<a href="#">Table 171, p. 397</a>
SDRAM Address Control Register	0x01410	<a href="#">Table 172, p. 398</a>
SDRAM Open Pages Control Register	0x01414	<a href="#">Table 173, p. 400</a>
SDRAM Operation Register	0x01418	<a href="#">Table 174, p. 400</a>
SDRAM Mode Register	0x0141C	<a href="#">Table 175, p. 401</a>
Extended DRAM Mode Register	0x01420	<a href="#">Table 176, p. 402</a>
DDR Controller Control (High) Register	0x01424	<a href="#">Table 177, p. 403</a>
DDR2 SDRAM Timing (Low) Register	0x01428	<a href="#">Table 178, p. 404</a>
SDRAM Operation Control Register	0x0142C	<a href="#">Table 179, p. 404</a>
SDRAM Interface Mbus Control (Low) Register	0x01430	<a href="#">Table 180, p. 404</a>
SDRAM Interface Mbus Control (High) Register	0x01434	<a href="#">Table 181, p. 405</a>
SDRAM Interface Mbus Timeout Register	0x01438	<a href="#">Table 182, p. 406</a>
DDR2 SDRAM Timing (High) Register	0x0147C	<a href="#">Table 183, p. 406</a>
SDRAM Initialization Control Register	0x01480	<a href="#">Table 184, p. 407</a>
Extended DRAM Mode 2 Register	0x0148C	<a href="#">Table 185, p. 407</a>
Extended DRAM Mode 3 Register	0x01490	<a href="#">Table 186, p. 407</a>
SDRAM ODT Control (Low) Register	0x01494	<a href="#">Table 187, p. 407</a>
SDRAM ODT Control (High) Register	0x01498	<a href="#">Table 188, p. 408</a>
DDR Controller ODT Control Register	0x0149C	<a href="#">Table 189, p. 409</a>
Read Buffer Select Register	0x014A4	<a href="#">Table 190, p. 410</a>
DDR SDRAM Address/Control Pads Calibration Register	0x014C0	<a href="#">Table 191, p. 410</a>
DDR SDRAM DQ Pads Calibration Register	0x014C4	<a href="#">Table 192, p. 411</a>
DDR SDRAM DQS Pads Calibration Register	0x014C8	<a href="#">Table 193, p. 412</a>

## A.4.1 SDRAM Address Decode

**Table 160: CPU CS Window0 Base Address Register**  
 Offset: 0x01500

Bit	Field	Type/InitVal	Description
23:0	Reserved	RO 0x0	Reserved
31:24	Base	RW 0x0	CPU CS Window0 Base Address. Corresponds to CPU core address bits[31:24].

**Table 161: CPU CS Window0 Size Register**  
 Offset: 0x01504

Bit	Field	Type/InitVal	Description
0	En	RW 0x1	Window Enable 0 = Disable 1 = Enable
1	Wr Protect	RW 0x0	Write Protect 0 = Disable 1 = Enable
3:2	Win0_CS	RW 0x0	CS Window0 M_CSn Mapping  For any hit in CS Window0 , the BIU outputs as M_CSn[Win0_CS]. 0 = M_CSn[0] 1 = M_CSn[1] 2 = M_CSn[2] 3 = M_CSn[3]
23:4	Reserved	RO 0xfffff	Reserved
31:24	Size	RW 0x0f	CPU CS Window0 Size Corresponds to Base Address bits[31:24]. Must be programmed from LSb to MSb as a sequence of 1's followed by a sequence of 0's.

**Table 162: CPU CS Window1 Base Address Register**  
 Offset: 0x01508

Bit	Field	Type/InitVal	Description
23:0	Reserved	RO 0x0	Reserved

**Table 162: CPU CS Window1 Base Address Register (Continued)**  
Offset: 0x01508

Bit	Field	Type/InitVal	Description
31:24	Base	RW 0x10	CPU CS Window1 Base Address. Corresponds to CPU core address bits[31:24].

**Table 163: CPU CS Window1 Size Register**  
Offset: 0x0150C

Bit	Field	Type/InitVal	Description
0	En	RW 0x1	Window Enable 0 = Disable 1 = Enable
1	Wr Protect	RW 0x0	Write Protect 0 = Disable 1 = Enable
3:2	Win1_CS	RW 0x1	CS Window1 M_CSn Mapping  For any hit in CS Window1, the BIU outputs as M_CSn[Win1_CS]. 0 = M_CSn[0] 1 = M_CSn[1] 2 = M_CSn[2] 3 = M_CSn[3]
23:4	Reserved	RO 0xfffff	Reserved
31:24	Size	RW 0x0f	CPU CS Window1 Size Corresponds to Base Address bits[31:24]. Must be programmed from LSb to MSb as a sequence of 1's followed by a sequence of 0's.

**Table 164: CPU CS Window2 Base Address Register**  
Offset: 0x01510

Bit	Field	Type/InitVal	Description
23:0	Reserved	RO 0x0	Reserved
31:24	Base	RW 0x20	CPU CS Window2 Base Address. Corresponds to CPU core address bits[31:24].



**Table 165: CPU CS Window2 Size Register**  
Offset: 0x01514

Bit	Field	Type/InitVal	Description
0	En	RW 0x1	Window Enable 0 = Disable 1 = Enable
1	Wr Protect	RW 0x0	Write Protect 0 = Disable 1 = Enable
3:2	Win2_CS	RW 0x2	CS Window2 M_CSn Mapping  For any hit in CS Window2, the BIU outputs as M_CSn[Win2_CS]. 0 = M_CSn[0] 1 = M_CSn[1] 2 = M_CSn[2] 3 = M_CSn[3]
23:4	Reserved	RO 0xfffff	Reserved
31:24	Size	RW 0x0f	CPU CS Window2 Size Corresponds to Base Address bits[31:24]. Must be programmed from LSb to MSb as a sequence of 1's followed by a sequence of 0's.

**Table 166: CPU CS Window3 Base Address Register**  
Offset: 0x01518

Bit	Field	Type/InitVal	Description
23:0	Reserved	RO 0x0	Reserved
31:24	Base	RW 0x30	CPU CS Window3 Base Address Corresponds to CPU core address bits[31:24].

**Table 167: CPU CS Window3 Size Register**  
Offset: 0x0151C

Bit	Field	Type/InitVal	Description
0	En	RW 0x1	Window Enable 0 = Disable 1 = Enable

**Table 167: CPU CS Window3 Size Register (Continued)**  
Offset: 0x0151C

Bit	Field	Type/InitVal	Description
1	Wr Protect	RW 0x0	Write Protect 0 = Disable 1 = Enable
3:2	Win3_CS	RW 0x3	CS Window3 M_CSn Mapping  For any hit in CS Window3 , the BIU outputs as M_CSn[Win3_CS]. 0 = M_CSn[0] 1 = M_CSn[1] 2 = M_CSn[2] 3 = M_CSn[3]
23:4	Reserved	RO 0xfffff	Reserved
31:24	Size	RW 0x0f	CPU CS Window3 Size Corresponds to Base Address bits[31:24]. Must be programmed from LSb to MSb as a sequence of 1's followed by a sequence of 0's.

## A.4.2 SDRAM Control Registers

**Table 168: SDRAM Configuration Register**  
Offset: 0x01400

Bit	Field	Type/InitVal	Description
13:0	Refresh	RW 0x0400	Refresh interval count value.
14	Reserved	RO 0x0	Reserved
15	Reserved	RW 0x0	Reserved Must be 0.
16	P2DW <sub>r</sub>	RW 0x0	CPU to Dram Write buffer policy <b>NOTE:</b> If any of the following device configurations are used, clear this bit: - CAS Latency = 3 - PCLK to HCLK ratio = 1:1 - PCLK to HCLK ratio = 3:2 0 = To Store: Write buffer configure to store & forward mode 1 = To Cut: Write buffer configure to cut through mode
17	RegDIMM	RW 0x0	Enable Registered DIMM or Equivalent Sampling Logic <b>NOTE:</b> Even if on-board DRAM devices are used, this register can be used to buffer DRAM address/control signals. For that configuration, set the <RegDIMM> bit to 1. 0 = Non buffered 1 = Registered: Registered DIMM or equivalent sampling logic



**Table 168: SDRAM Configuration Register (Continued)**  
Offset: 0x01400

Bit	Field	Type/InitVal	Description
19:18	Reserved_19_18	RW 0x0	Reserved
23:20	Reserved_23_20	RO 0x0	Reserved
24	SRMode	RW 0x1	Self Refresh Mode  0 = Entered self refresh: Once entered self refresh, exit only upon power on reset. 1 = Exit self refresh: Exit self refresh, when new DRAM access is requested.
25	Reserved_25	RW 0x1	Reserved Must be 0x1.
29:26	Reserved_29_26	RO 0x0	Reserved
31:30	Reserved	RSVD 0x1	Reserved

**Table 169: DDR Controller Control (Low) Register**  
Offset: 0x01404

Bit	Field	Type/InitVal	Description
3:0	Reserved	RO 0x0	Reserved
4	2T	RW 0x0	2T mode Useful when interfacing heavy DRAM load (multiple DRAM banks) at high frequency. 0 = Single Cycle: Address and command are driven for a single cycle. 1 = Two Cycle: Address and command are driven for two cycles (qualified with M_CPU CS that is still driven for one cycle).
5	SRClk	RW 0x0	Clock driven upon self refresh <b>NOTE:</b> If using clock buffers for distributing the clock to the DRAM devices, the clock must be kept driven during Self Refresh mode unless these buffers can tolerate clock gating. 0 = Driven clock: Clock is kept driven during self refresh. 1 = Gated clock: Clock is gated during self refresh.

**Table 169: DDR Controller Control (Low) Register (Continued)**  
Offset: 0x01404

Bit	Field	Type/InitVal	Description
6	CtrlPos	RW 0x1	<p>Address/Control Output Timing  <b>NOTE:</b> When interfacing on-board DRAM devices, it may be useful to use the falling edge of clock. This helps prevent DRAM hold time violations.</p> <p>Bits [11:9] of DDR_Controller_control (High) register (0x1424) along with bit[6] of DDR_Controller_control (Low ) register (0x1404) should be configured according to the DIMM Configuration table above.</p> <p>0 = Falling edge: On falling edge of clock  1 = Rising edge: On rising edge of clock</p>
11:7	Reserved	RO 0x0	Reserved
12	Reserved	RW 0x1	Reserved Must be 0x1.
13	Reserved	RW 0x1	Reserved Must be 0x1.
14	DPDEn	RW 0x0	<p>SDRAM interface input buffers power down enable</p> <p><b>NOTE:</b> Even when set to 0, the input buffer is turned off during reset and during self refresh (whenever address/control pins output enables are inactive).</p> <p>0 = Disable: Input buffer is always enabled (never powered down).  1 = Enable: Input buffer is turned off (powered down), whenever there is no active read transaction.</p>
17:15	Reserved	RO 0x0	Reserved
18	LockEn	RW 0x1	CPU Lock Transaction Enable/Disable 0 = Disable 1 = Enable
19	Reserved	RO 0x0	Reserved



**Table 169: DDR Controller Control (Low) Register (Continued)**  
Offset: 0x01404

Bit	Field	Type/InitVal	Description
23:20	SBOutDel	RW 0x3	<p>Number of cycles from first CAS cycle of burst-read to the assertion of M_STARTBURST signal.</p> <p><b>NOTE:</b> Even if using a DRAM device on board, the registered DIMM can be replaced with equivalent sampling logic.</p> <p>0 = 2 cycles: Recommended for CL 3, when SbOut_DE_Sel is enabled. 1 = 2.5 cycles: Recommended for CL 3, when SbOut_DE_Sel is disabled. 2 = 3 cycles: Recommended for CL 4 or registered DIMM (or equivalent sampling logic) with CL 3, when SbOut_DE_Sel is enabled. 3 = 3.5 cycles: Recommended for CL 4 or registered DIMM (or equivalent sampling logic) with CL 3, when SbOut_DE_Sel is disabled. 4 = 4 cycles: Recommended for CL 5 or registered DIMM (or equivalent sampling logic) with CL 4, when SbOut_DE_Sel is enabled. 5 = 4.5 cycles: Recommended for CL 5 or registered DIMM (or equivalent sampling logic) with CL 4, when SbOut_DE_Sel is disabled. 6 = 5 cycles: Recommended for CL 6 or registered DIMM (or equivalent sampling logic) with CL 5, when SbOut_DE_Sel is enabled. 7 = 5.5 cycles: Recommended for CL 6 or registered DIMM (or equivalent sampling logic) with CL 5, when SbOut_DE_Sel is disabled. 8 = 6 cycles: Recommended for registered DIMM (or equivalent sampling logic) with CL 6, when SbOut_DE_Sel is enabled. 9 = 6.5 cycles: Recommended for registered DIMM (or equivalent sampling logic) with CL 6, when SbOut_DE_Sel is disabled. 11 = 7 cycles: Recommended for registered DIMM (or equivalent sampling logic) with CL 7, when SbOut_DE_Sel is enabled. 12 = 7.5 cycles: Recommended for registered DIMM (or equivalent sampling logic) with CL 7, when SbOut_DE_Sel is disabled.</p>
27:24	SBInDel	RW 0x6	<p>Number of sample stages on M_STARTBURST_IN.</p> <p>Can be configured from CL to CL + 4 for unbuffered DIMM, or CL + 1 to CL + 5 for registered DIMM or equivalent sampling logic.</p> <p>For frequencies between 200 to 399 MHz, set to CL+ 3 for unbuffered DIMM or CL+ 4 for registered DIMM or equivalent sampling logic.</p> <p>For 400 MHz frequency, set to CL+ 4 for unbuffered DIMM or CL+ 5 for registered DIMM or equivalent sampling logic.</p>
30:28	Reserved	RW 0x3	Reserved Must be 0x3.
31	SBOut_DE_Sel	RW 0x0	Add 1/4 cycle delay to Start Burst Out. 0 = Normal: No additional delay. 1 = Add_Delay: Add 1/4 cycle delay.

**Table 170: SDRAM Timing (Low) Register**  
Offset: 0x01408

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The default values corresponds to working with DDR533 (4-4-4). Change these timing parameters according to DRAM type and operating frequency.			
3:0	tRAS	RW 0xB	Minimum Row Active Time (active to precharge) Value 0 means one cycle, value 1 means two cycles, etc. See also Bit 20.
7:4	tRCD	RW 0x3	Activate to Command Value 0 means one cycle, value 1 means two cycles, etc.
11:8	tRP	RW 0x3	Precharge Period (precharge to active) Value 0 means one cycle, value 1 means two cycles, etc.
15:12	tWR	RW 0x3	Write Command to Precharge Value 0 means one cycle, value 1 means two cycles, etc.
19:16	tWTR	RW 0x1	Write Command to Read Command Value 0 means one cycle, value 1 means two cycles, etc.
20	Extended tRAS	RW 0x0	Extended tRAS [4], tRAS[4:0] = {[20],[3:0]} Minimum Row Active Time (active to precharge) Value 0 means one cycle, value 1 means two cycles, etc.
23:21	Reserved	RO 0x0	Reserved
27:24	tRRD	RW 0x1	Activate Bank A to Activate Bank B Value 0 means one cycle, value 1 means two cycles, etc.
31:28	tRTP	RW 0x1	Read Command to Precharge. Value 0 means one cycle, value 1 means two cycles, etc.

**Table 171: SDRAM Timing (High) Register**  
Offset: 0x0140C

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The default values corresponds to working with DDR533 (4-4-4). Change these timing parameters according to DRAM type and operating frequency.			
6:0	tRFC	RW 0x34	Refresh Command Period Value 0 means one cycle, value 1 means two cycles, etc.



**Table 171: SDRAM Timing (High) Register (Continued)**  
Offset: 0x0140C

Bit	Field	Type/InitVal	Description
8:7	tR2R	RW 0x0	Minimum Gap Between DRAM Read Accesses This timing parameter is not part of the JEDEC standard. It is used to prevent contention between read data drive from two different DRAM devices. 0x0 = One cycle 0x1 = Two cycles 0x2, 0x3 = Reserved
10:9	tR2W_W2R	RW 0x0	Minimum Gap Between DRAM Read and Write Accesses This timing parameter is not part of the JEDEC standard. Used to prevent contention between read and write data driven from two different devices (same parameter for read after write and write after read). 0x0 = One cycle 0x1 = Two cycles 0x2, 0x3 = Reserved
12:11	tW2W	RW 0x0	Minimum Gap between Write accesses to different DRAM devices. Value 0 means no gap, value 1 means a one-cycle gap, etc.
31:13	Reserved	RO 0x0	Reserved

**Table 172: SDRAM Address Control Register**  
Offset: 0x01410

Bit	Field	Type/InitVal	Description
1:0	CS0Width	RW 0x0	SDRAM Address Select Determines what address bits to drive on M_A[14:0] and M_BA[2:0] during activate and command phases (row and column addresses). <b>NOTE:</b> See DDR SDRAM Address Multiplex. 0 = Reserved 1 = x16
3:2	CS0Size	RW 0x1	SDRAM Device Configuration Effects SDRAM row and column address bits multiplexing. See DDR SDRAM Addressing. 0 = 2 Gb: when supported 1 = 256 Mb 2 = 512 Mb 3 = 1 Gb
5:4	CS1Width	RW 0x0	SDRAM Address Select Determines what address bits to drive on M_DA[14:0] and M_BA[2:0] during activate and command phases (row and column addresses). <b>NOTE:</b> See DDR SDRAM Address Multiplex. 0 = Reserved 1 = x16

**Table 172: SDRAM Address Control Register (Continued)**  
Offset: 0x01410

Bit	Field	Type/InitVal	Description
7:6	CS1Size	RW 0x1	SDRAM Device Configuration Effects SDRAM row and column address bits multiplexing. See DDR SDRAM Addressing. 0 = 2 Gb: when supported 1 = 256 Mb 2 = 512 Mb 3 = 1 Gb
9:8	CS2Width	RW 0x0	SDRAM Address Select Determines what address bits to drive on M_DA[14:0] and M_BA[2:0] during activate and command phases (row and column addresses). <b>NOTE:</b> See DDR SDRAM Address Multiplex. 0 = Reserved 1 = x16
11:10	CS2Size	RW 0x1	SDRAM Device Configuration Effects SDRAM row and column address bits multiplexing. See DDR SDRAM Addressing. 0 = 2 Gb: when supported 1 = 256 Mb 2 = 512 Mb 3 = 1 Gb
13:12	CS3Width	RW 0x0	SDRAM Address Select Determines what address bits to drive on M_DA[14:0] and M_BA[2:0] during activate and command phases (row and column addresses). <b>NOTE:</b> See DDR SDRAM Address Multiplex. 0 = Reserved 1 = x16
15:14	CS3Size	RW 0x1	SDRAM Device Configuration Effects SDRAM row and column address bits multiplexing. See DDR SDRAM Addressing. 0 = 2 Gb: when supported 1 = 256 Mb 2 = 512 Mb 3 = 1 Gb
16	CS0AddrSel	RW 0x0	Address Select Mode for CS0 Defines how the address bits driven by the requesting unit to the DRAM controller are translated to row and column address bits. <b>NOTE:</b> See DDR SDRAM Addressing.
17	CS1AddrSel	RW 0x0	Address Select Mode for CS1 Defines how the address bits driven by the requesting unit to the DRAM controller are translated to row and column address bits. <b>NOTE:</b> See DDR SDRAM Addressing.



**Table 172: SDRAM Address Control Register (Continued)**  
Offset: 0x01410

Bit	Field	Type/InitVal	Description
18	CS2AddrSel	RW 0x0	Address Select Mode for CS2 Defines how the address bits driven by the requesting unit to the DRAM controller are translated to row and column address bits. <b>NOTE:</b> See DDR SDRAM Addressing.
19	CS3AddrSel	RW 0x0	Address Select Mode for CS3 Defines how the address bits driven by the requesting unit to the DRAM controller are translated to row and column address bits. <b>NOTE:</b> See DDR SDRAM Addressing.
31:20	Reserved	RO 0x0	Reserved.

**Table 173: SDRAM Open Pages Control Register**  
Offset: 0x01414

Bit	Field	Type/InitVal	Description
0	OPEn	RW 0x0	Open Page Enable 0 = Open page: DDR Controller keeps the corresponding bank page open whenever it can. 1 = Close page: DDR Controller always closes the page at the end of the transaction.
31:1	Reserved	RO 0x0	Reserved.

**Table 174: SDRAM Operation Register**  
Offset: 0x01418

Bit	Field	Type/InitVal	Description
3:0	Cmd	SC 0x0	DRAM Mode Select Setting Cmd results in DDR Controller execution of the required command to the DRAM. The DDR Controller then resets Cmd to the default 0x0 value. 0x6, 0xA-0xF = Reserved 0 = Normal: Normal SDRAM Mode 1 = Precharge: Precharge all banks command. 2 = Refresh: Refresh all banks command 3 = MRS: Mode Register Set (MRS) command 4 = EMRS: Extended Mode Register Set (EMRS) command 5 = NOP: NOP command 7 = SelfRefresh: Enter self refresh 8 = EMRS2: EMRS2 command 9 = EMRS3: EMRS3 command

**Table 174: SDRAM Operation Register (Continued)**  
Offset: 0x01418

Bit	Field	Type/InitVal	Description
31:4	Reserved	RO 0x0	Reserved.

**Table 175: SDRAM Mode Register**  
Offset: 0x0141C

Bit	Field	Type/InitVal	Description
2:0	BL	RW 0x2	Burst Length Must be set to 0x2 (BL = 4).
3	BT	RW 0x0	Burst Type/InitVal Must be set to 0.
6:4	CL	RW 0x4	CAS Latency 3 = CL_3 4 = CL_4 5 = CL_5 6 = CL_6
7	TM	RW 0x0	Test Mode 0 = Normal operation 1 = Test mode
8	DLLRst	RW 0x0	Reset DLL 0 = Normal operation 1 = Reset DLL
11:9	WR	RW 0x3	Write recovery for auto-precharge <b>NOTE:</b> This device does not support auto-precharge. Must be 0x3.
12	PD	RW 0x0	Active power down exit time Must be 0x0. Active power down is not supported. 0 = Fast exit 1 = Slow exit
14:13	Reserved	RW 0x0	Reserved
31:15	Reserved	RO 0x0	Reserved



**Table 176: Extended DRAM Mode Register**  
**Offset: 0x01420**

Bit	Field	Type/InitVal	Description
0	DLLDis	RW 0x0	DRAM DLL Enable 0 = Enable 1 = Disable
1	DS	RW 0x0	DRAM Drive Strength 0 = Normal 1 = Reduced
2	Rtt[0]	RW 0x0	Rtt[1:0] is ODT Control. 0 = ODT disable 1 = 75 ohm termination 2 = 150 ohm termination 3 = 50 ohm termination
5:3	AL	RW 0x0	Additive Latency Must be 0x0.
6	Rtt[1]	RW 0x0	See Rtt[0]. <b>NOTE:</b> Extended to bit2.
9:7	OCD	RW 0x0	OCD Program Must be 0x0.
10	DQS	RW 0x0	Differential DQS Enable 0 = Enabled 1 = Disabled
11	RDQS	RW 0x0	Read DQS Enable Must be 0 (no read DQS).
12	Qoff	RW 0x0	DRAM Output Buffer Enable 0 = Enabled 1 = Disabled
14:13	Reserved	RW 0x0	Reserved
31:15	Reserved	RO 0x0	Reserved

**Table 177: DDR Controller Control (High) Register**  
**Offset: 0x01424**

Bit	Field	Type/InitVal	Description
2:0	Reserved	RW 0x7	Reserved - Must be 0x7.
3	Mbus Burst Chop	RW 0x1	Mbus Burst Chop Enables chop of a long Mbus access to DRAM on 32B boundaries (giving the CPU a chance to interfere in the middle and access the DRAM). <b>NOTE:</b> Only applicable to 64b DRAM interface (with 32b interface, split is always enabled). 0 = Enable: Enable Mbus burst chop. 1 = Disable: Disable Mbus burst chop.
6:4	Reserved	RSVD 0x7	Reserved
7	Reserved	RW 0x0	Reserved Must be 0.
8	P2D Lat	RW 0x0	Adds a sample stage on the CPU to SDRAM write path. For frequencies of 266 MHz and above, this bit must be set to 0x1. 0 = Disable: No sample stage. 1 = Enable: Add sample stage.
9	AddHalfcc2DO	RW 0x0	Adds a half clock cycle on data out.  Bits [11:9] of DDR_Controller_control (High) register (0x1424) along with Bit[6] of DDR_Controller_control (Low ) register (0x1404) are configured according to the DIMM configuration table above. 0 = Normal: No addition 1 = Add: Add a half cycle.
10	PupZeroSkewSel	RW 0x0	Control PUP's clock to address/control signals skew select.  Bits [11:9] of DDR_Controller_control (High) register (0x1424) along with Bit[6] of DDR_Controller_control (Low ) register (0x1404) are configured according to the DIMM configuration table above. 0 = Normal: 1/4 cc skew 1 = ZeroSkew: 0 cc skew
11	WrMeshDelay	RW 0x0	Add 1/4 cc delay on write Mesh.  Bits [11:9] of DDR_Controller_control (High) register (0x1424) along with bit[6] of DDR_Controller_control (Low ) register (0x1404) are configured according to the DIMM configuration table above. 0 = Normal: No Delay. 1 = Add: Add 1/4 cc delay.
23:12	Reserved	RSVD 0xF	Reserved
31:24	Reserved	RO 0x0	Reserved

**Table 178: DDR2 SDRAM Timing (Low) Register**

Offset: 0x01428

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The default values corresponds to working with DDR533 (4-4-4). Change these timing parameters according to DRAM type and operating frequency.			
3:0	Reserved	RO 0x0	Reserved
7:4	tODT_ON_RD	RW 0x1	Number of cycles from Read command to assertion of the M_ODT signal. Value 0 means same cycle as read command, value 1 means one cycle later, etc. Value depends on SDRAM CL and tAOND timing parameters. For CL = 3 and tAOND = 2, set tODT_ON to 0.
11:8	tODT_OFF_RD	RW 0x4	Number of cycles from Read command to de-assertion of the M_ODT signal. Value depends on DRAM CL and tAOFD timing parameters. For CL = 3 and tAOFD = 2.5, set tODT_OFF to 0x3.
15:12	tODT_ON_CTL_RD	RW 0x4	Number of cycles from Read command to assertion of the internal ODT signal to the SDRAM controller I/O buffer. Same as tODT_ON.
19:16	tODT_OFF_CTL_RD	RW 0x7	Number of cycles from Read command to de-assertion of the internal ODT signal to the SDRAM controller I/O buffer. Same as tODT_OFF.
31:20	Reserved	RO 0x0	Reserved.

**Table 179: SDRAM Operation Control Register**

Offset: 0x0142C

Bit	Field	Type/InitVal	Description
1:0	CS	RW 0x0	DRAM chip select. Defines to which DRAM bank to issue an EMRS1 command. This is useful for setting different ODT values to different DRAM banks. 0 = CSn[0] 1 = CSn[1] 2 = CSn[2] 3 = CSn[3]
31:2	Reserved	RO 0x0	Reserved

**Table 180: SDRAM Interface Mbust Control (Low) Register**

Offset: 0x01430

Bit	Field	Type/InitVal	Description
3:0	Arb0	RW 0x0	Pizza Arbiter slice 0 unit ID.

**Table 180: SDRAM Interface Mbus Control (Low) Register (Continued)**  
Offset: 0x01430

Bit	Field	Type/InitVal	Description
7:4	Arb1	RW 0x1	Pizza Arbiter slice 1 unit ID.
11:8	Arb2	RW 0x2	Pizza Arbiter slice 2 unit ID.
15:12	Arb3	RW 0x3	Pizza Arbiter slice 3 unit ID.
19:16	Arb4	RW 0x4	Pizza Arbiter slice 4 unit ID.
23:20	Arb5	RW 0x5	Pizza Arbiter slice 5 unit ID.
27:24	Arb6	RW 0x6	Pizza Arbiter slice 6 unit ID.
31:28	Arb7	RW 0x7	Pizza Arbiter slice 7 unit ID

**Table 181: SDRAM Interface Mbus Control (High) Register**  
Offset: 0x01434

Bit	Field	Type/InitVal	Description
3:0	Arb8	RW 0x8	Pizza Arbiter slice 8 unit ID.
7:4	Arb9	RW 0x9	Pizza Arbiter slice 9 unit ID.
11:8	Arb10	RW 0xA	Pizza Arbiter slice 10 unit ID.
15:12	Arb11	RW 0xB	Pizza Arbiter slice 11 unit ID.
19:16	Arb12	RW 0xC	Pizza Arbiter slice 12 unit ID.
23:20	Arb13	RW 0xD	Pizza Arbiter slice 13 unit ID.
27:24	Arb14	RW 0xE	Pizza Arbiter slice 14 unit ID.
31:28	Arb15	RW 0xF	Pizza Arbiter slice 15 unit ID.



**Table 182: SDRAM Interface Mbus Timeout Register**  
Offset: 0x01438

Bit	Field	Type/InitVal	Description
7:0	Timeout	RW 0xFF	Mbus Arbiter Timeout Preset Value <b>NOTE:</b> Must keep value of 0xFF as long as Timeout counter is disabled.
15:8	Reserved	RO 0x0	Reserved
16	TimeoutDis	RW 0x1	Mbus Arbiter Timer enable 0 = Enable 1 = Disable
17	Unit Power save	RW 0x0	DDR Controller Power Save Management enable 0 = Disable 1 = Enable
31:18	Reserved	RO 0x0	Reserved

**Table 183: DDR2 SDRAM Timing (High) Register**  
Offset: 0x0147C

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The default values corresponds to working with DDR533 (4-4-4). Change these timing parameters according to DRAM type and operating frequency.			
3:0	tODT_ON_WR	RW 0x1	Number of cycles from the Write command to assertion of the M_ODT signal. A value of 0 means one cycle before write command, a value of 1 means the same cycle as write command, a value of 2 means one cycle later ,etc. The value depends on DRAM CL and tAOND timing parameters. For CL = 4 and tAOND = 2, set tODT_ON to 0x1.
7:4	tODT_OFF_WR	RW 0x4	Number of cycles from Write command to de-assertion of the M_ODT signal. The value depends on DRAM CL and tAOFD timing parameters. For CL = 4 and tAOFD = 2.5, set tODT_OFF to 0x4.
11:8	tODT_ON_CTL_WR	RW 0x4	Number of cycles from Write command to assertion of the internal ODT signal to the DRAM controller I/O buffer. Same as tODT_ON
15:12	tODT_OFF_CTL_W_R	RW 0x7	Number of cycles from Write command to de-assertion of the internal ODT signal to the DRAM controller I/O buffer. Same as tODT_OFF.
31:16	Reserved	RO 0x0	Reserved

**Table 184: SDRAM Initialization Control Register**  
Offset: 0x01480

Bit	Field	Type/InitVal	Description
0	InitEn	SC 0x0	Initialization Enable DRAM initialization sequence starts upon setting <InitEn> to 1. DRAM controller clears the bit upon initialization completion. <b>NOTE:</b> The initialization sequence is performed once. Setting <InitEn> again after initialization has been completed has no effect.
31:1	Reserved	RO 0x0	Reserved

**Table 185: Extended DRAM Mode 2 Register**  
Offset: 0x0148C

Bit	Field	Type/InitVal	Description
14:0	EMRS2	RW 0x0	EMRS2 value
31:15	Reserved	RO 0x0	Reserved

**Table 186: Extended DRAM Mode 3 Register**  
Offset: 0x01490

Bit	Field	Type/InitVal	Description
14:0	EMRS3	RW 0x0	EMRS3 value
31:15	Reserved	RO 0x0	Reserved

**Table 187: SDRAM ODT Control (Low) Register**  
Offset: 0x01494

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The 88F6180 has a single ODT pin, M_ODT. Bits [19:16] control write transactions for pin M_ODT, and bits [3:0] control read transactions for that pin.			
3:0	ODT0Rd	RW 0x0	M_ODT[0] control for read transactions. Bit[0] If set to 1, M_ODT[0] is asserted during read from DRAM CS0. Bit[1] If set to 1, M_ODT[0] is asserted during read from DRAM CS1. Bit[2] If set to 1, M_ODT[0] is asserted during read from DRAM CS2. Bit[3] If set to 1, M_ODT[0] is asserted during read from DRAM CS3.



**Table 187: SDRAM ODT Control (Low) Register (Continued)**  
Offset: 0x01494

Bit	Field	Type/InitVal	Description
7:4	ODT1Rd	RW 0x0	M_ODT[1] control for read transactions. Bit[0] If set to 1, M_ODT[1] is asserted during read from DRAM CS0. Bit[1] If set to 1, M_ODT[1] is asserted during read from DRAM CS1. Bit[2] If set to 1, M_ODT[1] is asserted during read from DRAM CS2. Bit[3] If set to 1, M_ODT[1] is asserted during read from DRAM CS3.
11:8	Reserved	RW 0x0	Reserved Must be 0.
15:12	Reserved	RW 0x0	Reserved Must be 0.
19:16	ODT0Wr	RW 0x0	M_ODT[0] control for write transactions. Bit[0] If set to 1, M_ODT[0] is asserted during write to DRAM CS0. Bit[1] If set to 1, M_ODT[0] is asserted during write to DRAM CS1. Bit[2] If set to 1, M_ODT[0] is asserted during write to DRAM CS2. Bit[3] If set to 1, M_ODT[0] is asserted during write to DRAM CS3.
23:20	ODT1Wr	RW 0x0	M_ODT[1] control for write transactions. Bit[0] If set to 1, M_ODT[1] is asserted during write to DRAM CS0. Bit[1] If set to 1, M_ODT[1] is asserted during write to DRAM CS1. Bit[2] If set to 1, M_ODT[1] is asserted during write to DRAM CS2. Bit[3] If set to 1, M_ODT[1] is asserted during write to DRAM CS3.
27:24	Reserved	RW 0x0	Reserved Must be 0.
31:28	Reserved	RW 0x0	Reserved Must be 0.

**Table 188: SDRAM ODT Control (High) Register**  
Offset: 0x01498

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The 88F6180 has a single ODT pin, M_ODT. Bits [3:0] enable that pin.			
1:0	ODT0En	RW 0x0	M_ODT[0] Enable 0 = Controlled by register: M_ODT[0] assertion/de-assertion is controlled by SDRAM ODT Control Low register. 1 = Never active: M_ODT[0] is never active. 2 = Controlled by register1: M_ODT[0] assertion/de-assertion is controlled by SDRAM ODT Control Low register. 3 = Always active: M_ODT[0] is always active.

**Table 188: SDRAM ODT Control (High) Register (Continued)**  
Offset: 0x01498

Bit	Field	Type/InitVal	Description
3:2	ODT1En	RW 0x0	M_ODT[1] Enable  0 = Controlled by register: M_ODT[1] assertion/de-assertion is controlled by SDRAM ODT Control Low register. 1 = Never active: M_ODT[1] is never active. 2 = Controlled by register1: M_ODT[1] assertion/de-assertion is controlled by SDRAM ODT Control Low register. 3 = Always active: M_ODT[1] is always active.
5:4	Reserved	RW 0x0	Reserved Must be 0.
7:6	Reserved	RW 0x0	Reserved Must be 0.
31:8	Reserved	RO 0x0	Reserved

**Table 189: DDR Controller ODT Control Register**  
Offset: 0x0149C

Bit	Field	Type/InitVal	Description
3:0	ODTRd	RW 0x0	DDR Controller I/O buffer ODT control for read transactions. Bit[0] If set to 1, internal ODT is asserted during read from DRAM bank 0. Bit[1] If set to 1, internal ODT is asserted during read from DRAM bank 1. Bit[2] If set to 1, internal ODT is asserted during read from DRAM bank 2. Bit[3] If set to 1, internal ODT is asserted during read from DRAM bank 3.
7:4	ODTWr	RW 0x0	DDR Controller I/O buffer ODT control for write transactions. Bit[0] If set to 1, internal ODT is asserted during write to DRAM bank 0. Bit[1] If set to 1, internal ODT is asserted during write to DRAM bank 1. Bit[2] If set to 1, internal ODT is asserted during write to DRAM bank 2. Bit[3] If set to 1, internal ODT is asserted during write to DRAM bank 3.
9:8	ODTEn	RW 0x0	DDR Controller I/O buffer ODT Enable. 0 = Controlled by fields: Internal ODT assertion/de-assertion is controlled by ODTRd/ODTWr fields. 1 = Never active: Internal ODT is never active. 2 = Controlled by fields1: Internal ODT assertion/de-assertion is controlled by ODTRd/ODTWr fields. 3 = Active: Internal ODT is always active.
11:10	DQ_ODTSel	RW 0x0	DDR Controller M_DQ, M_DM, and M_DQS I/O buffer ODT Select. 0 = Turned off 1 = 150 ohm 2 = 75 ohm 3 = 50 ohm



**Table 189: DDR Controller ODT Control Register (Continued)**  
Offset: 0x0149C

Bit	Field	Type/InitVal	Description
13:12	STARTBURST_ODT Sel	RW 0x2	DDR Controller M_STARTBURST_IN I/O buffer ODT Select. 0 = Turned off 1 = 150 ohm 2 = 75 ohm 3 = 50 ohm
14	STARTBURST_ODT En	RW 0x1	DDR Controller M_STARTBURST_IN ODT Enable 0 = Disable 1 = Enable
15	ODT Unit	RW 0x1	DDR IO ODT Unit 0 = Drive ODT calibration values 1 = Use ODT Block
20:16	ODT DrvN	RW 0x0	Pad N channel driving strength for ODT
25:21	ODT DrvP	RW 0x0	Pad P channel driving strength for ODT
31:26	Reserved	RO 0x0	Reserved

**Table 190: Read Buffer Select Register**  
Offset: 0x014A4

Bit	Field	Type/InitVal	Description
15:0	RdBuff	RW 0x0	Read buffer assignment per unit. When a bit is set to 0, the corresponding unit is assigned to receive read data from Read buffer 0. When set to 1, the corresponding unit is assigned to receive read data from Read buffer 1.  Bit[0] corresponds to unit ID 0x0, bit[1] corresponds to unit ID 0x1, etc. For example, setting to 0x80 stands for unit ID 0x7 receives read data from read buffer 1, while the rest of the units receive data from read buffer 0.
31:16	Reserved	RO 0x0	Reserved

**Table 191: DDR SDRAM Address/Control Pads Calibration Register**  
Offset: 0x014C0

Bit	Field	Type/InitVal	Description
5:0	DrvN	RW 0x1A	Pad N channel driving strength

**Table 191: DDR SDRAM Address/Control Pads Calibration Register (Continued)**  
Offset: 0x014C0

Bit	Field	Type/InitVal	Description
11:6	DrvP	RW 0x1A	Pad P channel driving strength
13:12	Reserved	RW 0x3	Reserved - Must be 0x3.
15:14	Reserved	RO 0x0	Reserved
16	TuneEn	RW 0x1	Setting to 1 enables the auto-calibration of pad driving strength.
22:17	LockN	RO 0x0	When auto-calibration is enabled, this field represents the final locked value of the N channel driving strength.
28:23	LockP	RO 0x0	When auto-calibration is enabled, this field represents the final locked value of the P channel driving strength.
30:29	Reserved	RO 0x0	Reserved.
31	WrEn	RW 0x0	Write Enable 0 = Read only: Register is read only (except for bit[31]). 1 = Writeable: Register is writable.

**Table 192: DDR SDRAM DQ Pads Calibration Register**  
Offset: 0x014C4

Bit	Field	Type/InitVal	Description
5:0	DrvN	RW 0x1A	Pad N channel driving strength
11:6	DrvP	RW 0x1A	Pad P channel driving strength
13:12	Reserved	RW 0x3	Reserved. Must be 0x3.
15:14	Reserved	RO 0x0	Reserved
16	TuneEn	RW 0x1	Setting to 1 enables the auto-calibration of pad driving strength.
22:17	LockN	RO 0x0	When auto-calibration is enabled, this field represents the final locked value of the N channel driving strength.



**Table 192: DDR SDRAM DQ Pads Calibration Register (Continued)**  
Offset: 0x014C4

Bit	Field	Type/InitVal	Description
28:23	LockP	RO 0x0	When auto-calibration is enabled, this field represents the final locked value of the P channel driving strength.
30:29	Reserved	RO 0x0	Reserved.
31	WrEn	RW 0x0	Write Enable 0 = Read only: Register is read only (except for bit[31]). 1 = Writeable: Register is writable.

**Table 193: DDR SDRAM DQS Pads Calibration Register**  
Offset: 0x014C8

Bit	Field	Type/InitVal	Description
5:0	DrvN	RW 0x1A	Pad N channel driving strength
11:6	DrvP	RW 0x1A	Pad P channel driving strength
13:12	Reserved	RW 0x3	Reserved - Must be 0x3.
15:14	Reserved	RO 0x0	Reserved
16	TuneEn	RW 0x1	Setting to 1 enables the auto-calibration of pad driving strength.
22:17	LockN	RO 0x0	When auto-calibration is enabled, this field represents the final locked value of the N channel driving strength.
28:23	LockP	RO 0x0	When auto-calibration is enabled, this field represents the final locked value of the P channel driving strength.
30:29	Reserved	RO 0x0	Reserved.
31	WrEn	RW 0x0	Write Enable 0 = Read only: Register is read only (except for bit[31]). 1 = Writeable: Register is writable.

## A.5

## Time Division Multiplexing (TDM) Unit Registers

The following table provides a summarized list of all of the Time Division Multiplexing (TDM) Unit registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 194: Register Map Table for the Time Division Multiplexing (TDM) Unit Registers**

Register Name	Offset	Table and Page
<b>SPI Control Registers</b>		
CSU System Clock Prescaler Register	0xD3100	<a href="#">Table 195, p. 414</a>
CSU Global Control Register	0xD3104	<a href="#">Table 196, p. 415</a>
SPI Control Register	0xD3108	<a href="#">Table 197, p. 415</a>
Codec Access Command Low Register	0xD3130	<a href="#">Table 198, p. 415</a>
Codec Access Command High Register	0xD3134	<a href="#">Table 199, p. 416</a>
Codec Registers Access Control	0xD3138	<a href="#">Table 200, p. 416</a>
Codec Read Data Register	0xD313C	<a href="#">Table 201, p. 417</a>
Codec Registers Access Control1	0xD3140	<a href="#">Table 202, p. 417</a>
<b>TDM Control Registers</b>		
PCM Control Register	0xD0000	<a href="#">Table 203, p. 418</a>
Channel Time Slot Control Register	0xD0004	<a href="#">Table 204, p. 420</a>
Channel 0 Delay Control Register	0xD0008	<a href="#">Table 205, p. 421</a>
Channel 1 Delay Control Register	0xD000C	<a href="#">Table 206, p. 422</a>
Channel 0/1 Enable and Disable Register (n=0–1)	Channel0: 0xD0010, Channel1: 0xD0020	<a href="#">Table 207, p. 422</a>
Channel 0/1 Buffer Ownership Register (n=0–1)	Channel0: 0xD0014, Channel1: 0xD0024	<a href="#">Table 208, p. 423</a>
Channel 0 Transmit Data Start Address Register	0xD0018	<a href="#">Table 209, p. 423</a>
Channel 0 Receive Data Start Address Register	0xD001C	<a href="#">Table 210, p. 424</a>
Channel 1 Transmit Data Start Address Register	0xD0028	<a href="#">Table 211, p. 424</a>
Channel 1 Receive Data Start Address Register	0xD002C	<a href="#">Table 212, p. 424</a>
Channel 0/1 Total Sample Count Register (n=0–1)	Channel0: 0xD0030, Channel1: 0xD0034	<a href="#">Table 213, p. 425</a>
Number of Time Slots Register	0xD0038	<a href="#">Table 214, p. 425</a>
TDM PCM Clock Rate Divisor Register	0xD003C	<a href="#">Table 215, p. 426</a>
Interrupt Event Mask Register	0xD0040	<a href="#">Table 216, p. 426</a>
Interrupt Status Mask Register	0xD0048	<a href="#">Table 217, p. 426</a>
Interrupt Reset Selection Register	0xD004C	<a href="#">Table 218, p. 427</a>
Interrupt Status Register	0xD0050	<a href="#">Table 219, p. 427</a>
Dummy Data for Dummy RX Write Register	0xD0054	<a href="#">Table 220, p. 428</a>
Miscellaneous Control Register	0xD0058	<a href="#">Table 221, p. 429</a>
Channel 0/1 Transmit Data Current Address Register (n=0–1)	Channel0: 0xD0060, Channel1: 0xD0068	<a href="#">Table 222, p. 429</a>
Channel 0/1 Receive Data Current Address Register (n=0–1)	Channel0: 0xD0064, Channel1: 0xD006C	<a href="#">Table 223, p. 429</a>
Current Time Slot Register	0xD0070	<a href="#">Table 224, p. 429</a>
TDM Revision Register	0xD0074	<a href="#">Table 225, p. 429</a>

**Table 194: Register Map Table for the Time Division Multiplexing (TDM) Unit Registers (Continued)**

Register Name	Offset	Table and Page
TDM Channel 0/1 Debug Register (n=0–1)	Channel0: 0xD0078, Channel1: 0xD007C	<a href="#">Table 226, p. 430</a>
TDM DMA Abort Register 1	0xD0080	<a href="#">Table 227, p. 431</a>
TDM DMA Abort Register 2	0xD0084	<a href="#">Table 228, p. 431</a>
TDM Channel0 Wideband Delay Control Register	0xD0088	<a href="#">Table 229, p. 431</a>
TDM Channel 1 Wideband Delay Control Register	0xD008C	<a href="#">Table 230, p. 432</a>
<b><i>TDM-to-Mbus Bridge Registers</i></b>		
SPI Interface Output Enable Control Register	0xD4000	<a href="#">Table 231, p. 432</a>
TDM-Mbus Configuration Register	0xD4010	<a href="#">Table 232, p. 433</a>
Window0 Control Register	0xD4030	<a href="#">Table 233, p. 433</a>
Window0 Base Register	0xD4034	<a href="#">Table 234, p. 433</a>
Window1 Control Register	0xD4040	<a href="#">Table 235, p. 434</a>
Window1 Base Register	0xD4044	<a href="#">Table 236, p. 434</a>
Window2 Control Register	0xD4050	<a href="#">Table 237, p. 434</a>
Window2 Base Register	0xD4054	<a href="#">Table 238, p. 435</a>
Window3 Control Register	0xD4060	<a href="#">Table 239, p. 435</a>
Window3 Base Register	0xD4064	<a href="#">Table 240, p. 436</a>
TDM-Mbus Configuration 1 Register	0xD4070	<a href="#">Table 241, p. 436</a>

## A.5.1 SPI Control Registers

**Table 195: CSU System Clock Prescaler Register**

Offset: 0xD3100

Bit	Field	Type/InitVal	Description
7:0	SclockDivLow	RW 0x10	<p>SPI/3-Wire, 4-Wire System Clock Prescaler Low</p> <p>This field is used to determine SCLK frequency when the &lt;LO_SPEED_CLK&gt; field in the CODEC Registers Access Control Register is set to 0.</p> <p>According to the following equation:</p> $\text{SCLK} = (\text{TCLK}) / (\text{SclockDivLow})$ <p><b>NOTE:</b> The value of this field must be even; a value of 0 is not allowed.</p>
15:8	SclockDivHigh	RW 0x2A	<p>SPI/3-Wire, 4-Wire System Clock Prescaler High</p> <p>This field is used to determine SCLK frequency when &lt;LO_SPEED_CLK&gt; field in the CODEC Registers Access Control Register is set to 1.</p> <p>According to the following equation:</p> $\text{SCLK} = (\text{TCLK}) / (\text{SclockDivHigh})$ <p><b>NOTE:</b> The value of this field must be even; a value of 0 is not allowed.</p>
31:16	Reserved	RSVD 0x0	Reserved

**Table 196: CSU Global Control Register**  
**Offset: 0xD3104**

Bit	Field	Type/InitVal	Description
0	CODEC Enable	RW 0x1	SLIC/CODEC registers read/write enable. 0 = CSU CODEC register read/write: CSU CODEC register read/write interface is not enabled 1 = CSU CODEC register read/write: CSU CODEC register read/write interface is enabled and 4 wire SPI interface (SCLK, CCS, SDO and SRWB) carries CODEC signals. Normal CSU SPI mode is disabled
7:1	Reserved	RW 0x40	Reserved Must not change default value.
31:8	Reserved	RSVD 0x0	Reserved

**Table 197: SPI Control Register**  
**Offset: 0xD3108**

Bit	Field	Type/InitVal	Description
9:0	Reserved	RW 0x0	Reserved. Must be 0x0.
10	SPIStat	RW 0x0	SPI Status Indicates the beginning of SPI transmission. Actually SPI bus transaction occurs when accessing data register to read or write SPI data. Writing 1 in this bit starts the CODEC register read/write operation. When the operation is complete, the device resets this bit to 0. The CPU polls this bit. When it becomes 0, it can start the next CODEC read/write transaction.
31:11	Reserved	RSVD 0x0	Reserved

**Table 198: Codec Access Command Low Register**  
**Offset: 0xD3130**

Bit	Field	Type/InitVal	Description
7:0	BYTE0	RW 0x00	First byte to be sent to the codec from the device on SDO serially. The 8-bit value programmed in this field is sent to the codec first. Depending on Codec register read/write protocol, the software must program valid values.
15:8	BYTE1	RW 0x00	Second byte to be sent to the codec from the device on SDO serially. The 8-bit value programmed in this field is sent to the codec after <BYTE0>. Depending on Codec register read/write protocol, the software must program valid values.
31:16	Reserved	RSVD 0x0	Reserved

**Table 199: Codec Access Command High Register**  
**Offset: 0xD3134**

Bit	Field	Type/InitVal	Description
7:0	BYTE2	RW 0x00	Third byte to be sent to the codec from the device on SDO serially. The 8-bit value programmed in this field is sent to the codec after <BYTE1> field in the Codec Register Access Command Low Register. Depending on the Codec register read/write protocol, the software must program valid values.
15:8	BYTE3	RW 0x00	Fourth byte to be sent to the codec from the device on SDO serially. The 8-bit value programmed in this field is sent to the codec after <BYTE2>. Depending on the Codec register read/write protocol, the software must program valid values.
31:16	Reserved	RSVD 0x0	Reserved

**Table 200: Codec Registers Access Control**  
**Offset: 0xD3138**

Bit	Field	Type/InitVal	Description
1:0	BYTES_TO_XFER	RW 0x0	Number of bytes that are sent to CODEC for register read/write from the device on SDO. 0 = 1 byte: 1 byte are sent to codec. 1 = 2 bytes: 2 bytes are sent to codec. 2 = 3 bytes: 3 bytes are sent to codec. 3 = 4 bytes: 4 bytes are sent to codec.
2	LSB_MSB	RW 0x0	Bytes sent to or received from the codec are MSB first order or LSB first order.
3	RD_WR_	RW 0x0	Register read/write operation to the codec
4	BYTE_TO_READ	RW 0x0	Number of bytes to read from the codec in case of Codec register read operation. <b>NOTE:</b> Upon a write access, must be set to 0.
5	LO_SPEED_CLK	RW 0x0	This bit determines if the frequency of the SCLK is determined according to field <SclockDivLow> or <SclockDivHigh> field in the CSU System Clock Prescaler Register.

**Table 200: Codec Registers Access Control (Continued)**  
Offset: 0xD3138

Bit	Field	Type/InitVal	Description
15:6	CS_HIGH_CNT_VA_L_READ	RW 0x0	<p>This 10-bit value indicates the codec chip select (CCS) high time before the read data comes out of the codec. Some codecs require that after sending command and address bytes (for Codec register read operation), CCS should be high for &gt; 1.5 us before the read data comes out of the codec.</p> <p>The software can program this value depending upon individual CODEC requirements. This value represents the number of SLOW BUS CLOCK cycles for which CCS will remain high before read data from the CODEC register comes out.</p> <p>0x0 = CCS will remain high for 16 SLOW BUS CLOCK cycles</p> <p>Any other value = CCS will remain high for the specified number of SLOW BUS CLOCK cycles (for example, for 1 SLOW BUS CLOCK cycle, set it to 0x1)</p>
31:16	Reserved	RSVD 0x0	Reserved

**Table 201: Codec Read Data Register**  
Offset: 0xD313C

Bit	Field	Type/InitVal	Description
7:0	CODEC_READ_DA_TA_LO	RO 0x0	For a Codec register read operation, after sending command and address to the codec, the codec sends out the read data on the SRWB signal. This field stores the low byte [7:0] of the read data. This field has valid value only when the <SPISat> field in the SPI Control Register is set to 0.
15:8	CODEC_READ_DA_TA_HI	RO 0x0	For a Codec register read operation, after sending command and address to the codec, the codec sends out the read data on the SRWB signal. This field stores the high byte [15:8] of the read data. This field has a valid value only when the <SPISat> field in the SPI Control Register is set to 0.
31:16	Reserved	RSVD 0x0	Reserved

**Table 202: Codec Registers Access Control1**  
Offset: 0xD3140

Bit	Field	Type/InitVal	Description
9:0	CS_HIGH_CNT_VA_L_WRITE	RW 0x0	<p>This 10-bit value indicates the codec chip select (CCS) high time between the bytes sent to the codec. Some codecs require that the CCS should go high for &gt; 1.5 us after each byte is sent to the codec (including address, command, write data bytes). The software can program this value depending upon individual codec requirements. This value represents the number of slow bus clocks for which CCS will remain high between every byte sent to the codec.</p> <p>For any other value = CCS will remain high for the specified number of Slow Bus Clock cycles (for example, for 1 Slow Bus Clock cycle, set it to 0x1)</p> <p>0 = CCS: CCS will remain high for 16 Slow Bus Clock cycles between each byte sent to the codec.</p>



**Table 202: Codec Registers Access Control1 (Continued)**  
Offset: 0xD3140

Bit	Field	Type/InitVal	Description
31:10	Reserved	RSVD 0x0	Reserved.

## A.5.2 TDM Control Registers

**Table 203: PCM Control Register**  
Offset: 0xD0000

Bit	Field	Type/InitVal	Description
0	MstrPclkn	RW 0x1	TDM is Master/Slave of PCM clock 1 = Use external PCM clock. The TDM will get PCM clock from some external source. 0 = Use internally generated PCM clock. See TDM PCM Clock Rate Divisor Register for details of PCM clock supported by TDM.
1	MasterFsn	RW 0x1	TDM is Master/Slave of Frame Sync. 1 = Use external FS. The TDM will get FS from some external source. 0 = Use internally generated FS.
2	DataPol	RW 0x0	Data Reference Polarity is referred as the clock edge where the data is sampled, asserted or de-asserted, by the external codec. The TDM drives the data at the opposite edge from what is sampled by the codec. Data Reference Polarity: This field can be changed without also changing the <FsPol> field.
3	FsPol	RW 0x0	Frame Sync Reference Polarity is referred as the clock edge where the FS is sampled, asserted or de-asserted, by the external codec. The TDM drives the FS at the opposite edge from what is sampled by codec. Frame sync Reference Polarity: This field cannot be changed without also changing the <DataPol> field.
4	InvFs	RW 0x0	Inverted Frame Sync When this bit is set, FS is interpreted as Active Low. Only valid when using short FS.
5	LongFs	RW 0x0	In Frame Sync Master mode, the TDM generates frame sync (FS). This bit defines the type of frame sync generated.
6	PcmSampleSize	RW 0x0	PCM Sample Size 0 = 1byte: Each sample is 1 byte and occupies one time slot. 1 = Linear: Use linear mode. In this mode, each sample is 2 bytes and occupies two time slots.
7	Reserved	RW 0x0	Reserved. Always write 0 to this bit.

**Table 203: PCM Control Register (Continued)**  
Offset: 0xD0000

Bit	Field	Type/InitVal	Description
8	CH0DlyEn	RW 0x0	Channel 0 Delay Control Enable 0 = Disable: Disable the use of the Channel 0 delay control <CH0_RX_DLY> and <CH0_TX_DLY> fields in the Channel 0 Delay Control Register. The values in <CH0_TX_TSLOT> and <CH0_RX_TSLOT> fields in the Channel Time Slot Control Register will be used instead for receive and transmit of channel 0 data. 1 = Enable: Enable the use of the Channel 0 delay control <CH0_RX_DLY> and <CH0_TX_DLY> fields in the Channel 0 Delay Control Register. The values in the <CH0_TX_TSLOT> and <CH0_RX_TSLOT> fields in the Channel Time Slot Control Register will not be used.
9	CH1DlyEn	RW 0x0	Channel 1 Delay Control Enable 0 = Disable: Disable the use of the Channel 1 delay control <CH1_RX_DLY> and <CH1_TX_DLY> fields in the Channel 1 Delay Control Register. The values in <CH1_TX_TSLOT> and <CH1_RX_TSLOT> fields in the Channel Time Slot Control Register will be used instead for receive and transmit of channel 1 data. 1 = Enable: Enable the use of the Channel 1 delay control <CH1_RX_DLY> and <CH1_TX_DLY> fields in the Channel 1 Delay Control Register. The values in the <CH1_TX_TSLOT> and <CH1_RX_TSLOT> fields in the Channel Time Slot Control Register will not be used.
10	CH0QualEn	RW 0x0	Channel 0 Rx and Tx Qualifier Enable 0 = Disable: Disable Qualifier for Channel 0 Rx and Tx. The external codec has time slot (or delay) programming registers to configure when to send and receive data. 1 = Enable: Enable Qualifier for Channel 0 Rx and Tx. There is one qualifier for receive and one for transmit. The qualifier becomes active for transmit or receive depending on the time slot (or delay) programmed for the Channel 0 receive and transmit. The <CH0_QUAL_TYP> field controls the duration for which the qualifier for the Channel 0 receive and transmit remains active.
11	CH1QualEn	RW 0x0	Channel 1 Rx and Tx Qualifier Enable 0 = Disable: Disable Qualifier for Channel 1 Rx and Tx. The external codec has time slot (or delay) programming registers to configure when to send and receive data. 1 = Enable: Enable Qualifier for Channel 1 Rx and Tx. There is one qualifier for receive and one for transmit. The qualifier becomes active for transmit or receive depending on the time slot (or delay) programmed for the Channel 1 receive and transmit. The <CH1_QUAL_TYP> field controls the duration for which the qualifier for the Channel 1 receive and transmit remains active.
12	QualPol	RW 0x0	Qualifier Reference Polarity is referred to as the clock edge where the qualifier is sampled, asserted or de-asserted, by the external codec. The design drives the qualifier at the opposite edge from what is sampled by the external codec.
13	CH0QualTyp	RW 0x0	This bit controls the way the qualifier for channel 0 receive and transmit will stay active.



**Table 203: PCM Control Register (Continued)**  
Offset: 0xD0000

Bit	Field	Type/InitVal	Description
14	CH1QualTyp	RW 0x0	This controls the way the qualifier for channel 1 receive and transmit will stay active.
15	DAA_CS_CTRL	RW 0x0	CCS[1:0] select When using two SPI chip select signals rather than chaining of SLIC/codec devices, this bit selects which of the two chip select is now active. This is useful for interfacing a SI3050 that does not support chaining.
16	CH0WBand	RW 0x0	<b>NOTE:</b> This bit cannot be set in GCI mode (i.e., when <GciMode> of this register is set to 1 <b>NOTE:</b> When using Wideband mode, <CH0DlyEn> in this register must be set to 1.
17	CH1WBand	RW 0x0	<b>NOTE:</b> This bit cannot be set in GCI mode, i.e., <GciMode> of this register is set to 1. <b>NOTE:</b> When using Wideband mode, <CH1DlyEn> in this register must be set to 1.
30:18	Reserved	RSVD 0x0	Reserved
31	PerfBit	RW 0x1	This bit controls how many locations (1 location is 32-bit wide) are popped or pushed in or out of the FIFO interfacing to the Mbus. This might be used when the Mbus is busy, as it helps the TDM design to be more tolerant of Mbus latencies. 0 = One location: Always one location of FIFO are read or written. 1 = Two locations: Always two locations of FIFO are read or written.

**Table 204: Channel Time Slot Control Register**  
Offset: 0xD0004

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> TDM supports a maximum 128 time slot (at 8.192 MHz PCM clock). Transmit and receive of Channel 0 and Channel 1 can be assigned to any one of the 128 time slot. The time slot assignment for Channel 0 transmit/receive and Channel 1 transmit/receive must be mutually exclusive. If these assignments overlap, contention will occur on the PCM bus. Channel 0 Tx/Rx and Channel 1 Tx/Rx can be assigned a time slot or a delay value (from assertion of frame sync). The delay is programmed via the Channel x Delay Control Register / and enabled via the PCM Control Register. If the Channel x Delay Control Register value is enabled in the PCM Control Register then the Channel time slot control register value is not used for the Tx and Rx of that channels. The TDM Tx time slot is CODEC Rx time slot and vice versa.			
7:0	CH0RxTSlot	RW 0x00	Receive (RX from CODEC to TDM) time slot for channel 0 See "Number of Time Slots Register" for available time slots at a particular PCM clock frequency. If the time slot value programmed is more than the available time slots, the TDM will not receive data. If <PCM_SAMPLE_SIZE> is set to 1, two time slots are used for RX data.

**Table 204: Channel Time Slot Control Register (Continued)**  
**Offset: 0xD0004**

Bit	Field	Type/InitVal	Description
15:8	CH0TxTSlot	RW 0x00	Transmit (TX from TDM to CODEC) time slot for channel 0 See table 16 for available time slots at particular PCM clock frequency. If the time slot value programmed is more than the available time slots, the TDM will not transmit data. If <PCM_SAMPLE_SIZE> is set to 1, two time slots are used for TX data.
23:16	CH1RxTSlot	RW 0x02	Receive (RX from CODEC to TDM) time slot for channel 1 See the "Number of Time Slots Register" for available time slots at a particular PCM clock frequency. If the time slot value programmed is more than the available time slots, the TDM will not receive data. If <PCM_SAMPLE_SIZE> is set to 1, two time slots are used for RX data.
31:24	CH1TxTSlot	RW 0x02	Transmit (TX from TDM to CODEC) time slot for channel 1 See the "Number of Time Slots Register" for available time slots at a particular PCM clock frequency. If the time slot value programmed is more than the available time slots, the TDM will not transmit data. If <PCM_SAMPLE_SIZE> field in the PCM Control Register is set to 1, two time slots are used for TX data.

**Table 205: Channel 0 Delay Control Register**  
**Offset: 0xD0008**

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The Channel 0 receive and transmit delay (from frame sync assertion) can be controlled by this register. The delay is defined in the number of PCLKs from the assertion of frame sync. The value in this register overrides the value in the <CH0TxTSlot> and <CH0RxTSlot> fields in the Channel Time Slot Control Register if the <CH0DlyEn> field in the PCM Control Register is set to 1.			
9:0	CH0RxDly	RW 0x00	Number of PCLK to wait before starting to receive data for Channel 0 from assertion of Frame Sync (FS). After the start of receive, 8-bit or 16-bit data will be received, depending on the <PCM_SAMPLE_SIZE>. This bit also controls the time when the first PCM data receive occurs in Wideband mode in CH0.
15:10	Reserved	RSVD 0x0	Reserved
25:16	CH0TxDly	RW 0x00	Number of PCLK to wait before starting to transmit data for Channel 0 from the assertion of FS. After the start of transmission, 8-bit or 16-bit data will be transmitted, depending on the <PCM_SAMPLE_SIZE> field in the PCM Control Register. This bit also controls the time when the first PCM data transmit is done in Wideband mode in CH0.
31:26	Reserved	RSVD 0x0	Reserved



**Table 206: Channel 1 Delay Control Register**  
Offset: 0xD000C

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The Channel1 receive and transmit delay (from frame sync assertion) can be controlled by this register. The delay is defined in the number of PCLKs from the assertion of frame sync. The value in this register overrides the value in the CH1TxTSlot and CH1RxTSlot fields in the Channel Time Slot Control Register if the <CH1DlyEn> field in the PCM Control Register is set to 1.			
9:0	CH1RxDly	RW 0x10	Number of PCLK to wait before starting to receive data for Channel 1 from assertion of FS. After the start of receive, 8-bit or 16-bit data will be received depending on the <PCM_SAMPLE_SIZE>. This bit also controls the time when the first PCM data receive occurs in Wideband mode in CH1.
15:10	Reserved	RSVD 0x0	Reserved
25:16	CH1TxDly	RW 0x10	Number of PCLK to wait before starting to transmit data for Channel 1 from the assertion of FS. After the start of transmission, 8-bit or 16-bit data will be transmitted depending on the <PCM_SAMPLE_SIZE> field in the PCM Control Register. This bit also controls the time when the first PCM data transmit is done in Wideband mode in CH1.
31:26	Reserved	RSVD 0x0	Reserved

**Table 207: Channel 0/1 Enable and Disable Register (n=0–1)**  
Offset: Channel0: 0xD0010, Channel1: 0xD0020

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> After the Channel 0/1 RX/TX bits are enabled the design starts to receive or transmit data in the programmed time slot (or delay) after the next FS is detected. Before programming the different enable bits in the Channel0 Delay Control Register or Channel1 Delay Control Register, TDM channel 0/1 should be initialized. See details of initialization in PCM data flow for TX and RX.  The software can disable the transmit operation of any channel, any time after enabling it. The disable only takes affect after all the PCM data in all the allocated TX buffers in memory is sent out to the CODEC. In case of receive, if the channel is disabled, the design stops to receive data from the PCM interface and starts padding the remaining PCM samples from the Dummy Data for Dummy RX Write Register. The transmit enable or receive enable is automatically disabled if an underflow (in case of transmit), or an overflow (in case of receive) situation is detected by the hardware. Refer to TDM underflow/overflow handling section for more detail. If the channel is not enabled, the TDM transmit 0/1 in allocated a TX time slot and does not receive any data in the RX time slot.			
0			
0	CHnRxEn	RW 0x0	Channel 0/1 Receive Enable 0 = Disable: Disable channel 0/1 receive. 1 = Enable: Enable channel 0/1 receive.
7:1	Reserved	RSVD 0x0	Reserved
8	CHnTxEn	RW 0x0	Channel 0/1 Transmit Enable 0 = Disable: Disable channel 0/1 transmit. 1 = Enable: Enable channel 0/1 transmit.

**Table 207: Channel 0/1 Enable and Disable Register (n=0–1) (Continued)**  
Offset: Channel0: 0xD0010, Channel1: 0xD0020

Bit	Field	Type/InitVal	Description
31:9	Reserved	RSVD 0x0	Reserved

**Table 208: Channel 0/1 Buffer Ownership Register (n=0–1)**  
Offset: Channel0: 0xD0014, Channel1: 0xD0024

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> This register defines the ownership of the Transmit Data Start Address for Channel 0/1 Register and Receive Data Start Address for Channel 0/1 Register.			
0	RX_DMA_ST_ADD_R_OWN_CHx	RW 0x0	This register defines the ownership of the Receive Data Start Address for Channel x Register. .
7:1	Reserved	RSVD 0x0	Reserved
8	TX_DMA_ST_ADD_R_OWN_CHx	RW 0x0	This register defines the ownership of the Transmit Data Start Address for Channel x Register.
31:9	Reserved	RSVD 0x0	Reserved

**Table 209: Channel 0 Transmit Data Start Address Register**  
Offset: 0xD0018

Bit	Field	Type/InitVal	Description
31:0	TX_DMA_ST_ADD_R_CH0	RW 0xC0000000	This register contains the start address of the allocated buffer for Channel0 (CH0) transmit data. Software should check the <TX_DMA_ST_ADDR_OWN_CHx> field in the Buffer Ownership for Channel x Register before programming this register. <b>NOTE:</b> Start address must be 32-byte aligned.



**Table 210: Channel 0 Receive Data Start Address Register**  
Offset: 0xD001C

Bit	Field	Type/InitVal	Description
31:0	RX_DMA_ST_ADD_R_CH0	RW 0xC0001000	This register contains the start address of the allocated buffer for CH0 receive data. Software should check the <RX_DMA_ST_ADDR_own_CHx> field in the Buffer Ownership for Channel x Register before programming this register. <b>NOTE:</b> Start address must be 32-byte aligned.

**Table 211: Channel 1 Transmit Data Start Address Register**  
Offset: 0xD0028

Bit	Field	Type/InitVal	Description
31:0	TX_DMA_ST_ADD_R_CH1	RW 0xC0002000	This register contains the start address of the allocated buffer for Channel1 (CH1) transmit data. Software should check the <TX_DMA_ST_ADDR_own_CHx> field in the Buffer Ownership for Channel x Register before programming this register. <b>NOTE:</b> Start address must be 32-byte aligned.

**Table 212: Channel 1 Receive Data Start Address Register**  
Offset: 0xD002C

Bit	Field	Type/InitVal	Description
31:0	RX_DMA_ST_ADD_R_CH1	RW 0xC0003000	This register contains the start address of the allocated buffer for CH1 receive data. Software should check the <RX_DMA_ST_ADDR_own_CHx> field in the Buffer Ownership for Channel x Register before programming this register. <b>NOTE:</b> Start address must be 32-byte aligned.

**Table 213: Channel 0/1 Total Sample Count Register (n=0–1)**  
Offset: Channel0: 0xD0030, Channel1: 0xD0034

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> This register sets the total number of sample (8-bit or 16-bit each) in the allocated buffer to be transmitted or received by Channel 0/1 (if enabled) and the number of samples after which the TDM will generate an interrupt to the CPU indicating the hardware is close to the end of total sample count value to make the next buffer ready.			
			The sample count is programmed before enabling the channel and cannot be changed until the channel is disabled. The value is copied internally after the channel is enabled and decremented by one with every FS detection (as the PCM data TX and RX starts). The hardware uses the interrupt sample count value (<CHx_INT_SMPL_CNT> field in the Total Sample Count Channel x Register) as a watermark for the availability of new buffers to continue to move data across the PCM interface.
7:0	CH0/1_TOTAL_SMPL_CNT	RW 0x50	<p>Total Sample count.  This register needs to be programmed if the buffer size is different than 80 samples. This can be programmed for up to 30 ms sample width (240 samples).</p> <p><b>NOTE:</b> The buffer size depends upon whether each PCM sample is 1 byte or 2 bytes long. As an example, if this bit is set to 80 and each PCM sample is 1 byte long, the buffer size will be 80 bytes while if each PCM sample is 2 byte long it will be 160 bytes.</p>
15:8	CH0/1_INT_SMPL_CNT	RW 0x40	Interrupt Sample count can be defined as the number of samples transferred to CODEC or received from CODEC after which the hardware will notify the software to create the next buffer. This register should be programmed with a value which is less than the value in the <CHx_TOTAL_SMPL_CNT> field in the Total Sample Count Channel x Register. The value should take into consideration the Mbus latencies and software latencies to make the next buffer available. See the detailed explanation in the "Tx Data Flow" and "Rx Data Flow".
31:16	Reserved	RSVD 0x0	Reserved

**Table 214: Number of Time Slots Register**  
Offset: 0xD0038

Bit	Field	Type/InitVal	Description
7:0	NO_OF_TS	RW 0x80	<p>This register contains the number of time slots (bytes) in a frame. This value is used to generate Frame sync signals in Frame sync master mode of TDM. Default is 128 considering the bit clock is 8.192 MHz. In Frame sync slave mode, this register does not mean anything.</p> <p>Number of time slots = PCLK (MHz)</p> <p><b>NOTE:</b> Any other values are reserved</p> <ul style="list-style-type: none"> <li>4 = 256 kHz</li> <li>8 = 512 kHz</li> <li>10 = 1.024 MHz</li> <li>20 = 2.048 MHz</li> <li>40 = 4.096 MHz</li> <li>80 = 8.192 MHz</li> </ul>
31:8	Reserved	RSVD 0x0	Reserved



**Table 215: TDM PCM Clock Rate Divisor Register**

Offset: 0xD003C

Bit	Field	Type/InitVal	Description
<p><b>NOTE:</b> When the TDM is configured to be the clock master, this register sets the PCM clock baud rate. The BBU_PCM_CLK divided by the value in this register and gives the frequency of PCM Clock. The divided ratios support only one hot encoded values (implemented in BBU).</p>			
7:0	PCM_CLK_DIV	RW 0x20	<p>PCM clock Rate Divisor PCM_CLK_DIV = PCM CLOCK Frequency <b>NOTE:</b> Any other values are reserved 1 = 256 kHz 2 = 512 kHz 4 = 1.024 MHz 8 = 0248 MHz 10 = 4.096 MHz 20 = 8.192 MHz</p>
31:8	Reserved	RSVD 0x0	Reserved

**Table 216: Interrupt Event Mask Register**

Offset: 0xD0040

Bit	Field	Type/InitVal	Description
17:0	IEMR	RW 0x3FFF	Interrupt event mask register
31:18	Reserved	RW 0x0	Reserved

**Table 217: Interrupt Status Mask Register**

Offset: 0xD0048

<p><b>NOTE:</b> This is a dummy register to facilitate the Interrupt latching in the Interrupt Status Register.</p>			
17:0	ISM	RW 0x3FFF	<p>Interrupt mask register 0 = The interrupt is masked: The interrupt is masked. To check the interrupt, poll the Interrupt Status Register (TDM_INT_STATUS). 1 = No interrupt is masked: The corresponding bit in the Interrupt Status Register causes an interrupt to CPU. By default, no interrupt is masked.</p>
31:18	Reserved	RW 0x0	Reserved

**Table 218: Interrupt Reset Selection Register**  
**Offset: 0xD004C**

Bit	Field	Type/InitVal	Description
17:0	IRS	RW 0x0	Interrupt reset select register configures the way the ISR bit will clear after being set.
31:18	Reserved	RW 0x0	Reserved

**Table 219: Interrupt Status Register**  
**Offset: 0xD0050**

Bit	Field	Type/InitVal	Description
0	OFLOW_CH0_INT	RW 0x0	Receive Overflow in CH0
1	UFLOW_CH0_INT	RW 0x0	Transmit Underflow in CH0
2	OFLOW_CH1_INT	RW 0x0	Receive Overflow in CH1
3	UFLOW_CH1_INT	RW 0x0	Transmit Underflow in CH1
4	SCOCH0_RX_INT	RW 0x0	Sample count over as per <CHx_INT_SMPL_CNT> field in the Total Sample Count Channel x Register value for Channel 0
5	SCOCH0_TX_INT	RW 0x0	Sample count over as per <CHx_INT_SMPL_CNT> field in the Total Sample Count Channel x Register value for Channel 0
6	SCOCH1_RX_INT	RW 0x0	Sample count over as per <CHx_INT_SMPL_CNT> field in the Total Sample Count Channel x Register value for Channel 1
7	SCOCH1_TX_INT	RW 0x0	Sample count over as per <CHx_INT_SMPL_CNT> field in the Total Sample Count Channel x Register value for Channel 1
8	CH0_RX_IDLE	RW 0x0	Channel 0 receive has gone from active state to idle This occurs when the total sample count Channel 0 is reached and channel has been closed by programming the <CHxRxEn> field in the Channel x Enable and Disable Register. This interrupt will not be generated when the Channel 0 get closed by itself because of overflow.
9	CH0_TX_IDLE	RW 0x0	Channel 0 transmit has gone from active state to idle This occurs when the total sample count Channel 0 is reached, no more buffers are pending and channel has been closed by programming the <CHxTxEn> field in the Channel x Enable and Disable Register. This interrupt will not be generated when the Channel 0 get closed by itself because of underflow.



**Table 219: Interrupt Status Register (Continued)**  
Offset: 0xD0050

Bit	Field	Type/InitVal	Description
10	CH1_RX_IDLE	RW 0x0	Channel 1 receive has gone from active state to idle. This occurs when the total sample count Channel 1 is reached and channel has been closed by programming the <CHxRxEn> field in the Channel x Enable and Disable Register. This interrupt will not be generated when the Channel 1 get closed by itself because of overflow.
11	CH1_TX_IDLE	RW 0x0	Channel 1 transmit has gone from active state to idle. This occurs when the total sample count Channel 1 is reached, no more buffers are pending and channel has been closed by programming the <CHxTxEn> field in the Channel x Enable and Disable Register. This interrupt will not be generated when the channel 1 get closed by itself because of underflow.
12	RXFIFO0_FULL_INT	RW 0x0	RXFIFO full for Channel 0 interrupt
13	TXFIFO0_EMPTY_INT	RW 0x0	TXFIFO empty for Channel 0 interrupt
14	RXFIFO1_FULL_INT	RW 0x0	RXFIFO full for Channel 1 interrupt
15	TXFIFO1_EMPTY_INT	RW 0x0	TXFIFO empty for Channel 1 interrupt
16	DMA_ABORTED_INT	RW 0x0	TDM DMA has aborted. This is caused by illegal memory space access by TDM.
17	CODEC_INT	RW 0x0	CODEC interrupt
31:18	Reserved	RSVD 0x0	Reserved

**Table 220: Dummy Data for Dummy RX Write Register**  
Offset: 0xD0054

Bit	Field	Type/InitVal	Description
31:0	DUMMY_DATA	RW 0x00	Contains the value of the padding that will be used by the TDM when RX is switched off by programming the <CHxRxEn> field in the Channel x Enable and Disable Register. The TDM will stop receiving data from PCM interface and fill the remaining allocated buffer with this dummy data.

**Table 221: Miscellaneous Control Register**  
Offset: 0xD0058

Bit	Field	Type/InitVal	Description
0	CODEC_RST	RW 0x0	Controls the CODEC reset signal which is output from TDM to CODEC. The CODEC reset is active low reset which should be asserted for sufficient time so that CODEC can sample PCM clock and Frame Sync to lock its internal PLL (See individual CODEC specs for more details).
31:1	Reserved	RSVD 0x0	Reserved

**Table 222: Channel 0/1 Transmit Data Current Address Register (n=0–1)**  
Offset: Channel0: 0xD0060, Channel1: 0xD0068

Bit	Field	Type/InitVal	Description
31:0	TX_DMA_CUR_AD DR_CH0/1	RO 0x00	Contains the current address of the allocated buffer for CH0/1 transmit DMA. Used for debug purposes.

**Table 223: Channel 0/1 Receive Data Current Address Register (n=0–1)**  
Offset: Channel0: 0xD0064, Channel1: 0xD006C

Bit	Field	Type/InitVal	Description
31:0	RX_DMA_CUR_AD DR_CH0/1	RO 0x0	Contains the current address of the allocated buffer for CH0/1 receive DMA. Used for debug purposes.

**Table 224: Current Time Slot Register**  
Offset: 0xD0070

Bit	Field	Type/InitVal	Description
7:0	CUR_TS	RO 0x0	Current Time Slot Identifies the time slot that is currently active. This can be used by FW as a check for precise time for changing control register of TDM while it is in operation.
31:8	Reserved	RSVD 0x0	Reserved

**Table 225: TDM Revision Register**  
Offset: 0xD0074

Bit	Field	Type/InitVal	Description
15:0	TDM_IP_REV	RO 0x0201	TDM IP revision



**Table 225: TDM Revision Register (Continued)**  
Offset: 0xD0074

Bit	Field	Type/InitVal	Description
31:16	Reserved	RSVD 0x0	Reserved

**Table 226: TDM Channel 0/1 Debug Register (n=0–1)**  
Offset: Channel0: 0xD0078, Channel1: 0xD007C

Bit	Field	Type/InitVal	Description
1:0	RX_WR_CTRL_STA TE_CH0/1_RX	RO 0x0	RX read control state machine
4:2	PCM_RX_IF_ STATE_CH0/1_RX	RO 0x0	RX PCM state machine state
5	GNT_CH0/1_RX	RO 0x0	CH0/1 RX grant
6	REQ_CH0/1_RX	RO 0x0	CH0/1 RX request to GbDMA
7	OVERFLOW_ PENDING_CH0/1_R X	RO 0x0	Overflow flag in CH0/1
15:8	CUR_SAMPLE_CN T_CH0/1_RX	RO 0x0	Current sample count for CH0/1 RX
18:16	TX_RD_CTRL_ STATE_CH0/1_TX	RO 0x0	Tx read control state machine
21:19	PCM_TX_IF_ STATE_CH0/1_TX	RO 0x0	TX PCM state machine state
22	GNT_CH0/1_TX	RO 0x0	CH0/1 TX grant
23	REQ_CH0/1x_TX	RO 0x0	CH0/1 TX request to GbDMA
31:24	CUR_SAMPLE_CN T_CH0/1_TX	RO 0x0	Current sample count for CH0/1 TX

**Table 227: TDM DMA Abort Register 1**  
**Offset: 0xD0080**

Bit	Field	Type/InitVal	Description
31:0	TDM_DMA_ABORT_ADDRESS	RO 0x0	In case of illegal Mbus address space access by TDM DMA, the DMA will abort and the address which was accessed will be latched in this register. Also, the TDM DMA Abort Register 2 will contain other DMA parameters. For debug purposes only.

**Table 228: TDM DMA Abort Register 2**  
**Offset: 0xD0084**

Bit	Field	Type/InitVal	Description
15:0	DMA_TRANSFER_LENGTH_ABORTED	RO 0x0	DMA transfer length when it was aborted.
27:16	DMA_ID_ABORTED	RO 0x0	DMA ID when it was aborted.
28	DMA_CYCLE_TYPE_ABORTED	RO 0x0	Read or write cycle DMA
31:29	Reserved	RSVD 0x0	Reserved

**Table 229: TDM Channel0 Wideband Delay Control Register**  
**Offset: 0xD0088**

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> Channel 0 receive and transmit delay (from frame sync assertion) is controlled by the TDM Channel 0 Wideband Delay Control Register for second PCM sample in Wideband mode. The delay is defined in the number of PCLK cycles from the assertion of frame sync. The <CH0DlyEn> field must be set to 1 in the PCM Control Register.			
9:0	CH0_WBAND_RX_DLY	RW 0x40	Number of PCLK cycles to wait before starting to receive a second PCM sample for Wideband mode for Channel 0 from the assertion of Frame Sync (FS). After the start of receive, 8-bit or 16-bit data will be received, depending on the setting of the <PcmSampleSize> field in the PCM Control Register.
15:10	Reserved	RSVD 0x0	Reserved
25:16	CH0_WBAND_TX_DLY	RW 0x40	Number of PCLK cycles to wait before starting to transmit a second PCM sample for Wideband mode for Channel 0 from the assertion of FS. After the start of transmission, 8-bit or 16-bit data will be transmitted depending on the setting of the <PcmSampleSize> field in the PCM Control Register.



**Table 229: TDM Channel0 Wideband Delay Control Register (Continued)**  
Offset: 0xD0088

Bit	Field	Type/InitVal	Description
31:26	Reserved	RSVD 0x0	Reserved

**Table 230: TDM Channel 1 Wideband Delay Control Register**  
Offset: 0xD008C

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> Channel 1 receive and transmit delay (from frame sync assertion) is controlled by the TDM Channel 1 Wideband Delay Control Register for second PCM sample in Wideband mode. The delay is defined in the number of PCLK cycles from the assertion of frame sync. The <CH1DlyEn> field must be set to 1 in the PCM Control Register.			
9:0	CH1_WBAND_RX_DLY	RW 0x50	Number of PCLK cycles to wait before starting to receive a second PCM sample for Wideband mode for Channel 1 from the assertion of Frame Sync (FS). After the start of receive, 8-bit or 16-bit data will be received, depending on the setting of the <PcmSampleSize> field in the PCM Control Register.
15:10	Reserved	RSVD 0x0	Reserved
25:16	CH1_WBAND_TX_DLY	RW 0x50	Number of PCLK cycles to wait before starting to transmit a second PCM sample for Wideband mode for Channel 1 from the assertion of FS. After the start of transmission, 8-bit or 16-bit data will be transmitted, depending on the setting of the <PcmSampleSize> field in the PCM Control Register.
31:26	Reserved	RSVD 0x0	Reserved

### A.5.3 TDM-to-Mbus Bridge Registers

**Table 231: SPI Interface Output Enable Control Register**  
Offset: 0xD4000

Bit	Field	Type/InitVal	Description
0	SPIOutEnn	RW 0x1	When this bit is cleared, SPI output pins and the codec reset output signal are enabled (have active output enables).  0 = EnableSPIOut 1 = DisableSPIOut
13:1	Reserved	RO 0x0	Reserved
31:14	Reserved	RSVD 0x3fff	Reserved

**Table 232: TDM-Mbus Configuration Register**  
**Offset: 0xD4010**

Bit	Field	Type/InitVal	Description
7:0	Timeout	RW 0xFF	Crossbar Arbiter Timeout Preset Value
15:8	Reserved	RSVD 0x0	Reserved
16	TimeoutEn	RW 0x1	Crossbar Arbiter Timer Enable
31:17	Reserved	RSVD 0x0	Reserved

**Table 233: Window0 Control Register**  
**Offset: 0xD4030**

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window 0 Enable 0 = Disabled: Window is disabled. 1 = Enabled: Window is enabled.
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the target interface associated with this window. See "Default Address Map".
15:8	Attr	RW 0x0E	Specifies the target interface attributes associated with this window. See "Default Address Map".
31:16	Size	RW 0xFFFF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as sequence of 1's followed by sequence of 0's. The number of 1's specifies the size of the window in 64-KByte granularity (e.g., a value of 0x00FF specifies 256 = 16 MByte). <b>NOTE:</b> A value of 0x0 specifies 64-KByte size.

**Table 234: Window0 Base Register**  
**Offset: 0xD4034**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved



**Table 234: Window0 Base Register (Continued)**  
Offset: 0xD4034

Bit	Field	Type/InitVal	Description
31:16	Base	RW 0x0000	Base Address Used with the <Size> field to set the address window size and location. Corresponds to transaction address[31:16].

**Table 235: Window1 Control Register**  
Offset: 0xD4040

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window1 Enable. See "Window0 Control Register".
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See "Window0 Control Register".
15:8	Attr	RW 0x0D	Target specific attributes depending on the target interface. See the "Window0 Control Register".
31:16	Size	RW 0x0FFF	Window Size See "Window0 Control Register".

**Table 236: Window1 Base Register**  
Offset: 0xD4044

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x1000	Base Address See "Window0 Base Register".

**Table 237: Window2 Control Register**  
Offset: 0xD4050

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window2 Enable See "Window0 Control Register".
3:1	Reserved	RSVD 0x0	Reserved

**Table 237: Window2 Control Register (Continued)**  
Offset: 0xD4050

Bit	Field	Type/InitVal	Description
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See "Window0 Control Register".
15:8	Attr	RW 0x0B	Target specific attributes depending on the target interface. See "Window0 Control Register".
31:16	Size	RW 0x0FFF	Window Size See "Window0 Control Register".

**Table 238: Window2 Base Register**  
Offset: 0xD4054

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x2000	Base Address See "Window0 Base Register".

**Table 239: Window3 Control Register**  
Offset: 0xD4060

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window3 Enable See "Window0 Control Register".
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See "Window0 Control Register".
15:8	Attr	RW 0x07	Target specific attributes depending on the target interface. See "Window0 Control Register".
31:16	Size	RW 0x0FFF	Window Size See "Window0 Control Register".



**Table 240: Window3 Base Register**

Offset: 0xD4064

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x3000	Base Address See "Window0 Base Register".

**Table 241: TDM-Mbus Configuration 1 Register**

Offset: 0xD4070

Bit	Field	Type/InitVal	Description
0	PCM I/F S/W Reset	RW 0x0	PCM Interface Software Reset  <b>NOTE:</b> If the device is using an internally generated PCM clock (the PCM Control register <MstrPclk> field (bit[0]) is cleared), assertion of this bit must be performed at least 2 ms after the <MstrPclk> field is cleared. <b>NOTE:</b> This register must normally be used only at the configuration stage. 0 = PCMRSTnON: Assert PCM interface software reset . 1 = PCMRSTnOFF: De-assert PCM interface software reset.
1	Reserved	RW 0x0	Reserved
2	Reserved	RW 0x0	Always write 1 to this bit.
31:3	Reserved	RW 0x0	Reserved

## A.6

## PCI Express Interface Registers

The following table provides a summarized list of all of the PCI Express Interface registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 242: Register Map Table for the PCI Express Interface Registers**

Register Name	Offset	Table and Page
<b>PCI Express Address Window Control Registers</b>		
PCI Express Window0 Control Register	0x41820	<a href="#">Table 243, p. 439</a>
PCI Express Window0 Base Register	0x41824	<a href="#">Table 244, p. 440</a>
PCI Express Window0 Remap Register	0x4182C	<a href="#">Table 245, p. 440</a>
PCI Express Window1 Control Register	0x41830	<a href="#">Table 246, p. 441</a>
PCI Express Window1 Base Register	0x41834	<a href="#">Table 247, p. 441</a>
PCI Express Window1 Remap Register	0x4183C	<a href="#">Table 248, p. 442</a>
PCI Express Window2 Control Register	0x41840	<a href="#">Table 249, p. 442</a>
PCI Express Window2 Base Register	0x41844	<a href="#">Table 250, p. 443</a>
PCI Express Window2 Remap Register	0x4184C	<a href="#">Table 251, p. 443</a>
PCI Express Window3 Control Register	0x41850	<a href="#">Table 252, p. 443</a>
PCI Express Window3 Base Register	0x41854	<a href="#">Table 253, p. 444</a>
PCI Express Window3 Remap Register	0x4185C	<a href="#">Table 254, p. 444</a>
PCI Express Window4 Control Register	0x41860	<a href="#">Table 255, p. 444</a>
PCI Express Window4 Base Register	0x41864	<a href="#">Table 256, p. 445</a>
PCI Express Window4 Remap Register	0x4186C	<a href="#">Table 257, p. 445</a>
PCI Express Window4 Remap (High) Register	0x41870	<a href="#">Table 258, p. 446</a>
PCI Express Window5 Control Register	0x41880	<a href="#">Table 259, p. 446</a>
PCI Express Window5 Base Register	0x41884	<a href="#">Table 260, p. 447</a>
PCI Express Window5 Remap Register	0x4188C	<a href="#">Table 261, p. 447</a>
PCI Express Window5 Remap (High) Register	0x41890	<a href="#">Table 262, p. 447</a>
PCI Express Default Window Control Register	0x418B0	<a href="#">Table 263, p. 447</a>
PCI Express Expansion ROM Window Control Register	0x418C0	<a href="#">Table 264, p. 448</a>
PCI Express Expansion ROM Window Remap Register	0x418C4	<a href="#">Table 265, p. 448</a>
<b>PCI Express BAR Control Registers</b>		
PCI Express BAR1 Control Register	0x41804	<a href="#">Table 266, p. 449</a>
PCI Express BAR2 Control Register	0x41808	<a href="#">Table 267, p. 449</a>
PCI Express Expansion ROM BAR Control Register	0x4180C	<a href="#">Table 268, p. 449</a>
<b>PCI Express Configuration Cycles Generation Registers</b>		
PCI Express Configuration Address Register	0x418F8	<a href="#">Table 269, p. 450</a>
PCI Express Configuration Data Register	0x418FC	<a href="#">Table 270, p. 450</a>
<b>PCI Express Configuration Header Registers</b>		
PCI Express Device and Vendor ID Register	0x40000	<a href="#">Table 271, p. 451</a>
PCI Express Command and Status Register	0x40004	<a href="#">Table 272, p. 451</a>
PCI Express Class Code and Revision ID Register	0x40008	<a href="#">Table 273, p. 453</a>
PCI Express BIST Header Type and Cache Line Size Register	0x4000C	<a href="#">Table 274, p. 454</a>



Table 242: Register Map Table for the PCI Express Interface Registers (Continued)

Register Name	Offset	Table and Page
PCI Express BAR0 Internal Register	0x40010	Table 275, p. 454
PCI Express BAR0 Internal (High) Register	0x40014	Table 276, p. 455
PCI Express BAR1 Register	0x40018	Table 277, p. 455
PCI Express BAR1 (High) Register	0x4001C	Table 278, p. 455
PCI Express BAR2 Register	0x40020	Table 279, p. 455
PCI Express BAR2 (High) Register	0x40024	Table 280, p. 456
PCI Express Subsystem Device and Vendor ID Register	0x4002C	Table 281, p. 456
PCI Express Expansion ROM BAR Register	0x40030	Table 282, p. 457
PCI Express Capability List Pointer Register	0x40034	Table 283, p. 457
PCI Express Interrupt Pin and Line Register	0x4003C	Table 284, p. 457
PCI Express Power Management Capability Header Register	0x40040	Table 285, p. 458
PCI Express Power Management Control and Status Register	0x40044	Table 286, p. 459
PCI Express MSI Message Control Register	0x40050	Table 287, p. 460
PCI Express MSI Message Address Register	0x40054	Table 288, p. 461
PCI Express MSI Message Address (High) Register	0x40058	Table 289, p. 461
PCI Express MSI Message Data Register	0x4005C	Table 290, p. 461
PCI Express Capability Register	0x40060	Table 291, p. 461
PCI Express Device Capabilities Register	0x40064	Table 292, p. 462
PCI Express Device Control Status Register	0x40068	Table 293, p. 463
PCI Express Link Capabilities Register	0x4006C	Table 294, p. 465
PCI Express Link Control Status Register	0x40070	Table 295, p. 466
PCI Express Advanced Error Report Header Register	0x40100	Table 296, p. 468
PCI Express Uncorrectable Error Status Register	0x40104	Table 297, p. 468
PCI Express Uncorrectable Error Mask Register	0x40108	Table 298, p. 469
PCI Express Uncorrectable Error Severity Register	0x4010C	Table 299, p. 470
PCI Express Correctable Error Status Register	0x40110	Table 300, p. 471
PCI Express Correctable Error Mask Register	0x40114	Table 301, p. 472
PCI Express Advanced Error Capability and Control Register	0x40118	Table 302, p. 473
PCI Express Header Log First DWORD Register	0x4011C	Table 303, p. 473
PCI Express Header Log Second DWORD Register	0x40120	Table 304, p. 474
PCI Express Header Log Third DWORD Register	0x40124	Table 305, p. 474
PCI Express Header Log Fourth DWORD Register	0x40128	Table 306, p. 474
<b>PCI Express Control and Status Registers</b>		
PCI Express Control Register	0x41A00	Table 307, p. 474
PCI Express Status Register	0x41A04	Table 308, p. 476
PCI Express Boot Address Register	0x41A08	Table 309, p. 477
PCI Express Root Complex Set Slot Power Limit Register	0x41A0C	Table 310, p. 477
PCI Express Completion Timeout Register	0x41A10	Table 311, p. 478

**Table 242: Register Map Table for the PCI Express Interface Registers (Continued)**

Register Name	Offset	Table and Page
PCI Express Root Complex Power Management Event Register	0x41A14	Table 312, p. 478
PCI Express Power Management Extended Register	0x41A18	Table 313, p. 479
PCI Express Flow Control Register	0x41A20	Table 314, p. 480
PCI Express Acknowledge Timers (1X) Register	0x41A40	Table 315, p. 480
PCI Express RAM Parity Protection Control Register	0x41A50	Table 316, p. 480
PCI Express Debug Control Register	0x41A60	Table 317, p. 481
PCI Express TL Control Register	0x41AB0	Table 318, p. 483
PCI Express PHY Indirect Access Register	0x41B00	Table 319, p. 484
<b><i>PCI Express Interrupt Registers</i></b>		
PCI Express Interrupt Cause Register	0x41900	Table 320, p. 484
PCI Express Interrupt Mask Register	0x41910	Table 321, p. 487
<b><i>PCI Express Mbus Control Registers</i></b>		
PCI Express Mbus Adapter Control Register	0x418D0	Table 322, p. 488
PCI Express Mbus Arbiter Control Register (Low)	0x418E0	Table 323, p. 489
PCI Express Mbus Arbiter Control Register (High)	0x418E4	Table 324, p. 489
PCI Express Mbus Arbiter Timeout Register	0x418E8	Table 325, p. 490

## A.6.1 PCI Express Address Window Control Registers

**Table 243: PCI Express Window0 Control Register**  
Offset: 0x41820

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window0 Enable 0 = Disable: Window is disabled. 1 = Enable: Window is enabled.
1	BarMap	RW 0x0	Mapping to BAR 0 = BAR1: Window is mapped to BAR1. 1 = BAR2: Window is mapped to BAR2.
2	SlvWrSpltCnt	RW 0x0	Slave Write Split Control When forwarded to the internal interface, controls the address boundary on which the slave write transactions are split. Only relevant for transactions that hit the respective address window. <b>NOTE:</b> When working with cache coherency, this bit must be set to 1. 0 = SplitTo128: Write split on 128-Byte address boundary. 1 = SplitTo32: Write split on 32-Byte address boundary.
3	Reserved	RSVD 0x0	Reserved



**Table 243: PCI Express Window0 Control Register (Continued)**  
Offset: 0x41820

Bit	Field	Type/InitVal	Description
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See the "PCI Express Address Decoding" section.
15:8	Attr	RW 0x0E	Target specific attributes, depending on the target interface. See the "PCI Express Address Decoding" section.
31:16	Size	RW 0x0FFF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64 KB granularity. (A value of 0xFFFF specifies 256 MB).

**Table 244: PCI Express Window0 Base Register**  
Offset: 0x41824

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x0000	Base Address Used with the <Size> field to set the address window size and location. Corresponds to transaction address [31:16].

**Table 245: PCI Express Window0 Remap Register**  
Offset: 0x4182C

Bit	Field	Type/InitVal	Description
0	RemapEn	RW 0x0	Remap Enable Bit 0 = disable: Remap disabled. 1 = enable: Remap enabled.
15:1	Reserved	RSVD 0x0	Reserved
31:16	Remap	RW 0x0	Remap Address Used with the <Size> field to specifies address bits[31:0] to be driven to the target interface.

**Table 246: PCI Express Window1 Control Register**  
**Offset: 0x41830**

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window Enable 0 = Disable 1 = Enable
1	BarMap	RW 0x0	Mapping to BAR 0 = BAR1: Window is mapped to BAR1. 1 = BAR2: Window is mapped to BAR2.
2	SlvWrSplCnt	RW 0x0	Slave Write Split Control Controls the address boundary on which slave write transactions are split, when forwarded to the internal interface. Only relevant for transactions that hit the respective address window. <b>NOTE:</b> When working with cache coherency, this bit must be set to 1. 0 = SplitTo128: Write split on 128-Byte address boundary. 1 = SplitTo32: Write split on 32-Byte address boundary.
3	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window.
15:8	Attr	RW 0x0D	Target specific attributes, depending on the target interface.
31:16	Size	RW 0x0FFF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64 KB granularity. (A value of 0x0FFF specifies 256 MB).)

**Table 247: PCI Express Window1 Base Register**  
**Offset: 0x41834**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x1000	Base Address



**Table 248: PCI Express Window1 Remap Register**  
Offset: 0x4183C

Bit	Field	Type/InitVal	Description
0	RemapEn	RW 0x0	Remap Enable Bit 0 = disable: Remap disabled. 1 = enable: Remap enabled.
15:1	Reserved	RSVD 0x0	Reserved
31:16	Remap	RW 0x0	Remap Address

**Table 249: PCI Express Window2 Control Register**  
Offset: 0x41840

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window Enable 0 = Disable 1 = Enable
1	BarMap	RW 0x0	Mapping to BAR 0 = BAR1: Window is mapped to BAR1. 1 = BAR2: Window is mapped to BAR2.
2	SlvWrSplCnt	RW 0x0	Slave Write Split Control Controls the address boundary on which slave write transactions are split, when forwarded to the internal interface. Only relevant for transactions that hit the respective address window. <b>NOTE:</b> When working with cache coherency, this bit must be set to 1. 0 = SplitTo128: Write split on 128-Byte address boundary. 1 = SplitTo32: Write split on 32-Byte address boundary.
3	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window.
15:8	Attr	RW 0x0B	Target specific attributes, depending on the target interface.
31:16	Size	RW 0x0FFF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64 KB granularity. (A value of 0x0FFF specifies 256 MB).

**Table 250: PCI Express Window2 Base Register**  
**Offset: 0x41844**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x2000	Base Address

**Table 251: PCI Express Window2 Remap Register**  
**Offset: 0x4184C**

Bit	Field	Type/InitVal	Description
0	RemapEn	RW 0x0	Remap Enable Bit 0 = disable: Remap disabled. 1 = enable: Remap enabled.
15:1	Reserved	RSVD 0x0	Reserved
31:16	Remap	RW 0x0	Remap Address

**Table 252: PCI Express Window3 Control Register**  
**Offset: 0x41850**

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window Enable 0 = Disable 1 = Enable
1	BarMap	RW 0x0	Mapping to BAR 0 = BAR1: Window is mapped to BAR1. 1 = BAR2: Window is mapped to BAR2.
2	SlvWrSplCnt	RW 0x0	Slave Write Split Control Controls the address boundary on which slave write transactions are split, when forwarded to the internal interface. Only relevant for transactions that hit the respective address window. <b>NOTE:</b> When working with cache coherency, this bit must be set to 1. 0 = 128B: Write split on 128-Byte address boundary. 1 = 32B: Write split on 32-Byte address boundary.
3	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window.



**Table 252: PCI Express Window3 Control Register (Continued)**  
Offset: 0x41850

Bit	Field	Type/InitVal	Description
15:8	Attr	RW 0x07	Target specific attributes, depending on the target interface.
31:16	Size	RW 0x0FFF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64 KB granularity. (A value of 0xFFFF specifies 256 MB).

**Table 253: PCI Express Window3 Base Register**  
Offset: 0x41854

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x3000	Base Address

**Table 254: PCI Express Window3 Remap Register**  
Offset: 0x4185C

Bit	Field	Type/InitVal	Description
0	RemapEn	RW 0x0	Remap Enable Bit 0 = disable: Remap disabled. 1 = enable: Remap enabled.
15:1	Reserved	RSVD 0x0	Reserved
31:16	Remap	RW 0x0	Remap Address

**Table 255: PCI Express Window4 Control Register**  
Offset: 0x41860

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window Enable 0 = Disable 1 = Enable
1	BarMap	RW 0x1	Mapping to BAR 0 = BAR1: Window is mapped to BAR1. 1 = BAR2: Window is mapped to BAR2.

**Table 255: PCI Express Window4 Control Register (Continued)**  
Offset: 0x41860

Bit	Field	Type/InitVal	Description
2	SlvWrSplCnt	RW 0x0	Slave Write Split Control Controls the address boundary on which slave write transactions are split when forwarded to the internal interface. Only relevant for transactions that hit the respective address window. <b>NOTE:</b> When working with cache coherency, this bit must be set to 1. 0 = 128B: Write split on 128-Byte address boundary. 1 = 32B: Write split on 32-Byte address boundary.
3	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x1	Specifies the unit ID (target interface) associated with this window.
15:8	Attr	RW 0x3B	Target specific attributes, depending on the target interface.
31:16	Size	RW 0x07FF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64 KB granularity. (A value of 0x07FF specifies 128 MB).

**Table 256: PCI Express Window4 Base Register**  
Offset: 0x41864

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0xF000	Base Address

**Table 257: PCI Express Window4 Remap Register**  
Offset: 0x4186C

Bit	Field	Type/InitVal	Description
0	RemapEn	RW 0x0	Remap Enable Bit 0 = disable: Remap disabled. 1 = enable: Remap enabled.
15:1	Reserved	RSVD 0x0	Reserved
31:16	Remap	RW 0x0	Remap Address



**Table 258: PCI Express Window4 Remap (High) Register**  
Offset: 0x41870

Bit	Field	Type/InitVal	Description
31:0	RemapHigh	RW 0x0	<p>Remap High Address Specifies address bits[63:32] to be driven to the target interface.</p> <p>Only relevant for target interfaces that support a more than 4-GByte address space</p> <p>When using target interface that do not support a more than 4-GByte address space, this register must be cleared.</p>

**Table 259: PCI Express Window5 Control Register**  
Offset: 0x41880

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	<p>Window Enable 0 = Disable 1 = Enable</p>
1	BarMap	RW 0x1	<p>Mapping to BAR 0 = BAR1: Window is mapped to BAR1. 1 = BAR2: Window is mapped to BAR2.</p>
2	SlvWrSplCnt	RW 0x0	<p>Slave Write Split Control Controls the address boundary on which slave write transactions are split, when forwarded to the internal interface. Only relevant for transactions that hit the respective address window.</p> <p><b>NOTE:</b> When working with cache coherency, this bit must be set to 1.</p> <p>0 = 128B: Write split on 128-Byte address boundary. 1 = 32B: Write split on 32-Byte address boundary.</p>
3	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x1	Specifies the unit ID (target interface) associated with this window.
15:8	Attr	RW 0x2F	Target specific attributes, depending on the target interface.
31:16	Size	RW 0x07FF	<p>Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64 KB granularity. (A value of 0x07FF specifies 128 MB).</p>

**Table 260: PCI Express Window5 Base Register**  
Offset: 0x41884

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0xF800	Base Address

**Table 261: PCI Express Window5 Remap Register**  
Offset: 0x4188C

Bit	Field	Type/InitVal	Description
0	RemapEn	RW 0x0	Remap Enable Bit 0 = disable: Remap disabled. 1 = enable: Remap enabled.
15:1	Reserved	RSVD 0x0	Reserved
31:16	Remap	RW 0x0	Remap Address

**Table 262: PCI Express Window5 Remap (High) Register**  
Offset: 0x41890

Bit	Field	Type/InitVal	Description
31:0	RemapHigh	RW 0x0	Remap High Address Specifies address bits[63:32] to be driven to the target interface. Only relevant for target interfaces that support a more than 4-GByte address space When using target interface that do not support a more than 4-GByte address space, this register must be cleared.

**Table 263: PCI Express Default Window Control Register**  
Offset: 0x418B0

Bit	Field	Type/InitVal	Description
1:0	Reserved	RSVD 0x0	Reserved
2	SlvWrSpltCnt	RW 0x0	Slave Write Split Control Controls the address boundary on which slave write transactions are split, when forwarded to the internal interface. Only relevant for transactions that hit the respective address window. 0 = SplitTo128: Write split on 128-Bytes address boundary. 1 = SplitTo32: Write split on 32-Bytes address boundary



**Table 263: PCI Express Default Window Control Register (Continued)**  
Offset: 0x418B0

Bit	Field	Type/InitVal	Description
3	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window:
15:8	Attr	RW 0x0	Target specific attributes depending on the target interface.
31:16	Reserved	RSVD 0x0	Reserved

**Table 264: PCI Express Expansion ROM Window Control Register**  
Offset: 0x418C0

Bit	Field	Type/InitVal	Description
3:0	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x1	Specifies the unit ID (target interface) associated with this window. Default represents Runit.
15:8	Attr	RW 0x2F	Target specific attributes depending on the target interface. Default corresponds to DEV_BOOTCSn.
31:16	Reserved	RSVD 0x0	Reserved

**Table 265: PCI Express Expansion ROM Window Remap Register**  
Offset: 0x418C4

Bit	Field	Type/InitVal	Description
0	RemapEn	RW 0x0	Remap Enable Bit 0 = Disabled: Remap disabled. 1 = Enabled: Remap enabled.
15:1	Reserved	RSVD 0x0	Reserved
31:16	Remap	RW 0x0	Remap Address

## A.6.2 PCI Express BAR Control Registers

**Table 266: PCI Express BAR1 Control Register**  
**Offset: 0x41804**

Bit	Field	Type/InitVal	Description
0	Bar1En	RW 0x1	BAR1 Enable 0 = disable 1 = enable
15:1	Reserved	RSVD 0x0	Reserved
31:16	Bar1Size	RW 0x3FFF	BAR1 Size BAR sizes range from 64 KByte to 4 GByte in powers of 2. default value: 03FFF = 1 GByte

**Table 267: PCI Express BAR2 Control Register**  
**Offset: 0x41808**

Bit	Field	Type/InitVal	Description
0	Bar2En	RW 0x1	BAR2 Enable 0 = disable 1 = size
15:1	Reserved	RSVD 0x0	Reserved
31:16	Bar2Size	RW 0x0FFF	BAR2 Size BAR sizes range from 64 KByte to 4 GByte in powers of 2. default value: 0x0FFF = 256 MByte

**Table 268: PCI Express Expansion ROM BAR Control Register**  
**Offset: 0x4180C**

Bit	Field	Type/InitVal	Description
0	ExpROMEn	RW 0x0	Expansion ROM BAR Enable 0 = disable 1 = enable
18:1	Reserved	RSVD 0x0	Reserved

**Table 268: PCI Express Expansion ROM BAR Control Register (Continued)**  
Offset: 0x4180C

Bit	Field	Type/InitVal	Description
21:19	ExpROMSz	RW 0x0	Expansion ROM Address Bank Size 3 = 4093: 4096 KByte 0 = 512: 512 KByte 1 = 1024: 1024 KByte 2 = 2048: 2048 KByte 3 = 4096: 4096 KByte
31:22	Reserved	RSVD 0x0	Reserved

### A.6.3 PCI Express Configuration Cycles Generation Registers

**Table 269: PCI Express Configuration Address Register**  
Offset: 0x418F8

Bit	Field	Type/InitVal	Description
1:0	Reserved	RSVD 0x0	Reserved
7:2	RegNum	RW 0x0	Register Number
10:8	FunctNum	RW 0x0	Function Number
15:11	DevNum	RW 0x0	Device Number.
23:16	BusNum	RW 0x0	Bus Number
27:24	ExtRegNum	RW 0x0	Extended Register Number
30:28	Reserved	RSVD 0x0	Reserved
31	ConfigEn	RW 0x0	Configuration Enable Bit

**Table 270: PCI Express Configuration Data Register**  
Offset: 0x418FC

Bit	Field	Type/InitVal	Description
31:0	ConfigData	RW 0x0	Write access to this register generates a corresponding Configuration TLP on the PCI Express port or an access to the PCI Express port configuration header registers.

## A.6.4 PCI Express Configuration Header Registers

**Table 271: PCI Express Device and Vendor ID Register**  
**Offset: 0x40000**

Bit	Field	Type/InitVal	Description
15:0	VenID	RO 0x11AB	Vendor ID This field identifies Marvell as the Vendor of the device.
31:16	DevID	RO 0x6281	Device ID For the 88F6180, the initial value is 0x6180. For the 88F6190 and 88F6192, the initial value is 0x6192. For the 88F6281, the initial value is 0x6281.

**Table 272: PCI Express Command and Status Register**  
**Offset: 0x40004**

Bit	Field	Type/InitVal	Description
0	IOEn	RW 0x0	IO Space Enable. The IO space is not enabled; therefore, this bit is not functional. 0 = Disable 1 = Enable
1	MemEn	RW 0x0	Memory Space Enable Controls the device response to PCI Express memory accesses. 0 = Disable: All Memory accesses from PCI Express are completed as Unsupported Requests. 1 = Enable
2	MasEn	RW 0x0	Master Enable This bit controls the ability of the device to act as a master on the PCI Express port. When set to 0, no memory or I/O read/write request packets are generated to PCI Express. <b>NOTE:</b> Messages and completions are transmitted to the PCI Express regardless of the setting of this bit. 0 = Disable: Neither memory nor I/O requests are transmitted to the PCI-E port. 1 = Enable: Memory and I/O requests are transmitted to the PCI-E port.
5:3	Reserved	RSVD 0x0	Does not apply to PCI Express devices. These bits are hardwired to 0.
6	PErrEn	RW 0x0	Parity Error Enable This bit controls the ability of the device to respond to poisoned data errors as a requestor (master) on the PCI Express port. Controls MasDataPerr status bit assertion in PCMDSTT register. <b>NOTE:</b> The setting of this bit does not affect the DetectedPErr status bit.  0 = Disabled: MasDataPerr assertion is disabled. 1 = Enabled: MasDataPerr assertion is enabled.



**Table 272: PCI Express Command and Status Register (Continued)**  
Offset: 0x40004

Bit	Field	Type/InitVal	Description
7	Reserved	RSVD 0x0	Does not apply to PCI Express. This bit is hardwired to 0.
8	SErrEn	RW 0x0	This bit controls the ability of the device to report fatal and non-fatal errors to the Root Complex. This bit affects both assertion of SSysErr status bit[30] in this register and uncorrectable error message generation. <b>NOTE:</b> PCI Express uncorrectable error messages are reported if enabled either through this bit or through bits NFErrRepEn or FErrRepEn in the PCI Express Device Control Status Register. 0 = Disabled: SSysErr assertion is disabled. 1 = Enabled: SSysErr assertion is enabled. In addition uncorrectable error messages generation is enabled.
9	Reserved	RSVD 0x0	Does not apply to PCI Express. This bit is hardwired to 0.
10	IntDis	RW 0x0	Interrupt Disable This bit controls the ability of the device to generate interrupt emulation messages. When set, interrupt messages are not generated. 0 = Enabled: Interrupt messages enabled. 1 = Disabled: Interrupt messages disabled.
18:11	Reserved	RSVD 0x0	These bits are hardwired to 0.
19	IntStat	RO 0x0	Interrupt Status When set, this bit indicates that an interrupt message is pending internally in the device. 0 = Disabled: No interrupt asserted. 1 = Enabled: Interrupt asserted.
20	CapList	RO 0x1	Capability List Support This bit indicates that the device configuration header includes capability list. This bit is hardwired to 1, since this is always supported in PCI Express.
23:21	Reserved	RSVD 0x0	These bits are hardwired to 0.
24	MasDataPerr	RW1C 0x0	Master Data Parity Error This bit is set when the device has poisoned data detected as a requestor (master). Set when PErrEn bit[6] is set and either: Poisoned completion is received for the PCI Express port. or Poisoned TLP is transmitted to the PCI Express port. Write 1 to clear.
26:25	Reserved	RSVD 0x0	Does not apply to PCI Express. These bits are hardwired to 0.

**Table 272: PCI Express Command and Status Register (Continued)**  
**Offset: 0x40004**

Bit	Field	Type/InitVal	Description
27	STarAbort	RW1C 0x0	Signaled Target Abort This bit is set when the device, as a completer (target), completes a transaction as a Completer Abort. Write 1 to clear.
28	RTAbort	RW1C 0x0	Received Target Abort This bit is set when the device, as a requester (master), receives a completion with the status Completer Abort. Write 1 to clear.
29	RMAbort	RW1C 0x0	Received Master Abort. This bit is set when the device, as a requester (master), receives a completion with the status Unsupported Request. Write 1 to clear.
30	SSysErr	RW1C 0x0	Signalled System Error This bit is set when the device sends an ERR_FATAL or ERR_NONFATAL message. This bit is not set if the <SErrEn> field in this register is de-asserted. Write 1 to clear.
31	DetParErr	RW1C 0x0	Detected Parity Error This bit is set when the device receives a poisoned TLP. <b>NOTE:</b> The bit is set regardless of the state of the <PErrEn> bit in this register. Write 1 to clear.

**Table 273: PCI Express Class Code and Revision ID Register**  
**Offset: 0x40008**

Bit	Field	Type/InitVal	Description
7:0	Revid	RO 0x2	Device Revision Number
15:8	ProgIF	RO 0x0	Register Level Programming Interface
23:16	SubClass	RO 0x80	Device Sub-class--Other Memory Controller
31:24	BaseClass	RO 0x05	Device Base Class--Memory Controller

**Table 274: PCI Express BIST Header Type and Cache Line Size Register**  
**Offset: 0x4000C**

Bit	Field	Type/InitVal	Description
7:0	CacheLine	RW 0x00	Device Cache Line Size
15:8	Reserved	RSVD 0x0	Does not apply to PCI Express. These bits are hardwired to 0.
23:16	HeadType	RO 0x0	Device Configuration Header Type The header is a Type 0 single-function configuration header.
27:24	BISTComp	RO 0x0	BIST Completion Code 0x0 = BIST passed. Other = BIST failed.
29:28	Reserved	RSVD 0x0	Reserved
30	BISTAct	RO 0x0	BIST Activate Bit BIST is not supported.
31	BISTCap	RO 0x0	BIST Capable Bit BIST is not supported. This bit must be set to 0.

**Table 275: PCI Express BAR0 Internal Register**  
**Offset: 0x40010**

Bit	Field	Type/InitVal	Description
0	Space	RO 0x0	Memory Space Indicator.
2:1	Type	RO 0x2	BAR Type. BAR can be located in 64-bit memory address space.
3	Prefetch	RO 0x1	Prefetch Enable Indicates that pre-fetching is enabled.
19:4	Reserved	RSVD 0x0	Reserved
31:20	Base	RW 0xD00	Internal register memory Base Address Indicates 1MByte address space. Corresponds to address bits[31:20].

**Table 276: PCI Express BAR0 Internal (High) Register**  
**Offset: 0x40014**

Bit	Field	Type/InitVal	Description
31:0	Base	RW 0x0	Base address Corresponds to address bits[63:32].

**Table 277: PCI Express BAR1 Register**  
**Offset: 0x40018**

Bit	Field	Type/InitVal	Description
0	Space	RO 0x0	Memory Space Indicator
2:1	Type	RO 0x2	BAR Type BAR can be located in 64-bit memory address space.
3	Prefetch	RO 0x1	Prefetch Enable Indicates that pre-fetching is enabled.
15:4	Reserved	RSVD 0x0	Reserved
31:16	Base/Reserved	RW 0x0000	Base address Defined according to PCI Express BAR1 Control Register. Indicates 64-KByte up to 4-GByte address space. Corresponds to address bits[31:16].

**Table 278: PCI Express BAR1 (High) Register**  
**Offset: 0x4001C**

Bit	Field	Type/InitVal	Description
31:0	Base	RW 0x0	Base address Corresponds to address bits[63:32].

**Table 279: PCI Express BAR2 Register**  
**Offset: 0x40020**

Bit	Field	Type/InitVal	Description
0	Space	RO 0x0	Memory Space Indicator



**Table 279: PCI Express BAR2 Register (Continued)**  
Offset: 0x40020

Bit	Field	Type/InitVal	Description
2:1	Type	RO 0x2	BAR Type BAR can be located in 64-bit memory address space.
3	Prefetch	RO 0x1	Prefetch Enable Indicates that pre-fetching is enabled.
15:4	Reserved	RSVD 0x0	Reserved
31:16	Base/Reserved	RW 0xF000	Base address Defined according to PCI Express BAR2 Control Register. Indicates 64-KByte up to 4-GByte address space. Corresponds to address bits[31:16].

**Table 280: PCI Express BAR2 (High) Register**  
Offset: 0x40024

Bit	Field	Type/InitVal	Description
31:0	Base	RW 0x0	Base address Corresponds to address bits[63:32].

**Table 281: PCI Express Subsystem Device and Vendor ID Register**  
Offset: 0x4002C

Bit	Field	Type/InitVal	Description
15:0	SSVenID	RO 0x11AB	Subsystem Manufacturer ID Number The default is the Marvell Vendor ID.
31:16	SSDevID	RO 0x11AB	Subsystem Device ID Number The default is the Marvell Vendor ID.

**Table 282: PCI Express Expansion ROM BAR Register**  
**Offset: 0x40030**

Bit	Field	Type/InitVal	Description
If expansion ROM is not enabled via the ExpROMEn bit[0] the PCI Express Expansion ROM BAR Control Register, this register is Reserved.			
0	RomEn	RW 0x0	<p>Expansion ROM Enable  <b>NOTE:</b> The device expansion ROM address space is enabled only if both this bit (RomEn) and &lt;MemEn&gt; in the PCI Express Command and Status Register are set.          0 = Disabled          1 = Enabled</p>
18:1	Reserved	RSVD 0x0	Reserved
21:19	RomBase/Reserved	RW 0x0	<p>These bits are defined according to field &lt;ExpROMSz&gt; in the PCI Express Expansion ROM BAR Control Register.          When ROM Size is:          0 = 512KByteSpace: Bits[21 19] are used as Expansion ROM Base Address[21 19].          1 = 1MByteSpace: Bits[21 20] are used as Expansion ROM Base Address[21 20], and bit[19] is reserved.          2 = 2MByteSpace: Bits[21] used as Expansion ROM Base Address[21], and bits[20 19] are reserved.          3 = 4MByteSpace: Bits[21 19] are reserved.</p>
31:22	RomBase	RW 0x0	Expansion ROM Base Address[31:22]

**Table 283: PCI Express Capability List Pointer Register**  
**Offset: 0x40034**

Bit	Field	Type/InitVal	Description
7:0	CapPtr	RO 0x40	<p>Capability List Pointer          The current value in this field points to the PCI Power Management capability set in the PCI Express Power Management Capability Header Register at offset 0x40.</p>
31:8	Reserved	RSVD 0x0	Reserved

**Table 284: PCI Express Interrupt Pin and Line Register**  
**Offset: 0x4003C**

Bit	Field	Type/InitVal	Description
7:0	IntLine	RW 0x00	Provides interrupt line routing information.



**Table 284: PCI Express Interrupt Pin and Line Register (Continued)**  
Offset: 0x4003C

Bit	Field	Type/InitVal	Description
15:8	IntPin	RO 0x01	Indicates that the device is using INTA in the interrupt emulation messages.
31:16	Reserved	RSVD 0x0	Does not apply to PCI Express This field is hardwired to 0.

**Table 285: PCI Express Power Management Capability Header Register**  
Offset: 0x40040

Bit	Field	Type/InitVal	Description
7:0	CapID	RO 0x01	Capability ID Current value identifies the PCI Power Management capability.
15:8	NextPtr	RO 0x50	Next Item Pointer Current value points to MSI capability.
18:16	PMCVer	RO 0x3	PCI Power Management Capability Version
20:19	Reserved	RSVD 0x0	Does not apply to PCI Express This field must be hardwired to 0.
21	DSI	RO 0x0	Device Specific Initialization The device does not require device specific initialization.
24:22	AuxCur	RO 0x0	Auxiliary Current Requirements
25	D1Sup	RO 0x1	This bit indicates whether the device supports D1 Power Management state. The device does not support D1 state.
26	D2Sup	RO 0x1	This bit indicates whether the device supports D2 Power Management state. The device does not support D2 state.
31:27	PMESup	RO 0x00	This field indicates whether the device supports PM Event generation. The device does not support the PMEn pin.

**Table 286: PCI Express Power Management Control and Status Register**  
**Offset: 0x40044**

<b>Bit</b>	<b>Field</b>	<b>Type/InitVal</b>	<b>Description</b>
1:0	PMState	RW 0x0	<p>Power State  This field controls the Power Management state of the device. The device supports all Power Management states.</p> <p>0 = D0  1 = D1  2 = D2  3 = D3</p>
2	Reserved	RSVD 0x0	Reserved
3	No_Soft_Reset	RO 0x0	<p>When set (0x1), this bit indicates that devices transitioning from D3hot to D0 because of Power State commands do not perform an internal reset. Configuration Context is preserved. Upon transition from the D3hot to the D0 Initialized state, no additional operating system intervention is required to preserve Configuration Context beyond writing the Power Management State &lt;PMState&gt; bits.</p> <p>When cleared (0x0), devices do perform an internal reset upon transitioning from D3hot to D0 via software control of the Power Management State &lt;PMState&gt; bits. Configuration Context is lost when performing the soft reset. Upon transition from the D3hot to the D0 state, a full reinitialization sequence is required to return the device to D0 Initialized. Regardless of this bit, devices that transition from D3hot to D0 by a system or bus segment reset will return to the device state D0 Uninitialized with only PME context preserved if PME is supported and enabled.</p>
7:4	Reserved	RSVD 0x0	Reserved
8	PME_en	RW 0x0	<p>PM_PME Message Generation Enable  <b>NOTE:</b> Sticky bit--not initialized by hot reset.</p> <p>0 = Disabled: Disabled  1 = Enabled: Enabled</p>
12:9	PMDaSel	RW 0x0	<p>Data Select  This four-bit field is used to select which data is to be reported through the &lt;PMDaScale&gt; and &lt;PMDa&gt; fields of this register.</p>
14:13	PMDaScale	RO 0x0	<p>Data Scale  Indicated the scaling factor to be used when interpreting the value of the &lt;PMDa&gt; field of this register. The read value depends on the setting of the &lt;PMDaSel&gt; field of this register.</p>



**Table 286: PCI Express Power Management Control and Status Register (Continued)**  
Offset: 0x40044

Bit	Field	Type/InitVal	Description
15	PMEStat	RW1C 0x0	PME Status  <b>NOTE:</b> Sticky bit--not initialized by hot reset.
23:16	Reserved	RSVD 0x0	Does not apply to PCI Express. This field must be hardwired to 0.
31:24	PMData	RO 0x0	State Data This field is used to report the state dependent data requested by the <PMDataSel> field of this register. The value of this field is scaled by the value reported by the <PMDataScale> field of this register.

**Table 287: PCI Express MSI Message Control Register**  
Offset: 0x40050

Bit	Field	Type/InitVal	Description
7:0	CapID	RO 0x5	Capability ID The current value of this field identifies the PCI MSI capability.
15:8	NextPtr	RO 0x60	Next Item Pointer The current value of this field points to PCI Express capability.
16	MSIEn	RW 0x0	MSI Enable Controls the device interrupt signaling mechanism. 0 = Disabled: Interrupts are signalled through the interrupt emulation messages. 1 = Enabled: Interrupts are signaled through MSI mechanism.
19:17	MultiCap	RO 0x0	Multiple Message Capable The device is capable of driving a single message.
22:20	MultiEn	RW 0x0	Multiple Messages Enable The number of messages the system allocates to the device (This number must be smaller or equal to the value that is in the <MultiCap> field).
23	Addr64	RO 0x1	64-bit Addressing Capable This field indicates whether the device is capable of generating a 64-bit message address. 0 = NotCapable 1 = Capable
31:24	Reserved	RSVD 0x0	Reserved

**Table 288: PCI Express MSI Message Address Register**  
**Offset: 0x40054**

Bit	Field	Type/InitVal	Description
1:0	Reserved	RSVD 0x0	Reserved
31:2	MSIAddr	RW 0x0	Message Address This field corresponds to Address[31:2] of the MSI MWr TLP.

**Table 289: PCI Express MSI Message Address (High) Register**  
**Offset: 0x40058**

Bit	Field	Type/InitVal	Description
31:0	MSIAddrh	RW 0x0	Message Upper Address This field corresponds to Address[63:32] of the MSI MWr TLP.

**Table 290: PCI Express MSI Message Data Register**  
**Offset: 0x4005C**

Bit	Field	Type/InitVal	Description
15:0	MSIData	RW 0x0	Message Data
31:16	Reserved	RSVD 0x0	Reserved

**Table 291: PCI Express Capability Register**  
**Offset: 0x40060**

Bit	Field	Type/InitVal	Description
7:0	CapID	RO 0x10	Capability ID The current value of this field identifies the PCI PE capability.
15:8	NextPtr	RO 0x0	Next Item Pointer The current value of this field points to the end of the capability list (NULL).
19:16	CapVer	RO 0x1	Capability Version This field indicates the PCI Express Base spec 1.1 version of the PCI Express capability.
23:20	DevType	RO 0x0	Device/Port Type



**Table 291: PCI Express Capability Register (Continued)**  
Offset: 0x40060

Bit	Field	Type/InitVal	Description
24	SlotImp	RO 0x0	Slot Implemented Hardwired to 0.
29:25	IntMsgNum	RO 0x0	Interrupt Message Number This bit is hardwired to 0.
31:30	Reserved	RSVD 0x0	Reserved

**Table 292: PCI Express Device Capabilities Register**  
Offset: 0x40064

Bit	Field	Type/InitVal	Description
2:0	MaxPldSizeSup	RO 0x0	Maximum Payload Size (MPS) Supported 128B MPS support. 0 = 128B 1 = 256B
5:3	Reserved	RO 0x0	Reserved These bits are hardwired to 0.
8:6	EPL0sAccLat	RO 0x2	Endpoint L0s Acceptable Latency This field indicates the amount of latency that can be absorbed by the device due to the transition from L0s to L0.  0 = Under64ns: Less than 64 ns. 1 = 64to128ns: 64 ns to 128 ns. 2 = 128to256ns: 128 ns to 256 ns. 3 = 256to512ns: 256 ns to 512 ns. 4 = 512nsto1us: 512 ns to 1 us. 5 = 1to2us: 1 us to 2 us. 6 = 2to4us: 2 us to 4 us. 7 = Above4us: More than 4 us.
11:9	EPL1AccLat	RO 0x0	Endpoint L1 Acceptable Latency This field indicates the amount of latency that can be absorbed by the device due to the transition from L1 to L0. The current value specifies less than 1 second.
12	Reserved	RO 0x0	Reserved
13	Reserved	RO 0x0	Reserved
14	Reserved	RO 0x0	Reserved

**Table 292: PCI Express Device Capabilities Register (Continued)**  
Offset: 0x40064

Bit	Field	Type/InitVal	Description
15	RuleBaseErrorRep	RO 0x1	<p>Role-Based Error Reporting</p> <p>When set, this bit, indicates that the device implements the functionality originally defined in the Error Reporting ECN for PCI Express Base Specification, Revision 1.0a, and later incorporated into PCI Express Base Specification, Revision 1.1.</p> <p>This bit must be set by all devices conforming to the ECN, PCI Express Base Specification, Revision 1.1, or subsequent PCI Express Base Specification revisions.</p>
17:16	Reserved	RSVD 0x0	Reserved
25:18	CapSPLVal	RO 0x0	Captured Slot Power Limit Value
27:26	CapSPLScl	RO 0x0	Captured Slot Power Limit Scale
31:28	Reserved	RSVD 0x0	Reserved

**Table 293: PCI Express Device Control Status Register**  
Offset: 0x40068

Bit	Field	Type/InitVal	Description
0	CorErrRepEn	RW 0x0	<p>Correctable Error Reporting Enable</p> <p>In Root Complex mode, reporting of errors is internal to status registers only. An external error message must not be generated; therefore, always write 0x0.</p> <p><b>NOTE:</b> ERR_NONFATAL error messages are still enabled when this field is 0, if the &lt;SErrEn&gt; bit in the PCI Express Command and Status Register is set.</p> <p>0 = Disabled: ERR_COR error messages are masked. Status bit is not masked.  1 = Enabled: ERR_COR error messages enabled.</p>
1	NFErrRepEn	RW 0x0	<p>Non-Fatal Error Reporting Enable</p> <p>In Root Complex mode, reporting of errors is internal to status registers only. An external error message must not be generated, therefore always write 0x0.</p> <p><b>NOTE:</b> ERR_NONFATAL error messages are still enabled when this field is 0, if the &lt;SErrEn&gt; bit in PCI Express Command and Status Register is set.</p> <p>0 = Disabled: ERR_NONFATAL error messages are masked. Status bit is not masked.  1 = Enabled: ERR_NONFATAL error messages enabled.</p>



Table 293: PCI Express Device Control Status Register (Continued)  
Offset: 0x40068

Bit	Field	Type/InitVal	Description
2	FErrRepEn	RW 0x0	<p>Fatal Error Reporting Enable</p> <p>In Root Complex mode, reporting of errors is internal to status registers only. An external error message must not be generated; therefore, always write 0x0.</p> <p><b>NOTE:</b> ERR_FATAL error messages are still enabled when this field is 0, if the &lt;SErrEn&gt; bit in PCI Express Command and Status Register is set.</p> <p>0 = Disabled: ERR_FATAL error messages are masked. Status bit is still affected. 1 = Enabled: ERR_FATAL error messages enabled.</p>
3	URRepEn	RW 0x0	<p>Unsupported Request (UR) Reporting Enable</p> <p>In Root Complex mode, reporting of errors is internal to status registers only. An external error message must not be generated, therefore always write 0x0.</p> <p><b>NOTE:</b> UR related error messages are still enabled when URRepEn=0, if &lt;SErrEn&gt; bit in PCI Express Command and Status Register is set.</p> <p>0 = Disabled: UR related error messages are masked. Status bit is not masked. 1 = Enabled: UR related error messages enabled.</p>
4	EnRO	RO 0x0	<p>Enable Relaxed Ordering</p> <p>The device never sets the Relaxed Ordering attribute in transactions it initiates as a requester.</p> <p>Hardwired to 0x0.</p> <p>0 = Disable 1 = Enable</p>
7:5	MaxPldSz	RW 0x0	<p>Maximum Payload Size</p> <p>The maximum payload size supported is 128B (refer to bit &lt;MaxPldSizeSup&gt; in the PCI Express Device Capabilities Register).</p> <p>0x0 = 128B Other = Reserved 0 = 128: 0 1 = 256: 1</p>
9:8	Reserved	RO 0x0	<p>Reserved</p> <p>These bits are hardwired to 0.</p>
10	Reserved	RSVD 0x0	Reserved
11	EnNS	RO 0x0	<p>Enable No Snoop</p> <p>The device never sets the No Snoop attribute in transactions it initiates as a requester.</p> <p>Hardwired to 0x0.</p> <p>0 = Disable 1 = Enable</p>

**Table 293: PCI Express Device Control Status Register (Continued)**  
Offset: 0x40068

Bit	Field	Type/InitVal	Description
14:12	MaxRdRqSz	RW 0x2	Maximum Read Request Size Limits the device maximum read request size as a requestor (master). Other = Reserved 0 = 128B 1 = 256B 2 = 512B 3 = 1 KByte 4 = 2 KByte 5 = 4 KByte
15	Reserved	RSVD 0x0	Reserved
16	CorErrDet	RW1C 0x0	Correctable Error Detected Indicates the status of the correctable errors detected by the device. Write 1 to clear.
17	NFErrDet	RW1C 0x0	Non-Fatal Error Detected Indicates the status of the Non-Fatal errors detected by the device. Write 1 to clear.
18	FErrDet	RW1C 0x0	Fatal Error Detected Indicates the status of the Fatal errors detected by the device. Write 1 to clear.
19	URDet	RW1C 0x0	Unsupported Request Detected Indicates that the device received an unsupported request. Write 1 to clear.
20	Reserved	RSVD 0x0	Reserved
21	TransPend	RO 0x0	Transactions Pending Indicates that the device has issued Non-Posted requests that have not been completed. 0 = Completed: All NP requests have been completed or terminated by the Completion Timeout Mechanism. 1 = Uncompleted: Not all NP requests have been completed or terminated.
31:22	Reserved	RSVD 0x0	Reserved

**Table 294: PCI Express Link Capabilities Register**  
Offset: 0x4006C

Bit	Field	Type/InitVal	Description
3:0	MaxLinkSpd	RO 0x1	Maximum Link Speed The current value identifies the 2.5 Gbps link.



**Table 294: PCI Express Link Capabilities Register (Continued)**  
Offset: 0x4006C

Bit	Field	Type/InitVal	Description
9:4	MaxLnkWdth	RO 0x1	Maximum Link Width
11:10	AspmSup	RO 0x3	Active State Link PM Support The current value identifies L0s and L1 are supported. 1 = L0s_Entry_Supported 3 = L0s_and_L1_Supported
14:12	L0sExtLat	RO 0x2	L0s Exit Latency The time required by the device to transition its Rx lane from L0s to L0.  0 = Under64ns: Less than 64 ns. 1 = 64to128ns: 64 ns-128 ns. 2 = 128to256ns: 128 ns-256 ns. 3 = 256to512ns: 256 ns-512 ns. 4 = 512nsto1us: 512 ns-1 us. 5 = 1to2us: 1 us-2 us. 6 = 2to4us: 2 us-4 us. 7 = Reserved
17:15	L1ExtLat	RO 0x7	Reserved
18	Reserved	RO 0x1	Reserved
23:19	Reserved	RSVD 0x0	Reserved
31:24	PortNum	RO 0x0	In Endpoint mode, indicates the PCI Express port number, as displayed by the Root Complex during the link training process.

**Table 295: PCI Express Link Control Status Register**  
Offset: 0x40070

Bit	Field	Type/InitVal	Description
1:0	AspmCnt	RW 0x0	Active State Link PM Control Controls the level of active state PM supported on the link. 0 = Disabled: Disabled. 1 = L0s_Entry_Enabled: Tx-L0s entry supported. 2 = L1_Entry_Enabled: L1-ASPM entry supported 3 = L0s_and_L1_Entry_Enabled: Tx-L0s and L1-ASPM entry supported.
2	Reserved	RSVD 0x0	Reserved

**Table 295: PCI Express Link Control Status Register (Continued)**  
Offset: 0x40070

Bit	Field	Type/InitVal	Description
3	RCB	RW 0x1	<p>Read Completion Boundary  <b>NOTE:</b> This bit has no effect on the device behavior. Completions are always returned with 128B read completion boundary.</p> <p>0 = 64B  1 = 128B</p>
4	LnkDis	RW 0x0	<p>Link Disable  <b>NOTE:</b> If configured as an Endpoint, this field is reserved and has no effect.</p> <p>Activation procedure:  1. Set this bit to trigger link disable.  2. Poll &lt;DLDown&gt; de-assertion (PCI Express Status Register, bit [0]) to ensure that the link is disabled.  3. Clear the bit to exit and to detect and enable the link again.</p>
5	RetrnLnk	RW 0x0	<p>Retrain Link  This bit forces the device to initiate link retraining.  Always returns 0 when read.  <b>NOTE:</b> If configured as an Endpoint, this field is reserved and has no effect.</p>
6	CmnClkCfg	RW 0x0	<p>Common Clock Configuration  When set by software, this bit indicates that both devices on the link use a distributed common reference clock.</p>
7	ExtdSnc	RW 0x0	<p>Extended Sync  When set, this bit forces extended transmission of 4096 FTS ordered sets, followed by a single skip ordered set, in exit from L0s, and extra (1024) TS1 at exit from L1.  <b>NOTE:</b> This bit is used for test and measurement.</p>
8	Reserved	RW 0x0	<p>Reserved.  Must write 0x0.</p>
15:9	Reserved	RSVD 0x0	Reserved
19:16	LnkSpd	RO 0x1	<p>Link Speed  This field indicates the negotiated link speed.  The current value indicates 2.5 Gbps.</p>
25:20	NegLnkWdth	RO 0x1	<p>Negotiated Link Width  This field indicates the negotiated link width.  This field is initialized by the hardware.  <b>NOTE:</b> This field is only valid once the link is up.  1 = X1</p>
26	Reserved	RO 0x0	Reserved



**Table 295: PCI Express Link Control Status Register (Continued)**  
Offset: 0x40070

Bit	Field	Type/InitVal	Description
27	LnkTrn	RO 0x0	Link Training
28	SltClkCfg	RO 0x1	Slot Clock Configuration 0 = Independent: The device uses an independent clock, irrespective of the presence of a reference clock on the connector. 1 = Reference: The device uses the reference clock that the platform provides.
31:29	Reserved	RSVD 0x0	Reserved

**Table 296: PCI Express Advanced Error Report Header Register**  
Offset: 0x40100

Bit	Field	Type/InitVal	Description
15:0	PECapID	RO 0x1	Extended Capability ID The current value of this field identifies the Advanced Error Reporting capability.
19:16	CapVer	RO 0x1	Capability Version
31:20	NextPtr	RO 0x0	Next Item Pointer This field indicates the last item in the extended capabilities linked list.

**Table 297: PCI Express Uncorrectable Error Status Register**  
Offset: 0x40104

Bit	Field	Type/InitVal	Description
All fields in this register are sticky, not initialized or modified by hot reset.			
All fields in this register (except for reserved fields) are SC. Write 1 to clear. A write of 0 has no effect.			
3:0	Reserved	RSVD 0x0	Reserved
4	DLPrtErr	RW1C 0x0	Data Link Protocol Error Status <b>NOTE:</b> Hot sticky bit--not initialized by hot reset.
11:5	Reserved	RSVD 0x0	Reserved
12	RPsnTlpErr	RW1C 0x0	Poisoned TLP Status <b>NOTE:</b> Hot sticky bit--not initialized by hot reset.

**Table 297: PCI Express Uncorrectable Error Status Register (Continued)**  
**Offset: 0x40104**

Bit	Field	Type/InitVal	Description
13	Reserved	RSVD 0x0	Reserved <b>NOTE:</b> Sticky bit--not initialized by hot reset.
14	CmpTOErr	RW1C 0x0	Completion Timeout Status <b>NOTE:</b> Sticky bit--not initialized by hot reset.
15	CAErr	RW1C 0x0	Completer Abort Status <b>NOTE:</b> Sticky bit--not initialized by hot reset.
16	UnexpCmpErr	RW1C 0x0	Unexpected Completion Status <b>NOTE:</b> Sticky bit--not initialized by hot reset.
17	Reserved	RW1C 0x0	Reserved <b>NOTE:</b> Sticky bit--not initialized by hot reset.
18	MalfTlpErr	RW1C 0x0	Malformed TLP Status <b>NOTE:</b> Sticky bit--not initialized by hot reset.
19	Reserved	RSVD 0x0	Reserved.
20	URErr	RW1C 0x0	Unsupported Request Error Status <b>NOTE:</b> Sticky bit--not initialized by hot reset.
31:21	Reserved	RSVD 0x0	Reserved

**Table 298: PCI Express Uncorrectable Error Mask Register**  
**Offset: 0x40108**

Bit	Field	Type/InitVal	Description
All fields in this register are sticky not initialized or modified by hot reset.			
3:0	Reserved	RSVD 0x0	Reserved
4	DLPrtErrMsk	RW 0x0	Data Link Protocol Error Mask  When an error is indicated in the PCI Express Uncorrectable Error Status Register and the corresponding bit is set: - the header is not logged in the Header Log Register - the First Error Pointer is not updated - an error message is not generated.  Status bit is set regardless of the mask setting. 0 = NotMasked 1 = Masked
11:5	Reserved	RSVD 0x0	Reserved



**Table 298: PCI Express Uncorrectable Error Mask Register (Continued)**  
Offset: 0x40108

Bit	Field	Type/InitVal	Description
12	RPsntlpErrMsk	RW 0x0	Poisoned TLP Error Mask
13	Reserved	RSVD 0x0	Reserved
14	CmpTOErrMsk	RW 0x0	Completion Timeout Mask
15	CAErrMsk	RW 0x0	Completer Abort Mask
16	UnexpCmpErrMsk	RW 0x0	Unexpected Completion Mask
17	Reserved	RW 0x0	Reserved
18	MalftlpErrMsk	RW 0x0	Malformed TLP Mask
19	Reserved	RSVD 0x0	Reserved.
20	URErrMsk	RW 0x0	Unsupported Request Error Mask
31:21	Reserved	RSVD 0x0	Reserved

**Table 299: PCI Express Uncorrectable Error Severity Register**  
Offset: 0x4010C

Bit	Field	Type/InitVal	Description
All fields in this register are hot sticky not initialized or modified by reset.			
3:0	Reserved	RSVD 0x0	Reserved
4	DLPrtErrSev	RW 0x1	Data Link Protocol Error Severity  Controls the severity indication of the Uncorrectable errors. Each bit controls the error type of the corresponding bit in the PCI Express Uncorrectable Error Status Register. 0 = NonFatal: Error type is Non-Fatal. 1 = Fatal: Error type is Fatal.
11:5	Reserved	RSVD 0x0	Reserved
12	RPsntlpErrSev	RW 0x0	Poisoned TLP Error Severity

**Table 299: PCI Express Uncorrectable Error Severity Register (Continued)**  
**Offset: 0x4010C**

Bit	Field	Type/InitVal	Description
13	Reserved	RSVD 0x0	Reserved
14	CmpTOErrSev	RW 0x0	Completion Timeout Severity
15	CAErrSev	RW 0x0	Completer Abort Severity
16	UnexpCmpErrSev	RW 0x0	Unexpected Completion Severity
17	Reserved	RW 0x1	Reserved
18	MalfTlpErrSev	RW 0x1	Malformed TLP Severity
19	Reserved	RSVD 0x0	Reserved
20	URErrSev	RW 0x0	Unsupported Request Error Severity
31:21	Reserved	RSVD 0x0	Reserved

**Table 300: PCI Express Correctable Error Status Register**  
**Offset: 0x40110**

Bit	Field	Type/InitVal	Description
All fields in this register are sticky not initialized or modified by hot reset.			
All fields in this register (except for reserved fields) are RW1C write 1 to clear.			
0	RcvErr	RW1C 0x0	Receiver Error Status <b>NOTE:</b> When set, this bit indicates that a Receiver error has occurred.
5:1	Reserved	RSVD 0x0	Reserved
6	BadTlpErr	RW1C 0x0	Bad TLP Status
7	BadDllpErr	RW1C 0x0	Bad DLLP Status
8	RplyRllovrErr	RW1C 0x0	Replay Number Rollover Status
11:9	Reserved	RSVD 0x0	Reserved
12	RplyTOErr	RW1C 0x0	Replay Timer Timeout status



**Table 300: PCI Express Correctable Error Status Register (Continued)**  
Offset: 0x40110

Bit	Field	Type/InitVal	Description
13	AdvNonFatalErr	RW1C 0x0	Advisory Non-Fatal Error Status
31:14	Reserved	RSVD 0x0	Reserved

**Table 301: PCI Express Correctable Error Mask Register**  
Offset: 0x40114

All fields in this register are sticky not initialized or modified by hot reset.			
Bit	Field	Type/InitVal	Description
0	RcvMsk	RW 0x0	Receiver Error Mask If set, an error message is not generated upon occurrence of a Receiver error. 0 = NotMasked 1 = Masked
5:1	Reserved	RSVD 0x0	Reserved
6	BadTlpMsk	RW 0x0	Bad TLP Mask
7	BadDlppErrMsk	RW 0x0	Bad DLLP Mask
8	RplyRllovrMsk	RW 0x0	Replay Number Rollover Mask
11:9	Reserved	RSVD 0x0	Reserved
12	RplyTOMsk	RW 0x0	Replay Timer Timeout Mask
13	AdvNonFatalMsk	RW 0x1	Advisory Non-Fatal Error Mask. This bit is set by default to enable compatibility with software that does not comprehend Role-Based Error Reporting.
31:14	Reserved	RSVD 0x0	Reserved

**Table 302: PCI Express Advanced Error Capability and Control Register**  
**Offset: 0x40118**

Bit	Field	Type/InitVal	Description
4:0	FrstErrPtr	RO 0x0	<p>First Error Pointer  This field reports the bit position of the first error reported in the PCI Express Uncorrectable Error Status Register.  This field locks upon receipt of the first uncorrectable error that is not masked. It remains locked until software clears it by writing 1 to the corresponding status bit. Upon receipt of the next uncorrectable error that is not masked, the field locks again until cleared as described above. This lock and clear process continues to repeat itself.</p> <p><b>NOTE:</b> The bits in this field are sticky bits--they are not initialized or modified by reset.</p> <ul style="list-style-type: none"> <li>4 = DLP: Data Link Protocol Error</li> <li>12 = TLP: Poisoned TLP Error</li> <li>13 = FCP: Flow Control Protocol Error</li> <li>14 = CT: Completion Timeout Error</li> <li>15 = CAS: Completer Abort Status</li> <li>16 = UCE: Unexpected Completion Error</li> <li>17 = ROE: Receiver Overflow Error</li> <li>18 = Mutant TLP: Malformed TLP Error</li> <li>19 = Reserved</li> <li>20 = URE: Unsupported Request Error</li> <li>Other = Reserved</li> </ul>
31:5	Reserved	RSVD 0x0	Reserved Bits [5] and [7] are Read Only and are hardwired to 0.

**Table 303: PCI Express Header Log First DWORD Register**  
**Offset: 0x4011C**

Bit	Field	Type/InitVal	Description
All fields in this register are sticky, not initialized or modified by hot reset.			
31:0	Hdrlog1DW	RO 0x0	<p>Header Log First DWORD  Logs the header of the first error reported in the PCI Express Uncorrectable Error Status Register (PEUEST).  This field locks upon receipt of the first uncorrectable error that is not masked. It remains locked until software clears it by writing 1 to the corresponding status bit. Upon receipt of the next uncorrectable error that is not masked, the field locks again until cleared as described above. This lock and clear process continues to repeat itself.</p>



**Table 304: PCI Express Header Log Second DWORD Register**  
Offset: 0x40120

Bit	Field	Type/InitVal	Description
All fields in this register are sticky, not initialized or modified by hot reset.			
31:0	Hdrlog2DW	RO 0x0	Header Log Second DWORD

**Table 305: PCI Express Header Log Third DWORD Register**  
Offset: 0x40124

Bit	Field	Type/InitVal	Description
All fields in this register are sticky, not initialized or modified by hot reset.			
31:0	Hdrlog3DW	RO 0x0	Header Log Third DWORD

**Table 306: PCI Express Header Log Fourth DWORD Register**  
Offset: 0x40128

Bit	Field	Type/InitVal	Description
All fields in this register are sticky, not initialized or modified by hot reset.			
31:0	Hdrlog4DW	RO 0x0	Header Log Fourth DWORD <b>NOTE:</b> All fields in this register are Hot sticky bit -- not initialized or modified by hot reset.

## A.6.5 PCI Express Control and Status Registers

**Table 307: PCI Express Control Register**  
Offset: 0x41A00

Bit	Field	Type/InitVal	Description
0	ConfLinkX1	RW 0x1	Must write 0x1.
1	ConfRoot Complex	RW 0x1	PCI Express Device Type Control <b>NOTE:</b> Cannot be configured during operation, must be configured only by reset strapping or from TWSI during auto_loader.  0 = Endpoint: Endpoint device (downstream device / upstream link). 1 = Root: Root Complex device (upstream device / downstream link).

**Table 307: PCI Express Control Register (Continued)**  
Offset: 0x41A00

Bit	Field	Type/InitVal	Description
2	CfgMapTo MemEn	RW 0x1	Configuration Header Mapping to Memory Space Enable When enabled, the configuration header registers can be accessed directly through the memory space. Access is enabled both from the internal bus and from the PCI Express port. 0 = Disabled: Mapping disabled. 1 = Enabled: Mapping enabled.
7:3	Reserved	RSVD 0x5	Reserved
9:8	Reserved	RW 0x0	Reserved Must write 0x0.
15:10	Reserved	RSVD 0x0	Reserved
23:16	Reserved	RW 0x14	Must write 0x14.
24	ConfMstrHot Reset	RW 0x0	Master Hot-Reset When set, a hot-reset command is transmitted downstream, causing the downstream PCI Express hierarchy to enter a hot-reset cycle. This bit can be set only if <LnkDis> in the PCI Express Link Control Status Register and bit[26] of this register are cleared. Activation procedure: 1. Set this bit to trigger a hot-reset cycle. 2. Poll <DLDown> de-assertion (PCI Express Status Register, bit 0) to ensure hot-reset cycle occurs. 3. Clear this bit to exit to detect and re-establish the PCI Express link. This bit should be used only when working in Root Complex mode. 0 = Off 1 = On
25	Reserved	RSVD 0x0	Reserved
26	ConfMstrLb	RW 0x0	Master Loopback When set, a loopback command is transmitted downstream, causing the downstream device to transmit back the traffic that it receives. This bit can be set only if bits[24] of this register and <LnkDis> in the PCI Express Link Control Status Register are cleared. <b>NOTE:</b> Use this bit only when working in Root Complex mode. 0 = Off 1 = On
27	ConfMstrDis Scrbm	RW 0x0	Master Disable Scrambling When set, a scrambling disable command is transmitted downstream, causing the scrambling to be disabled on both sides of the link. <b>NOTE:</b> This should be programmed before the start of link initialization. 0 = Off 1 = On



**Table 307: PCI Express Control Register (Continued)**  
Offset: 0x41A00

Bit	Field	Type/InitVal	Description
29:28	Reserved	RSVD 0x0	Reserved
30	Conf_Training_Disable	RW 0x1	Mac_Training_Disable = (Conf_Training_Disable and Training_Disable); Training_Disable is connected as a PCI Express input. If (Mac_Training_Disable ==1) => the MAC-LTSSM and the PHY remain at reset.
31	Crs_Enable	RW 0x0	When this bit is set, all configuration accesses will return with a status of: <CRS>.  0 = Disable 1 = Enable

**Table 308: PCI Express Status Register**  
Offset: 0x41A04

Bit	Field	Type/InitVal	Description
0	DLDown	RO 0x0	DL_Down indication 0 = Active: DL is up 1 = Inactive: DL is down
4:1	Reserved	RSVD 0x0	Reserved
7:5	Reserved	RSVD 0x0	Reserved
15:8	PexBusNum	RW 0x0	Bus Number Indication This field is used in the RequesterID field of the transmitted TLPs. In Endpoint mode, the field updates whenever a CfgWr access is received.
20:16	PexDevNum	RW 0x0	Device Number Indication This field is used in the RequesterID field of the transmitted TLPs. In Endpoint mode, the field updates whenever a CfgWr access is received.
23:21	Reserved	RSVD 0x0	Reserved
24	PexSlvHot Reset	RO 0x0	Slave Hot Reset Indication This field sets when the opposite device on the PCI Express port acts as a hot-reset master, and a hot-reset indication is received.
25	PexSlvDisLink	RO 0x0	Slave Disable Link Indication This field sets when the opposite device on the PCI Express port acts as a disable link master, and a link disabled indication is received.

**Table 308: PCI Express Status Register (Continued)**  
Offset: 0x41A04

Bit	Field	Type/InitVal	Description
26	PexSlvLb	RO 0x0	Slave Loopback Indication This field sets when the opposite device on the PCI Express port acts as a loopback master, and a loopback indication is received.
27	PexSlvDis Scrbm	RO 0x0	Slave Disable Scrambling Indication This field sets when the opposite device on the PCI Express port acts as a disable scrambling master, and a scrambling disabled indication is received.
31:28	Reserved	RSVD 0x0	Reserved

**Table 309: PCI Express Boot Address Register**  
Offset: 0x41A08

Bit	Field	Type/InitVal	Description
31:0	BootAddr	RW 0xFFFF0000	The 32-bit address of the BootROM header. After initializing the PCI Express to operate as an Endpoint, the BootROM sets this register to 0xFFFFFFFF and waits for the register to be updated by the Root Complex.

**Table 310: PCI Express Root Complex Set Slot Power Limit Register**  
Offset: 0x41A0C

Bit	Field	Type/InitVal	Description
7:0	SlotPowerLimitValue	RW 0x0	Slot Power Limit Value In combination with the Slot Power Limit Scale value, specifies the upper limit for power supplied by the slot. Power limit (in watts) is calculated by multiplying the value in this field by the value in the <SlotPowerLimitScale> field. Writes to this register, when the PCI Express interface is in Root Complex mode, also cause the port to send the SSPL message.
9:8	SlotPowerLimitScale	RW 0x0	Slot Power Limit Scale Specifies the scale used for the Slot Power Limit Value (see the Functional Specification). Range of values: 00b = 1.0x 01b = 0.1x 10b = 0.01x 11b = 0.001x Writes to this register, when the PCI Express interface is in Root Complex mode, also cause the port to send the SSPL message.
15:10	Reserved	RO 0x0	Reserved



**Table 310: PCI Express Root Complex Set Slot Power Limit Register (Continued)**  
Offset: 0x41A0C

Bit	Field	Type/InitVal	Description
16	SsplMsgEnable	RW 0x0	Set Slot Power Limit (SSPL) Message Enable 0 = Disable: No SSPL Message will be sent. 1 = Enable: SSPL Message will be send for PCI Express RootComplex where one of the following occurs: 1) DL status changes from DL Down to DL-Up or 2) A write to either the <SlotPowerLimitValue> or <SlotPowerLimitScale> fields.
31:17	Reserved	RO 0x0	Reserved

**Table 311: PCI Express Completion Timeout Register**  
Offset: 0x41A10

Bit	Field	Type/InitVal	Description
15:0	ConfCmpToThrshld	RW 0x2710	Completion Timeout Threshold This field controls the size of the completion timeout interval. The NP request is cleared from MCT within -0% +100% of the timeout value.  <b>NOTE:</b> Timescale: 256*symbol_time = 1us Initial Value (0x2710) represents a 10 ms value (10,000 decimal) 0x0 = Disabled. No timeout mechanism on N.P TLPs. Minimum Value: 40 (40 s) Maximum Value: 25K (25 ms)
31:16	Reserved	RSVD 0x0	Reserved

**Table 312: PCI Express Root Complex Power Management Event Register**  
Offset: 0x41A14

Bit	Field	Type/InitVal	Description
15:0	PME Requester ID	RO 0x0	PME Requestor ID This field indicates the PCI requestor ID of the last locked PME requestor. When the <PME Status> is 0x0, the value in the this field is regard as reserved.
16	PME Status	RO 0x0	PME Status This field indicates that PME was asserted by the requestor ID indicated in the <PME Requestor ID> field. This bit can be cleared by writing 0x1 to the <RcvPmPme> field in the PCI Express Interrupt Cause register.

**Table 312: PCI Express Root Complex Power Management Event Register (Continued)**  
**Offset: 0x41A14**

Bit	Field	Type/InitVal	Description
17	PME Pending	RO 0x0	Power Management Event (PME) Pending This read-only bit indicates that another PME is pending when the <PME Status> field is set. When the <PME Status> field is cleared by software, the PME is delivered by hardware, by setting the <PME Status> field again and updating the requestor ID appropriately. The PME pending bit is cleared by hardware, if no more PMEs are pending.
31:18	Reserved	RSVD 0x0	Reserved

**Table 313: PCI Express Power Management Extended Register**  
**Offset: 0x41A18**

Bit	Field	Type/InitVal	Description
0	L1_aspm_en	RW 0x0	Only for Endpoint. Enables the hardware to initialize the L1 ASPM (Active State Power Management) process. 0 = Idle: No L1-ASPM flow (as an Endpoint) 1 = StartL1AspmFlow
1	L1AspmAck	RW 0x1	Only for Root Complex mode. System allows acknowledging L1 ASPM request by an Endpoint. 0 = SendAspmNackTlp: Stop L1-ASPM flow. 1 = SendAspmAckDlp: Continue L1-ASPM flow.
3:2	Reserved	RSVD 0x0	Reserved
4	SendTurnOffAckMsg	RW 0x0	Only for Endpoint.  0 = Halt_L2L3_Flow: Do not send a TurnOffAck message after receiving a TurnOff-message. 1 = Cont_L2L3_Flow: Send a TurnOffAck-message after receiving TurnOff message and continue the L2L3 flow.
5	SendTurnOffMsg	RW 0x0	Host request to the PCI Express Unit to send a Turn-Off message
15:6	Reserved	RSVD 0x0	Reserved
16	Reserved	RW 0x0	Reserved. Must write 0x0.
31:17	Reserved	RW 0x0	Reserved



**Table 314: PCI Express Flow Control Register**  
Offset: 0x41A20

Bit	Field	Type/InitVal	Description
7:0	ConfPhlInitFc	RW 0x4	Posted Headers Flow Control Credit Initial Value
15:8	ConfNphlInitFc	RW 0x8	Non-Posted Headers Flow Control Credit Initial Value
23:16	ConfChlInitFc	RW 0x0	Completion Headers Flow Control Credit Initial Value Infinite.
31:24	ConfFc UpdateTo	RW 0x78	Flow Control Update Timeout This field controls the Flow Control update interval period. <b>NOTE:</b> Timescale: 64*symbol_time = 256 ns Minimum Value: 120 (30 microseconds) Maximum Value: 180 (45 microseconds) Note1: When extended sync is enabled, the check threshold should be configured to 120 microseconds. Note2: A value <8 is regard as reserved and must not be configured by the user. 120 = 30 seconds. (120 = 30 s)

**Table 315: PCI Express Acknowledge Timers (1X) Register**  
Offset: 0x41A40

Bit	Field	Type/InitVal	Description
15:0	AckLatTOX1	RW 0x50	Acknowledge Latency Timer Timeout Value for 1X Link Used when the PHY link width Auto-Negotiation result is 1X. <b>NOTE:</b> Timescale: symbol_time = 4 ns Minimum Value: 8 (8 symbol times) Maximum Value: 237 (237symbol times)
31:16	AckRplyTOX1	RW 0x320	Acknowledge Reply Timer Timeout Value for 1X <b>NOTE:</b> Timescale: symbol_time = 4 ns Initial value (0x320) represents a 800 symbol times timeout. Minimum Value: 711 (711 symbol times) Maximum Value: 64K-1 (64K-1 symbol times)

**Table 316: PCI Express RAM Parity Protection Control Register**  
Offset: 0x41A50

Bit	Field	Type/InitVal	Description
0	RxPrtyGen	RW 0x0	Rx RAM parity generation. 0 = Even Parity: Generate even parity -- Normal operation. 1 = Odd Parity: Generate odd parity -- Causes erroneous parity.

**Table 316: PCI Express RAM Parity Protection Control Register (Continued)**  
Offset: 0x41A50

Bit	Field	Type/InitVal	Description
1	RxPrtyChkEn	RW 0x1	Rx RAM Parity Check Enable. 0 = Checked: Even Parity is not checked when data is read from internal buffers. 1 = NotChecked: Even Parity is checked when data is read from internal buffers.
2	TxPrtyGen	RW 0x0	Tx RAM parity generation. 0 = Even: Generate even parity -- Normal operation. 1 = Odd: Generate odd parity -- Causes erroneous parity.
3	TxPrtyChkEn	RW 0x1	Tx RAM Parity Check Enable. 0 = Checked: Even Parity is not checked when data is read from internal buffers. 1 = NotChecked: Even Parity is checked when data is read from internal buffers.
31:4	Reserved	RO 0x0	Reserved

**Table 317: PCI Express Debug Control Register**  
Offset: 0x41A60

Bit	Field	Type/InitVal	Description
0	Reserved	RW 0x0	Reserved. Write only 0x0.
1	Reserved	RW 0x0	Reserved. Write only 0x0.
4:2	Reserved	RSVD 0x0	Reserved
5	Reserved	RW 0x0	Reserved
6	Reserved	RW 0x1	Reserved
7	Reserved	RW 0x1	Reserved
8	Reserved	RW 0x1	Reserved
11:9	Reserved	RSVD 0x0	Reserved
12	mask_soft_rst	RW 0x1	Mask soft reset from PCI Express unit reset output.  0 = Disable 1 = Enable

**Table 317: PCI Express Debug Control Register (Continued)**  
Offset: 0x41A60

Bit	Field	Type/InitVal	Description
13	soft_rst_regfile_rst	RW 0x1	Disable PCI Express Register file reset upon soft reset.  0 = Disable 1 = Enable
14	link_dis_regfile_rst	RW 0x1	Disable PCI Express Register file reset upon link disable. <b>NOTE:</b> Sticky bit -- not initialized by reset.  0 = Disable 1 = Enable
15	MaskLinkDis	RW 0x1	Mask link disable from PCI Express unit reset output. <b>NOTE:</b> Sticky bit -- not initialized by reset. 0 = Disable 1 = Enable
16	ConfMskLinkFail	RW 0x1	Mask linkfail from PCI Express unit reset output. <b>NOTE:</b> Sticky bit -- not initialized by reset. 0 = Disable 1 = Enable
17	ConfMskHotReset	RW 0x1	Mask hot reset from PCI Express unit reset output. <b>NOTE:</b> Sticky bit -- not initialized by reset. 0 = Disable 1 = Enable
18	Reserved	RW 0x0	Reserved. Must write 0.  0 = Disable 1 = Enable
19	dis_link_fail_reg_rst	RW 0x1	Disable PCI Express Register file reset upon link failure. <b>NOTE:</b> Sticky bit -- not initialized by reset.  0 = Disable 1 = Enable
20	SoftReset	RW 0x0	PCI Express Unit Soft Reset When set, generates an internal reset in the PCI Express unit. This bit should be cleared after the link is re-established. In Root Complex mode by default, this reset will not reset the register file or generate pex_RST_OUT_. Before setting this bit, make sure there are no pending TLP in the unit.  0 = Disable 1 = Enable

**Table 317: PCI Express Debug Control Register (Continued)**  
Offset: 0x41A60

Bit	Field	Type/InitVal	Description
21	dis_hot_reset_reg_rst	RW 0x1	<p>Disable PCI Express Register file reset upon hot reset.  <b>NOTE:</b> Sticky bit -- not initialized by reset.</p> <p>If set to 0x0 (reset is enabled), then TWSI serial initialization must be initiated again, if it overwrites PCI Express configurations.</p> <p>0 = Disable 1 = Enable</p>
22	Reserved	RW 0x1	<p>Reserved. Must write 1.</p> <p>0 = Disable 1 = Enable</p>
23	Reserved	RW 0x0	Reserved. Must write 0.
29:24	Reserved	RSVD 0x0	Reserved
31:30	Reserved	RW 0x0	Reserved. Must write 0.

**Table 318: PCI Express TL Control Register**  
Offset: 0x41AB0

Bit	Field	Type/InitVal	Description
0	RxCmplPushDis	RW 0x0	<p>Rx transactions ordering control (CMP vs. P)</p> <p>0 = Disable 1 = Enable: Rx completion may bypass posted.</p>
1	RxNpPushDis	RW 0x0	<p>Rx transactions ordering control (non-posted versus posted)</p> <p>0 = Disable 1 = Enable: Rx non-posted transactions may bypass posted transactions.</p>
2	TxCmplPushDis	RW 0x0	<p>Tx transactions ordering control (CMP vs. P)</p> <p>0 = Disable 1 = Enable: Tx completion may bypass posted.</p>
3	TxNpPushDis	RW 0x0	<p>Tx transactions ordering control (non-posted versus posted transactions)</p> <p>0 = Disable 1 = Enable: Tx non-posted transactions may bypass posted transactions.</p>
4	Reserved	RW 0x0	Reserved
5	Reserved	RW 0x0	Reserved

**Table 318: PCI Express TL Control Register (Continued)**  
Offset: 0x41AB0

Bit	Field	Type/InitVal	Description
6	Reserved	RW 0x0	Reserved
7	Reserved	RSVD 0x0	Reserved
11:8	Reserved	RW 0x4	Must write 0x4.
31:12	Reserved	RSVD 0x0	Reserved

**Table 319: PCI Express PHY Indirect Access Register**  
Offset: 0x41B00

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The default values of the PHY registers must never be changed, unless explicitly requested in one of the device documents..			
15:0	PhyData	RW 0x0	PCI Express PHY register data.
29:16	PhyAddr	RW 0x0	14-bit PCI Express PHY register address. Bits[7:0] select the register offset within a specific lane. Bits[13:8]: Write 0x0.
30	Reserved	RSVD 0x0	Reserved
31	PhyAccessMd	RW 0x0	PCI Express PHY Access Mode controls the indirect access type. 0 = Write: The data in <PhyData> field is written to the PHY register pointed by <PhyAddr> field. 1 = Read: Read preparation access. The value of the PHY register pointed by <PhyAddr> is stored in <PhyData> field.

## A.6.6 PCI Express Interrupt Registers

**Table 320: PCI Express Interrupt Cause Register**  
Offset: 0x41900

Bit	Field	Type/InitVal	Description
All bits, except bits[27:24], are RW0C only. A cause bit sets upon an event occurrence. A write of 0 clears the bit. A write of 1 has no affect. Bits[24:27] are set and cleared upon reception of interrupt emulation messages.			
0	TxReqInDldownErr	RW0C 0x0	Received Tx request while PCI Express is in DL-Down status.
1	MDis	RW0C 0x0	Attempt to generate a PCI transaction while master is disabled.

**Table 320: PCI Express Interrupt Cause Register (Continued)**  
Offset: 0x41900

Bit	Field	Type/InitVal	Description
2	Reserved	RW0C 0x0	Reserved
3	ErrWrToReg	RW0C 0x0	Erroneous Write Attempt to Internal Register Set when an erroneous write request to a PCI Express internal register is received, either from the PCI Express (EP set) or from the internal bus (bit[64] set).
4	HitDfltWinErr	RW0C 0x0	Hit Default Window Error
5	Reserved	RW0C 0x0	Reserved
6	RxRamParErr	RW0C 0x0	Rx RAM Parity Error
7	TxRamParErr	RW0C 0x0	Tx RAM Parity Error
8	CorErrDet	RW0C 0x0	Correctable Error Detected Indicates status of correctable errors detected by the device.
9	NFErrDet	RW0C 0x0	Non-Fatal Error Detected Indicates status of Non-Fatal errors detected by the device.
10	FErrDet	RW0C 0x0	Fatal Error Detected Indicates status of Fatal errors detected by the device.
11	DstateChange	RW0C 0x0	Dstate Change Indication Any change in the Dstate asserts this bit. <b>NOTE:</b> This bit is relevant only for Endpoint.
12	BIST	RW0C 0x0	PCI Express BIST Activated BIST is not supported.
13	Reserved	RW0C 0x0	Reserved
14	FlowCtrlProtocol	RW0C 0x0	Flow Control Protocol Error. Set when there is no FC DLLP for more than 200 us.
15	RcvUrCaErr	RW0C 0x0	Received either a UR or CA completion status.
16	RcvErrFatal	RW0C 0x0	Received ERR_FATAL message Set when a downstream device detected the error and sent an error message upstream. Relevant for Root Complex only.



**Table 320: PCI Express Interrupt Cause Register (Continued)**  
Offset: 0x41900

Bit	Field	Type/InitVal	Description
17	RcvErrNonFatal	RW0C 0x0	Received ERR_NONFATAL message Set when a downstream device detected the error and sent an error message upstream. Relevant for Root Complex only.
18	RcvErrCor	RW0C 0x0	Received ERR_COR message Set when a downstream device detected the error and sent an error message upstream. Relevant for Root Complex only.
19	RcvCRS	RW0C 0x0	Received CRS Completion Status A downstream PCI Express device can respond to a configuration request with CRS (Configuration Request Retry Status) if it is not ready yet to serve the request. A RcvCRS interrupt is set when such a completion status is received.
20	PexSlvHot Reset	RW0C 0x0	Received Hot Reset Indication The bit sets when a hot-reset indication is received from the opposite device on the PCI Express port. <b>NOTE:</b> Sticky bit--not initialized by reset.
21	PexSlvDisLink	RW0C 0x0	Slave Disable Link Indication The bit sets when the opposite device on the PCI Express port is acting as a disable link master, and the link was disabled. <b>NOTE:</b> Sticky bit--not initialized by reset.
22	PexSlvLb	RW0C 0x0	Slave Loopback Indication The bit sets when the opposite device on the PCI Express port is acting as a loopback master, and loopback mode was entered. <b>NOTE:</b> Sticky bit--not initialized by reset.
23	PexLinkFail	RW0C 0x0	Link Failure Indication PCI Express link dropped from active state (L0, L0s, or L1) to Detect state due to link errors. <b>NOTE:</b> When dropping to Detect via Hot Reset, Disable Link or Loopback states, the interrupt is not asserted.  <b>NOTE:</b> Sticky bit--not initialized by reset.
24	RcvIntA	RO 0x0	IntA status Reflects the IntA Interrupt message emulation status. Set when a IntA Assert message is received. Cleared when the IntA_De-assert message is received, or upon a link failure scenario (DL_down). <b>NOTE:</b> This bit is not RW0C, as the other bit in this register, since it is cleared by the interrupting device downstream. Relevant for Root Complex only.

**Table 320: PCI Express Interrupt Cause Register (Continued)**  
Offset: 0x41900

Bit	Field	Type/InitVal	Description
25	RcvIntB	RO 0x0	<p>IntB status  Reflects the IntB Interrupt message emulation status.  Set when a IntB Assert message is received.  Cleared when the IntB_De-assert message is received, or upon a link failure scenario (DL_down).</p> <p><b>NOTE:</b> This bit is not RW0C, as the other bit in this register, since it is cleared by the interrupting device downstream.  Relevant for Root Complex only.</p>
26	RcvIntC	RO 0x0	<p>IntC status  Reflects the IntC Interrupt message emulation status.  Set when a IntC Assert message is received.  Cleared when the IntC_De-assert message is received, or upon a link failure scenario (DL_down).</p> <p><b>NOTE:</b> This bit is not RW0C, as the other bit in this register, since it is cleared by the interrupting device downstream.  Relevant for Root Complex only.</p>
27	RcvIntD	RO 0x0	<p>IntDn status  Reflects the IntD Interrupt message emulation status.  Set when a IntD Assert message is received.  Cleared when the IntD_De-assert message is received, or upon a link failure scenario (DL_down).</p> <p><b>NOTE:</b> This bit is not RW0C, as the other bit in this register, since it is cleared by the interrupting device downstream.  Relevant for Root Complex only.</p>
28	RcvPmPme	RW0C 0x0	Received PM_PME (Power Management Event) message.
29	RcvTurnOff	RW0C 0x0	<p>This status field has two meanings:  - When the PCI Express interface is in Endpoint mode =&gt; RcvTurnOffMsg: Receives Turn Off message from the Root Complex .  - When the PCI Express interface is in Root Complex mode =&gt; Ready4TurnOff: Root Complex notifies the Host that the Turn Off process has been completed (either normally or by the timer).</p>
30	Reserved	RW0C 0x0	Reserved
31	RcvMsi	RW0C 0x0	Root Complex (RC) received a Memory Write that hit the MSI attributes.

**Table 321: PCI Express Interrupt Mask Register**  
Offset: 0x41910

Bit	Field	Type/InitVal	Description
19:0	Mask	RW 0x0	Mask bit per cause bit. If a bit is set to 1, the corresponding event is enabled. Mask does not affect setting of the Interrupt Cause register bits; it only affects the assertion of the interrupt.



**Table 321: PCI Express Interrupt Mask Register (Continued)**  
Offset: 0x41910

Bit	Field	Type/InitVal	Description
23:20	Mask	RW 0x0	Mask bit per cause bit. If a bit is set to 1, the corresponding event is enabled. Mask does not affect setting of the Interrupt Cause register bits; it only affects the assertion of the interrupt.
31:24	Mask	RW 0x0	Mask bit per cause bit. If a bit is set to 1, the corresponding event is enabled. Mask does not affect setting of the Interrupt Cause register bits; it only affects the assertion of the interrupt.

## A.6.7 PCI Express Mbus Control Registers

**Table 322: PCI Express Mbus Adapter Control Register**  
Offset: 0x418D0

Bit	Field	Type/InitVal	Description
0	Reserved	RW 0x1	Reserved Write only 0x1 to this field.
1	Reserved	RW 0x0	Reserved
3:2	Reserved	RW 0x0	Reserved
4	Reserved	RW 0x0	Reserved
5	Reserved	RW 0x0	Reserved
18:6	Reserved	RSVD 0x0	Reserved
19	RxDPPropEn	RW 0x1	Rx Data Poisoning Error Propagation Enable 0 = Disabled 1 = Enabled
20	TxDPPropEn	RW 0x1	Tx Data Poisoning Error Propagation Enable 0 = Enabled 1 = Disabled
31:21	Reserved	RSVD 0x0	Reserved

**Table 323: PCI Express Mbus Arbiter Control Register (Low)**  
**Offset: 0x418E0**

<b>Bit</b>	<b>Field</b>	<b>Type/InitVal</b>	<b>Description</b>
3:0	Arb0	RW 0x0	Slice 0 of arbiter.
7:4	Arb1	RW 0x1	Slice 1 of arbiter.
11:8	Arb2	RW 0x2	Slice 2 of arbiter.
15:12	Arb3	RW 0x3	Slice 3 of arbiter.
19:16	Arb4	RW 0x4	Slice 4 of arbiter.
23:20	Arb5	RW 0x5	Slice 5 of arbiter.
27:24	Arb6	RW 0x6	Slice 6 of arbiter.
31:28	Arb7	RW 0x7	Slice 7 of arbiter.

**Table 324: PCI Express Mbus Arbiter Control Register (High)**  
**Offset: 0x418E4**

<b>Bit</b>	<b>Field</b>	<b>Type/InitVal</b>	<b>Description</b>
3:0	Arb8	RW 0x8	Slice 8 of arbiter.
7:4	Arb9	RW 0x9	Slice 9 of arbiter.
11:8	Arb10	RW 0xa	Slice 10 of arbiter.
15:12	Arb11	RW 0xb	Slice 11 of arbiter.
19:16	Arb12	RW 0xc	Slice 12 of arbiter.
23:20	Arb13	RW 0xd	Slice 13 of arbiter.
27:24	Arb14	RW 0xe	Slice 14 of arbiter.
31:28	Arb15	RW 0xf	Slice 15 of arbiter.



**Table 325: PCI Express Mbus Arbiter Timeout Register**  
**Offset: 0x418E8**

Bit	Field	Type/InitVal	Description
7:0	Timeout	RW 0xFF	Mbus Arbiter Timeout Preset Value <b>NOTE:</b> Must keep the value of 0xFF, as long as the Timeout counter is not enabled.
15:8	Reserved	RSVD 0x0	Reserved
16	TimeoutEn	RW 0x1	Mbus Arbiter Timeout Timer Enable 0 = Enable 1 = Disable
31:17	Reserved	RSVD 0x0	Reserved

## A.7

## Serial-ATA Host Controller (SATAHC) Registers

The following table provides a summarized list of all of the Serial-ATA Host Controller (SATAHC) registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 326: Register Map Table for the Serial-ATA Host Controller (SATAHC) Registers**

Register Name	Offset	Table and Page
<b>Basic DMA Registers</b>		
Basic DMA Command Register	Port0: 0x82224, Port1: 0x84224	<a href="#">Table 327, p. 494</a>
Basic DMA Status Register	Port0: 0x82228, Port1: 0x84228	<a href="#">Table 328, p. 495</a>
Descriptor Table Low Base Address Register	Port0: 0x8222C, Port1: 0x8422C	<a href="#">Table 329, p. 496</a>
Descriptor Table High Base Address Register	Port0: 0x82230, Port1: 0x84230	<a href="#">Table 330, p. 497</a>
Data Region Low Address Register	Port0: 0x82234, Port1: 0x84234	<a href="#">Table 331, p. 497</a>
Data Region High Address Register	Port0: 0x82238, Port1: 0x84238	<a href="#">Table 332, p. 497</a>
<b>EDMA Registers</b>		
EDMA Configuration Register	Port0: 0x82000, Port1: 0x84000	<a href="#">Table 333, p. 498</a>
EDMA Interrupt Error Cause Register	Port0: 0x82008, Port1: 0x84008	<a href="#">Table 334, p. 501</a>
EDMA Interrupt Error Mask Register	Port0: 0x8200C, Port1: 0x8400C	<a href="#">Table 335, p. 503</a>
EDMA Request Queue Base Address High Register	Port0: 0x82010, Port1: 0x84010	<a href="#">Table 336, p. 503</a>
EDMA Request Queue In-Pointer Register	Port0: 0x82014, Port1: 0x84014	<a href="#">Table 337, p. 504</a>
EDMA Request Queue Out-Pointer Register	Port0: 0x82018, Port1: 0x84018	<a href="#">Table 338, p. 504</a>
EDMA Response Queue Base Address High Register	Port0: 0x8201C, Port1: 0x8401C	<a href="#">Table 339, p. 504</a>
EDMA Response Queue In-Pointer Register	Port0: 0x82020, Port1: 0x84020	<a href="#">Table 340, p. 504</a>
EDMA Response Queue Out-Pointer Register	Port0: 0x82024, Port1: 0x84024	<a href="#">Table 341, p. 505</a>
EDMA Command Register	Port0: 0x82028, Port1: 0x84028	<a href="#">Table 342, p. 505</a>
EDMA Status Register	Port0: 0x82030, Port1: 0x84030	<a href="#">Table 343, p. 506</a>
EDMA IORdy Timeout Register	Port0: 0x82034, Port1: 0x84034	<a href="#">Table 344, p. 508</a>
EDMA Command Delay Threshold Register	Port0: 0x82040, Port1: 0x84040	<a href="#">Table 345, p. 508</a>
EDMA Halt Conditions Register	Port0: 0x82060, Port1: 0x84060	<a href="#">Table 346, p. 508</a>



Table 326: Register Map Table for the Serial-ATA Host Controller (SATAHC) Registers (Continued)

Register Name	Offset	Table and Page
EDMA NCQ0 Done/TCQ0 Outstanding Status Register	Port0: 0x82094, Port1: 0x84094	Table 347, p. 509
<b>SATAHC Arbiter Registers</b>		
SATAHC Configuration Register	0x80000	Table 348, p. 509
SATAHC Request Queue Out-Pointer Register	0x80004	Table 349, p. 510
SATAHC Response Queue In-Pointer Register	0x80008	Table 350, p. 510
SATAHC Interrupt Coalescing Threshold Register	0x8000C	Table 351, p. 511
SATAHC Interrupt Time Threshold Register	0x80010	Table 352, p. 511
SATAHC Interrupt Cause Register	0x80014	Table 353, p. 512
SATAHC Main Interrupt Cause Register	0x80020	Table 354, p. 513
SATAHC Main Interrupt Mask Register	0x80024	Table 355, p. 514
SATAHC LED Configuration Register	0x8002C	Table 356, p. 515
Window0 Control Register	0x80030	Table 357, p. 515
Window0 Base Register	0x80034	Table 358, p. 516
Window1 Control Register	0x80040	Table 359, p. 516
Window1 Base Register	0x80044	Table 360, p. 517
Window2 Control Register	0x80050	Table 361, p. 517
Window2 Base Register	0x80054	Table 362, p. 517
Window3 Control Register	0x80060	Table 363, p. 518
Window3 Base Register	0x80064	Table 364, p. 518
<b>Serial-ATA Interface Registers</b>		
Serial-ATA Interface Configuration Register	Port0: 0x82050, Port1: 0x84050	Table 365, p. 518
Serial-ATA PLL Configuration Register	Port0: 0x82054, Port1: 0x84054	Table 366, p. 521
SStatus Register	Port0: 0x82300, Port1: 0x84300	Table 367, p. 522
SError Register	Port0: 0x82304, Port1: 0x84304	Table 368, p. 523
SControl Register	Port0: 0x82308, Port1: 0x84308	Table 369, p. 525
LTMode Register	Port0: 0x8230C, Port1: 0x8430C	Table 370, p. 526
PHY Mode 3 Register	Port0: 0x82310, Port1: 0x84310	Table 371, p. 527
PHY Mode 4 Register	Port0: 0x82314, Port1: 0x84314	Table 372, p. 529
PHY Mode 1 Register	Port0: 0x8232C, Port1: 0x8432C	Table 373, p. 531
PHY Mode 2 Register	Port0: 0x82330, Port1: 0x84330	Table 374, p. 532
BIST Control Register	Port0: 0x82334, Port1: 0x84334	Table 375, p. 534
BIST-Dword1 Register	Port0: 0x82338, Port1: 0x84338	Table 376, p. 534

**Table 326: Register Map Table for the Serial-ATA Host Controller (SATAHC) Registers (Continued)**

Register Name	Offset	Table and Page
BIST-Dword2 Register	Port0: 0x8233C, Port1: 0x8433C	<a href="#">Table 377, p. 535</a>
SError Interrupt Mask Register	Port0: 0x82340, Port1: 0x84340	<a href="#">Table 378, p. 535</a>
Serial-ATA Interface Control Register	Port0: 0x82344, Port1: 0x84344	<a href="#">Table 379, p. 535</a>
Serial-ATA Interface Test Control Register	Port0: 0x82348, Port1: 0x84348	<a href="#">Table 380, p. 537</a>
Serial-ATA Interface Status Register	Port0: 0x8234C, Port1: 0x8434C	<a href="#">Table 381, p. 538</a>
Vendor Unique Register	Port0: 0x8235C, Port1: 0x8435C	<a href="#">Table 382, p. 540</a>
FIS Configuration Register	Port0: 0x82360, Port1: 0x84360	<a href="#">Table 383, p. 540</a>
FIS Interrupt Cause Register	Port0: 0x82364, Port1: 0x84364	<a href="#">Table 384, p. 541</a>
FIS Interrupt Mask Register	Port0: 0x82368, Port1: 0x84368	<a href="#">Table 385, p. 543</a>
FIS Dword0 Register	Port0: 0x82370, Port1: 0x84370	<a href="#">Table 386, p. 543</a>
FIS Dword1 Register	Port0: 0x82374, Port1: 0x84374	<a href="#">Table 387, p. 543</a>
FIS Dword2 Register	Port0: 0x82378, Port1: 0x84378	<a href="#">Table 388, p. 543</a>
FIS Dword3 Register	Port0: 0x8237C, Port1: 0x8437C	<a href="#">Table 389, p. 543</a>
FIS Dword4 Register	Port0: 0x82380, Port1: 0x84380	<a href="#">Table 390, p. 544</a>
FIS Dword5 Register	Port0: 0x82384, Port1: 0x84384	<a href="#">Table 391, p. 544</a>
FIS Dword6 Register	Port0: 0x82388, Port1: 0x84388	<a href="#">Table 392, p. 544</a>
PHYMODE9_GEN2 Register	Port0: 0x82398, Port1: 0x84398	<a href="#">Table 393, p. 544</a>
PHYMODE9_GEN1 Register	Port0: 0x8239C, Port1: 0x8439C	<a href="#">Table 394, p. 545</a>
PHY Configuration Register	Port0: 0x823A0, Port1: 0x843A0	<a href="#">Table 395, p. 546</a>
PHYTCTL Register	Port0: 0x823A4, Port1: 0x843A4	<a href="#">Table 396, p. 547</a>
PHY Mode 10 Register	Port0: 0x823A8, Port1: 0x843A8	<a href="#">Table 397, p. 547</a>
PHY Mode 12 Register	Port0: 0x823B0, Port1: 0x843B0	<a href="#">Table 398, p. 548</a>

## A.7.1 Basic DMA Registers

**Table 327: Basic DMA Command Register**

Offset: Port0: 0x82224 Port1: 0x84224

Bit	Field	Type/InitVal	Description
0	Start	RW 0x0	<p>Basic DMA operation is enabled by setting this bit to 1.</p> <p>Basic DMA operation begins when this bit is detected changing from a "0" to a 1. The Basic DMA transfers data between the ATA device and memory only when this bit is set.</p> <p>The Basic DMA operation can be halted by writing a "0" to this bit. All state information is lost when a "0" is written. The Basic DMA operation cannot be stopped and then resumed. This bit is intended to be reset by the CPU after the data transfer is completed.</p> <p>If a Basic DMA operation is aborted during command execution to the drive, the host software must set bit &lt;eAtaRst&gt;(bit [2]) to recover.</p>
2:1	Reserved	RW 0x0	Reserved
3	Read	RW 0x0	<p>This bit sets the direction of the Basic DMA transfer:</p> <p>This bit must not be changed when the Basic DMA is active.</p> <p>0 = Read: Memory Basic reads are performed. Read from system memory and store in the device.</p> <p>1 = Write: Memory Basic writes are performed. Load from the device and write to system memory.</p>
7:4	Reserved	RW 0x0	Reserved
8	DRegionValid	RW 0x0	<p>This bit indicates if the &lt;DataRegionByteCount&gt; field [31:16] in this register, Data Region Low Address Register and Data Region High Address Register are valid and to be used by the DMA.</p> <p>This bit must not be changed when the Basic DMA is active.</p> <p>0 = Invalid: Data Region is not valid</p> <p>1 = Valid: Data Region is valid</p>
9	DataRegionLast	RW 0x0	<p>This bit is valid only if bit[8] &lt;DRegionValid&gt; is set to 1.</p> <p>This bit indicates if the data region described by the &lt;DataRegionByteCount&gt; field [31:16] in the Data Region Low Address and Data Region High Address registers are the last data region to be transferred.</p> <p>This bit must not be changed when the Basic DMA is active.</p> <p>0 = More data: Not last data region.</p> <p>1 = Last data: Last data region to be transferred in the PRD table.</p>
10	ContFromPrev	RW 0x0	<p>This bit indicates if the Basic DMA needs to continue from the point where it stopped on its previous transaction or if it needs to load a new PRD table.</p> <p>The Basic DMA ignores the value of this bit, if the &lt;BasicDMAPaused&gt; bit in the Basic DMA Status Register is cleared to 0 and a new PRD table is loaded.</p> <p>This bit must not be changed when the Basic DMA is active.</p> <p>0 = New table: Load new PRD table</p> <p>1 = Old table: Continue from the PRD table associated with its previous DMA transaction</p>

**Table 327: Basic DMA Command Register (Continued)**  
Offset: Port0: 0x82224 Port1: 0x84224

Bit	Field	Type/InitVal	Description
15:11	Reserved	RW 0x0	Reserved
31:16	DataRegionByteCount	RW 0x0	<p>Data Region Byte Count</p> <p>This field indicates the count of the data region in bytes.</p> <p>Bit [0] is force to 0.</p> <p>There is a 64 KB maximum. A value of 0 indicates 64 KB.</p> <p>The data in the buffer must not cross the boundary of the 32-bit address space, that is, the 32-bit high address of all data in the buffer must be identical.</p> <p>This field value is updated by the DMA and indicates the completion status of the DMA when bit &lt;BasicDMAActive&gt; in the Basic DMA Status Register is cleared to 0.</p> <p>The host must not write to this bit when the Basic DMA is active.</p>

**Table 328: Basic DMA Status Register**  
Offset: Port0: 0x82228 Port1: 0x84228

Bit	Field	Type/InitVal	Description
0	BasicDMAActive	RO 0x0	<p>This bit is set when the start bit is written to the command register.</p> <p>This bit is cleared when the last transfer for the region is performed, where EOT for that region is set in the region descriptor or a complete FIS is transferred.</p> <p>It is also cleared when the start bit is cleared in the command register.</p> <p>When this bit is read as a 0, all data transferred from the drive during the previous Basic DMA is visible in system memory, unless the Basic DMA command was aborted.</p> <p>This bit is set when the start bit is written to the command register.</p> <p>This bit is cleared when:</p> <ul style="list-style-type: none"> <li>- The last transfer for the region is performed, where EOT for that region is set in the region descriptor OR</li> <li>- Bit &lt;eEDMAFBS&gt; in EDMA Configuration Register is set and a complete FIS is received or a complete FIS is transmitted OR</li> <li>-The start bit is cleared in the command register.</li> </ul> <p>When bit &lt;eEarlyCompletionEn&gt; is set, this bit is cleared as soon as last data leaves the DMA.</p> <p>When bit &lt;eEarlyCompletionEn&gt; is cleared and this bit is read as a 0, all data transferred from the drive in a read transaction during the previous Basic DMA is visible in system memory, unless the Basic DMA command was aborted.</p> <p>0 = Idle DMA  1 = Active DMA</p>
1	BasicDMAError	RO 0x0	<p>This bit is valid when bit[0] &lt;BasicDMAActive&gt; in this register is cleared.</p> <p>This bit is set when an error is encountered or when the DMA halts abnormally</p> <p>0 = No error  1 = Error</p>



**Table 328: Basic DMA Status Register (Continued)**  
Offset: Port0: 0x82228 Port1: 0x84228

Bit	Field	Type/InitVal	Description
2	BasicDMAPaused	RO 0x0	<p>This bit is valid when bit[0] &lt;BasicDMAActive&gt; in this register is cleared. This bit is set when:</p> <ul style="list-style-type: none"><li>- Bit &lt;eEDMAFBS&gt; in the EDMA Configuration Register is set and</li><li>- Bit[0] &lt;BasicDMAActive&gt; is cleared and</li><li>- The last transfer for the region is not yet performed, or EOT for that region is not set in the region descriptor.</li></ul> <p>This bit is cleared when:</p> <ul style="list-style-type: none"><li>- Bit[0] &lt;BasicDMAActive&gt; is cleared and</li><li>- The last transfer for the region is performed, where EOT for that region is set in the region descriptor OR</li><li>- The start bit is cleared in the command register.</li></ul> <p>0 = Idle state: The DMA is in idle state 1 = Paused: The DMA is paused</p>
3	BasicDMALast	RO 0x0	<p>This bit is valid when bit[0] &lt;BasicDMAActive&gt; in this register is cleared. This bit is set when:</p> <ul style="list-style-type: none"><li>- Bit &lt;eEDMAFBS&gt; in EDMA Configuration Register is set and</li><li>- Bit &lt;BasicDMAActive&gt; is cleared and</li><li>- EOT for that region is set in the region descriptor</li></ul> <p>This bit is cleared when:</p> <ul style="list-style-type: none"><li>- Bit &lt;BasicDMAActive&gt; is cleared and</li><li>- EOT for that region is cleared in the region descriptor.</li></ul> <p>0 = Halts before: DMA halts before the last data region 1 = Halts in: DMA halts in the last data region</p>
31:4	Reserved	RSVD 0x0	Reserved

**Table 329: Descriptor Table Low Base Address Register**  
Offset: Port0: 0x8222C Port1: 0x8422C

Bit	Field	Type/InitVal	Description
<p><b>NOTE:</b> Enhanced Physical Region Descriptors are in use to enable 64-bit memory addressing. See "EDMA Physical Region Descriptors (ePRD) Table Data Structure". The CPU accesses this register for direct access to the device when the EDMA is disabled.</p>			
3:0	Reserved	RSVD 0x0	Reserved
31:4	Descriptor Table Base Address	RW 0x0	The Descriptor Table Base Address corresponds to address A[31:4]. The descriptor table must be 16-byte aligned.

**Table 330: Descriptor Table High Base Address Register**  
Offset: Port0: 0x82230 Port1: 0x84230

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> Enhanced Physical Region Descriptors are in use to enable 64-bit memory addressing. See "EDMA Physical Region Descriptors (ePRD) Table Data Structure".			
31:0	Descriptor Table Base Address	RW 0x0	The Descriptor Table Base Address corresponds to address A[63:32]

**Table 331: Data Region Low Address Register**  
Offset: Port0: 0x82234 Port1: 0x84234

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The CPU accesses this register for direct access to the device when the EDMA is disabled. Bit <eEnEDMA> in EDMA Command Register (see Table 310 on page 380) is cleared. While the EDMA is enabled, the host must not write this register. If any of these bits are written when bit <eEnEDMA> in EDMA Command Register is set, the write transaction will cause unpredictable behavior.			
31:0	DataRegion[31:0]	RW 0x0	<p><b>Data Region</b>  This DWORD contains bit [31:1] of the current physical region starting address. Bit 0 must be 0.</p> <p>This field is updated by the DMA and indicates the completion status of the DMA when bit &lt;BasicDMAActive&gt; in the Basic DMA Status Register is cleared to 0.</p> <p>The host must not write to this bit when the Basic DMA is active.</p>

**Table 332: Data Region High Address Register**  
Offset: Port0: 0x82238 Port1: 0x84238

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The CPU accesses this register for direct access to the device when the EDMA is disabled. Bit <eEnEDMA> in EDMA Command Register is cleared. While the EDMA is enabled, the host must not write this register. If any of these bits are written when bit <eEnEDMA> in EDMA Command Register is set, the write transaction will cause unpredictable behavior			
31:0	DataRegion[63:32]	RW 0x0	<p><b>Data Region</b>  This DWORD contains bits [64:32] of the current physical region starting address.</p> <p>This field is updated by the DMA and indicates the completion status of the DMA when bit &lt;BasicDMAActive&gt; in the Basic DMA Status Register is cleared to 0.</p> <p>The host must not write to this bit when the Basic DMA is active.</p>



## A.7.2 EDMA Registers

**Table 333: EDMA Configuration Register**  
Offset: Port0: 0x82000 Port1: 0x84000

Bit	Field	Type/InitVal	Description
4:0	Reserved	RW 0x1F	Reserved
5	eSATANatvCmdQue	RW 0x0	EDMA SATA Native Command Queuing. When EDMA uses SATA Native Command Queuing, this bit must be set to 1. When this bit is set, bit[9] <eQue> is ignored. 0 = Disable: Native Command Queuing is not in use. 1 = Enable: Native Command Queuing is in use.
7:6	Reserved	RW 0x0	Reserved
8	eRdBSz	RW 0x0	EDMA Burst Size This bit sets the maximum burst size initiated by the SATAHC-DMA to the memory. This bit is related to read operation. <b>NOTE:</b> This field must be set to 0. 0 = 128B 1 = Reserved
9	eQue	RW 0x0	EDMA Queued When EDMA uses queued DMA commands, this bit must be set to 1. 0 = Non-Queued: Non-Queued DMA commands are in use 1 = Queued: Only Queued DMA commands are in use
10	Reserved	RW 0x0	Reserved
11	eRdBSzExt	RO 0x0	EDMA Burst Size Ext. This bit sets the maximum burst size initiated by the SATAHC-DMA to the memory. This bit is related to the read operation. <b>NOTE:</b> This field must be set to 0. 0 = Configured: EDMA Burst size is configured according to bit <eRdBSz> 1 = Reserved
12	Reserved	RW 0x1	Reserved This field must be set to 0x1
13	eWrBufferLen	RO 0x0	EDMA Write Buffers Length This bit defines the length of the write buffers. <b>NOTE:</b> This field must be set to 0. 0 = 256B: Write buffer is 256B 1 = Reserved
15:14	Reserved	RW 0x0	Reserved

**Table 333: EDMA Configuration Register (Continued)**  
Offset: Port0: 0x82000 Port1: 0x84000

Bit	Field	Type/InitVal	Description
16	eEDMAFBS	RW 0x0	EDMA FIS (Frame Information Structure) Based Switching When cleared to 0, the EDMA issue a new command only when the previous command was completed or released by the drive. When set to 1, EDMA issues a new command to the drive as soon as the target device is available, regardless to the status of all of the other devices.
17	eCutThroughEn	RW 0x0	This bit enables the Basic DMA cut-through operation when writing data to the drive. When the maximum read burst size initiated by the SATAHC-DMA to the memory in a read is limited to 128B (bits [8] <eRdBSz> and [11] <eRdBSzExt> in this register are both 0), the value of the <eCutThroughEn> bit is ignored and the cut-through operation is disabled. 0 = Store/forward 1 = Cut through
18	eEarlyCompletionEn	RW 0x0	This bit enables Basic DMA Early completion. When this bit is set to 1, Basic DMA Early completion is enabled. The DMA completes operation when all data is transferred from the disk to the memory, and it updates the following bits immediately, instead of waiting for this data to be visible in the system memory: Bit <BasicDMAActive> in the Basic DMA Status Register is cleared to 0. When EDMA is disabled, Bits[3:0] <DMAxDone> in the SATAHC Interrupt Cause Register is set to 1. Regardless to this bit value, when EDMA is enabled, the EDMA ensures all data is visible in system memory and only then it sets bit <SaCrbp0Done> in the EDMA Interrupt Error Cause Register. When EDMA controls the Basic DMA, it is recommended to set this bit to 1. When the CPU controls the Basic DMA directly, it is recommended to clear this bit to 0. 0 = Disable: Early DMA completion disabled 1 = Enable: Early DMA completion enabled
19	Reserved	RO 0x0	Reserved
21:20	Reserved	RW 0x0	Reserved
22	eHostQueueCacheEn	RW 0x0	EDMA Host Queue Cache Enable When set to 1, the EDMA is capable of storing up to four commands internally, before sending the commands to the drive. This bit enables the EDMA to send multiple commands across multiple drives much faster. It also enables the EDMA to send commands to any available drive, while other drives are busy executing previous commands. 0 = Disable: Host Queue Cache is disabled. EDMA stores a single command internally. Only when the command is sent to the drive, it fetches a new command from the CRQB. 1 = Enable: Host Queue Cache is enabled.
23	eMaskRxPM	RW 0x0	This bit masks the PM field in the incoming FISs 0 = Mask: Mask the PM field in incoming FISs 1 = No mask: Do not mask

**Table 333: EDMA Configuration Register (Continued)**  
Offset: Port0: 0x82000 Port1: 0x84000

Bit	Field	Type/InitVal	Description
24	ResumeDis	RW 0x0	When set to 1, the EDMA never sets bit <ContFromPrev> in the Basic DMA Command Register. When cleared to 0, the EDMA sets bit <ContFromPrev> in the Basic DMA Command Register whenever possible.
25	Reserved	RW 0x0	Reserved
26	eDMAFBS	RW 0x0	<p>Port Multiplier, FIS Based Switching mode. When cleared to 0, the Basic DMA continues the data transfer until the end of the PRD table. Then, it clears bit &lt;BasicDMAActive&gt;, clears bit &lt;BasicDMAPaused&gt;, and halts. These two bits are in the Basic DMA Status Register. When this bit is set to 1, the Basic DMA stops on the FIS boundary.</p> <p>During a write command, the Basic DMA:</p> <ol style="list-style-type: none"><li>1. Completes the 8-KB FIS transmission to the drive (Max frame transmission size can be change by the &lt;TransFrmSiz&gt; field in the Serial-ATA Interface Test Control Register).</li><li>2. Clears bit &lt;BasicDMAActive&gt;.</li><li>3. Sets bit &lt;BasicDMAPaused&gt; if the command was not completed.</li><li>4. Halts.</li></ol> <p>During a read command, the Basic DMA:</p> <ol style="list-style-type: none"><li>1. Completes the data transfer associates with a single FIS reception.</li><li>2. If bit &lt;eEarlyCompletionEn&gt; is set it waits for data visible in the system memory.</li><li>3. Clears bit &lt;BasicDMAActive&gt;.</li><li>4. Sets bit &lt;BasicDMAPaused&gt; if command was not completed.</li><li>5. Halts.</li></ol> <p><b>NOTE:</b> This bit must be set to 1 when FIS based switching mode is used. See "Port Multiplier-FIS-Based Switching".</p> <p><b>NOTE:</b> For the best Performance when a single drive is connected, clear this bit to 0.</p> <p>When bit[16] &lt;eEDMAFBS&gt; in this register is set to 1, the Basic DMA ignores the value of this bit, and a value of 1 is assumed.</p> <p>0 = Disable: FIS based Switching mode disabled. Basic DMA stops on a command (PRD) boundary.</p> <p>1 = Enable: FIS Based Switching mode enabled. Basic DMA stops on a FIS boundary.</p>
31:27	Reserved	RSVD 0x0	Reserved

**Table 334: EDMA Interrupt Error Cause Register**  
Offset: Port0: 0x82008 Port1: 0x84008

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> A corresponding cause bit is set every time that an interrupt occurs. A write of 0 clears the bit. A write of 1 has no effect.			
1:0	Reserved	RW0C 0x0	Reserved
2	eDevErr	RW0C 0x0	EDMA Device Error This bit is set to 1 when: 1. The EDMA is enabled (i.e., bit <eEnEDMA> in the EDMA Command Register is set to 1) and 2. Register - Device to Host FIS (Frame Information Structure) or Set Device Bits FIS is received with bit <ERR> set to 1. See "Device Error Indications".
3	eDevDis	RW0C 0x0	EDMA Device Disconnect This bit is set to 1 if the device is disconnected. <b>NOTE:</b> After DevDis interrupt, device hard reset is required. Set the <eAtaRst>bit[2] in the EDMA Command Register to 1. See "Device Disconnect". 0 = Not disconnected: Device was not disconnected. 1 = Disconnected: Device was disconnected.
4	eDevCon	RW0C 0x0	EDMA Device Connected This bit is set to 1 if the device was disconnected and is connected again.  0 = Not reconnected: Device was not reconnected. 1 = Reconnected: Device was reconnected.
5	SerrInt	RW0C 0x0	This bit is set to 1 when at least one bit in SError Register is set to 1 and the corresponding bit in SError Interrupt Mask Register is enabled. See "SError Register Errors". <b>NOTE:</b> This bit should be cleared only after clearing the SError Register. 0 = Not set: A bit in SError Register was not set to 1. 1 = Set: A bit in SError Register was set to 1.
6	Reserved	RW0C 0x0	Reserved
7	eSelfDis	RW0C 0x0	EDMA Self Disable This bit is set to 1 if the EDMA encounters error and clears bit <eEnEDMA>. 0 = No self disable: EDMA was not self disabled 1 = Self disabled: EDMA was self disabled
8	eTranslnt	RW0C 0x0	Transport Layer Interrupt This bit is set when at least one bit in the FIS Interrupt Cause Register is set to 1 and the corresponding bit in the FIS Interrupt Mask Register is set to 1 (enabled). <b>NOTE:</b> This bit should be cleared only after clearing the FIS Interrupt Cause Register. 0 = No wait: Transport layer does not wait for host 1 = Wait: Transport layer waits for host



**Table 334: EDMA Interrupt Error Cause Register (Continued)**  
Offset: Port0: 0x82008 Port1: 0x84008

Bit	Field	Type/InitVal	Description
11:9	Reserved	RW0C 0x0	Reserved
12	eIORdyErr	RW0C 0x0	EDMA IORdy Error IORdy timeout occurred. See "EDMA IORdy Timeout Register". <b>NOTE:</b> This bit is only set when the EDMA is disabled.
16:13	LinkCtlRxErr	RW0C 0x0	Link Control Receive Error This field indicates when a control FIS is received with errors. Bit [0] of this field (i.e., bit [13] of this register) is set to 1 when a SATA CRC error occurs. Bit [1] of this field is set to 1 when an internal FIFO error occurs. Bit [2] of this field is set to 1 when the Link Layer is reset (to Idle state) by the reception of SYNC primitives from the device. Bit [3] of this field is set to 1 when Link state errors, coding errors, or running disparity errors occur during FIS reception. See "Serial-ATA II Link Layer Error During Reception of a Control Frame".
20:17	LinkDataRxErr	RW0C 0x0	Link Data Receive Error This field indicates when a data FIS is received with errors. Bit [0] of this field (i.e., bit [17] of this register) is set to 1 when a SATA CRC error occurs. Bit [1] of this field is set to 1 when an internal FIFO error occurs. Bit [2] of this field is set to 1 when the Link Layer is reset (to Idle state) by the reception of SYNC primitives from the device. Bit [3] of this field is set to 1 when Link state errors, coding errors, or running disparity errors occur during FIS reception. See "Serial-ATA II Link Layer Error During Reception of a Data Frame".
25:21	LinkCtlTxErr	RW0C 0x0	Link Control Transmit Error This field indicates when a control FIS is transmitted with errors. Bit [0] of this field (i.e., bit [21] of this register) is set to 1 when a SATA CRC error occurs. Bit [1] of this field is set to 1 when an internal FIFO error occurs. Bit [2] of this field is set to 1 when the Link Layer is reset (to Idle state) by the reception of SYNC primitives from the device. Bit [3] of this field is set to 1 when the Link Layer accepts a DMAT primitive from the device. Bit [4] of this field is set to 1 to indicate when FIS transmission is aborted due to collision with Rx traffic. See Section "Serial-ATA II Link Layer Error During Transmission of a Control Frame".

**Table 334: EDMA Interrupt Error Cause Register (Continued)**  
Offset: Port0: 0x82008 Port1: 0x84008

Bit	Field	Type/InitVal	Description
30:26	LinkDataTxErr	RW0C 0x0	<p>Link Data Transmit Error  This field indicates when a data FIS is transmitted with errors.  Bit [0] of this field (i.e., bit [26] of this register) is set to 1 when a SATA CRC error occurs.  Bit [1] of this field is set to 1 when an internal FIFO error occurs.  Bit [2] of this field is set to 1 when the Link Layer is reset (to Idle state) by the reception of SYNC primitives from the device.  Bit [3] of this field is set to 1 when the Link Layer accepts a DMAT primitive from the device.  Bit [4] of this field is set to 1 to indicate when FIS transmission is aborted due to collision with Rx traffic.  See "Serial-ATA II Link Layer Error During Transmission of a Data Frame".  .</p>
31	TransProtErr	RW0C 0x0	<p>Transport Protocol Error  This bit is set when a control FIS is received with a non transient transport protocol error.  This bit is set when a:  - Link error indication is set during reception of a DMA/PIO data frame or control frame (i.e., when the Link Layer is reset to Idle state by the reception of SYNC primitives from the device).  During control frame reception, bit [15] in this register is also set to 1.  During data frame reception, bit [19] in this register is also set to 1.  - Control frame is too short or too long.  - Incorrect FIS type  -Illegal transfer count on a PIO transaction  See "Serial-ATA II Transport Layer Protocol Non Transient Errors".</p>

**Table 335: EDMA Interrupt Error Mask Register**  
Offset: Port0: 0x8200C Port1: 0x8400C

Bit	Field	Type/InitVal	Description
31:0	eIntErrMsk	RW 0x0	<p>EDMA Interrupt Error Mask Bits  Each of these bits checks the corresponding bit in the EDMA Interrupt Error Cause Register, and if these bits are set (0), they mask the interrupt.  0 = Mask  1 = No Mask</p>

**Table 336: EDMA Request Queue Base Address High Register**  
Offset: Port0: 0x82010 Port1: 0x84010

Bit	Field	Type/InitVal	Description
31:0	eRqQBA[63:32]	RW 0x0	The EDMA Request Queue Base Address corresponds to bits [63:32].



**Table 337: EDMA Request Queue In-Pointer Register**  
Offset: Port0: 0x82014 Port1: 0x84014

Bit	Field	Type/InitVal	Description
4:0	Reserved	RSVD 0x0	Reserved
9:5	eRqQIP	RW 0x0	EDMA Request Queue In-Pointer The EDMA request queue in-pointer is written/increment by the system driver to indicate that a new CRQB has been placed on the request queue. The EDMA compares the EDMA Request Queue In-Pointer to the EDMA Request Queue Out-Pointer to determine when the request queue is empty. If there are CRQBs in the request queue, then, when the EDMA is ready, it processes the commands.
31:10	eRqQBA[31:10]	RW 0x0	EDMA Request Queue Base Address corresponds to bits [31:10].

**Table 338: EDMA Request Queue Out-Pointer Register**  
Offset: Port0: 0x82018 Port1: 0x84018

Bit	Field	Type/InitVal	Description
4:0	Reserved	RSVD 0x0	Reserved
9:5	eRqQOP	RW 0x0	EDMA Request Queue Out Pointer The EDMA request queue out-pointer is updated/increment by the EDMA each time a CRQB is copied from the command queue into the EDMA internal memory. The system driver reads this register to determine if the request queue is full.
11:10	Reserved	RW 0x0	Reserved
31:12	Reserved	RSVD 0x0	Reserved

**Table 339: EDMA Response Queue Base Address High Register**  
Offset: Port0: 0x8201C Port1: 0x8401C

Bit	Field	Type/InitVal	Description
31:0	eRpQBA[63:32]	RW 0x0	The EDMA Response Queue Base Address corresponds to bits [63:32].

**Table 340: EDMA Response Queue In-Pointer Register**  
Offset: Port0: 0x82020 Port1: 0x84020

Bit	Field	Type/InitVal	Description
2:0	Reserved	RSVD 0x0	Reserved

**Table 340: EDMA Response Queue In-Pointer Register (Continued)**  
Offset: Port0: 0x82020 Port1: 0x84020

Bit	Field	Type/InitVal	Description
7:3	eRpQIP	RW 0x0	EDMA Response Queue In-Pointer The EDMA response queue in-pointer is updated/increment by the EDMA each time a command execution is completed and a new CRPB is moved from the EDMA internal memory to the response queue by the EDMA. The system driver compares the EDMA Response Queue In-Pointer with the EDMA Response Queue Out-Pointer to determine if there is a CRPB in the response queue that needs processing.
9:8	Reserved	RW 0x0	Reserved
31:10	Reserved	RSVD 0x0	Reserved

**Table 341: EDMA Response Queue Out-Pointer Register**  
Offset: Port0: 0x82024 Port1: 0x84024

Bit	Field	Type/InitVal	Description
2:0	Reserved	RSVD 0x0	Reserved
7:3	eRPQOP	RW 0x0	EDMA Response Queue Out-Pointer The EDMA response queue out-pointer is updated/increment by the system driver after the driver completes processing the CRPB pointed to by this register. The EDMA compares the EDMA Response Queue Out-Pointer to the EDMA Response Queue In-Pointer to determine if the response queue is full.
31:8	eRPQBA[31:8]	RW 0x0	The EDMA Response Queue base address corresponds to bits [31:8]

**Table 342: EDMA Command Register**  
Offset: Port0: 0x82028 Port1: 0x84028

Bit	Field	Type/InitVal	Description
0	eEnEDMA	RW 0x0	Enable EDMA When this bit is set to 1, the EDMA is activated. All control registers, request queues, and response queues must be initialized before this bit is set to 1. The EDMA clears this bit when any of the following events occur: <ul style="list-style-type: none"> <li>- Bit &lt;eDsEDMA&gt; (bit [1]) is set.</li> <li>- An error condition occurs and the corresponding bit in EDMA Halt Conditions Register is not masked.</li> <li>- Link is down and the corresponding bit in EDMA Halt Conditions Register is not masked .</li> </ul> Writing 0 to this bit is ignored. 0 = Idle: The EDMA is idle. 1 = Active: The EDMA is active.



**88F6180/88F619x/88F6281**  
**Functional Specifications**

**Table 342: EDMA Command Register (Continued)**  
Offset: Port0: 0x82028 Port1: 0x84028

Bit	Field	Type/InitVal	Description
1	eDsEDMA	SC 0x0	Disable EDMA This bit is self-negated. When this bit is set to 1, the EDMA aborts the current Command and then clears bit <eEnEDMA> (bit [0]). If the EDMA operation is aborted during command execution, the host software must set bit <eAtaRst> (bit [2]) to recover.
2	eAtaRst	RW 0x0	ATA Device Reset When this bit is set to 1, Serial-ATA transport, link, and physical layers are reset. All Serial-ATA Interface Registers (see "Serial-ATA Interface Registers Map") are reset, except the Serial-ATA Interface Configuration Register, and the OOB COMRESET signal is sent to the device, after configuring the <DET> field in the SControl Register. Host must not read/write Serial-ATA Interface Registers. If Host initiates a read/write to Serial-ATA Interface Registers when this bit is set, the transaction gets stuck until EDMA IORDy Timeout Register expires. - When this bit is set and EDMA is enabled (bit <eEnEDMA> in the EDMA Command Register is set to 1) the EDMA operation must be aborted: Set bit <eDsEDMA> in the EDMA Command Register to 1. - When this bit is set and the Basic DMA is enabled (bit <Start> in Basic DMA Command Register is set to 1) the Basic DMA operation must be aborted: Clear bit <Start> in Basic DMA Command Register to 0. 0 = Normal: Normal operation. 1 = Reset: Device reset.
3	Reserved	RW 0x0	Write only 0x0.
4	eEDMAFrz	RW 0x0	EDMA Freeze When this bit is set, EDMA does not pull new commands from CRQB. If commands are pending in EDMA cache, these commands are sent to the drive regardless to this bit value. 0 = Pull: Pull new commands from CRQB and insert them into the drive. 1 = Do not pull: Do not pull new commands from CRQB.
31:5	Reserved	RSVD 0x0	Reserved

**Table 343: EDMA Status Register**  
Offset: Port0: 0x82030 Port1: 0x84030

Bit	Field	Type/InitVal	Description
4:0	eDevQueTAG	RO 0x0	EDMA Tag This field indicates the tag of the last command used by the EDMA. The EDMA updates this field with field <cDeviceQueTag> of the CRQB: - When a new command is written to the device, OR - When the tag is read from the device after a service command (TCQ), or - When the tag is read from the device in DMA Setup (SU) FIS (NCQ).

**Table 343: EDMA Status Register (Continued)**  
Offset: Port0: 0x82030 Port1: 0x84030

Bit	Field	Type/InitVal	Description
5	eDevDir	RO 0x0	<p>Device Direction</p> <p>The EDMA updates this field with field &lt;cDIR&gt; of the CRQB when a new command is written to the device, or when the tag is read from the device after a service command (TCQ), or when the tag is read from the device in DMA Setup FIS (NCQ).</p> <p>0 = Mem to device: System memory to device 1 = Device to mem: Device to system memory</p>
6	eCacheEmpty	RO 0x1	<p>Cache Empty</p> <p>This field indicates if EDMA cache is empty.</p> <p>0 = Not empty 1 = Empty</p>
7	EDMAidle	RO 0x0	<p>This bit is set to 1 when all of the following conditions occur:</p> <ul style="list-style-type: none"> <li>- No commands are pending in EDMA request queue AND</li> <li>- All command EDMAs taken from the EDMA request queue were delivered to the drive AND</li> <li>- All command completions EDMA received from the drive were delivered to the host response queue AND</li> <li>- All commands sent to the drives were either completed or released AND</li> <li>- EDMA is in idle state.</li> <li>- EDMA is enabled.</li> </ul> <p><b>NOTE:</b> This bit may be set when a command is still pending in the drive.</p>
15:8	eSTATE	RO 0x11	<p>EDMA State</p> <p>These bits indicate the current state of the machine.</p>
21:16	eIOId	RO 0x0	<p>EDMA IO ID</p> <p>This field indicates the IO ID of the last command used by the EDMA. The EDMA updates this field when a new command is written to the device; or, When the device sends a completion FIS (NonQ, TCQ -- RegD2H with &lt;REL&gt; bit cleared and &lt;DRQ&gt; bit cleared. NCQ -- reception of SDB FIS); or, When the DMA is activated (after a reception of DMA Activate FIS or a reception of a DATA FIS).</p>
31:22	Reserved	RSVD 0x0	Reserved



**Table 344: EDMA IORDy Timeout Register**  
Offset: Port0: 0x82034 Port1: 0x84034

Bit	Field	Type/InitVal	Description
15:0	eIORDyTimeout	RW 0xBC	<p>EDMA IORDy Signal Timeout Value If SATAHC unit is not ready to complete the transaction for the number of system cycles set in this field, a timeout will occur and the transaction will be completed. If the transaction is terminated as a result of the IORDy timeout, bit &lt;eIORDyErr&gt; is set.</p> <p>The Serial-ATA spec defines the IORDY time-out value to be 1250 ns. The current default value (0xBC) is calculated with the assumption that the SysClock == 150 MHz (1/150M = 6.666 ns) =&gt; 6.666 * 188 (0xBC) = 1250 ns. When the SysClock is 200 MHz (5 nSec), the value of this field should be =&gt; 1250/5ns = 0xFA</p> <p>When clearing this field to 0 =&gt; eIORDy, timeout is ignored; the transaction will not be aborted.</p> <p>When setting this field to x (x &gt; 0) =&gt;, timeout occurs after x* system clocks.</p>
31:16	Reserved	RSVD 0x0	Reserved

**Table 345: EDMA Command Delay Threshold Register**  
Offset: Port0: 0x82040 Port1: 0x84040

Bit	Field	Type/InitVal	Description
15:0	CmdDelayThrshd	RW 0x0	This field indicates the length of delay to insert before sending a command to the drive when [31], <CMDDataoutDelayEn> is enabled.
30:16	Reserved	RSVD 0x0	Reserved
31	CMDDataoutDelayEn	RW 0x0	<p>When this bit is set to 1, when the DMA completes write data transaction, the content of field [15:0] &lt;CmdDelayThrshd&gt; is written to a 16-bit counter. This counter is decremented every 16 clock cycles until it reaches 0. Then it stops.</p> <p>A new command is issued to the drive only when the value of this counter is 0 (i.e., a delay of &lt;CmdDelayThrshd&gt; *16 clock cycles is inserted after the data transaction from the system memory to the drive completes and before it issues a new command to the drive).</p> <p>0 = Disabled: No delay is inserted before command retransmission.</p> <p>1 = Enabled: A delay is inserted before command retransmission.</p>

**Table 346: EDMA Halt Conditions Register**  
Offset: Port0: 0x82060 Port1: 0x84060

Bit	Field	Type/InitVal	Description
31:0	eHaltMask	RW 0xFC1E0E1F	EDMA halts when any bit is set to 1 in the EDMA Interrupt Error Cause Register, and is not masked by the corresponding bit in this field.

**Table 347: EDMA NCQ0 Done/TCQ0 Outstanding Status Register**  
Offset: Port0: 0x82094 Port1: 0x84094

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> When the EDMA is disabled, the fields in this register are Read Only.			
31:0	NCQDone_TCQOutstand	RW 0x0	<p>When in NCQ mode: NCQ Completion Status  Each bit represents a completed NCQ command corresponding to the host command ID. When set the NCQ command corresponding to the host command ID has been completed but not yet updated in the host response queue in system memory [CRPB].</p> <p>When in TCQ mode: TCQ Outstanding Commands Status  Each bit represents an outstanding command corresponding to the host command ID. When set, the command was sent to the device but it has not yet completed and updated in the host response queue in system memory [CRPB]. EDMA sets the corresponding bit when sent a new command, EDMA clears the corresponding bit when the command is completed and updated in the host response queue in system memory [CRPB].</p>

### A.7.3 SATAHC Arbiter Registers

**Table 348: SATAHC Configuration Register**  
Offset: 0x80000

Bit	Field	Type/InitVal	Description
7:0	Timeout	RW 0xFF	SATAHC interface Mbus Arbiter Timeout Preset Value
8	DmaBS	RW 0x1	Basic DMA Byte swap 0 = Byte swap 1 = No byte swap
9	EDmaBS	RW 0x1	EDMA Byte swap 0 = Byte swap 1 = No byte swap
10	PrdpBS	RW 0x1	PRDP Byte swap 0 = Byte swap 1 = No byte swap
15:11	Reserved	RSVD 0x0	Reserved
16	TimeoutEn	RW 0x1	Mbus Arbiter Timer Enable 0 = Enable 1 = Disable
23:17	Reserved	RSVD 0x0	Reserved



**Table 348: SATAHC Configuration Register (Continued)**  
Offset: 0x80000

Bit	Field	Type/InitVal	Description
24	CoalDis	RW 0x0	<p>Coalescing Disable</p> <p>When a bit in field &lt;CoalDis&gt; is set to 1, the completing indication of the corresponding port is ignored in the "SATAHC Interrupt Time Threshold Register".</p> <p>CoalDis[1]</p> <p>0 = Coalescing enabled for port 1</p> <p>1 = Coalescing disable for port 1</p> <p>CoalDis[2]</p> <p>0 = Coalescing enabled for port 2</p> <p>1 = Coalescing disable for port 3</p> <p>CoalDis[3]</p> <p>0 = Coalescing enabled for port 3</p> <p>1 = Coalescing disable for port</p>
31:25	Reserved	RSVD 0x0	Reserved

**Table 349: SATAHC Request Queue Out-Pointer Register**  
Offset: 0x80004

Bit	Field	Type/InitVal	Description
4:0	eRQQOP0	RO 0x0	<p>EDMA Request Queue Out-Pointer</p> <p>This field reflects the value of bits [9:5] in the EDMA Request Queue Out-Pointer Register located in the port.</p>
6:5	Reserved	RO 0x0	
31:7	Reserved	RSVD 0x0	Reserved

**Table 350: SATAHC Response Queue In-Pointer Register**  
Offset: 0x80008

Bit	Field	Type/InitVal	Description
4:0	eRPQIP0	RO 0x0	<p>EDMA Response Queue In-Pointer Register</p> <p>This field reflects the value of bits [7:3] in the EDMA Response Queue In-Pointer Register located in the port.</p>
6:5	Reserved	RO 0x0	Reserved
31:7	Reserved	RSVD 0x0	Reserved

**Table 351: SATAHC Interrupt Coalescing Threshold Register**  
**Offset: 0x8000C**

Bit	Field	Type/InitVal	Description
7:0	SAICOALT	RW 0x0	<p>SATA Interrupt Coalescing Threshold</p> <p>This field provides a way to minimize the number of interrupts to off load the CPU. It defines the number of SaCrpbXDone indications before asserting the &lt;SalntCoal&gt; bit in the SATAHC Interrupt Cause Register.</p> <p>Once the accumulated number of SaCrpbXDone indications provided by all SATAHC ports reaches the SAICOALT value, a &lt;SalntCoal&gt; interrupt is asserted.</p> <p>When SalntCoal is negated or when the SATAHC Interrupt Coalescing Threshold Register is written, the interrupts counter is cleared.</p> <p>0x0 =&gt; Assertion of SaCrpbXDone causes an immediate assertion of &lt;SalntCoal&gt;.</p> <p>0x1-0xFF =&gt; &lt;SalntCoal&gt; assertion is provided for every ,ÄoSAICOALT' times of SaCrpbXDone assertion.</p>
31:8	Reserved	RSVD 0x0	Reserved

**Table 352: SATAHC Interrupt Time Threshold Register**  
**Offset: 0x80010**

Bit	Field	Type/InitVal	Description
23:0	SAITMTH	RW 0x0	<p>SATA Interrupt Time Threshold</p> <p>This field provides a way to ensure maximum delay between &lt;SaCrpb0Done&gt; assertion and assertion bit &lt;SalntCoal&gt; in SATAHC Interrupt Cause Register (even if the number of &lt;SaCrpb0Done&gt; indications did not reach the &lt;SAICOALT&gt; value).</p> <p>When &lt;SalntCoal&gt; is negated or when the SATAHC Interrupt Time Threshold Register is written, the down counter is cleared. A new count is enabled in the assertion of the next &lt;SaCrpb0Done&gt; indication.</p> <p>When set to 0 = Assertion of &lt;SaCrpb0Done&gt; causes an immediate assertion of bit &lt;SalntCoal&gt;.</p> <p>When set from 1-23 = Up to &lt;n&gt; internal clocks between assertion of &lt;SaCrpb0Done&gt; and assertion of &lt;SalntCoal&gt;.</p>
31:24	Reserved	RSVD 0x0	Reserved

**Table 353: SATAHC Interrupt Cause Register**

Offset: 0x80014

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> A corresponding cause bit is set every time that an interrupt occurs. A write of 0 clears the bit. A write of 1 has no effect.			
0	SaCrpb0Done/DMA0Done	RW0C 0x0	<p>SATA CRPB 0 Done / Basic DMA 0 Done</p> <p>When EDMA is enable: (Bit &lt;eEnEDMA&gt; in the EDMA Command Register is set.)</p> <p>This bit is set when the EDMA in the port places a new CRPB in the response queue.</p> <p>0 = A new CRPB was not placed in the response queue. 1 = A new CRPB was placed in the response queue.</p> <p>When EDMA is disabled: (Bit eEnEDMA in the EDMA Command Register is cleared.)</p> <p>This bit is set when the Basic DMA in the port completes the data transfer, clears bit &lt;BasicDMAActive&gt;, and moves to idle state.</p> <p>0 = Basic DMA has not completed the data transfer. 1 = Basic DMA completed the data transfer.</p> <p>0 = NotCompleted: Basic DMA has not completed the data transfer. 1 = Completed: Basic DMA completed the data transfer.</p>
1	SaCrpb1Done/DMA1Done	RW0C 0x0	<p>SATA CRPB 1 Done / Basic DMA 1 Done</p> <p>When EDMA is enable: (Bit &lt;eEnEDMA&gt; in the EDMA Command Register is set.)</p> <p>This bit is set when the EDMA in the port places a new CRPB in the response queue.</p> <p>0 = A new CRPB was not placed in the response queue. 1 = A new CRPB was placed in the response queue.</p> <p>When EDMA is disabled: (Bit eEnEDMA in the EDMA Command Register is cleared.)</p> <p>This bit is set when the Basic DMA in the port completes the data transfer, clears bit &lt;BasicDMAActive&gt;, and moves to idle state.</p> <p>0 = Basic DMA has not completed the data transfer. 1 = Basic DMA completed the data transfer.</p> <p>0 = NotCompleted: Basic DMA has not completed the data transfer. 1 = Completed: Basic DMA completed the data transfer.</p>
3:2	Reserved	RSVD 0x0	Reserved
4	SaIntCoal	RW0C 0x0	<p>SATA Interrupt Coalescing</p> <p>This bit is set:</p> <p>When the accumulated number of &lt;SaCrpb0Done&gt; indications from the ports that participate in the coalescing mechanism according to &lt;CoalDis&gt; setting in the SATAHC Configuration Register - since the last &lt;SaIntCoal&gt; negation - reaches the value set in the SATAHC Interrupt Coalescing Threshold Register, or</p> <p>When the time from first &lt;SaCrpb0Done&gt; assertion after &lt;SaIntCoal&gt; negation reaches the value set in the SATAHC Interrupt Time Threshold Register.</p> <p>0 = No occurrence: Cause for Interrupt Coalescing did not occur. 1 = Occurs: Cause for Interrupt Coalescing occurs.</p>
7:5	Reserved	RSVD 0x0	Reserved

**Table 353: SATAHC Interrupt Cause Register (Continued)**  
Offset: 0x80014

Bit	Field	Type/InitVal	Description
8	SaDevInterrupt0	RW0C 0x0	SATA Device Interrupt This bit is set if bit <eEnEDMA> in the EDMA Command Register is cleared, and the ATA interrupt line in port 0 is active. 0 = Inactive: The ATA interrupt line was not active. 1 = Active: The ATA interrupt line was active.
9	SaDevInterrupt1	RW0C 0x0	SATA Device Interrupt This bit is set if bit <eEnEDMA> in the EDMA Command Register is cleared, and the ATA interrupt line in port 1 is active. 0 = Inactive: The ATA interrupt line was not active. 1 = Active: The ATA interrupt line was active.
31:10	Reserved	RSVD 0x0	Reserved

**Table 354: SATAHC Main Interrupt Cause Register**  
Offset: 0x80020

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The bits in this register mirror the interrupt indications coming from the other SATAHC interrupt cause registers.			
0	Sata0Err	RO 0x0	SATA port0 error
1	Sata0Done	RO 0x0	SATA port0 command done This bit is set when one of the following occurs: <SaCrpb0Done/DMA0Done> bit in SATAHC Interrupt Cause Register is set. <SaDevInterrupt0> in SATAHC Interrupt Cause Register is set.
2	Sata1Err	RO 0x0	SATA port1 error
3	Sata1Done	RO 0x0	SATA port1 command done This bit is set when one of the following occurs: <SaCrpb1Done/DMA1Done> bit in SATAHC Interrupt Cause Register is set. <SaDevInterrupt1> in SATAHC Interrupt Cause Register is set.
4	Sata0DmaDone	RO 0x0	SATA port0 DMA done This bit is set when both of the following occur: <SaCrpb0Done/DMA0Done> bit in SATAHC Interrupt Cause Register of SATAHC is set. <SaDevInterrupt0> in SATAHC Interrupt Cause Register of SATAHC is set.
5	Sata1DmaDone	RO 0x0	SATA port1 DMA done This bit is set when both of the following occur: <SaCrpb1Done/DMA1Done> bit in SATAHC Interrupt Cause Register of SATAHC is set. <SaDevInterrupt1> in SATAHC Interrupt Cause Register of SATAHC is set.



**Table 354: SATAHC Main Interrupt Cause Register (Continued)**  
Offset: 0x80020

Bit	Field	Type/InitVal	Description
7:6	Reserved	RO 0x0	Reserved
8	SataCoalDone	RO 0x0	SATA ports coalescing done This bit is set when bit <SalntCoal> in the SATAHC Interrupt Cause Register, of SATAHC is set. <b>NOTE:</b> SATA completion bit <Sata0Done> must be mask to use this bit.
31:9	Reserved	RO 0x0	Reserved

**Table 355: SATAHC Main Interrupt Mask Register**  
Offset: 0x80024

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> Mask only affects the assertion of interrupt pins. It does not affect the setting of bits in the Cause register			
0	Sata0Err	RW 0x0	Mask SATA port0 error 0 = Mask: Interrupt is masked. 1 = Enable: Interrupt is enabled.
1	Sata0Done	RW 0x0	Mask SATA port0 command done 0 = Mask: Interrupt is masked. 1 = Enable: Interrupt is enabled.
2	Sata1Err	RW 0x0	Mask SATA port1 error 0 = Mask: Interrupt is masked. 1 = Enable: Interrupt is enabled.
3	Sata1Done	RW 0x0	Mask SATA port1 command done 0 = Mask: Interrupt is masked. 1 = Enable: Interrupt is enabled.
4	Sata0DmaDone	RW 0x0	Mask SATA port0 DMA done 0 = Mask: Interrupt is masked. 1 = Enable: Interrupt is enabled.
5	Sata1DmaDone	RW 0x0	Mask SATA port1 DMA done 0 = Mask: Interrupt is masked. 1 = Enable: Interrupt is enabled.
7:6	Reserved	RSVD 0x0	Reserved
8	SataCoalDone	RW 0x0	Mask SATA ports coalescing done 0 = Mask: Interrupt is masked. 1 = Enable: Interrupt is enabled.

**Table 355: SATAHC Main Interrupt Mask Register (Continued)**  
Offset: 0x80024

Bit	Field	Type/InitVal	Description
31:9	Reserved	RO 0x0	Reserved

**Table 356: SATAHC LED Configuration Register**  
Offset: 0x8002C

Bit	Field	Type/InitVal	Description
0	act_led_blink	RW 0x0	Active LED Blink 0 = As is: Use indication as is. 1 = Blinking: Set indication to blinking.
1	Reserved	RW 0x0	Reserved. Write only 0x1.
2	act_presence	RW 0x0	Active Presence 0 = Only active indication: Use active indication only. 1 = Multiplex/presence indication: Multiplex active and presence indication on the same LED.
3	led_polarity	RW 0x0	LED Polarity 0 = Invert: Invert the active indication. 1 = No change: Do not change the active indication.
31:4	Reserved	RSVD 0x0	Reserved

**Table 357: Window0 Control Register**  
Offset: 0x80030

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window 0 Enable 0 = Disable 1 = Enable
1	WrBL	RW 0x0	Write to Mbus Burst Limit 0 = Limit: No limit (up to 128B burst). 1 = No limit: Limit not to cross 32B boundary.
3:2	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the target interface associated with this window. <b>NOTE:</b> Must be configured to the SDRAM Controller.



**Table 357: Window0 Control Register (Continued)**  
Offset: 0x80030

Bit	Field	Type/InitVal	Description
15:8	Attr	RW 0x0E	Specifies the target interface attributes associated with this window.
31:16	Size	RW 0x0FFF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as sequence of 1's followed by sequence of 0's. The number of 1's specifies the size of the window in 64 KByte granularity (e.g., a value of 0x0FFF specifies 256 MByte). <b>NOTE:</b> A value of 0x0 specifies 64-KByte size.

**Table 358: Window0 Base Register**  
Offset: 0x80034

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x0	Base Address Used with the <Size> field to set the address window size and location. Corresponds to transaction address[31:16].

**Table 359: Window1 Control Register**  
Offset: 0x80040

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window1 Enable See "Window0 Control Register".
1	WrBL	RW 0x0	Write to Mbus Burst Limit 0 = No Limit: No limit (up to 128B burst). 1 = Limit: Limit not to cross 32B boundary.
3:2	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See "Window0 Control Register".
15:8	Attr	RW 0x0D	Target specific attributes depending on the target interface. See "Window0 Control Register".
31:16	Size	RW 0x0FFF	Window Size See "Window0 Control Register".

**Table 360: Window1 Base Register**  
**Offset: 0x80044**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x1000	Base Address See "Window0 Base Register".

**Table 361: Window2 Control Register**  
**Offset: 0x80050**

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window2 Enable See "Window0 Control Register".
1	WrBL	RW 0x0	Write to Mbus burst limit 0 = No Limit: No limit (up to 128B burst). 1 = Limit: Limit not to cross 32B boundary.
3:2	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See "Window0 Control Register".
15:8	Attr	RW 0x0B	Target specific attributes depending on the target interface. See "Window0 Control Register".
31:16	Size	RW 0x0FFF	Window Size See "Window0 Control Register".

**Table 362: Window2 Base Register**  
**Offset: 0x80054**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x2000	Base Address See "Window0 Base Register".

**Table 363: Window3 Control Register**

Offset: 0x80060

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window3 Enable See "Window0 Control Register".
1	WrBL	RW 0x0	Write to Mbus Burst Limit 0 = No Limit: No limit (up to 128B burst). 1 = Limit: Limit not to cross 32B boundary.
3:2	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See "Window0 Control Register".
15:8	Attr	RW 0x07	Target specific attributes depending on the target interface. See "Window0 Control Register".
31:16	Size	RW 0x0FFF	Window Size See "Window0 Control Register".

**Table 364: Window3 Base Register**

Offset: 0x80064

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x3000	Base Address See "Window0 Base Register".

## A.7.4 Serial-ATA Interface Registers

**Table 365: Serial-ATA Interface Configuration Register**

Offset: Port0: 0x82050 Port1: 0x84050

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> After any modification in this register, the host must set bit[2]<eAtaRst> in EDMA Command Register.			
1:0	RefClkCnf	RW 0x01	PHY PLL Reference clock frequency. <b>NOTE:</b> Must be set to 01. 0 = 20 MHz 1 = 25 MHz 2 = 30 MHz 3 = 40 MHz

**Table 365: Serial-ATA Interface Configuration Register (Continued)**  
Offset: Port0: 0x82050 Port1: 0x84050

Bit	Field	Type/InitVal	Description
3:2	Reserved	RW 0x01	Reserved
5:4	Reserved	RW 0x01	Reserved
6	PhySSCEn	RW 0x0	SSC enable 0 = Disable 1 = Enable
7	Gen2En	RW 0x1	Generation 2 communication speed support. 0 = Disable 1 = Enable
8	CommEn	RW 0x0	PHY communication enable signal to override field <DET> (see "SControl Register" setting). 0 = No override: DET setting is not overridden. 1 = Override: If DET value is 0x4, its value is overridden with a value of 0x0.
9	PhyShutdown	RW 0x0	PHY shutdown. 0 = Functional mode 1 = Shutdown mode
10	TargetMode	RW 0x0	Target Mode. This bit defines the Serial-ATA port that functions as a target during Target mode operation. To move to Target mode, you need to clear this field and set bit[11] <ComChannel>. Any other option will result in the mode being Initiator mode. 0 = Target 1 = Initiator
11	ComChannel	RW 0x0	Communication Channel Operating Mode. This bit defines if the Serial-ATA port functions in Target mode operation.  <b>NOTE:</b> In Target mode, the ATA task registers are updated when the Register Device to Host FIS is received, regardless to the value of <BSY> bit in the ATA Status register (see "Shadow Register Block Registers Map"). 0 = Disk controller 1 = Target mode operation
12	Reserved	RW 0x1	Reserved
13	EMPH_LVLADJ_EN	RW 0x1	Emphasis Level Adjust 0 = Disable: (1/2pre + 1/2de); 1 = Enable
14	TX_EMPH_EN	RW 0x1	TX Emphasis Enable 0 = Disable 1 = Enable



**Table 365: Serial-ATA Interface Configuration Register (Continued)**  
Offset: Port0: 0x82050 Port1: 0x84050

Bit	Field	Type/InitVal	Description
15	EMPH_TYPE_PRE	RW 0x0	Emphasis Selection 0 = de-emphasis 1 = pre-emphasis
17:16	Reserved	RW 0x3	Reserved
18	Reserved	RW 0x0	Reserved, Must write 0x0 to this bit.
19	CLK_DET_EN	RW 0x1	1: Enable TX&RX auto recovery circuit.
20	Reserved	RW 0x1	Reserved
21	Reserved	RW 0x0	Reserved, Must write 0x0 to this bit.
22	Reserved	RW 0x0	Reserved, Must write 0x0 to this bit.
23	Reserved	RW 0x1	Reserved, Must write 0x1 to this bit.
24	IgnoreBsy	RW 0x0	When this bit is set to 1, the ATA task registers are updated when Register Device to Host FIS is received or when the host writes to the ATA Status registers, regardless of the value of the <BSY> bit in the ATA Status register (see "Shadow Register Block Registers Map"). When EDMA is enabled the transport layer ignores the value of this bit and assume a value of 1. This bit must be cleared before the host issues any PIO commands.
25	LinkRstEn	RW 0x0	When this bit is set to 1 and when bit <RST> is set to 1 in the ATA status register (see "Shadow Register Block Registers Map") in the middle of data FIS reception, the link layer responds with a SYNC primitive to reception of data FIS. When the transport layer receives SYNC in the back channel in response, it sends the Control FIS with the <SRST> bit set to 1. When this bit is cleared to 0 and when bit <RST> is set to 1 in the ATA status register, in a middle of data FIS reception, the transport layer drops the data of the incoming FIS. When the incoming data FIS is finished, it sends the Control FIS with the <SRST> bit set to 1.

**Table 365: Serial-ATA Interface Configuration Register (Continued)**  
Offset: Port0: 0x82050 Port1: 0x84050

Bit	Field	Type/InitVal	Description
26	CmdRetxDs	RW 0x0	<p>When this bit is cleared to 0 and Register Host to Device FIS transmission was not completed successfully as indicated by bits &lt;FISTxDone&gt;and &lt;FISTxErr&gt; in "FIS Interrupt Cause Register", the transport layer retransmits the Register Host to Device FIS.</p> <p>When this bit is set to 1 and Register Host to Device FIS transmission was not completed successfully, as indicated by bits &lt;FISTxDone&gt;and &lt;FISTxErr&gt; in "FIS Interrupt Cause Register", the transport layer does not retransmit the Register Host to Device FIS.</p> <p>When the EDMA is enabled, this bit value is ignored and assumed to be 1. EDMA retransmits the Register Host to Device FIS.</p>
31:27	Reserved	RSVD 0x0	Reserved

**Table 366: Serial-ATA PLL Configuration Register**  
Offset: Port0: 0x82054 Port1: 0x84054

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> After any modification in this register, the host must set bit[2]<eAtaRst> in EDMA Command Register.			
7:0	Gen1 FBDIV	RW 0xD2	<p>Gen1 PLL Feedback Divider  <math>F_{vco} = (F_{REQ}/REFDIV) * FBDIV</math></p> <p>Bit 7:6  00: div-by-5  01: div-by-2  10: div-by-4  11: div-by-3</p> <p>Bit 5:0  N+2</p>
9:8	Gen1 REFDIV	RW 0x0	<p>Gen1 Reference Frequency Divider  0 = by 1  1 = by 2  2 = by 4  3 = by 3</p>
12:10	Gen1 INTPI	RW 0x4	<p>Gen1 TX&amp;RX Interpolator Bias Current  Speed = 1  000: 20uA, where each step is 10uA</p> <p>Speed = 0  000: 10uA, where each step is 5uA.</p>



**Table 366: Serial-ATA PLL Configuration Register (Continued)**  
Offset: Port0: 0x82054 Port1: 0x84054

Bit	Field	Type/InitVal	Description
20:13	Gen2 FBDIV	RW 0x9c	Gen2 PLL Feedback Divider $F_{vco} = (F_{REQ}/REFDIV) * FBDIV$ Bit 7:6 00: div-by-5 01: div-by-2 10: div-by-4 11: div-by-3  Bit 5:0 N+2
22:21	Gen2 REFDIV	RW 0x0	Gen2 Reference Frequency Divider 0 = by 1 1 = by 2 2 = by 4 3 = by 3
25:23	Gen2 INTPI	RW 0x6	Gen2 TX&RX Interpolator Bias Current Speed = 1 000: 20uA, where each step is 10uA Speed = 0 000: 10uA, where each step is 5uA.
31:26	Reserved	RSVD 0x0	Reserved

**Table 367: SStatus Register**  
Offset: Port0: 0x82300 Port1: 0x84300

Bit	Field	Type/InitVal	Description
NOTE: See the Serial-ATA specification for a detailed description.			
3:0	DET	RO 0x4	These bits set the interface device detection and PHY state. All other values are reserved. 0 = No communication: No device detected, and PHY communication is not established. 1 = Device with no comm: Device presence detected, but PHY communication is not established. 3 = Device with comm: Device presence detected, and PHY communication is established. 4 = PHY offline: PHY in offline mode as a result of the interface being disabled or running in a loopback mode.

**Table 367: SStatus Register (Continued)**  
Offset: Port0: 0x82300 Port1: 0x84300

Bit	Field	Type/InitVal	Description
7:4	SPD	RO 0x0	<p>These bits set whether the negotiated interface communication speed is established.  All other values are reserved.</p> <p>0 = No negotiation: No negotiated speed. The device is not present or communication is not established.</p> <p>1 = Gen1 negotiation: Generation 1 communication rate negotiated.</p> <p>2 = Gen2 negotiation: Generation 2 communication rate negotiated.</p>
11:8	IPM	RO 0x0	<p>These bits set the current interface power management state.  All other values are reserved.</p> <p>0 = No communication: Device not present, or communication not established.</p> <p>1 = Active: Interface in active state.</p> <p>2 = Partial: Interface in PARTIAL power management state.</p> <p>6 = Slumber: Interface in SLUMBER power management state.</p>
31:12	Reserved	RSVD 0x0	Reserved

**Table 368: SError Register**  
Offset: Port0: 0x82304 Port1: 0x84304

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> A write of 1 clears the bits in this register. A write of 0 has no effect.			
0	Reserved	RSVD 0x0	Reserved
1	M	RW 0x0	<p>Recovered Communication Error  Communication between the device and host was temporarily lost but was re-established.</p> <p>This can arise from:</p> <ul style="list-style-type: none"> <li>- A device temporarily being removed.</li> <li>- From a temporary loss of PHY synchronization.</li> <li>- From other causes and may be derived from the PhyNRdy signal between the PHY and Link layers.</li> </ul> <p>No action is required by the host software since the operation ultimately succeeded.</p> <p>However, the host software may elect to track such recovered errors to gauge overall communications integrity and potentially step-down the negotiated communication speed.</p>
10:2	Reserved	RSVD 0x0	Reserved
15:11	Reserved	RSVD 0x0	Reserved
16	N	RW 0x0	<p>PhyRdy change  When set to 1, this bit indicates that the PhyRdy changed state since the last time this bit was cleared.</p>



**Table 368: SError Register (Continued)**  
Offset: Port0: 0x82304 Port1: 0x84304

Bit	Field	Type/InitVal	Description
17	Reserved	RSVD 0x0	Reserved
18	W	RW 0x0	Comm Wake When set to 1, this bit indicates that a Comm Wake signal was detected by the PHY since the last time this bit was cleared.
19	B	RW 0x0	10-bit to 8-bit Decode Error When set to 1, this bit indicates that one or more 10-bit to 8-bit decoding errors occurred since the bit was last cleared.
20	D	RW 0x0	Disparity Error When set to one, this bit indicates that incorrect disparity was detected one or more times since the last time the bit was cleared.
21	C	RW 0x0	CRC Error When set to 1, this bit indicates that one or more CRC errors occurred with the Link Layer since the bit was last cleared.
22	H	RW 0x0	Handshake Error When set to one, this bits indicates that one or more R_ERR handshake responses was received in response to frame transmission. Such errors may be the result of: - A CRC error detected by the recipient - A disparity or 10-bit to 8-bit decoding error - Other error conditions leading to a negative handshake on a transmitted frame.
23	S	RW 0x0	Link Sequence Error When set to 1, this bit indicates that one or more Link state machine error conditions was encountered since the last time this bit was cleared. The Link Layer state machine defines the conditions under which the link layer detects an erroneous transition.
24	T	RW 0x0	Transport State Transition Error When set to 1, this bit indicates that an error has occurred in the transition from one state to another within the Transport layer since the last time this bit was cleared.
25	Reserved	RSVD 0x0	Reserved
26	X	RW 0x0	Exchanged When set to 1 this bit indicates that device presence has changed since the last time this bit was cleared. The means by which the implementation determines that the device presence has changed is vendor specific. This bit may be set anytime a PHY reset initialization sequence occurs as determined by reception of the COMINIT signal whether in response to: a new device being inserted, a COMRESET having been issued, or power-up.

**Table 368: SError Register (Continued)**  
Offset: Port0: 0x82304 Port1: 0x84304

Bit	Field	Type/InitVal	Description
31:27	Reserved	RSVD 0x0	Reserved

**Table 369: SControl Register**  
Offset: Port0: 0x82308 Port1: 0x84308

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> See the Serial-ATA specification for a detailed description.			
3:0	DET	RW 0x4	<DET> field controls the host adapter device detection and interface initialization. All other values are reserved. 0 = No request: No device detection or initialization action requested. 1 = Interface comm: Perform interface communication initialization sequence to establish communication. 2 = Reserved 3 = Reserved 4 = Disable PHY: Disable the Serial-ATA interface and put the PHY in offline mode.
7:4	SPD	RW 0x0	<SPD> field represents the highest allowed communication speed the interface is able to negotiate. All other values are reserved. 0 = No restrictions: No speed negotiation restrictions. 1 = Less than Gen1 rate: Limit speed negotiation to a rate not greater than Generation 1 communication rate. 2 = Less than Gen2 rate: Limit speed negotiation to a rate not greater than Generation 2 communication rate.
11:8	IPM	RW 0x0	<IPM> field represents the enabled interface power management states that can be invoked via the Serial-ATA interface power management capabilities. All other values are reserved. 0 = No restrictions: No interface power management state restrictions. 1 = Partial: Transition to the PARTIAL power management state disabled. 2 = Slumber: Transition to the SLUMBER power management state disabled. 3 = Partial/Slumber: Transition to both the PARTIAL and SLUMBER power management states disabled.
15:12	SPM	RW 0x0	<SPM> field is used to select a power management state. A value written to this field is treated as a one-shot. This field will be read as 0000. All other values are reserved. This field is read only 0000. 0 = No request: No power management state transition requested. 1 = Partial: Transition to the PARTIAL power management state initiated. 2 = Slumber: Transition to the SLUMBER power management state initiated. 3 = Active: Transition to the active power management state initiated.
31:16	Reserved	RSVD 0x0	Reserved



**Table 370: LTMode Register**

Offset: Port0: 0x8230C Port1: 0x8430C

Bit	Field	Type/InitVal	Description
5:0	RcvWaterMark	RW 0x30	WaterMark Receiving flow control settings (values in Dwords). When the available entry of the Rx FIFO is less than this value, Serial-ATA flow control is performed by sending HOLD primitives.
6	Reserved	RSVD 0x0	Reserved
7	NearEndLBEen	RW 0x1	Near-End Loopback Enable. 0 = Disable: Near-end loopback (BIST) is disabled. 1 = Enable: Near-end loopback (BIST) is enabled.
8	Reserved	RW 0x0	Reserved. <b>NOTE:</b> Perform a read-modify-write access to this field to avoid the contents being changed.
13:9	Reserved	RW 0x0	Reserved. <b>NOTE:</b> Perform a read-modify-write access to this field to avoid the contents being changed.
14	Reserved	RW 0x0	Reserved. <b>NOTE:</b> Perform a read-modify-write access to this field to avoid the contents being changed.
15	Reserved	RW 0x0	Reserved. <b>NOTE:</b> Perform a read-modify-write access to this field to avoid the contents being changed.
16	Reserved	RW 0x1	Reserved. <b>NOTE:</b> Perform a read-modify-write access to this field to avoid the contents being changed.
18:17	Reserved	RW 0x1	Reserved. <b>NOTE:</b> Perform a read-modify-write access to this field to avoid the contents being changed.
19	Reserved	RW 0x0	Reserved. <b>NOTE:</b> Perform a read-modify-write access to this field to avoid the contents being changed.
20	Reserved	RW 0x0	Reserved. <b>NOTE:</b> Perform a read-modify-write access to this field to avoid the contents being changed.
23:21	Reserved	RW 0x0	Reserved

**Table 370: LTMode Register (Continued)**  
Offset: Port0: 0x8230C Port1: 0x8430C

Bit	Field	Type/InitVal	Description
24	Reserved	RW 0x1	Reserved. <b>NOTE:</b> Perform a read-modify-write access to this field to avoid the contents being changed.
31:25	Reserved	RW 0x0	Reserved

**Table 371: PHY Mode 3 Register**  
Offset: Port0: 0x82310 Port1: 0x84310

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> This register configures the PHY's operation. It is initialized upon a power-on reset. It can be read and written by the CPU.			
0	TX_OFFSET_READ Y	RW 0x0	When set, it asserts a pulse to load INIT_TXOFFS value into the TX clock frequency loop for adjustment. This pulse is self-cleared.
1	Reserved	RW 0x1	Reserved
3:2	AVG_WINDOW	RW 0x2	TX Frequency Average Window Selection (30 KHz sampling per window) 0 = 1 1 = 8 2 = 32 3 = 128
13:4	INIT_TXOFFS	RW 0x0	Initial Transmitter Frequency Offset [9:0] Bit 9 of [9:0] is the sign bit 0 = 1/16384 1 = 1/8192 2 = 1/4096 3 = 1/2048 4 = 1/1024 5 = 1/512 6 = 1/256 7 = 1/128 8 = 2/128
14	FRC_TXOFFS	RW 0x0	Transmitter Frequency Offset Initialization Offset value is defined in INIT_TXOFFS 0 = Disable 1 = Enable
15	AUTO_TX_OFFSET	RW 0x0	Auto RX Clock Recovery Tracking Enable When set, the TX clock's frequency tracks the RX clock (FRC_TXOFFS = 1), and the INIT_TXOFFS[9:0] is ignored. The offset value is directly from the RX peak detector and averaging circuit.



**Table 371: PHY Mode 3 Register (Continued)**  
Offset: Port0: 0x82310 Port1: 0x84310

Bit	Field	Type/InitVal	Description
18:16	SEL_DSPREAD	RW 0x2	Delta Error Selection 0 = 1000 ppm 1 = 2000 ppm 2 = 3000 ppm 3 = 4000 ppm 4 = 5000 ppm 5 = 6000 ppm
19	SSC_DSPREAD	RW 0x1	Down-Spread SSC Selection 0 = Mid-Spread 1 = Down-Spread
20	SSC_EN_MASK_XOR	RW 0x0	Mask of Spread Spectrum Clocking signal Reading this bit returns the XOR result of SSC_EN_MASK and SSC_EN value, that is the same value that drives the PHY. 0 = SSC is off 1 = SSC is on
22:21	SEL_MUCNT_LEN	RW 0x1	Training Length Selection 0 = 520 us 1 = 260 us 2 = 130 us 3 = 65 us
24:23	SELMUFF	RW 0x2	Select Frequency Multiplier after counter switches 0 = Divided by 128 1 = Divided by 256 2 = Divided by 512 3 = Divided by 1024
26:25	SELMUFI	RW 0x2	Select Multiplier Frequency before counter switches It is also used when counter is disabled. 0 = Divided by 128 1 = Divided by 256 2 = Divided by 512 3 = Divided by 1024
28:27	SELMUPF	RW 0x2	Select Multiplier Phase after counter switches 0 = Divided by 1 1 = Divided by 2 2 = Divided by 4 3 = Divided by 8
30:29	SELMUPI	RW 0x2	Select Multiplier Phase before counter switches. It is also used when counter is disabled. 0 = Divided by 1 1 = Divided by 2 2 = Divided by 4 3 = Divided by 8

**Table 371: PHY Mode 3 Register (Continued)**  
Offset: Port0: 0x82310 Port1: 0x84310

Bit	Field	Type/InitVal	Description
31	MUCNT_EN	RW 0x1	Enable MU Counter 0 = Disable 1 = Enable

**Table 372: PHY Mode 4 Register**  
Offset: Port0: 0x82314 Port1: 0x84314

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> This register configures the PHY's operation. It is initialized upon a power-on reset. It can be read and written by the CPU.			
0	SATU_OD8	RW 0x1	Second Order Loop Saturation Method 0 = Saturate to 2.4% 1 = Saturate to 0.8%
1	RXSAT_DIS	RW 0x0	Receiver CDR Frequency Offset Method 0 = Saturation: 2.4% or 0.8% 1 = Wrap-Around
2	FLOOP_EN	RW 0x1	RX CDR Frequency Loop Enable 0 = Disable 1 = Enable
10:3	INIT_RXOFFS	RW 0x0	Initial Receiver Frequency Offset Bit [9] of [7:0] is the sign bit. 0 = 1/4096 1 = 1/2048 2 = 1/1024 3 = 1/512 4 = 1/256 5 = 1/128 6 = 2/128
11	RFRC_RXOFFS	RW 0x0	Receiver Frequency Offset Initialization Enable 0 = Disable 1 = Enable
14:12	Reserved	RW 0x4	Reserved Must write 0x4.
15	Reserved	RW 0x0	Reserved Must write 0x0.
16	Reserved	RW 0x1	Must write 0x1. Read value is reserved.
17	Reserved	RW 0x0	Reserved



**Table 372: PHY Mode 4 Register (Continued)**  
Offset: Port0: 0x82314 Port1: 0x84314

Bit	Field	Type/InitVal	Description
20:18	HOTPLUG_TIMER	RW 0x6	Hot-Plug Detection Timer 0 = 2 ms 1 = 4 ms 2 = 10 ms 3 = 16 ms 4 = 32 ms 5 = 64 ms 6 = 128 ms 7 = 256 ms
21	RXCLK_SEL	RW 0x0	RXCLK_SEL Clock Phase Select 0 = Non-inverting: Non-inverting of RXCLK 1 = Inverting: Inverting of RXCLK
22	TXCLK_SEL	RW 0x0	TXCLK_SEL Clock Phase Select 0 = Non-inverting: Non-inverting of TXCLK 1 = Inverting: Inverting of TXCLK
23	CLK_MONITOR_EN	RW 0x0	Enable monitoring of clocks on test pins. 0 = Shut off: Shut off clock outputs to test pins 1 = Enable: Enable clock outputs to test pins
24	SQUELCH_FLOOP_ON	RW 0x0	When set, the Squelch signal only freezes the frequency loop. The phase loop will continue to track the input data. This bit is for testing RX sensitivity.
25	PortSelector	RW 0x0	Port Selector
26	DISABLE_MISMATCH	RW 0x0	Disable Mismatch
27	SPEED_CHANGE_SEND_IDLE	RW 0x0	Speed change send Idle.
28	FREEZE_AFTER_LOCK	RW 0x0	Freeze after lock.
29	DISSwap	RW 0x0	Hot Swap Detection 0 = On 1 = Off: When hot swapping, no COMRESET/COMINIT will be sent.
30	PARTIAL_TRAINING	RW 0x0	Training mode when resuming from power states  0 = No training sequence: (normal) 1 = Perform training sequence: Perform training sequence during wakeup from a PARTIAL power save mode.

**Table 372: PHY Mode 4 Register (Continued)**  
Offset: Port0: 0x82314 Port1: 0x84314

Bit	Field	Type/InitVal	Description
31	OOB_Bypass	RW 0x0	Out of Band Bypass PHY Ready method selection 0 = Normal operation 1 = Bypass OOB handshake: Bypass OOB handshake, and force PhyRdy.

**Table 373: PHY Mode 1 Register**  
Offset: Port0: 0x8232C Port1: 0x8432C

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> This register must be fully written on every write access to any of its fields.			
0	Reserved	RW 0x0	Reserved
2:1	VTHVCOCAL	RW 0x2	Calibration Threshold of VCO 0 = 1.1V 1 = 1.15V 2 = 1.20V 3 = 1.25V
5:3	EXTKVCO	RW 0x3	Programmable KVCO values When reading these bits, it returns the calibrated KVCO result, and when EXTKVCO_EN = 1, these bits are writable. 000: Smallest Cap (Cap= 0), Fastest VCO, Lowest VDDVCO ..... 111: Largest Cap (Cap = 7), Slowest VCO, Highest VDDVCO
6	EXTKVCO_EN	RW 0x0	External KVCO Value Selection 0 = Calibration result: Use Calibration result 1 = EXTKVCO: Use values from EXTKVCO (PHYMODE1[5:3])
10:7	ICP	RW 0x7	PLL Charge Pump Current Control to adjust loop bandwidth(5uA per step) 0h: 5uA (1x as lowest bandwidth) 1h: 10uA 5h: 30uA Fh: 80uA (16x as highest bandwidth)
11	Reserved	RW 0x0	Must write 0x0. Read value is reserved.
15:12	Reserved	RSVD 0x5	Reserved
17:16	RxVCom	RW 0x1	Receiver Common-mode voltage 0 = 0.20V 1 = 0.25V 2 = 0.30V 3 = 0.35V



**Table 373: PHY Mode 1 Register (Continued)**  
Offset: Port0: 0x8232C Port1: 0x8432C

Bit	Field	Type/InitVal	Description
19:18	RxVDD	RW 0x1	Receive VDD 0 = VDD 1 = 1.2V 2 = 1.3V 3 = 1.4V
21:20	TxVDD	RW 0x1	Transmit VDD 0 = VDD 1 = 1.2V 2 = 1.3V 3 = 1.4V
23:22	Reserved	RSVD 0x1	Reserved
26:24	PREDRV_VDD	RW 0x0	Reserved. Must write 0x0
28:27	Reserved	RW 0x0	Reserved
31:29	Reserved	RW 0x2	Reserved

**Table 374: PHY Mode 2 Register**  
Offset: Port0: 0x82330 Port1: 0x84330

Bit	Field	Type/InitVal	Description
This register must be fully written on every write access to any of its fields.			
0	FORCE_PU_TX	RW 0x1	Transmitter Power Up 0 = Power-down: Transmitter power state is updated by Partial, Slumber, and Active modes. 1 = Power-up: Transmitter is powered up regardless of the power management state.
1	FORCE_PU_RX	RW 0x1	Receiver Power Up 0 = Power-down: Receiver power state is updated by Partial, Slumber, and Active modes. 1 = Power-up: Receiver is powered up regardless of the power management state.
2	PU_PLL	RW 0x1	PLL Power Up 0 = Power-down: PLL is in Power-down mode. 1 = Power-up: PLL is powered up.
3	PU_IVREF	RW 0x1	IVREF Power Up 0 = Power-down: IVREF is in Power-down mode. 1 = Power-up: IVREF is powered up.

**Table 374: PHY Mode 2 Register (Continued)**  
Offset: Port0: 0x82330 Port1: 0x84330

Bit	Field	Type/InitVal	Description
4	PD_TX_AUTO	RW 0x1	Tx Automatic Power Down Enable 0 = Disable: Disable automatic power down of the transmitter in either Partial or Slumber mode 1 = Enable: Enable Automatic power down of the transmitter
5	Reserved	RSVD 0x0	Reserved
6	TM_CLK_STAT	RO 0x0	TM Clock Status (applicable when R5402[10] -- TM_CLKDWN_EN = 1) 0 = Active: TM Clock is active 1 = Gated off: TM Clock is gated off
10:7	Reserved	RSVD 0x8	Reserved
11	LOOPBACK	RW 0x0	Local analog Loopback Enable 0 = Normal Mode: Analog PHY is in normal mode 1 = Loopback Mode: Analog PHY is in Loopback mode
19:12	Reserved	RSVD 0x2C	Reserved
23:20	TXIMP	RW 0x6	Transmitter Impedance TC When reading these bits, they returns calibrated TXIMP values. When EXTRIMP_EN = 1, firmware can write to set the TX Impedance value. 0h: 90.9 ohm ..... 9h: 50.0 ohm ..... Fh: 38.5 ohm
24	EXTIMP_EN	RW 0x0	Impedance Value Selection 0 = Calibration: Result from calibration is used. 1 = EXTRIMP: EXTRIMP is used.
25	ND	RW 0x1	Not Defined
29:26	EXTRXIMP	RW 0x6	RX Impedance by 1000/(11+RXIMP) When reading these bits, they return calibrated RXIMP values When EXTRIMP_EN = 1, FW can write to set the RX Impedance value. 0h: 90.9 ohm (1000/[11 + 0] = 90.9) ..... 9h: 50.0 ohm ..... Fh: 38.5 ohm
30	PLL_CAL_EN	RW 0x0	When changed from 0 to 1, force a PLL calibration
31	IMP_CAL_EN	RW 0x0	When changed from 0 to 1, force an impedance calibration

**Table 375: BIST Control Register**

Offset: Port0: 0x82334 Port1: 0x84334

Bit	Field	Type/InitVal	Description
7:0	BISTPattern	RW 0x0	BIST Pattern Test pattern, refer to bits [15:8] of the first Dword of the BIST Activate FIS.
8	BISTMode	RW 0x0	BIST mode Test direction. 0 = Receiver: BIST Activate FIS Receiver mode. 1 = Transmitter: BIST Activate FIS Transmitter mode.
9	BISTEn	RW 0x0	BIST Test enable On the assertion of the signal, BIST mode starts. 0 = Disable 1 = Enable
10	BISTResult	RO 0x0	BIST Test Pass 0 = Passed 1 = Failed
15:11	Reserved	RSVD 0x0	Reserved <b>NOTE:</b> Perform a read-modify-write access to this field to avoid the contents being changed.
31:16	Reserved	RSVD 0x80	Reserved

**Table 376: BIST-Dword1 Register**

Offset: Port0: 0x82338 Port1: 0x84338

Bit	Field	Type/InitVal	Description
31:0	BistDw1	RW 0x0	In BIST mode: (Bit <BISTEn> is set to 1 in BIST Control Register). This field is the second Dword of the BIST Activate FIS.  In PHY Loopback mode: (Bit <LBEnable> is set to 1 in BIST Control Register). This field is the high Dword [63:32] of the user specified loopback pattern of the BIST Activate FIS.

**Table 377: BIST-Dword2 Register**

Offset: Port0: 0x8233C Port1: 0x8433C

Bit	Field	Type/InitVal	Description
31:0	BistDw2	RW 0x0	<p>In BIST mode: (Bit &lt;BISTEn&gt; is set to 1 in BIST Control Register). This field is the third Dword of the BIST Activate FIS.</p> <p>In PHY Loopback mode: (Bit &lt;LBEnable&gt; is set to 1 in BIST Control Register). This field is the low Dword [31:0] of the user specified loopback pattern of the BIST Activate FIS.</p>

**Table 378: SError Interrupt Mask Register**

Offset: Port0: 0x82340 Port1: 0x84340

Bit	Field	Type/InitVal	Description
31:0	eSErrIntMsk	RW 0x019C0000	<p>SError Interrupt Mask Bits Each of these bits checks the corresponding bit in SError Register, and if these bits are disabled (0), they mask the interrupt. 0 = Mask 1 = No mask</p>

**Table 379: Serial-ATA Interface Control Register**

Offset: Port0: 0x82344 Port1: 0x84344

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> When bit <eEnEDMA> in the EDMA Command Register is set, this register must not be written. If this register is written when bit <eEnEDMA> is set, the write transaction will cause unpredictable behavior.			
3:0	PMportTx	RW 0x0	<p>Port Multiplier Transmit. This field specifies the Port Multiplier (bits [11:8] in DW0 of the FIS header) of any transmitted FIS.</p>
7:4	Reserved	RW 0x0	Reserved
8	VendorUqMd	RW 0x0	<p>Vendor Unique mode This bit is set to 1 to indicate that the next FIS, going to be transmitted, is a Vendor Unique FIS. Only when this bit is set, the host may write to the Vendor Unique Register. When this bit is cleared, bits &lt;VendorUqDn&gt; and &lt;VendorUqErr&gt; of the Serial-ATA Interface Status Register are also cleared. Before setting this bit the Host must verify: - No pending commands are in progress. - The EDMA is disabled, bit &lt;eEnEDMA&gt; in the EDMA Command Register is cleared.</p>



**Table 379: Serial-ATA Interface Control Register (Continued)**  
Offset: Port0: 0x82344 Port1: 0x84344

Bit	Field	Type/InitVal	Description
9	VendorUqSend	RW 0x0	<p>Send vendor Unique FIS. This bit is set to 1 by the host SW to indicate that the next DWORD written to the Vendor Unique Register is the last DWORD in the payload.</p> <p>Bit &lt;VendorUqDn&gt; of the Serial-ATA Interface Status Register is set when the transmission is completed; bit &lt;VendorUqErr&gt; of that same register is also set if the transmission ends with an error.</p> <p>When the host SW writes to the Vendor Unique Register with this bit set to 1, the Serial-ATA transport layer closes the FIS and clears this bit.</p>
15:10	Reserved	RSVD 0x0	Reserved
16	eDMAActivate	RW 0x0	<p>DMA Activate This bit has an effect only if the Serial-ATA port is in Target mode operation (i.e., bit &lt;ComChannel&gt; in the EDMA Configuration Register, is set).</p> <p>When this bit is set the transport layer sends (multiple) data FISs, as long as the DMA is active, to complete the data transaction associated with the command.</p> <p>When the port functions as a target in Target mode operation, this bit is set by the target host software to activate read data transactions from the target to the initiator.</p> <p>When the port functions as an initiator in Target mode operation, this bit is set when a DMA Activate FIS is received to activate the write data transaction from the initiator to the target.</p> <p>This bit is cleared when the DMA completes the data transaction associated with the command.</p> <p>0 = Not sent: Transport layer does not send DMA data FISs.</p> <p>1 = Sent: Transport layer keeps sending (multiple) DMA data FISs until the data transaction associated with the command is completed.</p>
23:17	Reserved	RSVD 0x0	Reserved
24	ClearStatus	SC 0x0	<p>Status Self-Clear This bit clears bits [16], [30], and [31] in the Serial-ATA Interface Status Register.</p> <p>0 = No clear: Does not clear bits.</p> <p>1 = Clear: Clears the bits.</p>
25	SendSftRst	SC 0x0	<p>Self-Negate When this bit is set to 1, the transport layer sends Register - Host to Device control FIS to the device.</p>
31:26	Reserved	RSVD 0x0	Reserved

**Table 380: Serial-ATA Interface Test Control Register**  
Offset: Port0: 0x82348 Port1: 0x84348

Bit	Field	Type/InitVal	Description
0	MBistEn	RW 0x0	Memory BIST Enable Start Memory BIST test in MBIST mode. 0 = Disable: Memory BIST test disabled. 1 = Enable: Memory BIST test enabled.
1	TransFrmSizExt	RW 0x0	Maximum Transmit Frame Size Extended See field <TransFrmSiz> [15:14].
3:2	Reserved	RW 0x0	Reserved
7:4	Reserved	RSVD 0x0	Reserved
8	LEnable	RW 0x0	PHY Loopback Enable This bit enables SATA near-end PHY loopback. 0 = Disable 1 = Enable
12:9	LBPattern	RW 0x0	PHY Loopback pattern
13	LBStartRd	RW 0x0	Loopback Start BIST-DW1 Register is used as User-specified test pattern low DWORD. BIST-DW2 Register is used as User-specified test pattern high DWORD. These registers must be initiated before this bit is set. 0 = Disable 1 = Enable
15:14	TransFrmSiz	RW 0x0	Maximum Transmit Frame Size When <TransFrmSizExt> is 0: 0 = Maximum Transmit Frame Size is 8 KB 1 = Maximum Transmit Frame Size is 512B 2 = Maximum Transmit Frame Size is 64B 3 = Maximum Transmit Frame Size is 128B  When <TransFrmSizExt> is 1: 0 = Maximum Transmit Frame Size is 256B 1 = Maximum Transmit Frame Size is 1 KB 2 = Maximum Transmit Frame Size is 2 KB 3 = Maximum Transmit Frame Size is 4 KB
31:16	PortNumDevErr	RO 0x0	Each bit in this field is set to 1 when Register - Device to Host FIS or Set Device Bits FIS is received with bit <ERR> from the corresponding PM port set to 1. All bits in this field are cleared to 0 when the value of the <eEnEDMA> bit in the EDMA Command Register is changed from 0 to 1.



**Table 381: Serial-ATA Interface Status Register**  
Offset: Port0: 0x8234C Port1: 0x8434C

Bit	Field	Type/InitVal	Description
7:0	FISTypeRx	RO 0x0	FIS Type Received This field specifies the FIS Type of the last received FIS.
11:8	PMportRx	RO 0x0	Port Multiplier Received This field specifies the Port Multiplier (bits [11:8] in DW0 of the FIS header) of the last received FIS. It also specifies the Port Multiplier (bits [11:8] in DW0 of the FIS header) of the next Data - Host to Device transmit FISs.
12	VendorUqDn	RO 0x0	Vendor Unique FIS Transmission Done This bit is set when the Vendor Unique FIS transmission has completed. 0 = Incomplete: Vendor Unique FIS transmission has not completed. 1 = Complete: Vendor Unique FIS transmission has completed.
13	VendorUqErr	RO 0x0	Vendor Unique FIS Transmission Error This bit indicates if the Vendor Unique FIS transmission has completed successfully. This bit is valid when bit <VendorUqDn> of this register is set. 0 = Completed: Vendor Unique FIS transmission has completed successfully. 1 = Error: Vendor Unique FIS transmission has completed with error.
14	MBistRdy	RO 0x1	Memory BIST Ready This bit indicates when the memory BIST test is completed. 0 = Incomplete: Memory BIST test is not completed. 1 = Complete: Memory BIST test is completed.
15	MBistFail	RO 0x0	Memory BIST Fail This bit indicates if the memory BIST test passed. It is valid when bit <MBistRdy> of this register is set. 0 = Pass 1 = Fail
16	AbortCommand	RO 0x0	Abort Command This bit indicates if the transport has aborted a command as a response to collision with incoming FIS.  <b>NOTE:</b> This bit is cleared when the <ClearStatus> bit[24] in the Serial-ATA Interface Control Register is set to 1. 0 = Not aborted: Command was not aborted. 1 = Abort: Command was aborted.
17	LBPass	RO 0x0	Near-end Loopback Pass This bit indicates if the Near-end Loopback test passed. 0 = Fail 1 = Pass

**Table 381: Serial-ATA Interface Status Register (Continued)**  
Offset: Port0: 0x8234C Port1: 0x8434C

Bit	Field	Type/InitVal	Description
18	DMAAct	RO 0x0	DMA Active This bit indicates if the transport DMA FSM is active. 0 = Idle 1 = Active
19	PIOAct	RO 0x0	PIO Active This bit indicates if the transport PIO FSM is active. 0 = Idle 1 = Active
20	RxHdAct	RO 0x0	Rx Header Active This bit indicates if the transport Rx Header FSM is active. 0 = Idle 1 = Active
21	TxHdAct	RO 0x0	Tx Header Active This bit indicates if the transport Tx Header FSM is active. 0 = Idle 1 = Active
22	PlugIn	RO 0x0	Cable plug-in indicator and device presence indication. This signal becomes invalid when the core is in SATA Power Management modes. This indicator is also reflected in the <X> bit in SError Register. 0 = No detection: Device presence not detected. 1 = Detection: Device presence detected.
23	LinkDown	RO 0x1	SATA communication is not ready. Primitives or FISs are not able to be transmitted or received. 0 = Ready: Link is ready. 1 = Not Ready: Link is not ready.
28:24	TransFsmSts	RO 0x0	Transport Layer FSM status 0x00 = Transport layer is idle. 0x00-0x1F = Transport layer is not idle.
29	Reserved	RSVD 0x0	Reserved
30	RxBIST	RO 0x0	Set to 1 when BIST FIS is received <b>NOTE:</b> This bit is cleared when the <ClearStatus> bit[24] in the Serial-ATA Interface Control Register is set to 1.
31	N	RO 0x0	Set to 1 when the Set Devices Bits FIS is received with the Notification (N) bit set to 1. <b>NOTE:</b> This bit is cleared when the <ClearStatus> bit[24] in the Serial-ATA Interface Control Register is set to 1.



**Table 382: Vendor Unique Register**

Offset: Port0: 0x8235C Port1: 0x8435C

Bit	Field	Type/InitVal	Description
31:0	VendorUqDw	RW 0x0	Vendor Unique DWORD The data written to this register is transmitted as a vendor unique FIS. This Data includes the FIS header as well as the payload.

**Table 383: FIS Configuration Register**

Offset: Port0: 0x82360 Port1: 0x84360

Bit	Field	Type/InitVal	Description
7:0	FISWait4RdyEn	RW 0x0	This field identifies whether the transport layer waits for the upper layer (EDMA or host) to acknowledge reception of the current FIS before enabling the link layer to response with R_RDY for the next FIS. When bit <eEnEDMA> is set to 1 in EDMA Command Register, the transport layer ignores the value of bits [4:0] of this field and assume a value of 0x1F, i.e., it waits for the EDMA to acknowledge reception of the current FIS before enabling the link layer to response with R_RDY for the next FIS. The function of each bit in this field is: <FISWait4RdyEn>[0]: Register - Device to Host FIS <FISWait4RdyEn>[1]: SDB FIS is received with <N> bit cleared to 0. <FISWait4RdyEn>[2]: DMA Activate FIS <FISWait4RdyEn>[3]: DMA Setup FIS <FISWait4RdyEn>[4]: Data FIS first DW <FISWait4RdyEn>[5]: Data FIS entire FIS <FISWait4RdyEn>[7:6]: Reserved 0 = No wait: Do not wait for host ready. 1 = Wait: Wait for host ready.
15:8	FISWait4HostRdyEn	RW 0x0	This field identifies whether the transport layer waits for the upper layer (host) to acknowledge reception of the current FIS before enabling the link layer to response with R_RDY for the next FIS. The function of each bit in this field is: <FISWait4HostRdyEn>[0]: Register - Device to Host FIS with <ERR> or <DF> bit set to 1. <FISWait4HostRdyEn>[1]: SDB FIS is received with <N> bit set to 1. <FISWait4HostRdyEn>[2]: SDB FIS is received with <ERR> bit set to 1. <FISWait4HostRdyEn>[3]: BIST activate FIS <FISWait4HostRdyEn>[4]: PIO Setup FIS <FISWait4HostRdyEn>[5]: Data FIS with Link error <FISWait4HostRdyEn>[6]: Unrecognized FIS type <FISWait4HostRdyEn>[7]: Any FIS 0 = No wait: Do not wait for host ready. 1 = Wait: Wait for host ready.
16	FISDMAActiveSync Resp	RW 0x0	This bit identifies whether the transport layer responses with a single SYNC primitive after the DMA activates FIS reception. 0 = Normal: Normal response 1 = SYNC primitive: Response with single SYNC primitive. Must be set to 1 when bit <eEDMAFBS> in EDMA Configuration Register is set to 1.

**Table 383: FIS Configuration Register (Continued)**

Offset: Port0: 0x82360 Port1: 0x84360

Bit	Field	Type/InitVal	Description
17	FISUnrecTypeCont	RW 0x0	<p>When this bit is set, the transport layer state machine ignores incoming FIS with unrecognized FIS type.</p> <p>0 = Error state: When an unrecognized FIS type is received, the transport layer goes into error state and asserts a protocol error.</p> <p>1 = No error state: When an unrecognized FIS type is received, the transport layer does not go into error state and does not assert a protocol error.</p>
31:18	Reserved	RSVD 0x0	Reserved

**Table 384: FIS Interrupt Cause Register**

Offset: Port0: 0x82364 Port1: 0x84364

Bit	Field	Type/InitVal	Description
A corresponding cause bit is set every time that an interrupt occurs. A write of 0 clears the bit. A write of 1 has no effect.			
7:0	FISWait4Rdy	RW0C 0x0	<p>This field indicates the reception of the following FISs.</p> <p>&lt;FISWait4Rdy&gt;[0]: Register Device to Host FIS</p> <p>&lt;FISWait4Rdy&gt;[1]: SDB FIS is received with &lt;N&gt; bit cleared to 0.</p> <p>&lt;FISWait4Rdy&gt;[2]: DMA Activate FIS</p> <p>&lt;FISWait4Rdy&gt;[3]: DMA Setup FIS</p> <p>&lt;FISWait4Rdy&gt;[4]: Data FIS first Dword</p> <p>&lt;FISWait4Rdy&gt;[5]: Data FIS entire FIS</p> <p>&lt;FISWait4Rdy&gt;[7:6]: Reserved</p> <p>For any FIS other than data FIS, the corresponding bit is set when the entire FIS is received from the link layer without an error, that is, FIS Dword0 Register through FIS Dword6 Register are updated with the content of the FIS. If the non-data FIS length is shorter than 7 Dwords, only the relevant registers are updated with the content of the FIS. If the non-data FIS length is longer than 7 Dwords, FIS Dword0 Register through FIS Dword6 Register are updated with the content of the FIS. The rest of FIS is dropped.</p> <p>For data FIS, &lt;FISWait4Rdy&gt; [4] is set when the first Dword of the FIS is received from the link layer and &lt;FISWait4Rdy&gt; [5] is set when the entire FIS is received from the link layer, regardless of whether or not an error occurred. Only FIS Dword0 Register is updated with the content of the FIS. FIS Dword1 Register through FIS Dword6 Register are not updated.</p> <p>When at least one bit in this field is set and the corresponding bit in the &lt;FISWait4RdyEn&gt; field of the FIS Configuration Register is enabled (set to 1), the transport layer prevents assertion of the primitive R_RDY and the reception of the next FIS.</p> <p>0 = No interrupt: No interrupt indication.</p> <p>1 = Interrupt: Corresponding interrupt occurs.</p>



**Table 384: FIS Interrupt Cause Register (Continued)**  
Offset: Port0: 0x82364 Port1: 0x84364

Bit	Field	Type/InitVal	Description
15:8	FISWait4HostRdy	RW0C 0x0	<p>This field indicates the reception of the following FISs.</p> <p>&lt;FISWait4HostRdy&gt;[0]: Register Device to Host FIS with &lt;ERR&gt; bit set to 1.</p> <p>&lt;FISWait4HostRdy&gt;[1]: SDB FIS is received with &lt;N&gt; bit set to 1.</p> <p>&lt;FISWait4HostRdy&gt;[2]: SDB FIS is received with &lt;ERR&gt; bit set to 1.</p> <p>&lt;FISWait4HostRdy&gt;[3]: BIST activates FIS</p> <p>&lt;FISWait4HostRdy&gt;[4]: PIO Setup FIS</p> <p>&lt;FISWait4HostRdy&gt;[5]: Data FIS with Link error</p> <p>&lt;FISWait4HostRdy&gt;[6]: Unrecognized FIS type</p> <p>&lt;FISWait4HostRdy&gt;[7]: Any FIS</p> <p>For any FIS other than data FIS, the corresponding bit is set when the FIS is received from the link layer without an error, that is, FIS Dword0 Register through FIS Dword6 Register are updated with the content of the FIS up to the FIS length.</p> <p>For the data FIS, the corresponding bit is set when the entire FIS is received from the link layer, if a link error occurs. Only the FIS Dword0 Register is updated with the content of the FIS.</p> <p>FIS Dword1 Register through FIS Dword6 Register are not updated.</p> <p>If the non-data FIS length is shorter than 7 Dwords, only the relevant registers are updated with the content of the FIS.</p> <p>If the non-data FIS length is longer than 7 Dwords, FIS Dword0 Register through FIS Dword6 Register are updated with the content of the FIS. The rest of FIS is dropped.</p> <p>When at least one bit in this field is set and the corresponding bit in the &lt;FISWait4HostRdyEn&gt; field of the FIS Configuration Register is enabled (set to 1), the transport layer prevents assertion of the primitive R_RDY and the reception of the next FIS.</p> <p>0 = No interrupt: No interrupt indication.</p> <p>1 = Interrupt: Corresponding interrupt occurs.</p>
23:16	Reserved	RW0C 0x0	Reserved.
24	FISTxDone	RW0C 0x0	<p>This bit is set to 1 when the FIS transmission is done, either aborted or completed with R_OK or R_ERR.</p> <p>0 = Continues: Frame transmission continues.</p> <p>1 = Completed: Frame transmission completed, either with R_ERR or R_OK, or frame transmission aborted.</p>
25	FISTxErr	RW0C 0x0	<p>This bit is valid when bit[24] &lt;FISTxDone&gt; is set to 1.</p> <p>This bit is set to 1 when the FIS transmission is done, bit[24] &lt;FISTxDone&gt; is set to 1 and one of the following occurs:</p> <p>FIS transmission is aborted due to collision with the received FIS.</p> <p>FIS transmission is completed with R_ERR.</p> <p>0 = Complete: Frame transmission was completed successful with R_OK.</p> <p>1 = Incomplete: Frame transmission was not completed successful.</p>
31:26	Reserved	RO 0x0	Reserved

**Table 385: FIS Interrupt Mask Register**  
Offset: Port0: 0x82368 Port1: 0x84368

Bit	Field	Type/InitVal	Description
25:0	FISIntMask	RW 0x0000A00	FIS Interrupt Error Mask Bits Each of these bits mask the corresponding bit in the FIS Interrupt Cause Register. If a bit in the FIS Interrupt Cause Register is set and the corresponding bit in this register is set to 1, bit <eTransInt> in EDMA Interrupt Error Cause Register is also set to 1. 0 = Mask 1 = Do not mask
31:26	Reserved	RO 0x0	Reserved

**Table 386: FIS Dword0 Register**  
Offset: Port0: 0x82370 Port1: 0x84370

Bit	Field	Type/InitVal	Description
31:0	RxFISDW0	RO 0x0	This field contains Dword 0 of the incoming data or non-data FIS.

**Table 387: FIS Dword1 Register**  
Offset: Port0: 0x82374 Port1: 0x84374

Bit	Field	Type/InitVal	Description
31:0	RxFISDW1	RO 0x0	This field contains Dword 1 of the incoming non-data FIS.

**Table 388: FIS Dword2 Register**  
Offset: Port0: 0x82378 Port1: 0x84378

Bit	Field	Type/InitVal	Description
31:0	RxFISDW2	RO 0x0	This field contains Dword 2 of the incoming non-data FIS.

**Table 389: FIS Dword3 Register**  
Offset: Port0: 0x8237C Port1: 0x8437C

Bit	Field	Type/InitVal	Description
31:0	RxFISDW3	RO 0x0	This field contains Dword 3 of the incoming non-data FIS.



**Table 390: FIS Dword4 Register**

Offset: Port0: 0x82380 Port1: 0x84380

Bit	Field	Type/InitVal	Description
31:0	RxFISDW4	RO 0x0	This field contains Dword 4 of the incoming non-data FIS.

**Table 391: FIS Dword5 Register**

Offset: Port0: 0x82384 Port1: 0x84384

Bit	Field	Type/InitVal	Description
31:0	RxFISDW5	RO 0x0	This field contains Dword 5 of the incoming non-data FIS.

**Table 392: FIS Dword6 Register**

Offset: Port0: 0x82388 Port1: 0x84388

Bit	Field	Type/InitVal	Description
31:0	RxFISDW6	RO 0x0	This field contains Dword 6 of the incoming non-data FIS.

**Table 393: PHYMODE9\_GEN2 Register**

Offset: Port0: 0x82398 Port1: 0x84398

Bit	Field	Type/InitVal	Description
3:0	TXAMP[3:0]	RW 0x5	TX Driver Amplitude This setting, along with TXAMP[4], defines the final output amplitude. 0x00 = Minimum amplitude 0x15 = Default value 0x1F = Maximum amplitude
7:4	TX_PRE_EMPH	RW 0x3	Pre-Emphasis Level Control 0x0 = Minimum amplitude 0x3 = Default value 0x7 = Maximum amplitude Above 0x7 = Reserved
8	Reserved	RW 0x0	Reserved
9	Reserved	RW 0x0	Reserved Write only 0x0 to this field.
10	Reserved	RW 0x0	Reserved Must be 0x0.

**Table 393: PHYMODE9\_GEN2 Register (Continued)**  
Offset: Port0: 0x82398 Port1: 0x84398

Bit	Field	Type/InitVal	Description
13:11	Reserved	RW 0x0	Reserved Must be 0x0.
14	TXAMP[4]	RW 0x1	Part of TX Driver Output Amplitude (TXAMP[4]). See more details in the TXAMP[3:0] field.
15	Reserved	RW 0x0	Reserved. Write only 0x0 to this field.
31:16	Reserved	RO 0x0	Reserved

**Table 394: PHYMODE9\_GEN1 Register**  
Offset: Port0: 0x8239C Port1: 0x8439C

Bit	Field	Type/InitVal	Description
3:0	TXAMP[3:0]	RW 0x5	TX Driver Amplitude This setting, along with TXAMP[4], defines the final output amplitude. 0x00 = Minimum amplitude 0x15 = Default value 0x1F = Maximum amplitude
7:4	TX_PRE_EMPH	RW 0x3	Pre-Emphasis Level Control 0x0 = Minimum amplitude 0x3 = Default value 0x7 = Maximum amplitude Above 0x7 = Reserved
8	Reserved	RW 0x0	Reserved
9	Reserved	RW 0x0	Reserved Write only 0x0 to this field.
10	Reserved	RW 0x0	Reserved Must be 0x0.
13:11	Reserved	RW 0x0	Reserved Must be 0x0.
14	TXAMP[4]	RW 0x1	Part of "TX Driver output amplitude" (TXAMP[4]). See more details in MP[3:0] field.



**Table 394: PHYMODE9\_GEN1 Register (Continued)**  
Offset: Port0: 0x8239C Port1: 0x8439C

Bit	Field	Type/InitVal	Description
15	Reserved	RW 0x0	Reserved Write only 0x0 to this field.
31:16	Reserved	RO 0x0	Reserved

**Table 395: PHY Configuration Register**  
Offset: Port0: 0x823A0 Port1: 0x843A0

Bit	Field	Type/InitVal	Description
3:0	SQ_THRESHOLD	RW 0x3	Squelch Detector Threshold 0 = 50 mVp-p 1 = 70 mVp-p 2 = 90 mVp-p 3 = 110 mVp-p 4 = 130 mVp-p 5 = 150 mVp-p 6 = 170 mVp-p 7 = 190 mVp-p 8 = 50 mVp-p 9 = 80 mVp-p 10 = 110 mVp-p 11 = 140 mVp-p 12 = 170 mVp-p 13 = 200 mVp-p 14 = 230 mVp-p 15 = 260 mVp-p
8:4	VTH_DISCON	RW 0x7	Host disconnect's voltage level threshold (25 mVp-p step) 0: 25 mVp-p 1: 50 mVp-p 2: 75 mVp-p 3: 100 mVp-p 4: 125 mVp-p 5: 150 mVp-p 6: 175 mVp-p 7: 200 mVp-p 8: 225 mVp-p 9: 250 mVp-p 10: 275 mVp-p 11: 300 mVp-p 12: 325 mVp-p 13: 350 mVp-p 14: 375 mVp-p 15: 400 mVp-p ..... 31: 800mVp-p
15:9	Reserved	RW 0x1	Reserved

**Table 395: PHY Configuration Register (Continued)**  
Offset: Port0: 0x823A0 Port1: 0x843A0

Bit	Field	Type/InitVal	Description
31:16	Reserved	RSVD 0x0	Reserved

**Table 396: PHYTCTL Register**  
Offset: Port0: 0x823A4 Port1: 0x843A4

Bit	Field	Type/InitVal	Description
PHY-LINK Test Control			
5:0	TEST_ANA	RW 0x10	Analog Test Port Select
11:6	ND	RW 0x0	Not Defined
15:12	LINK_TESTSEL	RW 0x0	Link Test Port Select (MONITORSEL)
31:16	Reserved	RSVD 0x102	Reserved

**Table 397: PHY Mode 10 Register**  
Offset: Port0: 0x823A8 Port1: 0x843A8

Bit	Field	Type/InitVal	Description
TX Clock Frequency Offset Manual Update			
9:0	AVG	RO 0x0	Tx Frequency Offset
10	Reserved	RSVD 0x0	Reserved
11	AVG_READY	RO 0x0	When set, the AVG field is valid
12	AVG_READ_EN	RW 0x0	When toggled, it triggers the TX Frequency Offset calculation. This can only be triggered once after power-up. To retrigger, it requires a PHY RESET.
13	Reserved	RW 0x0	Reserved
15:14	ND	RW 0x0	Not Defined
25:16	NXT_AVG	RO 0x0	Tx Frequency Offset
26	Reserved	RO 0x0	Reserved



**Table 397: PHY Mode 10 Register (Continued)**  
Offset: Port0: 0x823A8 Port1: 0x843A8

Bit	Field	Type/InitVal	Description
27	PEAK_READY	RO 0x0	When set, the NXT_AVG value is valid (this signal will toggle per update window as defined in AVG_WINDOW[1:0])
31:28	ND	RW 0x0	Not Defined

**Table 398: PHY Mode 12 Register**  
Offset: Port0: 0x823B0 Port1: 0x843B0

Bit	Field	Type/InitVal	Description
31:0	Reserved	RW 0x0	Reserved

## A.7.5 Shadow Register Block Map

**Table 399: Shadow Register Block Registers Map**

<b>Bits 31-24</b>	<b>Bits 23-16</b>	<b>Bits 15-8</b>	<b>Bits 7-0</b>	<b>Offset</b>	<b>Comment</b>
Reserved	Reserved		Device PIO Data register	Port 0: 0xA2100 Port 1: 0xA4100	see ATA/ATAPI specification
Reserved	Reserved	Reserved	Device - Features (Features Current)/Error register	Port 0: 0xA2104 Port 1: 0xA4104	
Reserved	Reserved	Reserved	Device - Sector Count (Sector Count Current) register	Port 0: 0xA2108 Port 1: 0xA4108	
Reserved	Reserved	Reserved	Device - LBA Low register	Port 0: 0xA210C Port 1: 0xA410C	
Reserved	Reserved	Reserved	Device - LBA Mid register	Port 0: 0xA2110 Port 1: 0xA4110	
Reserved	Reserved	Reserved	Device - LBA High register	Port 0: 0xA2114 Port 1: 0xA4114	
Reserved	Reserved	Reserved	Device - Device/Head (Device) register	Port 0: 0xA2118 Port 1: 0xA4118	
Reserved	Reserved	Reserved	Device - Command/Status register	Port 0: 0xA211C Port 1: 0xA411C	
Reserved	Reserved	Reserved	Device - Control/ Alternate Status register	Port 0: 0xA2120 Port 1: 0xA4120	

## A.8

## Gigabit Ethernet Controller Registers

**NOTE:** If the Power Management Control register offset 0x2011c bits [2:1] for port0,1 is set to 0 (deactivated), the specific ports registers cannot be accessed. An attempt to read from a deactivated ports registers causes a system hang.

The following table provides a summarized list of all of the Gigabit Ethernet Controller registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 400: Register Map Table for the Gigabit Ethernet Controller Registers**

Register Name	Offset	Table and Page
<b><i>Gigabit Ethernet Unit Global Registers</i></b>		
PHY Address Register	Port0: 0x72000, Port1: 0x76000	<a href="#">Table 401, p. 554</a>
SMI Register	Port0: 0x72004, Port1: 0x76004	<a href="#">Table 402, p. 554</a>
Ethernet Unit Default Address (EUDA) Register	Port0: 0x72008, Port1: 0x76008	<a href="#">Table 403, p. 555</a>
Ethernet Unit Default ID (EUDID) Register	Port0: 0x7200C, Port1: 0x7600C	<a href="#">Table 404, p. 555</a>
Ethernet Unit Interrupt Cause (EUIC) Register	Port0: 0x72080, Port1: 0x76080	<a href="#">Table 405, p. 556</a>
Ethernet Unit Interrupt Mask (EUIM) Register	Port0: 0x72084, Port1: 0x76084	<a href="#">Table 406, p. 557</a>
Ethernet Unit Error Address (EUEA) Register	Port0: 0x72094, Port1: 0x76094	<a href="#">Table 407, p. 557</a>
Ethernet Unit Internal Address Error (EUIAE) Register	Port0: 0x72098, Port1: 0x76098	<a href="#">Table 408, p. 557</a>
Ethernet Unit Control (EUC) Register	Port0: 0x720B0, Port1: 0x760B0	<a href="#">Table 409, p. 557</a>
Base Address Register (n=0–5)	Port0: BA0: 0x72200, BA1: 0x72208, BA2: 0x72210, BA3: 0x72218, BA4: 0x72220, BA5: 0x72228 Port1: BA0: 0x76200, BA1: 0x76208, BA2: 0x76210, BA3: 0x76218, BA4: 0x76220, BA5: 0x76228	<a href="#">Table 410, p. 558</a>
Size (S) Register (n=0–5)	Port0: SR0: 0x72204, SR1: 0x7220C, SR2: 0x72214, SR3: 0x7221C, SR4: 0x72224, SR5: 0x7222C Port1: SR0: 0x76204, SR1: 0x7620C, SR2: 0x76214, SR3: 0x7621C, SR4: 0x76224, SR5: 0x7622C	<a href="#">Table 411, p. 559</a>
High Address Remap (HA)1 Register (n=0–3)	Port0: HARR0: 0x72280, HARR1: 0x72284, HARR2: 0x72288, HARR3: 0x7228C Port1: HARR0: 0x76280, HARR1: 0x76284, HARR2: 0x76288, HARR3: 0x7628C	<a href="#">Table 412, p. 559</a>
Base Address Enable (BARE) Register	Port0: 0x72290, Port1: 0x76290	<a href="#">Table 413, p. 559</a>

**Table 400: Register Map Table for the Gigabit Ethernet Controller Registers (Continued)**

Register Name	Offset	Table and Page
Ethernet Port Access Protect (EPAP) Register	Port0: 0x72294, Port1: 0x76294	Table 414, p. 560
Port0/1 Mbus Top Arbiter Register	Port0: 0xE20C0, Port1: 0xE60C0	Table 415, p. 560
<b>Port Control Registers</b>		
Port Configuration (PxC) Register	Port0: 0x72400, Port1: 0x76400	Table 416, p. 561
Port Configuration Extend (PxCX) Register	Port0: 0x72404, Port1: 0x76404	Table 417, p. 562
MII Serial Parameters Register	Port0: 0x72408, Port1: 0x76408	Table 418, p. 562
VLAN EtherType (EVLANE) Register	Port0: 0x72410, Port1: 0x76410	Table 419, p. 563
MAC Address Low (MACAL) Register	Port0: 0x72414, Port1: 0x76414	Table 420, p. 563
MAC Address High (MACAH) Register	Port0: 0x72418, Port1: 0x76418	Table 421, p. 563
SDMA Configuration (SDC) Register	Port0: 0x7241C, Port1: 0x7641C	Table 422, p. 564
IP Differentiated Services CodePoint 0 to Priority (DSCP0) Register	Port0: 0x72420, Port1: 0x76420	Table 423, p. 565
IP Differentiated Services CodePoint 1 to Priority (DSCP1) Register	Port0: 0x72424, Port1: 0x76424	Table 424, p. 565
IP Differentiated Services CodePoint 2 to Priority (DSCP2/3/4/5) Register (n=0–3)	Port0: DSCP0: 0x72428, DSCP1: 0x7242C, DSCP2: 0x72430, DSCP3: 0x72434 Port1: DSCP0: 0x76428, DSCP1: 0x7642C, DSCP2: 0x76430, DSCP3: 0x76434	Table 425, p. 566
IP Differentiated Services CodePoint 6 to Priority (DSCP6) Register	Port0: 0x72438, Port1: 0x76438	Table 426, p. 566
Port Serial Control0 (PSC0) Register	Port0: 0x7243C, Port1: 0x7643C	Table 427, p. 566
VLAN Priority Tag to Priority (VPT2P) Register	Port0: 0x72440, Port1: 0x76440	Table 428, p. 569
Ethernet Port Status 0 (PS0) Register	Port0: 0x72444, Port1: 0x76444	Table 429, p. 569
Port Serial Control1 (PSC1) Register	Port0: 0x7244C, Port1: 0x7644C	Table 430, p. 571
Ethernet Port Status1 (PS1) Register	Port0: 0x72450, Port1: 0x76450	Table 431, p. 573
Marvell Header Register	Port0: 0x72454, Port1: 0x76454	Table 432, p. 575
Port Interrupt Cause (IC) Register	Port0: 0x72460, Port1: 0x76460	Table 433, p. 577
Port Interrupt Cause Extend (ICE) Register	Port0: 0x72464, Port1: 0x76464	Table 434, p. 580
Port Interrupt Mask (PIM) Register	Port0: 0x72468, Port1: 0x76468	Table 435, p. 582



Table 400: Register Map Table for the Gigabit Ethernet Controller Registers (Continued)

Register Name	Offset	Table and Page
Port Extend Interrupt Mask (PEIM) Register	Port0: 0x7246C, Port1: 0x7646C	<a href="#">Table 436, p. 582</a>
Port Tx FIFO Urgent Threshold (PxTFUT) Register	Port0: 0x72474, Port1: 0x76474	<a href="#">Table 437, p. 582</a>
Port Rx Minimal Frame Size (PxMFS) Register	Port0: 0x7247C, Port1: 0x7647C	<a href="#">Table 438, p. 583</a>
Port Rx Discard Frame Counter (PxDFC) Register	Port0: 0x72484, Port1: 0x76484	<a href="#">Table 439, p. 583</a>
Port Overrun Frame Counter (PxOFC) Register	Port0: 0x72488, Port1: 0x76488	<a href="#">Table 440, p. 583</a>
Port Internal Address Error (EUIAE) Register	Port0: 0x72494, Port1: 0x76494	<a href="#">Table 441, p. 584</a>
Ethernet Type Priority Register	Port0: 0x724BC, Port1: 0x764BC	<a href="#">Table 442, p. 584</a>
Ethernet Current Receive Descriptor Pointers (CRDP) Register (n=0–7)	Port0: Q0: 0x7260C, Q1: 0x7261C, Q2: 0x7262C, Q3: 0x7263C, Q4: 0x7264C, Q5: 0x7265C, Q6: 0x7266C, Q7: 0x7267C Port1: Q0: 0x7660C, Q1: 0x7661C, Q2: 0x7662C, Q3: 0x7663C, Q4: 0x7664C, Q5: 0x7665C, Q6: 0x7666C, Q7: 0x7667C	<a href="#">Table 443, p. 585</a>
Receive Queue Command (RQC) Register	Port0: 0x72680, Port1: 0x76680	<a href="#">Table 444, p. 585</a>
Transmit Current Served Descriptor Pointer Register	Port0: 0x72684, Port1: 0x76684	<a href="#">Table 445, p. 586</a>
Transmit Current Queue Descriptor Pointer (TCQDP) Register (n=0–7)	Port0: Q0: 0x726C0, Q1: 0x726C4, Q2: 0x726C8, Q3: 0x726CC, Q4: 0x726D0, Q5: 0x726D4, Q6: 0x726D8, Q7: 0x726DC Port1: Q0: 0x766C0, Q1: 0x766C4, Q2: 0x766C8, Q3: 0x766CC, Q4: 0x766D0, Q5: 0x766D4, Q6: 0x766D8, Q7: 0x766DC	<a href="#">Table 446, p. 586</a>
Destination Address Filter Special Multicast Table (DFSMT) Register (n=0–63)	Port0: Register0: 0x73400, Register1: 0x73404...Register63: 0x7 34FC Port1: Register0: 0x77400, Register1: 0x77404...Register63: 0x7 74FC	<a href="#">Table 447, p. 586</a>
Destination Address Filter Other Multicast Table (DFOMT) Register (n=0–63)	Port0: Register0: 0x73500, Register1: 0x73504...Register63: 0x7 35FC Port1: Register0: 0x77500, Register1: 0x77504...Register63: 0x7 75FC	<a href="#">Table 448, p. 587</a>

**Table 400: Register Map Table for the Gigabit Ethernet Controller Registers (Continued)**

Register Name	Offset	Table and Page
Destination Address Filter Unicast Table (DFUT) Register (n=0–3)	Port0: Register0: 0x73600, Register1: 0x73604, Register2: 0x73608, Register3: 0x7360C Port1: Register0: 0x77600, Register1: 0x77604, Register2: 0x77608, Register3: 0x7760C	Table 449, p. 589
<b>Tx Queues Arbiter</b>		
Transmit Queue Command (TQC) Register	Port0: 0x72448, Port1: 0x76448	Table 450, p. 590
Transmit Queue Fixed Priority Configuration (TQFPC) Register	Port0: 0x724DC, Port1: 0x764DC	Table 451, p. 591
Port Transmit Token-Bucket Rate Configuration (PTTBRC) Register	Port0: 0x724E0, Port1: 0x764E0	Table 452, p. 591
Transmit Queue Command1 (TQC1) Register	Port0: 0x724E4, Port1: 0x764E4	Table 453, p. 591
Port Maximum Transmit Unit (PMTU) Register	Port0: 0x724E8, Port1: 0x764E8	Table 454, p. 592
Port Maximum Token Bucket Size (PMTBS) Register	Port0: 0x724EC, Port1: 0x764EC	Table 455, p. 592
Queue Transmit Token-Bucket Counter (QxTTBC) Register (n=0–7)	Port0: Q0: 0x72700, Q1: 0x72710, Q2: 0x72720, Q3: 0x72730, Q4: 0x72740, Q5: 0x72750, Q6: 0x72760, Q7: 0x72770 Port1: Q0: 0x76700, Q1: 0x76710, Q2: 0x76720, Q3: 0x76730, Q4: 0x76740, Q5: 0x76750, Q6: 0x76760, Q7: 0x76770	Table 456, p. 593
Transmit Queue Token Bucket Configuration (TQxTBC) Register (n=0–7)	Port0: Q0: 0x72704, Q1: 0x72714, Q2: 0x72724, Q3: 0x72734, Q4: 0x72744, Q5: 0x72754, Q6: 0x72764, Q7: 0x72774 Port1: Q0: 0x76704, Q1: 0x76714, Q2: 0x76724, Q3: 0x76734, Q4: 0x76744, Q5: 0x76754, Q6: 0x76764, Q7: 0x76774	Table 457, p. 593
Transmit Queue Arbiter Configuration (TQxAC) Register (n=0–7)	Port0: Q0: 0x72708, Q1: 0x72718, Q2: 0x72728, Q3: 0x72738, Q4: 0x72748, Q5: 0x72758, Q6: 0x72768, Q7: 0x72778 Port1: Q0: 0x76708, Q1: 0x76718, Q2: 0x76728, Q3: 0x76738, Q4: 0x76748, Q5: 0x76758, Q6: 0x76768, Q7: 0x76778	Table 458, p. 594
Port Transmit Token-Bucket Counter (PTTBC) Register	Port0: 0x72780, Port1: 0x76780	Table 459, p. 595
Transmission Queue IPG (TQxIPG) Register (n=2–3)	Port0: Q2: 0x727A8, Q3: 0x727B8 Port1: Q2: 0x767A8, Q3: 0x767B8	Table 460, p. 595
High Token in Low Packet (HITKNinLOPKT) Register	Port0: 0x727C0, Port1: 0x767C0	Table 461, p. 595



Table 400: Register Map Table for the Gigabit Ethernet Controller Registers (Continued)

Register Name	Offset	Table and Page
High Token in Asynchronous Packet (HITKNinASYNCPKT) Register	Port0: 0x727C4, Port1: 0x767C4	<a href="#">Table 462, p. 595</a>
Low Token in Asynchronous Packet (LOTKNinASYNCPKT) Register	Port0: 0x727C8, Port1: 0x767C8	<a href="#">Table 463, p. 596</a>
Transmission Speed (TS) Register	Port0: 0x727D0, Port1: 0x767D0	<a href="#">Table 464, p. 596</a>

## A.8.1 Gigabit Ethernet Unit Global Registers

Table 401: PHY Address Register

Offset: Port0: 0x72000 Port1: 0x76000

Bit	Field	Type/InitVal	Description
4:0	PhyAd	RW 0x8	PHY Device Address This field defines the PHY address of the PHY connected to the port. This PHY address is used during Auto-Negotiation.
5	Reserved	RSVD 0x0	Reserved
31:6	Reserved	RO 0x0	Reserved

Table 402: SMI Register

Offset: Port0: 0x72004 Port1: 0x76004

Bit	Field	Type/InitVal	Description
15:0	Data	RW 0x0	Management for SMI READ operation: Two transactions are required: (1) Management write to the SMI register where <Opcode> = 1, <PhyAd>, <RegAd> with the data having any value. (2) Management read from the SMI register. When reading back the SMI register, the data is the addressed PHY register content if the <ReadValid> bit[27] is 1. The data remains undefined as long as <ReadValid> is 0. Management for SMI WRITE operation: One Management transaction is required: Management writes to the SMI register with <Opcode> = 0, <PhyAd>, <RegAd> with the data to be written to the addressed PHY register.
20:16	PhyAd	RW 0x0	PHY Device Address This field defines the PHY address used during the CPU SMI READ or WRITE access.
25:21	RegAd	RW 0x0	PHY Device Register Address

**Table 402: SMI Register (Continued)**  
Offset: Port0: 0x72004 Port1: 0x76004

Bit	Field	Type/InitVal	Description
26	Opcode	RW 0x1	Selects SMI Write/Read 0 = Write 1 = Read
27	ReadValid	RO 0x0	1 = Indicates that the READ operation for the addressed RegAd register has completed, and that the data is valid in the <Data> field.
28	Busy	RO 0x0	1 = Indicates that an operation is in progress and that the CPU should not write to the SMI register during this time.
31:29	N/A	RO 0x0	These bits should be driven to 0x0 during any WRITE to the SMI register.

**Table 403: Ethernet Unit Default Address (EUDA) Register**  
Offset: Port0: 0x72008 Port1: 0x76008

Bit	Field	Type/InitVal	Description
31:0	DAR	RW 0x0	Specifies the Default Address to which the Ethernet unit directs no match, multiple address hits, and address protect violations. Occurrence of this event may be the result of programming errors of the descriptor pointers or buffer pointers.

**Table 404: Ethernet Unit Default ID (EUDID) Register**  
Offset: Port0: 0x7200C Port1: 0x7600C

Bit	Field	Type/InitVal	Description
3:0	DIDR	RW 0x0	Specifies the ID of the target unit to which the Ethernet unit directs no match and address protect violations. Identical to Base Address register's <Target> field encoding.
11:4	DATTR	RW 0xE	Specifies the Default Attribute of the target unit to which the Ethernet unit directs no match and address protect violations. Identical to Base Address register's <Attr> field encoding.
31:12	Reserved	RO 0x0	Read Only



**Table 405: Ethernet Unit Interrupt Cause (EUIC) Register**  
Offset: Port0: 0x72080 Port1: 0x76080

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> Write 0 to clear interrupt bits. Writing 1 does not effect the interrupt bits.			
0	EtherIntSum	RO 0x0	Ethernet Unit Interrupt Summary This bit is a logical OR of the unmasked bits[12:1] in the register.
1	Parity	RWOC 0x0	Parity Error  Effect on Tx DMA operation: - If a parity error occurs upon descriptor fetch, the Tx DMA asserts the Tx Error interruption. - If the parity error occurred on the first descriptor fetch, the DMA stops and disables the queue. - If parity error is detected on a packet's data, the Tx DMA continues with the transmission but does not ask to generate CRC at the end of the packet.  Effect on Rx DMA operation: - If a parity error occurs upon descriptor fetch, the Rx_DMA asserts the Rx Error interrupt. - If the parity error occurred on the first descriptor fetch, the DMA stops and disables the queue.
2	Address Violation	RWOC 0x0	This bit is set if an Ethernet DMA violates a window access protection
3	Address NoMatch	RWOC 0x0	This bit is set if an Ethernet DMA address does not match any of the Ethernet address decode windows.
4	SMIdone	RWOC 0x0	SMI Command Done Indicates the SMI completed a MII management command (either read or write) initiated by the CPU writing to the SMI register.
5	Count_wa	RWOC 0x0	Counters Wrap Around Indication MIB Counter WrapAround Interrupt is set if one of the MIB counters wrapped around (passed 32 bits).
6	Reserved	RSVD 0x0	Reserved
7	Internal AddrError	RWOC 0x0	Internal Address Error is set when there is an access to an illegal offset of the internal registers. When set, the Internal Address Error register locks the address that caused the error.
13:8	Reserved	RSVD 0x0	Reserved
31:14	Reserved	RO 0x0	Reserved

**Table 406: Ethernet Unit Interrupt Mask (EUIIM) Register**  
Offset: Port0: 0x72084 Port1: 0x76084

Bit	Field	Type/InitVal	Description
12:0	Various	RW 0x0	Mask bits for Unit Interrupt Cause register 0 = Mask 1 = Do not mask
31:13	Reserved	RO 0x0	Reserved

**Table 407: Ethernet Unit Error Address (EUEA) Register**  
Offset: Port0: 0x72094 Port1: 0x76094

Bit	Field	Type/InitVal	Description
31:0	Error Address	RO 0x0	Locks the address, if there is an address violation of the DMA such as: Multiple Address window hit, No Hit, or Access Violations. The Address is locked until the register is read. (Used for software debug after address violation interrupt is raised.) This field is read only

**Table 408: Ethernet Unit Internal Address Error (EUIAE) Register**  
Offset: Port0: 0x72098 Port1: 0x76098

Bit	Field	Type/InitVal	Description
1:0	Reserved	RO 0x0	Reserved
15:2	Internal Address	RO 0x0	If there is an address violation of unmapped access to the Gigabit Ethernet unit top registers, it locks the relevant internal address bits (bits [15:2]). The Address is locked until the register is read. <b>NOTE:</b> This field is used for software debugging after an address violation interrupt is raised.
31:16	Reserved	RO 0x0	Reserved

**Table 409: Ethernet Unit Control (EUC) Register**  
Offset: Port0: 0x720B0 Port1: 0x760B0

Bit	Field	Type/InitVal	Description
0	Port_DPPar	RW 0x0	Gigabit Ethernet port data path parity select: <b>NOTE:</b> Should be set to even parity for normal operation. Odd parity is for debugging only. 0 = Even Parity 1 = Odd Parity



**Table 409: Ethernet Unit Control (EUC) Register (Continued)**  
Offset: Port0: 0x720B0 Port1: 0x760B0

Bit	Field	Type/InitVal	Description
1	Polling	RW 0x1	Polling Enable To disable the polling operation, the software must reset this bit. 0 = Disable 1 = Enable
3:2	Reserved	RO 0x0	Reserved
5:4	Reserved	RW 0x0	Reserved
6	Reserved	RW 0x0	Reserved Must be 0.
15:7	Reserved	RO 0x0	Reserved
16	Reserved	RW 0x1	Reserved
19:17	Reserved	RO 0x0	Reserved
20	Port Reset	RW 0x0	Port reset 0 = Inactive: No reset to the port 1 = Active: Port in software reset
31:21	Reserved	RO 0x0	Reserved

**Table 410: Base Address Register (n=0–5)**  
Offset: Port0: BA0: 0x72200, BA1: 0x72208, BA2: 0x72210, BA3: 0x72218, BA4: 0x72220, BA5: 0x72228  
Port1: BA0: 0x76200, BA1: 0x76208, BA2: 0x76210, BA3: 0x76218, BA4: 0x76220, BA5: 0x76228

Bit	Field	Type/InitVal	Description
3:0	Target	RW 0x0	Specifies the target resource associated with this window. See Address Decoding chapter for full details
7:4	Reserved	RW 0x0	Reserved
15:8	Attr	RW 0x0	Specifies target specific attributes depending on the target interface. See Address Decoding chapter for full details.
31:16	Base	RW 0x0	Base address Used together with the size register to set the address window size and location within the range of 4 GB space. An address driven by one of the Ethernet SDMAs is considered as a window hit if: $(address   size) == (base   size)$ .

**Table 411: Size (S) Register (n=0–5)**

Offset: Port0: SR0: 0x72204, SR1: 0x7220C, SR2: 0x72214, SR3: 0x7221C, SR4: 0x72224,

SR5: 0x7222C

Port1: SR0: 0x76204, SR1: 0x7620C, SR2: 0x76214, SR3: 0x7621C, SR4: 0x76224,

SR5: 0x7622C

Bit	Field	Type/InitVal	Description
15:0	Reserved	RO 0x0	Reserved
31:16	Size	RW 0x0	Window size Used together with the size register to set the address window size and location within the range of 4 GB space. Must be programmed from LSB to MSB as sequence of 1s followed by sequence of 0s. The number of 1s specifies the size of the window in 64-KB granularity (for example, a value of 0x00FF specifies 256x64k = 16 MB). An address driven by one of the Ethernet MACs is considered as a window hit if: $(address \mid size) == (base \mid size)$ .

**Table 412: High Address Remap (HA)1 Register (n=0–3)**

Offset: Port0: HARR0: 0x72280, HARR1: 0x72284, HARR2: 0x72288, HARR3: 0x7228C

Port1: HARR0: 0x76280, HARR1: 0x76284, HARR2: 0x76288, HARR3: 0x7628C

Bit	Field	Type/InitVal	Description
Remap 0 corresponds to Base Address register 0; Remap 1 to Base Address register 1; Remap 2 to Base Address register 2; and Remap 3 to Base Address register 3.			
31:0	Remap	RW 0x0	Remap address Specifies address bits[63:32] to be driven to the target interface. Relevant only for target interfaces that supports more than 4 GB of address space.

**Table 413: Base Address Enable (BARE) Register**

Offset: Port0: 0x72290 Port1: 0x76290

Bit	Field	Type/InitVal	Description
5:0	En	RW 0x3F	Address window enable This is one bit per window. If it is set to 0, the corresponding address window is enabled. (Bit[0] matches to Window0; bit[1] matches to Window1, etc.) 0 = Enable 1 = Disable
31:6	Reserved	RO 0x0	Reserved



**Table 414: Ethernet Port Access Protect (EPAP) Register**  
Offset: Port0: 0x72294 Port1: 0x76294

Bit	Field	Type/InitVal	Description
1:0	Win0	RW 0x3	Window0 access control In case of access violation (for example, write data to a read only region), an interrupt is set, and the transaction is written or read from the default address register. 0 = No Access 1 = Read Only 2 = Reserved 3 = Full Access: (Read or Write)
3:2	Win1	RW 0x3	Window1 access control (the same as Win0 access control)
5:4	Win2	RW 0x3	Window2 access control (the same as Win0 access control)
7:6	Win3	RW 0x3	Window3 access control (the same as Win0 access control)
9:8	Win4	RW 0x3	Window4 access control (the same as Win0 access control)
11:10	Win5	RW 0x3	Window5 access control (the same as Win0 access control)
31:12	Reserved	RO 0x0	Reserved

**Table 415: Port0/1 Mbus Top Arbiter Register**  
Offset: Port0: 0xE20C0 Port1: 0xE60C0

Bit	Field	Type/InitVal	Description
1:0	MbusRegretCounter Value	RW 0x0	Mbus Regret Counter Value Controls the time until regret will be done on read request with no Mbus acknowledge. This field is set to avoid starvation on reads.  Must be set to 0x0. 0 = 256 cycles 1 = 128 cycles 2 = 64 cycles 3 = 32 cycles
2	G0Only	RW 0x0	Must be set to 0x0. 0 = Port0: Working with port0 only. 1 = Dual Ports: Working with dual ports.
31:3	Reserved	RSVD 0x0	Reserved

## A.8.2 Port Control Registers

**Table 416: Port Configuration (PxC) Register**

Offset: Port0: 0x72400 Port1: 0x76400

Bit	Field	Type/InitVal	Description
0	UPM	RW 0x0	Unicast Promiscuous mode 0 = Normal: Unicast frames are received only if the destination address is found in the DA-filter table and DA is matched against the port DA MAC Address base. 1 = Promiscuous: Unicast unmatched frames are received to the Rx queue.
3:1	RXQ	RW 0x0	Default Rx Queue Is the Default Rx Queue for not matched Unicast frames when UPM bit is set. It is also the default Rx Queue for all MAC broadcast (except for ARP broadcast that has a different field for default queue) if receiving them is enabled.
6:4	RXQArp	RW 0x0	Default Rx Queue for ARP Broadcasts, if receiving ARP Broadcasts is enabled.
7	RB	RW 0x0	Reject mode of MAC Broadcasts that are not IP or ARP Broadcast. 0 = Receive: To RXQ queue 1 = Reject
8	RBIP	RW 0x0	Reject mode of MAC Broadcasts that are IP (EtherType 0x800). 0 = Receive: To RXQ queue 1 = Reject
9	RBArp	RW 0x0	Reject mode of MAC Broadcasts that are ARP (EtherType 0x806). 0 = Receive: To RXQArp queue 1 = Reject
11:10	Reserved	RW 0x0	Reserved
12	AMNoTxES	RW 0x0	Automatic mode not updating Error Summary in Tx descriptor. When set, for each Transmitted packet with <AM>=1 on the first descriptor, no status will be reported in the last descriptor (see Transmit Descriptor--Byte Count). The advantage of using this bit is that it avoids another write to memory to update the error status.
13	Reserved	RW 0x0	Reserved Must be set to 0.
14	TCP_CapEn	RW 0x0	Capture TCP frames to <TCPQ> 0 = Disable 1 = Enable
15	UDP_CapEn	RW 0x0	Capture UDP frames to <UDPQ> 0 = Disable 1 = Enable



**Table 416: Port Configuration (PxC) Register (Continued)**  
Offset: Port0: 0x72400 Port1: 0x76400

Bit	Field	Type/InitVal	Description
18:16	TCPQ	RW 0x0	Captured TCP frames are directed to this Queue number.
21:19	UDPQ	RW 0x0	Captured UDP frames are directed to this Queue number.
24:22	BPDUQ	RW 0x7	Captured BPDU frames (if the Port Configuration Extended register <Span> field is set) are directed to this Queue number.
25	RxCs	RW 0x1	Rx TCP checksum mode 0 = No header: Calculate without pseudo-header. 1 = With header: Calculation include pseudo-header.
31:26	Reserved	RO 0x0	Reserved

**Table 417: Port Configuration Extend (PxCX) Register**  
Offset: Port0: 0x72404 Port1: 0x76404

Bit	Field	Type/InitVal	Description
0	Reserved	RO 0x0	Reserved
1	Span	RW 0x0	Spanning Tree packets capture enable 0 = Treated: BPDU packets are treated as normal Multicast packets. 1 = Trapped: BPDU packets are trapped and sent to the Port Configuration register BPDU queue.
2	Reserved	RO 0x0	Reserved
3	TxCRCDis	RW 0x0	Tx CRC generation 0 = Enable: Port generates good CRC to all Tx packets (except for error conditions). 1 = Disable: The software determines whether CRC is added at the end of the packet or not (for debug purpose only).
31:4	Reserved	RO 0x0	Reserved

**Table 418: MII Serial Parameters Register**  
Offset: Port0: 0x72408 Port1: 0x76408

Bit	Field	Type/InitVal	Description
12:0	Reserved	RO 0x0	Reserved

**Table 418: MII Serial Parameters Register (Continued)**  
Offset: Port0: 0x72408 Port1: 0x76408

Bit	Field	Type/InitVal	Description
16:13	IPG-DATA	RW 0xC	The step is 8-bit times. The value may vary between 12- and 124-bit times. <b>NOTE:</b> These bits can only be changed when the <PortEn> field is set to 0 in the Port Control Register (The port is disabled).
31:17	Reserved	RO 0x0	Reserved

**Table 419: VLAN EtherType (EVLANE) Register**  
Offset: Port0: 0x72410 Port1: 0x76410

Bit	Field	Type/InitVal	Description
15:0	VL_EtherType	RW 0x8100	The EtherType for packets carrying the VLAN tag, for 802.1p priority field processing, and for continued parsing of the received frames layer3/4 headers.
31:16	Reserved	RO 0x0	Reserved

**Table 420: MAC Address Low (MACAL) Register**  
Offset: Port0: 0x72414 Port1: 0x76414

Bit	Field	Type/InitVal	Description
15:0	MAC[15:0]	RW 0x0	The least significant bits of the MAC Address Used for both flow-control Pause frames as a source address, as well as for address filtering (see "Parsing the Frames").
31:16	Reserved	RO 0x0	Read Only

**Table 421: MAC Address High (MACAH) Register**  
Offset: Port0: 0x72418 Port1: 0x76418

Bit	Field	Type/InitVal	Description
31:0	MAC[47:16]	RW 0x0	The most significant bits of the MAC Address Used for both flow-control Pause frames as source address, as well as for address filtering (see "Parsing the Frames"). <b>NOTE:</b> <MAC[40]> is the Multicast/Unicast bit.



**Table 422: SDMA Configuration (SDC) Register**  
Offset: Port0: 0x7241C Port1: 0x7641C

Bit	Field	Type/InitVal	Description
0	RIFB	RW 0x0	Receive Interrupt on Frame Boundaries When set, the SDMA Rx generates interrupts only on frame boundaries (i.e. after writing the frame status to the descriptor). See also the <IPGIntRx> field description (bits[21:7]) for further masking.
3:1	RxBSz	RW 0x4	Rx Burst Size Sets the maximum burst size for Rx SDMA transactions: <b>NOTE:</b> This field effects only data-transfers. Descriptor fetch is always done with 4LW burst size. 0 = 1 64-bit: Burst limited to 1 64-bit words 1 = 2 64-bit: Burst limited to 2 64-bit words 2 = 4 64-bit: Burst limited to 4 64-bit words 3 = 8 64-bit: Burst limited to 8 64-bit words 4 = 16 64-bit: Burst limited to 16 64-bit words
4	BLMR	RW 0x1	Big/Little Endian Receive Mode The DMA supports Big or Little Endian configurations per channel. The BLMR bit only affects data transfer to memory. 0 = Byte swap 1 = No swap
5	BLMT	RW 0x1	Big/Little Endian Transmit Mode The DMA supports Big or Little Endian configurations per channel. The BLMT bit only affects data transfer from memory. 0 = Byte swap 1 = No swap
6	SwapMode	RW 0x0	Swap mode The DMA supports swapping, for descriptors only, for both receive and transmit ports, on every access to memory space. 0 = No swapping 1 = Byte swap: In every 64-bit word of the descriptor, the byte order is swapped such that byte 0 is placed in byte 7, byte 7 is placed in byte 0, byte 1 is placed in byte 6, byte 6 is placed in byte 1, byte 2 is placed in byte 5, etc.
21:7	IPGIntRx	RW 0x0	Rx frame IPG between interrupts counter and enable This field provides a way to force a delay from the last ICR[RxBufferQueue] interrupt from any of the queues, to the next RxBufferQueue interrupt from any of the queues. The ICR bits still reflect the new interrupt, but this masking is reflected by potentially not propagating to the chip main interrupt cause register. This provides a way for interrupt coalescing on receive packet events. The time is calculated in multiples of 64 clock cycles. Valid values are 0 (No delay between packets to CPU, the counter is effectively disabled.), through 0x3FFF (1,048,544 clock cycles). <b>NOTE:</b> See the detailed description in "Interrupt Coalescing".

**Table 422: SDMA Configuration (SDC) Register (Continued)**  
Offset: Port0: 0x7241C Port1: 0x7641C

Bit	Field	Type/InitVal	Description
24:22	TxB SZ	RW 0x4	<p>Tx Burst Size  Sets the maximum burst size for Tx SDMA transactions:  <b>NOTE:</b> This field effects only data-transfers. Descriptor fetch is done always with 4LW burst size.</p> <p>0 = 1 64-bit: Burst limited to 1 64-bit words  1 = 2 64-bit: Burst limited to 2 64-bit words  2 = 4 64-bit: Burst limited to 4 64-bit words  3 = 8 64-bit: Burst limited to 8 64-bit words  4 = 16 64-bit: Burst limited to 16 64-bit words</p>
25	IPGIntRx	RW 0x0	MSB bit of IPGIntRx bits[21:7]
31:26	Reserved	RO 0x0	Read Only

**Table 423: IP Differentiated Services CodePoint 0 to Priority (DSCP0) Register**  
Offset: Port0: 0x72420 Port1: 0x76420

Bit	Field	Type/InitVal	Description
29:0	TOS_Q[29:0]	RW 0x0	<p>The Priority queue mapping of received frames with DSCP values 0 (corresponding to TOS_Q[2:0]) through 9 (corresponding to TOS_Q[29:27]).  <b>NOTE:</b> The initial value means that ToS does not effect queue decisions.</p>
31:30	Reserved	RO 0x0	Reserved

**Table 424: IP Differentiated Services CodePoint 1 to Priority (DSCP1) Register**  
Offset: Port0: 0x72424 Port1: 0x76424

Bit	Field	Type/InitVal	Description
29:0	TOS_Q[59:30]	RW 0x0	<p>The Priority queue mapping of received frames with DSCP values 10 (corresponding to TOS_Q[32:30]) through 19 (corresponding to TOS_Q[59:57]).  <b>NOTE:</b> The initial value means that ToS does not effect queue decisions.</p>
31:30	Reserved	RO 0x0	Reserved

**Table 425: IP Differentiated Services CodePoint 2 to Priority (DSCP2/3/4/5) Register (n=0–3)**

Offset: Port0: DSCP0: 0x72428, DSCP1: 0x7242C, DSCP2: 0x72430, DSCP3: 0x72434  
 Port1: DSCP0: 0x76428, DSCP1: 0x7642C, DSCP2: 0x76430, DSCP3: 0x76434

Bit	Field	Type/InitVal	Description
This represents DSCP2-DSCP5.			
29:0	TOS_Q[89:60]	RW 0x0	The Priority queue mapping of received frames with DSCP values 20 (corresponding to TOS_Q[62:60]) through 29 (corresponding to TOS_Q[89:87]). <b>NOTE:</b> The initial value means that ToS does not effect queue decisions.
31:30	Reserved	RO 0x0	Reserved

**Table 426: IP Differentiated Services CodePoint 6 to Priority (DSCP6) Register**

Offset: Port0: 0x72438 Port1: 0x76438

Bit	Field	Type/InitVal	Description
11:0	TOS_Q[191:180]	RW 0x0	The Priority queue mapping of received frames with DSCP values 60 (corresponding to TOS_Q[2:0]) through 63 (corresponding to TOS_Q[191:180]). <b>NOTE:</b> The initial value means that ToS does not effect queue decisions.
31:12	Reserved	RO 0x0	Reserved

**Table 427: Port Serial Control0 (PSC0) Register**

Offset: Port0: 0x7243C Port1: 0x7643C

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The following steps for changing the value of the Port Serial Control register bits do not apply to <ForceLink Pass> bit[1], <ForceFCMode> bits[6:5], <ForceBPMODE> bits[8:7], and <ForceLinkFail> bit [10].			
When changing the value of the Port Serial Control register bits, the following steps must be taken:			
If the Tx DMA is enabled before, it must be disabled through its command registers. The CPU must verify that it is disabled by polling the TxQ Command register, then the CPU must poll each port's Port Status register to verify that:			
There is no transmission in progress (<TxInProg> bit[7] is 0) It's Tx FIFO is empty (<TcFIFOEmpl> bit[10] is 1)			
Read the Port Serial Control register.			
Disable the Serial port by writing a 0 to bit[0] (<PortEn>) of the Port Serial Control register.			
Set the desired bits in the Port Serial Control register.			
Enable the Serial port by writing a 1 to bit[0] (<PortEn>) of the Port Serial Control register.			
Re-enable the Tx DMAs, according to their initialization sequence, as necessary.			
0	PortEn	RW 0x0	Serial Port Enable No frames will be received or transmitted while the serial port is disabled. <b>NOTE:</b> Disabling the port should not occur during Tx DMA operation. See the guidelines at the top of this table. 0 = Disable: Serial Port is disabled. 1 = Enable: Serial Port is enabled.

**Table 427: Port Serial Control0 (PSC0) Register (Continued)**  
Offset: Port0: 0x7243C Port1: 0x7643C

Bit	Field	Type/InitVal	Description
1	ForceLinkPass	RW 0x0	Force Link status on port to Link UP state 0 = Not Force: Do NOT Force Link Pass 1 = Force: Force Link pass
2	AN_Duplex	RW 0x0	Enable Auto-Negotiation for duplex mode <b>NOTE:</b> Half-Duplex mode is not supported in 1000 Mbps mode. 0 = Enable 1 = Disable
3	AN_FC	RW 0x1	Enable Auto-Negotiation for Flow Control When enabled, the port can either advertise no flow control or symmetric flow-control according to the Port Serial Control Register <Pause_Adv> bit. Asymmetric flow-control advertisement is not supported. 0 = Enable 1 = Disable
4	Pause_Adv	RW 0x1	Flow control advertise The port does not modify this bit as result of the Auto-Negotiation process (unlike the Port Status Register's <EnFC> bit, which may be modified based on Auto-Negotiation results). 0 = Flow control: Advertise no flow control. 1 = Sym flow control: Advertise symmetric flow control support in Auto-Negotiation.
6:5	ForceFCMode	RW 0x0	When the flow-control operation is enabled in the <EnFC> field in the Ethernet Port Status0 Register, the port will transmit, in full duplex, Pause enable or disable frames, depending on the CPU writing the values 00 or 01 to this field. <b>NOTE:</b> When the link fails, this field changes to disabled 0x00, and it must be reprogrammed only after the link is up. 0 = No Pause: No Pause disable frames are sent. However, when the value of the field is changed by the CPU from 01 to 00, the port will send a single Pause enable packet (timer 0x0000) to enable the other side to transmit. 1 = Pause: When this field is set to 01 value, and Flow Control is enabled (The <EnFC> field in the Ethernet Port Status0 Register is set) then Pause disable frames (timer 0xFFFF) are retransmitted at least every 30 ms (30 ms assumes 166 MHz TClik). 2 = Reserved 3 = Reserved
8:7	ForceBPMMode	RW 0x0	When this bit is set, the port will start transmitting JAM on the line (Backpressure) in half-duplex (which is only supported in 10/100 Mbps modes) according to the settings listed below: <b>NOTE:</b> When the link fails, this field changes to disabled 0x00, and it must be reprogrammed only after the link is up. 0 = No JAM: (no backpressure) 1 = JAM: Is transmitted continuously, on next frame boundary. 3 = Reserved
9	Reserved	RW 0x1	Reserved



**Table 427: Port Serial Control0 (PSC0) Register (Continued)**  
Offset: Port0: 0x7243C Port1: 0x7643C

Bit	Field	Type/InitVal	Description
10	ForceLinkFail	RW 0x0	Force Link status on port to Link DOWN state 0 = Force: Force Link Fail 1 = Not Force: Do NOT Force Link Fail
11	Reserved	RSVD 0x0	Reserved
12	Reserved	RW 0x0	Reserved
13	ANSpeed	RW 0x0	Enable Auto-Negotiation of interface speed 0 = Enable: Enable update 1 = Disable: Disable update
14	DTEAdvert	RW 0x0	DTE advertise The value of this bit is written to bit 9.10 of the 1000BaseT PHY device after power up or detection of a link failure.
15	Reserved	RW 0x0	Reserved Must be 0x0.
16	Reserved	RW 0x0	Reserved Must be 0x0.
19:17	MRU	RW 0x1	The Maximal Receive Packet Size <b>NOTE:</b> Modes 3-5 are supported only when operating in 1000 Mbps mode. Receiving 9700 byte frames is supported only during 1000 Mbps operation. Receiving frames over 2 KB, during 100 Mbps operation, may result in overrun/underrun in some cases. 0 = 1518 bytes: Accept packets up to 1518 bytes in length 1 = 1522 bytes: Accept packets up to 1522 bytes in length 2 = 1552 bytes: Accept packets up to 1552 bytes in length 3 = 9022 bytes: Accept packets up to 9022 bytes in length 4 = 9192 bytes: Accept packets up to 9192 bytes in length 5 = 9700 bytes: Accept packets up to 9700 bytes in length 6 = Reserved 7 = Reserved
20	Reserved	RW 0x0	Reserved
21	Set_FullDx	RW 0x1	Half/Full Duplex Mode <b>NOTE:</b> This bit has no function when the <AN_Duplex> Port Serial Control0 (PSC0) Register is set to enable. 0 = Half Duplex: Port works in Half Duplex mode 1 = Full Duplex: Port works in Full Duplex mode

**Table 427: Port Serial Control0 (PSC0) Register (Continued)**  
Offset: Port0: 0x7243C Port1: 0x7643C

Bit	Field	Type/InitVal	Description
22	SetFCEn	RW 0x1	<p>Enable receiving and transmitting of 802.3 Flow Control frames in full duplex, or enabling of backpressure in half duplex.</p> <p><b>NOTE:</b> This bit has no function when the &lt;AN_FC&gt; Port Serial Control0 (PSC0) Register is set to enable.</p> <p>0 = Disabled 1 = Enabled</p>
23	SetGMII Speed	RW 0x1	<p><b>NOTE:</b> This bit has no function when &lt;ANSpeed&gt; is set to enable.</p> <p>0 = 10/100 Mbps: Port works at 10/100 Mbps 1 = 1000 Mbps: Port works at 1000 Mbps</p>
24	SetMII Speed	RW 0x1	<p>If Speed Auto-Negotiation is disabled, &lt;ANSpeed&gt; in the Port Serial Control0 (PSC0) Register = 0, then this bit should set to the speed of the MII interface.</p> <p><b>NOTE:</b> This bit has no function when the &lt;ANSpeed&gt; Port Serial Control0 (PSC0) Register is enabled.</p> <p>0 = 10 Mbps: Port works in 10 Mbps. 1 = 100 Mbps: Port works in 100 Mbps.</p>
25	Reserved	RW 0x0	Reserved
27:26	Reserved	RW 0x0	Reserved Must be 0.
31:28	Reserved	RO 0x0	Reserved Read Only

**Table 428: VLAN Priority Tag to Priority (VPT2P) Register**  
Offset: Port0: 0x72440 Port1: 0x76440

Bit	Field	Type/InitVal	Description
23:0	Priority[23:0]	RW 0x0	<p>The Priority queue mapping of received frames with 802.1p priority field values 0 (corresponding to Priority[2:0]) through 7 (corresponding to Priority[23:21]).</p> <p><b>NOTE:</b> The initial value means that Priority does not effect the queue decisions.</p>
31:24	Reserved	RO 0x0	Reserved

**Table 429: Ethernet Port Status 0 (PS0) Register**  
Offset: Port0: 0x72444 Port1: 0x76444

Bit	Field	Type/InitVal	Description
0	Reserved_0	RO 0x0	Reserved



**Table 429: Ethernet Port Status 0 (PS0) Register (Continued)**  
Offset: Port0: 0x72444 Port1: 0x76444

Bit	Field	Type/InitVal	Description
1	LinkUp	RO 0x0	The Link Status This bit is set forced to 1 when <ForceLinkPass> in the Port Serial Control0 (PSC0) Register = 1. This bit is set forced to 0 when <ForceLinkFail> in the Port Serial Control0 (PSC0) Register = 0. 0 = Link is down 1 = Link is up
2	FullDx	RO 0x0	Half-/Full-Duplex mode. This bit may change in any time when the <AN_Duplex> field in the Port Serial Control0 (PSC0) Register is set to enable. When <AN_Duplex> is clear (disabled), this bit is set by the management in PSCR <Set_FullDx>. <b>NOTE:</b> Half-Duplex is not supported in 1000 Mbps. Default value is determined by Auto-Negotiation for duplex mode, if the Port Control register's <AN_Duplex> bit is enabled. 0 = Half-Duplex: Port works in Half-Duplex mode. 1 = Full-Duplex: Port works in Full-Duplex mode.
3	EnFC	RO 0x0	<b>NOTE:</b> Set by Flow_Control Auto-Negotiation if PSCR <AN_FC> is enabled. Enables receiving 802.3 Flow_Control frames in full-duplex mode: If the <AN_FC> field in the Port Serial Control0 (PSC0) Register is enabled, then each time that Auto-Negotiation is performed, the value in the <EnFC> bit may change. 0 = Disabled 1 = Enabled
4	GMII Speed	RO 0x0	Meaningful when the interface mode is GMII/MII only. If the <ANSpeed> field in the Port Serial Control0 (PSC0) Register is enabled, then each time that Auto-Negotiation is performed, the value in the <GMII Speed> field may change. When this bit is 0, the <MII Speed> defines whether it is 10 Mbps or 100 Mbps When the <ANSpeed> bit is disabled, this bit is set by the management in the <SetGMII Speed> field of the Port Serial Control0 (PSC0) Register. 0 = 10/100 Mbps: Port works in 10/100 Mbps. 1 = 1000 Mbps: Port works in 1000 Mbps.
5	MII Speed	RO 0x0	MII Speed If the <ANSpeed> field in the Port Serial Control0 (PSC0) Register is enabled, then each time that Auto-Negotiation is performed, the value in the <MII Speed> may change. When <ANSpeed> is disabled, this bit is set by the management in the <SetMII Speed> field in the Port Serial Control0 (PSC0) Register. <b>NOTE:</b> The default value is determined in Auto-Negotiation for duplex mode when <ANSpeed> is enabled. 0 = 10 Mbps: Port works at 10 Mbps 1 = 100 Mbps: Port works at 100 Mbps
6	Reserved_6	RO 0x0	Reserved

**Table 429: Ethernet Port Status 0 (PS0) Register (Continued)**  
Offset: Port0: 0x72444 Port1: 0x76444

Bit	Field	Type/InitVal	Description
7	TxInProg	RO 0x0	Transmit in Progress Indicates that the port's transmitter is in an active transmission state.
8	Reserved_8	RO 0x0	Reserved
9	Reserved_9	RO 0x0	Reserved
10	TxFIFOEmp	RO 0x0	Set when the port Transmit FIFO is empty
11	RxFIFO1Emp	RO 0x1	Set when the port Receive FIFO1 is empty
12	RxFIFO2Emp	RO 0x1	Set when the port Receive FIFO2 is empty
31:13	Reserved_31_13	RO 0x0	Read Only

**Table 430: Port Serial Control1 (PSC1) Register**  
Offset: Port0: 0x7244C Port1: 0x7644C

Bit	Field	Type/InitVal	Description
0	Reserved	RW 0x0	Must be set to 0.
1	PcsLB	RW 0x0	Loopback on the PCS Block  <b>NOTE:</b> A change of this field must be done when the port's link is down. 0 = No loopback: Normal operation, no loopback 1 = Loopback: Loopback on the PCS block, the PCS Tx is connected directly to the PCS Rx, the Loopback clock used is according to the setting of <Clk125 BypassEn>.
2	Reserved	RW 0x0	Must set to 0.
3	RGMIIEn	RW 0x1	Selects MII/RGMII mode 0 = MII/GMII: MII/GMII interface 1 = RGMII: RGMII interface
4	PortReset	RW 0x1	Controls port reset 0 = No reset: Port is not reset. 1 = Reset: Port is reset.



**Table 430: Port Serial Control1 (PSC1) Register (Continued)**  
Offset: Port0: 0x7244C Port1: 0x7644C

Bit	Field	Type/InitVal	Description
5	Clk125 BypassEn	RW 0x0	<p>Enables the use of the external clock connected to this port regardless of the ports speed.</p> <p><b>NOTE:</b> A change of this field must be done when the port's link is down.</p> <p>0 = SERDES clock: The clock from the SERDES is used as the Tx Reference clock.</p> <p>1 = Interface clock: In Regular port, used to connect GSD_LB_CLK as the clock for loopback tests, and enables the CPU port to use the CPU_TX_MII_CLK.</p>
6	InBandAnEn	RW 0x0	<p>In Band Auto-Negotiation Enable</p> <p>In Band Auto-Negotiation is done, if the port is 1000Base-X for link and flow control support.</p> <p>When &lt;PortType&gt; = 1 (1000Base-X), this field must be set to 1.</p> <p>When &lt;PortType&gt; = 0, In Band flow control Auto-Negotiation is not supported (only speed and duplex are negotiated). In this case, flow control support is resolved according to SMI auto negotiation, if it is enabled (&lt;ANFC&gt; = 1) or according to &lt;SetFCEn&gt;, if it is disabled.</p> <p>A change of this field must be done when the port's link is down.</p> <p>0 = Disable</p> <p>1 = Enable</p>
7	InBandAnByPassEn	RW 0x1	<p>In Band Auto-Negotiation Bypass Enable</p> <p>Relevant only when Inband Auto-Negotiation is enabled (&lt;InBandAnEn&gt;= 1).</p> <p>When this bit is set to 1, if the link partner does not respond to the Auto-Negotiation process, the link is established by bypassing the Auto-Negotiation procedure.</p> <p>After bypassing Auto-Negotiation, Link is set to be UP, and duplex is set according to &lt;SetFullDx&gt;.</p> <p><b>NOTE:</b> A change of this field must be done when the port's link is down.</p> <p>0 = No bypass: Auto-Negotiation cannot be bypassed.</p> <p>1 = Bypass: Auto-Negotiation bypassed.</p>
8	InBandReStartAn	Special 0x0	<p>In Band Restart Auto-Negotiation</p> <p>Relevant only when Inband Auto-Negotiation is enabled (&lt;InBandAnEn&gt;= 1).</p> <p>Setting this bit restarts In Band Auto-Negotiation. This bit is reset by the device immediately after restarting the Auto-Negotiation process.</p>
10:9	Reserved	RSVD 0x2	Reserved
11	PortType	RW 0x0	<p>This bit indicates the port type.</p> <p>0 = All modes: Port can run 10/100/1000 Mbps (MII/RGMII)</p> <p>1 = 1000Base-X: Port is 1000Base-X only</p>
12	Reserved	RW 0x0	Must set to 0.
13	Reserved	RW 0x0	Reserved
14	Reserved	RSVD 0x0	Reserved

**Table 430: Port Serial Control1 (PSC1) Register (Continued)**  
Offset: Port0: 0x7244C Port1: 0x7644C

Bit	Field	Type/InitVal	Description
15	EnColOnBp	RW 0x1	Collision during Back-Pressure MIB counting  0 = Disable 1 = Enable
21:16	ColDomainLimit	RW 0x23	Even though collision domain is 64 bytes, there are quite a few cycles from the time the collision is seen on the line, until the Tx Lower that calculates collision domain, sees this indication. At that point, its counter has exceeded the 64 bytes of the collision domain, and so even though the collision was before 64 bytes were transmitted on the line, it is considered a late collision. To prevent this, the collision domain counter limit is configured by this input. The resolution of this limit is 2Bytes.
22	EnMIIOddPre	RW 0x0	Relevant when the port's speed is 10 or 100 Mbps (At those speeds, the interface is made of nibbles). <b>NOTE:</b> A change of this field must be done when the port's link is down. 0 = Drop packet: drop packet due to odd number of preamble nibbles 1 = Receive packet: receive packet with odd number of preamble nibbles
31:23	Reserved	RO 0x0	Reserved

**Table 431: Ethernet Port Status1 (PS1) Register**  
Offset: Port0: 0x72450 Port1: 0x76450

Bit	Field	Type/InitVal	Description
0	PortRxPause	RO 0x0	Relevant when the port is in Full Duplex mode <FullDx> in the Ethernet Port Status0 (PS0) Register = 1 and <EnFC> in the same register = 1. The port received a 802.3 pause frame and stopped transmission of packets. 0 = Normal: The port is in normal state and sending packets, either an XON packet was received or the pause Timer has expired. 1 = Pause: The port received a pause and pause Timer has not expired yet. The port halts its transmission until the timer is expired.
1	PortTxPause	RO 0x0	Relevant when the port is in Full Duplex mode <FullDx> in the Ethernet Port Status0 (PS0) Register = 1 and <EnFC> in the Ethernet Port Status0 (PS0) Register = 1. The port is sending 802.3 flow control pause frames due to lack of buffers. 0 = Xon Status: The port has buffers for new packets reception (in Xon Status) 1 = Xoff Status: The number of buffers allocated for this port is above its configured Xoff threshold, and it is sending 802.3 flow control pause frames with Timer 0xFFFF, new packets are received until the port reaches its configured buffers limit threshold



**Table 431: Ethernet Port Status1 (PS1) Register (Continued)**  
Offset: Port0: 0x72450 Port1: 0x76450

Bit	Field	Type/InitVal	Description
2	PortDoingPressure	RO 0x0	Relevant when the port is in Half Duplex mode <FullDx> in the Ethernet Port Status0 (PS0) Register = 0 and <ForceBPMMode> in the same register = 1. Indicates that the port is doing back pressure due to lack of buffers 0 = Open Buffers: The port has buffers for new packets reception 1 = Full Buffers: The port has depleted all of its buffers and does not have buffers for the reception of new packets, while not transmitting packets, the port sends a jam pattern to prevent any packet reception
3	SyncFail10ms	RO 0x0	The 10-bit state machine has been in <SyncOk> = FAIL for more than 10 ms or less than 1.6us. Refer to IEEE 802.3 clause 36. 0 = Not in Sync 1 = In Sync
4	AnDone	RO 0x0	Auto-Negotiation done 0 = In progress: Auto-negotiation in progress 1 = Complete: Auto-negotiation is done
5	InBandAuto-NegBypassAct	RO 0x0	This bit indicates that In Band Auto-Negotiation Bypass was activated, When Doing Inband Auto-Negotiation and the peer does not reply and Inband Auto-Negotiation by pass is enabled. Link is set to be UP. Speed is set according to <SetMII Speed> in the Port Serial Control0 (PSC0) Register. Duplex is set according to <SetFullDx> in that same register. FC is set according to <SetFcEn> in that same register. <b>NOTE:</b> When port is In 1000Base-T, <PortType> = 0) In band flow control auto negotiation is not supported. In this case, flow control support is resolved according to SMI auto negotiation if it is enabled (<ANFC>= 1) or according to <SetFcEn> if it is disabled. 0 = No Bypass: Auto-Negotiation is cannot be bypassed 1 = Bypassed: Auto-Negotiation is bypassed
6	SerdessPLLLocked	RO 0x0	The 1.25Gbps SERDES incorporates a PLL, this signal indicates that the PLL is locked and the clock generated by the PLL is stable. <b>NOTE:</b> SW should clear port reset only after PLL is locked 0 = Not locked: The 1.25Gbps SERDES PLL is not locked. 1 = Locked: The 1.25Gbps SERDES PLL is locked.
7	SyncOk	RO 0x0	PCS has Detected a few comma patterns and is synchronized with its peer PCS Layer 0 = No comma: the PCS layer has not detected comma patterns 1 = Comma: PCS has detected several comma patterns and is synchronized with its peer PCS Layer

**Table 431: Ethernet Port Status1 (PS1) Register (Continued)**  
Offset: Port0: 0x72450 Port1: 0x76450

Bit	Field	Type/InitVal	Description
8	SquelchNot Detected	RO 0x0	The 1.25Gbps incorporates a receiver squelch detector, if the input signal swing is smaller than the threshold configured in <SquelchDetect Threshold> this field is set to 1. 0 = Above threshold: Input signal swing is larger than the threshold configured in <SquelchDetect Threshold>. 1 = Below threshold: Input signal swing is smaller than the threshold configured in <SquelchDetect Threshold>.
9	Reserved	RSVD 0x0	Reserved
31:10	Reserved	RO 0x0	Reserved Dual media PHY (bit[9]) is not supported.

**Table 432: Marvell Header Register**  
Offset: Port0: 0x72454 Port1: 0x76454

Bit	Field	Type/InitVal	Description
0	MHEn	RW 0x0	Marvell Header Enable  0 = Disable 1 = Enable: 2 octets of Marvell header are placed between SFD and DA.



**Table 432: Marvell Header Register (Continued)**  
Offset: Port0: 0x72454 Port1: 0x76454

Bit	Field	Type/InitVal	Description
2:1	DAPrefix	RW 0x0	<p>Rx queuing policy setting. If set to 0x0, regular Rx priority queuing. 0x1 - 0x3, queuing based on Marvell Header or DSA tag fields. It also depends on the number of queues available, as configured in MH_Mask.</p> <p>If using Marvell Header, configure as follows: DAPrefix = 0x1: 8 queues: queue # = PRI[2:0] 4 queues: queue # = PRI[2:1] 2 queues: queue# = PRI[2] DAPrefix = 0x2: 8 queues: queue# = (DBNum[0], PRI[2:1]) 4 queues: queue # = (DBNum[0], PRI[2]) 2 queues: queue# = DBNUM[0] DAPrefix = 0x3: 8 queues: queue# = (SPID[3:0]==&lt;SPID&gt;,PRI[2:1]) 4 queues: queue# = (SPID[3:0]==&lt;SPID&gt;,PRI[2]) 2 queues: queue# = (SPID[3:0]==&lt;SPID&gt;)</p> <p>If using DSA tag, configure as follows: DAPrefix = 0x1: 8 queues: queue # = UP[2:0] 4 queues: queue # = UP[2:1] 2 queues: queue# = UP[2] DAPrefix = 0x2: If the received packet is a TO_CPU type, receive queue is determined according to CPU_Code field of the DSA tag. If it is an extended DSA tag, mapping from CPU_Code to Rx queue is extracted from DFSMT table; if it is a none extended DSA tag, Rx queue is extracted from DFUT table. If the received packet is FORWARD format, Rx queue is calculated like in the case of DAPrefix = 0x1. DAPrefix = 0x3: 8 queues: queue# = (SrcPort==&lt;SPID&gt; &amp; SrcDev = &lt;SDID&gt;,UP[2:1]) 4 queues: queue# = (SrcPort==&lt;SPID&gt; &amp; SrcDev = &lt;SDID&gt;,UP[2]) 2 queues: queue# = (SrcPort==&lt;SPID&gt; &amp; SrcDev = &lt;SDID&gt;) <b>NOTE:</b> When using &lt;DA_PREFIX&gt; = 0x3, if there is no SrcPort (FORWARD format, with SrclsTrunk bit set), Rx queue is determined according to UP[2:0] field like in the case of DA_PREFIX = 0x1. If the received packet is not TO_CPU nor FORWARD format, packet is queued as if DA_PREFIX is 0x0 (regular queuing).</p>
3	Reserved	RO 0x0	Reserved
7:4	SPID[3:0]	RW 0x0	Source Port ID Setting to match the Switch SPID reported in Marvell Header for frame receive queuing.
9:8	MHMask	RW 0x0	When enabling receive queuing based on Marvell header (DAPrefix != 0x0), defines how many queues will be used: 0 = 8 queues: 8 queues (0-7) 1 = 4 queues: 4 queues (0-3) 3 = 2 queues: 2 queues (0-1)

**Table 432: Marvell Header Register (Continued)**  
Offset: Port0: 0x72454 Port1: 0x76454

Bit	Field	Type/InitVal	Description
11:10	DSAEn	RW 0x0	<p>DSA Tag enable</p> <p><b>NOTE:</b> DSA tag and Marvell Header can not work simultaneously. If using DSA tag, &lt;MH_En&gt; bit[0] must be 0.</p> <p>0 = Disable  1 = Non-extended: Non-extended (4 byte) DSA tag  2 = Extended: Extended (8 byte) DSA tag  3 = Reserved</p>
13:12	SPID[5:4]	RW 0x0	MSB bits of Source Port ID (for the case of a switch that supports more than 16 ports)
14	SDIDEn	RW 0x0	<p>Source Device ID enable</p> <p>0 = Ignore: Ignore DSA tag Source Port ID field in queuing decision  1 = Not Ignore: Take into account DSA tag Source Port ID field in queuing decision</p>
15	Reserved	RO 0x0	Reserved
20:16	SDID	RW 0x0	Source Device ID setting to match the switch SrcDev reported in DSA tag. Used for queuing decision
31:21	Reserved	RO 0x0	Reserved

**Table 433: Port Interrupt Cause (IC) Register**  
Offset: Port0: 0x72460 Port1: 0x76460

Bit	Field	Type/InitVal	Description
0	RxBuffer	RW0C 0x0	<p>Rx Buffer Return  Indicates a Rx buffer returned to CPU ownership or that the port finished reception of a Rx frame in either priority queues.</p> <p><b>NOTE:</b> To obtain a Rx Buffer return per priority queue, use bits[9:2]. These bits are set upon closing any Rx descriptor that has its &lt;EI&gt; bit set. To limit the interrupts to frame (rather than buffer) boundaries, the user should set the SDCR &lt;RIFB&gt; field in the SDMA Configuration (SDC) Register. When &lt;RIFB&gt; is set, an interrupt will be generated only upon closing the first descriptor of a received packet if this descriptor has its &lt;EI&gt; bit set.</p>
1	Extend	RW0C 0x0	Interrupt Cause Extend register (ICERx) of this port has a bit set.
2	RxBufferQueue[0]	RW0C 0x0	Rx Buffer Return in Priority Queue[0] indicates a Rx buffer returned to CPU ownership, or that the port completed reception of a Rx frame in receive priority queue[0].



**Table 433: Port Interrupt Cause (IC) Register (Continued)**  
Offset: Port0: 0x72460 Port1: 0x76460

Bit	Field	Type/InitVal	Description
3	RxBufferQueue[1]	RW0C 0x0	Rx Buffer Return in Priority Queue[1] indicates a Rx buffer returned to CPU ownership, or that the port completed reception of a Rx frame in receive priority queue[1].
4	RxBufferQueue[2]	RW0C 0x0	Rx Buffer Return in Priority Queue[2] indicates a Rx buffer returned to CPU ownership, or that the port completed reception of a Rx frame in receive priority queue[2].
5	RxBufferQueue[3]	RW0C 0x0	Rx Buffer Return in Priority Queue[3] indicates a Rx buffer returned to CPU ownership, or that the port completed reception of a Rx frame in receive priority queue[3].
6	RxBufferQueue[4]	RW0C 0x0	Rx Buffer Return in Priority Queue[4] indicates a Rx buffer returned to CPU ownership, or that the port completed reception of a Rx frame in receive priority queue[4].
7	RxBufferQueue[5]	RW0C 0x0	Rx Buffer Return in Priority Queue[5] indicates a Rx buffer returned to CPU ownership, or that the port completed reception of a Rx frame in receive priority queue[5].
8	RxBufferQueue[6]	RW0C 0x0	Rx Buffer Return in Priority Queue[6] indicates a Rx buffer returned to CPU ownership, or that the port completed reception of a Rx frame in receive priority queue[6].
9	RxBufferQueue[7]	RW0C 0x0	Rx Buffer Return in Priority Queue[7] indicates a Rx buffer returned to CPU ownership, or that the port completed reception of a Rx frame in receive priority queue[7].
10	RxError	RW0C 0x0	Rx Resource Error indicates a Rx resource error event in either Rx priority queues (Rx Resource Error stands for Null descriptor pointer, or not owned descriptor, or parity error during descriptor fetch). To get a Rx Resource Error Indication per priority queue, use bits[18:11].
11	RxErrorQueue[0]	RW0C 0x0	Rx Resource Error in Priority Queue[0] indicates a Rx resource error event in receive priority queue[0].
12	RxErrorQueue[1]	RW0C 0x0	Rx Resource Error in Priority Queue[1] indicates a Rx resource error event in receive priority queue[1].
13	RxErrorQueue[2]	RW0C 0x0	Rx Resource Error in Priority Queue[2] indicates a Rx resource error event in receive priority queue[2].
14	RxErrorQueue[3]	RW0C 0x0	Rx Resource Error in Priority Queue[3] indicates a Rx resource error event in receive priority queue[3].

**Table 433: Port Interrupt Cause (IC) Register (Continued)**  
Offset: Port0: 0x72460 Port1: 0x76460

Bit	Field	Type/InitVal	Description
15	RxErrorQueue[4]	RW0C 0x0	Rx Resource Error in Priority Queue[0] indicates a Rx resource error event in receive priority queue[4].
16	RxErrorQueue[5]	RW0C 0x0	Rx Resource Error in Priority Queue[1] indicates a Rx resource error event in receive priority queue[5].
17	RxErrorQueue[6]	RW0C 0x0	Rx Resource Error in Priority Queue[2] indicates a Rx resource error event in receive priority queue[6].
18	RxErrorQueue[7]	RW0C 0x0	Rx Resource Error in Priority Queue[3] indicates a Rx resource error event in receive priority queue[7].
19	TxEnd0	RW 0x0	Tx End for Priority Queue 0 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
20	TxEnd1	RW 0x0	Tx End for Priority Queue 1 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
21	TxEnd2	RW 0x0	Tx End for Priority Queue 2 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
22	TxEnd3	RW 0x0	Tx End for Priority Queue 3 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
23	TxEnd4	RW 0x0	Tx End for Priority Queue 4 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
24	TxEnd5	RW 0x0	Tx End for Priority Queue 5 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
25	TxEnd6	RW 0x0	Tx End for Priority Queue 6 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
26	TxEnd7	RW 0x0	Tx End for Priority Queue 7 indicates that the Tx DMA stopped processing the queue after a stop command (DISQ), or that it reached the end of the descriptor chain (through a null pointer or not owned descriptor).
30:27	Reserved	RSVD 0x0	Reserved



**Table 433: Port Interrupt Cause (IC) Register (Continued)**  
Offset: Port0: 0x72460 Port1: 0x76460

Bit	Field	Type/InitVal	Description
31	EtherIntSum	RO 0x0	Ethernet Interrupt Summary This bit is a logical OR of the (unmasked) bits[30:0] in the Interrupt Cause register of the port.

**Table 434: Port Interrupt Cause Extend (ICE) Register**  
Offset: Port0: 0x72464 Port1: 0x76464

Bit	Field	Type/InitVal	Description
0	TxBuffer[0]	RW 0x0	Tx Buffer for Queue 0 Indicates a Tx buffer returned to CPU ownership, or that the port finished transmission of a Tx frame. <b>NOTE:</b> This bit is set upon closing any Tx descriptor which has its EI bit set. To limit the interrupts to frame (rather than buffer) boundaries, the user should set EI only in the last descriptor.
1	TxBuffer[1]	RW 0x0	Tx Buffer for Queue 1 Indicates a Tx buffer returned to CPU ownership, or that the port finished transmission of a Tx frame.
2	TxBuffer[2]	RW 0x0	Tx Buffer for Queue 2 Indicates a Tx buffer returned to CPU ownership, or that the port finished transmission of a Tx frame.
3	TxBuffer[3]	RW 0x0	Tx Buffer for Queue 3 Indicates a Tx buffer returned to CPU ownership, or that the port finished transmission of a Tx frame.
4	TxBuffer[4]	RW 0x0	Tx Buffer for Queue 4 Indicates a Tx buffer returned to CPU ownership, or that the port finished transmission of a Tx frame.
5	TxBuffer[5]	RW 0x0	Tx Buffer for Queue 5 Indicates a Tx buffer returned to CPU ownership, or that the port finished transmission of a Tx frame.
6	TxBuffer[6]	RW 0x0	Tx Buffer for Queue 6 Indicates a Tx buffer returned to CPU ownership, or that the port finished transmission of a Tx frame.
7	TxBuffer[7]	RW 0x0	Tx Buffer for Queue 7 Indicates a Tx buffer returned to CPU ownership, or that the port finished transmission of a Tx frame.
8	TxError0	RW 0x0	Tx Resource Error for Priority Queue 0 Indicates a Tx resource error event during packet transmission from the queue.

**Table 434: Port Interrupt Cause Extend (ICE) Register (Continued)**  
Offset: Port0: 0x72464 Port1: 0x76464

Bit	Field	Type/InitVal	Description
9	TxError1	RW 0x0	Tx Resource Error for Priority Queue 1 Indicates a Tx resource error event during packet transmission from the queue.
10	TxError2	RW 0x0	Tx Resource Error for Priority Queue 2 Indicates a Tx resource error event during packet transmission from the queue.
11	TxError3	RW 0x0	Tx Resource Error for Priority Queue 3 Indicates a Tx resource error event during packet transmission from the queue.
12	TxError4	RW 0x0	Tx Resource Error for Priority Queue 4 Indicates a Tx resource error event during packet transmission from the queue.
13	TxError5	RW 0x0	Tx Resource Error for Priority Queue 5 Indicates a Tx resource error event during packet transmission from the queue.
14	TxError6	RW 0x0	Tx Resource Error for Priority Queue 6 Indicates a Tx resource error event during packet transmission from the queue.
15	TxError7	RW 0x0	Tx Resource Error for Priority Queue 7 Indicates a Tx resource error event during packet transmission from the queue.
16	PhySTC	RW0C 0x0	PHY Status Change Indicates a status change reported by the PHY connected to this port. If there is any change in the link, speed, duplex mode, or flow control capability as it is detected by the MDIO interface with the PHY, this interrupt will be set. This interrupt is set regardless of the actual changes in the status register, since Auto-Negotiation might be set to disabled on some of the parameters.
17	PTP	RW0C 0x0	PTP This bit indicates the PTP packet identification.
18	RxOVR	RW0C 0x0	Rx Overrun Indicates an overrun event that occurred during reception of a packet.
19	TxDrd	RW0C 0x0	Tx Underrun Indicates an underrun event that occurred during transmission of packet from either queue.
20	LinkChange	RW0C 0x0	Link State Change This bit is set by upon a change in the link state (down->up, up->down).
22:21	Reserved	RW0C 0x0	Reserved.

**Table 434: Port Interrupt Cause Extend (ICE) Register (Continued)**  
Offset: Port0: 0x72464 Port1: 0x76464

Bit	Field	Type/InitVal	Description
23	InternalAddr Error	RW0C 0x0	Internal Address Error is set when there is an access to an illegal offset of the internal registers. When set, the Internal Address Error register locks the address that caused the error.
24	Reserved	RSVD 0x0	Reserved
25	PRBSError	ROC 0x0	Asserted when PRBS checker is enabled and an error was detected
30:26	Reserved	RO 0x0	Reserved.
31	EtherIntSum	RO 0x0	Ethernet Interrupt Extend Summary This bit is a logical OR of the (unmasked) bits[25:0] in this register.

**Table 435: Port Interrupt Mask (PIM) Register**  
Offset: Port0: 0x72468 Port1: 0x76468

Bit	Field	Type/InitVal	Description
31:0	Various	RW 0x0	Mask bits for Interrupt Cause register 0 = Mask 1 = Not mask

**Table 436: Port Extend Interrupt Mask (PEIM) Register**  
Offset: Port0: 0x7246C Port1: 0x7646C

Bit	Field	Type/InitVal	Description
31:0	Various	RW 0x0	Mask bits for Port Extend Interrupt Cause register 0 = Mask 1 = Not Mask

**Table 437: Port Tx FIFO Urgent Threshold (PxTFUT) Register**  
Offset: Port0: 0x72474 Port1: 0x76474

Bit	Field	Type/InitVal	Description
3:0	Reserved	RO 0x0	Reserved.

**Table 437: Port Tx FIFO Urgent Threshold (PxTFUT) Register (Continued)**  
Offset: Port0: 0x72474 Port1: 0x76474

Bit	Field	Type/InitVal	Description
19:4	IPGIntTx	RW 0x0	<p>Tx frame IPG between interrupt counter and enable.</p> <p>This field provides a way to force a delay from the last Port Interrupt Cause Extend (ICE) TxBuffer interrupt (Table 402 on page 463), from any of the queues, to the next TxBuffer interrupt, from any of the queues.</p> <p>The ICE bits still reflect the new interrupt, but this masking is reflected by potentially not propagating to the chip main interrupt cause register. This provides a way for interrupt coalescing on receive packet events.</p> <p>The time is calculated in multiples of 64 clock cycles.</p> <p>Valid values are 0 (no delay between packets to CPU, the counter is effectively disabled), through 0x3FFF (1,048,544 clock cycles).</p> <p>See the detailed description in Section 8.7.1 "Interrupt Coalescing".</p>
31:20	Reserved	RO 0x0	Read Only.

**Table 438: Port Rx Minimal Frame Size (PxMFS) Register**  
Offset: Port0: 0x7247C Port1: 0x7647C

Bit	Field	Type/InitVal	Description
1:0	RxMFS[1:0]	RO 0x0	Read Only
6:2	RxMFS[6:2]	RW 0x10	<p>Contains the Receive Minimal Frame Size in bytes.</p> <p>Valid Range of &lt;RxMFS[6:2]&gt; is 0xA-0x10.</p> <p>(RxMFS = 40,44,48,52,56,60,64 bytes)</p> <p>(Corresponding to 64 bytes)</p>
31:7	Reserved	RO 0x0	Read Only

**Table 439: Port Rx Discard Frame Counter (PxDFC) Register**  
Offset: Port0: 0x72484 Port1: 0x76484

Bit	Field	Type/InitVal	Description
31:0	Rx Discard[31:0]	ROC 0x0	<p>Number of frames that were discarded because of address filtering or resource error</p> <p>This register is reset every time the CPU reads from it.</p>

**Table 440: Port Overrun Frame Counter (PxOFC) Register**  
Offset: Port0: 0x72488 Port1: 0x76488

Bit	Field	Type/InitVal	Description
31:0	Rx Overrun[31:0]	RO 0x0	<p>Number of frames that were received and overrun</p> <p>This register is reset every time the CPU reads from it.</p>



**Table 441: Port Internal Address Error (EUIAE) Register**  
Offset: Port0: 0x72494 Port1: 0x76494

Bit	Field	Type/InitVal	Description
8:0	InternalAddress	RO 0x0	Locks the relevant internal address bits (bit 12 and bits 9:2), if there is an address violation of unmapped access to the port registers. The Address is locked until the register is read.
31:9	Reserved	RO 0x0	Reserved

**Table 442: Ethernet Type Priority Register**  
Offset: Port0: 0x724BC Port1: 0x764BC

Bit	Field	Type/InitVal	Description
0	EtherTypePriEn	RW 0x0	EtherType Priority Enable 0 = Disable: No EtherType priority (equivalent to the current GbE port, formal transparent) 1 = Enable: EtherType priority is enabled.
1	EtherTypePriFrstEn	RW 0x0	EtherType Priority Enable and Queuing EtherType priority enable and queuing decision is upon whether field <EtherTypePriQ> is hit, regardless of other Ethernet packet parameters. If enabled, this bit overrides the <EtherTypePriEn> field. 0 = Disable: No EtherType priority (equivalent to the current GbE port, formal transparent) 1 = Enable: EtherType priority enabled, and queueing is determined only by the priority setting.
4:2	EtherTypePriQ	RW 0x0	EtherType Priority Queue Queue number for ARP responses, when enabled by <EtherTpyePriEn> = 0x1 or <EtherTypePriFrstEn> = 0x1.
20:5	EtherTyprPriVal	RW 0x0806	EtherType Priority Value EtherType value that is assigned priority when enabled by <EtherTypePriEn> = 0x1 or <EtherTypePriFrstEn> = 0x1.
21	ForceUnicstHit	RW 0x0	Force Unicast Hit
31:22	Reserved	RO 0x0	Reserved

**Table 443: Ethernet Current Receive Descriptor Pointers (CRDP) Register (n=0–7)**

Offset: Port0: Q0: 0x7260C, Q1: 0x7261C, Q2: 0x7262C, Q3: 0x7263C, Q4: 0x7264C, Q5: 0x7265C,

Q6: 0x7266C, Q7: 0x7267C

Port1: Q0: 0x7660C, Q1: 0x7661C, Q2: 0x7662C, Q3: 0x7663C, Q4: 0x7664C, Q5: 0x7665C,

Q6: 0x7666C, Q7: 0x7667C

Bit	Field	Type/InitVal	Description
31:0	RxCDP	RW 0x0	Receive Current Queue Descriptor Pointer

**Table 444: Receive Queue Command (RQC) Register**

Offset: Port0: 0x72680 Port1: 0x76680

Bit	Field	Type/InitVal	Description
7:0	ENQ	RW 0x0	<p>Enable Queue[7:0]  One bit per each queue. Writing these bits set to 1 enables the queue. The Receive DMA will fetch the first descriptor programmed to the RxCDP register for that queue and start the Receive process.  Writing 1 to ENQ bit resets the matching &lt;DISQ&gt; bit.  Writing 1 to ENQ bit of a DMA that is already in enable state, has no effect.  Writing 0 to ENQ bit has no effect.  When the receive DMA encounters a queue ended by a null terminated descriptor pointer or a descriptor with a parity error, the DMA will clear the ENQ bit for that queue. Thus reading these bits reports the active enable status for each queue.</p> <p><b>NOTE:</b> Reaching a CPU owned descriptor, a null terminated descriptor or a descriptor that is read with a parity error in the middle of a packet will result in closing the status of the packet with a resource error condition.</p> <p>Reaching a CPU-owned descriptor either in the middle or on start of new packet will not result in disabling the DMA, and DMA will continue to read the descriptor it is using, when a new packet arrives to this queue, until it gets ownership of it.</p> <p>For these cases - a unowned descriptor, a null terminated descriptor, or a parity error on descriptor - the DMA will assert the resource RxErrorQueue interrupt.</p>
15:8	DISQ	RW 0x0	<p>Disable Queue[7:0]  One bit per each queue. Writing these bits set to 1 disables the queue. The transmit DMA will stop the Receive process to this queue, on the next packet boundary.  Writing 1 to DISQ bit resets the matching ENQ bit after the RxDMA finished processing the queue, if the ENQ bit was used while the CPU wrote the DISQ for it.  Writing 0 to DISQ bit has no effect.  When transmit DMA encounters a queue ended either by a null terminated descriptor pointer or a descriptor with a parity error, the DMA will disable the queue but not set the DISQ bit for that queue, thus reading DISQ and ENQ bits discriminates between queues disabled by CPU and those stopped by DMA due to a null pointer or a parity error on descriptor.</p>
31:16	Reserved	RO 0x0	Read Only



**Table 445: Transmit Current Served Descriptor Pointer Register**  
Offset: Port0: 0x72684 Port1: 0x76684

Bit	Field	Type/InitVal	Description
31:0	TxCDP	RO 0x0	Transmit Current Descriptor Pointer

**Table 446: Transmit Current Queue Descriptor Pointer (TCQDP) Register (n=0–7)**  
Offset: Port0: Q0: 0x726C0, Q1: 0x726C4, Q2: 0x726C8, Q3: 0x726CC, Q4: 0x726D0, Q5: 0x726D4,  
Q6: 0x726D8, Q7: 0x726DC  
Port1: Q0: 0x766C0, Q1: 0x766C4, Q2: 0x766C8, Q3: 0x766CC, Q4: 0x766D0, Q5: 0x766D4,  
Q6: 0x766D8, Q7: 0x766DC

Bit	Field	Type/InitVal	Description
31:0	TxCDP	RW 0x0	Transmit Current Queue Descriptor Pointer

**Table 447: Destination Address Filter Special Multicast Table (DFSMT) Register (n=0–63)**  
Offset: Port0: Register0: 0x73400, Register1: 0x73404...Register63: 0x734FC  
Port1: Register0: 0x77400, Register1: 0x77404...Register63: 0x774FC  
Offset Formula: Port0: 0x73400+n\*4: where n (0-63) represents Register  
Port1: 0x77400+n\*4: where n (0-63) represents Register

Bit	Field	Type/InitVal	Description
NOTE: Every register holds four entries. A total of 64 registers appear in the table in consecutive order.			
0	Pass[0]	RW 0x0	Determines whether to filter or accept, for pointer index 0. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
3:1	Queue[0]	RW 0x0	For pointer index 0: Determines the Queue number if Pass[0]=1.
4	Reserved[0]	RW 0x0	Reserved Must be set to 0
7:5	Unused[0]	RO 0x0	Reserved
8	Pass[1]	RW 0x0	Determines whether to filter or accept, for pointer index 1. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
11:9	Queue[1]	RW 0x0	For pointer index 1: Determines the Queue number if Pass[1]=1.

**Table 447: Destination Address Filter Special Multicast Table (DFSMT) Register (n=0–63) (Continued)**

Offset: Port0: Register0: 0x73400, Register1: 0x73404...Register63: 0x734FC

Port1: Register0: 0x77400, Register1: 0x77404...Register63: 0x774FC

Offset Formula: Port0: 0x73400+n\*4: where n (0-63) represents Register

Port1: 0x77400+n\*4: where n (0-63) represents Register

Bit	Field	Type/InitVal	Description
12	Reserved[1]	RW 0x0	Reserved Must be set to 0
15:13	Unused[1]	RO 0x0	Reserved
16	Pass[2]	RW 0x0	Determines whether to filter or accept, for pointer index 2. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
19:17	Queue[2]	RW 0x0	For pointer index 2: Determines the Queue number if Pass[2]=1.
20	Reserved[2]	RW 0x0	Reserved Must be set to 0
23:21	Unused[2]	RO 0x0	Reserved
24	Pass[3]	RW 0x0	Determines whether to filter or accept, for pointer index 3. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
27:25	Queue[3]	RW 0x0	For pointer index 0: Determines the Queue number if Pass[3]=1.
28	Reserved[3]	RW 0x0	Reserved Must be set to 0
31:29	Unused[3]	RO 0x0	Reserved

**Table 448: Destination Address Filter Other Multicast Table (DFOMT) Register (n=0–63)**

Offset: Port0: Register0: 0x73500, Register1: 0x73504...Register63: 0x735FC

Port1: Register0: 0x77500, Register1: 0x77504...Register63: 0x775FC

Offset Formula: Port0: 0x73500+n\*4: where n (0-63) represents Register

Port1: 0x77500+n\*4: where n (0-63) represents Register

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> Every register holds four entries. A total of 64 registers appear in this table in consecutive order.			
0	Pass[0]	RW 0x0	Determines whether to filter or accept, for pointer index 0. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
3:1	Queue[0]	RW 0x0	For pointer index 0: Determines the Queue number if Pass[0]=1.



**Table 448: Destination Address Filter Other Multicast Table (DFOMT) Register (n=0–63) (Continued)**

Offset: Port0: Register0: 0x73500, Register1: 0x73504...Register63: 0x735FC

Port1: Register0: 0x77500, Register1: 0x77504...Register63: 0x775FC

Offset Formula: Port0: 0x73500+n\*4: where n (0-63) represents Register

Port1: 0x77500+n\*4: where n (0-63) represents Register

Bit	Field	Type/InitVal	Description
4	Reserved[0]	RW 0x0	Reserved Must be set to 0
7:5	Unused[0]	RO 0x0	Reserved
8	Pass[1]	RW 0x0	Determines whether to filter or accept, for pointer index 1. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
11:9	Queue[1]	RW 0x0	For pointer index 1: Determines the Queue number if Pass[1]=1.
12	Reserved[1]	RW 0x0	Reserved Must be set to 0
15:13	Unused[1]	RO 0x0	Reserved
16	Pass[2]	RW 0x0	Determines whether to filter or accept, for pointer index 2. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
19:17	Queue[2]	RW 0x0	For pointer index 2: Determines the Queue number if Pass[2]=1.
20	Reserved[2]	RW 0x0	Reserved Must be set to 0
23:21	Unused[2]	RO 0x0	Reserved
24	Pass[3]	RW 0x0	Determines whether to filter or accept, for pointer index 3. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
27:25	Queue[3]	RW 0x0	For pointer index 3: Determines the Queue number if Pass[3]=1.
28	Reserved[3]	RW 0x0	Reserved Must be set to 0
31:29	Unused[3]	RO 0x0	Reserved

**Table 449: Destination Address Filter Unicast Table (DFUT) Register (n=0–3)**

Offset: Port0: Register0: 0x73600, Register1: 0x73604, Register2: 0x73608, Register3: 0x7360C  
 Port1: Register0: 0x77600, Register1: 0x77604, Register2: 0x77608, Register3: 0x7760C

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> Every register holds four entries. A total of four registers appear in this table in consecutive order.			
0	Pass[0]	RW 0x0	Determines whether to filter or accept, for pointer index 0. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
3:1	Queue[0]	RW 0x0	For pointer index 0: Determines the Queue number if Pass[0]=1.
4	Reserved[0]	RW 0x0	Reserved Must be set to 0
7:5	Unused[0]	RO 0x0	Reserved
8	Pass[1]	RW 0x0	Determines whether to filter or accept, for pointer index 1. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
11:9	Queue[1]	RW 0x0	For pointer index 1: Determines the Queue number if Pass[1]=1.
12	Reserved[1]	RW 0x0	Reserved Must be set to 0
15:13	Unused[1]	RO 0x0	Reserved
16	Pass[2]	RW 0x0	Determines whether to filter or accept, for pointer index 2. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
19:17	Queue[2]	RW 0x0	For pointer index 2: Determines the Queue number if Pass[2]=1.
20	Reserved[2]	RW 0x0	Reserved Must be set to 0
23:21	Unused[2]	RO 0x0	Reserved
24	Pass[3]	RW 0x0	Determines whether to filter or accept, for pointer index 3. 0 = Reject: Reject (filter) frame 1 = Accept: Accept frame
27:25	Queue[3]	RW 0x0	For pointer index 3: Determines the Queue number if Pass[3]=1.



**Table 449: Destination Address Filter Unicast Table (DFUT) Register (n=0–3) (Continued)**

Offset: Port0: Register0: 0x73600, Register1: 0x73604, Register2: 0x73608, Register3: 0x7360C  
Port1: Register0: 0x77600, Register1: 0x77604, Register2: 0x77608, Register3: 0x7760C

Bit	Field	Type/InitVal	Description
28	Reserved[3]	RW 0x0	Reserved Must be set to 0
31:29	Unused[3]	RO 0x0	Reserved

### A.8.2.1 Tx Queues Arbiter

**Table 450: Transmit Queue Command (TQC) Register**

Offset: Port0: 0x72448 Port1: 0x76448

Bit	Field	Type/InitVal	Description
7:0	ENQ	RW 0x0	<p>Enable Queue One bit per queue. Writing these bits set to 1 enables the queue. The transmit DMA will fetch the first descriptor programmed to the FDP register for that queue and start the transmit process. Writing 1 to the &lt;ENQ&gt; bit resets the matching &lt;DISQ&gt; bit. Writing 1 to the &lt;ENQ&gt; bit of a DMA that is already in enable state, has no effect. Writing 0 to the &lt;ENQ&gt; bit has no effect. When transmit DMA encounters queue end, either by a null terminated descriptor pointer or a descriptor with a parity error or a CPU owned descriptor, the DMA will clear the &lt;ENQ&gt; bit for that queue. Thus, reading these bits reports the active enable status for each queue.</p> <p><b>NOTE:</b> The &lt;ENQ&gt; bits are cleared on link down. After link is up, the CPU has to restart the DMA (to set the &lt;ENQ&gt; bits).</p>
15:8	DISQ	RW 0x0	<p>Disable Queue One bit per each queue. Writing 1 disables the queue. The transmit DMA will stop the transmit process from this queue on the next packet boundary. Writing 1 to the &lt;DISQ&gt; bit resets the matching &lt;ENQ&gt; bit (when the DMA is finished with the queue and if the current active queue has been disabled). Writing 0 to the &lt;DISQ&gt; bit has no effect. When transmit DMA encounters queue end, either by a null terminated descriptor pointer or by a CPU owned descriptor or by a descriptor with a parity error, the DMA will disable the queue but not set the &lt;DISQ&gt; bit for the queue. Thus reading &lt;DISQ&gt; and &lt;ENQ&gt; bits discriminates between queues disabled by the CPU and those stopped by the DMA due to a null pointer or a CPU owned descriptor, or a parity error on descriptor.</p> <p><b>NOTE:</b> The &lt;DISQ&gt; bits are cleared on link down.</p>
31:16	Reserved	RO 0x0	Reserved

**Table 451: Transmit Queue Fixed Priority Configuration (TQFPC) Register**  
Offset: Port0: 0x724DC Port1: 0x764DC

Bit	Field	Type/InitVal	Description
7:0	FIXPR	RW 0xFF	Fixed Priority Queue [7:0] Setting this bit to 1 configures the transmit queue as fixed priority. A 0 value means the queue is subject to the WRR algorithm, see "Fixed Priority Mode".
31:8	Reserved	RO 0x0	Reserved

**Table 452: Port Transmit Token-Bucket Rate Configuration (PTTBRC) Register**  
Offset: Port0: 0x724E0 Port1: 0x764E0

Bit	Field	Type/InitVal	Description
14:0	PTKNRT	RW 0x7FFF	Port Token Rate Configurable value 0-32767, in units of 1/64 byte, is added to Token-Bucket of the queues, as selected by the <PTP_SYNC_ENB> and <EJP_ENB> fields, either every 8 system clock cycles or every clock cycle generated by the PTP module. The Port Token Bucket is used to limit the port transmit bandwidth, see "Weighted Round-Robin Priority Mode" and "Transmit Queue Bandwidth Limitation". <b>NOTE:</b> The initial value means that there is no bandwidth limitation on the port.
31:15	Reserved	RO 0x0	Reserved

**Table 453: Transmit Queue Command1 (TQC1) Register**  
Offset: Port0: 0x724E4 Port1: 0x764E4

Bit	Field	Type/InitVal	Description
0	WRR_EJP_INIT	RW 0x1	When set to 1, this bit acts as a software reset to the module. It initializes the WRR/EJP module: - The WRR_BYT_COUNT variable of all queues are cleared to 0. - The TIME_FROM_LAST_PKT_TX variable is initialized to 0xFFFF. - Queue token bucket update is disabled. The token buckets should be initialized in this state.
1	PTP_SYNC_ENB	RW 0x0	When the EJP_ENB bit is set to 1, the PTP_SYNC_ENB bit defines the event for queue and port token buckets update. 0 = SystemClock: Every 8 system clock cycles. 1 = PTP: Every rising edge of the clock signal generated by the PTP module.
2	EJP_ENB	RW 0x0	This bit enables the Egress Jitter Pacing (EJP) mechanism. 0 = Disable: Use fixed priority/WRR 1 = Enable: EJP



**Table 453: Transmit Queue Command1 (TQC1) Register (Continued)**  
Offset: Port0: 0x724E4 Port1: 0x764E4

Bit	Field	Type/InitVal	Description
3	Reserved	RW 0x1	To this field must be written "0".
31:4	Reserved	RO 0x0	Reserved

**Table 454: Port Maximum Transmit Unit (PMTU) Register**  
Offset: Port0: 0x724E8 Port1: 0x764E8

Bit	Field	Type/InitVal	Description
5:0	PMTU	RW 0x24	Port Maximum Transmit Unit 9 KB/256 = 36 (=0x24) Configurable value in 256-byte units, up to 16 KB, used to implement the port Token-Bucket bandwidth limitation. If PTKNBKT is greater than MTU, the port bandwidth limitation is OFF (transmit enabled).
31:6	Reserved	RO 0x0	Reserved

**Table 455: Port Maximum Token Bucket Size (PMTBS) Register**  
Offset: Port0: 0x724EC Port1: 0x764EC

Bit	Field	Type/InitVal	Description
15:0	PMTBS	RW 0xFFFF	Port Maximum Token Bucket Size Configurable value in 256-byte units, used to implement the port Token-Bucket bandwidth limitation. The PTKNBKT is incremented by PTKNRT up to PMTBS*256*64. (Maximum accumulated transmit credit.)
31:16	Reserved	RO 0x0	Reserved

**Table 456: Queue Transmit Token-Bucket Counter (QxTTBC) Register (n=0–7)**

Offset: Port0: Q0: 0x72700, Q1: 0x72710, Q2: 0x72720, Q3: 0x72730, Q4: 0x72740, Q5: 0x72750,  
 Q6: 0x72760, Q7: 0x72770  
 Port1: Q0: 0x76700, Q1: 0x76710, Q2: 0x76720, Q3: 0x76730, Q4: 0x76740, Q5: 0x76750,  
 Q6: 0x76760, Q7: 0x76770

Bit	Field	Type/InitVal	Description
30:0	QTKNBKT	RW 0x0	<p>Queue Token Bucket</p> <p>Byte counter, in units of 1/64 byte, incremented by the &lt;QTKNRT&gt; value and decremented by transmitted bytes x 64 from this queue.</p> <p>The Byte counter is incremented, as selected by the &lt;PTP_SYNC_ENB&gt; and &lt;EJP_ENB&gt; fields, either every 8 system clock cycles or every cycle of the clock signal generated by the PTP module.</p> <p>This field is in two's complement format. That means that the Queue Token bucket may contain a negative number of tokens.</p> <p>See "Transmit Queues Egress Jitter Pacing (EJP) Arbitration".</p> <p><b>NOTE:</b> The CPU may write to this counter to initialize or update the Queue Token-bucket. It is recommended that the CPU write to the token bucket, only when the &lt;WRR_EJP_INIT&gt; field is set.</p>
31	Reserved	RO 0x0	Reserved Read only

**Table 457: Transmit Queue Token Bucket Configuration (TQxTBC) Register (n=0–7)**

Offset: Port0: Q0: 0x72704, Q1: 0x72714, Q2: 0x72724, Q3: 0x72734, Q4: 0x72744, Q5: 0x72754,  
 Q6: 0x72764, Q7: 0x72774  
 Port1: Q0: 0x76704, Q1: 0x76714, Q2: 0x76724, Q3: 0x76734, Q4: 0x76744, Q5: 0x76754,  
 Q6: 0x76764, Q7: 0x76774

Bit	Field	Type/InitVal	Description
14:0	QTKNRT	RW 0x7FFF	<p>Queue Token Rate</p> <p>Legal values are between 0-32767. The Token Rate, in units of 1/64 byte, is added to the Queue Token Bucket, as selected by the &lt;PTP_SYNC_ENB&gt; and &lt;EJP_ENB&gt; fields, either every 8 system clock cycles or every cycle of the clock signal generated by the PTP module.</p> <p>The Queue Token Bucket is used to limit the transmit bandwidth per queue (see "Weighted Round-Robin Priority Mode" and "Transmit Queue Bandwidth Limitation").</p> <p><b>NOTE:</b> The initial value actually means that there is no bandwidth limitation on the queue.</p>
15	Reserved	RSVD 0x0	Reserved



**Table 457: Transmit Queue Token Bucket Configuration (TQxTBC) Register (n=0–7) (Continued)**

Offset: Port0: Q0: 0x72704, Q1: 0x72714, Q2: 0x72724, Q3: 0x72734, Q4: 0x72744, Q5: 0x72754,

Q6: 0x72764, Q7: 0x72774

Port1: Q0: 0x76704, Q1: 0x76714, Q2: 0x76724, Q3: 0x76734, Q4: 0x76744, Q5: 0x76754,

Q6: 0x76764, Q7: 0x76774

Bit	Field	Type/InitVal	Description
31:16	QMTBS	RW 0xFFFF	Queue Maximum Token Bucket Size Configurable value, in 256-byte units, used to implement the Token-Bucket bandwidth limitation. The Queue Token Bucket is incremented by the <QTKNRT> field value up to QMTBS*256*64. (Maximum accumulated transmit credit.)

**Table 458: Transmit Queue Arbiter Configuration (TQxAC) Register (n=0–7)**

Offset: Port0: Q0: 0x72708, Q1: 0x72718, Q2: 0x72728, Q3: 0x72738, Q4: 0x72748, Q5: 0x72758,

Q6: 0x72768, Q7: 0x72778

Port1: Q0: 0x76708, Q1: 0x76718, Q2: 0x76728, Q3: 0x76738, Q4: 0x76748, Q5: 0x76758,

Q6: 0x76768, Q7: 0x76778

Bit	Field	Type/InitVal	Description
7:0	WRRWGT	RW 0x0	Transmit Weighted-Round-Robin Weight The configurable value of WRR weight is 256 bytes per unit, spans 0-64 KB, in 256-bytes increments. The transmit bandwidth assigned to a queue is equal to WRRWGT/(Sum of all "non fixed priority" plus "maximum limited" WRRWGTs) part of the remaining bandwidth (not consumed by the fixed priority). See "Transmit Weighted Round-Robin Arbitration"
25:8	WRR_BC[17:0]	RO 0x0	Internal counter for implementing the WRR algorithm. The data format is as following: Bits[15:0]: Byte_count as percentage of <WRRWGT>. Bits[17:16]: Integer (byte_count / <WRRWGT>). This field is initialized to 0, when the <EJP_INIT> is set to 1.
31:26	QMTU	RW 0x0	Queue Maximum Transmit Unit Configurable value in 256-byte units, up to 16 Kbyte, used to implement the Queue Token-Bucket bandwidth limitation. If QTKNBKT is greater than QMTU, then the queue bandwidth limitation is OFF (transmit enabled).

**Table 459: Port Transmit Token-Bucket Counter (PTTBC) Register**  
Offset: Port0: 0x72780 Port1: 0x76780

Bit	Field	Type/InitVal	Description
29:0	PTKNBKT	RW 0x0	Port Token Bucket Byte counter, in units of 1/64 byte, incremented by PTKNRT (up to PMTBS*256*64) and decremented by transmitted bytes x 64 from this port. The byte counter is incremented by the <PTKNRT> value, as selected by the <PTP_SYNC_ENB> and <EJP_ENB> fields, either every 8 system clock cycles or every cycle of the clock signal generated by the PTP module. The CPU may write to this counter to override with Token-Bucket operation.
31:30	Reserved	RO 0x0	Reserved Read only

**Table 460: Transmission Queue IPG (TQxIPG) Register (n=2–3)**  
Offset: Port0: Q2: 0x727A8, Q3: 0x727B8  
Port1: Q2: 0x767A8, Q3: 0x767B8

Bit	Field	Type/InitVal	Description
13:0	IPG	RW 0x0	IPG in units of system-clock cycles. This field is used for transmit queue selection in EJP mode, for queues 2 and 3.
31:14	Reserved	RO 0x0	Reserved Read only

**Table 461: High Token in Low Packet (HiTKNinLOPKT) Register**  
Offset: Port0: 0x727C0 Port1: 0x767C0

Bit	Field	Type/InitVal	Description
19:0	HiTKNinLoPkt	RW 0x0	Number of tokens that are accumulated in the IsoHi (queue#3) token bucket, during transmission of the longest IsoLo (queue#2) packet. This field is used for transmit queue selection in EJP mode.
31:20	Reserved	RO 0x0	Reserved

**Table 462: High Token in Asynchronous Packet (HiTKNinASYNCPKT) Register**  
Offset: Port0: 0x727C4 Port1: 0x767C4

Bit	Field	Type/InitVal	Description
19:0	HiTKNinAsyncPkt	RW 0x0	Number of tokens that are accumulated in the IsoHi (queue#3) token bucket, during transmission of the longest asynchronous (queues#1 and #0) packet. This field is used for transmit queue selection in EJP mode.



**Table 462: High Token in Asynchronous Packet (HITKNinASYNCPKT) Register (Continued)**  
Offset: Port0: 0x727C4 Port1: 0x767C4

Bit	Field	Type/InitVal	Description
31:20	Reserved	RO 0x0	Reserved

**Table 463: Low Token in Asynchronous Packet (LOTKNinASYNCPKT) Register**  
Offset: Port0: 0x727C8 Port1: 0x767C8

Bit	Field	Type/InitVal	Description
19:0	LoTKNinAsyncPkt	RW 0x0	Number of tokens that are accumulated in the IsoLo (queue#2) token bucket, during transmission of the longest asynchronous (queues#1 and #0) packet. This field is used for transmit queue selection in EJP mode.
31:20	Reserved	RO 0x0	Reserved

**Table 464: Transmission Speed (TS) Register**  
Offset: Port0: 0x727D0 Port1: 0x767D0

Bit	Field	Type/InitVal	Description
14:0	TS	RW 0x0	Ethernet Transmission Speed in units of system clocks/1024 bytes This field is used in EJP mode for packet duration calculation.
31:15	Reserved	RO 0x0	Reserved

### A.8.3

## Precise Time Protocol (PTP) Registers

The following are the register bits used for configuration and status information to and from the software/CPU sub-system for Precise Time Protocol logic for audio-video bridging (AVB) applications.



#### Note

- These registers are accessed using the two Global registers:
  - PTP Command Register (0x7C000)
  - PTP Data Register (0x7C008).
- For the registers in Table 466, Table 467, Table 470–Table 478, Table 480, Table 481, Table 483, Table 484, Table 486–Table 496, bits [31:16] are not implemented and will return undefined values.

**Table 465: Register Map Table for the PTP Registers**

Register Name	Offset	Table and Page
<b>PTP Global Core Access Registers</b>		
PTP Command Register	0x7C000	<a href="#">Table 466, p. 598</a>
PTP Data Register	0x7C008	<a href="#">Table 467, p. 599</a>
<b>PTP Configuration Registers</b>		
PTP Reset Register	0x7C010	<a href="#">Table 468, p. 599</a>
PTP Clock Select Register	0x7C018	<a href="#">Table 469, p. 600</a>
<b>PTP Global Configuration Registers</b>		
PTP Global Configuration0 Register	0x0, or Decimal 0	<a href="#">Table 470, p. 601</a>
PTP Global Configuration1 Register	0x1, or Decimal 1	<a href="#">Table 471, p. 601</a>
PTP Global Configuration2 Register	0x2, or Decimal 2	<a href="#">Table 472, p. 602</a>
PTP Global Configuration3 Register	0x3, or Decimal 3	<a href="#">Table 473, p. 602</a>
<b>PTP Global Status Data Registers</b>		
PTP Global Status0 Register	0x8, or Decimal 8	<a href="#">Table 474, p. 603</a>
<b>PTP Port Configuration Data Registers</b>		
PTP Port Configuration0 Register	0x0, or Decimal 0	<a href="#">Table 475, p. 604</a>
PTP Port Configuration1 Register	0x1, or Decimal 1	<a href="#">Table 476, p. 604</a>
PTP Port Configuration2 Register	0x 2, or Decimal 2	<a href="#">Table 477, p. 605</a>
<b>PTP Port Status Registers</b>		
PTP Port Status0 Register	0x8, or Decimal 8	<a href="#">Table 478, p. 608</a>
PTP Port Status1 Register	0x9 and 0xA, or Decimal 9 and Decimal 10	<a href="#">Table 479, p. 609</a>
PTP Port Status2 Register	0xB, or Decimal 11	<a href="#">Table 480, p. 609</a>
PTP Port Status3 Register	0xC, or Decimal 12	<a href="#">Table 481, p. 609</a>
PTP Port Status4 Register	0x D and 0xE, or Decimal 13 and 14	<a href="#">Table 482, p. 611</a>
PTP Port Status5 Register	0xF, or Decimal 15	<a href="#">Table 483, p. 611</a>
PTP Port Status6 Register	0x10, or Decimal 16	<a href="#">Table 484, p. 611</a>
PTP Port Status7 Register	0x11 and 0x12, or Decimal 17 and 18	<a href="#">Table 485, p. 613</a>
PTP Port Status8 Register	0x13, or Decimal 19	<a href="#">Table 486, p. 613</a>

**Table 465: Register Map Table for the PTP Registers (Continued)**

Register Name	Offset	Table and Page
PTP Port Status9 Register	0x15, or Decimal 21	<a href="#">Table 487, p. 613</a>
<b>TAI Global Configuration Data Registers</b>		
TAI Global Configuration, PTP Port = 0xE	0x0, or Decimal 0	<a href="#">Table 488, p. 615</a>
TAI Global Configuration Register, PTP Port = 0xE and 0xF	0x1, or Decimal 1	<a href="#">Table 489, p. 617</a>
TAI Global Configuration0 Register	0x2 and 0x3, or Decimal 2 and 3	<a href="#">Table 490, p. 618</a>
TAI Global Configuration1 Register	0x4, or Decimal 4	<a href="#">Table 491, p. 618</a>
TAI Global Configuration2 Register	0x5, or Decimal 5	<a href="#">Table 492, p. 618</a>
TAI Global Configuration3 Register	0x6, or Decimal 6	<a href="#">Table 493, p. 619</a>
TAI Global Configuration4 Register	0x7, or Decimal 7	<a href="#">Table 494, p. 619</a>
<b>TAI Global Status Registers</b>		
TAI Global Status1 Register	0x8, or Decimal 8	<a href="#">Table 495, p. 620</a>
TAI Global Status1 Register	0x9, or Decimal 9	<a href="#">Table 496, p. 621</a>
TAI Global Status2 Register	0xA and 0xB, or Decimal 10 and 11	<a href="#">Table 497, p. 621</a>
PTP Global Status1 Register	0xE and 0xF, or Decimal 14 and 15	<a href="#">Table 498, p. 622</a>

### A.8.3.1 PTP Core Access Registers

**Table 466: PTP Command Register**  
Offset: 0x7C000

Bits	Field	Type	Description
15	PTPBusy	SC 0x0	PTP Unit Busy. This bit must be set to 0x1 to start an PTP operation (see PTPOp below). Only one PTP operation can be executing at a time, so this bit must be 0x0 before setting it to a 0x1. When the requested PTP operation completes, this bit will automatically be cleared to a 0x0.
14:12	PTPOp	RWR 0x0	PTP Unit Operation Code. The devices support the following PTP operations (all of these operations can be executed while frames are transiting through the device):  000 = No Operation 001 = Reserved 010 = Reserved 011 = Write to the register pointed by PTPAddr below. PTPData registers content gets written into the selected register. 100 = Read from the register pointed to by PTPAddr below. The data read from the selected register is transferred to PTPData register. 101 = Reserved 110 = Read with post increment of register address defined in bits PTPAddr bits below. For PTP data structures, this command instructs the hardware to take a snap-shot of four consecutive data registers, starting with the PTPAddr location. This is used for capturing time counter values that are more than 16-bits wide, along with the sequence identifier information. 111 = Reserved

**Table 466: PTP Command Register (Continued)**  
Offset: 0x7C000

Bits	Field	Type	Description
11:8	PTPPort	RWR 0x0	<p>This indicates the physical port of this device. These bits indicate the PTP port that is being accessed in the PTP Command register.</p> <p>For example, if this field is programmed to a 0x1, it indicates that PTP registers belonging to port number 1 are being accessed.</p> <p>To access PTP global registers, set the PTPBlock to 0x0 and set PTPPort to 0xF.</p> <p>To access Time Application Interface (TAI) Global registers, set the PTPBlock to 0x0 and set PTPPort to 0xE.</p> <p>To access PTP Policy global registers, set the PTPBlock to 0x1 and set PTPPort to 0xF.</p> <p>To access QAV global registers, set the PTPBlock to 0x2 and set PTPPort to 0xF.</p>
7:5	PTPBlock	RWR 0x0	<p>This field indicates the block of addresses within the device. For example within the PTP register space, this field selects the block like PTP, SRP, and QAV.</p> <p>The PTPAddr field below selects the specific address within a selected block.</p> <p>0x0 = To select PTP register space  0x1 = To select PTP Policy register space  0x2 = To select QAV register space  0x3–0x7 = Reserved for future use.</p> <p><b>NOTE:</b> Accessing registers in the reserved range of the PTPBlock will return all zeroes back for the PTP Command register.</p>
4:0	PTPAddr	RWR 0x0	These bits indicate the address bits for the register operation being specified in the PTPOp bits specified above.

**Table 467: PTP Data Register**  
Offset: 0x7C008

Bits	Field	Type	Description
15:0	PTPData	RWR 0x0	<p>PTP Data bits</p> <p>These data bits indicate either the read data or the write data bits, depending on the PTP Command register.</p> <p>For a read operation, the hardware logic fetches the data bits from the specified address in PTP Command register and stores them into these bits.</p> <p>For a write operation, the hardware logic utilizes the data bits in this field to write to the specified address location in PTP Command register.</p>

### A.8.3.2 PTP Configuration Registers

**Table 468: PTP Reset Register**  
Offset: 0x7C010

Bits	Field	Type	Description
31:1	Reserved	RSVD 0x0	Reserved

**Table 468: PTP Reset Register (Continued)**  
Offset: 0x7C010

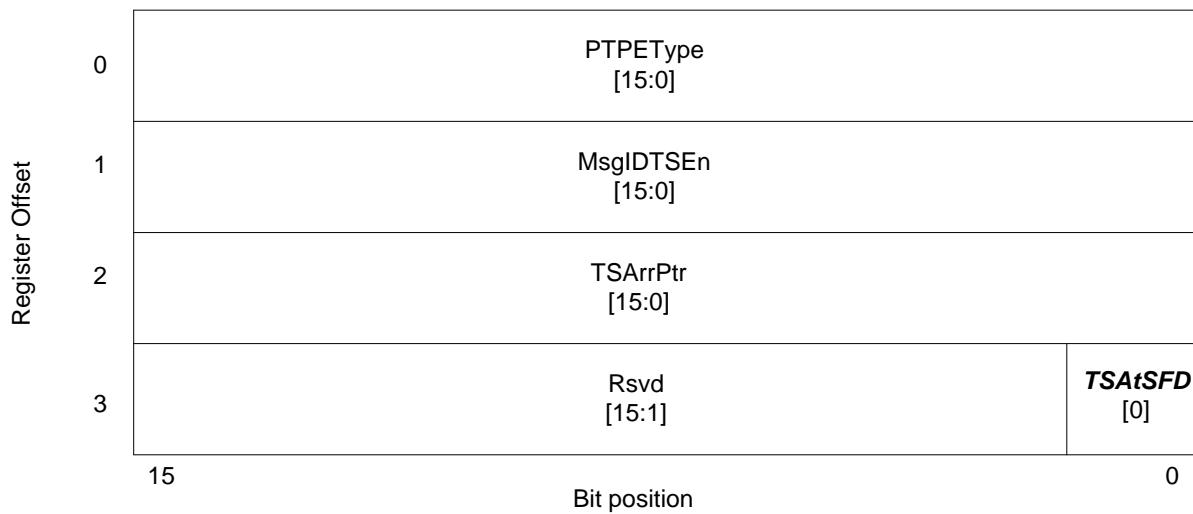
Bits	Field	Type	Description
0	PTPReset	RW 0x0	PTP Reset 0 = PTP is reset. 1 = PTP is enabled.

**Table 469: PTP Clock Select Register**  
Offset: 0x7C018

Bits	Field	Type	Description
31:1	Reserved	RSVD 0x0	Reserved
0	PTPClkSelect	RW 0x0	PTP Clock Select 0 = An external 125 MHz clock is used as the PTP module clock. 1 = An external PTP clock is used as the PTP module clock.

### A.8.3.3 PTP Global Configuration Registers

**Figure 89: PTP Configuration Data Structure Registers**



**Table 470: PTP Global Configuration0 Register**  
Offset: 0x0, or Decimal 0

Bits	Field	Type	Description
15:0	PTPType	RWR 0x0	<p>Precise Time Protocol EtherType.</p> <p>All PTP frames are recognized using a combination of a specific EtherType and MessageID values (part of the PTP Common Header). The actual numeric value is not yet defined in the IEEE802.1AS standard. It is possible that all IEEE1588 time sync frames and IEEE802.11 wireless LAN location estimation time sync messages follow the same EtherType but varying Ether subtypes (aka messageID).</p> <p>The MsgIDTSEn (specified below) qualifies the types of frames that the hardware needs to time stamp.</p>

**Table 471: PTP Global Configuration1 Register**  
Offset: 0x1, or Decimal 1

Bits	Field	Type	Description
15:0	MsgIDTSEn	RWR 0x0	<p>Message Identifier Time Stamp Enable.</p> <p>MessageID is part of the PTP common header for time sync frames. There are PTP frames which need to be time stamped by hardware and some that do not need to be. This field identifies the PTP frame types that need to be time stamped by the hardware.</p> <p>The MsgIDTSEn refers to the bit mask enables, where each bit indicates whether the vectorized MessageID value needs to be time stamped or not.</p> <p>0x0 = Indicates to hardware to NOT time stamp both incoming and/or outgoing PTP frames which match the MessageID.</p> <p>0x1= Indicates to hardware to time stamp both incoming and/or outgoing PTP frames which match the MessageID.</p> <p>For example, if MessageID field (in the PTP common header) with a value of 0x4 ought to be time stamped in hardware, then MsgIDTSEn[4] should be configured to a 0x1. Then for the incoming PTP frame with the MessageID field of 0x4, one of the two arrival counters get updated (PTPArr0Time or PTPArr1Time). The exact time counter is identified by the TSArrPtr field below). For an outgoing PTP frame with the MessageID field of 0x4, the TSDepTime counter gets updated for an incoming frame with the MessageID field from the PTCPCommon header matching 0x4.</p>

**Table 472: PTP Global Configuration2 Register**  
Offset: 0x2, or Decimal 2

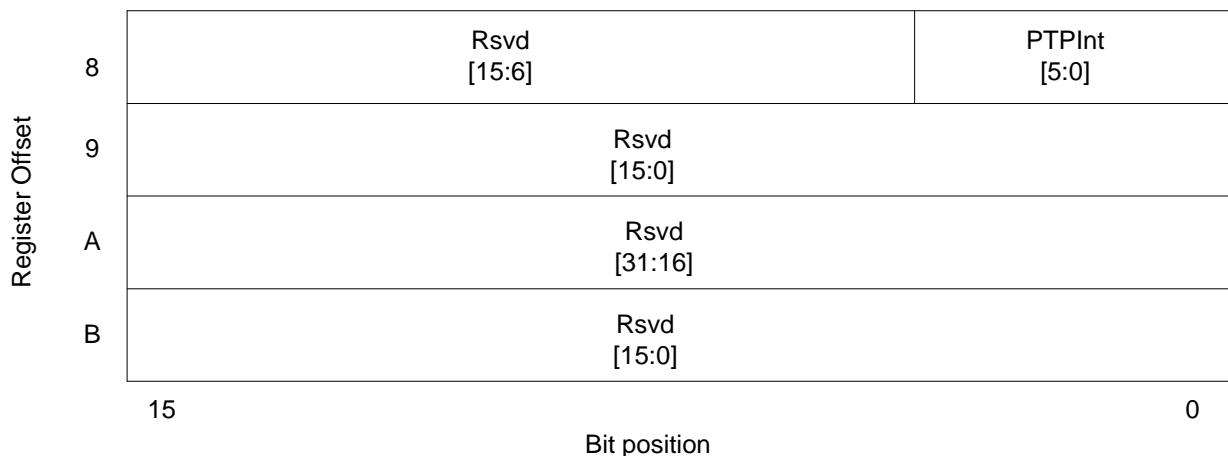
Bits	Field	Type	Description
15:0	TSArrPtr	RWR 0x0	<p>Time Stamp Arrival Time Counter Pointer.</p> <p>If the incoming PTP frame needs to be time stamped (based on MsgIDTSEn), this field determines whether the hardware logic should use PTPArr0Time or PTPArr1Time for storing the arriving frame's time stamp information.</p> <p>Each bit in this field corresponds to the sixteen combinations of the vectorized MessageID field. For example, if TSArrPtr[2] is set to a 0x1, it indicates to the hardware that if MsgIDTSEn[2] is set. Then use the PTPArr1Time counter for storing the incoming PTP frame's time stamp.</p> <p>However, if TSArrPtr[2] is set to a 0x0 that indicates to the hardware that if MsgIDTSEn[2] is set. Then use the PTPArr0Time counter for storing the incoming PTP frame's time stamp.</p> <p>Vectorized: Vectorized term here refers to converting the hexadecimal MessageID field into a sixteen bit binary number.</p>

**Table 473: PTP Global Configuration3 Register**  
Offset: 0x3, or Decimal 3

Bits	Field	Type	Description
15:0	Reserved	RSVD	Reserved for future use.

#### A.8.3.4 PTP Global Status Data Structure Registers

**Figure 90: PTP Global Status Data Structure Registers**

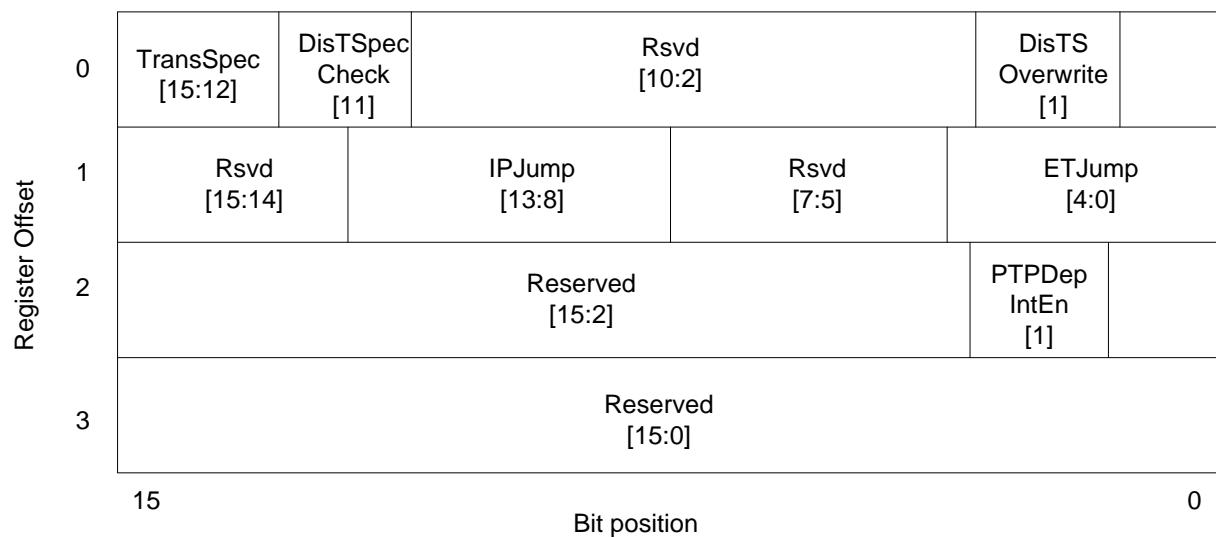


**Table 474: PTP Global Status0 Register**  
Offset: 0x8, or Decimal 8

Bits	Field	Type	Description
15:6	Reserved	RSVD	Reserved for future use.
5:0	PTPInt	RWR 0x0	Precise Time Protocol Interrupt  The PTP Interrupt bit gets set for a given port when an incoming PTP frame is time stamped and PTPArrIntEn for that port is set to 0x1. Similarly PTP Interrupt bit gets set for a given port when an outgoing PTP frame is time stamped and PTPDepIntEn for that port is set to 0x1.  The hardware logic sets this per port bit, based on the above criteria, and it is cleared upon software reading and clearing the corresponding time counter valid bits that are valid for that port.

#### A.8.3.5 PTP Port Configuration Data Structure Registers

**Figure 91: PTP Port Configuration Data Structure Registers**



**Table 475: PTP Port Configuration0 Register**  
Offset: 0x0, or Decimal 0

Bits	Field	Type	Description
15:12	TransSpec	RWS 0x1	<p>PTP Transport Specific value.</p> <p>The Transport Specific bits present in PTP Common header are used to differentiate between IEEE1588, IEEE802.1AS frames. This is to differentiate between various timing protocols running on either Layer2 or higher protocol layers.</p> <p>In hardware, in addition to comparing the EtherType to determine that the incoming frame is a PTP frame, it compares the TransSpec bits to the incoming PTP common header's Transport Specific bits. If there is a match, then hardware logic time stamps the frames indicated by MsgIDTSEn and optionally interrupts the CPU. If there is no match, then the hardware does not perform any operations in the PTP core.</p> <p>For IEEE 1588 networks, this bit is expected to be configured to a 0x0. For IEEE 802.1AS networks, this is expected to be configured to 0x1.</p>
11	DisTSpecCheck	RWR 0x0	<p>Disable Transport Specific Check.</p> <p>0x1= Disables checking for the Transport Specific segment of the PTP Common header, between the incoming packet data and the configured TransSpec (PTP).</p> <p>0x0 = Enables checking for the Transport Specific segment of the PTP Common header between the incoming packet data and the configured TransSpec (PTP Port Configuration, Offset 0x0).</p>
10:2	Reserved	RSVD	Reserved for future use.
1	DisTSOverwrite	RWR 0x0	<p>Disable Time Stamp Counter Overwriting.</p> <p>When set to 0x1, PTPArr0Time, PTPArr1Time, and PTPDepTime values do not get overwritten until their corresponding valid bits (defined in PTP Port Status Data Structure below) are not cleared. This situation only arises when a port based time stamp counter is written by hardware logic, but the software layer has not read the data.</p> <p>When set to 0x0, PTPArr0Time, PTPArr1Time, and PTPDepTime values do get overwritten, even though their corresponding valid bits (defined in PTP Port Status Data Structure below), are not cleared.</p>
0	DisPTP	RWR 0x0	<p>Disable Precise Time Stamp logic (per-port bit).</p> <p>0x0 = PTP logic within the chip is enabled. 0x1 = PTP logic is disabled, i.e., hardware logic does not recognize or timestamp PTP frames. Even interrupt generation logic is disabled.</p>

**Table 476: PTP Port Configuration1 Register**  
Offset: 0x1, or Decimal 1

Bits	Field	Type	Description
15:14	Reserved	RSVD	Reserved for future use.

**Table 476: PTP Port Configuration1 Register (Continued)**  
Offset: 0x1, or Decimal 1

Bits	Field	Type	Description
13:8	IPJump	RWS 0x2	<p>Internet Protocol Jump.</p> <p>This field specifies to the PTP hardware logic how many bytes it should skip, starting at the value specified by ETJump (PTP Port Configuration register, Offset 0x1).</p> <p><b>NOTE:</b> This specifies the jump to the beginning of the IPv4 or IPv6 headers for the hardware parser.</p> <p>This allows flexibility in the hardware to skip past the protocol chains that are specific to customer networks including MPLS.</p> <p>For example, if ETJump is programmed to 0xC and IPJump is programmed to 0x16, this indicates to hardware to skip 0x22 bytes to get to the IP header. It can be either an IPv4 or IPv6 header.</p>
7:5	Reserved	RSVD	Reserved for future use.
4:0	ETJump	RWS 0xC	<p>EtherType Jump.</p> <p>This field specifies to the PTP hardware logic how many bytes should it skip starting from MAC-DA of the frame to get to the EtherType of the frame. The hardware would skip that many bytes, and then compare the next 2 bytes to the configured PTPEType (PTP Global Configuration Register, Offset 0x0).</p> <p>This allows flexibility in the hardware to skip past the protocol chains that are specific to customer networks including IEEE802.1q tag, Provider tag.</p>

**Table 477: PTP Port Configuration2 Register**  
Offset: 0x2, or Decimal 2

Bits	Field	Type	Description
15:2	Reserved	RSVD	Reserved for future use.
1	PTPDepIntEn	RWR 0x0	<p>Precise Time Protocol Port Departure Interrupt enable.</p> <p>This field indicates the per-port interrupt enable for outgoing PTP frame from a given port. When a bit is enabled in this field, it indicates that whenever hardware logic time stamps a PTP frame to this port, it needs to send an interrupt to the CPU.</p> <p>0x0 = Disable PTP departure counter, based interrupt generation. Even if the PTPDepTimeValid is set to 0x1, PTPInt does not get generated by hardware logic for outgoing PTP frames.</p> <p>0x1 = Enable PTP departure counter, based interrupt generation. If the PTPDepTimeValid is set to 0x1, PTPInt does get generated by hardware logic for outgoing PTP frames.</p> <p><b>NOTE:</b> Hardware logic only time stamps the PTP frames when configured to do so by MsgIDTSEn field (as specified above).</p>

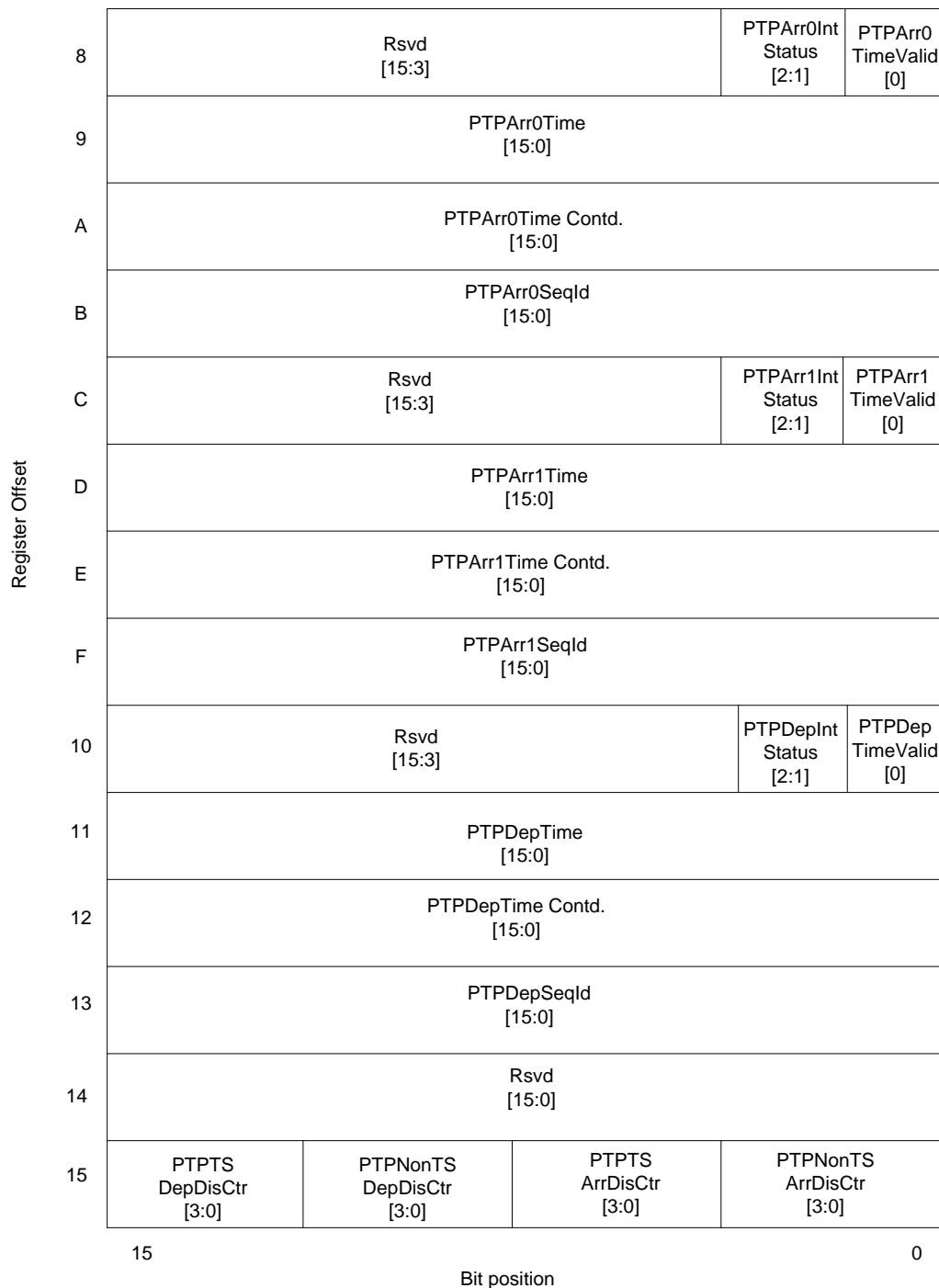


**Table 477: PTP Port Configuration2 Register (Continued)**  
Offset: 0x 2, or Decimal 2

Bits	Field	Type	Description
0	PTPArrIntEn	RWR 0x0	<p>Precise Time Protocol Port Arrival Interrupt enable.</p> <p>This field indicates the per-port interrupt enable for incoming PTP frames from a given port. When a bit is enabled in this field it indicates that whenever hardware logic time stamps a PTP frame from this port, it needs to send an interrupt to the CPU.</p> <p>0x0 = Disable PTP arrival counter, based interrupt generation. Even if the PTPArr0TimeValid or PTPArr1TimeValid is set to 0x1 for that port, PTPInt does not get generated by hardware logic for incoming PTP frames from this port.</p> <p>0x1 = Enable PTP arrival counter, based interrupt generation. If the PTPArr0TimeValid or PTPArr1TimeValid is set to 0x1, PTPInt does get generated by hardware logic for incoming PTP frames.</p> <p><b>NOTE:</b> Hardware logic only time stamps the PTP frames when configured to do so by MsgIDTSEn field (specified above).</p>

### A.8.3.6 PTP Port Status Registers

**Figure 92: PTP Port Status Data Structure Registers**



**Table 478: PTP Port Status0 Register**

Offset: 0x8, or Decimal 8

Bits	Field	Type	Description
15:3	Reserved	RSVD	Reserved for future use.
2:1	PTPArr0IntStatus	RWR 0x0	<p>Precise Time Protocol Arrival Time 0 Interrupt Status</p> <p>The PTP Arrival time 0 Interrupt bit gets set for a given port, when an incoming PTP frame is time stamped in PTPArr0Time counter.</p> <p>0x0 = Normal, i.e., none of the error conditions stated below are valid for this packet.</p> <p>0x1 = If the PTPArr0Time counter with its associated valid and SequenceID was overwritten since more than one PTP frame that needed to use the arrival0 counters, arrived at the device through this port before CPU cleared the corresponding valid and counter bits for the previous PTP frame(s).</p> <p>0x2 = If the incoming frame could not be time stamped in hardware because the DisTSOverwrite was set to a 0x1 and PTPArr0TimeValid was 0x1, when this PTPFrame was processed in hardware logic. This can occur, when more than one PTP frame that needs time stamping into the arrival0 counters, arrives at the device before CPU clears the valid bits for the previous frame.</p> <p>0x3 = Reserved</p> <p><b>NOTE:</b> If the PTP frame gets discarded inside the device, for policy, CRC, queue congestion, or any other reasons then one of the PTP arrival discard counters is updated (PTPNonTSArrDisCtr or PTPTSArrDisCtr). See the discard counter description for further details.</p>
0	PTPArr0TimeValid	RWR 0x0	<p>Precise Time Protocol Arrival 0 Time Valid</p> <p>When the PTPArr0Time value is updated by hardware, this bit is set to a 0x1 validating the time counter.</p> <p>0x0 = PTPArr0Time is not valid.</p> <p>0x1 = PTPArr0Time is valid and PTPArr0IntStatus represents the status information for the PTPArr0Time counter. This is set by hardware for the frames that are assured of reaching the CPU. For frames with a CRC error, this bit is not set, but either PTPNonTSArrCtr or PTPTSArrCtr is updated.</p> <p><b>NOTE:</b> This valid bit needs to be cleared by software, after reading the value, and hardware does not provide any auto-clearing mechanisms. This is because hardware has no way to figure out if software is finished reading all the relevant registers for a particular Time counter before clearing the valid bit.</p>

**Table 479: PTP Port Status1 Register**

Offset: 0x9 and 0xA, or Decimal 9 and Decimal 10

Bits	Field	Type	Description
31:0	PTPArr0Time	RWR 0x0	<p>Precise Time Protocol Arrival 0 Time counter.</p> <p>This indicates the PTP Arrival 0 time stamp value that is captured by the PTP logic for a PTP frame that needs to be time stamped. The captured time stamp value is from a Global Timer counter running off of a device internal clock.</p> <p>The value in this counter is validated by the PTPArr0TimeValid bit, and PTPArr0IntStatus indicates the status of the PTP frame through the device, as described above.</p> <p><b>NOTE:</b> The maximum Jitter associated with time stamping within the hardware is one TSClkPer value.</p>

**Table 480: PTP Port Status2 Register**

Offset: 0xB, or Decimal 11

Bits	Field	Type	Description
15:0	PTPArr0SeqId	RWR 0x0	<p>Precise Time Protocol Arrival 0 Sequence Identifier.</p> <p>This indicates the sequence identifier (extracted in hardware from the incoming frames PTP Common Header) for the frame whose time stamp information has been captured by hardware logic in PTPArr0Time register.</p>

**Table 481: PTP Port Status3 Register**

Offset: 0xC, or Decimal 12

Bits	Field	Type	Description
15:3	Reserved	RSVD	Reserved for future use.

**Table 481: PTP Port Status3 Register (Continued)**  
Offset: 0xC, or Decimal 12

Bits	Field	Type	Description
2:1	PTPArr1IntStatus	RWR 0x0	<p>Precise Time Protocol Arrival Time 1 Interrupt Status</p> <p>The PTP Arrival time 1 Interrupt bit gets set for a given port when an incoming PTP frame is time stamped in PTPArr1Time counter.</p> <p>0x0 = Normal, i.e., none of the error conditions stated below are valid for this packet.</p> <p>0x1 = If the PTPArr1Time counter with its associated valid and SequenceID was overwritten since more than one PTP frame that needed to use the arrival1 counters, arrived at the device through this port before CPU cleared the corresponding valid and counter bits for the previous PTP frame(s).</p> <p>0x2 = If the incoming frame could not be time stamped in hardware because the DisTSOverwrite was set to a 0x1 and PTPArr1TimeValid was 0x1, when this PTPFrame was processed in hardware logic. This can occur, when more than one PTP frame that needs time stamping into the arrival0 counters, arrives at the device before CPU clears the valid bits for the previous frame.</p> <p>0x3 = Reserved</p> <p><b>NOTE:</b> If the PTP frame gets discarded inside the device for policy, CRC, queue congestion, or any other reasons, then one of the PTP arrival discard counters is updated (PTPNonTSArrDiscr or PTPTSArrDiscr). See the discard counter description for further details.</p>
0	PTPArr1TimeValid	RWR 0x0	<p>Precise Time Protocol Arrival 1 Time Valid</p> <p>When the PTPArr1Time value is updated by hardware, this bit is set to a 0x1 validating the time counter.</p> <p>0x0 = PTPArr1Time is not valid.</p> <p>0x1 = PTPArr1Time is valid, and PTPArr1IntStatus represents the status information for the PTPArr1Time counter. This is set by hardware for the frames that are assured of reaching the CPU. For frames with a CRC error, this bit is not set, but either PTPNonTSArrCtr or PTPTSArrCtr is updated.</p> <p><b>NOTE:</b> This valid bit needs to be cleared by software, after reading the value, and hardware does not provide any auto-clearing mechanisms. This is because hardware has no way to figure out if software is finished reading all the relevant registers for a particular Time counter before clearing the valid bit.</p>

**Table 482: PTP Port Status4 Register**

Offset: 0x D and 0xE, or Decimal 13 and 14

Bits	Field	Type	Description
31:0	PTPArr1Time	RWR 0x0	<p>Precise Time Protocol Arrival 1 Time counter.</p> <p>This indicates the PTP Arrival 1 time stamp value that is captured by the PTP logic for a PTP frame that needs to be time stamped. The captured time stamp value is from a Global Timer counter running off of a device internal clock.</p> <p>The value in this counter is validated by the PTPArr1TimeValid bit and the PTPArr1IntStatus bit indicates the status of the PTP frame through the device described above.</p> <p><b>NOTE:</b> The maximum Jitter associated with time stamping within the hardware is one TSClkPer value.</p>

**Table 483: PTP Port Status5 Register**

Offset: 0xF, or Decimal 15

Bits	Field	Type	Description
15:0	PTPArr1SeqId	RWR 0x0	<p>Precise Time Protocol Arrival 1 Sequence Identifier.</p> <p>This indicates the sequence identifier (extracted in hardware from incoming frames PTPCommonHeader) for the frame whose time stamp information has been captured by hardware logic in PTPArr1Time register.</p>

**Table 484: PTP Port Status6 Register**

Offset: 0x10, or Decimal 16

Bits	Field	Type	Description
15:3	Reserved	RSVD	Reserved for future use.

**Table 484: PTP Port Status6 Register (Continued)**  
Offset: 0x10, or Decimal 16

Bits	Field	Type	Description
2:1	PTPDepIntStatus	RWR 0x0	<p>Precise Time Protocol Departure Time Interrupt Status</p> <p>The PTP Departure time Interrupt bit gets set for a given port when an incoming PTP frame is time stamped in the PTPDepTime counter.</p> <p>0x0 = Normal, i.e., none of the error conditions stated below are valid for this packet.</p> <p>0x1 = The PTPDepTime counter, with its associated valid and SequenceID, was overwritten, since more than one PTP frame that needed to use departure counter exited the device through this port before CPU cleared the corresponding valid and counter bits for the previous PTP frame(s).</p> <p>0x2 = The outgoing frame could not be time stamped in hardware because the DisTSOverwrite was set to a 0x1 and PTPDepTimeValid was 0x1 when this PTPFrame was processed in hardware logic. This can occur when more than one PTP frame that needs time stamping into the departure counter leaves the device, before CPU clears the valid bits for the previous frame.</p> <p>0x3 = Reserved</p> <p><b>NOTE:</b> If the PTP frame gets discarded inside the device, for CRC reasons, then the PTP departure discard counter gets updated (PTPNonTSDepDisCtr or PTPTSDepDisCtr). See the discard counter description for further details.</p>
0	PTPDepTimeValid	RWR 0x0	<p>Precise Time Protocol Departure Time Valid</p> <p>When the PTPDepTime value is updated by hardware, this bit is set to a 0x1 validating the time counter.</p> <p>0x0 = PTPDepTime is not valid.</p> <p>0x1 = PTPDepTime is valid and PTPDepIntStatus represents the status information for the PTPDepTime counter. This is set by hardware for the frames that are assured to depart the port. For frames with a CRC error, this bit is not set, but either PTPNonTSDepCtr or PTPTSDepCtr is updated.</p> <p><b>NOTE:</b> This valid bit needs to be cleared by software, after reading the value, and hardware does not provide any auto-clearing mechanisms. This is because hardware has no way to figure out if software is finished reading all the relevant registers for a particular Time counter before clearing the valid bit.</p>

**Table 485: PTP Port Status7 Register**

Offset: 0x11 and 0x12, or Decimal 17 and 18

Bits	Field	Type	Description
31:0	PTPDepTime	RWR 0x0	<p>Precise Time Protocol Departure Time counter.</p> <p>This indicates the PTP Departure time stamp value that is captured by the PTP logic for a PTP frame that needs to be time stamped. The captured time stamp value is from a Global Timer counter running off of a device internal clock.</p> <p>The value in this counter is validated by the PTPDepTimeValid bit and the PTPDepIntStatus indicates the status of the PTP frame through the device described above.</p> <p><b>NOTE:</b> The maximum Jitter associated with time stamping within the hardware is one TSClkPer value.</p>

**Table 486: PTP Port Status8 Register**

Offset: 0x13, or Decimal 19

Bits	Field	Type	Description
15:0	PTPDepSeqId	RWR 0x0	<p>Precise Time Protocol Departure Sequence Identifier.</p> <p>This indicates the sequence identifier (extracted in hardware from the incoming frames PTP Common Header) for the frame whose time stamp information has been captured by hardware logic in the PTPDepTime register.</p>

**Table 487: PTP Port Status9 Register**

Offset: 0x15, or Decimal 21

Bits	Field	Type	Description
15:12	PTPTSDepDisCtr	RWR 0x0	<p>Precise Time Protocol Departure frame discard counter for PTP frames that need hardware time stamping.</p> <p>This counter is incremented by the hardware logic whenever it discards a PTP frame that needs hardware time stamping (i.e., PTPEtype is a match, and the MsgIDTSEn bit for the corresponding MsgID field in the outgoing PTP frame is set). The PTP frame could be discarded because of CRC reasons in the egress pipe.</p> <p>This counter wraps around in hardware.</p>
11:8	PTPNonTSDepDisCtr	RWR 0x0	<p>Precise Time Protocol Departure frame discard counter for PTP frames that do not need hardware time stamping.</p> <p>This counter is incremented by the hardware logic whenever it discards a PTP frame that does not need to be time stamped (i.e., PTPEtype is a match, but the MsgIDTSEn bit for the corresponding MsgID field in the outgoing PTP frame is not set). The PTP frame could be discarded because of CRC reasons in the egress pipe.</p> <p>This counter wraps around in hardware.</p>

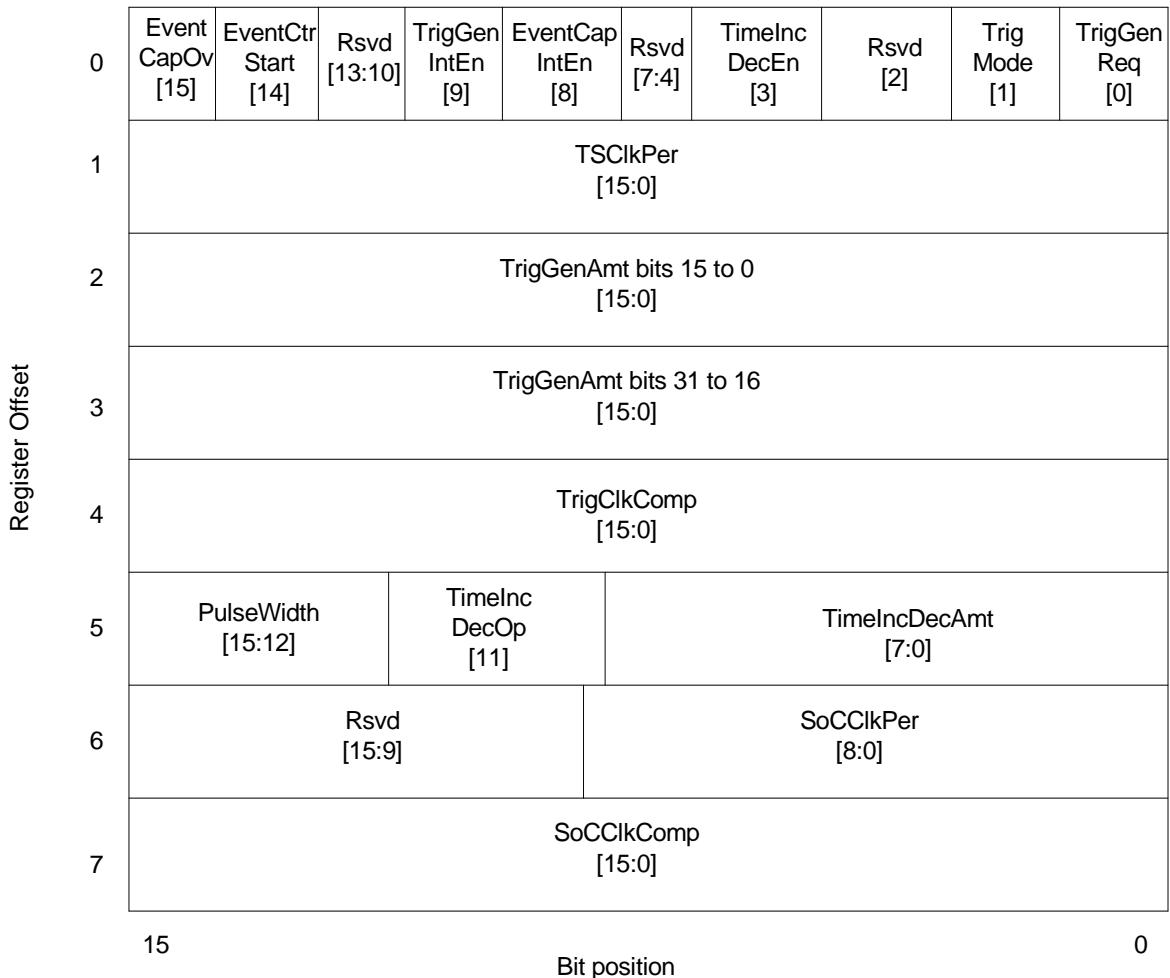


**Table 487: PTP Port Status9 Register (Continued)**  
Offset: 0x15, or Decimal 21

Bits	Field	Type	Description
7:4	PTPTSArrDisCtr	RWR 0x0	Precise Time Protocol arrival frame discard counter for PTP frames that need hardware time stamping.  This counter is incremented by the hardware logic whenever it discards a PTP frame that needs hardware time stamping (i.e., PTPEtype is a match, and the MsgIDTSEn bit for the corresponding MsgID field in the outgoing PTP frame is set). The PTP frame could be discarded because of CRC, Policy, Queue congestion, or any other reason inside the device.  This counter wraps around in hardware.
3:0	PTPNonTSArrDis Ctr	RWR 0x0	Precise Time Protocol Non time stamp Arrival frame discard counter.  This counter is incremented by the hardware logic whenever it discards a PTP frame that does not need hardware time stamping (i.e., PTPEtype is a match, but the MsgIDTSEn bit for the corresponding MsgID field in the outgoing PTP frame is not set). The PTP frame could be discarded because of CRC, Policy, Queue congestion, or any other reason inside the device.  This counter wraps around in hardware.

### A.8.3.7 Timing Applications Interface Registers

**Figure 93: TAI Global Configuration Data Structure**



**Table 488: TAI Global Configuration, PTP Port = 0xE**  
**Offset: 0x0, or Decimal 0**

Bits	Field	Type	Description
15	EventCapOv	RWR 0x0	<p>Event Capture Overwrite</p> <p>When 0x1, this bit enables overwriting the EventCapRegister (PTP Global Status Register, Offset 0xE). The hardware would only overwrite the EventCapRegister if the previously captured event register has not been read by the software.</p> <p>When 0x0, this bit specifies to hardware logic to capture an event namely, take a snapshot of PTP Global Timer value at the rising edge of PTP_EVENT_REQ (a input into the core) and wait for software to read the EventCapRegister before capturing another event.</p>

**Table 488: TAI Global Configuration, PTP Port = 0xE (Continued)**  
Offset: 0x0, or Decimal 0

Bits	Field	Type	Description
14	EventCtrStart	RWR 0x0	<p>Event Counter Start</p> <p>When 0x1, this bit enables the hardware logic to start incrementing the EventCapCtr register (PTP Global Status Register, 0xD) whenever it captures a low-to-high transition on the PTP_EVENT_REQ signal.</p> <p>When 0x0, the EventCapCtr is not modified by hardware logic, even when it captures a low-to-high transition on the PTP_EVENT_REQ signal.</p>
13:10	Reserved	RSVD	Reserved for future use.
9	TrigGenIntEn	RWR 0x0	<p>Trigger Generator Interrupt Enable.</p> <p>When 0x1, the TAI block would generate an interrupt whenever a PTP_TRIG_GEN event has been sent out on the PTP_TRIG_GEN signal output.</p> <p>When 0x0, no interrupts are generated by the PTP_TRIG_GEN logic.</p>
8	EventCapIntEn	RWR 0x0	<p>Event Cap Interrupt Enable</p> <p>When 0x1, the TAI block would generate an interrupt whenever an event has been captured on the PTP_EVENT_REQ signal.</p> <p>When 0x0, no interrupts are generated by the EventCap logic.</p>
7:4	Reserved	RSVD	Reserved for future use.
3	TimeIncDecEn	SC 0x0	<p>Time Increment Decrement Enable</p> <p>This is used to adjust the PTP Global Time counter value, with respect to the phase offset computed by the PTP firmware, using the PTP control messages like Sync, PDelayReq, and PDelayResp. The assumption is that the software maintains the 64-bit seconds field of the PTP time of day and hardware maintains the 32-bit field (in PTP clock increments) in the PTP Global Time counter (TAI Global Status Register, Offset 0xE and 0xF).</p> <p>When 0x1, this bit enables the hardware logic to increment or decrement the PTP Global Time counter by the value specified by TimeIncDecAmt (PTP Global Configuration Register, Offset 0x6).</p> <p>When 0x0, the hardware logic does not modify the PTP Global Time counter value.</p> <p><b>NOTE:</b> This function is executed once by the hardware, and upon execution this bit is cleared to 0x0.</p>
2	Reserved	RSVD	Reserved for future use.

**Table 488: TAI Global Configuration, PTP Port = 0xE (Continued)**  
Offset: 0x0, or Decimal 0

Bits	Field	Type	Description
1	TrigMode	RWR 0x0	<p>Trigger Mode  When 0x1, the hardware logic matches the PTP Global Timer (TAI Global Status, Offset 0xE and 0xF) with the TrigGenAmt (TAI Global Configuration, Offset 0x2 and 0x3) and generates a pulse on the PTP_TRIG_GEN output signal. The pulse width for this is specified by PulseWidth (TAI Global Configuration, Offset 0x5).  <b>NOTE:</b> The minimum pulse width that can be generated is one TSClkPer value, and the maximum pulse width is 15 times the TSClkPer value.</p> <p>When 0x0, the hardware logic uses the value specified in the TrigGenAmt as the period for generating periodic pulses on the PTP_TRIG_GEN signal with a 50% duty cycle clock.  <b>NOTE:</b> The minimum clock period that can be generated on the PTP_TRIG_GEN output signal is 2 times the TSClkPer value.</p> <p>For example, if a 1 pps signal needs to be generated, the TrigMode is set to 0x0, if the TSClkPer is set to 8 ns and the TrigGenAmt is set to <math>125 \times 10^6</math> cycles.</p>
0	TrigGenReq	RWR 0x0	<p>Trigger Generation Request  When 0x1, it validates the TrigGenAmt, TrigMode, and TrigClkComp fields. This enables the hardware logic to generate either a trigger, based on the TrigGenAmt, or a clock, based on the period specified in TrigGenAmt.</p>

**Table 489: TAI Global Configuration Register, PTP Port = 0xE and 0xF**  
Offset: 0x1, or Decimal 1

Bits	Field	Type	Description
15:0	TSClkPer	RWS 0x1F40	<p>Time Stamping Clock Period in pico seconds.  This field specifies the clock period for the time stamping clock supplied to the PTP hardware logic.  This is the clock that is used by the hardware logic to update the PTP Global Time counter (PTP Global register, Offset 0x9 and 0xA).  The default is calculated based on 125 MHz period clock.</p>

**Table 490: TAI Global Configuration0 Register**

Offset: 0x2 and 0x3, or Decimal 2 and 3

Bits	Field	Type	Description
15:0	TrigGenAmt	RWR 0x0	<p>Trigger Generation Amount</p> <p>This field specifies the PTP Time Application Interface trigger generation time value.</p> <p>When TrigMode is 0x1, the value specified in this field is compared with the PTP Global Timer (PTP Global Status register, Offset 0x9 and 0xA) and, whenever it matches, a pulse is generated whose width is configured using the PulseWidth field (TAI Global Configuration register, Offset 0x5).</p> <p>When TrigMode is 0x0, the value is used as a clock period in TSClkPer increments to generate an output clock on the PTP_TRIG_GEN signal.</p> <p>When TrigMode is 0x0, the TrigClkComp value constantly gets accumulated internally, and when this accumulated value exceeds the value specified in TSClkPer, a TSClkPer value gets added to the clock output momentarily.</p>

**Table 491: TAI Global Configuration1 Register**

Offset: 0x4, or Decimal 4

Bits	Field	Type	Description
15:0	TrigClkComp	RWR 0x0	<p>Trigger mode Clock Compensation Amount in pico seconds</p> <p>This field is valid only when TrigGenReq is 0x1 and TrigMode is set to 0x0.</p> <p>The field specifies the remainder amount for the clock that is being generated with a period specified by the TrigGenAmt.</p> <p>When TrigMode is 0x0, the TrigClkComp value constantly gets accumulated internally, and when this accumulated value exceeds, the value specified in TSClkPer, a TSClkPer value gets added to the clock output momentarily.</p>

**Table 492: TAI Global Configuration2 Register**

Offset: 0x5, or Decimal 5

Bits	Field	Type	Description
15:12	PulseWidth	RWS 0xF	<p>Clock high pulse width in units of TSClkPer.</p> <p>This specifies the pulse width of the clock that gets generated on the TrigModeResp, when TrigMode is 0x1 in units defined by the TSClkPer.</p> <p><b>NOTE:</b> When configured to a 0x0, the results are un-deterministic. Therefore, do not program this field to 0x0.</p>

**Table 492: TAI Global Configuration2 Register (Continued)**  
Offset: 0x5, or Decimal 5

Bits	Field	Type	Description
11	TimeIncDecOp	RWR 0x0	<p>Time increment decrement operation.  When 0x0, TimeIncDecAmt is considered as an increment value that needs to be added to the PTP Global Time Counter when TimeIncDecEn is 0x1.</p> <p>When 0x1, TimeIncDecAmt is considered as a decrement value that needs to be subtracted from the PTP Global Time Counter when TimeIncDecEn is 0x1.</p> <p>All updates are completed within the same cycle of TimeIncDecEn changing state from 0x0 to 0x1.</p>
10:0	TimeIncDecAmt	RWR 0x0	<p>Time Increment Decrement amount.  This field is valid only when TimeIncDecEn is 0x1.</p> <p>This field specifies the number of units of PTP Global Time that need to be incremented or decremented based on the TimeIncDecOp field. This is used for adjusting the PTP Global Time counter value.</p>

**Table 493: TAI Global Configuration3 Register**  
Offset: 0x6, or Decimal 6

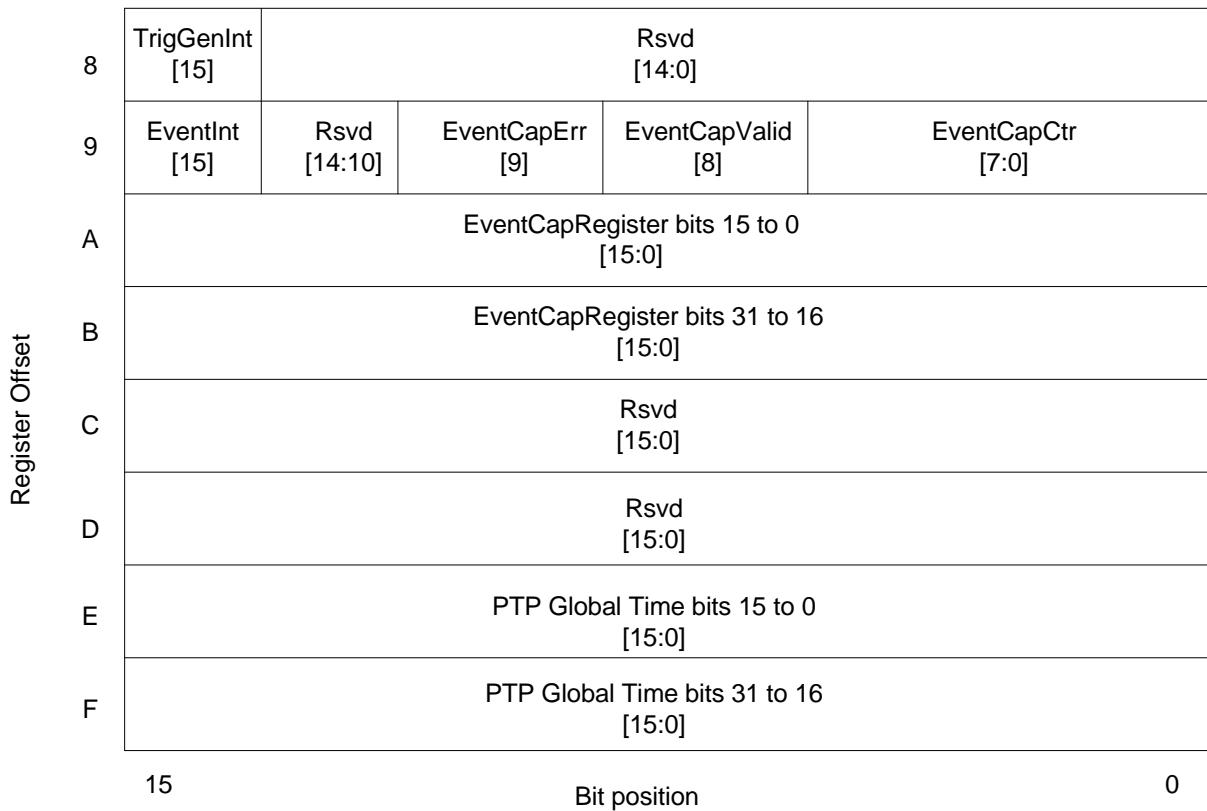
Bits	Field	Type	Description
15:9	Reserved	RSVD	Reserved for future use.
8:0	SoCClkPer	RWS 0x186	<p>System on a Chip Clock Period.  This specifies the clock period for the clock that gets generated from the PTP block to the rest of the device. The period is specified in TSClkPer increments.</p> <p>For example, if the TSClkPer is 8 ns, and the device clock needs to be toggling every 3.125 us periods, then this field needs to be programmed to 0x186 and the SoCClkComp field needs to be programmed to 0x1388.</p> <p>3125 ns / 8 ns = 390.625 TSClkPer cycles.  Decimal 390 in hexadecimal is 0x186 and 0.625 x 8000 ps = 0x1388 ps.</p>

**Table 494: TAI Global Configuration4 Register**  
Offset: 0x7, or Decimal 7

Bits	Field	Type	Description
15:0	SoCClkComp	RWS 0x1388	<p>System on a Chip Clock Compensation Amount in pico seconds  The field specifies the remainder amount, when the clock is generated with a period specified by the SoCClkPer field. The hardware logic keeps track of the remainder for every clock tick generation and compensates for it.</p>

### A.8.3.8 PTP Time Application Interface Registers

**Figure 94: PTP Time Application Interface Global Status Data Structure**



**Table 495: TAI Global Status0 Register**  
Offset: 0x8, or Decimal 8

Bits	Field	Type	Description
15	TrigGenInt	RWR 0x0	<p>Trigger generate mode Interrupt.</p> <p>The TrigGenInt bit is set by the TAI block, when the TrigGenIntEn is 0x1 and when the hardware logic captures a trigger on the PTP_TRIG_GEN signal.</p> <p>This interrupt gets tied to the device interrupt pin.</p>
14:0	Reserved	RSVD	Reserved for future use.

**Table 496: TAI Global Status1 Register**  
Offset: 0x9, or Decimal 9

Bits	Field	Type	Description
15	EventInt	RWR 0x0	<p>Event Capture Interrupt.</p> <p>This bit is set by the TAI block when the EventIntEn field is 0x1 and when the hardware logic captures an event in the EventCapRegister.</p> <p>This interrupt gets tied to the device interrupt pin.</p>
14:10	Reserved	RSVD	Reserved for future use.
9	EventCapErr	RWR 0x0	<p>Event Capture Error.</p> <p>This bit is set by the hardware logic when an event has been observed on the PTP_EVENT_REQ signal, but the EventCapValid field is already set to 0x1. This condition can occur when events are observed on the PTP_EVENT_REQ signal faster than the local CPU can read the captured event related counter values.</p>
8	EventCapValid	RWR 0x0	<p>Event Capture Valid.</p> <p>When 0x1, this bit validates the EventCapRegister.</p>
7:0	EventCapCtr	RSVD	<p>Event Capture Counter.</p> <p>This field is incremented by TAI block when the EventCtrStart field is set to 0x1. This field is incremented, whenever an event (a low-to-high transition) has been registered on the PTP_EVENT_REQ signal.</p> <p><b>NOTE:</b> There is no special logic provided to detect this counter's wrap arounds.</p>

**Table 497: TAI Global Status2 Register**  
Offset: 0xA and 0xB, or Decimal 10 and 11

Bits	Field	Type	Description
31:0	EventCap Register	RWR 0x0	<p>Event Capture Register</p> <p>This register captures the value of the PTP Global Timer when an event (a low-to-high transition) has been registered by the TAI block on the PTP_EVENT_REQ signal. If the EventCapOv field is 0x1, then this register indicates the time captured, for the last event in the hardware.</p> <p>When the EventCapErr field is 0x1, the contents in this register are invalid.</p> <p><b>NOTE:</b> The maximum jitter for the EventCapRegister time value, with respect to the rising edge of the PTP_EVENT_REQ input signal, is one TSClkPer value.</p> <p>The minimum PTP_EVENT_REQ signal high or low width has to be equal to or greater than 1.5 times the TSClkPer value.</p> <p>For hardware to capture the Event request on the PTP_EVENT_REQ signal, the minimum gap between two consecutive events must be 150 ns (<b>7 times clk_sysp</b>) plus 5 times the TSClkPer value.</p>

**Table 498: PTP Global Status1 Register**  
Offset: 0xE and 0xF, or Decimal 14 and 15

Bits	Field	Type	Description
31:0	PTPGlobalTime	RWR 0x0	<p>Precise Time Protocol Global Timer.</p> <p>This indicates the global timer value that is running off of the free running device clock. Based on PTP protocol time of day computations, this field can be either incremented or decremented, using the TimeIncDecAmt field (TAI Global Configuration, Offset 0x5) and by turning on the TimeIncDecEn field (TAI Global Configuration, Offset 0x0) to 0x1 and by selecting an increment or a decrement operation using the TimeIncDecOp field (TAI Global Configuration, Offset 0x5).</p> <p><b>NOTE:</b> This register is updated in the same cycle when the TimeIncDecEn field goes high.</p> <p>This counter wraps around in hardware.</p>

## A.8.4 MAC MIB Counters

**Table 499: MAC MIB Counters**

Offset: Port0: 0x73000–0x7307C, Port1: 0x77000–0x7707C

Offset	Width	Counter Name	Description
0x0	64	GoodOctetsReceived	The sum of lengths of all good Ethernet frames received: frames that are not Bad frames NOR MAC Control frames. <b>NOTE:</b> This does <i>not</i> include IEEE 802.3 pause messages, but, does include bridge control packets like LCAP and BPDU.
0x8	32	BadOctetsReceived	The sum of lengths of all bad Ethernet frames received
0x10	32	GoodFramesReceived	The number of Ethernet frames received that are not Bad Ethernet frames or MAC Control packets. <b>NOTE:</b> This does include Bridge Control packets.
0xC	32	MACTransError	The number of frames not transmitted correctly or dropped due to internal MAC transmit error, for example, underrun.
0x14	32	BadFramesReceived	The number of bad Ethernet frames received
0x18	32	BroadcastFramesReceived	The number of good frames received that had a Broadcast destination MAC address
0x1C	32	MulticastFramesReceived	The number of good frames received that had a Multicast destination MAC address <b>NOTE:</b> This does <i>not</i> include IEEE 802.3 Flow Control messages as they are considered MAC Control messages.
0x20	32	Frames64Octets	The total number of received and transmitted, Good and Bad frames that are 64 bytes in size or are between the minimum-size (as specified in RxMFS in the MFSR register) and 64 bytes. <b>NOTE:</b> This does <i>not</i> include MAC Control frames.
0x24	32	Frames65to127Octets	The total number of received and transmitted, Good and Bad frames that are 65 to 127 bytes in size. <b>NOTE:</b> This does <i>not</i> include MAC Control frames.
0x28	32	Frames128to255Octets	The total number of received and transmitted, Good and Bad frames that are 128 to 255 bytes in size. <b>NOTE:</b> This does <i>not</i> include MAC Control frames.
0x2C	32	Frames256to511Octets	The total number of received and transmitted, Good and Bad frames that are 256 to 511 bytes in size. <b>NOTE:</b> This does <i>not</i> include MAC Control frames.
0x30	32	Frames512to1023Octets	The total number of received and transmitted, Good and Bad frames that are 512 to 1023 bytes in size. <b>NOTE:</b> This does <i>not</i> include MAC Control frames.
0x34	32	Frames1024toMaxOctets	The total number of received and transmitted, Good and Bad frames that are more than 1023 bytes in size and less than the MRU. <b>NOTE:</b> This does not include MAC Control frames.
0x38	64	GoodOctetsSent	The sum of lengths of all good Ethernet frames sent from this MAC. This does not include IEEE 802.3 Flow Control frames NOR packets dropped due to excessive collision NOR packets with an Tx Error Event.
0x40	32	GoodFramesSent	The number of Ethernet frames sent from this MAC. This does not include IEEE 802.3 Flow Control frames NOR packets dropped due to excessive collision NOR packets with an Tx Error Event.

**Table 499: MAC MIB Counters (Continued)**

Offset: Port0: 0x73000–0x7307C, Port1: 0x77000–0x7707C

Offset	Width	Counter Name	Description
0x44	32	ExcessiveCollision	The number of frames dropped in the transmit MAC due to excessive collision condition. This counter will not be incremented if the PSCR's <Retr_forever> bit is set. This is applicable for half-duplex mode only.
0x48	32	MulticastFramesSent	The number of good frames sent that had a Multicast destination MAC address. <b>NOTE:</b> This does NOT include IEEE 802.3 Flow Control messages, as they are considered MAC Control messages, NOR does it include packets with an Tx Error Event. This counter counts frames of all sizes, including frames smaller than the Minimal Frame Size and frames larger than the MRU.
0x4C	32	BroadcastFramesSent	The number of good frames sent that had a Broadcast destination MAC address. This does not include IEEE 802.3 Flow Control frames NOR packets dropped due to excessive collision NOR packets with an Tx Error Event. <b>NOTE:</b> This counter counts frames of all sizes, including frames smaller than the Minimal Frame Size and frames larger than the MRU.
0x50	32	UnrecogMACControl Received	The number of received MAC Control frames that have an opcode different than 00-01.
0x58	32	GoodFCReceived	The number of good flow control messages received
0x5C	32	BadFCReceived	The number of bad flow control frames received
0x60	32	Undersize	The number of undersize packets received
0x54	32	FCSent	The number of flow control frames sent
0x64	32	Fragments	The number of fragments received
0x68	32	Oversize	The number of oversize packets received
0x6C	32	Jabber	The number of jabber packets received
0x70	32	MACRcvError	The number of Rx Error events seen by the receive side of the MAC
0x74	32	BadCRC	The number CRC error events
0x78	32	Collisions	The number of collision events seen by the MAC
0x7C	32	Late Collision	The number of late collisions seen by the MAC



**Note** The IEEE 802.3 SingleCollisionFrames and IEEE 802.3 MultipleCollisionFrames are not implemented.

## A.9 USB 2.0 Registers

The following table provides a summarized list of all of the USB 2.0 registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 500: Register Map Table for the USB 2.0 Registers**

Register Name	Offset	Table and Page
<b>USB 2.0 Bridge Address Decoding Registers</b>		
USB 2.0 Window0 Control Register	0x50320	<a href="#">Table 501, p. 625</a>
USB 2.0 Window0 Base Register	0x50324	<a href="#">Table 502, p. 626</a>
USB 2.0 Window1 Control Register	0x50330	<a href="#">Table 503, p. 626</a>
USB 2.0 Window1 Base Register	0x50334	<a href="#">Table 504, p. 627</a>
USB 2.0 Window2 Control Register	0x50340	<a href="#">Table 505, p. 627</a>
USB 2.0 Window2 Base Register	0x50344	<a href="#">Table 506, p. 628</a>
USB 2.0 Window3 Control Register	0x50350	<a href="#">Table 507, p. 628</a>
USB 2.0 Window3 Base Register	0x50354	<a href="#">Table 508, p. 629</a>
<b>USB 2.0 Bridge Control and Status Registers</b>		
USB 2.0 Bridge Control Register	0x50300	<a href="#">Table 509, p. 629</a>
<b>USB 2.0 Bridge Interrupt and Error Registers</b>		
USB 2.0 Bridge Interrupt Cause Register	0x50310	<a href="#">Table 510, p. 629</a>
USB 2.0 Bridge Interrupt Mask Register	0x50314	<a href="#">Table 511, p. 630</a>
USB 2.0 Bridge Error Address Register	0x5031C	<a href="#">Table 512, p. 630</a>
<b>USB 2.0 PHY Registers</b>		
USB 2.0 PHY Configuration0 Register	0x50360	<a href="#">Table 513, p. 630</a>
USB 2.0 Power Control Register	0x50400	<a href="#">Table 514, p. 631</a>

### A.9.1 USB 2.0 Bridge Address Decoding Registers

**Table 501: USB 2.0 Window0 Control Register**

Offset: 0x50320

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x0	Window0 Enable 0 = Disable 1 = Enable
1	WrBL	RW 0x0	USB to Mbus Burst Write Limit 0 = No limit 1 = Limit: Limit writes not to cross 32B boundary
3:2	Reserved	RSVD 0x0	Reserved



**Table 501: USB 2.0 Window0 Control Register (Continued)**  
Offset: 0x50320

Bit	Field	Type/InitVal	Description
7:4	Target	RW 0x0	Specifies the target interface associated with this window. See the Units IDs and Attributes table. <b>NOTE:</b> Must be configured to the SDRAM Controller.
15:8	Attr	RW 0x0	Specifies the target interface attributes associated with this window. See the Units IDs and Attributes table.
31:16	Size	RW 0x0	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as sequence of 1s followed by sequence of 0's. The number of 1s specifies the size of the window (for example, a value of 0x00FF specifies 256 x 64k = 16 MB).

**Table 502: USB 2.0 Window0 Base Register**  
Offset: 0x50324

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x0	Base Address Used with the size field to set the address window size and location. Corresponds to transaction address[31:16]

**Table 503: USB 2.0 Window1 Control Register**  
Offset: 0x50330

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x0	Window1 Enable 0 = Disable 1 = Enable
1	WrBL	RW 0x0	USB to Mbus Burst Write Limit 0 = No limit: No limit 1 = Limit: Limit writes not to cross 32B boundary
3:2	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the target interface associated with this window. See the Units IDs and Attributes table. <b>NOTE:</b> Must be configured to the SDRAM Controller.

**Table 503: USB 2.0 Window1 Control Register (Continued)**  
Offset: 0x50330

Bit	Field	Type/InitVal	Description
15:8	Attr	RW 0x0	Specifies the target interface attributes associated with this window. See the Units IDs and Attributes table.
31:16	Size	RW 0x0	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as sequence of 1s followed by sequence of 0s. The number of 1s specifies the size of the window (for example, a value of 0x00FF specifies 256x64k = 16 MB).

**Table 504: USB 2.0 Window1 Base Register**  
Offset: 0x50334

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x0	Base Address Used with the size field to set the address window size and location. Corresponds to transaction address[31:16]

**Table 505: USB 2.0 Window2 Control Register**  
Offset: 0x50340

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x0	Window2 Enable 0 = Disable 1 = Enable
1	WrBL	RW 0x0	USB to Mbus Burst Write Limit 0 = No limit 1 = Limit: Limit writes not to cross 32B boundary
3:2	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the target interface associated with this window. See the Units IDs and Attributes table. <b>NOTE:</b> Must be configured to the SDRAM Controller.
15:8	Attr	RW 0x0	Specifies the target interface attributes associated with this window. See the Units IDs and Attributes table.



**Table 505: USB 2.0 Window2 Control Register (Continued)**  
Offset: 0x50340

Bit	Field	Type/InitVal	Description
31:16	Size	RW 0x0	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as sequence of 1s followed by sequence of 0s. The number of 1s specifies the size of the window (for example, a value of 0x00FF specifies 256x64k = 16 MB).

**Table 506: USB 2.0 Window2 Base Register**  
Offset: 0x50344

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x0	Base Address Used with the size field to set the address window size and location. Corresponds to transaction address[31:16]

**Table 507: USB 2.0 Window3 Control Register**  
Offset: 0x50350

Bit	Field	Type/InitVal	Description
0	win_en	RW 0x0	Window3 Enable 0 = Disable 1 = Enable
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the target interface associated with this window. See the Units IDs and Attributes table.
15:8	Attr	RW 0x0	Specifies the target interface attributes associated with this window. See the Units IDs and Attributes table.
31:16	Size	RW 0x0	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as sequence of 1s followed by sequence of 0s. The number of 1's specifies the size of the window (for example, a value of 0x00ff specifies 256x64k = 16 MB).

**Table 508: USB 2.0 Window3 Base Register**  
**Offset: 0x50354**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x0	Base Address Used with the size field to set the address window size and location. Corresponds to transaction address[31:16]

## A.9.2 USB 2.0 Bridge Control and Status Registers

**Table 509: USB 2.0 Bridge Control Register**  
**Offset: 0x50300**

Bit	Field	Type/InitVal	Description
3:0	Reserved	RO 0x0	Reserved
4	BS	RW 0x0	USB Core Byte Swap 0 = Byte swap: Byte swap over 64-bit qword, upon USB core read/write access from memory. 1 = No byte swap
31:5	Reserved	RSVD 0x0	Reserved

## A.9.3 USB 2.0 Bridge Interrupt and Error Registers

**Table 510: USB 2.0 Bridge Interrupt Cause Register**  
**Offset: 0x50310**

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> All cause bits are clear only. They are set to `1? upon an interrupt event and cleared when the software writes a value of 0. Writing 1 has no affect.			
0	AddrDecErr	RWOC 0x0	Address Decoding Error Asserted upon address decoding error
1	HOF	RWOC 0x0	USB Host Underflow
2	DOF	RWOC 0x0	USB Host/Device overflow
3	DUF	RWOC 0x0	USB Device underflow
4	Reserved	RWOC 0x0	Reserved



**Table 510: USB 2.0 Bridge Interrupt Cause Register (Continued)**  
Offset: 0x50310

Bit	Field	Type/InitVal	Description
6:5	Reserved	RW0C 0x0	Reserved
31:7	Reserved	RO 0x0	Reserved

**Table 511: USB 2.0 Bridge Interrupt Mask Register**  
Offset: 0x50314

Bit	Field	Type/InitVal	Description
3:0	Mask	RW 0x0	If set to 1, the related interrupt is enabled.
6:4	Reserved	RW 0x0	Reserved
31:7	Reserved	RSVD 0x0	Reserved

**Table 512: USB 2.0 Bridge Error Address Register**  
Offset: 0x5031C

Bit	Field	Type/InitVal	Description
31:0	ErrAddr	RO 0x0	Error Address Latched upon any of the address decoding errors (address miss, multiple hit). Once the address is latched, no new address is latched until SW reads it (Read access to USB 2.0 Bridge Error Address Register).

## A.9.4 USB 2.0 PHY Registers

**Table 513: USB 2.0 PHY Configuration0 Register**  
Offset: 0x50360

Bit	Field	Type/InitVal	Description
6:0	StartIPG	RW 0x15	Inter-Packet Gap
7	Reserved	RW 0x0	Reserved
14:8	NonStartIPG	RW 0xD	Start of Frame Inter Packet Gap
15	Reserved	RSVD 0x0	Reserved
31:16	Phy_RSVD	RW 0x3F00	Connected to PHY Reserved input [15:0]

**Table 514: USB 2.0 Power Control Register**  
**Offset: 0x50400**

<b>Bit</b>	<b>Field</b>	<b>Type/InitVal</b>	<b>Description</b>
0	Pu	RW 0x1	Input Power Up
1	PuPll	RW 0x1	Input Power Up PLL
2	SUSPENDM	RW 0x1	Input SUSPENDM
3	VBUS_PWR_FAULT	RW 0x0	Vbus Power Fault Connect to the Core, not to the PHY
4	PWRCTL_WAKEUP	RW 0x0	USB Power Control Wake Up Connect to the Core, not to the PHY
5	Reserved	RW 0x1	Reserved
7:6	Reserved	RSVD 0x1	Reserved
8	REG_ARC_DPPM_MODE	RW 0x1	Mux is in the USB PHY. 0 = Register: Use register programmed pulldown. 1 = Controller core: Use dp_pulldown and dm_pulldown from controller core.
9	REG_DP_PULLDOWN	RW 0x0	Register DP Pull 0 = No DP pulldown 1 = Pulldown DP
10	REG_DM_PULLDOWN	RW 0x0	Register DM Pull 0 = No DM pulldown 1 = Pulldown DM
22:11	Reserved	RSVD 0x0	Reserved
23	utmi_sessend	RW 0x0	UTMI Session End
24	utmi_vbus_valid	RW 0x1	UTMI Vbus Valid
25	utmi_avalid	RW 0x1	UTMI A Valid
26	utmi_bvalid	RW 0x1	UTMI B Valid
27	TX_BIT_STUFF	RW 0x1	Transmit Bit Stuff
31:28	Reserved	RSVD 0xF	Reserved

## A.9.5 USB Controller Registers

**NOTE: USB 2.0 Controller Registers (0x50000–0x502FF):** refer to *ARC USB-HS OTG High-Speed USB On-The-Go Controller Core V 4.0.1 Reference*. The base address for the controller registers is 0x50000. The offsets remain the same as in the above document.

**Table 515: USB Controller Register Map (Offsets: 0x50000–0x502FF)**

Register	Offset
ID	0x50000
HWGENERAL	0x50004
HWHOST	0x50008
HWDEVICE	0x5000C
HWTXBUF	0x50010
HWRXBUF	0x50014
HWTTTXBUF	0x50018
HWTRRXBUF	0x5001C
Reserved	0x50020–0x500FC
CAPLENGTH	0x50100
Reserved	0x50101
HCIVERSION	0x50102
HCSPARAMS	0x50104
HCCPARAMS	0x50108
Reserved	0x5010C–0x5011F
DCIVERSION	0x50120
Reserved	0x50122
DCCPARAMS	0x50124
Reserved	0x50128–0x5013C
USBCMD	0x50140
USBSTS	0x50144
USBINTR	0x50148
FRINDEX	0x5014C
Reserved	0x50150
PERIODICLISTBASE / Device Addr	0x50154
ASYNCLISTADDR / Endpointlist Addr	0x50158
TTCTRL	0x5015C
BURSTSIZE	0x50160
TXFILLTUNING	0x50164
TXTTFILLTUNING	0x50168
Reserved	0x5016C
Reserved	0x50170–0x5017C
CONFIGFLAG	0x50180
PORTSC1	0x50184
OTGSC	0x501A4
USBMODE	0x501A8

**Table 515: USB Controller Register Map (Offsets: 0x50000–0x502FF) (Continued)**

<b>Register</b>	<b>Offset</b>
ENPDTSETUPSTAT	0x501A
ENDPTPRIME	0x501B
ENDPTFLUSH	0x501B4
ENDPTSTATUS	0x501B8
ENDPTCOMPLETE	0x501BC
ENDPTCTRL0	0x501C0
ENDPTCTRL1	0x501C4
ENDPTCTRL2	0x501C8
ENDPTCTRL3	0x501CC

## A.10 Cryptographic Engine and Security Accelerator (CESA) Registers

NOTE. Access to the Cryptographic engine and Security accelerator registers is limited to 32b word access (byte and half word accesses are not supported).

The following table provides a summarized list of all of the Cryptographic Engine and Security Accelerator (CESA) registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 516: Register Map Table for the Cryptographic Engine and Security Accelerator (CESA) Registers**

Register Name	Offset	Table and Page
<b><i>AES Decryption Interface Registers</i></b>		
AES Decryption Key Column 7 Register	0x3DDC0	<a href="#">Table 517, p. 636</a>
AES Decryption Key Column 6 Register	0x3DDC4	<a href="#">Table 518, p. 636</a>
AES Decryption Key Column 5 Register	0x3DDC8	<a href="#">Table 519, p. 636</a>
AES Decryption Key Column 4 Register	0x3DDCC	<a href="#">Table 520, p. 636</a>
AES Decryption Key Column 3 Register	0x3DDD0	<a href="#">Table 521, p. 636</a>
AES Decryption Key Column 2 Register	0x3DDD4	<a href="#">Table 522, p. 637</a>
AES Decryption Key Column 1 Register	0x3DDD8	<a href="#">Table 523, p. 637</a>
AES Decryption Key Column 0 Register	0x3DDDC	<a href="#">Table 524, p. 637</a>
AES Decryption Data In/Out Column 3 Register	0x3DDE0	<a href="#">Table 525, p. 637</a>
AES Decryption Data In/Out Column 2 Register	0x3DDE4	<a href="#">Table 526, p. 637</a>
AES Decryption Data In/Out Column 1 Register	0x3DDE8	<a href="#">Table 527, p. 638</a>
AES Decryption Data In/Out Column 0 Register	0x3DDEC	<a href="#">Table 528, p. 638</a>
AES Decryption Command Register	0x3DDF0	<a href="#">Table 529, p. 638</a>
<b><i>AES Encryption Interface Registers</i></b>		
AES Encryption Key Column 7 Register	0x3DD80	<a href="#">Table 530, p. 639</a>
AES Encryption Key Column 6 Register	0x3DD84	<a href="#">Table 531, p. 639</a>
AES Encryption Key Column 5 Register	0x3DD88	<a href="#">Table 532, p. 639</a>
AES Encryption Key Column 4 Register	0x3DD8C	<a href="#">Table 533, p. 639</a>
AES Encryption Key Column 3 Register	0x3DD90	<a href="#">Table 534, p. 640</a>
AES Encryption Key Column 2 Register	0x3DD94	<a href="#">Table 535, p. 640</a>
AES Encryption Key Column 1 Register	0x3DD98	<a href="#">Table 536, p. 640</a>
AES Encryption Key Column 0 Register	0x3DD9C	<a href="#">Table 537, p. 640</a>
AES Encryption Data In/Out Column 3 Register	0x3DDA0	<a href="#">Table 538, p. 640</a>
AES Encryption Data In/Out Column 2 Register	0x3DDA4	<a href="#">Table 539, p. 641</a>
AES Encryption Data In/Out Column 1 Register	0x3DDA8	<a href="#">Table 540, p. 641</a>
AES Encryption Data In/Out Column 0 Register	0x3DDAC	<a href="#">Table 541, p. 641</a>
AES Encryption Command Register	0x3DDB0	<a href="#">Table 542, p. 641</a>
<b><i>DES Engine Registers</i></b>		
DES Initial Value Low Register	0x3DD40	<a href="#">Table 543, p. 642</a>
DES Initial Value High Register	0x3DD44	<a href="#">Table 544, p. 642</a>
DES Key0 Low Register	0x3DD48	<a href="#">Table 545, p. 642</a>
DES Key0 High Register	0x3DD4C	<a href="#">Table 546, p. 642</a>
DES Key1 Low Register	0x3DD50	<a href="#">Table 547, p. 643</a>

**Table 516: Register Map Table for the Cryptographic Engine and Security Accelerator (CESA) Registers**

Register Name	Offset	Table and Page
DES Key1 High Register	0x3DD54	Table 548, p. 643
DES Command Register	0x3DD58	Table 549, p. 643
DES Key2 Low Register	0x3DD60	Table 550, p. 644
DES Key2 High Register	0x3DD64	Table 551, p. 644
DES Data Buffer Low Register	0x3DD70	Table 552, p. 644
DES Data Buffer High Register	0x3DD74	Table 553, p. 645
DES Data Out Low Register	0x3DD78	Table 554, p. 645
DES Data Out High Register	0x3DD7C	Table 555, p. 645
<b>Interrupt Cause Registers</b>		
Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register	0x3DE20	Table 556, p. 645
Cryptographic Engines and Security Accelerator Interrupt Mask Register	0x3DE24	Table 557, p. 647
<b>Security Accelerator Registers</b>		
Security Accelerator Command Register	0x3DE00	Table 558, p. 647
Security Accelerator Descriptor Pointer Register	0x3DE04	Table 559, p. 647
Security Accelerator Configuration Register	0x3DE08	Table 560, p. 648
Security Accelerator Status Register	0x3DE0C	Table 561, p. 648
<b>SHA-1 and MD5 Interface Registers</b>		
SHA-1/MD5 Initial Value/Digest A Register	0x3DD00	Table 562, p. 649
SHA-1/MD5 Initial Value/Digest B Register	0x3DD04	Table 563, p. 649
SHA-1/MD5 Initial Value/Digest C Register	0x3DD08	Table 564, p. 649
SHA-1/MD5 Initial Value/Digest D Register	0x3DD0C	Table 565, p. 649
SHA-1 Initial Value/Digest E Register	0x3DD10	Table 566, p. 650
SHA-1/MD5 Authentication Command Register	0x3DD18	Table 567, p. 650
SHA-1/MD5 Bit Count Low Register	0x3DD20	Table 568, p. 651
SHA-1/MD5 Bit Count High Register	0x3DD24	Table 569, p. 651
SHA-1/MD5 Data In Register	0x3DD38	Table 570, p. 652
<b>TDMA Address Decoding Registers</b>		
Base Address Register (n=0–3)	BAR0: 0x30A00, BAR1: 0x30A08, BAR2: 0x30A10, BAR3: 0x30A18	Table 571, p. 652
Window Control Register (n=0–3)	SR0: 0x30A04, SR1: 0x30A0C, SR2: 0x30A14, SR3: 0x30A1C	Table 572, p. 652
<b>TDMA Control Registers</b>		
Control Register	0x30840	Table 573, p. 653
<b>TDMA Descriptor Registers</b>		
TDMA Byte Count Register	0x30800	Table 574, p. 654
TDMA Source Address Register	0x30810	Table 575, p. 655
TDMA Destination Address Register	0x30820	Table 576, p. 655
Next Descriptor Pointer Register	0x30830	Table 577, p. 655
Current Descriptor Pointer Register	0x30870	Table 578, p. 655
<b>TDMA Interrupt Registers</b>		



**Table 516: Register Map Table for the Cryptographic Engine and Security Accelerator (CESA) Registers**

Register Name	Offset	Table and Page
TDMA Error Cause Register	0x308C8	<a href="#">Table 579, p. 656</a>
TDMA Error Mask Register	0x308CC	<a href="#">Table 580, p. 656</a>

## A.10.1 AES Decryption Interface Registers

**Table 517: AES Decryption Key Column 7 Register**

Offset: 0x3DDC0

Bit	Field	Type/InitVal	Description
31:0	AesDecKeyCol7	RW 0x0	Contains Column 7 of the AES decryption key

**Table 518: AES Decryption Key Column 6 Register**

Offset: 0x3DDC4

Bit	Field	Type/InitVal	Description
31:0	AesDecKeyCol6	RW 0x0	Contains Column 6 of the AES decryption key

**Table 519: AES Decryption Key Column 5 Register**

Offset: 0x3DDC8

Bit	Field	Type/InitVal	Description
31:0	AesDecKeyCol5	RW 0x0	Contains Column 5 of the AES decryption key

**Table 520: AES Decryption Key Column 4 Register**

Offset: 0x3DDCC

Bit	Field	Type/InitVal	Description
31:0	AesDecKeyCol4	RW 0x0	Contains Column 4 of the AES decryption key

**Table 521: AES Decryption Key Column 3 Register**

Offset: 0x3DDD0

Bit	Field	Type/InitVal	Description
31:0	AesDecKeyCol3	RW 0x0	Contains Column 3 of the AES decryption key

**Table 522: AES Decryption Key Column 2 Register**  
**Offset: 0x3DDD4**

Bit	Field	Type/InitVal	Description
31:0	AesDecKeyCol2	RW 0x0	Contains Column 2 of the AES decryption key

**Table 523: AES Decryption Key Column 1 Register**  
**Offset: 0x3DDD8**

Bit	Field	Type/InitVal	Description
31:0	AesDecKeyCol1	RW 0x0	Contains Column 1 of the AES decryption key

**Table 524: AES Decryption Key Column 0 Register**  
**Offset: 0x3DDDC**

Bit	Field	Type/InitVal	Description
31:0	AesDecKeyCol0	RW 0x0	Contains Column 0 of the AES decryption key

**Table 525: AES Decryption Data In/Out Column 3 Register**  
**Offset: 0x3DDE0**

Bit	Field	Type/InitVal	Description
31:0	AesDecDatCol3	RW 0x0	At first this field contains Column 3 of the input data block to be decrypted. When the AES completes the calculation, this field will contain the Column 3 of the AES result.

**Table 526: AES Decryption Data In/Out Column 2 Register**  
**Offset: 0x3DDE4**

Bit	Field	Type/InitVal	Description
31:0	AesDecDatCol2	RW 0x0	At first this field contains Column 2 of the input data block to be decrypted. When the AES completes the calculation, this field will contain the Column 2 of the AES result.



**Table 527: AES Decryption Data In/Out Column 1 Register**  
**Offset: 0x3DDE8**

Bit	Field	Type/InitVal	Description
31:0	AesDecDatCol1	RW 0x0	At first this field contains Column 1 of the input data block to be decrypted. When the AES completes the calculation, this field will contain the Column 1 of the AES result.

**Table 528: AES Decryption Data In/Out Column 0 Register**  
**Offset: 0x3DDEC**

Bit	Field	Type/InitVal	Description
31:0	AesDecDatCol0	RW 0x0	At first this field contains column 0 of the input data block to be decrypted. When the AES completes the calculation, this field will contain the Column 0 of the AES result.

**Table 529: AES Decryption Command Register**  
**Offset: 0x3DDF0**

Bit	Field	Type/InitVal	Description
1:0	AesDecKeyMode	RW 0x0	These bits specify what AES128 key size is used. 0 = 128-bit key 1 = 192-bit key 2 = 256-bit key 3 = Reserved
2	AesDecMakeKey	RW 0x0	This bit controls whether the decryption key is calculated in the engine prior to the data decryption. 0 = NoDecrypt: No decryption key calculation 1 = Decrypt: Decryption key calculation
3	Reserved	RSVD 0x0	Reserved
4	DataByteSwap	RW 0x0	This bit controls whether data byte swap is activated on input. 0 = No byte swap 1 = Byte swap
7:5	Reserved	RSVD 0x0	Reserved
8	OutByteSwap	RW 0x0	This bit controls whether byte swap is activated for output. 0 = No byte swap 1 = Byte swap
29:9	Reserved	RSVD 0x0	Reserved

**Table 529: AES Decryption Command Register (Continued)**  
Offset: 0x3DDF0

Bit	Field	Type/InitVal	Description
30	AesDecKeyReady	RO 0x0	This bit is set to 1 whenever the key generation process is done. This bit is cleared to 0 whenever any of the AES Dec Key registers is written.
31	Termination	RO 0x1	This bit is set by the engine to indicate completion of a AES calculation process. Any write to the decryption engine will clear this bit.

## A.10.2 AES Encryption Interface Registers

**Table 530: AES Encryption Key Column 7 Register**  
Offset: 0x3DD80

Bit	Field	Type/InitVal	Description
31:0	AesEncKeyCol7	RW 0x0	Contains Column 7 of the AES encryption key or Column 7 of the decryption key when AES Key Read Mode is set.

**Table 531: AES Encryption Key Column 6 Register**  
Offset: 0x3DD84

Bit	Field	Type/InitVal	Description
31:0	AesEncKeyCol6	RW 0x0	Contains Column 6 of the AES encryption key or Column 6 of the decryption key when AES Key Read Mode is set.

**Table 532: AES Encryption Key Column 5 Register**  
Offset: 0x3DD88

Bit	Field	Type/InitVal	Description
31:0	AesEncKeyCol5	RW 0x0	Contains Column 5 of the AES encryption key or Column 5 of the decryption key when AES Key Read Mode is set.

**Table 533: AES Encryption Key Column 4 Register**  
Offset: 0x3DD8C

Bit	Field	Type/InitVal	Description
31:0	AesEncKeyCol4	RW 0x0	Contains Column 4 of the AES encryption key or Column 4 of the decryption key when AES Key Read Mode is set.

**Table 534: AES Encryption Key Column 3 Register**  
Offset: 0x3DD90

Bit	Field	Type/InitVal	Description
31:0	AesEncKeyCol3	RW 0x0	Contains Column 3 of the AES encryption key or Column 3 of the decryption key when AES Key Read Mode is set.

**Table 535: AES Encryption Key Column 2 Register**  
Offset: 0x3DD94

Bit	Field	Type/InitVal	Description
31:0	AesEncKeyCol2	RW 0x0	Contains Column 2 of the AES encryption key or Column 2 of the decryption key when AES Key Read Mode is set.

**Table 536: AES Encryption Key Column 1 Register**  
Offset: 0x3DD98

Bit	Field	Type/InitVal	Description
31:0	AesEncKeyCol1	RW 0x0	Contains Column 1 of the AES encryption key or Column 1 of the decryption key when AES Key Read Mode is set.

**Table 537: AES Encryption Key Column 0 Register**  
Offset: 0x3DD9C

Bit	Field	Type/InitVal	Description
31:0	AesEncKeyCol0	RW 0x0	Contains Column 0 of the AES encryption key or Column 0 of the decryption key when AES Key Read Mode is set.

**Table 538: AES Encryption Data In/Out Column 3 Register**  
Offset: 0x3DDA0

Bit	Field	Type/InitVal	Description
31:0	AesEncDatCol3	RW 0x0	At first this field contains Column 3 of the input data block to be encrypted. When the AES completes the calculation, this field will contain the Column 3 of the AES result.

**Table 539: AES Encryption Data In/Out Column 2 Register**  
**Offset: 0x3DDA4**

Bit	Field	Type/InitVal	Description
31:0	AesEncDatCol2	RW 0x0	At first this field contains Column 2 of the input data block to be encrypted. When the AES completes the calculation, this field will contain the Column 2 of the AES result.

**Table 540: AES Encryption Data In/Out Column 1 Register**  
**Offset: 0x3DDA8**

Bit	Field	Type/InitVal	Description
31:0	AesEncDatCol1	RW 0x0	At first this field contains Column 1 of the input data block to be encrypted. When the AES completes the calculation, this field will contain the Column 1 of the AES result.

**Table 541: AES Encryption Data In/Out Column 0 Register**  
**Offset: 0x3DDAC**

Bit	Field	Type/InitVal	Description
31:0	AesEncDatCol0	RW 0x0	At first this field contains Column 0 of the input data block to be encrypted. When the AES completes the calculation, this field will contain the Column 0 of the AES result.

**Table 542: AES Encryption Command Register**  
**Offset: 0x3DDB0**

Bit	Field	Type/InitVal	Description
1:0	AesEncKeyMode	RW 0x0	This field specifies the AES128 key size used. 0 = 128-bit key 1 = 192-bit key 2 = 256-bit key 3 = Reserved
3:2	Reserved	RSVD 0x0	Reserved
4	DataByteSwap	RW 0x0	This bit controls whether data byte swap is activated on input. 0 = No byte swap 1 = Byte swap
7:5	Reserved	RSVD 0x0	Reserved



**Table 542: AES Encryption Command Register (Continued)**  
Offset: 0x3DDB0

Bit	Field	Type/InitVal	Description
8	OutByteSwap	RW 0x0	This bit controls whether byte swap is activated for output. 0 = No byte swap 1 = Byte swap
30:9	Reserved	RSVD 0x0	Reserved
31	Termination	RO 0x1	This bit is set by the engine to indicate completion of a AES calculation process. Any write to the encryption engine will clear this bit.

### A.10.3 DES Engine Registers

**Table 543: DES Initial Value Low Register**  
Offset: 0x3DD40

Bit	Field	Type/InitVal	Description
31:0	DESIVLo	RW 0x0	Contains low bits of the Initial Value in CBC mode. (This register is ignored in ECB mode).

**Table 544: DES Initial Value High Register**  
Offset: 0x3DD44

Bit	Field	Type/InitVal	Description
31:0	DESIVHi	RW 0x0	Contains high bits of the Initial Value in CBC mode. (This register is ignored in ECB mode.)

**Table 545: DES Key0 Low Register**  
Offset: 0x3DD48

Bit	Field	Type/InitVal	Description
31:0	DESKey0Lo	RW 0x0	Contains the low bits of the DES key or of the first key of the Triple DES keys.

**Table 546: DES Key0 High Register**  
Offset: 0x3DD4C

Bit	Field	Type/InitVal	Description
31:0	DESKey0Hi	RW 0x0	Contains the high bits of the DES key or of the first key of the Triple DES keys.

**Table 547: DES Key1 Low Register**  
**Offset: 0x3DD50**

Bit	Field	Type/InitVal	Description
31:0	DESKey1Lo	RW 0x0	Contains the low bits of the second key of the Triple DES keys. (This register is ignored in DES mode.)

**Table 548: DES Key1 High Register**  
**Offset: 0x3DD54**

Bit	Field	Type/InitVal	Description
31:0	DESKey1Hi	RW 0x0	Contains the high bits of the second key of the Triple DES keys. (This register is ignored in DES mode.)

**Table 549: DES Command Register**  
**Offset: 0x3DD58**

Bit	Field	Type/InitVal	Description
0	Direction	RW 0x0	This bit controls the direction of the operation: Encryption or Decryption. 0 = Encryption 1 = Decryption
1	Algorithm	RW 0x0	This bit controls whether the DES or Triple DES algorithm is used. 0 = DES 1 = 3DES: Triple DES
2	TripleDESMODE	RW 0x0	This bit controls the Triple DES encryption/decryption mode. 0 = EEE 1 = EDE
3	DESMODE	RW 0x0	This bit controls the DEC encryption/decryption mode. 0 = ECB 1 = CBC
4	DataByteSwap	RW 0x0	This bit controls whether data byte swap is activated on input. 0 = No byte swap 1 = Byte swap
5	Reserved	RSVD 0x0	Reserved
6	IVByteSwap	RW 0x0	This bit controls whether initial value byte swap is activated. 0 = No byte swap 1 = Byte swap
7	Reserved	RSVD 0x0	Reserved



**Table 549: DES Command Register (Continued)**  
Offset: 0x3DD58

Bit	Field	Type/InitVal	Description
8	OutByteSwap	RSVD 0x0	This bit controls whether byte swap is activated for output. 0 = No byte swap 1 = Byte swap
28:9	Reserved	RSVD 0x0	Reserved
29	WriteAllow	RW 0x1	This bit indicates that the host can write data to the engine. 0 = NotAllowed: Write data to engine not allowed 1 = Allowed: Write data to engine allowed
30	AllTermination	RW 0x1	This bit indicates to the host that the encryption calculation has been completed, the encryption parameters may be updated and data may be written.
31	Termination	RO 0x1	This bit is set by the engine to indicate completion of a DES calculation process. Any write to the encryption engine will clear this bit.

**Table 550: DES Key2 Low Register**  
Offset: 0x3DD60

Bit	Field	Type/InitVal	Description
31:0	DESKey2Lo	RW 0x0	Contains the low bits of the third key of the Triple DES keys. (This register is ignored in DES mode.)

**Table 551: DES Key2 High Register**  
Offset: 0x3DD64

Bit	Field	Type/InitVal	Description
31:0	DESKey2Hi	RW 0x0	Contains the high bits of the third key of the Triple DES keys. (This register is ignored in DES mode.)

**Table 552: DES Data Buffer Low Register**  
Offset: 0x3DD70

Bit	Field	Type/InitVal	Description
31:0	DataBufLo	WO 0x0	The host writes data blocks of low words to be encrypted/decrypted to this register.

**Table 553: DES Data Buffer High Register**  
Offset: 0x3DD74

Bit	Field	Type/InitVal	Description
31:0	DataBufHi	WO 0x0	The host writes data blocks of high words to be encrypted/decrypted to this register.

**Table 554: DES Data Out Low Register**  
Offset: 0x3DD78

Bit	Field	Type/InitVal	Description
31:0	DataOutLo	RO 0x0	When the DES (or the Triple DES) completes the calculation, this field will contain the low bits of the DES result.

**Table 555: DES Data Out High Register**  
Offset: 0x3DD7C

Bit	Field	Type/InitVal	Description
31:0	DataOutHi	RO 0x0	When the DES (or the Triple DES) completes the calculation, this field will contain the high bits of the DES result.

## A.10.4 Interrupt Cause Registers

**Table 556: Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register**  
Offset: 0x3DE20

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> The cryptographic engine has a dedicated Interrupt Cause register. This register is set by events occurring in the engine. Clearing this register's bits is done by writing 0 to the cause bits. Writing 1 to a bit has no effect. This register is shared by the DES and the Authentication engine.			
0	ZInt0	RW0C 0x0	This bit is the authentication termination clear indication. The interrupt is set when the authentication engine finishes the calculation process.
1	ZInt1	RW0C 0x0	This bit is the DES encryption all termination clear indication. The interrupt is set when the encryption engine finishes the calculation process.
2	Zin2	RW0C 0x0	This bit is the AES encryption termination clear indication. The interrupt is set when the AES encryption engine finishes the calculation process.
3	Zint3	RW0C 0x0	This bit is the AES decryption termination clear indication. The interrupt is set when the AES decryption engine finishes the calculation process.



**Table 556: Cryptographic Engine/Security Accelerator/TDMA Interrupt Cause Register (Continued)**  
Offset: 0x3DE20

Bit	Field	Type/InitVal	Description
4	ZInt4	RW0C 0x0	This bit is the encryption termination clear indication. The interrupt is set when the encryption engine finishes the calculation process.
5	AccInt0	RW0C 0x0	This bit is the Security accelerator session 0 termination clear indication. The interrupt is set when the Security accelerator session 0 completes its operation.
6	Reserved	RSVD 0x0	Reserved
7	AccAndTDMAInt	RW0C 0x0	Acceleration and TDMA Interrupt This bit is set to 1 when the entire security accelerator process is completed, including both encryption/authentication process and its associate TDMA operation. When the TDMA is configured to copy the outcome of the security accelerator process back to the DDR (that is, when the <ActivateTDMA> field in the Security Accelerator Configuration Register, <AccAndTDMAInt> is set to 1 after the TDMA completes copying the data back to the DDR). When the TDMA is NOT configured to copy the outcome of the security accelerator process back to the DDR (that is, when the <ActivateTDMA> field in the Security Accelerator Configuration Register is set to 0), <AccAndTDMAInt> is set after the security accelerator completes the process and data is valid in the local SRAM. <b>NOTE:</b> If ZDMA is configured to continuous mode, AccAndTDMAInt is asserted only with the completion of the last packet in the chain. AccAndTDMAInt_CM interrupt is set with the completion of each packet.
8	Reserved	RSVD 0x0	Reserved
9	TDMAComplinterrup t	RW0C 0x0	Indicates completion of TDMA operation.
10	TDMAOwnlinterrupt	RW0C 0x0	Indicates and ownership error in the TDMA.
11	AccAndTDMAInt_C M	RW0C 0x0	If the security accelerator is configured to accelerate continuous mode, this interrupt is set when finish processing of previous packet (fetch first TDMA descriptor of next packet).
12	Reserved	RSVD 0x0	Reserved
31:13	Reserved	RW0C 0x0	Reserved

**Table 557: Cryptographic Engines and Security Accelerator Interrupt Mask Register**  
**Offset: 0x3DE24**

Bit	Field	Type/InitVal	Description
11:0	Mask	RW 0x0	Mask bit for each cause bit Mask only affects the assertion of interrupt pins. It does not affect the setting of bits in the Cause register. 0 = Masked: Interrupt masked. 1 = Enabled: Interrupt enabled.
31:12	Reserved	RW 0x0	Must be 0

## A.10.5 Security Accelerator Registers

**Table 558: Security Accelerator Command Register**  
**Offset: 0x3DE00**

Bit	Field	Type/InitVal	Description
0	EnSecurityAccl	SC 0x0	Setting this bit activates an accelerator session. After operation completion, this bit is cleared to zero by the hardware. Writing zero to this bit has no effect. 0 = Session is idle. 1 = Session is set to active. 0 = Idle: Session 0 is idle. 1 = Active: Session 0 is set to active.
1	Reserved	RSVD 0x0	Reserved
2	DsSecurityAccl	RW 0x0	Disable accelerator This bit is self negated. When this bit is set to 1, the accelerator aborts the current command and then clears bits Acclnt0, Acclnt1.  <b>NOTE:</b> ARZ: Auto-Reset to Zero after the Acclnt0, Acclnt1bits are cleared.
31:3	Reserved	RO 0x0	Reserved

**Table 559: Security Accelerator Descriptor Pointer Register**  
**Offset: 0x3DE04**

Bit	Field	Type/InitVal	Description
15:0	SecurityAcclDescPtr 0	RW 0x0	Security accelerator descriptor pointer for session 0 (DWORD aligned) Bits [0], [1], [2], [13], [14] and [15] are reserved and are assumed to be and/are read as 0 regardless of programming.
31:16	Reserved	RW 0x0	Reserved

**Table 560: Security Accelerator Configuration Register**  
**Offset: 0x3DE08**

Bit	Field	Type/InitVal	Description
0	StopOnDecodeDigestErr	RW 0x1	Controls whether the engine stops when digest error in decode. 0 = No stop: Do not stop on digest decode error. 1 = Stop: Stop on digest decode error.
1	Reserved	RW 0x0	Must be 0.
6:2	Reserved	RSVD 0x0	
7	WaitForTDMA	RW 0x0	Wait for TDMA When set to 1, security accelerator is activated only when TDMA ownership error occurs. This bit also configures the TDMA to start working when the Security accelerator is enabled.
8	Reserved	RSVD 0x0	Reserved
9	ActivateTDMA	RW 0x0	Activation for TDMA When set to 1, the Security accelerator completes the current process.
10	Reserved	RSVD 0x0	Reserved
11	MultiPacketChainMode	RW 0x0	Security Acceleration Multi-Packet Chain Mode  <b>NOTE:</b> The Multi-Packet Chain mode may be enabled only if using Security Accelerator Enhanced Mode (<WaitForTDMA> and <ActivateTDMA> bits are set). 0 = Single packet: Security accelerator halts after processing a single packet. 1 = Multi-Packet: Security accelerator continues to process packets until TDMA reach NULL pointer (no more packets to process).
12	Reserved	RW 0x0	Must be 0
31:13	Reserved	RSVD 0x0	

**Table 561: Security Accelerator Status Register**  
**Offset: 0x3DE0C**

Bit	Field	Type/InitVal	Description
0	AccActive	RO 0x0	Session State This bit equals <AccInt>. 0 = Idle: Session 0 is idle 1 = Active: Session 0 is active

**Table 561: Security Accelerator Status Register (Continued)**  
Offset: 0x3DE0C

Bit	Field	Type/InitVal	Description
7:1	Reserved	RO 0x0	Reserved
8	DecodeDigestErr	RO 0x0	Signals a decode digest error during the session. This bit is cleared when the session is activated.
12:9	Reserved	RO 0x0	Reserved
31:13	AcclState	RO 0x0	Internal State of the accelerator

## A.10.6 SHA-1 and MD5 Interface Registers

**Table 562: SHA-1/MD5 Initial Value/Digest A Register**  
Offset: 0x3DD00

Bit	Field	Type/InitVal	Description
31:0	IVDigA	RW 0x67452301	IV A contains the first word of the Initial Value, and Digest A contains the first word of the digest.

**Table 563: SHA-1/MD5 Initial Value/Digest B Register**  
Offset: 0x3DD04

Bit	Field	Type/InitVal	Description
31:0	IVDigB	RW 0xEFCDAB89	IV B contains the second word of the Initial Value, and Digest B contains the second word of the digest.

**Table 564: SHA-1/MD5 Initial Value/Digest C Register**  
Offset: 0x3DD08

Bit	Field	Type/InitVal	Description
31:0	IVDigC	RW 0x98BADCCE	IV C contains the third word of the Initial Value, and Digest C contains the third word of the digest.

**Table 565: SHA-1/MD5 Initial Value/Digest D Register**  
Offset: 0x3DD0C

Bit	Field	Type/InitVal	Description
31:0	IVDigD	RW 0x10325476	IV D contains the fourth word of the Initial Value, and Digest D contains the fourth word of the digest.

**Table 566: SHA-1 Initial Value/Digest E Register**  
**Offset: 0x3DD10**

Bit	Field	Type/InitVal	Description
31:0	IVDigE	RW 0xC3D2E1F0	IV E contains the fifth word of the Initial Value, and Digest E contains the fifth word of the digest. <b>NOTE:</b> This register is only used in SHA-1, since SHA mode requires a 5-word initial value to produce the 5-word SHA signature.

**Table 567: SHA-1/MD5 Authentication Command Register**  
**Offset: 0x3DD18**

Bit	Field	Type/InitVal	Description
0	Algorithm	RW 0x0	This bit controls the mode of operation: SHA-1 or MD5. These are two different algorithms for calculating the authentication signature. They are described in the references. SHA calculation takes 85 clock cycles; where MD5 takes 65 clock cycles. SHA mode results in a 5-word signature (and a 5-word initial value is required) where MD5 results in a 4-word signature (and a 4-word initial value is required). The MD5 is byte swapped compared to the SHA. These algorithms differ in their complexity and security levels, and it is left to the user to choose the algorithm. 0 = MD5 1 = SHA1
1	Mode	RW 0x0	This bit controls whether the initial value is used or the operation continues from the last value. Both SHA and MD5 algorithms do a computational process on chunks of 512 bits where the last 64 bits in the last chunk are reserved for packet size. When a packet length is less than 448 bits, the host must add one bit of 1 to the end of the packet and pad it to 448-bit size with zeros. Then, the host adds a double word (64 bits) that contains the length. Then the chunk is ready for processing by the engine. Packets may be of arbitrary length (up to $2^{64}$ bits). They are broken into 512-bit chunks. The last chunk of the packet is padded to 448 bits as described above, and 64 bits representing packet length are added to make a 512-bit block. Prior to writing the first chunk of a packet, the host must select the Initial mode (0) in the command register. After the first chunk is processed, all the proceeding chunks of the packet must be processed using Continue mode (1). In Initial mode the engine starts processing the data block, using the initial values of the algorithm. In Continue mode the results of the previous calculation are used. The user may wish to share the engine for multiple packet signature calculations. This is done by calculating a chunk or chunks of a specific packet, reading the intermediate digest, and saving the digest in a memory. Then it is possible to start to process another packet. To continue processing the first packet, the host must write the intermediate digest that was saved in the memory, to the initial values registers and continue packet processing in Continue mode. 0 = Initial value: Use initial value 1 = Last value: Continue from the last value

**Table 567: SHA-1/MD5 Authentication Command Register (Continued)**  
Offset: 0x3DD18

Bit	Field	Type/InitVal	Description
2	DataByteSwap	RW 0x0	This bit controls whether data-byte swap is activated. Packet data written to the engine can be used as is, or swapped by the engine before processing. The main purpose of this field is for processing different notations of packet data--data may be annotated as Big Endian or Little Endian. 0 = No byte swap: Data to engine W0...W15 -- 0x01234567. 1 = Byte swap: Data to engine W0...W15 -- 0x67452301.
3	Reserved	RSVD 0x0	Reserved
4	IVByteSwap	RW 0x0	This bit controls whether initial value byte swap is activated. This is the same as the data swap, but only for initial values written to the IV/Digest registers. 0 = No byte swap 1 = Byte swap
30:5	Reserved	RSVD 0x0	Reserved
31	Termination	RO 0x1	This bit is set by the engine to indicate completion of a hash calculation process. Any write to the Authentication engine will clear this bit.

**Table 568: SHA-1/MD5 Bit Count Low Register**  
Offset: 0x3DD20

Bit	Field	Type/InitVal	Description
31:0	BitCntLo	WO 0x0	Fourteenth word of data in the array This register is accessed only when automatic padding is needed.

**Table 569: SHA-1/MD5 Bit Count High Register**  
Offset: 0x3DD24

Bit	Field	Type/InitVal	Description
31:0	BitCntHi	WO 0x0	Fifteenth word of data in the array This register is accessed only when automatic padding is needed.

**Table 570: SHA-1/MD5 Data In Register**

Offset: 0x3DD38

Bit	Field	Type/InitVal	Description
31:0	DataIn	WO 0x0	Words of the 512-bit hash block should be written to this register. With each write, the data in this field is pushed into the authentication engine's 16-word FIFO.

## A.10.7 TDMA Address Decoding Registers

**Table 571: Base Address Register (n=0–3)**

Offset: BAR0: 0x30A00, BAR1: 0x30A08, BAR2: 0x30A10, BAR3: 0x30A18

Bit	Field	Type/InitVal	Description
3:0	Reserved	RSVD 0x0	Reserved
7:4	Reserved	RO 0x0	Reserved
15:8	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x0	Window Base Address <b>NOTE:</b> The initial value for BARI1 is 0x1000

**Table 572: Window Control Register (n=0–3)**

Offset: SR0: 0x30A04, SR1: 0x30A0C, SR2: 0x30A14, SR3: 0x30A1C

Bit	Field	Type/InitVal	Description
0	Enable	RW 0x1	Window Enable <b>NOTE:</b> The initial value for SR2/3 is 0x0.
3:1	Reserved	RSVD 0x0	Reserved
7:4	TargetID	RW 0x0	Target Unit ID

**Table 572: Window Control Register (n=0–3) (Continued)**  
Offset: SR0: 0x30A04, SR1: 0x30A0C, SR2: 0x30A14, SR3: 0x30A1C

Bit	Field	Type/InitVal	Description
15:8	Attr	RW 0xE	<p>Attribute</p> <p>If TargetID is Dunit:  0x0e: SCS[0]  0x0d: SCS[1]  0x0b: SCS[2]  0x07: SCS[3]</p> <p>If TargetID is Runit:  0x1e: CS[0]  0x1d: CS[1]  0x1b: CS[2]  0x17: BootCS</p> <p>If TargetID is PEXunit:  0x59: Mem  0x51: I/O</p> <p><b>NOTE:</b> The initial value for SR1 is 0xD.  The initial value for SR2/3 is 0x0.</p>
31:16	Size	RW 0x0FFF	<p>Window Size</p> <p>The number of 1's specifies the size of the window in 64 KB granularity.</p> <p><b>NOTE:</b> The initial value for BAR2/3 is 0x0.</p>

## A.10.8 TDMA Control Registers

**Table 573: Control Register**  
Offset: 0x30840

Bit	Field	Type/InitVal	Description
2:0	DstBurstLimit	RW 0x3	<p>0 = Reserved  1 = Reserved  2 = Reserved  3 = 32 Bytes  4 = 128 Bytes  5 = Reserved  6 = Reserved  7 = Reserved</p>
3	Reserved	RW 0x0	Reserved
4	OutstandingRdEn	RW 0x0	<p>Outstanding Read Enable</p> <p><b>NOTE:</b> When enabled, the target unit must return the read data in order.</p> <p>0 = Disable: Outstanding read is disabled  1 = Enable: TDMA may assert up to two outstanding reads</p>
5	Reserved	RW 0x0	Reserved

**Table 573: Control Register (Continued)**  
Offset: 0x30840

Bit	Field	Type/InitVal	Description
8:6	SrcBurstLimit	RW 0x3	Burst Limit in Each TDMA Access 0 = Reserved 1 = Reserved 2 = Reserved 3 = 32 Bytes 4 = 128 Bytes 5 = Reserved 6 = Reserved 7 = Reserved
9	ChainMode	RW 0x0	Chained Mode 0 = Chained mode 1 = Non-Chained mode
10	Reserved	RW 0x0	Reserved
11	BS	RW 0x1	Byte swap on TDMA access to Mbus 0 = Byte swap 1 = No byte swap
12	TDMAEn	RW 0x0	TDMA Enable 0 = Disabled 1 = Enabled
13	FetchND	RW0C 0x0	Fetch Next Descriptor If set to 1, forces a fetch of the next descriptor. Cleared by HW after the fetch is completed. FetchND is only relevant in chain mode. <b>NOTE:</b> This bit can be set by writing to this register only when TDMA is IDLE. Trying to set this bit is When <TDMA Act> bit is asserted or when <Own> bit is set to CPU is illegal.
14	TDMA Act	RO 0x0	TDMA Active Read Only
31:15	Reserved	RW 0x0	Reserved

## A.10.9 TDMA Descriptor Registers

**Table 574: TDMA Byte Count Register**  
Offset: 0x30800

Bit	Field	Type/InitVal	Description
15:0	ByteCnt	RW 0x0	Number of bytes left for the TDMA to transfer
30:16	Reserved	RSVD 0x0	Reserved

**Table 574: TDMA Byte Count Register (Continued)**  
Offset: 0x30800

Bit	Field	Type/InitVal	Description
31	Own	RO 0x1	<p>Ownership Bit</p> <p>This bit indicates whether the descriptor is owned by the CPU (0) or the TDMA engine (1).</p> <p>This bit is relevant only when the descriptor is fetched.</p> <p>0 = CPU: CPU owned 1 = TDMA: TDMA engine owned</p>

**Table 575: TDMA Source Address Register**  
Offset: 0x30810

Bit	Field	Type/InitVal	Description
31:0	SrcAdd	RW 0x0	Bits [31:0] of the TDMA source address

**Table 576: TDMA Destination Address Register**  
Offset: 0x30820

Bit	Field	Type/InitVal	Description
31:0	DestAdd	RW 0x0	Bits [31:0] of the TDMA destination address

**Table 577: Next Descriptor Pointer Register**  
Offset: 0x30830

Bit	Field	Type/InitVal	Description
31:0	NextDescPtr	RW 0x0	Bits [31:0] of the TDMA next descriptor address The address must be 32-byte aligned (bits [3:0] must be 0x0)

**Table 578: Current Descriptor Pointer Register**  
Offset: 0x30870

Bit	Field	Type/InitVal	Description
31:0	CDPTR0	RO 0x0	Bits [31:0] of the address from which the current descriptor was fetched

## A.10.10 TDMA Interrupt Registers

**Table 579: TDMA Error Cause Register**

Offset: 0x308C8

Bit	Field	Type/InitVal	Description
0	Miss	RW0C 0x0	Set when the TDMA is enabled and configured so that both source and destination addresses miss all BARs.
1	DoubleHit	RW0C 0x0	Set when the TDMA is enabled and configured so that the source or destination address hit is in multiple BARs.
2	BothHit	RW0C 0x0	Set when the TDMA is enabled and configured so that both source and destination addresses hit any of the BARs.
3	DataError	RW0C 0x0	Set when bit 64 arrives as 0x0 from the crossbar.
31:4	Reserved	RSVD 0x0	Reserved

**Table 580: TDMA Error Mask Register**

Offset: 0x308CC

Bit	Field	Type/InitVal	Description
31:0	Mask	RW 0x0	<p>Mask bit per each err cause bit            Mask only affects the assertion of error pin. It does not affect the setting of bits in the Error Cause register.</p> <p>0 = Masked: Error is masked.            1 = Enabled: Error is enabled</p>

## A.11 XOR Engine Registers

The following table provides a summarized list of all of the XOR Engine registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 581: Register Map Table for the XOR Engine Registers**

Register Name	Offset	Table and Page
<b>XOR Engine Address Decoding Registers</b>		
XOR Engine [0..1] Window Control (XExWCR) Register (n=0–1)	Port0: XOR0: 0x60A40, XOR1: 0x60A44 Port1: XOR0: 0x60B40, XOR1: 0x60B44	Table 582, p. 659
XOR Engine Base Address (XEVARx) Register (n=0–7)	Port0: XEBAR0: 0x60A50, XEVAR1: 0x60A54, XEVAR2: 0x60A58, XEVAR3: 0x60A5C, XEVAR4: 0x60A60, XEVAR5: 0x60A64, XEVAR6: 0x60A68, XEVAR7: 0x60A6C Port1: XEBAR0: 0x60B50, XEVAR1: 0x60B54, XEVAR2: 0x60B58, XEVAR3: 0x60B5C, XEVAR4: 0x60B60, XEVAR5: 0x60B64, XEVAR6: 0x60B68, XEVAR7: 0x60B6C	Table 583, p. 660
XOR Engine Size Mask (XESMRx) Register (n=0–7)	Port0: XESMR0: 0x60A70, XESMR1: 0x60A74, XESMR2: 0x60A78, XESMR3: 0x60A7C, XESMR4: 0x60A80, XESMR5: 0x60A84, XESMR6: 0x60A88, XESMR7: 0x60A8C Port1: XESMR0: 0x60B70, XESMR1: 0x60B74, XESMR2: 0x60B78, XESMR3: 0x60B7C, XESMR4: 0x60B80, XESMR5: 0x60B84, XESMR6: 0x60B88, XESMR7: 0x60B8C	Table 584, p. 660
XOR Engine High Address Remap (XEHARRx) Register (n=0–3)	Port0: XEHARR0: 0x60A90, XEHARR1: 0x60A94, XEHARR2: 0x60A98, XEHARR3: 0x60A9C Port1: XEHARR0: 0x60B90, XEHARR1: 0x60B94, XEHARR2: 0x60B98, XEHARR3: 0x60B9C	Table 585, p. 661
XOR Engine [0..1] Address Override Control (XEAOCR) Register (n=0–1)	Port0: XEAOCR0: 0x60AA0, XEAOCR1: 0x60AA4 Port1: XEAOCR0: 0x60BA0, XEAOCR1: 0x60BA4	Table 586, p. 661



Table 581: Register Map Table for the XOR Engine Registers (Continued)

Register Name	Offset	Table and Page
<b>XOR Engine Control Registers</b>		
XOR Engine Channel Arbiter (XECHAR) Register	Port0: 0x60800, Port1: 0x60900	Table 587, p. 663
XOR Engine [0..1] Configuration (XExCR) Register (n=0–1)	Port0: XOR0: 0x60810, XOR1: 0x60814 Port1: XOR0: 0x60910, XOR1: 0x60914	Table 588, p. 664
XOR Engine [0..1] Activation (XExACTR) Register (n=0–1)	Port0: XOR0: 0x60820, XOR1: 0x60824 Port1: XOR0: 0x60920, XOR1: 0x60924	Table 589, p. 665
<b>XOR Engine Descriptor Registers</b>		
XOR Engine [0..1] Next Descriptor Pointer (XExNDPR) Register (n=0–1)	Port0: XOR0: 0x60A00, XOR1: 0x60A04 Port1: XOR0: 0x60B00, XOR1: 0x60B04	Table 590, p. 666
XOR Engine [0..1] Current Descriptor Pointer (XExCDPR) Register (n=0–1)	Port0: XOR0: 0x60A10, XOR1: 0x60A14 Port1: XOR0: 0x60B10, XOR1: 0x60B14	Table 591, p. 666
XOR Engine [0..1] Byte Count (XExBCR) Register (n=0–1)	Port0: XOR0: 0x60A20, XOR1: 0x60A24 Port1: XOR0: 0x60B20, XOR1: 0x60B24	Table 592, p. 667
<b>XOR Engine Interrupt Registers</b>		
XOR Engine Interrupt Cause (XEICR1) Register	Port0: 0x60830, Port1: 0x60930	Table 593, p. 667
XOR Engine Interrupt Mask (XEIMR) Register	Port0: 0x60840, Port1: 0x60940	Table 594, p. 668
XOR Engine Error Cause (XEECR) Register	Port0: 0x60850, Port1: 0x60950	Table 595, p. 669
XOR Engine Error Address (XEEAR) Register	Port0: 0x60860, Port1: 0x60960	Table 596, p. 669
<b>XOR Engine Memory Initialization</b>		
XOR Engine 0 and 1 Destination Pointer (XExDPR0) Register (n=0–1)	Port0: XOR0: 0x60AB0, XOR1: 0x60AB4 Port1: XOR0: 0x60BB0, XOR1: 0x60BB4	Table 597, p. 670
XOR Engine 0 and 1 Block Size (XExBSR) Register (n=0–1)	Port0: XOR0: 0x60AC0, XOR1: 0x60AC4 Port1: XOR0: 0x60BC0, XOR1: 0x60BC4	Table 598, p. 670
XOR Engine Initial Value Low (XEIVRL) Register	Port0: 0x60AE0, Port1: 0x60BE0	Table 599, p. 670
XOR Engine Initial Value High (XEIVRH) Register	Port0: 0x60AE4, Port1: 0x60BE4	Table 600, p. 670

## A.11.1 XOR Engine Address Decoding Registers

**Table 582: XOR Engine [0..1] Window Control (XExWCR) Register (n=0–1)**

Offset: Port0: XOR0: 0x60A40, XOR1: 0x60A44  
Port1: XOR0: 0x60B40, XOR1: 0x60B44

Bit	Field	Type/InitVal	Description
0	Win0en	RW 0x0	Window0 Enable 0 = Disable 1 = Enable
1	Win1en	RW 0x0	Window1 Enable
2	Win2en	RW 0x0	Window2 Enable
3	Win3en	RW 0x0	Window3 Enable
4	Win4en	RW 0x0	Window4 Enable
5	Win5en	RW 0x0	Window5 Enable
6	Win6en	RW 0x0	Window6 Enable
7	Win7en	RW 0x0	Window7 Enable
15:8	Reserved	RO 0x0	Reserved
17:16	Win0acc	RW 0x3	Window0 Access control In case of write protect violation (e.g. write data to a read only region), an interrupt is set, and the transaction is not driven to the target interface 0 = No access allowed 1 = Read Only 2 = Reserved 3 = Full access: Read or Write
19:18	Win1acc	RW 0x3	Window1 access control
21:20	Win2acc	RW 0x3	Window2 access control
23:22	Win3acc	RW 0x3	Window3 access control
25:24	Win4acc	RW 0x3	Window4 access control
27:26	Win5acc	RW 0x3	Window5 access control
29:28	Win6acc	RW 0x3	Window6 access control



**Table 582: XOR Engine [0..1] Window Control (XExWCR) Register (n=0–1) (Continued)**

Offset: Port0: XOR0: 0x60A40, XOR1: 0x60A44  
Port1: XOR0: 0x60B40, XOR1: 0x60B44

Bit	Field	Type/InitVal	Description
31:30	Win7acc	RW 0x3	Window7 access control

**Table 583: XOR Engine Base Address (XEBA Rx) Register (n=0–7)**

Offset: Port0: XEBAR0: 0x60A50, XEBAR1: 0x60A54, XEBAR2: 0x60A58, XEBAR3: 0x60A5C,  
XEBAR4: 0x60A60, XEBAR5: 0x60A64, XEBAR6: 0x60A68, XEBAR7: 0x60A6C  
Port1: XEBAR0: 0x60B50, XEBAR1: 0x60B54, XEBAR2: 0x60B58, XEBAR3: 0x60B5C,  
XEBAR4: 0x60B60, XEBAR5: 0x60B64, XEBAR6: 0x60B68, XEBAR7: 0x60B6C

Bit	Field	Type/InitVal	Description
3:0	Target	RW 0x0	Specifies the target interface associated with this window: See Address Decoding chapter for full details
7:4	Reserved	RO 0x0	Reserved
15:8	Attr	RW 0x0	Specifies target specific attributes depending on the target interface.
31:16	Base	RW 0x0	Base Address Used with the size register to set the address window size and location within the range of 4 GB space.

**Table 584: XOR Engine Size Mask (XESMRx) Register (n=0–7)**

Offset: Port0: XESMR0: 0x60A70, XESMR1: 0x60A74, XESMR2: 0x60A78, XESMR3: 0x60A7C,  
XESMR4: 0x60A80, XESMR5: 0x60A84, XESMR6: 0x60A88, XESMR7: 0x60A8C  
Port1: XESMR0: 0x60B70, XESMR1: 0x60B74, XESMR2: 0x60B78, XESMR3: 0x60B7C,  
XESMR4: 0x60B80, XESMR5: 0x60B84, XESMR6: 0x60B88, XESMR7: 0x60B8C

Bit	Field	Type/InitVal	Description
15:0	Reserved	RO 0x0	Reserved
31:16	SizeMask	RW 0x0	Window Size Used with the size register to set the address window size and location within the range of 4 GB space. Must be programmed from LSB to MSB as sequence of 1s followed by sequence of 0s. The number of 1s specifies the size of the window in 64 KB granularity (e.g. a value of 0x00ff specifies 256x64k = 16 MB).

**Table 585: XOR Engine High Address Remap (XEHARRx) Register (n=0–3)**

Offset: Port0: XEHARR0: 0x60A90, XEHARR1: 0x60A94, XEHARR2: 0x60A98, XEHARR3: 0x60A9C  
 Port1: XEHARR0: 0x60B90, XEHARR1: 0x60B94, XEHARR2: 0x60B98, XEHARR3: 0x60B9C

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> High Address Remap Register #N corresponds to Base Address register #N, respectively.			
31:0	Remap	RW 0x0	<p>Remap Address                      Specifies address bits[63:32] to be driven to the target interface.                      Only relevant for target interfaces that supports more than 4 GB address space.                      When using target interface that do not support more than 4 GB address space, this register must be cleared.</p>

**Table 586: XOR Engine [0..1] Address Override Control (XEAOCR) Register (n=0–1)**

Offset: Port0: XEAOCR0: 0x60AA0, XEAOCR1: 0x60AA4  
 Port1: XEAOCR0: 0x60BA0, XEAOCR1: 0x60BA4

Bit	Field	Type/InitVal	Description
0	SA0OvrEn	RW 0x0	Override Source Address #0 Control
2:1	SA0OvrPtr	RW 0x0	<p>Override Source Address #0 Pointer                      Specifies the register from which the override parameters will be taken.  <b>NOTE:</b> Valid only if SA0OvrEn is set.                      0 = XEHARR0: Target and attributes are taken from XEBAR0. Address[63:32] taken from XEHARR0.                      1 = XEHARR1: Target and attributes are taken from XEBAR1. Address[63:32] taken from XEHARR1.                      2 = XEHARR2: Target and attributes are taken from XEBAR2. Address[63:32] taken from XEHARR2.                      3 = XEHARR3: Target and attributes are taken from XEBAR3. Address[63:32] taken from XEHARR3.</p>
3	SA1OvrEn	RW 0x0	Override Source Address #1 Control
5:4	SA1OvrPtr	RW 0x0	<p>Override Source Address #1 Pointer  <b>NOTE:</b> Valid only if SA1OvrEn is set.</p>
6	SA2OvrEn	RW 0x0	Override Source Address #2 Control
8:7	SA2OvrPtr	RW 0x0	<p>Override Source Address #2 Pointer  <b>NOTE:</b> Valid only if SA2OvrEn is set.</p>
9	SA3OvrEn	RW 0x0	Override Source Address #3 Control
11:10	SA3OvrPtr	RW 0x0	<p>Override Source Address #3 Pointer  <b>NOTE:</b> Valid only if SA3OvrEn is set.</p>
12	SA4OvrEn	RW 0x0	Override Source Address #4 Control



**Table 586: XOR Engine [0..1] Address Override Control (XEAOCR) Register (n=0-1) (Continued)**

Offset: Port0: XEAOCR0: 0x60AA0, XEAOCR1: 0x60AA4

Port1: XEAOCR0: 0x60BA0, XEAOCR1: 0x60BA4

Bit	Field	Type/InitVal	Description
14:13	SA4OvrPtr	RW 0x0	Override Source Address #4 Pointer <b>NOTE:</b> Valid only if SA4OvrEn is set.
15	SA5OvrEn	RW 0x0	Override Source Address #5 Control
17:16	SA5OvrPtr	RW 0x0	Override Source Address #5 Pointer <b>NOTE:</b> Valid only if SA5OvrEn is set.
18	SA6OvrEn	RW 0x0	Override Source Address #6 Control
20:19	SA6OvrPtr	RW 0x0	Override Source Address #6 Pointer <b>NOTE:</b> Valid only if SA6OvrEn is set.
21	SA7OvrEn	RW 0x0	Override Source Address #7 Control
23:22	SA7OvrPtr	RW 0x0	Override Source Address #7 Pointer <b>NOTE:</b> Valid only if SA7OvrEn is set.
24	DAOvrEn	RW 0x0	Override Destination Address Control 0 = No override 1 = Override enable
26:25	DAOvrPtr	RW 0x0	Override Destination Address Pointer Specifies the register from which the override parameters will be taken. <b>NOTE:</b> Valid only if DAOvrEn is set. 0 = XEHARR0: Target and attributes are taken from XEBAR0. Address[63:32] taken from XEHARR0. 1 = XEHARR1: Target and attributes are taken from XEBAR1. Address[63:32] taken from XEHARR1. 2 = XEHARR2: Target and attributes are taken from XEBAR2. Address[63:32] taken from XEHARR2. 3 = XEHARR3: Target and attributes are taken from XEBAR3. Address[63:32] taken from XEHARR3.
27	NDAOvrEn	RW 0x0	Override Next Descriptor Address Control 0 = No Override 1 = Override enable

**Table 586: XOR Engine [0..1] Address Override Control (XEAOCR) Register (n=0–1) (Continued)**

Offset: Port0: XEAOCR0: 0x60AA0, XEAOCR1: 0x60AA4

Port1: XEAOCR0: 0x60BA0, XEAOCR1: 0x60BA4

Bit	Field	Type/InitVal	Description
29:28	NDAOvrPtr	RW 0x0	Override Next Descriptor Address Pointer Specifies the register from which the override parameters will be taken. <b>NOTE:</b> Valid only if NDAOvrEn is set. 0 = XEHARR0: Target and attributes are taken from XEBAR0. Address[63:32] taken from XEHARR0. 1 = XEHARR1: Target and attributes are taken from XEBAR1. Address[63:32] taken from XEHARR1. 2 = XEHARR2: Target and attributes are taken from XEBAR2. Address[63:32] taken from XEHARR2. 3 = XEHARR3: Target and attributes are taken from XEBAR3. Address[63:32] taken from XEHARR3.
31:30	Reserved	RSVD 0x0	Reserved

## A.11.2 XOR Engine Control Registers

**Table 587: XOR Engine Channel Arbiter (XECHAR) Register**

Offset: Port0: 0x60800 Port1: 0x60900

Bit	Field	Type/InitVal	Description
0	Slice0	RW 0x0	Slice #0 of the channel pizza arbiter. 0 = Channel0: Slice is owned by channel 0 1 = Channel1: Slice is owned by channel 1
1	Slice1	RW 0x1	Slice #1 of the channel pizza arbiter
2	Slice2	RW 0x0	Slice #2 of the channel pizza arbiter
3	Slice3	RW 0x1	Slice #3 of the channel pizza arbiter
4	Slice4	RW 0x0	Slice #4 of the channel pizza arbiter
5	Slice5	RW 0x1	Slice #5 of the channel pizza arbiter
6	Slice6	RW 0x0	Slice #6 of the channel pizza arbiter
7	Slice7	RW 0x1	Slice #7 of the channel pizza arbiter
31:8	Reserved	RO 0x0	Reserved



**Table 588: XOR Engine [0..1] Configuration (XExCR) Register (n=0–1)**

Offset: Port0: XOR0: 0x60810, XOR1: 0x60814

Port1: XOR0: 0x60910, XOR1: 0x60914

Bit	Field	Type/InitVal	Description
2:0	OperationMode	RW 0x0	Specifies the type of operation to be carried out by XOR Engine. 0 = XOR calculate operation 1 = CRC-32 calculate operation 2 = DMA operation 3 = Reserved 4 = Memory Initialization operation 5 = Reserved 6 = Reserved 7 = Reserved
3	Reserved	RW 0x0	Reserved
6:4	SrcBurstLimit	RW 0x4	Burst Limit in each source read request access over the Internal Mbus 0 = Reserved 1 = Reserved 2 = 32 Bytes 3 = 64 Bytes 4 = 128 Bytes 5 = Reserved 6 = Reserved 7 = Reserved
7	Reserved	RW 0x0	Reserved
10:8	DstBurstLimit	RW 0x4	Burst Limit in each destination write request access over the Internal Mbus 0 = Reserved 1 = Reserved 2 = 32 Bytes 3 = 64 Bytes 4 = 128 Bytes 5 = Reserved 6 = Reserved 7 = Reserved
11	Reserved	RW 0x0	Reserved
12	DrdResSwp	RW 0x0	Data Read Response Endianess Swap control If swapping is enabled, byte0 is exchanged with byte7, byte1 with byte 6 etc. 0 = No Swap 1 = Swap
13	DwrReqSwp	RW 0x0	Data Write request Endianess Swap control If swapping is enabled, byte0 is exchanged with byte7, byte1 with byte 6 etc. 0 = No swap 1 = Swap

**Table 588: XOR Engine [0..1] Configuration (XExCR) Register (n=0–1) (Continued)**

Offset: Port0: XOR0: 0x60810, XOR1: 0x60814  
 Port1: XOR0: 0x60910, XOR1: 0x60914

Bit	Field	Type/InitVal	Description
14	DesSwp	RW 0x0	Descriptor read/write Endianess Swap control If swapping is enabled, byte0 is exchanged with byte7, byte1 with byte 6 etc. 0 = No swap 1 = Swap
15	RegAccProtect	RW 0x1	Internal Register Access protection Enable 0 = Disable 1 = Enable
31:16	Reserved	RO 0x0	Reserved

**Table 589: XOR Engine [0..1] Activation (XExACTR) Register (n=0–1)**

Offset: Port0: XOR0: 0x60820, XOR1: 0x60824  
 Port1: XOR0: 0x60920, XOR1: 0x60924

Bit	Field	Type/InitVal	Description
0	XEStart	WO 0x0	<b>NOTE:</b> Software must confirm that the <XEstatus> field is in the Channel not active (0x0) state. Setting <XEstart> when XOR Engine is active will be disregarded. 0 = No meaning: Clearing this bit has no meaning and will be disregarded by XOR Engine. 1 = Start: When the software sets this bit, it activates the relevant XOR Engine channel. After entering active state, XOR Engine will signal the software by setting the <XEstatus> field to Channel active (0x1).
1	XEstop	WO 0x0	<b>NOTE:</b> Setting <XEstop> when XOR Engine is inactive or paused will be disregarded. 0 = No meaning: Clearing this bit has no meaning and will be disregarded by XOR Engine. 1 = Stop: When the software sets this bit, it de-activates the relevant XOR Engine channel. XOR Engine will stop the current operation at the earliest opportunity (refer to Stop Operation section of the Functional Specification). After entering de-active state XOR Engine will signal the software by setting the <XEstatus> field to Channel not active (0x0) and asserting the stopped interrupt.
2	XEpause	WO 0x0	XOR Engine Pause Control 0 = No meaning: Clearing this bit has no meaning and will be disregarded by XOR Engine. 1 = Pause state: When the software sets this bit, it pauses the relevant XOR Engine channel. XOR Engine will suspend at the earliest opportunity (refer to Pause Operation section of the Functional Specification). After entering paused state, the XOR Engine will signal the software by setting the <XEstatus> field to Channel paused (0x2) and asserting the paused interrupt.

**Table 589: XOR Engine [0..1] Activation (XExACTR) Register (n=0–1) (Continued)**

Offset: Port0: XOR0: 0x60820, XOR1: 0x60824  
 Port1: XOR0: 0x60920, XOR1: 0x60924

Bit	Field	Type/InitVal	Description
3	XERestart	WO 0x0	XOR Engine Restart after Pause Control 0 = No meaning: Clearing this bit has no meaning and will be disregarded by the XOR Engine. 1 = Pause state: The XOR Engine restart after pause. Setting this bit after the XOR Engine channel enters the pause state re-activates the channel and resumes the suspended operation execution.
5:4	XEstatus	RO 0x0	XOR Engine Status indication 0 = Channel not active 1 = Channel active 2 = Channel paused 3 = Reserved
31:6	Reserved	RO 0x0	Reserved

### A.11.3 XOR Engine Descriptor Registers

**Table 590: XOR Engine [0..1] Next Descriptor Pointer (XExNDPR) Register (n=0–1)**

Offset: Port0: XOR0: 0x60A00, XOR1: 0x60A04  
 Port1: XOR0: 0x60B00, XOR1: 0x60B04

Bit	Field	Type/InitVal	Description
31:0	NextDescPtr	RW 0x0	XOR Engine's next descriptor address pointer In XOR mode, bits[5:0] must be zero In CRC/DMA mode, bits[4:0] must be zero <b>NOTE:</b> The value 0x0 is reserved for end of chain NULL indication. Descriptors must not be placed at address 0x0. The XOR Engine will ignore attempts to read a descriptor from that address.

**Table 591: XOR Engine [0..1] Current Descriptor Pointer (XExCDPR) Register (n=0–1)**

Offset: Port0: XOR0: 0x60A10, XOR1: 0x60A14  
 Port1: XOR0: 0x60B10, XOR1: 0x60B14

Bit	Field	Type/InitVal	Description
31:0	CurrentDescPtr	RO 0x0	XOR Engine current descriptor address pointer. Points to the last descriptor that was fetched.

**Table 592: XOR Engine [0..1] Byte Count (XExBCR) Register (n=0–1)**

Offset: Port0: XOR0: 0x60A20, XOR1: 0x60A24

Port1: XOR0: 0x60B20, XOR1: 0x60B24

Bit	Field	Type/InitVal	Description
31:0	ByteCnt	RO 0x0	<p>Number of bytes left for the XOR Engine to execute the current descriptor operation.</p> <p>In XOR, DMA, and Memory Initialization modes: updated after every write action.</p> <p>In CRC mode: updated after every calculation operation.</p>

## A.11.4 XOR Engine Interrupt Registers

**Table 593: XOR Engine Interrupt Cause (XEICR1) Register**

Offset: Port0: 0x60830 Port1: 0x60930

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> All cause bits are clear only. They are set to 1 upon an interrupt event and cleared when the software writes a value of 0. Writing 1 has no effect (don't care). The XOR Engine disregards such write attempts.			
0	EODO	RW0C 0x0	<p>End of Descriptor</p> <p>Asserted when the XOR Engine finished the transfer of the current descriptor operation (byteCount==0). Software can control EOD interrupt assertion per descriptor through &lt;EODIntEn&gt; bit in the Command field of the descriptor.</p>
1	EOC0	RW0C 0x0	<p>End of Chain</p> <p>Asserted when the XOR Engine finished transfer of current descriptor operation, and it is currently the last in the descriptor chain (byteCount==0) and (XENDP=NULL).</p> <p>Also asserted upon end of chain processing due to error condition.</p>
2	Stopped0	RW0C 0x0	XOR Engine completed stopping routine, after receiving stop command (setting <XEstop>). It has entered Inactive state.
3	Paused0	RW0C 0x0	XOR Engine completed Pausing routine, after receiving pause command (setting <XEpause>).
3			It has entered Pause state.
4	AddrDecode0	RW0C 0x0	<p>Failed address decoding</p> <p>Address is not in any window or matches more than one window.</p>
5	AccProt0	RW0C 0x0	<p>Access Protect Violation</p> <p>Trying to access an address in a window in which access is not allowed.</p>
6	WrProt0	RW0C 0x0	<p>Write Protect Violation</p> <p>Trying to write to a window that is write protected.</p>
7	OwnErr0	RW0C 0x0	<p>Descriptor Ownership Violation</p> <p>Attempt to access the descriptor owned by the CPU.</p>



**Table 593: XOR Engine Interrupt Cause (XEICR1) Register (Continued)**  
Offset: Port0: 0x60830 Port1: 0x60930

Bit	Field	Type/InitVal	Description
8	IntParityErr0	RW0C 0x0	Parity Error Caused by erroneous internal buffer read.
9	XbarErr0	RW 0x0	Mbus Parity Error Caused by erroneous read response from the Mbus.
15:10	Reserved	RO 0x0	Reserved
25:16	Channel 1	RW0C 0x0	Same for XOR Engine channel #1
31:26	Reserved	RO 0x0	Reserved

**Table 594: XOR Engine Interrupt Mask (XEIMR) Register**  
Offset: Port0: 0x60840 Port1: 0x60940

Bit	Field	Type/InitVal	Description
0	EODMask0	RW 0x0	If set to 1, EOD interrupt is enabled
1	EOCMask0	RW 0x0	If set to 1, EOC interrupt is enabled
2	StoppedMask0	RW 0x0	If set to 1, Stopped interrupt is enabled
3	PauseMask0	RW 0x0	If set to 1, Paused interrupt is enabled
4	AddrDecodeMask0	RW 0x0	If set to 1, AddrDecode interrupt is enabled
5	AccProtMask0	RW 0x0	If set to 1, AccProt interrupt is enabled
6	WrProtMask0	RW 0x0	If set to 1, WrProt interrupt is enabled
7	OwnMask0	RW 0x0	If set to 1, OwnErr interrupt is enabled
8	IntParityMask0	RW 0x0	If set to 1, IntParityErr interrupt is enabled
9	XbarMask0	RW 0x0	If set to 1, XbarErr interrupt is enabled
15:10	Reserved	RW 0x0	Reserved
25:16	Channel1	RO 0x0	Same for XOR Engine Channel #1

**Table 594: XOR Engine Interrupt Mask (XEIMR) Register (Continued)**  
Offset: Port0: 0x60840 Port1: 0x60940

Bit	Field	Type/InitVal	Description
31:26	Reserved	RO 0x0	Reserved

**Table 595: XOR Engine Error Cause (XEECR) Register**  
Offset: Port0: 0x60850 Port1: 0x60950

Bit	Field	Type/InitVal	Description
4:0	ErrorType	ROC 0x0	<p>Specifies the error event currently reported in the Error Address register:  0x0 = Null  0x1 - 0x3 = Reserved.  0x4 = AddrDecode0  0x5 = AccProt0  0x6 = WrProt0  0x7- 0x13 = Reserved.  0x14 = AddrDecode1  0x15 = AccProt1  0x16 = WrProt1  0x17-0x1F = Reserved.</p> <p>This field is self cleared by reading the Error Address Register (XEEAR). Once the error cause is latched, no new error cause or address is latched to XEECR or XEEAR, until SW reads XEEAR.</p> <p>The Software should read XEECR first, and than XEEAR.</p>
31:5	Reserved	RO 0x0	Reserved

**Table 596: XOR Engine Error Address (XEEAR) Register**  
Offset: Port0: 0x60860 Port1: 0x60960

Bit	Field	Type/InitVal	Description
31:0	ErrAddr	RO 0x0	<p>Bits[31:0] of Error Address</p> <p>Latched upon any of the address windows violation event (address miss, multiple hit, access protect, write protect).</p> <p>Once the address is latched, no new address is latched until SW reads it (Read access to XEEAR).</p> <p>SW should read XEECR first, and than XEEAR.</p>

## A.11.5 XOR Engine Memory Initialization

**Table 597: XOR Engine 0 and 1 Destination Pointer (XExDPR0) Register (n=0–1)**

Offset: Port0: XOR0: 0x60AB0, XOR1: 0x60AB4

Port1: XOR0: 0x60BB0, XOR1: 0x60BB4

Bit	Field	Type/InitVal	Description
31:0	DstPtr	RW 0x0	Points to the target block of the Memory Initialization operations. <b>NOTE:</b> Valid only for Memory Initialization mode.

**Table 598: XOR Engine 0 and 1 Block Size (XExBSR) Register (n=0–1)**

Offset: Port0: XOR0: 0x60AC0, XOR1: 0x60AC4

Port1: XOR0: 0x60BC0, XOR1: 0x60BC4

Bit	Field	Type/InitVal	Description
31:0	BlockSize	RW 0x0	Size of block in bytes for Memory Initialization operation. This field along with XE0DPR or XE1DPR (Destination pointer registers), defines the target block for these operations. Minimum value: 128B Maximum Value: 4GB The value 0x00000000 stands for 4-GB block size. The block must not cross 4-GB boundary. <b>NOTE:</b> Valid only for Memory Initialization mode.

**Table 599: XOR Engine Initial Value Low (XEIVRL) Register**

Offset: Port0: 0x60AE0 Port1: 0x60BE0

Bit	Field	Type/InitVal	Description
31:0	InitValL	RW 0x0	LSB of Initial Value to be written cyclically to target block in MemInit mode. Mapped to bits[31:0] of initial value. This register is shared between the two XOR Engine channels. The XOR Engine will compose a 64 bit Initial Value out of InitValL and InitValH registers and write it cyclically to the target block. Target block can be of any alignment. <b>NOTE:</b> Valid only on MemInit modes.

**Table 600: XOR Engine Initial Value High (XEIVRH) Register**

Offset: Port0: 0x60AE4 Port1: 0x60BE4

Bit	Field	Type/InitVal	Description
31:0	InitValH	RW 0x0	MSB of Initial Value to be written cyclically to target block in MemInit mode. Mapped to bits[63:32] of initial value. This register is shared between the two XOR Engine channels. The XOR Engine will compose a 64 bit Initial Value out of InitValL and InitValH registers and write it cyclically to the target block. Target block can be of any alignment. <b>NOTE:</b> Valid only on MemInit modes.

## A.12 TWSI Registers

The following table provides a summarized list of all of the TWSI registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 601: Register Map Table for the TWSI Registers**

Register Name	Offset	Table and Page
TWSI Slave Address Register	0x11000,	<a href="#">Table 602, p. 671</a>
TWSI Data Register	0x11004,	<a href="#">Table 603, p. 671</a>
TWSI Control Register	0x11008,	<a href="#">Table 604, p. 671</a>
TWSI Status Register	0x1100C,	<a href="#">Table 605, p. 673</a>
TWSI Baud Rate Register	0x1100C,	<a href="#">Table 606, p. 674</a>
TWSI Extended Slave Address Register	0x11010,	<a href="#">Table 607, p. 674</a>
TWSI Soft Reset Register	0x1101C,	<a href="#">Table 608, p. 674</a>
TWSI Initialization Last Data Register	0x11098	<a href="#">Table 609, p. 674</a>

**Table 602: TWSI Slave Address Register**  
Offset: 0x11000

Bit	Field	Type/InitVal	Description
31:0	Reserved	RSVD 0x0	Reserved

**Table 603: TWSI Data Register**  
Offset: 0x11004

Bit	Field	Type/InitVal	Description
7:0	Data	RW 0x0	Data/Address byte to be transmitted by the TWSI master or slave, or data byte received In the case of the Address byte, bit [0] is the Read/Write Command bit.
31:8	Reserved	RO 0x0	Reserved

**Table 604: TWSI Control Register**  
Offset: 0x11008

Bit	Field	Type/InitVal	Description
1:0	Reserved	RSVD 0x0	Reserved



**Table 604: TWSI Control Register (Continued)**  
Offset: 0x11008

Bit	Field	Type/InitVal	Description
2	ACK	RW 0x0	Acknowledge When set to 1, the TWSI drives an acknowledge bit on the bus in response to a received address (slave mode), or in response to a data received (read data in master mode, write data in slave mode). For a master to signal a TWSI target with a read of last data, the CPU core must clear this bit (generating no acknowledge bit on the bus). For the slave to respond, this bit must always be set back to 1.
3	IFlg	RW 0x0	Interrupt Flag If any of the status codes other than 0xF8 are set, the TWSI hardware sets the bit to 1. If set to 1 and TWSI interrupts are enabled through bit [7], an interrupt is asserted. Cleared by a CPU core write of 0.
4	Stop	RW 0x0	Stop When set to 1, the TWSI master initiates a stop condition on the bus. The bit is set only. It is cleared by TWSI hardware after a stop condition is driven on the bus.
5	Start	RW 0x0	Start When set to 1, the TWSI master initiates a start condition on the bus, when the bus is free, or a repeated start condition, if the master already drives the bus. The bit is set only. It is cleared by TWSI hardware after a start condition is driven on the bus.
6	TWSIEn	RW 0x0	TWSI Enable 0 = Ignore: TW_SDA and TW_SCK inputs are ignored. The TWSI slave does not respond to any address on the bus. 1 = Respond: The TWSI slave responds to calls to its slave address, and to general calls if enabled.
7	IntEn	RW 0x0	Interrupt Enable When set to 1, an interrupt is generated each time the interrupt flag is set.
31:8	Reserved	RO 0x0	Reserved

**Table 605: TWSI Status Register**  
**Offset: 0x1100C**

Bit	Field	Type/InitVal	Description
7:0	Stat	RO 0xf8	<p>TWSI Status            0 = BusError            8 = StartTransmit: Start condition transmitted.            16 = StartTransmitRepeat: Repeated start condition transmitted.            24 = AddWrAck: Address plus write bit transmitted, acknowledge received.            32 = AddWrNoAck: Address plus write bit transmitted, acknowledge not received.            40 = MstrTxAck: Master transmitted data byte, acknowledge received.            48 = MstrTxNoAck: Master transmitted data byte, acknowledge not received.            56 = MstrLostArb: Master lost arbitration during address or data transfer.            64 = AddRdAck: Address plus read bit transmitted, acknowledge received.            72 = AddRdNoAck: Address +read bit transmitted, acknowledge not received.            80 = MstrRxAck: Master received read data, acknowledge transmitted.            88 = MstrRxNoAck: Master received read data, acknowledge not transmitted.            96 = SIRcdAddAck: Slave received slave address, acknowledge transmitted.            104 = MstrArbLost: Master lost arbitration during address transmit, address is targeted to the slave (write access), acknowledge transmitted.            112 = GnrRxAck: General call received, acknowledge transmitted.            120 = MstrArbLost: Master lost arbitration during address transmit, general call address received, acknowledge transmitted.            128 = SIRxWrDataAck: Slave received write data after receiving slave address, acknowledge transmitted.            136 = SIRxWrDataSlaveNoAck: Slave received write data after receiving slave address, acknowledge not transmitted.            144 = SIRxWrDataAck: Slave received write data after receiving general call, acknowledge transmitted.            152 = SIRxWrDataNoAck: Slave received write data after receiving general call, acknowledge not transmitted.            160 = SLRxStopStart: Slave received stop or repeated start condition.            168 = SIRxAddAck: Slave received address plus read bit, acknowledge transmitted.            176 = MstrArbLost: Master lost arbitration during address transmit, address is targeted to the slave (read access), acknowledge transmitted.            184 = SITxRdDataAck: Slave transmitted read data, acknowledge received.            192 = SITxRdDataNoAck: Slave transmitted read data, acknowledge not received.            200 = SITxRdByteAck: Slave transmitted last read byte, acknowledge received.            208 = SecondAddWrBitTxAck: Second address plus write bit transmitted, acknowledge received.            216 = SecondAddWrBitTxNoAck: Second address plus write bit transmitted, acknowledge not received.            224 = SecondAddRdBitTxAck: Second address plus read bit transmitted, acknowledge received.            232 = SecondAddRdBitTxNoAck: Second address + read bit transmitted, acknowledge not received.            248 = NotRelevant: No relevant status (default). Interrupt flag remains at 0.         </p>
31:8	Reserved	RSVD 0x0	Reserved



**Table 606: TWSI Baud Rate Register**

Offset: 0x1100C

Bit	Field	Type/InitVal	Description
2:0	N	WO 0x4	See exact frequency calculation in the TWSI section. Write only.
6:3	M	WO 0x4	See exact frequency calculation in the TWSI section. Write only.
31:7	Reserved	RSVD 0x0	Reserved

**Table 607: TWSI Extended Slave Address Register**

Offset: 0x11010

Bit	Field	Type/InitVal	Description
7:0	SAddr	RW 0x0	Bits [7:0] of the 10-bit slave address
31:8	Reserved	RO 0x0	Reserved

**Table 608: TWSI Soft Reset Register**

Offset: 0x1101C

Bit	Field	Type/InitVal	Description
31:0	Rst	WO 0x0	Write Only Write to this register resets the TWSI logic and sets all TWSI registers to their reset values.

**Table 609: TWSI Initialization Last Data Register**

Offset: 0x11098

Bit	Field	Type/InitVal	Description
31:0	Last	RW 0xFFFFFFFF	Serial initialization last data indication. Can be changed during the serial initialization sequence to terminate the initialization with a different pattern.

## A.13 NAND Flash Registers

The following table provides a summarized list of all of the NAND Flash registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 610: Register Map Table for the NAND Flash Registers**

Register Name	Offset	Table and Page
NAND Read Parameters Register	0x10418	<a href="#">Table 611, p. 675</a>
NAND Write Parameters Register	0x1041C	<a href="#">Table 612, p. 676</a>
NAND Flash Control Register	0x10470	<a href="#">Table 613, p. 676</a>

**Table 611: NAND Read Parameters Register**

Offset: 0x10418

Bit	Field	Type/InitVal	Description
This register defines the timing gaps of the different phases in a NAND flash READ transaction.			
5:0	TurnOff	RW 0x0F	The number of cycles in a READ access between the negation of NF_CEn and the following assertion of NF_CEn. Minimal value = 0x2 Number of cycles = <TurnOff> + 4
11:6	Acc2First	RW 0x1F	Defines the number of cycles in a READ access between the assertion of NF_CEn[0] and the cycle containing the first data sampled by the device. Minimal value = 0x6 Number of cycles = <Acc2First> - 4
16:12	Reserved	RW 0x0	Do not write a value other than 0x0 to this field.
22:17	Acc2Next	RW 0x1F	The number of cycles in a burst READ access between the cycle containing the first data sampled by the device and the cycle containing the next data sampled. Minimal value = 0x4
27:23	Reserved	RW 0x0	Do not write a value other than 0x0 to this field.
29:28	Reserved	RW 0x0	Do not write a value other than 0x0 to this field.
31:30	Reserved	RW 0x0	Do not write a value other than 0x0 to this field.



**Table 612: NAND Write Parameters Register**  
Offset: 0x1041C

Bit	Field	Type/InitVal	Description
5:0	CEn2WEn	RW 0xf	Defines the number of cycles in a WRITE access from NF_CEn assertion to the assertion of NF_WEn. Minimal value = 0x5 Number of cycles = <CEn2WEn> - 4
7:6	Reserved	RSVD 0x0	Reserved
13:8	WrLow	RW 0xf	The number of cycles in a WRITE access that the NF_WEn signal is kept active.
15:14	Reserved	RSVD 0x0	Reserved
21:16	WrHigh	RW 0xf	The number of cycles in a WRITE access between NF_WEn de-assertion and NF_CEn de-assertion.
31:22	Reserved	RSVD 0x0	Reserved

**Table 613: NAND Flash Control Register**  
Offset: 0x10470

Bit	Field	Type/InitVal	Description
0	Reserved	RW 0x1	Reserved
1	NFActCEnBoot	RW 0x0	When this bit is set, NAND_CEn is forced to 0x0. This bit is used to assure that CEn is active throughout the entire read phase in a Care NAND flash device. 0 = CEnDon'tCare 1 = CEnCare: CEn is asserted throughout the entire read phase.
2	Reserved	RW 0x0	Reserved
3	Reserved	RW 0x0	Reserved
4	Reserved	RW 0x0	Reserved
5	Reserved	RW 0x0	Reserved
6	Reserved	RW 0x1	Must not write 0x0 to this field.
7	NFActCen	RW 0x0	Forces NF_CEn to 0. This bit is used for CE Care NAND Flash.

**Table 613: NAND Flash Control Register (Continued)**  
Offset: 0x10470

Bit	Field	Type/InitVal	Description
8	NFISD	RW 0x1	Flash Initialization Sequence Disabled Sampled at reset.
13:9	NFOEnW	RW 0xC	Defines NF_OEn high width <(NFOEnHW+1) Core clocks>. For the default: 0x0C, the calculation is: (12/166 MHz ~72 ns).
18:14	NFTr	RW 0x1F	NAND Flash Time Ready Defines the maximum time for the boot NAND Flash to transfer the data from the array to the register. <(NFTr+1) x 1024 Core clocks>. The CPU is forced to reset during this time, before it starts the boot process. When using Accelerate mode, the timer is set to 1024 cycles no matter what is written to this field. For the default value, 0x1F, the calculation is: (32x1024/166 MHz ~ 197 us).
19	NFOEnDel	RW 0x0	NAND Flash OEn Delay . 0 = Delay: In access to Don't Care NAND flashes, NF_OEn is asserted one core clock cycle after NF_CEn. 1 = SameEdge: NF_OEn and NF_CEn are asserted at same cycle.
20	Reserved	RW 0x0	Reserved
21	Reserved	RW 0x0	Reserved
23:22	NF_NumAddrPhase	RW 0x3	When the Boot device is NAND flash, this field defines the number of address phases. 0 = NoInitialization: No initialization 1 = Init3AddrCycles: Initialization sequence enabled, 3 address cycles. 2 = Init4AddrCycles: Initialization sequence enabled, 4 address cycles. 3 = Init5AddrCycles: Initialization sequence enabled, 5 address cycles.
24	NF_Boot_Type	RW 0x1	Defines the type of the NAND flash read initialization sequence. 0 = NotEndedWith0x30: Initialization sequence does not end with command = 0x30. 1 = EndedWith0x30: Initialization sequence is ended with command = 0x30.
31:25	Reserved	RSVD 0x0	Reserved

## A.14 UART Registers

The following table provides a summarized list of all of the UART registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 614: Register Map Table for the UART Registers**

Register Name	Offset	Table and Page
Transmit Holding (THR) Register	UART0: 0x12000, UART1: 0x12100	<a href="#">Table 615, p. 678</a>
Divisor Latch Low (DLL) Register	UART0: 0x12000, UART1: 0x12100	<a href="#">Table 616, p. 679</a>
Receive Buffer (RBR) Register	UART0: 0x12000, UART1: 0x12100	<a href="#">Table 617, p. 679</a>
Interrupt Enable (IER) Register	UART0: 0x12004, UART1: 0x12104	<a href="#">Table 618, p. 679</a>
Divisor Latch High (DLH) Register	UART0: 0x12004, UART1: 0x12104	<a href="#">Table 619, p. 680</a>
Interrupt Identity (IIR) Register	UART0: 0x12008, UART1: 0x12108	<a href="#">Table 620, p. 680</a>
FIFO Control (FCR) Register	UART0: 0x12008, UART1: 0x12108	<a href="#">Table 621, p. 681</a>
Line Control (LCR) Register	UART0: 0x1200C, UART1: 0x1210C	<a href="#">Table 622, p. 681</a>
Modem Control (MCR) Register	UART0: 0x12010, UART1: 0x12110	<a href="#">Table 623, p. 682</a>
Line Status (LSR) Register	UART0: 0x12014, UART1: 0x12114	<a href="#">Table 624, p. 683</a>
Modem Status (MSR) Register	UART0: 0x12018, UART1: 0x12118	<a href="#">Table 625, p. 684</a>
Scratch Pad (SCR) Register	UART0: 0x1201C, UART1: 0x1211C	<a href="#">Table 626, p. 684</a>

**Table 615: Transmit Holding (THR) Register**

Offset: **UART0: 0x12000** **UART1: 0x12100**

Bit	Field	Type/InitVal	Description
7:0	TxHold	WO 0x0	<p>The THR is a write-only register that contains data to be transmitted from the serial port. Any time that the Transmit Holding Register Empty &lt;THRE&gt; bit of the Line Status Register (LSR) is set, data can be written to the LSR &lt;TxEmpty&gt; to be transmitted from the serial port.</p> <p>If FIFOs are not enabled and THRE is set, writing a single word to the THR resets the THRE and any additional writes to the THR before the &lt;THRE&gt; is set again causes the THR data to be overwritten.</p> <p>If FIFOs are enabled and &lt;THRE&gt; is set, up to 16 words of data may be written to the THR before the FIFO is full. Any attempt to write data when the FIFO is full results in the write data being lost.</p>
31:8	Reserved	RSVD 0x0	Reserved

**Table 616: Divisor Latch Low (DLL) Register**

Offset: UART0: 0x12000 UART1: 0x12100

Bit	Field	Type/InitVal	Description
7:0	DivLatchLow	WO 0x0	The DLH (Divisor Latch High) register in conjunction with DLL (Divisor Latch Low) register forms a 16-bit, read/write, Divisor Latch register that contains the baud rate divisor for the UART. It is accessed by first setting the DivLatchRdWrt bit in the Line Control Register (LCR). The output baud rate is equal to the input clock frequency divided by sixteen times the value of the baud rate divisor. $\text{baud} = (\text{clock frequency}) / (16 * \text{divisor})$
31:8	Reserved	RSVD 0x0	Reserved

**Table 617: Receive Buffer (RBR) Register**

Offset: UART0: 0x12000 UART1: 0x12100

Bit	Field	Type/InitVal	Description
7:0	RxBuf	RO 0x0	The RBR is a read-only register that contains the data byte transmitted to the serial port. The data in this register is valid only if the LSR <DataRxStat> bit in the Line Status Register (LSR) is set. In the non-FIFO mode (fifo_mode = 0), the data in the RBR must be read before the next data arrives; otherwise it will be overwritten, resulting in an overrun error. In the FIFO mode (fifo_mode = 1), this register accesses the head of the receive FIFO. If the receive FIFO is full and this register is not read before the next data word arrives, then the data already in the FIFO will be preserved but any incoming data will be lost.
31:8	Reserved	RSVD 0x0	Reserved

**Table 618: Interrupt Enable (IER) Register**

Offset: UART0: 0x12004 UART1: 0x12104

Bit	Field	Type/InitVal	Description
0	RxDatalntEn	RW 0x0	Enable Received Data Available Interrupt (ERBFI) When the FIFO mode is set in FIFO Control Register, this interrupt provides a character timeout indication. 0 = Disable: Disable interrupt 1 = Enable: Enable interrupt
1	TxHoldIntEn	RW 0x0	Enable Transmitter Holding Register Empty Interrupt (ETBEI) 0 = Disable: Disable interrupt 1 = Enable: Enable interrupt



**Table 618: Interrupt Enable (IER) Register (Continued)**  
Offset: UART0: 0x12004 UART1: 0x12104

Bit	Field	Type/InitVal	Description
2	RxLineStatIntEn	RW 0x0	Enable Receiver Line Status Interrupt (ELSI) 0 = Disable: Disable interrupt 1 = Enable: Enable interrupt
3	ModStatIntEn	RW 0x0	Enable Modem Status Interrupt (EDSSI) 0 = Disable: Disable interrupt 1 = Enable: Enable interrupt
31:4	Reserved	RSVD 0x0	Reserved

**Table 619: Divisor Latch High (DLH) Register**  
Offset: UART0: 0x12004 UART1: 0x12104

Bit	Field	Type/InitVal	Description
7:0	DivLatchHigh	WO 0x0	The DLH (Divisor Latch High) register in conjunction with DLL (Divisor Latch Low) register forms a 16-bit, read/write, Divisor Latch register that contains the baud rate divisor for the UART. It is accessed by first setting the DivLatchRdWrt bit in the Line Control Register (LCR). The output baud rate is equal to the input clock frequency divided by sixteen times the value of the baud rate divisor. $\text{baud} = (\text{clock frequency}) / (16 * \text{divisor})$
31:8	Reserved	RSVD 0x0	Reserved

**Table 620: Interrupt Identity (IIR) Register**  
Offset: UART0: 0x12008 UART1: 0x12108

Bit	Field	Type/InitVal	Description
3:0	InterruptID	RO 0x1	Interrupt ID 0 = Modem: Modem status changed. 1 = NoInterruptPen: No interrupt pending. 2 = THREEmpty 4 = RxData: Received data is available. 6 = RxStatus: Receiver Status 12 = TO: Character Timeout
5:4	Reserved	RSVD 0x0	Reserved
7:6	FIFOEn	RO 0x0	FIFO Enable 0 = Disable: FIFOs are disabled (default in FIFO mode). 3 = Enable: FIFOs are enabled.
31:8	Reserved	RO 0x0	Reserved

**Table 621: FIFO Control (FCR) Register**  
Offset: UART0: 0x12008 UART1: 0x12108

Bit	Field	Type/InitVal	Description
0	FIFOEn	WO 0x0	Enable transmit and receive FIFOs. This register controls the read and write data FIFO operation. 0 = Disable: Disable FIFOs 1 = Enable: Enable FIFOs
1	RxFIFOReset	WO 0x0	Receive FIFO reset 0 = NoFlushRx: Do not flush data from the receive FIFO. 1 = FlushRx: Flush data from the receive FIFO.
2	TxFIFOReset	WO 0x0	Transmit FIFO reset 0 = NoFlushTx: Do not flush data from the transmit FIFO 1 = FlushTx: Flush data from the transmit FIFO
5:3	Reserved	RSVD 0x0	Reserved
7:6	RxTrigger	WO 0x0	Receive Trigger 0 = 1Byte: 1 byte in FIFO 1 = 4Byte: 4 bytes in FIFO 2 = 8Byte: 8 bytes in FIFO 3 = 14Byte: 14 bytes FIFO
31:8	Reserved	RSVD 0x0	Reserved

**Table 622: Line Control (LCR) Register**  
Offset: UART0: 0x1200C UART1: 0x1210C

Bit	Field	Type/InitVal	Description
1:0	WLS	RW 0x0	Number of bits per character 0 = 5Bits 1 = 6Bits 2 = 7Bits 3 = 8Bits
2	Stop	RW 0x0	Stop Bits Transmitted 0 = 1Bits 1 = 2Bits
3	PEN	RW 0x0	Parity Enable 0 = Disable 1 = Enable
4	EPS	RW 0x0	Even or Odd Parity Select 0 = Odd: Odd parity 1 = Even: Even parity



**Table 622: Line Control (LCR) Register (Continued)**  
Offset: UART0: 0x1200C UART1: 0x1210C

Bit	Field	Type/InitVal	Description
5	Reserved	RSVD 0x0	Reserved
6	Break	RW 0x0	The <Break> bit sends a break signal by holding the SOUT line low until the <Break> bit is reset. 0 = No Signal: Do not send a break signal 1 = Signal: Send a break signal
7	DivLatchRdWr	RW 0x0	This bit must be set to address (reading and writing) of the Divisor Latch Low (DLL) Register and Divisor Latch High (DLH) Register to set the baud rate of the UART. This bit must be cleared to address the Receive Buffer Register (RBR), Transmit Holding Register (THR) and the Interrupt Enable Register (IER).
31:8	Reserved	RSVD 0x0	Reserved

**Table 623: Modem Control (MCR) Register**  
Offset: UART0: 0x12010 UART1: 0x12110

Bit	Field	Type/InitVal	Description
0	Reserved	RSVD 0x0	Reserved
1	RTS	RW 0x0	Request To Send The <RTS> bit is inverted and then drives the corresponding UA_RTSt output.
3:2	Reserved	RSVD 0x0	Reserved
4	Loopback	RW 0x0	Loopback The <Loopback> bit loops the data on the sout line back to the sin line. In this mode all the interrupts are fully functional. This feature is used for diagnostic purposes. 0 = NoLoopback 1 = Loopback
31:5	Reserved	RW 0x0	Reserved

**Table 624: Line Status (LSR) Register**  
Offset: UART0: 0x12014 UART1: 0x12114

Bit	Field	Type/InitVal	Description
0	DataRxStat	RO 0x0	Receive buffer status This bit is cleared when the Receive Buffer Register (RBR) is read. 0 = Empty: No characters in the receive buffer or FIFO. 1 = NotEmpty: Receive buffer or FIFO contains at least one character. A read operation of the Receiver Buffer Register clears this bit.
1	OverRunErr	ROC 0x0	Overrun Error 0 = NoError: No overrun. 1 = Error: Overrun error has occurred.
2	ParErr	ROC 0x0	Parity Error This bit indicates a parity error in the receiver if the <PEN> bit in the Line Control Register (LCR) is set. In the FIFO mode, since the parity error is associated with a character received, it is revealed when the character with the bad parity comes to the head of the FIFO.
3	FrameErr	ROC 0x0	Frame Error The FE bit flags a framing error in the receiver. A framing error occurs when the receiver does not detect a valid STOP bit in the received data. In the FIFO mode, since the framing error is associated with a character received, it is revealed when the character with the framing error comes to the head of the FIFO. The OE, PE and FE bits are reset when a read on the Line Control Register (LCR) is performed.
4	BI	ROC 0x0	The <BI> bit is set whenever the serial input (sin) is held in a logic 0 state for longer than the sum of start time + data bits + parity + stop bits. In the FIFO mode, the BI indication is carried through the FIFO and is revealed when the character is at the top of the FIFO. Reading the Line Control Register (LCR) clears the <BI>.
5	THRE	RO 0x1	Transmit Holding If the <THRE> bit is set, the device can accept a new character for transmission. If interrupts are enabled, it can cause an interrupt to occur when data from the Transmit Holding Register (THR) is transmitted to the transmit shift register. 0 = Closed: Do not accept a new character for transmission. 1 = Open: Accept a new character for transmission.
6	TxEmpty	RO 0x1	Transmitter Empty bit In FIFO mode, this bit is set whenever the Transmit Holding Register (THR), the Transmitter Shift Register, and the FIFO are all empty.
7	RxFIFOErr	ROC 0x0	This bit is only active when FIFOs are enabled. It is set when there is at least one parity error, framing error, or break indication in the FIFO. This bit is cleared when the Line Control Register (LCR) is read.
31:8	Reserved	RSVD 0x0	Reserved



**Table 625: Modem Status (MSR) Register**

Offset: UART0: 0x12018 UART1: 0x12118

Bit	Field	Type/InitVal	Description
0	DCTS	RW 0x0	The <DCTS> bit records whether the modem control line UA_CTSn has changed since the last time the CPU core read the MSR. In Loopback mode, <DCTS> reflects changes on Modem Control Register (MCR) bit[1] <RTS>. .
3:1	Reserved	RSVD 0x0	Reserved
4	CTS	RSVD 0x0	The CTS Modem Status bit--<CTS> --contains information on the current state of the modem control line. <CTS> is the compliment of UA_CTSn. In Loopback Mode, <CTS> is the same as Modem Control Register (MCR) bit[1] <RTS>.
31:5	Reserved	RSVD 0x0	Reserved

**Table 626: Scratch Pad (SCR) Register**

Offset: UART0: 0x1201C UART1: 0x1211C

Bit	Field	Type/InitVal	Description
7:0	Scratch	RW 0x0	The SCR register is an 8-bit read/write register for programmers to use as a temporary storage space.
31:8	Reserved	RSVD 0x0	Reserved

## A.15 SPI Registers

The following table provides a summarized list of all of the SPI registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 627: Register Map Table for the SPI Registers**

Register Name	Offset	Table and Page
Serial Memory Interface Control Register	0x10600	<a href="#">Table 628, p. 685</a>
Serial Memory Interface Configuration Register	0x10604	<a href="#">Table 629, p. 686</a>
Serial Memory Data Out Register	0x10608	<a href="#">Table 630, p. 687</a>
Serial Memory Data In Register	0x1060C	<a href="#">Table 631, p. 687</a>
Serial Memory Interface Interrupt Cause Register	0x10610	<a href="#">Table 632, p. 687</a>
Serial Memory Interface Interrupt Mask Register	0x10614	<a href="#">Table 633, p. 687</a>
Serial Memory Direct Write Configuration Register	0x10620	<a href="#">Table 634, p. 688</a>
Serial Memory Direct Write Header Register	0x10624	<a href="#">Table 635, p. 688</a>

**Table 628: Serial Memory Interface Control Register**

Offset: 0x10600

Bit	Field	Type/InitVal	Description
0	CSnAct	RW 0x0	Serial Memory Control 0 = Deactivated: Serial Memory Interface is deactivated. The SPI_CSn output is driven HIGH. 1 = Activated: Serial Memory Interface is activated. The SPI_CSn output is driven LOW.
1	SMemRdy	RO 0x1	Serial Memory Data Transfer Ready
3:2	Reserved	RW 0x0	Must write 0x0.
31:4	Reserved	RSVD 0x0	Reserved



**Table 629: Serial Memory Interface Configuration Register**  
**Offset: 0x10604**

Bit	Field	Type/InitVal	Description
4:0	SpiClkPrescale	RW 0x19	SPI_SCK prescaler from Core clock.  Default: Core clock / 18  00h-11h: Reserved 12h: Core clock / 4 13h: Core clock / 6 14h: Core clock / 8 15h: Core clock / 10 ... 1Dh: Core clock / 26 1Eh: Core clock / 28 1Fh: Core clock / 30
5	BYTE_LEN	RW 0x0	Number of bytes in each serial flash I/O transfer. 0 = 1Byte 1 = 2Byte
6	Reserved	RSVD 0x0	Reserved
7	Reserved_0	RW 0x0	Reserved
9:8	DirectAddrLen	RW 0x2	Address length of SPI. This field defines the number of bytes of the address phase in a direct accesses (READ or WRITE) to SPI. For direct Writes, this field is considered only if <Direct_Wr_Addr_En> is active. 0 = 1Byte: SPI has a 1-byte address length. 1 = 2Byte: SPI has a 2-byte address length. 2 = 3Byte: SPI has a 3-byte address length. 3 = 4Byte: SPI has a 4-byte address length.
10	DirectRdCommand	RW 0x0	Defines the SPI read command, when there is a direct read from the SPI. 0 = Read: Read command (opcode cycle = 0x03) 1 = Fast_Read: High speed clock read command (opcode cycle = 0x0B and a one byte dummy write after the address phase)
13:11	Reserved	RSVD 0x0	Reserved
14	Reserved_2	RW 0x1	Must be 0x1.
15	Reserved_1	RW 0x0	Must be 0x0.
31:16	Reserved	RSVD 0x0	Reserved

**Table 630: Serial Memory Data Out Register**  
**Offset: 0x10608**

Bit	Field	Type/InitVal	Description
15:0	SMemOut	RW 0x0	Writing to this register initiates a transfer out to external serial memory. The content of this register will be shifted out serially. When BYTE_LEN = 1, bits 7:0 are shifted out. When BYTE_LEN=2, bits 7:0 are shifted out, followed by bits 15:8. To shift in data from the serial memory, a dummy write can be used to advance SPI_SCK.
31:16	Reserved	RSVD 0x0	Reserved

**Table 631: Serial Memory Data In Register**  
**Offset: 0x1060C**

Bit	Field	Type/InitVal	Description
15:0	SMemIn	RW 0x0	While SMem_Out register bits are shifted out on SPI_MOSI output, the value of SPI_MISO input is shifted into this register. When BYTE_LEN = 1, data is shifted into bit 0 and left shifted. When BYTE_LEN = 2, data is shifted into bits7:0 first and then bits [15:8].
31:16	Reserved	RSVD 0x0	Reserved

**Table 632: Serial Memory Interface Interrupt Cause Register**  
**Offset: 0x10610**

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> Write 0 to clear interrupt bits. Writing 1 does not effect the interrupt bits.			
0	SMemRdilnt	RW0C 0x0	Serial memory data transfer ready interrupt.
31:1	Reserved	RSVD 0x0	Reserved

**Table 633: Serial Memory Interface Interrupt Mask Register**  
**Offset: 0x10614**

Bit	Field	Type/InitVal	Description
0	SMemRdilntMask	RW 0x1	Mask bit for the serial memory data transfer ready interrupt. Mask only affects the assertion of interrupt pin. It does not affect the setting of bits in the Cause register 0 = Masked: Interrupt masked 1 = Enabled: Interrupt enabled



**Table 633: Serial Memory Interface Interrupt Mask Register (Continued)**  
Offset: 0x10614

Bit	Field	Type/InitVal	Description
31:1	Reserved	RSVD 0x0	Reserved

**Table 634: Serial Memory Direct Write Configuration Register**  
Offset: 0x10620

Bit	Field	Type/InitVal	Description
0	Direct Write Hdr Enable	RW 0x0	When this bit is active (high), every direct write access to SPI will be prefixed with the first <Direct Wr Hdr Size> bytes of <DirectWrHdrData> .
2:1	Direct Wr Hdr Size	RW 0x0	This field specifies the number of bytes to be transmitted as the data prefix. 0 = 1 Bytes: Hdr[7:0] will be transmitted 1 = 2 Bytes: Hdr[15:0] will be transmitted 2 = 3 Bytes: Hdr[23:0] will be transmitted 3 = 4 Bytes: Hdr[31:0] will be transmitted
7:3	Reserved	RSVD 0x0	Reserved
8	Direct Wr Addr Enable	RW 0x0	When this bit is active (high), every direct write access to SPI will be prefixed with the first <DirectAddrLen> bytes of the request address.
15:9	Reserved	RSVD 0x0	Reserved
16	Direct Wr Deassert Cs	RW 0x1	When this bit is active (high), direct writes will assert SPI_CS prior to the data transmission and de-assert SPI_CS upon completion of the data transmission. If this feature is de-activated (set to 0) , then prior to any direct write, SPI_CS assertion should be guaranteed by software.
19:17	Direct Wr Cs Hold	RW 0x0	In Direct Writes, this field defines the minimal number of core clock cycles between assertion of CS and the first SPI_CLK.
31:20	Reserved	RSVD 0x0	Reserved

**Table 635: Serial Memory Direct Write Header Register**  
Offset: 0x10624

Bit	Field	Type/InitVal	Description
31:0	SpiDirectHdr	RW 0x0	Upon a direct access to SPI, the value in this field will be added as a header to the data. The number of bytes can be configured thru <>

## A.16 Audio Interface Registers

The following table provides a summarized list of all of the Audio Interface registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 636: Register Map Table for the Audio Interface Registers**

Register Name	Offset	Table and Page
<b>Audio Clocking Control</b>		
DCO Control Register	0xA1204	<a href="#">Table 637, p. 691</a>
SPCR and DCO Status Register	0xA120C	<a href="#">Table 638, p. 691</a>
Sample Counters Control Register	0xA1220	<a href="#">Table 639, p. 692</a>
Playback Sample Counter Register	0xA1224	<a href="#">Table 640, p. 693</a>
Recording Sample Counter Register	0xA1228	<a href="#">Table 641, p. 693</a>
Clocks Control Register	0xA1230	<a href="#">Table 642, p. 693</a>
Reserved Register	0xA1238	<a href="#">Table 643, p. 694</a>
<b>Audio Address Window Control Registers</b>		
<b>Address Decode DMA</b>		
Recording Window Base Address Register	0xA0A00	<a href="#">Table 644, p. 694</a>
Recording Window Control Register	0xA0A04	<a href="#">Table 645, p. 694</a>
Playback Window Base Address Register	0xA0A08	<a href="#">Table 646, p. 695</a>
Playback Window Control Register	0xA0A0C	<a href="#">Table 647, p. 695</a>
<b>Audio Interrupt and Error Registers</b>		
Audio Error Cause Register	0xA1300	<a href="#">Table 648, p. 695</a>
Audio Error Mask Register	0xA1304	<a href="#">Table 649, p. 696</a>
Audio Interrupt Cause Register	0xA1308	<a href="#">Table 650, p. 696</a>
Audio Interrupt Mask Register	0xA130C	<a href="#">Table 651, p. 698</a>
Recorded Byte Count for Interrupt Register	0xA1310	<a href="#">Table 652, p. 698</a>
Playback Byte Count for Interrupt Register	0xA1314	<a href="#">Table 653, p. 699</a>
<b>Audio Playback</b>		
<b>General Playback Control</b>		
Playback Control Register	0xA1100	<a href="#">Table 654, p. 699</a>
Playback Start Address Register	0xA1104	<a href="#">Table 655, p. 702</a>
Playback Buffer Size Register	0xA1108	<a href="#">Table 656, p. 702</a>
Playback Buffer Byte Counter Register	0xA110C	<a href="#">Table 657, p. 703</a>
Playback Byte Counter for Interrupt Register	0xA1318	<a href="#">Table 658, p. 703</a>
<b>I2S Playback Control</b>		
I2S Playback Control Register	0xA2508	<a href="#">Table 659, p. 703</a>
<b>SPDIF Playback Control</b>		
SPDIF Playback Control Register	0xA2204	<a href="#">Table 660, p. 704</a>
SPDIF Playback Channel Status Left n Register (n=0–5)	status0: 0xA2280, status1: 0xA2284, status2: 0xA2288, status3: 0xA228C, status4: 0xA2290, status5: 0xA2294	
	<a href="#">Table 661, p. 705</a>	



Table 636: Register Map Table for the Audio Interface Registers (Continued)

Register Name	Offset	Table and Page
SPDIF Playback Channel Status Right n Register (n=0–5)	status0: 0xA22A0, status1: 0xA22A4, status2: 0xA22A8, status3: 0xA22AC, status4: 0xA22B0, status5: 0xA22B4	Table 662, p. 705
SPDIF Playback User Bits Left n Register (n=0–5)	User0: 0xA22C0, User1: 0xA22C4, User2: 0xA22C8, User3: 0xA22CC, User4: 0xA22D0, User5: 0xA22D4	Table 663, p. 705
SPDIF Playback User Bits Right n Register (n=0–5)	User0: 0xA22E0, User1: 0xA22E4, User2: 0xA22E8, User3: 0xA22EC, User4: 0xA22F0, User5: 0xA22F4	Table 664, p. 706
<b>Audio Recording</b>		
<b>General Recording Control</b>		
Recording Control Register	0xA1000	Table 665, p. 706
Recording Start Address Register	0xA1004	Table 666, p. 708
Recording Buffer Size Register	0xA1008	Table 667, p. 708
Recording Buffer Byte Counter Register	0xA100C	Table 668, p. 709
Recorded Byte Counter for Interrupt Register	0xA131C	Table 669, p. 709
<b>I2S Recording Control</b>		
I2S Recording Control Register	0xA2408	Table 670, p. 709
<b>SPDIF Recording Control</b>		
SPDIF Recording General Register	0xA2004	Table 671, p. 710
SPDIF Recording Interrupt Cause and Mask Register	0xA2008	Table 672, p. 711
SPDIF Recording Channel Status Left n Register (n=0–5)	status0: 0xA2180, status1: 0xA2184, status2: 0xA2188, status3: 0xA218C, status4: 0xA2190, status5: 0xA2194	Table 673, p. 712
SPDIF Recording Channel Status Right n Register (n=0–5)	status0: 0xA21A0, status1: 0xA21A4, status2: 0xA21A8, status3: 0xA21AC, status4: 0xA21B0, status5: 0xA21B4	Table 674, p. 713
SPDIF Recording User Bits Left n Register (n=0–5)	User0: 0xA21C0, User1: 0xA21C4, User2: 0xA21C8, User3: 0xA21CC, User4: 0xA21D0, User5: 0xA21D4	Table 675, p. 713
SPDIF Recording User Bits Right n Register (n=0–5)	User0: 0xA21E0, User1: 0xA21E4, User2: 0xA21E8, User3: 0xA21EC, User4: 0xA21F0, User5: 0xA21F4	Table 676, p. 713

## A.16.1 Audio Clocking Control

**Table 637: DCO Control Register**  
**Offset: 0xA1204**

Bit	Field	Type/InitVal	Description
This register controls the DCO base frequency and offset. After this register is written, before it can be re-written, it must be read. After the register is read, a delay of 400 ns should be inserted and then a new value can be written.			
1:0	dco_ctrlfs	RW 0x0	Control FS selects the base frequency of the DCO.  0 = 11.2896 MHz 1 = 12.288 MHz 2 = 24.576 MHz 3 = Reserved
13:2	dco_ctrloffset	RW 0x800	Offset control in which each step equals to 1.27 ppm.  $dco\_mclk = \text{base frequency} * (1 + 1.27 * (\text{dco\_ctrloffset} - 2048) / 1e6)$  Offset values are: 0x020: -2016*1.27 ppm . . 0x800: 0 ppm . . 0xFD0: +2000*1.27 ppm  <b>NOTE:</b> Values above 0xFD0 or below 0x020 are reserved and should not be used.
31:14	Reserved	RSVD 0x0	Reserved

**Table 638: SPCR and DCO Status Register**  
**Offset: 0xA120C**

Bit	Field	Type/InitVal	Description
2:0	spsc_ctrlfs	RO 0x0	Defines the output base frequency FS. 0 = 11.2896 MHz 1 = 12.288 MHz 2 = 24.576 MHz 3 = Lower than 11.2896 MHz 4 = Higher than 24.576 MHz 5 = Reserved 6 = Reserved 7 = Other frequency
15:3	Reserved	RSVD 0x0	Reserved



**Table 638: SPCR and DCO Status Register (Continued)**  
Offset: 0xA120C

Bit	Field	Type/InitVal	Description
16	dco_lock	RO 0x0	DCO Lock Indication 0 = Unlocked DCO 1 = Locked DCO
31:17	Reserved	RSVD 0x0	Reserved

**Table 639: Sample Counters Control Register**  
Offset: 0xA1220

Bit	Field	Type/InitVal	Description
0	Activate Recording Counter	RW 0x0	When set together with bit [1], counters will start counting on the same cycle.  0 = Stop Counter 1 = Activate Counter
1	Activate Playback Counter	RW 0x0	When set together with bit [0], counters will start counting on the same cycle.  0 = Stop Counter 1 = Activate Counter
7:2	Reserved	RSVD 0x0	Reserved
8	Clear Recording Counter	WO 0x0	When set together with bit [9], both counters will be cleared on the same cycle.
9	Clear Playback Counter	WO 0x0	When set together with bit [8], both counters will be cleared on the same cycle.
31:10	Reserved	RSVD 0x0	Reserved

**Table 640: Playback Sample Counter Register**  
**Offset: 0xA1224**

Bit	Field	Type/InitVal	Description
31:0	Playback Sample Counter	RO 0x0	<p>Incremented by 1 after each left channel and right channel pair has been transmitted.</p> <p>Notes:</p> <ul style="list-style-type: none"> <li>1. This counter is a wrap around counter.</li> <li>2. If SPDIF playback is active, then this counter counts SPDIF samples; otherwise, I2S outgoing samples are counted. In the case of simultaneous I2S and SPDIF playback, samples are sent to both interfaces at the same frequency.</li> </ul>

**Table 641: Recording Sample Counter Register**  
**Offset: 0xA1228**

Bit	Field	Type/InitVal	Description
31:0	Recorded Sample Counter	RO 0x0	<p>Incremented by 1 for each incoming left channel and right channel pair.</p> <p>In Mono mode, only one of the channels is recorded so the counter reflects the number of recorded samples.</p> <p>In Stereo mode, the counter reflects the number of stereo samples that were received.</p> <p><b>NOTE:</b> This counter is a wrap around counter.</p>

**Table 642: Clocks Control Register**  
**Offset: 0xA1230**

Bit	Field	Type/InitVal	Description
1:0	MCLK source	RW 0x0	<p>MCLK = 256 Fs, where Fs = 44.1 kHz, 48 kHz, 96 kHz.</p> <p><b>NOTE:</b> It is possible to move between DCO mclk and SPCR mclk "on the fly" with no hardware glitches.</p> <p>Selection of External mclk or a move from the external mclk to another mclk source should be done when playback is not active.</p> <p>0 = DCO: MCLK is from DCO      2 = SPCR: MCLK is from SPCR      3 = External: MCLK is from external source</p>
14:2	Reserved	RSVD 0x0	Reserved
15	RSVD	RW 0x0	Always write 0 to this bit.
31:16	Reserved	RSVD 0x0	Reserved

**Table 643: Reserved Register**  
Offset: 0xA1238

Bit	Field	Type/InitVal	Description
0	RSVD	RW 0x1	Always write 1 to this bit.
31:1	Reserved	RW 0x7FFF8000	Reserved

## A.16.2 Audio Address Window Control Registers

### A.16.2.1 Address Decode DMA

**Table 644: Recording Window Base Address Register**  
Offset: 0xA0A00

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Recording Base	RW 0x0	Window Base Address

**Table 645: Recording Window Control Register**  
Offset: 0xA0A04

Bit	Field	Type/InitVal	Description
0	RSVD	RW 0x1	Always write 1'b1 to this bit.
3:1	RSVD	RSVD 0x0	Reserved
7:4	Recording TargetID	RW 0x0	Target Unit ID
15:8	Recording Attr	RW 0xE	Attribute
31:16	Recording Win Size	RW 0xFFFF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64 KByte granularity (e.g., a value of 0x00FF specifies 16 MByte). <b>NOTE:</b> The Window Size must be an integer multiple of the <Recording DMA Burst Size> in the Recording Control Register.

**Table 646: Playback Window Base Address Register**  
**Offset: 0xA0A08**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Playback Base	RW 0x1000	Window Base Address

**Table 647: Playback Window Control Register**  
**Offset: 0xA0A0C**

Bit	Field	Type/InitVal	Description
0	RSVD	RW 0x1	Always write 1'b1 to this bit.
3:1	RSVD	RSVD 0x0	Reserved
7:4	Playback TargetID	RW 0x0	Target Unit ID
15:8	Playback Attr	RW 0xD	Attribute
31:16	Playback Size	RW 0x0FFF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1s followed by a sequence of 0s. The number of 1s specifies the size of the window in 64 KByte granularity (e.g., a value of 0x00FF specifies 16 MByte). <b>NOTE:</b> The Window Size must be an integer multiple of the <Playback DMA Burst Size> in the Playback Control Register.

### A.16.3 Audio Interrupt and Error Registers

**Table 648: Audio Error Cause Register**  
**Offset: 0xA1300**

Bit	Field	Type/InitVal	Description
All bits are RW1C only. A cause bit sets upon an event occurrence. A write of 1 clears the bit. A write of 0 has no effect.			
0	Recording Addr Miss	RW1C 0x0	Set when the recording buffer address misses the BAR.
1	SPDIF Recording Par Error	RW1C 0x0	Set when the SPDIF recorded data has a parity error.
2	Playback Addr Miss	RW1C 0x0	Set when the playback buffer address misses the BAR.



**Table 648: Audio Error Cause Register (Continued)**  
Offset: 0xA1300

Bit	Field	Type/InitVal	Description
3	Playback Data Err	RW1C 0x0	Set when the playback data has a parity error indication.
4	Recording Overrun Err	RW1C 0x0	Set when an overrun error occurs during recording. The DMA Engine fails to transfer recording data to memory before new audio data arrives.
5	Playback I2S Underrun Err	RW1C 0x0	Set when an underrun error occurs during I2S playback. The DMA Engine fails to supply playback data on time.
6	Playback SPDIF Underrun Err	RW1C 0x0	Set when an underrun error occurs during SPDIF playback. The DMA Engine fails to supply playback data on time.
31:7	Reserved	RSVD 0x0	Reserved

**Table 649: Audio Error Mask Register**  
Offset: 0xA1304

Bit	Field	Type/InitVal	Description
31:0	AudioErrorMask	RW 0x0	Mask bit per cause bit. If a bit is set to 1, the corresponding event is enabled. Mask does not affect setting of the Interrupt Cause register bits; it only affects the assertion of the Audio Error interrupt.

**Table 650: Audio Interrupt Cause Register**  
Offset: 0xA1308

Bit	Field	Type/InitVal	Description
Most of the bits are RW1C only. A cause bit sets upon an event occurrence. A write of 1 clears the bit. A write of 0 has no effect.			
0	DMA Recording 1st Qtr Buf	RW1C 0x0	Set by the hardware when the recording DMA engine reaches the end of the first quarter in the recording buffer. Firmware should move this newly recorded data from this quarter of the cyclic buffer to avoid loosing the data when the Recording DMA re-writes this quarter of the cyclic buffer.
1	DMA Recording Half Buf	RW1C 0x0	Set by the hardware when the recording DMA engine reaches half of the recording buffer. Firmware should move this newly recorded data from this quarter/half of the cyclic buffer to another memory location to avoid loosing the data when the Recording DMA re-writes this quarter/half of the cyclic buffer.

**Table 650: Audio Interrupt Cause Register (Continued)**  
Offset: 0xA1308

Bit	Field	Type/InitVal	Description
2	DMA Recording 3rd Qtr Buf	RW1C 0x0	Set by the hardware when the recording DMA engine reaches the end of the third quarter in the recording buffer. Firmware should move this newly recorded data from this quarter of the cyclic buffer to another memory location to avoid loosing the data when the Recording DMA re-writes this quarter of the cyclic buffer.
3	DMA Recording End Buf	RW1C 0x0	Set by the hardware when the recording DMA engine reaches the end of the recording buffer. Firmware should move this newly recorded data from this end of the cyclic buffer to another memory location to avoid loosing the data when the Recording DMA re-writes this quarter/half of the cyclic buffer.
4	DMA Playback 1st Qtr Buf	RW1C 0x0	Set by the hardware when the playback DMA engine reaches the first quarter of the playback buffer. Firmware should write new playback data into this first quarter of the cyclic buffer.
5	DMA Playback Half Buf	RW1C 0x0	Set by the hardware when the playback DMA engine reaches half of the playback buffer. Firmware should write new playback data into this quarter/half of the cyclic buffer.
6	DMA Playback 3rd Qtr Buf	RW1C 0x0	Set by the hardware when the playback DMA engine reaches the end of the third quarter in the playback buffer. Firmware should write new playback data into this quarter of the cyclic buffer.
7	DMA Playback Buf End	RW1C 0x0	Set by the hardware when the playback DMA engine reaches the end of the playback buffer. Firmware should write new playback data into this quarter/half of the cyclic buffer.
8	spdif_rc_block_end	RW1C 0x0	End of SPDIF recording block.
9	spdif_rc_int	RO 0x0	Indicates that at least one of the non masked interrupts in the SPDIF Recording Interrupt Cause and Mask Register was asserted.
10	Reserved	RSVD 0x0	Reserved
11	spdif_pb_block_start	RW1C 0x0	Indicates start of SPDIF playback block.  <b>NOTE:</b> When status/user bits are read from the register file, they may be changed before the first operation and after each start of block interrupt. If status/user bits need to be changed in the next block, then the user has the time it takes to play one block, to update the new status/user registers.
12	Reserved	RSVD 0x0	Reserved

**Table 650: Audio Interrupt Cause Register (Continued)**  
Offset: 0xA1308

Bit	Field	Type/InitVal	Description
13	Recorded num of Bytes Int	RW1C 0x0	This interrupt is set whenever the value in the <Recorded Byte Counter for Interrupt> field of the Recorded Byte Counter for Interrupt Register reaches the value that was configured in the <Recorded Byte Count for Interrupt> field of the Recorded Byte Count for Interrupt Register.
14	Played num of Bytes Int	RW1C 0x0	This interrupt is set whenever the value in the <Playback Byte Counter for Interrupt> field of the Playback Byte Counter for Interrupt Register reaches the value that was configured in the <Playback Byte Count for Interrupt> field of the Playback Byte Count for Interrupt Register.
20:15	Reserved	RSVD 0x0	Reserved
21	spdif_rc_locked	RW1C 0x0	When asserted indicates a change of <spdif_rc_lock_status> from unlocked to locked. Current lock status is indicated at <spdif_rc_lock_status>
22	spdif_rc_lock_loss	RW1C 0x0	When asserted indicates that SPDIF recording lock has been lost. <Spdif_rc_lock_status> was changed from locked to unlocked. Current lock status is indicated at <spdif_rc_lock_status>
23	spdif_rc_lock_status	RO 0x0	When asserted, SPDIF recording is synchronized (locked to the input stream).
31:24	Reserved	RSVD 0x0	Reserved

**Table 651: Audio Interrupt Mask Register**  
Offset: 0xA130C

Bit	Field	Type/InitVal	Description
31:0	AudioInterruptMask	RW 0x0	Mask bit per cause bit. If a bit is set to 1, the corresponding event is enabled. Mask does not affect setting of the Interrupt Cause register bits; it only affects the assertion of the Audio interrupt. Always write 0 to bits [15], [20], [24], and [25].

**Table 652: Recorded Byte Count for Interrupt Register**  
Offset: 0xA1310

Bit	Field	Type/InitVal	Description
23:0	Recorded Byte Count for Interrupt	RW 0x800000	An interrupt is asserted whenever the configured number of bytes has been recorded. <b>NOTE:</b> A value of 0 is reserved.
31:24	Reserved	RSVD 0x0	Reserved

**Table 653: Playback Byte Count for Interrupt Register**  
**Offset: 0xA1314**

Bit	Field	Type/InitVal	Description
23:0	Playback Byte Count for int	RW 0x800000	An interrupt is asserted whenever the configured number of bytes has been played. <b>NOTE:</b> A value of 0 is reserved.
31:24	Reserved	RSVD 0x0	Reserved

## A.16.4 Audio Playback

### A.16.4.1 General Playback Control

**Table 654: Playback Control Register**  
**Offset: 0xA1100**

Bit	Field	Type/InitVal	Description
2:0	Playback Sample Size	RW 0x1	<p>This field determines the memory structure during playback. If playback data was recorded using the Audio unit then the value of this field should be the same as the value of the &lt;Recording Sample Size&gt; field in the Recording Control Register, that was used in the recording.</p> <p><b>NOTE:</b></p> <ul style="list-style-type: none"> <li>1. The SPDIF playback only supports up to 24 bits.</li> <li>2. When I2S is played, always configure in addition, the &lt;I2S PB Sample Size&gt; in the I2S Playback Control Register to the sample size of the real audio stream.</li> </ul> <p>0 = 32 bits: I2S mode only. Reserved for SPDIF mode. Each 32-bit word in memory includes 32 bits of audio data.</p> <p>1 = 24 bits: Each 32-bit word in memory includes 24 bits of audio data at bits [23:0]. If this mode is used for a sampling size of less than 24 bits, then bit [23] should still hold the MS bit of the data. Redundant LS bits should be 0.</p> <p>2 = 20 bits: Each 32-bit word in memory includes 20 bits of audio data at bits [19:0].</p> <p>3 = 16 bits compact: Each 32-bit word in memory consists of two halfwords of 16 bits of audio data. NOTE: Using this mode for SPDIF means that user and status bits are stored in the internal buffers and cannot be stored with the data.</p> <p>7 = 16 bits: Each 32-bit word in memory includes 16 bits of audio data at bits [15:0].</p>



**Table 654: Playback Control Register (Continued)**  
Offset: 0xA1100

Bit	Field	Type/InitVal	Description
3	I2S Playback Enable	RW 0x0	<p>Used to enable/disable playback DMA, as well as I2S data, I2S bit clock, and I2S left/right clock.</p> <p><b>NOTE:</b></p> <p>It is possible to simultaneously playback the same data through I2S and SPDIF. In this case, both interfaces should be enabled and disabled concurrently.</p> <p>Use &lt;SPDIF Playback Mute&gt; or &lt;I2S Playback Mute&gt; if only one interface should remain active, after both interfaces are simultaneously enabled.</p> <p>0 = I2S playback disabled 1 = I2S playback enabled</p>
4	SPDIF Playback Enable	RW 0x0	<p>Used to enable/disable SPDIF playback and the playback DMA.</p> <p><b>NOTE:</b></p> <p>It is possible to simultaneously playback the same data through I2S and SPDIF. In this case, both interfaces should be enabled and disabled concurrently.</p> <p>Use &lt;SPDIF Playback Mute&gt; or &lt;I2S Playback Mute&gt; if only one interface should remain active, after both interfaces are simultaneously enabled.</p> <p>0 = SPDIF playback disabled 1 = SPDIF playback enabled</p>
6:5	Playback Mono	RW 0x0	<p>When the Mono mode is used, the ADU Engine treats playback data as mono data.</p> <p><b>NOTE:</b></p> <ol style="list-style-type: none"><li>1. If only one of the channels is used in Mono (data is not duplicated to both channels) mode, then the unused channel data is zero.</li><li>2. If recorded data does not contain block start information. For example, since only the right channel was recorded, then user bits cannot be taken from memory if they are required to align to the start of block.</li><li>3. If the SPDIF start of block information should be taken from memory (since user bits are taken from memory and should align to the start of block) then values 2 and 3 cannot be used, i.e., only Stereo mode and Left Channel Mono mode are permissible.</li><li>4. If stereo data is in the playback buffer and the playback is in Mono mode, the playback engine treats the data as if it belonged to a single channel. Therefore, the audio is played at half the speed, and lasts twice as long.</li></ol> <p>0 = Stereo Mode: Mono mode is off. 1 = Left Channel Mono: Left channel is used in Mono playback mode. 2 = Right Channel Mono: Right channel is used in Mono playback mode. 3 = Both Channels Mono: Same playback data is duplicated to both channels.</p>

**Table 654: Playback Control Register (Continued)**  
Offset: 0xA1100

Bit	Field	Type/InitVal	Description
7	I2S Playback Mute	RW 0x0	<p>When this bit is set, the DMA playback continues, but playback data is not transmitted.  Mute is different from Pause: during Pause the DMA playback stops.  In normal operation, this bit must be de-asserted.</p> <p><b>NOTE:</b> &lt;I2S Playback Mute&gt; must not be disabled when &lt;I2S Playback Enable&gt; is deasserted. After playback is disabled, &lt;I2S Playback Mute&gt; must remain active until &lt;Playback Busy&gt; is read twice as IDLE.</p> <p>0 = Mute disabled  1 = Mute enabled</p>
8	SPDIF Playback Mute	RW 0x0	<p>When this bit is set, the DMA playback continues, but playback data is not transmitted.  Mute is different from Pause since DMA playback stops while pausing.  In normal operation, this bit must be deasserted.</p> <p><b>NOTE:</b></p> <ol style="list-style-type: none"> <li>If the validity bit is taken from the &lt;SPDIF PB Reg Validity&gt; bit of the SPDIF Playback Control Register, then before mute begins and until it ends, it is recommended to assert the &lt;SPDIF PB Reg Validity&gt;.</li> <li>&lt;SPDIF Playback Mute&gt; must not be disabled when &lt;SPDIF Playback Enable&gt; is deasserted. After Playback is disabled, &lt;SPDIF Playback Mute&gt; must remain active until &lt;Playback Busy&gt; is read twice as IDLE.</li> </ol> <p>0 = Mute disabled  1 = Mute enabled</p>
9	Playback Pause	RW 0x0	<p>When this bit is set, the playback DMA stops.  Pause differs from mute, in which the DMA playback operation is on going.</p> <p><b>NOTE:</b> &lt;Playback Pause&gt; must not be disabled when &lt;SPDIF Playback Enable&gt; or &lt;I2S Playback Enable&gt; are deasserted. After playback is disabled, &lt;Playback Pause&gt; must remain active until &lt;Playback Busy&gt; is read twice as IDLE.</p> <p>0 = Pause disabled  1 = Pause enabled</p>
10	Loop Back	RW 0x0	<p>When Loopback is enabled, playback data is looped back to be recorded.  Loopback mode is used for test.</p> <p>0 = Normal: Normal operation  1 = Loopback: Loopback test mode is enabled.</p>
12:11	Playback DMA Burst Size	RW 0x2	Specifies the Burst Size of the DMA. 1 = 32 Bytes 2 = 128 Bytes 3 = Reserved
15:13	Reserved	RSVD 0x0	Reserved



**Table 654: Playback Control Register (Continued)**  
Offset: 0xA1100

Bit	Field	Type/InitVal	Description
16	Playback Busy	RO 0x0	<p>When asserted: playback is on going.</p> <p><b>NOTE:</b> After either I2S or SPDIF playback is disabled, this bit must be read twice as "idle" before any "Audio Playback" register can be changed. 0 = DMA is idle 1 = DMA is active</p>
31:17	Reserved	RSVD 0x0	Reserved

**Table 655: Playback Start Address Register**  
Offset: 0xA1104

Bit	Field	Type/InitVal	Description
31:0	Playback Buffer Start Address	RW 0x0	<p>Contains the pointer to the starting location of the playback buffer in memory. Data in that location will be the first data to be played.</p> <p><b>NOTE:</b></p> <ol style="list-style-type: none"><li>1. The Base Address must be 8-byte aligned.</li><li>2. For less DMA memory bandwidth consumption, it is highly recommended that the address be aligned to the &lt;Playback DMA Burst Size&gt; as configured in the Playback Control Register.</li></ol>

**Table 656: Playback Buffer Size Register**  
Offset: 0xA1108

Bit	Field	Type/InitVal	Description
21:0	Playback Buffer Size	RW 0x0	<p>Contains the size of the playback buffer in Word units (32 bit) minus one. If set to 0, playback buffer is 1 Word (4 bytes total). If set to 1, playback buffer is 2 Words (8 bytes total).</p> <p>.</p> <p>.</p> <p>If set to 0x3FFFF (max allowed value), playback buffer size is 4M words or 16 MBytes.</p> <p><b>NOTE:</b> Playback Buffer Size must be an integer multiple of &lt;Playback DMA Burst Size&gt;, as determined in the Playback Control register. The buffer size should be greater than the burst size.</p>
31:22	Reserved	RSVD 0x0	Reserved

**Table 657: Playback Buffer Byte Counter Register**  
**Offset: 0xA110C**

Bit	Field	Type/InitVal	Description
23:0	Playback buf Byte Counter	RO 0x0	<p>Number of bytes that were transferred by the playback DMA. Modulates the playback buffer size.</p> <p><b>NOTE:</b> This counter is reset when its value reaches the value of the &lt;Playback Buffer Size&gt; field in the Playback Buffer Size Register or when playback is disabled.</p>
31:24	Reserved	RSVD 0x0	Reserved

**Table 658: Playback Byte Counter for Interrupt Register**  
**Offset: 0xA1318**

Bit	Field	Type/InitVal	Description
23:0	Playback Byte Counter for Interrupt	RO 0x0	<p>Number of bytes that were transferred by the playback DMA from playback enable or since the last Played-num-of-Bytes-Int interrupt assertion.</p> <p>This counter is reset when the counter reaches the value set in the Playback Byte Count for Interrupt register or when playback is disabled.</p>
31:24	Reserved	RSVD 0x0	Reserved

#### A.16.4.2 I2S Playback Control

**NOTE:** Only 32-bit access is allowed to the I2S Playback registers.

**Table 659: I2S Playback Control Register**  
**Offset: 0xA2508**

Bit	Field	Type/InitVal	Description
22:0	Reserved	RSVD 0x0	Reserved
23	RSVD	RW 0x0	Always write 0 to this bit.
25:24	RSVD	RW 0x0	The value of this field must be 2'b0.
29:26	I2S PB Justification	RW 0x5	<p><b>NOTE:</b></p> <p>1.The same value should be configured to &lt;I2S Recording Justification&gt;.</p> <p>2. Bit [26] determines the polarity of I2S LRCLK.</p> <p>0 = I2S_Left_J: I2S Left Justified      5 = I2S      8 = I2S_Right_J: I2S Right Justified</p>

**Table 659: I2S Playback Control Register (Continued)**  
**Offset: 0xA2508**

Bit	Field	Type/InitVal	Description
31:30	I2S PB sample size	RW 0x0	<p>I2S playback real sample size.          Redundant bits are sent as zeros (I2S always uses 32 bits per channel).</p> <p><b>NOTE:</b> The same sample size must be configured in the &lt;Playback Sample Size&gt; of the Playback Control Register.</p> <p>0 = 32bit          1 = 24bit          2 = 20bit          3 = 16bit: Use this configuration for both 16 bit and 16 bit compact mode memory structure configuration.</p>

#### A.16.4.3 SPDIF Playback Control

**NOTE:** Only 32-bit access is allowed to the SPDIF Playback registers.

**Table 660: SPDIF Playback Control Register**  
**Offset: 0xA2204**

Bit	Field	Type/InitVal	Description
0	SPDIF PB Block Start	RW 0x0	<p>When both SPDIF user and status bits are taken from the Audio unit register file internal buffers, then this bit should be 0. Otherwise, configure it to 1.</p> <p>0 = Internally: SPDIF frame block start is managed internally by the Audio unit.          1 = FIFO data word: SPDIF frame block start is controlled by the values present in the memory data word.</p>
1	SPDIF PB Enable Mem Validity	RW 0x0	<p>0 = Register Validity: SPDIF Frame validity bit is set to &lt;SPDIF PB Reg Validity&gt;.          1 = Memory Validity: SPDIF Frame validity bit is set to the value that was read from memory.</p>
2	SPDIF PB Mem User En	RW 0x0	<p>0 = User Buffered: User bit is used from the internal user buffers.          1 = User in Mem: User bit is read from memory.</p>
3	RSVD	RW 0x0	Always write 0 to this field.
4	Force Parity Error	RW 0x0	<p>When this bit is set, if the Perr bit (parity error indication bit) in memory is also asserted, then an erroneous parity will be sent.</p> <p>0 = Disable: Force parity error from memory is disabled.          1 = Enable: Force parity error from memory is enabled.</p>
5	RSVD	RW 0x0	Always write 0 to this bit.
9:6	Reserved	RSVD 0x0	Reserved

**Table 660: SPDIF Playback Control Register (Continued)**  
Offset: 0xA2204

Bit	Field	Type/InitVal	Description
15:10	RSVD	RW 0x0	Always write 0 to this field.
16	SPDIF PB Reg Validity	RW 0x0	Validity bit value when <SPDIF PB Enable Mem Validity> = 0. <b>NOTE:</b> During Non-PCM stream, this bit should be set to 1, to comply with the SPDIF standard.
17	SPDIF PB Non PCM	RW 0x0	Defines the audio stream type.  0 = PCM 1 = Non-PCM
19:18	RSVD	RW 0x0	Always write 0 to this field.
31:20	RSVD	RW 0x0	Always write 0 to this field.

**Table 661: SPDIF Playback Channel Status Left n Register (n=0–5)**  
Offset: status0: 0xA2280, status1: 0xA2284, status2: 0xA2288, status3: 0xA228C, status4: 0xA2290, status5: 0xA2294

Bit	Field	Type/InitVal	Description
31:0	Left Channel Status Buffer	RW 0x0	Left Channel Status internal buffer. Register offset 0x80 represents bits [31:0] of left channel status bits. Register offset 0x94 represents bits [191:160] of the left channel status bits.

**Table 662: SPDIF Playback Channel Status Right n Register (n=0–5)**  
Offset: status0: 0xA22A0, status1: 0xA22A4, status2: 0xA22A8, status3: 0xA22AC, status4: 0xA22B0, status5: 0xA22B4

Bit	Field	Type/InitVal	Description
31:0	Right Channel Status Buffer	RW 0x0	Right Channel Status internal buffer. Register offset 0xA0 represents bits [31:0] of the right channel status bits. Register offset 0xB4 represents bits [191:160] of right channel status bits.

**Table 663: SPDIF Playback User Bits Left n Register (n=0–5)**  
Offset: User0: 0xA22C0, User1: 0xA22C4, User2: 0xA22C8, User3: 0xA22CC, User4: 0xA22D0, User5: 0xA22D4

Bit	Field	Type/InitVal	Description
31:0	Left Channel User Buffer	RW 0x0	Left Channel User bits internal buffer. Register offset 0xc0 represents bits [31:0] of the left channel user bits. Register offset 0xd4 represents bits [191:160] of the left channel user bits.



**Table 664: SPDIF Playback User Bits Right n Register (n=0–5)**

Offset: User0: 0xA22E0, User1: 0xA22E4, User2: 0xA22E8, User3: 0xA22EC, User4: 0xA22F0,  
User5: 0xA22F4

Bit	Field	Type/InitVal	Description
31:0	Right Channel User Buffer	RW 0x0	Right Channel User bits internal buffer. Register offset 0xe0 represents bits [31:0] of the right channel user bits. Register offset 0xf4 represents bits [191:160] of the right channel user bits.

## A.16.5 Audio Recording

### A.16.5.1 General Recording Control

**Table 665: Recording Control Register**

Offset: 0xA1000

Bit	Field	Type/InitVal	Description
2:0	Recording Sample Size	RW 0x1	<p>This field determines the memory structure during recording.</p> <p><b>NOTE:</b></p> <p>1. SPDIF only supports up to 24 bits.</p> <p>2. In the case of I2S also configure: the &lt;I2S Recording Sample Size&gt; in the I2S Recording Control Register.</p> <p>0 = 32Bits: I2S mode only. Reserved for SPDIF. Each 32-bit word in memory includes 32 bits of audio data.</p> <p>1 = 24Bits: Each 32-bit word in memory includes 24 bits of audio data at bits [23:0]. If this mode is used for a sampling size of less than 24 bits then bit [23] holds the MS bit of the data. Redundant LS bits will be 0 in the case of I2S.</p> <p>2 = 20Bits: Each 32-bit word in memory includes 20 bits of audio data at bits [19:0].</p> <p>3 = 16Bits Compact: Each 32-bit word in memory consists of two halfwords of 16 bits of audio data. Using this mode for SPDIF means that the user and status bits are stored in the internal buffers and cannot be stored with the data.</p> <p>7 = 16Bits: Each 32-bit word in memory includes 16 bits of audio data at bits [15:0].</p>
3	Recorded Mono Channel	RW 0x0	<p><b>NOTE:</b></p> <p>1. When recording SPDIF in Mono mode, this bit should be set to 0 if it is required that start of block (that only exists in the left channel) will be recorded with the data.</p> <p>2. Start of block is required to be used from memory if user bits are taken from memory and are required to align to the start of block (or if status bits could be used from memory).</p> <p>0 = Left Channel Mono: Left channel is used in Mono recording mode. Right channel data is discarded.</p> <p>1 = Right Channel Mono: Right channel is used in Mono recording mode. Left channel data is discarded.</p>

**Table 665: Recording Control Register (Continued)**  
Offset: 0xA1000

Bit	Field	Type/InitVal	Description
4	Recording Mono	RW 0x0	<p>When set, only one of the channels is recorded. Data arriving from the other channel is discarded.</p> <p>The &lt;Recorded Mono Channel&gt; bit determines which channel is recorded.</p> <p>0 = Stereo mode: Mono mode is off. 1 = Mono mode</p>
6:5	Recording DMA Burst Size	RW 0x2	<p>Specifies the burst size of the DMA.</p> <p>1 = 32 Bytes 2 = 128 Bytes 3 = Reserved</p>
7	RSVD	RW 0x0	Always write 0 to this bit.
8	Recording Mute	RW 0x0	<p>When this bit is set, the DMA recording continues but the recorded data is forced to zero (except for SPDIF control bits).</p> <p>In normal operation, this bit must be cleared to zero.</p> <p>Mute is different from Pause: during pause the DMA recording stops.</p> <p>0 = Mute disabled 1 = Mute enabled</p>
9	Recording Pause	RW 0x0	<p>When set, the DMA engine stops working after it completes the transfer of the current burst of audio data to memory. The audio data received after the DMA stops is discarded.</p> <p><b>NOTE:</b></p> <p>1. Pause differs from mute since during mute the DMA recording operation is on going. 2. I2S Pause differs from I2S stop (I2S Recording Enable is set to 0), since pause does not disable the I2S recording clocks.</p> <p>0 = Pause disabled 1 = Pause enabled</p>
10	I2S Recording Enable	RW 0x0	<p>Used to enable/disable I2S recording.</p> <p>SPDIF recording should be disabled before I2S recording is enabled.</p> <p><b>NOTE:</b></p> <p>1. If &lt;I2S Recording Enable&gt; is set to 0 and I2S playback is also disabled, then the I2S clocks are disabled. 2. If the I2S clocks should remain active while I2S Recording DMA should be stopped, use &lt;Recording Pause&gt; and leave I2S recording enabled. 3. If both SPDIF recording and I2S recording are enabled, I2S is recorded.</p> <p>0 = I2S recording disabled 1 = I2S recording enabled</p>



**Table 665: Recording Control Register (Continued)**  
Offset: 0xA1000

Bit	Field	Type/InitVal	Description
11	SPDIF Recording Enable	RW 0x0	<p>Used to enable/disable SPDIF recording. I2S recording should be disabled before SPDIF recording is enabled.</p> <p><b>NOTE:</b> If both SPDIF recording and I2S recording are enabled, I2S is recorded.</p> <p>0 = SPDIF recording disabled 1 = SPDIF recording enabled</p>
31:12	Reserved	RSVD 0x0	Reserved

**Table 666: Recording Start Address Register**  
Offset: 0xA1004

Bit	Field	Type/InitVal	Description
31:0	Recording Buffer Start Address	RW 0x0	<p>Contains pointer to the starting location of the recording buffer in memory. When recording starts, data in this location is the first to be received.</p> <p><b>NOTE:</b></p> <ol style="list-style-type: none"><li>1. Base Address must be 8-Byte aligned.</li><li>2. It is highly recommended that the address be aligned to the &lt;Recording DMA Burst Size&gt; as configured in the Recording Control register. This will save 50% of the DMA accesses to memory.</li></ol>

**Table 667: Recording Buffer Size Register**  
Offset: 0xA1008

Bit	Field	Type/InitVal	Description
21:0	Recording Buffer Size	RW 0x0	<p>Contains the size of the recording buffer in Word units (32 bit) minus one. If set to 0, recording buffer is 1 Word (4 bytes total). If set to 1, recording buffer is 2 Words (8 bytes total).</p> <p>.</p> <p>.</p> <p>.</p> <p>If set to 0x3FFFF (maximum allowed value), recording buffer size is 4M words or 16 MBytes.</p> <p><b>NOTE:</b> Recording Buffer Size must be an integer multiple of &lt;Recording DMA Burst Size&gt;, as determined in the Recording Control register. The buffer size should be greater than the burst size.</p>
31:22	Reserved	RSVD 0x0	Reserved

**Table 668: Recording Buffer Byte Counter Register**  
**Offset: 0xA100C**

Bit	Field	Type/InitVal	Description
23:0	Recording Buf Byte Count	RO 0x0	<p>Number of bytes transferred by the recording DMA. Modulates the recording buffer size.</p> <p><b>NOTE:</b> This counter is reset when its value reaches the value of the &lt;Recording Buffer Size&gt; field in the Recording Buffer Size Register or when recording is re-enabled.</p>
31:24	Reserved	RSVD 0x0	Reserved

**Table 669: Recorded Byte Counter for Interrupt Register**  
**Offset: 0xA131C**

Bit	Field	Type/InitVal	Description
23:0	Recorded Byte Counter for Interrupt	RO 0x0	<p>Number of bytes that were transferred by the recording-DMA from recording-enable or since the last Recorded-num-of-Bytes-Int interrupt assertion.</p> <p><b>NOTE:</b> This counter is reset when its value reaches the value in the &lt;Recorded Byte Count for Interrupt&gt; field of the Recorded Byte Count for Interrupt Register, or when recording is re-enabled.</p>
31:24	Reserved	RSVD 0x0	Reserved

### A.16.5.2 I2S Recording Control

**NOTE:** Only 32-bit access is allowed to the I2S Recording registers.

**Table 670: I2S Recording Control Register**  
**Offset: 0xA2408**

Bit	Field	Type/InitVal	Description
<b>NOTE:</b> Only 32-bit access is allowed to the I2S recording registers.			
24:0	Reserved	RSVD 0x0	Reserved
25	RSVD	RW 0x0	Always write 0 to this bit.



**Table 670: I2S Recording Control Register (Continued)**  
Offset: 0xA2408

Bit	Field	Type/InitVal	Description
29:26	I2S Recording Justification	RW 0x5	<b>NOTE:</b> 1. Bit [26] determines the polarity of I2S LRCLK. 0 = Left: I2S data is in the left channel when I2S LRCLK is high. 1 = Right: I2S data is in the right channel when I2S LRCLK is high 2. The same value should be configured to <I2S Playback Justification> when recording and playback are concurrent. 0 = I2S_Left_J: I2S Left Justified. 5 = I2S 8 = I2S_Right_J: I2S Right Justified.
31:30	I2S Recording Sample Size	RW 0x0	I2S Recording Sample Size. Redundant bits are written as zeros to memory.  <b>NOTE:</b> The same sample size must also be configured in the <Recording Sample Size> field at the Recording Control register.  0 = 32bit 1 = 24bit 2 = 20bit 3 = 16bit

### A.16.5.3 SPDIF Recording Control

**NOTE:** Only 32-bit access is allowed to the SPDIF Recording registers.

**Table 671: SPDIF Recording General Register**  
Offset: 0xA2004

Bit	Field	Type/InitVal	Description
0	RSVD	RW 0x0	Always write 0 to this bit.
2:1	Core Clk Frequency	RW 0x0	2 = 166.6666 MHz: For the 88F6180 and 88F6192. 3 = 200 MHz: For the 88F6281.
3	RSVD	RW 0x0	Always write 0 to this bit.
4	RSVD	RW 0x0	Always write 0 to this bit.
6:5	Reserved	RSVD 0x0	Reserved
7	Valid PCM info	RO 0x0	Valid PCM information indicates that the value of the NonPCM (bit-14) is valid. 0 = Not valid: The values are not valid. 1 = Valid: The values are valid.
13:8	Reserved	RSVD 0xF	Reserved

**Table 671: SPDIF Recording General Register (Continued)**  
**Offset: 0xA2004**

Bit	Field	Type/InitVal	Description
14	NonPCM	RO 0x0	<p>Incoming SPDIF audio format.  <b>NOTE:</b> This bit is only valid if &lt;Valid PCM info&gt; is asserted.</p> <p>0 = LinearPCM: Linear PCM samples  1 = NonLinearPCM: Non Linear PCM samples</p>
31:15	Reserved	RSVD 0x0	Reserved

**Table 672: SPDIF Recording Interrupt Cause and Mask Register**  
**Offset: 0xA2008**

Bit	Field	Type/InitVal	Description
0	RSVD	RW 0x0	Always write 0 to this bit.
1	Right Ch User int mask	RW 0x0	<p>Right Channel User interrupt mask</p> <p>0 = Masked  1 = Not Masked</p>
2	Left Ch user int mask	RW 0x0	<p>Left Channel User interrupt mask</p> <p>0 = Masked  1 = Not Masked</p>
3	Right Ch Status int mask	RW 0x0	<p>Right Channel Status interrupt mask</p> <p>0 = Mask  1 = Not Masked</p>
4	Left Channel Status mask	RW 0x0	<p>Left Channel Status interrupt mask</p> <p>0 = Masked  1 = Not Masked</p>
5	RSVD	RW 0x0	Always write 0 to this bit.
6	RSVD	RW 0x0	Always write 0 to this bit.
7	RSVD	RW 0x0	Always write 0 to this bit.
8	RSVD	RW 0x0	Always write 0 to this bit.
16:9	Reserved	RSVD 0x0	Reserved



**Table 672: SPDIF Recording Interrupt Cause and Mask Register (Continued)**  
Offset: 0xA2008

Bit	Field	Type/InitVal	Description
17	Right Channel User Int	RW1C 0x0	Right Channel User Interrupt (Ch 2)  0 = No Right Ch User Int: No changes were detected in the right channel user bits buffer compared to the previous block. 1 = Right Ch User Int: A change was detected in the right channel user bits buffer compared to the previous block.
18	Left Channel User Int	RW1C 0x0	Left Channel User interrupt (Ch1)  0 = No left Ch User Int: No changes were detected in the left channel user bits buffer, compared to the previous block. 1 = Left Ch User Int: A change was detected in the left channel user bits buffer, compared to the previous block.
19	Right Ch Status Int	RW1C 0x0	Right Channel Status interrupt (Ch 2)  0 = No Right Channel Status Int: No changes were detected in the right status buffer compared to the previous block. 1 = Right Channel Status Int: A change was detected in the right status buffer compared to the previous block.
20	Left Ch Status Int	RW1C 0x0	Left Channel Status Interrupt (Ch 1)  0 = No Left Status Int: No changes were detected in the left status buffer compared to the previous block. 1 = Left Status Int: A change was detected in the left status buffer compared to the previous block.
31:21	Reserved	RSVD 0x0	Reserved

**Table 673: SPDIF Recording Channel Status Left n Register (n=0–5)**  
Offset: status0: 0xA2180, status1: 0xA2184, status2: 0xA2188, status3: 0xA218C, status4: 0xA2190,  
status5: 0xA2194

Bit	Field	Type/InitVal	Description
31:0	Left Channel Status Buffer	RO 0x0	Left Channel Status internal buffer. Register offset 0x180 represents bits [31:0] of the left channel status buffer. Register offset 0x194 represents bits [191:160] of the left channel status buffer.

**Table 674: SPDIF Recording Channel Status Right n Register (n=0–5)**

Offset: status0: 0xA21A0, status1: 0xA21A4, status2: 0xA21A8, status3: 0xA21AC, status4: 0xA21B0,  
status5: 0xA21B4

Bit	Field	Type/InitVal	Description
31:0	Right Channel Status Buffer	RO 0x0	Right Channel Status internal buffer. Register offset 0x1A0 represents bits [31:0] of the right channel status buffer. Register offset 0x1B4 represents bits [191:160] of the right channel status buffer.

**Table 675: SPDIF Recording User Bits Left n Register (n=0–5)**

Offset: User0: 0xA21C0, User1: 0xA21C4, User2: 0xA21C8, User3: 0xA21CC, User4: 0xA21D0,  
User5: 0xA21D4

Bit	Field	Type/InitVal	Description
31:0	User Bits Left Channel Buffer	RO 0x0	User Bits Left Channel internal buffer. Register offset 0x1c0 represents bits [31:0] of the left channel user buffer. Register offset 0x1d4 represents bits [191:160] of the left channel user buffer.

**Table 676: SPDIF Recording User Bits Right n Register (n=0–5)**

Offset: User0: 0xA21E0, User1: 0xA21E4, User2: 0xA21E8, User3: 0xA21EC, User4: 0xA21F0,  
User5: 0xA21F4

Bit	Field	Type/InitVal	Description
31:0	User Bits Right Channel Buffer.	RO 0x0	User Bits Right Channel internal buffer. Register offset 0x1e0 represents bits [31:0] of the right channel user buffer. Register offset 0x1f4 represents bits [191:160] of right channel user buffer.

## A.17 SDIO Registers

The following table provides a summarized list of all of the SDIO registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 677: Register Map Table for the SDIO Registers**

Register Name	Offset	Table and Page
DMA Buffer Address 16 LSB Register	0x90000	Table 678, p. 715
DMA Buffer Address 16 MSB Register	0x90004	Table 679, p. 715
Data Block Size Register	0x90008	Table 680, p. 716
Data Block Count Register	0x9000C	Table 681, p. 716
Argument in Command 16 LSB Register	0x90010	Table 682, p. 716
Argument in Command 16 MSB Register	0x90014	Table 683, p. 717
Transfer Mode Register	0x90018	Table 684, p. 717
Command Register	0x9001C	Table 685, p. 719
Response Halfword 0 Register	0x90020	Table 686, p. 720
Response Halfword 1 Register	0x90024	Table 687, p. 720
Response Halfword 2 Register	0x90028	Table 688, p. 720
Response Halfword 3 Register	0x9002C	Table 689, p. 721
Response Halfword 4 Register	0x90030	Table 690, p. 721
Response Halfword 5 Register	0x90034	Table 691, p. 721
Response Halfword 6 Register	0x90038	Table 692, p. 721
Response Halfword 7 Register	0x9003C	Table 693, p. 722
16-bit Data Word Accessed by CPU Register	0x90040	Table 694, p. 722
CRC7 of I Response Register	0x90044	Table 695, p. 722
Host Present State 16 LSB Register	0x90048	Table 696, p. 723
Host Control Register	0x90050	Table 697, p. 724
Data Block Gap Control Register	0x90054	Table 698, p. 727
Clock Control Register	0x90058	Table 699, p. 728
Software Reset Register	0x9005C	Table 700, p. 728
Normal Interrupt Status Register	0x90060	Table 701, p. 729
Error Interrupt Status Register	0x90064	Table 702, p. 730
Normal Interrupt Status Enable Register	0x90068	Table 703, p. 732
Error Interrupt Status Enable Register	0x9006C	Table 704, p. 733
Normal Interrupt Status Interrupt Enable Register	0x90070	Table 705, p. 734
Error Interrupt Status Interrupt Enable Register	0x90074	Table 706, p. 735
Auto CMD12 Interrupt Status Register	0x90078	Table 707, p. 736
Current Number of Bytes Remaining in Data Block Register	0x9007C	Table 708, p. 736
Current Number of Data Blocks Left to Be Transferred Register	0x90080	Table 709, p. 737
Argument in Auto Cmd12 Command 16 LSB Transferred Register	0x90084	Table 710, p. 737
Argument in Auto Cmd12 Command 16 MSB Transferred Register	0x90088	Table 711, p. 737

**Table 677: Register Map Table for the SDIO Registers (Continued)**

Register Name	Offset	Table and Page
Index of Auto Cmd12 Commands Transferred Register	0x9008C	Table 712, p. 738
Auto Cmd12 Response Halfword 0 Register	0x90090	Table 713, p. 738
Auto Cmd12 Response Halfword 1 Register	0x90094	Table 714, p. 738
Auto Cmd12 Response Halfword 2 Register	0x90098	Table 715, p. 738
Mbus Control Low Register	0x90100	Table 716, p. 739
Mbus Control High Register	0x90104	Table 717, p. 739
Window0 Control Register	0x90108	Table 718, p. 740
Window0 Base Register	0x9010C	Table 719, p. 740
Window1 Control Register	0x90110	Table 720, p. 741
Window1 Base Register	0x90114	Table 721, p. 741
Window2 Control Register	0x90118	Table 722, p. 741
Window2 Base Register	0x9011C	Table 723, p. 742
Window3 Control Register	0x90120	Table 724, p. 742
Window3 Base Register	0x90124	Table 725, p. 743
Clock Divider Value Register	0x90128	Table 726, p. 743
Address Decoder Error Register	0x9012C	Table 727, p. 743
Address Decoder Error Mask Register	0x90130	Table 728, p. 743

**Table 678: DMA Buffer Address 16 LSB Register**  
Offset: 0x90000

Bit	Field	Type/InitVal	Description
15:0	DmaAddrLo	RW 0x0	16 LSB of the DMA system buffer starting byte address, word-aligned.
31:16	Reserved	RSVD 0x0	Reserved

**Table 679: DMA Buffer Address 16 MSB Register**  
Offset: 0x90004

Bit	Field	Type/InitVal	Description
15:0	DmaAddrHi	RW 0x0	16 MSB of the DMA system buffer starting byte address.
31:16	Reserved	RSVD 0x0	Reserved



**Table 680: Data Block Size Register**

Offset: 0x90008

Bit	Field	Type/InitVal	Description
11:0	BlockSize	RW 0x0	<p>This register specifies the block size for data transfers. 0 = If &lt;BlockCount&gt; =0, it is an infinite transfer. The block count should be 0 if &lt;BlockSize&gt; = 0. 1 = 1 byte . . . 2048 = 2048 bytes The current value of this &lt;BlockSize&gt; field is reflected in Current Number of Bytes Remaining n Data Block Register.</p>
15:12	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 681: Data Block Count Register**

Offset: 0x9000C

Bit	Field	Type/InitVal	Description
15:0	BlockCount	RW 0x0	<p>This field holds the initial value of the number of blocks. It is not decremented during the transfer. Instead, the remaining blocks left to send are listed in the Current Number of Bytes Remaining in Data Block Register (offset: 0x8007C) 0 = Infinite block count until the Host driver stops the transfer. If &lt;BlockSize&gt; is 0, it is an infinite transfer. If &lt;BlockSize&gt; is not 0, the Host controller will act as if it transferred multiple blocks, with the block count infinite. 1 = 1 block . . . 65535 = 65535 blocks</p>
31:16	Reserved	RSVD 0x0	Reserved

**Table 682: Argument in Command 16 LSB Register**

Offset: 0x90010

Bit	Field	Type/InitVal	Description
15:0	ArgLow	RW 0x0	16 LSB of the command argument. This value is inserted into the 48-bit command token (bits [23:8]).

**Table 682: Argument in Command 16 LSB Register (Continued)**  
Offset: 0x90010

Bit	Field	Type/InitVal	Description
31:16	Reserved	RSVD 0x0	Reserved

**Table 683: Argument in Command 16 MSB Register**  
Offset: 0x90014

Bit	Field	Type/InitVal	Description
15:0	ArgHigh	RW 0x0	16 MSB of the command argument. This value is inserted into the 48-bit command token bits[39:24].
31:16	Reserved	RSVD 0x0	Reserved

**Table 684: Transfer Mode Register**  
Offset: 0x90018

Bit	Field	Type/InitVal	Description
rcp controls the pause when recording.			
0	SwWrDataStart	RW 0x0	Write 1 to this bit after software decode of the response initiates the write data transfer. 0 = NoTransfer: No write data transfer. 1 = InitWrDataXfer: After software decodes of the response, initiates the write data transfer.
1	HwWrDataEn	RW 0x0	When the Host controller hardware receives response from the card, it has a option to initiate a write data transfer to the card or to wait for the software to enable the write data transfer by writing 1 to bit[0] of this register. For read data transfer, the hardware always looks for the start bit of the read data from the card. 0 = HWInitWrDataXfer: The hardware always initiates the write data transfer after receiving a response, regardless whether there is an error response or not. 1 = SWInitWrDataXfer: The hardware waits for the software to initiate the write data transfer by writing to bit[0] of this register.
2	AutoCMD12En	RW 0x0	Multiple block transfer for memory requires CMD12 to stop the transaction. When this bit is 1, Host controller automatically issues CMD12 when the last block transfer is completed. If this bit is set to 0, the software is responsible for issuing the CMD12 to stop the transfer, and soft reset the host controller. 0 = HostIssuesCMD12: Host controller automatically issues CMD12 when the last block transfer is completed. 1 = SWIssuesCMD12: The software must issue the CMD12 to stop the transfer, and soft reset the host controller.



**Table 684: Transfer Mode Register (Continued)**  
Offset: 0x90018

Bit	Field	Type/InitVal	Description
3	IntChkEn	RW 0x0	In case of SDIO, if this bit is set to 1, the Host controller checks for interrupts on DAT[1] in both 1-bit an 4-bits mode. 0 = NoInterruptCheck: The Host controller does not check for interrupts on DAT[1]. 1 = CheckInterrupts: The Host controller checks for interrupts on DAT[1] in both 1-bit an 4-bits mode.
4	DataXferTowardHost	RW 0x0	Data transfer direction selection This bit defines the direction of DAT line data transfer. The bit is set to 1 by Host Driver to transfer data from SD card to SD host controller, and it is set to 0 for all other commands. 0 = OtherCommands: All other commands 1 = XferDataToSDHost: Transfers data from the SD card to the SD host controller
5	StopClkEn	RW 0x0	This bit is set to 1 to stop the SD clocks in the following cases, even if clk_en is set to 1. 1) A command with no response: stop 16 clocks after the host command end bit 2) A command with response without data: stop 16 clocks after the card response end bit. 3) A read data transaction: 16 clocks after the end bit of the last data block. 4) A write data transaction: 16 clocks after the CRC status token returned by card for last data block. If this bit is set to 0, SDCLK is always active 0 = SDClocksActive: SD clocks are always active 1 = StopSDClocks: Stops the SD clocks, even clk_en is set to 1.
6	HostXferMode	RW 0x0	Host Transfer Mode 0 = DMA 1 = SW Write
15:7	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 685: Command Register**  
**Offset: 0x9001C**

Bit	Field	Type/InitVal	Description
1:0	RespType	RW 0x0	<p>Response type select.            Note: CRC field for R3 and R4 is expected to be all 1 bits. CRC check should be disabled for these response types.</p> <p>0 = NoResponse: No response.            1 = 136BitResponse: Response length is 136 bits.            2 = 48BitResponse: Response length is 48 bits.            3 = 48BitResponseChkBusy: Response length is 48 bits and check busy after response.</p>
2	DataCrc16ChkEn	RW 0x0	<p>Data CRC16 Check Enable            0 = DisableCrc16Chk: Disable read data crc16 check.            1 = EnableCrc16Chk: Enable read data crc16 check.</p>
3	CmdCrcChkEn	RW 0x0	<p>If this bit is set to 1, Host controller checks the CRC field in the response. If an error is detected, it is reported as a Command CRC error. The number of bits checked by CRC field value varies according to the length of response.</p> <p>0 = NoCheck: No checking of CRC.            1 = HCCrcChk: Host controller checks the CRC field in the response. If an error is detected, it is reported as a Command CRC error.</p>
4	CmdIndexChkEn	RW 0x0	<p>Command index check enable.            If this bit is set to 1, the Host controller checks the index field in the response to see if it has the same value as the command index. If the value does not match, it is reported as a Command Index Error.</p> <p>0 = NoCheck: Host controller does not check the index field.            1 = HCChkIndex: Host controller checks the index field.</p>
5	DataPresent	RW 0x0	<p>This bit is set to 1 to indicate that data is present and will be transferred using the DATA line.            It is set to 0 for the followings:            1) Commands using only CMD lines            2) Commands with no data transfer, but using busy signal on DAT[0] line (for example, CMD38).</p> <p>0 = NoDataTransfer: 1) Commands using only CMD lines OR 2) Commands with no data transfer, but using busy signal on DAT[0] line (for example, CMD38).            1 = DataAwaitsTransfer: Indicates that data is present and will be transferred using the DATA line.</p>
6	RSVD	RO 0x0	Reserved Read only.
7	UnexpectedRespEn	RW 0x0	Enable hardware to detects unexpected response from device
13:8	CmdIndex	RW 0x0	<p>Command index            These bits will be inserted into Command token bits[45:40] of the response pattern (for more details, see the SD Memory Card Specifications).</p>



**Table 685: Command Register (Continued)**  
Offset: 0x9001C

Bit	Field	Type/InitVal	Description
15:14	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 686: Response Halfword 0 Register**  
Offset: 0x90020

Bit	Field	Type/InitVal	Description
15:0	Resp0	RO 0x0	1) For 48-bit response token: This register contains bits[45:30] of the response token. 2) For 136-bit response token: This register contains bits[133:118] of the response token.
31:16	Reserved	RSVD 0x0	Reserved

**Table 687: Response Halfword 1 Register**  
Offset: 0x90024

Bit	Field	Type/InitVal	Description
15:0	Resp1	RO 0x0	1) For 48-bit response token: This register contains bits[29:14] of the response token. 2) For 136-bit response token: This register contains bits[117:102] of the response token.
31:16	Reserved	RSVD 0x0	Reserved

**Table 688: Response Halfword 2 Register**  
Offset: 0x90028

Bit	Field	Type/InitVal	Description
15:0	Resp2	RO 0x0	1) For 48-bit response token: This register contains bits[13:8] of the response token. 2) For 136-bit response token: This register contains bits[101:86] of the response token.
31:16	Reserved	RSVD 0x0	Reserved

**Table 689: Response Halfword 3 Register**  
**Offset: 0x9002C**

Bit	Field	Type/InitVal	Description
15:0	Resp3	RO 0x0	1) For 48-bit response token: Not relevant. 2) For 136-bit response token: This register contains bits[85:70] of the response token.
31:16	Reserved	RSVD 0x0	Reserved

**Table 690: Response Halfword 4 Register**  
**Offset: 0x90030**

Bit	Field	Type/InitVal	Description
15:0	Resp4	RO 0x0	1) For 48-bit response token: Not relevant. 2) For 136-bit response token: This register contains bits[69:54] of the response token.
31:16	Reserved	RSVD 0x0	Reserved

**Table 691: Response Halfword 5 Register**  
**Offset: 0x90034**

Bit	Field	Type/InitVal	Description
15:0	Resp5	RO 0x0	1) For 48-bit response token: Not relevant. 2) For 136-bit response token: This register contains bits[53:38] of the response token.
31:16	Reserved	RSVD 0x0	Reserved

**Table 692: Response Halfword 6 Register**  
**Offset: 0x90038**

Bit	Field	Type/InitVal	Description
15:0	Resp6	RO 0x0	1) For 48-bit response token: Not relevant. 2) For 136-bit response token: This register contains bits[37:22] of the response token.
31:16	Reserved	RSVD 0x0	Reserved



**Table 693: Response Halfword 7 Register**  
Offset: 0x9003C

Bit	Field	Type/InitVal	Description
13:0	Resp7	RO 0x0	1) For 48-bit response token: Not relevant. 2) For 136-bit response token: This register contains bits[21:8] of the response token.
15:14	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 694: 16-bit Data Word Accessed by CPU Register**  
Offset: 0x90040

Bit	Field	Type/InitVal	Description
15:0	CpuData	RW 0x0	The CPU can access the SD data in the FIFO through this data port register. The first access is to the 16 LSB of the FIFO word, and the second access is to the 16 MSB of FIFO word. <b>NOTE:</b> The firmware needs to write or read multiple numbers of WORDS to/from this register even though the transfer size is not a multiple of words. The hardware expects that the CPU write/read multiple of words to/from this register. SD side knows when to stop the transfer of data to/from the SD cards because it has the byte and block count.
31:16	Reserved	RSVD 0x0	Reserved

**Table 695: CRC7 of I Response Register**  
Offset: 0x90044

Bit	Field	Type/InitVal	Description
6:0	CRC7RespToken	RW 0x0	This field contains bits[7:1] (CRC7) of the response token.
15:7	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 696: Host Present State 16 LSB Register**  
**Offset: 0x90048**

<b>Bit</b>	<b>Field</b>	<b>Type/InitVal</b>	<b>Description</b>
0	CmdInhibitCmd	RO 0x0	If this bit is 0, it indicates the CMD line is not in use, and the Host controller can issue a command using the CMD line. This bit is set after the Command register is written. This bit is cleared when the command response is received. Even if the CmdInhibitDat is set to 1, commands using only the CMD line can be issued if this bit is 0. Changing from 1 to 0 generates a command complete interrupt in the Normal Interrupt status register. If the Host controller cannot issue the command because of a command conflict error, this bit remains 1, and the command complete is not set. 0 = CmdRespRecv: This bit is cleared when the command response is received. 1 = CmdRegWrite: This bit is set after the command register is written.
1	CardBusy	RO 0x0	Card busy This bit tells the host that Card is busy, It is set and clear by Host controller.
2	RSVD	RO 0x0	Reserved
6:3	DatLevel	RO 0xf	DAT[3:0] Line Signal Level. This status is used to check the DAT line level to recover from errors, and for debugging. This field is especially useful in detecting the busy signal level from DAT[0].
7	CmdLevel	RO 0x1	CMD line Signal Level This status is used to check the CMD line level to recover from errors, and for debugging.
8	TxActive	RO 0x0	Indicates that write transfer is active. If this bit is 0, it means no valid write data exists in the Host controller. A transfer complete interrupt is generated when all write data is out. Besides, during a write transaction, a Block Gap Event interrupt generated when this bit is changed to 0, as result of the Stop At Block Gap Request being set. This status is useful for the Host driver in determining when to issue commands during write busy. 0 = TxDisabled: No valid write data exists in the Host controller. This bit is cleared in either of the following cases: 1) After getting the CRC status of the last data block as specified by the transfer count (single and multiple). 2) After getting the CRC status of any block where data transmission is about to be stopped by a Stop At Block Gap Request. 1 = TxEnabled: This bit is set either of the following cases: 1) After the end bit of the write command. 2) When writing a 1 to Continue Request in the Block Gap Control register, to restart a write transfer.



**Table 696: Host Present State 16 LSB Register (Continued)**  
Offset: 0x90048

Bit	Field	Type/InitVal	Description
9	RxActive	RO 0x0	Indicates that the read transfer is active. Note: Sop_at_blk_gap status is set to 1, if this bit is changed from 1 to 0. 0 = RxEnabled: This bit is cleared to 0 for the followings: 1) When the last data block as specified by block length is transferred to the system. Transfer complete status is set to 1, if this bit is changed from 1 to 0. 2) When all valid data blocks have been transferred to the system, and no current block transfers are being sent as a result of the Stop At Block Gap. 1 = RxDisabled: This bit is set to 1 for the followings: 1) After the end bit of the read command 2) When writing 1 to Continue Request in the Block Gap Control register to restart a read transfer.
11:10	RSVD	RO 0x0	Reserved
12	FifoFull	RO 0x0	FIFO Full
13	FifoEmpty	RO 0x1	FIFO Empty
14	AutoCmd12Active	RO 0x0	Indicates that auto_cmd12 is active.
15	RSVD	RO 0x0	Reserved
31:16	Reserved	RSVD 0x0	Reserved

**Table 697: Host Control Register**  
Offset: 0x90050

Bit	Field	Type/InitVal	Description
0	PushPullEn	RW 0x0	Push-pull enable. The host driver should program this bit to 1 (push-pull type) after the Card ID phase is completed. 0 = OpenDrain: Open drain on CMD line 1 = PushPullEn: Push-pull on CMD line
2:1	CardType	RW 0x0	Card type 0 = MemoryOnly: Memory only SD card 1 = IoOnly: IO only SD card 2 = IoMemCombo: IO and memory combo card 3 = MMC: MMC card

**Table 697: Host Control Register (Continued)**  
Offset: 0x90050

Bit	Field	Type/InitVal	Description
3	BigEndian	RW 0x0	Big Endian Enable 0 = LittleEndian: Little Endian: for transmitting, the host controller gets 32 bits of data from the internal buffer and shifts out the MSB bit first. For receiving, it does the same thing. 1 = BigEndian: Big Endian: for transmitting, the host controller swaps data bytes within a 32-bit data word, i.e., byte0 becomes byte3, byte 1 becomes byte 2, byte2 becomes byte 1, and byte 3 becomes byte 0. For receiving, the host controller swaps as was done for transmitting before writing to the internal buffer.
4	LsbFirst	RW 0x0	After bit[3] of this register <BigEndian> does its job, this bit will do the following for both transmitting and receiving: 0 = MSB: Shifts MSB (bit[32] of a Word) first. 1 = LSB: Shifts LSB (bit[0] of a Word) first.
8:5	RSVD	RO 0x0	Reserved Read only.
9	DataWidth	RW 0x0	Data width. 0 = 1BitDataMode: 1-bit data mode, using only DAT[0]. 1 = 4BitDataMode: 4-bit data mode enabled.
10	HiSpeedEn	RW 0x0	High speed enable. If this bit is set to 0 (default), the Host controller outputs CMD and DAT lines at the falling edge of the SD clock (up to 25 MHz). If this bit is set to 1, the Host controller outputs CMD and DAT lines at the rising edge of the SD clock (up to 50 MHz). 0 = NormalSpeed: Normal speed (high speed disable) 1 = HighSpeedEnable: High speed enable



**Table 697: Host Control Register (Continued)**  
Offset: 0x90050

Bit	Field	Type/InitVal	Description
14:11	TimeoutValue	RW 0xF	<p>Determines the interval by which the DAT line timeouts are detected. This timeout is initiated in these cases:</p> <p>1) Read transaction: waiting for data from cards. This is Nac timing value in the SD specification, specifying the maximum timing from a read command to a read data (card data access time).</p> <p>2) Write transaction: waiting for data from the IMB slave, IMB Master, or CPU.</p> <p>For others transaction, there are fixed timeout as followings (unit in SDCLK cycles)</p> <p>-Card side</p> <p>Ncr=64: maximum timing value from command to response. Nid=64 (5 in specification): maximum timing value from command to OCR response.</p> <p>- Host side</p> <p>Nrc=8: minimum timing value from response to next command. Ncc=8: minimum timing value from command to next command. Nwr=2: minimum timing value from data CRC status (from card in write transaction) to next write data in multiple write blocks Nst=2: minimum timing from STOP command to end of write data</p> <p>0 = SDCLK x 2^12: Timeout value = SDCLK x 2^12 1 = SDCLK x 2^13: Timeout value = SDCLK x 2^13 2 = SDCLK x 2^14: Timeout value = SDCLK x 2^14 3 = SDCLK x 2^15: Timeout value = SDCLK x 2^15 4 = SDCLK x 2^16: Timeout value = SDCLK x 2^16 5 = SDCLK x 2^17: Timeout value = SDCLK x 2^17 6 = SDCLK x 2^18: Timeout value = SDCLK x 2^18 7 = SDCLK x 2^19: Timeout value = SDCLK x 2^19 8 = SDCLK x 2^20: Timeout value = SDCLK x 2^20 9 = SDCLK x 2^21: Timeout value = SDCLK x 2^21 10 = SDCLK x 2^22: Timeout value = SDCLK x 2^22 11 = SDCLK x 2^23: Timeout value = SDCLK x 2^23 12 = SDCLK x 2^24: Timeout value = SDCLK x 2^24 13 = SDCLK x 2^25: Timeout value = SDCLK x 2^25 14 = SDCLK x 2^26: Timeout value = SDCLK x 2^26 15 = SDCLK x 2^27: Timeout value = SDCLK x 2^27</p>
15	TimeoutEn	RW 0x0	<p>Enable timeout value counter.</p> <p>If 0, host controller never receives a timeout.</p> <p>0 = NoTimeout: Host controller never receives a timeout.</p> <p>1 = Timeout: Host controller receives a timeout.</p>
31:16	Reserved	RSVD 0x0	Reserved

**Table 698: Data Block Gap Control Register**  
Offset: 0x90054

Bit	Field	Type/InitVal	Description
0	StopAtBlockGapReq	RW 0x0	<p>Stop at block gap request.</p> <p>This it is used to stop executing a transaction at the next block gap for both DMA and non-DMA transfers. Until the transfer complete is set to 1, indicating a transfer completion, the Host driver leaves this bit set to 1. Clearing both the Stop At Block Gap Request and the Continue Request does not cause the transaction to restart. Read Wait is used to stop the read transaction at the block gap. The host controller stops the clock At Block Gap Request for write transfer, but for read transfer, it stops the clock if &lt;RdWaitCtl&gt; is 0. Otherwise, the host controller issues a Read Wait command to stop read data.</p>
1	ContReq	RW1C 0x0	<p>Continue request.</p> <p>This bit is used to restart a transaction which was stopped using the Stop At Block Gap Request. To cancel stop at the block gap, set Stop At Block Gap Request to 0 and set this bit to 1 to restart the transfer.</p> <p>The host controller automatically clears this bit in either of the following cases:</p> <ol style="list-style-type: none"> <li>1) During a read transaction, the DAT Line Active changes from 0 to 1 as a read transaction restart.</li> <li>2) During a write transaction, the Write Transfer Active changes from 0 to 1 as the write transaction restarts.</li> </ol> <p>Therefore, it is not necessary for Host driver to set this bit to 0. If Stop At Block Gap Request is set to 1, any write to this bit is ignored.</p>
2	RdWaitCtl	RW 0x0	<p>Enable read wait protocol</p> <p>If the card supports read wait, set this bit to enable use of the read wait protocol to stop reading data using the DAT[2] line by Host hardware. Otherwise, the Host controller has to stop the SD clock to hold read data. When the Host driver detects a card insertion, it sets this bit according to the CCCR of the SDIO card.</p> <p><b>NOTE:</b> This bit is looked at only at block gap. Within a block, hardware stalls the clock top stop read data if the host cannot accept additional data because of the FIFO buffer is full.</p> <p><b>NOTE:</b> When this bit is cleared by firmware, the operation continues. During read wait, firmware can issue a different command for a different operation, as long as it does not require DATA lines. To continue the waiting operation, firmware needs to write 0 to this register.</p> <p>0 = Disable: Disable read wait protocol  1 = EnableRdWait: Enable read wait protocol</p>
3	StopDatXfer	RW 0x0	<p>Stop data transfer</p> <p>0 = Disable: Data transfer continues.  1 = StopDataXferEn: Stops data transfer immediately.</p>
4	Resume	Special 0x0	<p>Write 1 to this register to resume the operation.</p> <p>Note: A write to the Command register is required to start the resume operation.</p> <p>This bit is cleared by hardware when hardware resumes.</p> <p><b>NOTE:</b> Write 1 and clear.</p>



**Table 698: Data Block Gap Control Register (Continued)**  
Offset: 0x90054

Bit	Field	Type/InitVal	Description
5	Suspend	Special 0x0	Write 1 to this register to suspend the operation at the block gap. This bit is cleared by hardware when the hardware is suspended. When suspended, which is indicated in status register, the firmware needs to store all register values, and the remaining block counts in the Current Number of Data Blocks Left to be Transferred Register are restored to the Data Block Count Register when resumed.  <b>NOTE:</b> Write 1 and clear.
15:6	RSVD	RO 0x0	Reserved
31:16	Reserved	RSVD 0x0	Reserved

**Table 699: Clock Control Register**  
Offset: 0x90058

Bit	Field	Type/InitVal	Description
0	SclkMasterEn	RW 0x0	SD clock master enable. For valid operation, this field must be 0x0.  0 = SdclkNotSet: The SD clock toggles sync with valid data. 1 = SdclkEn: The SD clock toggles always.
15:1	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 700: Software Reset Register**  
Offset: 0x9005C

Bit	Field	Type/InitVal	Description
7:0	RSVD	RO 0x0	Reserved Read only.
8	SwReset	Special 0x0	Software reset for all. This reset affects the status, state machine, and FIFOs synchronously. The configuration is not affected. <b>NOTE:</b> This register is Read Write and Clear.
15:9	RSVD	RO 0x0	Reserved Read only.

**Table 700: Software Reset Register (Continued)**  
Offset: 0x9005C

Bit	Field	Type/InitVal	Description
31:16	Reserved	RSVD 0x0	Reserved

**Table 701: Normal Interrupt Status Register**  
Offset: 0x90060

Bit	Field	Type/InitVal	Description
0	CmdComplete	RW1C 0x0	Command Complete This bit is set when the end bit of the command response is received (except for the Auto CMD12). <b>NOTE:</b> The Command Timeout Error has higher priority than the Command complete.
1	XferComplete	RW1C 0x0	Transfer Complete This bit is set when a read or write transaction is completed. 1) Read transaction: This bit is set at the falling edge of Read Transfer Active Status. There are two cases when this occurs: - Data transfer is completed as specified by the data length. - Data transfer is stopped at the block gap, and data transfer is completed by setting the Stop At Block Gap Request. 2) Write transaction: This bit is set at the falling edge of the DAT Line Active status. There are two cases when this occurs: - Data transfer is completed as specified by the data length and the busy signal is released. - Data transfer stopped at the block gap, and data transfer is completed by setting the Stop At Block Gap Request.
2	BlockGapEv	RW1C 0x0	Block gap event. If the Stop At Block Gap Request in the Block Gap Control register is set, this bit is set when both a read or a write transaction is stopped at a block gap. If Stop At Block Gap Request is not set to 1, this bit is not set to 1.
3	Dmalnt	RW1C 0x0	DMA interrupt. This status is set if the host controller detects DMA end.
4	TxRdy	RO 0x1	The FIFO has room for the CPU to write 16 bits of data.
5	RxRdy	RO 0x0	At least 1 byte of data is in the FIFO and ready to be read by the CPU.
7:6	RSVD	RO 0x0	Reserved Read only.
8	CardInt	RW1C 0x0	Card interrupt. This bit is set to 1 if the host controller detects an interrupt from the card.



**Table 701: Normal Interrupt Status Register (Continued)**  
Offset: 0x90060

Bit	Field	Type/InitVal	Description
9	ReadWaitOn	RW1C 0x0	Read Wait state is on. This bit is set to 1 to indicate to the firmware that the host controller is in Read Wait state (for SDIO cards). The host writes 1 to this bit to clear and restart data transfer, and to exit the host controller from Read Wait state.
10	ImbFifo8wFull	RO 0x0	There are at least eight filled entries for data to be read from the FIFO.
11	ImbFifo8wAvail	RO 0x1	There are at least eight empty entries for data to be written in the FIFO.
12	SuspenseOn	RW1C 0x0	Indicates that the hardware is suspended as the result of writing 1 to the <Suspend> field of the Transfer Mode register.
13	AutoCmd12Complete	RW1C 0x0	Auto_cmd12 is completed.
14	UnexpectedRespDet	RW1C 0x0	Unexpected response from devices detected.
15	ErrInt	RO 0x0	Error interrupt. If any of bits in the Error Interrupt status register are set, then this bit is set.
31:16	Reserved	RSVD 0x0	Reserved

**Table 702: Error Interrupt Status Register**  
Offset: 0x90064

Bit	Field	Type/InitVal	Description
0	CmdTimeoutErr	RW1C 0x0	Command timeout error. This bit is set when no response is returned within 64 SDCLK cycles from the end bit of the command.
1	CmdCrcErr	RW1C 0x0	Command CRC Error This bit is set in two cases: 1) Upon detecting a CRC error in the command response. 2) When the host controller detects a CMD line conflict, by monitoring the CMD line when a command is issued. The host controller aborts the command (stops driving the CMD line). The Command Timeout Error is also set to 1 to distinguish the CMD line conflict.
2	CmdEndBitErr	RW1C 0x0	Command end bit error This bit is set when detecting that the end bit of a command response is 0.

**Table 702: Error Interrupt Status Register (Continued)**  
Offset: 0x90064

Bit	Field	Type/InitVal	Description
3	CmdIndexErr	RW1C 0x0	Command Index Error This bit is set when a command index error occurs in the command response.
4	DataTimeoutErr	RW1C 0x0	Data timeout error This bit is set when detecting one of the followings: - Busy timeout after Write CRC status - Write CRC status timeout - Read data timeout
5	RdDataCRCErr	RW1C 0x0	Read data CRC error This bit is set to 1 when transferring read data, which uses the DAT line, or when detecting the write CRC status having a value of other than 010.
6	RdDataEndBitErr	RW1C 0x0	Read data end bit error This bit is set to 1 when detecting 0 at the end bit position of the read data, which uses the DAT line, or at the end bit position of the CRC status.
7	RSVD	RO 0x0	Reserved Read only.
8	AutoCmd12Err	RW1C 0x0	Auto CMD12 error. This error occurs when detecting that one of the bits in the Auto CMD12 Error Status register has changed from 0 to 1.
9	CmdStartBitErr	RW 0x0	Command start bit error
10	XferSizeErr	RW1C 0x0	Transfer size mismatched error. This bit indicate that the transferred data size on the SD and on the host size is mismatched. For instance, SD side keeps track of the block count and block size. It knows when the data ends. The host side, that is the IMB Slave and CPU access, is supposed to know the transferred data size as well. In the normal case, both sides end with no extra byte remaining. If there is any data left in the FIFO when the block count and block size are exhausted, this interrupt is generated. In the case of an IMB Master Read (read from cards), the SD side will indicate the end of the packet in FIFO data bit [33]. In the case of an IMB Master Write, because of synchronization, the SD side may end while the IMB Master still requests data from the SDRAM. In this case, the SD side will tell the DMA to stop the request and flush the FIFO.
11	RespTbitErr	RW1C 0x0	Response T bit error.



**Table 702: Error Interrupt Status Register (Continued)**  
Offset: 0x90064

Bit	Field	Type/InitVal	Description
12	CrcEndBitErr	RW1C 0x0	CRC end bit error The CRC status end bit is not at the expected logic level in a write transaction.
13	CrcStartBitErr	RW1C 0x0	CRC start bit error The CRC status start bit is not at the expected logic level in a write transaction.
14	CrcStatErr	RW1C 0x0	CRC status error The CRC status returned from the card is not good in a write transaction.
15	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 703: Normal Interrupt Status Enable Register**  
Offset: 0x90068

Bit	Field	Type/InitVal	Description
0	CmdCompleteEn	RW 0x0	Command complete enable.
1	XferCompleteEn	RW 0x0	Transfer complete enable.
2	BlockGapEvEn	RW 0x0	Block gap event enable.
3	DmaIntEn	RW 0x0	DMA interrupt enable.
4	WrRdyEn	RW 0x0	Buffer write ready enable.
5	RdRdyEn	RW 0x0	Buffer read ready enable.
7:6	RSVD	RO 0x0	Reserved Read only.
8	CardIntEn	RW 0x0	Card interrupt enable.
9	RdWaitOnEn	RW 0x0	Read Wait on enable.
11:10	RSVD	RO 0x0	Reserved Read only.
12	SuspenseOnEn	RW 0x0	Suspense on enable.

**Table 703: Normal Interrupt Status Enable Register (Continued)**  
Offset: 0x90068

Bit	Field	Type/InitVal	Description
13	AutoCmd12CompleteEn	RW 0x0	Auto_cmd12 complete enable.
14	UnexpectedRespDetEn	RW 0x0	Unexpected response from devices detected enable
15	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 704: Error Interrupt Status Enable Register**  
Offset: 0x9006C

Bit	Field	Type/InitVal	Description
0	CmdTimeoutErrEn	RW 0x0	Command timeout error enable.
1	CmdCrcErrEn	RW 0x0	Command CRC error enable.
2	CmdEndBitErrEn	RW 0x0	Command end bit error enable.
3	CmdIndexErrEn	RW 0x0	Command index error enable.
4	DataTimeoutErrEn	RW 0x0	Data timeout error enable.
5	RdDataCrcErrEn	RW 0x0	Data CRC error enable.
6	RdDataEndBitErrEn	RW 0x0	Data end bit error enable.
7	RSVD	RO 0x0	Reserved Read only.
8	AutoCmd12ErrEn	RW 0x0	Auto CMD12 error enable.
9	CmdStartBitErrEn	RW 0x0	Command start bit error enable.
10	SizeErrEn	RW 0x0	Size error enable.
11	RespTbitErrEn	RW 0x0	Response T bit error enable.
12	CrcEndBitErrEn	RW 0x0	CRC status end bit error enable.



**Table 704: Error Interrupt Status Enable Register (Continued)**  
Offset: 0x9006C

Bit	Field	Type/InitVal	Description
13	CrcStartBitErrEn	RW 0x0	CRC status start bit error enable.
14	CrcStatusErrEn	RW 0x0	CRC status error enable.
15	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 705: Normal Interrupt Status Interrupt Enable Register**  
Offset: 0x90070

Bit	Field	Type/InitVal	Description
0	CmdCompleteIntEn	RW 0x0	Command complete interrupt enable.
1	XferCompleteIntEn	RW 0x0	Transfer complete interrupt enable.
2	BlockGapEvtIntEn	RW 0x0	Block gap event interrupt enable.
3	DmaIntIntEn	RW 0x0	DMA interrupt interrupt enable.
4	TxRdyIntEn	RW 0x0	Buffer write ready interrupt enable.
5	RxRdyIntEn	RW 0x0	Buffer read ready interrupt enable.
7:6	RSVD	RW 0x0	Reserved Read only.
8	CardIntIntEn	RW 0x0	Card interrupt interrupt enable.
9	RdWaitOnIntEn	RW 0x0	Read Wait on interrupt enable.
10	ImbFifo8wFullIntEn	RO 0x0	IMB FIFO 8 word filled interrupt enable.
11	ImbFifo8wAvailIntEn	RO 0x0	IMB FIFO 8 word available interrupt enable.
12	SuspenseOnIntEn	RW 0x0	Suspense on interrupt enable.
13	AutoCmd12CompleIntEn	RO 0x0	Auto Cmd12 complete interrupt enable.

**Table 705: Normal Interrupt Status Interrupt Enable Register (Continued)**  
Offset: 0x90070

Bit	Field	Type/InitVal	Description
14	UnexpectedRespDetIntEn	RO 0x0	Unexpected response from devices detected interrupt enable.
15	RSVD	RW 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 706: Error Interrupt Status Interrupt Enable Register**  
Offset: 0x90074

Bit	Field	Type/InitVal	Description
0	CmdTimeoutErrIntEn	RW 0x0	Command timeout error interrupt enable.
1	CmdCrcErrIntEn	RW 0x0	Command CRC error interrupt enable.
2	CmdEndBitErrIntEn	RW 0x0	Command end bit error interrupt enable.
3	CmdIndexErrIntEn	RW 0x0	Command index error interrupt enable.
4	DataTimeoutErrIntEn	RW 0x0	Data timeout error interrupt enable.
5	RdDataCrcErrIntEn	RW 0x0	Data CRC error interrupt enable.
6	RdDataEndBitErrIntEn	RW 0x0	Data end bit error interrupt enable.
7	RSVD	RO 0x0	Reserved Read only.
8	AutoCmd12ErrIntEn	RW 0x0	Auto CMD12 error interrupt enable.
9	CmdStartBitErrIntEn	RW 0x0	Command start bit error interrupt enable.
10	SizeErrIntEn	RW 0x0	Size error interrupt enable.
11	RespTbitErrIntEn	RW 0x0	Response T bit error interrupt enable.
12	CrcStatusEndBitErrIntEn	RW 0x0	CRC status end bit error interrupt enable.
13	CrcStatusStartBitErrIntEn	RW 0x0	CRC status start bit error interrupt enable.



**Table 706: Error Interrupt Status Interrupt Enable Register (Continued)**  
Offset: 0x90074

Bit	Field	Type/InitVal	Description
14	CrcStatusErrIntEn	RW 0x0	CRC status error interrupt enable.
15	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 707: Auto CMD12 Interrupt Status Register**  
Offset: 0x90078

Bit	Field	Type/InitVal	Description
0	AutoCmd12NotExe	RW1C 0x0	Occurs when host controller cannot issue Auto Cmd12 to stop multiple block data transfer due to some errors.
1	AutoCmd12TimeoutEr	RW1C 0x0	Occurs if no response is returned within 64 SDCLK cycles from the end bit of the command.
2	AutoCmd12CrcEr	RW1C 0x0	Occurs when detecting CRC error in the command response.
3	AutoCmd12EndBitEr	RW1C 0x0	Occurs when detecting that the end bit of command response is 0.
4	AutoCmd12IndexEr	RW1C 0x0	Occurs if the command index error occurs in response to a command error.
5	AutoCmd12RespTBitEr	RW1C 0x0	Auto Cmd12 Response T bit error.
6	AutoCmd12RespStartBitEr	RW1C 0x0	Auto Cmd12 response start bit not detected, and timeout.
15:7	RSVD	RW1C 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 708: Current Number of Bytes Remaining in Data Block Register**  
Offset: 0x9007C

Bit	Field	Type/InitVal	Description
11:0	BlockSize	RO 0x0	Block Size This field shows the remaining number of bytes in the current block.

**Table 708: Current Number of Bytes Remaining in Data Block Register (Continued)**  
Offset: 0x9007C

Bit	Field	Type/InitVal	Description
15:12	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 709: Current Number of Data Blocks Left to Be Transferred Register**  
Offset: 0x90080

Bit	Field	Type/InitVal	Description
15:0	BlockCount	RO 0x0	Block count. This field shows the remaining number of blocks in the current transfer.
31:16	Reserved	RSVD 0x0	Reserved

**Table 710: Argument in Auto Cmd12 Command 16 LSB Transferred Register**  
Offset: 0x90084

Bit	Field	Type/InitVal	Description
15:0	AutoCmd12ArgLo	RW 0x0	The 16 LSB of the auto cmd12 command argument. This value is inserted into the 48-bit auto cmd12 command token bits[23:8].
31:16	Reserved	RSVD 0x0	Reserved

**Table 711: Argument in Auto Cmd12 Command 16 MSB Transferred Register**  
Offset: 0x90088

Bit	Field	Type/InitVal	Description
15:0	AutoCmd12ArgHi	RW 0x0	The 16 MSB of the auto cmd12 command argument. This value is inserted into the 48-bit auto cmd12 command token bits[39:24].
31:16	Reserved	RSVD 0x0	Reserved



**Table 712: Index of Auto Cmd12 Commands Transferred Register**  
Offset: 0x9008C

Bit	Field	Type/InitVal	Description
0	AutoCmd12BusyChkEn	RW 0x0	Checks for busy after the auto cmd12 response.
1	AutoCmd12IndexChkEn	RW 0x0	Index check for auto cmd12.
7:2	RSVD	RO 0x0	Reserved Read only.
13:8	AutoCmd12Index	RW 0x0	Auto cmd12 command index. These bits will be inserted into the auto cmd12 command token bits[45:40].
15:14	RSVD	RO 0x0	Reserved Read only.
31:16	Reserved	RSVD 0x0	Reserved

**Table 713: Auto Cmd12 Response Halfword 0 Register**  
Offset: 0x90090

Bit	Field	Type/InitVal	Description
15:0	AutoCmd12Resp0	RO 0x0	This register contains bits[45:30] of the cmd12 response token.
31:16	Reserved	RSVD 0x0	Reserved

**Table 714: Auto Cmd12 Response Halfword 1 Register**  
Offset: 0x90094

Bit	Field	Type/InitVal	Description
15:0	AutoCmd12Resp1	RO 0x0	This register contains bits[29:14] of cmd12 response token.
31:16	Reserved	RSVD 0x0	Reserved

**Table 715: Auto Cmd12 Response Halfword 2 Register**  
Offset: 0x90098

Bit	Field	Type/InitVal	Description
15:0	AutoCmd12Resp2	RO 0x0	This register contains bits[13:8] of the cmd12 response token.

**Table 715: Auto Cmd12 Response Halfword 2 Register (Continued)**  
Offset: 0x90098

Bit	Field	Type/InitVal	Description
31:16	Reserved	RSVD 0x0	Reserved

**Table 716: Mbus Control Low Register**  
Offset: 0x90100

Bit	Field	Type/InitVal	Description
3:0	SdArbEntry0	RW 0x0	SD Arbitrator Entry 0.
7:4	SdArbEntry1	RW 0x1	SD Arbitrator Entry 1.
11:8	SdArbEntry2	RW 0x2	SD Arbitrator Entry 2.
15:12	SdArbEntry3	RW 0x3	SD Arbitrator Entry 3.
19:16	SdArbEntry4	RW 0x4	SD Arbitrator Entry 4.
23:20	SdArbEntry5	RW 0x5	SD Arbitrator Entry 5.
27:24	SdArbEntry6	RW 0x6	SD Arbitrator Entry 6.
31:28	SdArbEntry7	RW 0x7	SD Arbitrator Entry 7.

**Table 717: Mbus Control High Register**  
Offset: 0x90104

Bit	Field	Type/InitVal	Description
3:0	SdArbEntry8	RW 0x0	SD Arbitrator Entry 8.
7:4	SdArbEntry9	RW 0x1	SD Arbitrator Entry 9.
11:8	SdArbEntry10	RW 0x2	SD Arbitrator Entry 10.
15:12	SdArbEntry11	RW 0x3	SD Arbitrator Entry 11.
19:16	SdArbEntry12	RW 0x4	SD Arbitrator Entry 12.
23:20	SdArbEntry13	RW 0x5	SD Arbitrator Entry 13.



**Table 717: Mbust Control High Register (Continued)**  
Offset: 0x90104

Bit	Field	Type/InitVal	Description
27:24	SdArbEntry14	RW 0x6	SD Arbitrator Entry 14.
31:28	SdArbEntry15	RW 0x7	SD Arbitrator Entry 15.

**Table 718: Window0 Control Register**  
Offset: 0x90108

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window 0 Enable 0 = DisableWindow: Window is disabled. 1 = EnableWindow: Window is enabled.
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the target interface associated with this window.
15:8	Attr	RW 0x0E	Specifies the target interface attributes associated with this window.
31:16	Size	RW 0x0FFF	Window size. Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1's followed by a sequence of 0's. The number of 1's specifies the size of the window in 64 KByte granularity (e.g., a value of 0x00FF specifies 256 = 16 MByte). <b>NOTE:</b> A value of 0x0 specifies 64-KByte size.

**Table 719: Window0 Base Register**  
Offset: 0x9010C

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x0	Base address Used with the <Size> field of the Window0 Control Register to set the address window size and location. Corresponds to transaction address[31:16].

**Table 720: Window1 Control Register**  
**Offset: 0x90110**

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window 1 Enable 0 = DisableWindow: Window is disabled. 1 = EnableWindow: Window is enabled.
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the target interface associated with this window.
15:8	Attr	RW 0x0D	Specifies the target interface attributes associated with this window.
31:16	Size	RW 0xFFFF	Window size. Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1's followed by a sequence of 0's. The number of 1's specifies the size of the window in 64 KByte granularity (e.g., a value of 0x00FF specifies 256 = 16 MByte). <b>NOTE:</b> A value of 0x0 specifies 64-KByte size.

**Table 721: Window1 Base Register**  
**Offset: 0x90114**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x1000	Base address Used with the <Size> field of the Window1 Control Register to set the address window size and location. Corresponds to transaction address[31:16].

**Table 722: Window2 Control Register**  
**Offset: 0x90118**

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window 2 Enable 0 = DisableWindow: Window is disabled. 1 = EnableWindow: Window is enabled.
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the target interface associated with this window.
15:8	Attr	RW 0x0B	Specifies the target interface attributes associated with this window.



**Table 722: Window2 Control Register (Continued)**  
Offset: 0x90118

Bit	Field	Type/InitVal	Description
31:16	Size	RW 0x0FFF	Window size. Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1's followed by a sequence of 0's. The number of 1's specifies the size of the window in 64 KByte granularity (e.g., a value of 0x00FF specifies 256 = 16 MByte). <b>NOTE:</b> A value of 0x0 specifies 64-KByte size.

**Table 723: Window2 Base Register**  
Offset: 0x9011C

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x2000	Base address Used with the <Size> field of the Window2 Control Register to set the address window size and location. Corresponds to transaction address[31:16].

**Table 724: Window3 Control Register**  
Offset: 0x90120

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window 3 Enable 0 = DisableWindow: Window is disabled. 1 = EnableWindow: Window is enabled.
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the target interface associated with this window.
15:8	Attr	RW 0x07	Specifies the target interface attributes associated with this window.
31:16	Size	RW 0xFFFF	Window size. Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as a sequence of 1's followed by a sequence of 0's. The number of 1's specifies the size of the window in 64 KByte granularity (e.g., a value of 0x00FF specifies 256 = 16 MByte). <b>NOTE:</b> A value of 0x0 specifies 64-KByte size.

**Table 725: Window3 Base Register**  
**Offset: 0x90124**

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x3000	Base address Used with the <Size> field of the Window3 Control Register to set the address window size and location. Corresponds to transaction address[31:16].

**Table 726: Clock Divider Value Register**  
**Offset: 0x90128**

Bit	Field	Type/InitVal	Description
10:0	ClkDvdrMValue	RW 0x7CF	Clock divider value (m): SDIO clock is: $100/(m+1)$
31:11	Reserved	RSVD 0x0	Reserved

**Table 727: Address Decoder Error Register**  
**Offset: 0x9012C**

Bit	Field	Type/InitVal	Description
0	Add_Dec_Miss_Err	RW0C 0x0	Address Decoding Error Asserted upon address decoding miss error.
1	Add_Dec_Multi_Err	RW0C 0x0	Address Decoding Error Asserted upon address decoding multiple hit error.
31:2	Reserved	RSVD 0x0	Reserved

**Table 728: Address Decoder Error Mask Register**  
**Offset: 0x90130**

Bit	Field	Type/InitVal	Description
1:0	Various	RW 0x0	Mask bits for Address Decoder Error register. 0 = Mask 1 = Do not mask
31:2	Reserved	RSVD 0x0	Reserved

## A.18 Transport Stream (TS) Registers

The following table provides a summarized list of all of the Transport Stream (TS) registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 729: Register Map Table for the Transport Stream (TS) Registers**

Register Name	Offset	Table and Page
<b>TS-to-Mbus Bridge</b>		
TSU Modes Register	0xB4000	<a href="#">Table 730, p. 744</a>
TSU-Mbus Configuration Register	0xB4010	<a href="#">Table 731, p. 745</a>
Window0 Control Register	0xB4030	<a href="#">Table 732, p. 745</a>
Window0 Base Register	0xB4034	<a href="#">Table 733, p. 746</a>
Window1 Control Register	0xB4040	<a href="#">Table 734, p. 746</a>
Window1 Base Register	0xB4044	<a href="#">Table 735, p. 746</a>
Window2 Control Register	0xB4050	<a href="#">Table 736, p. 747</a>
Window2 Base Register	0xB4054	<a href="#">Table 737, p. 747</a>
Window3 Control Register	0xB4060	<a href="#">Table 738, p. 747</a>
Window3 Base Register	0xB4064	<a href="#">Table 739, p. 748</a>

### A.18.1 TS-to-Mbus Bridge

**Table 730: TSU Modes Register**

Offset: 0xB4000

Bit	Field	Type/InitVal	Description
0	Reserved	RW 0x1	Reserved. Must write 0x1.
13:1	Reserved	RSVD 0x0	Reserved
14	TS_Parallel_Mode	RW 0x0	Controls the connections of the TSU IOs to the chip MPPs. <b>NOTE:</b> The corresponding TSU register must also be configured to determine its operation in either serial or parallel mode. 0 = Serial_TSU_IO: All the serial TSU channel0 and channel1 I/Os are connected to the MPPs. 1 = Parallel_TSU_CH0_IO: All the Parallel TSU channel0 I/Os are connected to the MPPs. Channel1 I/Os are disconnected from the MPPs.
16:15	TSCK88	RW 0x0	TSCK88 Frequency Configuration  Note: Since parallel mode uses 8 data bits, TCK88 in this mode must be further divided (normally by 8), using the <Output Clock Frequency> field in the TS Interface Configuration Register. 0 = 83.333 MHz: 1GHz/12 1 = 76.923 MHz: 1GHz/13 2 = 90.909 MHz: 1GHz/11 3 = 100 MHz: 1GHz/10

**Table 730: TSU Modes Register (Continued)**  
Offset: 0xB4000

Bit	Field	Type/InitVal	Description
31:17	Reserved	RW 0x0	Reserved

**Table 731: TSU-Mbus Configuration Register**  
Offset: 0xB4010

Bit	Field	Type/InitVal	Description
7:0	Timeout	RW 0xFF	Mbus Arbiter Timeout Preset Value
15:8	RSVD1	RSVD 0x0	Reserved
16	TimeoutEn	RW 0x1	Mbus Arbiter Timer Enable
31:17	RSVD0	RSVD 0x0	Reserved

**Table 732: Window0 Control Register**  
Offset: 0xB4030

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window 0 Enable 0 = Disabled: Window is disabled. 1 = Enabled: Window is enabled.
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the target interface associated with this window. See "Default Address Map"
15:8	Attr	RW 0x0E	Specifies the target interface attributes associated with this window. See "Default Address Map"
31:16	Size	RW 0xFFFF	Window Size Used with the Base register to set the address window size and location. Must be programmed from LSB to MSB as sequence of 1's followed by sequence of 0's. The number of 1's specifies the size of the window in 64 KByte granularity (e.g., a value of 0x00FF specifies 256 = 16 MByte). <b>NOTE:</b> A value of 0x0 specifies 64-KByte size.



**Table 733: Window0 Base Register**

Offset: 0xB4034

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x0000	Base Address Used with the <Size> field to set the address window size and location. Corresponds to transaction address[31:16].

**Table 734: Window1 Control Register**

Offset: 0xB4040

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window1 Enable. See "Window0 Control Register".
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See "Window0 Control Register".
15:8	Attr	RW 0x0D	Target specific attributes depending on the target interface. See the "Window0 Control Register".
31:16	Size	RW 0x0FFF	Window Size See "Window0 Control Register".

**Table 735: Window1 Base Register**

Offset: 0xB4044

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x1000	Base Address See "Window0 Base Register".

**Table 736: Window2 Control Register**

Offset: 0xB4050

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window2 Enable See "Window0 Control Register".
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See "Window0 Control Register".
15:8	Attr	RW 0x0B	Target specific attributes depending on the target interface. See "Window0 Control Register".
31:16	Size	RW 0x0FFF	Window Size See "Window0 Control Register".

**Table 737: Window2 Base Register**

Offset: 0xB4054

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x2000	Base Address See "Window0 Base Register".

**Table 738: Window3 Control Register**

Offset: 0xB4060

Bit	Field	Type/InitVal	Description
0	WinEn	RW 0x1	Window3 Enable See "Window0 Control Register".
3:1	Reserved	RSVD 0x0	Reserved
7:4	Target	RW 0x0	Specifies the unit ID (target interface) associated with this window. See "Window0 Control Register".
15:8	Attr	RW 0x07	Target specific attributes depending on the target interface. See "Window0 Control Register".
31:16	Size	RW 0x0FFF	Window Size See "Window0 Control Register".



**Table 739: Window3 Base Register**

Offset: 0xB4064

Bit	Field	Type/InitVal	Description
15:0	Reserved	RSVD 0x0	Reserved
31:16	Base	RW 0x3000	Base Address See "Window0 Base Register".

## A.18.2 Transport Stream Interface General Registers

**Table 740: Register Map for TSU Registers**

Register Name	Offset	Table and Page
TS Interface Configuration Register	Port0: 0xB8000 Port1: 0xB8800	<a href="#">Table 741, p. 750</a>
TS DMA Parameter Register	Port0: 0xB8004 Port1: 0xB8804	<a href="#">Table 742, p. 752</a>
Done Queue Base Register	Port0: 0xB8008 Port1: 0xB8808	<a href="#">Table 743, p. 752</a>
Descriptor Queue Base Register	Port0: 0xB800C Port1: 0xB880C	<a href="#">Table 744, p. 752</a>
Done Queue Write Pointer Register	Port0: 0xB8010 Port1: 0xB8810	<a href="#">Table 745, p. 753</a>
Done Queue Read Pointer Register	Port0: 0xB8014 Port1: 0xB8814	<a href="#">Table 746, p. 753</a>
Descriptor Queue Write Pointer Register	Port0: 0xB8018 Port1: 0xB8818	<a href="#">Table 747, p. 754</a>
Descriptor Queue Read Pointer Register	Port0: 0xB801C Port1: 0xB881C	<a href="#">Table 748, p. 754</a>
Current Descriptor Register	Port0: 0xB8020 Port1: 0xB8820	<a href="#">Table 749, p. 754</a>
Current DMA Address Register	Port0: 0xB8024 Port1: 0xB8824	<a href="#">Table 750, p. 755</a>
Current DMA Length Register	Port0: 0xB8028 Port1: 0xB8828	<a href="#">Table 751, p. 755</a>
TSU Enable Access Register	Port0: 0xB802C Port1: 0xB882C	<a href="#">Table 752, p. 755</a>
TSU Timestamp Register	Port0: 0xB8030 Port1: 0xB8830	<a href="#">Table 753, p. 756</a>
TSU Status Register	Port0: 0xB8034 Port1: 0xB8834	<a href="#">Table 754, p. 756</a>
TSU Timestamp Control Register	Port0: 0xB8038 Port1: 0xB8838	<a href="#">Table 755, p. 757</a>
TSU Test Register	Port0: 0xB803C Port1: 0xB883C	<a href="#">Table 756, p. 757</a>
TSU Interrupt Source Register	Port0: 0xB8040 Port1: 0xB8840	<a href="#">Table 757, p. 758</a>
TSU Interrupt Mask Register	Port0: 0xB8044 Port1: 0xB8844	<a href="#">Table 758, p. 758</a>
IRQ Parameter Register	Port0: 0xB8048 Port1: 0xB8848	<a href="#">Table 759, p. 759</a>
TSU Next Descriptor1 Register	Port0: 0xB8050 Port1: 0xB8850	<a href="#">Table 760, p. 759</a>
TSU Next Descriptor2 Register	Port0: 0xB8054 Port1: 0xB8854	<a href="#">Table 761, p. 759</a>
TSU SyncByte Detection Register	Port0: 0xB8058 Port1: 0xB8858	<a href="#">Table 762, p. 760</a>
TSU Revision Register	Port0: 0xB805C Port1: 0xB885C	<a href="#">Table 763, p. 760</a>
TSU Aggregation Control Register	Port0: 0xB8060 Port1: 0xB8860	<a href="#">Table 764, p. 760</a>
TSU Timestamp Interval Register	Port0: 0xB8064 Port1: 0xB8864	<a href="#">Table 765, p. 761</a>

### A.18.2.1 TS Interface Configuration Register

TSU Interface Configuration Register controls the protocol behavior.

**Table 741: TS Interface Configuration Register**

Offset: Port0: 0xB8000 Port1: 0xB8800

Bits	Field (Short)	Type/InitVal	Description
31:24	TS Packet Size	RW 0xBB	Defines the size of the TS Packets to send or receive in bytes (0 = 1 byte,.., 0xBB = 188 bytes,.., 0xFF = 256 bytes).
23	TS_SYNC Used	RW 0x1	Defines if TS_SYNC is used for data transmission: 0x0 = Not used 0x1 = Used
22	TS_SYNC Polarity	RW 0x1	Defines the TS_SYNC signal polarity: 1'b0 = Low active 1'b1 = High active
21	TS_VAL Used	RW 0x1	Defines if TS_VAL is used for data transmission: 0x0 = Not used 0x1 = Used
20	TS_VAL Polarity	RW 0x1	Defines the TS_VAL signal polarity: 1'b0 = Low active 1'b1 = High active
19	TS_ERR Used	RW 0x0	Defines if TS_ERR is used for data transmission: 0x0 = Not used 0x1 = Used
18	TS_ERR Polarity	RW 0x1	Defines the TS_ERR signal polarity: 1'b0 = Low active 1'b1 = High active
17	Reserved	RW 0x0	Always write 0x1 to this bit.
16	TS Output Clock Frequency Mode	RW 0x1	Defines the TS Output Clock Frequency Mode: 1'b0 = Low frequency mode 1'b1 = High frequency mode
15	TS Data Transmission Edge	RW 0x0	Defines the edge with which data is sampled/transmitted: 1'b0 = Rising edge 1'b1 = Falling edge
14	TS Serial Data Order	RW 0x0	Defines the bit direction within a byte: 0x0 = MSB first 0x1 = LSB first
13	Serial TS_SYNC Active	RW 0x0	Defines the behavior of TS_SYNC in serial mode: 0x0 = Active for 8 bit-times (complete first byte). 0x1 = Active for 1 bit-time (first bit only).
12	TS Serial Clock Mode	RW 0x0	Defines the clock behavior in serial mode: 0x0 = Continuous mode 0x1 = Gapped mode

**Table 741: TS Interface Configuration Register (Continued)**  
Offset: Port0: 0xB8000 Port1: 0xB8800

<b>Bits</b>	<b>Field (Short)</b>	<b>Type/InitVal</b>	<b>Description</b>
11:10	Output Clock Frequency	RW 0x0	In conjunction with the Output Frequency Mode, this field defines the frequency of the output clock: <TS Output Clock Frequency Mode> = High 1b'00 = Source Clock/4 1b'01 = Source Clock/2 1b'10 = Source Clock 1b'11 = External clock  <TS Output Clock Frequency Mode> = Low 1b'00 = Source Clock/32 1b'01 = Source Clock/16 1b'10 = Source Clock/8 1b'11 = External clock
9	TS Data Mode	RW 0x0	Defines the data transmission mode: 0x0 = Serial 0x1 = Parallel
8	TS Data Direction	RW 0x0	Defines the mode in which the TSU is operating: 0x0 = Input mode 0x1 = Output mode
7	Aggressive Read on	EXEC_RB 0x1	Aggressive Read Timing for DMA On/Off.
6	Aggressive Read off		ON is executed, if bits [7:6] are set to 0x2. OFF is executed, if bits [7:6] are set to 0x1.
5	IRQ Clear on write	EXEC_RB 0x1	Mode for Interrupt bit in IRQ Register. Executed, if appropriate bit is set to 1.
4	IRQ Clear on read		0x1 = Clear on read 0x2 = Clear on write
3	Operational Mode on	EXEC_RB 0x1	Operational Mode on/off. ON is executed, if bits [3:2] are set to 0x2. OFF is executed, if bits [3:2] are set to 0x1. If Operational Mode is set to off, the TS Unit Registers can be configured, but the TS Unit state machines are held in their IDLE state. Reset by Hardware Reset.
2	Operational Mode off		
1	Reset Clear	EXEC_RB 0x1	Set/Clear Software Reset. Executed, if appropriate bit is set to 1.
0	Reset Set		If Software Reset is set, the TS Unit is held in RESET state. Reset by Hardware Reset.

### A.18.2.2 TS DMA Parameter Register

The TS packets have a fixed size, which can be configured in this register. The TS interface and the DMA Engine use this value.

**Table 742: TS DMA Parameter Register**

Offset: Port0: 0xB8004 Port1: 0xB8804

Bits	Field (Short)	Type/InitVal	Description
31:28	TS Done Queue Size	RW 0x6	Defines the Number of Dwords in the TSU Done Queue. If set to n the Queue can hold $2^{**n}$ Dwords. The maximum value is 10.
27:24	TS Descriptor Queue Size	RW 0x6	Defines the Number of Dwords in the TSU Descriptor Queue. If set to n the Queue can hold $2^{**n}$ Dwords. The maximum value is 10.
23:16	TS Data Watermark	RW 0x20	TS to memory direction: Filling Level of Data FIFO, in bytes, to start a TS Data DMA Memory to TS direction: Number of free bytes in Data FIFO, to start a TS Data DMA. <b>NOTE:</b> Always write 0x20 (32-byte watermark), for both receive and transmit. The TS-to-Mbus bridge maximum burst is 32 bytes, and it cannot cross the 32-byte boundary.
15:0	TS DMA Length	RW 0x00BC	TS DMA Length in bytes (0 = 0 bytes, 0xBC = 188 bytes, 0x100 = 256 bytes). <b>NOTE:</b> The TS DMA length is the same as the packet size.

### A.18.2.3 Done Queue Base Register

The TSU Done Queue has a programmable size of up to 1K Dwords.

**Table 743: Done Queue Base Register**

Offset: Port0: 0xB8008 Port1: 0xB8808

Bits	Field (Short)	Type/InitVal	Description
31:2	Done Queue Pointer Base	RW 0x0	Base address of TSU Done Queue
1:0	Reserved	0x0	Reserved

### A.18.2.4 Descriptor Queue Base Register

The TSU Descriptor Queue has a programmable size of up to 1K Dwords.

**Table 744: Descriptor Queue Base Register**

Offset: Port0: 0xB800C Port1: 0xB880C

Bits	Field (Short)	Type/InitVal	Description
31:2	Descriptor Queue Pointer Base	RW 0x0	Base address of TSU Descriptor Queue
1:0	Reserved	0x0	Reserved

### A.18.2.5 Done Queue Write Pointer Register

The bit positions in the table below apply to the maximum queue size setting of 10 (1K Dwords) only. If a smaller queue size is configured, the size of the Queue Pointer bit field is reduced by the appropriate number of bits, and the Rollover Bit moves down by the same number. For example, for a queue size of six (64 Dwords), the Queue Pointer field is bits [7:0], and the Rollover Bit is bit [8].

**Table 745: Done Queue Write Pointer Register**

Offset: Port0: 0xB8010 Port1: 0xB8810

Bits	Field (Short)	Type/InitVal	Description
31:13	Reserved		Reserved
12	Rollover Bit	RW 0x0	Rollover Bit
11:0	Done Queue Write Pointer	RW 0x0	Done Queue Write Pointer Stores the location of the Done queue where the next pointer is going to be written.

### A.18.2.6 Done Queue Read Pointer Register

The bit positions in the table below apply to the maximum queue size setting of 10 (1K Dwords) only. If a smaller queue size is configured, the size of the Queue Pointer bit field is reduced by the appropriate number of bits, and the Rollover Bit moves down by the same number. For example, for a queue size of six (64 Dwords), the Queue Pointer field is bits [7:0], and the Rollover Bit is bit [8].

**Table 746: Done Queue Read Pointer Register**

Offset: Port0: 0xB8014 Port1: 0xB8814

Bits	Field (Short)	Type/InitVal	Description
31:13	Reserved		Reserved
12	Rollover Bit	RW 0x0	Rollover Bit
11:0	Done Queue Read Pointer	RW 0x0	Done Queue Read Pointer Stores the location of the Done queue where the next pointer is going to be read.

### A.18.2.7 Descriptor Queue Write Pointer Register

The bit positions in the table below apply to the maximum queue size setting of 10 (1K Dwords) only. If a smaller queue size is configured, the size of the Queue Pointer bit field is reduced by the appropriate number of bits, and the Rollover Bit moves down by the same number. For example, for a queue size of six (64 Dwords), the Queue Pointer field is bits [7:0], and the Rollover Bit is bit [8].

**Table 747: Descriptor Queue Write Pointer Register**  
Offset: Port0: 0xB8018 Port1: 0xB8818

Bits	Field (Short)	Type/InitVal	Description
31:13	Reserved		Reserved
12	Rollover Bit	RW 0x0	Rollover Bit
11:0	Descriptor Queue Write Pointer	RW 0x0	Descriptor Queue Write Pointer Stores the location of the Descriptor queue where the next pointer is going to be written.

### A.18.2.8 Descriptor Queue Read Pointer Register

The bit positions in the table below apply to the maximum queue size setting of 10 (1K Dwords) only. If a smaller queue size is configured, the size of the Queue Pointer bit field is reduced by the appropriate number of bits, and the Rollover Bit moves down by the same number. For example, for a queue size of six (64 Dwords) the Queue Pointer field is bits [7:0] and the Rollover Bit is bit [8].

**Table 748: Descriptor Queue Read Pointer Register**  
Offset: Port0: 0xB801C Port1: 0xB881C

Bits	Field (Short)	Type/InitVal	Description
31:13	Reserved		Reserved
12	Rollover Bit	RW 0x0	Rollover Bit
11:0	Descriptor Queue Read Pointer	RW 0x0	Descriptor Queue Read Pointer Stores the location of the Descriptor queue where the next pointer is going to be read.

### A.18.2.9 Current Descriptor Register

For debug purposes only.

**Table 749: Current Descriptor Register**  
Offset: Port0: 0xB8020 Port1: 0xB8820

Bits	Field (Short)	Type/InitVal	Description
31:0	Current Descriptor	TO 0x0	Current Descriptor Stores the location of the current buffer to be written to / read from.

### A.18.2.10 Current DMA Address Register

For debug purposes only.

**Table 750: Current DMA Address Register**

Offset: Port0: 0xB8024 Port1: 0xB8824

Bits	Field (Short)	Type/InitVal	Description
31:0	Current DMA Address	TO 0x0	Current DMA Address Stores the next location within the current buffer to be written to / read from.

### A.18.2.11 Current DMA Length Register

For debug purposes only.

**Table 751: Current DMA Length Register**

Offset: Port0: 0xB8028 Port1: 0xB8828

Bits	Field (Short)	Type/InitVal	Description
31:16	Reserved	0x0	Reserved
15:0	Current DMA Length	TO 0x0	Current DMA Length Remaining bytes to transfer for current DMA request.

### A.18.2.12 TSU Enable Access Register

**Table 752: TSU Enable Access Register**

Offset: Port0: 0xB802C Port1: 0xB882C

Bits	Field (Short)	Type/InitVal	Description
31:24	Enable Access Descriptor Read	RW 0x04	Enables Access to Mbus Agents—Descriptor Read Set all bits to 1 to enable this field.
23:16	Enable Access Descriptor Write	RW 0x04	Enables Access to Mbus Agents—Descriptor Write Set all bits to 1 to enable this field.
15:8	Enable Access TS Write	RW 0x04	Enables Access to Mbus Agents—TS Write Set all bits to 1 to enable this field.
7:0	Enable Access TS Read	RW 0x04	Enables Access to Mbus Agents—TS Read Set all bits to 1 to enable this field.



### A.18.2.13 TSU Timestamp Register

Auto clear on read.

Table 753: TSU Timestamp Register

Offset: Port0: 0xB8030 Port1: 0xB8830

Bits	Field (Short)	Type/InitVal	Description
31	Valid	RO_STAT 0x0	Timestamp valid flag: Set on Read Timer command; cleared on a read of the Timestamp Register. 0x0 = Timestamp not valid 0x1 = Timestamp valid
30:28	Reserved	0x0	Reserved
27:0	Timestamp	TO 0x0	Timestamp Value

### A.18.2.14 TSU Status Register

Table 754: TSU Status Register

Offset: Port0: 0xB8034 Port1: 0xB8834

Bits	Field (Short)	Type/InitVal	Description
31:11	Reserved	0x0	Reserved
10	TS Connection Error	TO 0x0	TS Connection Error signalled by the TS interface. Valid only in the TS-to-Mbus direction
9	TS FIFO Overflow Error	TO 0x0	FIFO Overflow Error signalled by the TS interface. Valid only in the TS-to-Mbus direction
8	TS IF Error	TO 0x0	TS Error signalled by the TS interface. Valid only in the TS-to-Mbus direction
7:0	Packet Length	TO 0x0	Packet Length Valid only in the TS-to-Mbus direction

### A.18.2.15 TSU Timestamp Control Register

**Table 755: TSU Timestamp Control Register**

Offset: Port0: 0xB8038 Port1: 0xB8838

Bits	Field (Short)	Type/InitVal	Description
31:5	Reserved	0x0	Reserved
4	Read Timer	EXEC_HP 0x0	When this command is set, the Timestamp Timer is sampled and the value can be read from the Timestamp register.
3	Automatic Adjust On	EXEC_RB 0x1	Automatic Adjust feature for TS Output Mode: 0b01 = Timestamp Timer can only be reloaded by firmware.
2	Automatic Adjust Off		0b10 = If Timestamp Value in Descriptor is lower than current value of the Timestamp Timer (packet is late), then the Timestamp Timer reloads with the value from the descriptor.
1	Enable Timer	EXEC_RB 0x1	Disable/Enable Timestamp Timer. If disabled, the Timestamp Timer holds its value.
0	Disable Timer		If enabled, the Timestamp Timer starts counting beginning at the current value.

### A.18.2.16 TSU Test Register

For test purposes only.

**Table 756: TSU Test Register**

Offset: Port0: 0xB803C Port1: 0xB883C

Bits	Field (Short)	Type/InitVal	Description
31:2	Reserved	0x0	Reserved
1	Enable Loop TS1 to TS0	RW 0x0	Enables loopback from the TS1 interface signals to the TS0 interface signals. <b>NOTE:</b> This is only possible if the TS Test Register of both TSU ports is set to TSLOOP2!
0	Enable Loop TS0 to TS1	RW 0x0	Enables loopback from the TS0 interface signals to the TS1 interface signals. <b>NOTE:</b> This is only possible if the TS Test Register of both TSU ports is set to TSLOOP1!

### A.18.2.17 TSU Interrupt Source Register

Auto clear on read.

**Table 757: TSU Interrupt Source Register**

Offset: Port0: 0xB8040 Port1: 0xB8840

Bits	Field (Short)	Type/InitVal	Description
31:7	Reserved	0x0	Reserved
6	Interrupt on CLK Sync Timer	ROC	An interrupt is asserted after the Clock Synchronization Timer expires. This bit is auto cleared on read.
5	Interrupt on TS Connection Error	ROC	An interrupt is asserted if a TS Connection Error has occurred on a packet. This bit is auto cleared on read.
4	Interrupt on FIFO Overflow Error	ROC	An interrupt is asserted if a FIFO Overflow Error has occurred on a packet. This bit is auto cleared on read.
3	Interrupt on TS Interface Error	ROC	An interrupt is asserted if TS_ERR signal has been asserted for a packet. This bit is auto cleared on read.
2	Interrupt Done Aggregated Packet	ROC	An interrupt is asserted after the last Status/Timestamp of the aggregated packet has been placed in the TS Done Queue. This bit is auto cleared on read.
1	Interrupt Done Put Descriptor	ROC	An interrupt is asserted after the Descriptor has been place in the TS Done Queue. This bit is auto cleared on read.
0	Interrupt Descr Threshold	ROC	An interrupt is asserted when the difference of the Descriptor RDPT and WRPT is as defined in DESC_THR. This bit is auto cleared on read.

### A.18.2.18 TSU Interrupt Mask Register

**Table 758: TSU Interrupt Mask Register**

Offset: Port0: 0xB8044 Port1: 0xB8844

Bits	Field (Short)	Type/InitVal	Description
31:7	Reserved	0x0	Reserved
6	Mask CLK Sync Timer	RW 0x0	Mask Interrupt on Clock Synchronization Timer
5	Mask TS Connection Error	RW 0x0	Mask Interrupt on TS Connection Error
4	Mask FIFO Overflow Error	RW 0x0	Mask Interrupt on FIFO Overflow Error
3	Mask Packet Error	RW 0x0	Mask Interrupt on TS Interface Error
2	Mask Done Aggregated Packet	RW 0x0	Mask Interrupt on Done Aggregated Packet

**Table 758: TSU Interrupt Mask Register (Continued)**  
Offset: Port0: 0xB8044 Port1: 0xB8844

Bits	Field (Short)	Type/InitVal	Description
1	Mask Done Put Descriptor	RW 0x0	Mask Interrupt on Done Put Descriptor
0	Mask Descr Threshold	RW 0x0	Mask Interrupt on Descriptor Threshold

### A.18.2.19 IRQ Parameter Register

**Table 759: IRQ Parameter Register**  
Offset: Port0: 0xB8048 Port1: 0xB8848

Bits	Field (Short)	Type/InitVal	Description
31:20	Reserved	0x0	Reserved
19:8	Clock Synchronization Timer	RW 0x0	Defines the time (in milliseconds) to assert an interrupt to synchronize the clocks of two systems.
7:0	Descriptor IRQ Threshold	RW 0x0	Defines the number of outstanding descriptors where an interrupt should be asserted to prepare new descriptors.

### A.18.2.20 TSU Next Descriptor1 Register

**Table 760: TSU Next Descriptor1 Register**  
Offset: Port0: 0xB8050 Port1: 0xB8850

Bits	Field (Short)	Type/InitVal	Description
31:0	Next Descriptor	TO 0x0	The next descriptor that was fetched while the current descriptor is in use.

### A.18.2.21 TSU Next Descriptor2 Register

**Table 761: TSU Next Descriptor2 Register**  
Offset: Port0: 0xB8054 Port1: 0xB8854

Bits	Field (Short)	Type/InitVal	Description
31:29	Reserved	0x0	Reserved
28	TS_ERR	TO 0x0	Indicates that for this Packet the TS_ERR signal was asserted on the sender side.
27:0	Sync Value	TO 0x0	The synchronization value to be compared with the Timestamp Timer.

### A.18.2.22 TSU SyncByte Detection Register

**Table 762: TSU SyncByte Detection Register**

Offset: Port0: 0xB8058 Port1: 0xB8858

Bits	Field (Short)	Type/InitVal	Description
31:8	Reserved	0x0	Reserved
7:4	SyncByte Loss Count	RW 0x2	The number of synchronization byte checks that have to fail before a new synchronization process starts.
3:0	SyncByte Count	RW 0x2	The number of synchronization byte to be detected before transferring data into the FIFO.

### A.18.2.23 TSU Revision Register

**Table 763: TSU Revision Register**

Offset: Port0: 0xB805C Port1: 0xB885C

Bits	Field (Short)	Type/InitVal	Description
31:0	TSU Revision	RO 0x0	Fixed revision number of the TSU Read only fixed value.

### A.18.2.24 TSU Aggregation Control Register

**Table 764: TSU Aggregation Control Register**

Offset: Port0: 0xB8060 Port1: 0xB8860

Bits	Field (Short)	Type/InitVal	Description
31	Aggregation on	EXEC_RB 0x1	TSU Aggregation On/Off Executed, if appropriate bit is set to 1.  0x1 = Aggregation off 0x2 = Aggregation on
30	Aggregation off		
29	Timestamp to packet	EXEC_RB 0x1	Timestamp Mode for Aggregation Executed, if appropriate bit is set to 1.  0x1 = Write Timestamp to Done Queue 0x2 = Write Timestamp to packet buffer
28	Timestamp to Done Queue		
27	Flush Erroneous Packets on	EXEC_RB 0x1	Flush Mode for Aggregation Executed, if appropriate bit is set to 1.  0x1 = Keep erroneous packets 0x2 = Flush erroneous packets
26	Flush Erroneous Packets off		
25:12	Reserved	0x0	Reserved

**Table 764: TSU Aggregation Control Register (Continued)**  
Offset: Port0: 0xB8060 Port1: 0xB8860

Bits	Field (Short)	Type/InitVal	Description
11:8	Timestamp Offset	RW 0x4	Timestamp offset between packets (in bytes) When 0, Type 1 Aggregation. When non-zero, Type 2 Aggregation.
7:0	Aggregated Packets	RW 0x7	Number of TS packets to aggregate

### A.18.2.25 TSU Timestamp Interval Register

**Table 765: TSU Timestamp Interval Register**  
Offset: Port0: 0xB8064 Port1: 0xB8864

Bits	Field (Short)	Type/InitVal	Description
31:28	Reserved	0x0	Reserved
27:0	Timestamp Interval	TO 0x0	The timestamp interval is fetched from the packet buffer, before packet data is fetched. It determines the inter-packet gap between the TS packets of an aggregated packet in Type 1 Aggregation.

## A.19 General Purpose Port Registers

The following table provides a summarized list of all of the General Purpose Port registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 766: Register Map Table for the General Purpose Port Registers**

Register Name	Offset	Table and Page
GPIO Data Out Register	0x10100	<a href="#">Table 767, p. 762</a>
GPIO Data Out Enable Control Register	0x10104	<a href="#">Table 768, p. 762</a>
GPIO Blink Enable Register	0x10108	<a href="#">Table 769, p. 763</a>
GPIO Data In Polarity Register	0x1010C	<a href="#">Table 770, p. 763</a>
GPIO Data In Register	0x10110	<a href="#">Table 771, p. 763</a>
GPIO Interrupt Cause Register	0x10114	<a href="#">Table 772, p. 763</a>
GPIO Interrupt Mask Register	0x10118	<a href="#">Table 773, p. 764</a>
GPIO Interrupt Level Mask Register	0x1011C	<a href="#">Table 774, p. 764</a>
GPIO High Data Out Register	0x10140	<a href="#">Table 775, p. 764</a>
GPIO High Data Out Enable Control Register	0x10144	<a href="#">Table 776, p. 764</a>
GPIO High Blink Enable Register	0x10148	<a href="#">Table 777, p. 765</a>
GPIO High Data In Polarity Register	0x1014C	<a href="#">Table 778, p. 765</a>
GPIO High Data In Register	0x10150	<a href="#">Table 779, p. 765</a>
GPIO High Interrupt Cause Register	0x10154	<a href="#">Table 780, p. 765</a>
GPIO High Interrupt Mask Register	0x10158	<a href="#">Table 781, p. 766</a>
GPIO High Interrupt Level Mask Register	0x1015C	<a href="#">Table 782, p. 766</a>

**Table 767: GPIO Data Out Register**

Offset: 0x10100

Bit	Field	Type/InitVal	Description
31:0	GPIOOut	RW 0x0	GPIO Output Pins Value, bit per each GPIO pin

**Table 768: GPIO Data Out Enable Control Register**

Offset: 0x10104

Bit	Field	Type/InitVal	Description
31:0	GPIOOutEn	RW 0xFFFFFFFF	GPIO Port Output Enable This field is active low. Data is driven when the corresponding bit value is 0.

**Table 769: GPIO Blink Enable Register**  
**Offset: 0x10108**

Bit	Field	Type/InitVal	Description
31:0	GPIOBlink	RW 0x0	GPIO Data Blink When set and the corresponding bit in GPIO Data Out Enable Control Register is enabled, the GPIO pin blinks every ~100 ms (a period of 2^24 TCLK clocks).

**Table 770: GPIO Data In Polarity Register**  
**Offset: 0x1010C**

Bit	Field	Type/InitVal	Description
31:0	GPIODataInActLow	RW 0x0	GPIO Data in Active Low When set to 1, GPIO Data In Register reflects the inverted value of the corresponding pin.

**Table 771: GPIO Data In Register**  
**Offset: 0x10110**

Bit	Field	Type/InitVal	Description
31:0	GPIODIn	RO 0x0	Each bit in this field reflects the value of the corresponding GPIO pin. If corresponding bit in GPIO Data In Polarity Register is cleared to 0, the bit reflects the pin value with no change. If corresponding bit in GPIO Data In Polarity Register is set to 1, the bit reflects the pin inverted value.

**Table 772: GPIO Interrupt Cause Register**  
**Offset: 0x10114**

Bit	Field	Type/InitVal	Description
31:0	GPIOInt	RWOC 0x0	A bit in this field is set on the transition of the corresponding bit in the <GPIODIn> field, in the GPIO Data In Register, from 0 to 1.



**Table 773: GPIO Interrupt Mask Register**  
Offset: 0x10118

Bit	Field	Type/InitVal	Description
31:0	GPIOIntEdgeMask	RW 0x0	GPIO Interrupt Edge Sensitive Mask The mask bit for each cause bit in the <GPIOInt> field of the GPIO Interrupt Cause Register. The mask only affects the assertion of the interrupt bits in Main Interrupt Cause Register. It does not affect the setting of bits in the GPIO Interrupt Cause Register. 0=Interrupt mask; 1=Interrupt enable;

**Table 774: GPIO Interrupt Level Mask Register**  
Offset: 0x1011C

Bit	Field	Type/InitVal	Description
To set an edge sensitive interrupt, set the corresponding bit in the GPIO Interrupt Mask Register. To set a level sensitive interrupt, set the corresponding bit in the GPIO Interrupt Level Mask Register.			
31:0	GPIOIntLevelMask	RW 0x0	GPIO Interrupt Level Sensitive Mask The mask bit for each bit in the <GPIODataIn> field of the GPIO Data In Register. The mask only affects the assertion of the interrupt bit in Main Interrupt Cause Register. It does not affect the value of bits in the GPIO Data In Register. 0=Interrupt mask; 1=Interrupt enable;

**Table 775: GPIO High Data Out Register**  
Offset: 0x10140

Bit	Field	Type/InitVal	Description
31:0	GPIOHDOOut	RW 0x0	GPIO Output Pins Value, bit per each GPIO pin

**Table 776: GPIO High Data Out Enable Control Register**  
Offset: 0x10144

Bit	Field	Type/InitVal	Description
31:0	GPIOHDOOutEn	RW 0xFFFFFFFF	GPIO Port Output Enable This field is active low. Data is driven when the corresponding bit value is 0.

**Table 777: GPIO High Blink Enable Register**  
**Offset: 0x10148**

Bit	Field	Type/InitVal	Description
31:0	GPIOHDBlink	RW 0x0	GPIO Data Blink When set and the corresponding bit in GPIO Data Out Enable Control Register is enabled, the GPIO pin blinks every ~100 ms (a period of 2^24 TCLK clocks).

**Table 778: GPIO High Data In Polarity Register**  
**Offset: 0x1014C**

Bit	Field	Type/InitVal	Description
31:0	GPIOHDatainActLow	RW 0x0	GPIO Data in Active Low When set to 1, GPIO Data In Register reflects the inverted value of the corresponding pin.

**Table 779: GPIO High Data In Register**  
**Offset: 0x10150**

Bit	Field	Type/InitVal	Description
31:0	GPIODIn	RO 0x0	Each bit in this field reflects the value of the corresponding GPIO pin. If corresponding bit in GPIO Data In Polarity Register is cleared to 0, the bit reflects the pin value with no change. If corresponding bit in GPIO Data In Polarity Register is set to 1, the bit reflects the pin inverted value.

**Table 780: GPIO High Interrupt Cause Register**  
**Offset: 0x10154**

Bit	Field	Type/InitVal	Description
31:0	GPIOHInt	RWOC 0x0	A bit in this field is set on the transition of the corresponding bit in the <GPIODIn> field, in the GPIO Data In Register, from 0 to 1.

**Table 781: GPIO High Interrupt Mask Register**  
**Offset: 0x10158**

Bit	Field	Type/InitVal	Description
31:0	GPIOHIntEdgeMask	RW 0x0	<p>GPIO Interrupt Edge Sensitive Mask  The mask bit for each cause bit in the &lt;GPIOInt&gt; field of the GPIO Interrupt Cause Register.</p> <p>The mask only affects the assertion of the interrupt bits in Main Interrupt Cause Register. It does not affect the setting of bits in the GPIO Interrupt Cause Register.</p> <p>0=Interrupt mask;  1=Interrupt enable;</p>

**Table 782: GPIO High Interrupt Level Mask Register**  
**Offset: 0x1015C**

Bit	Field	Type/InitVal	Description
<p>To set an edge sensitive interrupt, set the corresponding bit in the GPIO Interrupt Mask Register.  To set a level sensitive interrupt, set the corresponding bit in the GPIO Interrupt Level Mask Register.</p>			
31:0	GPIOHIntLevelMask	RW 0x0	<p>GPIO Interrupt Level Sensitive Mask  The mask bit for each bit in the &lt;GPIODataIn&gt; field of the GPIO Data In Register.</p> <p>The mask only affects the assertion of the interrupt bit in Main Interrupt Cause Register. It does not affect the value of bits in the GPIO Data In Register.</p> <p>0=Interrupt mask;  1=Interrupt enable;</p>

## A.20 RTC Registers

The following table provides a summarized list of all of the RTC registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 783: Register Map Table for the RTC Registers**

Register Name	Offset	Table and Page
RTC Time Register	0x10300	<a href="#">Table 784, p. 767</a>
RTC Date Register	0x10304	<a href="#">Table 785, p. 768</a>
RTC Alarm Time Configuration Register	0x10308	<a href="#">Table 786, p. 768</a>
RTC Alarm Date Configuration Register	0x1030C	<a href="#">Table 787, p. 769</a>
RTC Interrupt Mask Register	0x10310	<a href="#">Table 788, p. 770</a>
RTC Interrupt Cause Register	0x10314	<a href="#">Table 789, p. 770</a>

**Table 784: RTC Time Register**

Offset: 0x10300

Bit	Field	Type/InitVal	Description
This register provides time information. Note that the values in this register are maintained between power up and power down (with consideration for the real-time passing)			
6:0	Second	RW 0x0	Second within the minute (0..59). Bits[6:4] set the decimal digit. Bits[3:0] set the unit digit.
7	Reserved	RSVD 0x0	Reserved
14:8	Minute	RW 0x0	Minute within the hour (0..59) Bits[14:12] set the decimal digit. Bits[11:8] set the unit digit.
15	Reserved	RSVD 0x0	Reserved
21:16	Hour	RW 0x0	Hour within the day (0..23 or 1..12) In 12-hour format: Bit[21] is an AM/PM indication, where: 1 = PM 0 = AM Bit[20] sets the decimal digit. Bits[19:16] set the unit digit.  In 24-hour format: Bits[19:16] sets the unit digit. Bits[21:20] set the decimal digit.
22	HourFormat	RW 0x0	12- or 24-hour format configuration: 0 = 24 1 = 12



**Table 784: RTC Time Register (Continued)**  
Offset: 0x10300

Bit	Field	Type/InitVal	Description
23	Reserved	RSVD 0x0	Reserved
26:24	WeekDay	RW 0x1	Day within the week (1..7)
31:27	Reserved	RSVD 0x0	Reserved

**Table 785: RTC Date Register**  
Offset: 0x10304

Bit	Field	Type/InitVal	Description
This register provides date information. Note that the values in this register are maintained between power up and power down (with consideration for the real-time passing)			
5:0	Day	RW 0x1	Day within the month (1..31)
7:6	Reserved	RSVD 0x0	Reserved
12:8	Month	RW 0x1	Month within the year (1..12)
15:13	Reserved	RSVD 0x0	Reserved
23:16	Year	RW 0x0	Year within the 21st century (0..99)
31:24	Reserved	RSVD 0x0	Reserved

**Table 786: RTC Alarm Time Configuration Register**  
Offset: 0x10308

Bit	Field	Type/InitVal	Description
6:0	AlarmSeconds	RW 0x0	When the <AlarmSecondValid> field is set to 1, an alarm interrupt will occur only when the value in this field matches the value in the <Second> field in the RTC Time register.
7	AlarmSecondValid	RW 0x0	When this field is set to 1, the <AlarmSecond> field must match the <Second> field in the RTC Time register for an alarm interrupt to occur.

**Table 786: RTC Alarm Time Configuration Register (Continued)**  
**Offset: 0x10308**

Bit	Field	Type/InitVal	Description
14:8	AlarmMinutes	RW 0x0	When the <AlarmMinueValid> field is set to 1, an alarm interrupt will occur only when the value in this field matches the value in <Minute> field in the RTC Time register.
15	AlarmMinuteValid	RW 0x0	When this field is set to 1, the <AlarmMinute> field must match <Minute> field in the RTC Time register for an alram interrupt to occur.
21:16	AlarmHour	RW 0x0	When the <AlarmHourValid> field is set to 1, an alarm interrupt will occur only when the value in this field matches the value in the <Hour> field in the RTC Time register.
22	Reserved	RSVD 0x0	Reserved
23	AlarmHourValid	RW 0x0	When this field is set to 1, the value in the <AlarmHour> field must match the value in the <Hour> field in the RTC Time register for an alram interrupt to occur.
31:24	Reserved	RSVD 0x0	Reserved

**Table 787: RTC Alarm Date Configuration Register**  
**Offset: 0x1030C**

Bit	Field	Type/InitVal	Description
5:0	AlarmDay	RW 0x1	When the <AlarmDayValid> field is set to 1, an alarm interrupt will occur only when the value in this field matches the value in the <Day> field in the RTC Date register.
6	Reserved	RSVD 0x0	Reserved
7	AlarmDayValid	RW 0x0	When this field is set to 1, the <AlarmDay> field must match the <Day> field in the RTC Date register for an alram interrupt to occur.
12:8	AlarmMonth	RW 0x1	When the <AlarmMonthValid> field is set to 1, an alarm interrupt will occur only when the value in this field matches the value in the <Month> field in the RTC Date register.
14:13	Reserved	RSVD 0x0	Reserved
15	AlarmMonthValid	RW 0x0	When this field is set to 1, the <AlarmMonth> field must match the <Month> field in the RTC Date register for an alram interrupt to occur.



**Table 787: RTC Alarm Date Configuration Register (Continued)**  
Offset: 0x1030C

Bit	Field	Type/InitVal	Description
23:16	AlarmYear	RW 0x0	When the <AlarmYearValid> field is set to 1, an alarm interrupt will occur only when the value in this field matches the value in the <Year> field in the RTC Date register.
24	AlarmYearValid	RW 0x0	When this field is set to 1, the <AlarmYear> field must match the <Year> field in the RTC Date register for an alarm interrupt to occur.
31:25	Reserved	RSVD 0x0	Reserved

**Table 788: RTC Interrupt Mask Register**  
Offset: 0x10310

Bit	Field	Type/InitVal	Description
0	AlarmInterruptEnable	RW 0x0	Mask bit for the alarm interrupt Mask only effects the assertion of interrupt pin. It does not effect the setting of bits in the RTC Interrupt Cause register. 0 = Masked: Interrupt masked. 1 = Enabled: Interrupt enabled.
31:1	Reserved	RSVD 0x0	Reserved

**Table 789: RTC Interrupt Cause Register**  
Offset: 0x10314

Bit	Field	Type/InitVal	Description
0	AlarmInterrupt	RWOC 0x0	Set when any of the alarm valid bits is active and all the alarm fields that have an alarm valid bit set match the corresponding real time field. Cleared when written 0.
31:1	Reserved	RSVD 0x0	Reserved

## A.21 Boot ROM Registers

The following table provides a summarized list of all of the Boot ROM registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 790: Register Map Table for the Boot ROM Registers**

Register Name	Offset	Table and Page
Boot ROM Routine and Error Code Register	0x100D0	<a href="#">Table 791, p. 771</a>

**Table 791: Boot ROM Routine and Error Code Register**

Offset: 0x100D0

Bit	Field	Type/InitVal	Description
7:0	Error Code	RW 0x0	Error Code: 0x00 = No Error 0x11 = Invalid Header ID 0x12 = Invalid Header Checksum 0x13 = Invalid Image Checksum 0x15 = Invalid Extended Header Checksum 0x17 = SATA Device is Busy 0x18 = SATA Device Link Error 0x19 = SATA DMA transfer Error 0x1A = SATA PIO transfer Error 0x1D = Unknown Boot Device 0x21 = Image Size not Alligned to 32bit 0x22 = Source Address not Alligned to 32bit 0x23 = Destination Address not Alligned to 32bit 0x30 = Failed to Read from NAND Flash 0x31 = Invalid Bootrom Checksum 0x32 = Nand image not aligned to 512 bytes 0x33 = Nand ECC error 0x34 = Nand Timeout. 0x35 = Nand excessive bad blocks 0x36 = Nand bad block encountered
11:8	Error Location	RW 0x0	Error Location 0 = Initialization 4 = SATA: Boot from SATA interface 7 = SPI: Boot from SPI interface 9 = NAND_flash: Boot from NAND flash interface 11 = Exception: Exception Handler 12 = Main: Main execution loop 14 = PCI_Express: Boot from PCI Express interface
12	Reserved	RW 0x0	Reserved Write only 0x0.



**Table 791: Boot ROM Routine and Error Code Register (Continued)**  
Offset: 0x100D0

Bit	Field	Type/InitVal	Description
14:13	Reserved	RW 0x0	Reserved Write only 0x0.
15	Reserved	RW 0x0	Reserved Write only 0x0.
23:16	Retry_Count	RW 0xFF	The retry count is incremented in the Error Handler routine.
27:24	Reserved	RW 0xf	Reserved Write only 0xF.
31:28	Reserved	RW 0xf	Reserved Write only 0xF.

## A.22 MPP Registers

The following table provides a summarized list of all of the MPP registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 792: Register Map Table for the MPP Registers**

Register Name	Offset	Table and Page
MPP Control 0 Register	0x10000	<a href="#">Table 793, p. 773</a>
MPP Control 1 Register	0x10004	<a href="#">Table 794, p. 774</a>
MPP Control 2 Register	0x10008	<a href="#">Table 795, p. 774</a>
MPP Control 3 Register	0x1000C	<a href="#">Table 796, p. 775</a>
MPP Control 4 Register	0x10010	<a href="#">Table 797, p. 776</a>
MPP Control 5 Register	0x10014	<a href="#">Table 798, p. 776</a>
MPP Control 6 Register	0x10018	<a href="#">Table 799, p. 777</a>
Sample at Reset Register	0x10030	<a href="#">Table 800, p. 778</a>

**Table 793: MPP Control 0 Register**

Offset: 0x10000

Bit	Field	Type/InitVal	Description
3:0	MPPSel0	RW SAR	MPP0 Select See the MPP Function Summary table in Hardware Specifications for this device.
7:4	MPPSel1	RW SAR	MPP1 Select See field MPPSel0.
11:8	MPPSel2	RW SAR	MPP2 Select See field MPPSel0.
15:12	MPPSel3	RW SAR	MPP3 Select See field MPPSel0.
19:16	MPPSel4	RW SAR	MPP4 Select See field MPPSel0.
23:20	MPPSel5	RW SAR	MPP5 Select See field MPPSel0.
27:24	MPPSel6	RW 0x1	MPP6 Select See field MPPSel0.
31:28	MPPSel7	RW 0x1	MPP7 Select See field MPPSel0.



**Table 794: MPP Control 1 Register**

Offset: 0x10004

Bit	Field	Type/InitVal	Description
3:0	MPPSel8	RW SAR	MPP8 Select See the MPP Function Summary table in the Hardware Specifications for this device.
7:4	MPPSel9	RW SAR	MPP9 Select See field MPPSel8.
11:8	MPPSel10	RW 0x0	MPP10 Select See field MPPSel8.
15:12	MPPSel11	RW 0x0	MPP11 Select See field MPPSel8.
19:16	MPPSel12	RW 0x0	MPP12 Select See field MPPSel8.
23:20	MPPSel13	RW 0x0	MPP13 Select See field MPPSel8.
27:24	MPPSel14	RW 0x0	MPP14 Select See field MPPSel8.
31:28	MPPSel15	RW 0x0	MPP15 Select See field MPPSel8.

**Table 795: MPP Control 2 Register**

Offset: 0x10008

Bit	Field	Type/InitVal	Description
3:0	MPPSel16	RW 0x0	MPP16 Select See the MPP Function Summary table in the Hardware Specifications for this device.
7:4	MPPSel17	RW 0x0	MPP17 Select See field MPPSel16.
11:8	MPPSel18	RW SAR	MPP18 Select See field MPPSel16.
15:12	MPPSel19	RW SAR	MPP19 Select See field MPPSel16.

**Table 795: MPP Control 2 Register (Continued)**  
Offset: 0x10008

Bit	Field	Type/InitVal	Description
19:16	MPPSel20	RW 0x0	MPP20 Select See field MPPSel16.
23:20	MPPSel21	RW 0x0	MPP21 Select See field MPPSel16.
27:24	MPPSel22	RW 0x0	MPP22 Select See field MPPSel16.
31:28	MPPSel23	RW 0x0	MPP23 Select See field MPPSel16.

**Table 796: MPP Control 3 Register**  
Offset: 0x1000C

Bit	Field	Type/InitVal	Description
3:0	MPPSel24	RW 0x0	MPP24 Select See the MPP Function Summary table in the Hardware Specifications for this device.
7:4	MPPSel25	RW 0x0	MPP25 Select See field MPPSel24.
11:8	MPPSel26	RW 0x0	MPP26 Select See field MPPSel24.
15:12	MPPSel27	RW 0x0	MPP27 Select See field MPPSel24.
19:16	MPPSel28	RW 0x0	MPP28 Select See field MPPSel24.
23:20	MPPSel29	RW 0x0	MPP29 Select See field MPPSel24.
27:24	MPPSel30	RW 0x0	MPP30 Select See field MPPSel24.
31:28	MPPSel31	RW 0x0	MPP31 Select See field MPPSel24.



**Table 797: MPP Control 4 Register**  
**Offset: 0x10010**

Bit	Field	Type/InitVal	Description
3:0	MPPSel32	RW 0x0	MPP32 Select See the MPP Function Summary table in the Hardware Specifications for this device.
7:4	MPPSel33	RW 0x0	MPP33 Select See field MPPSel32.
11:8	MPPSel34	RW 0x0	MPP34 Select See field MPPSel32.
15:12	MPPSel35	RW 0x0	MPP35 Select See field MPPSel32.
19:16	MPPSel36	RW 0x0	MPP36 Select See field MPPSel32.
23:20	MPPSel37	RW 0x0	MPP37 Select See field MPPSel32.
27:24	MPPSel38	RW 0x0	MPP38 Select See field MPPSel32.
31:28	MPPSel39	RW 0x0	MPP39 Select See field MPPSel32.

**Table 798: MPP Control 5 Register**  
**Offset: 0x10014**

Bit	Field	Type/InitVal	Description
3:0	MPPSel40	RW 0x0	MPP40 Select See the MPP Function Summary table in the Hardware Specifications for this device.
7:4	MPPSel41	RW 0x0	MPP41 Select See field MPPSel40.
11:8	MPPSel42	RW 0x0	MPP42 Select See field MPPSel40.
15:12	MPPSel43	RW 0x0	MPP43 Select See field MPPSel40.

**Table 798: MPP Control 5 Register (Continued)**  
Offset: 0x10014

Bit	Field	Type/InitVal	Description
19:16	MPPSel44	RW 0x0	MPP44 Select See field MPPSel40.
23:20	MPPSel45	RW 0x0	MPP45 Select See field MPPSel40.
27:24	MPPSel46	RW 0x0	MPP46 Select See field MPPSel40.
31:28	MPPSel47	RW 0x0	MPP47 Select See field MPPSel40.

**Table 799: MPP Control 6 Register**  
Offset: 0x10018

Bit	Field	Type/InitVal	Description
3:0	MPPSel48	RW 0x0	MPP48 Select See the MPP Function Summary table in the Hardware Specifications for this device.
7:4	MPPSel49	RW 0x0	MPP49 Select See field MPPSel48.
31:8	Reserved	RSVD 0x0	Reserved

**Table 800: Sample at Reset Register**  
**Offset: 0x10030**

Bit	Field	Type/InitVal	Description
31:0	SampleAtReset	RW SAR	<p><b>NOTE:</b> Initial value is sampled at reset. See the device Hardware Specifications for more details.</p> <p><b>88F6180:</b> Bit[0] = TWSI Serial ROM initialization. Bits[4:2] = CPU/DDR/L2 cache clocks select. Bit[11:5] = Used for internal testing. {Bit[1], Bits[13:12]} = Boot device. Bit[14] = PCI Express clock configuration. Bit[15] = SSCG disable. Bits[31:16] = Used for internal testing.</p> <p><b>88F6190 and 88F6192:</b> Bit[0] = TWSI Serial ROM initialization. Bits[14:12] = Boot device. Bit[15] = SSCG disable. Bit[16] = PCI Express clock configuration.</p> <p>The following bits are for internal testing: {Bits[4:3], Bit[22], Bits[1]} = Should be 0x4 for the 88F6190. Should be 0x6 for the 88F6192. Bits[8:5] = Should be 0x4 for the 88F6190. Should be 0x6 for the 88F6192. {Bit[19], Bits[10:9]} = Should be 0x1. Bit[2] = Should be 0x1. Bit[21] = Should be 0x1. Bit[18] = Should be 0x0. Bits[31:23,20:19,17,11]: Reserved</p> <p><b>88F6281:</b> Bit[0] = TWSI Serial ROM initialization. {Bits[4:3], Bit[22], Bits[1]} = CPU CLK frequency select. Bits[8:5] = CPU CLK to DDR CLK (HCLK) ratio. {Bit[19], Bits[10:9]} = CPU CLK to CPU L2 clock ratio. Bits[14:12] = Boot device. Bit[15] = SSCG disable. Bit[16] = PCI Express clock configuration.</p> <p>The following bits are for internal testing: Bit[2] = Should be 0x1. Bit[21] = Should be 0x0. Bit[18] = Should be 0x0. Bits[31:23,20:19,17,11]]: Reserved</p>

## A.23 eFuse Registers

The following table provides a summarized list of all of the eFuse registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 801: Register Map Table for the eFuse Registers**

Register Name	Offset	Table and Page
eFuse Protection Register	0x1008C	<a href="#">Table 802, p. 779</a>
eFuse0 Low Register	0x100A4	<a href="#">Table 803, p. 780</a>
eFuse0 High Register	0x100A8	<a href="#">Table 804, p. 780</a>
eFuse1 Low Register	0x100AC	<a href="#">Table 805, p. 780</a>
eFuse1 High Register	0x100B0	<a href="#">Table 806, p. 781</a>
eFuse Control Register	0x100B4	<a href="#">Table 807, p. 781</a>

**Table 802: eFuse Protection Register**

Offset: 0x1008C

Bit	Field	Type/InitVal	Description
0	FSB0	RW 0x0	eFuse0 Security Bit. This bit is not writable when BurnMode = 0. When BurnMode = 1, software may change the field value from 0 to 1. A 0x1 in this field followed by a <eFuse0_Write_Trigger> programs the FSB of eFuse_0 to 0x1. After this scenario, a write cannot be performed to eFuse_0. When this field is 0x1, writes to eFuse0_High and eFuse0_Low will be ignored.
1	FSB1	RW 0x0	eFuse1 Security Bit. This bit is not writable when BurnMode = 0. When BurnMode = 1, software may change the field value from 0 to 1. A 0x1 in this field followed by a <eFuse1_Write_Trigger> programs the FSB of eFuse_1 to 0x1. After this scenario, a write cannot be performed to eFuse_1. When this field is 0x1, writes to eFuse1_High and eFuse1_Low will be ignored.
31:2	Reserved	RSVD 0x0	Reserved



**Table 803: eFuse0 Low Register**  
Offset: 0x100A4

Bit	Field	Type/InitVal	Description
31:0	eFuse0 Low	RW 0x0	<p>Corresponds to eFuse0[31:0].</p> <p>This register is writable only when &lt;FSB0&gt; = 0.</p> <p>Upon reset de-assertion, data from eFuse0[31:0] is loaded into this register. If the protection bit of eFuse0 is not set, then software has the ability to write to this register the value that it will burn into eFuse0[31:0].</p> <p><b>NOTE:</b> Per bit, clearing a value (i.e., writing 0x0 to a bit that hold the value of 0x1) is not permitted.</p> <p>Since an eFuse bit can not be erased, clearing a bit will cause the data stored in the eFuse after the trigger command to be different than the data in this register.</p>

**Table 804: eFuse0 High Register**  
Offset: 0x100A8

Bit	Field	Type/InitVal	Description
31:0	eFuse0 High	RW 0x0	<p>Corresponds to eFuse0[63:32].</p> <p>This register is writable only when &lt;FSB0&gt; = 0.</p> <p>Upon reset de-assertion, data from eFuse0[63:32] is loaded into this register. If the protection bit of eFuse0 is not set, then software has the ability to write to this register the value that it will burn into eFuse0[63:32].</p> <p><b>NOTE:</b> Per bit, clearing a value (i.e., writing 0x0 to a bit that is 0x1) is not permitted.</p> <p>Since an eFuse bit can not be erased, clearing a bit will cause the data stored in the eFuse after the trigger command to be different than the data in this register.</p>

**Table 805: eFuse1 Low Register**  
Offset: 0x100AC

Bit	Field	Type/InitVal	Description
31:0	eFuse1 Low	RW 0x0	<p>Corresponds to eFuse1[31:0].</p> <p>This register is writable only when &lt;FSB1&gt; = 0.</p> <p>Upon reset de-assertion, data from eFuse1[31:0] is loaded into this register. If the protection bit of eFuse1 is not set, then software has the ability to write to this register the value that it will burn into eFuse1[31:0].</p> <p><b>NOTE:</b> Per bit, clearing a value (i.e., writing 0x0 to a bit that is 0x1) is not permitted.</p> <p>Since an eFuse bit can not be erased, clearing a bit will cause the data stored in the eFuse after the trigger command to be different than the data in this register.</p>

**Table 806: eFuse1 High Register**  
**Offset: 0x100B0**

Bit	Field	Type/InitVal	Description
31:0	eFuse1 High	RW 0x0	<p>Corresponds to eFuse1[63:32].</p> <p>This register is writable only when &lt;FSB1&gt; = 0.</p> <p>Upon reset de-assertion, data from eFuse1[63:32] is loaded into this register. If the protection bit of eFuse1 is not set, then software is able to write to this register the value that it will burn into eFuse1[63:32].</p> <p><b>NOTE:</b> Per bit, clearing a value (i.e., writing 0x0 to a bit that is 0x1) is not permitted.</p> <p>Since an eFuse bit can not be erased, clearing a bit will cause the data stored in the eFuse after the trigger command to be different than the data in this register.</p>

**Table 807: eFuse Control Register**  
**Offset: 0x100B4**

Bit	Field	Type/InitVal	Description
0	Burn Mode	RW 0x0	<p>Setting this bit is possible only when &lt;FSB0&gt; = 0 or &lt;FSB1&gt; = 0.</p> <p>Any writes to eFuse data registers (&lt;eFuse0_Low&gt;, &lt;eFuse0_High&gt;, &lt;eFuse1_Low&gt;, &lt;eFuse1_High&gt;), eFuse Protection or to eFuse Burn Trigger registers is ignored when this bit is not set.</p>
1	eFuse0 Write Trigger	WO 0x0	When Burn mode is active and 0x1 is written to this field, then the data in eFuse0_High, eFuse0_Low and FSB0 are serially written in to the eFuse_0 instance.
2	eFuse1 Write Trigger	WO 0x0	When Burn mode is active and 0x1 is written to this field, then the data in eFuse1_High, eFuse1_Low and FSB1 are serially written in to the eFuse_1 instance.
15:3	Reserved	RSVD 0x0	Reserved
16	eFuse Burn Done	RO 0x0	Cleared when one of the Write Triggers is written. Set upon completion of burn process.
31:17	Reserved	RSVD 0x0	Reserved

## A.24 Miscellaneous Registers

The following table provides a summarized list of all of the Miscellaneous registers, including the register names, their type, offset, and a reference to the corresponding table and page for a detailed description of each register and its fields.

**Table 808: Register Map Table for the Miscellaneous Registers**

Register Name	Offset	Table and Page
Device ID Register	0x10034	<a href="#">Table 809, p. 782</a>
Clock Control Register	0x1004C	<a href="#">Table 810, p. 782</a>
SYSRSTn Length Counter Register	0x10050	<a href="#">Table 811, p. 783</a>
Analog Group Configuration Register	0x1007C	<a href="#">Table 812, p. 783</a>
SSCG Configuration Register	0x100D8	<a href="#">Table 813, p. 783</a>
PTP Clock Configuration Register	0x100DC	<a href="#">Table 814, p. 784</a>
IO Configuration 0 Register	0x100E0	<a href="#">Table 815, p. 784</a>

**Table 809: Device ID Register**

Offset: 0x10034

Bit	Field	Type/InitVal	Description
1:0	Device ID	RO 0x2	For the 88F6180, the initial value is 0x0. For the 88F6190 and 88F6192, the initial value is 0x1. For the 88F6281, the initial value is 0x2.
31:2	Reserved	RSVD 0x0	Reserved

**Table 810: Clock Control Register**

Offset: 0x1004C

Bit	Field	Type/InitVal	Description
0	DCO_REF_CLK_SEL	RW 0x0	Defines the reference clock of the DCO in the audio unit. 0 = REF_CLK_XIN 1 = PTP_IN
31:1	Reserved	RSVD 0x0	Reserved

**Table 811: SYSRSTn Length Counter Register**  
**Offset: 0x10050**

Bit	Field	Type/InitVal	Description
28:0	Count	RO 0x0	SYSRST Length Count. The value in this field specifies the number of REF_CLK cycles that the SYSRSTn was asserted. Software may clear this field by writing 0x1 to <Clr>.
30:29	Reserved	RSVD 0x0	Reserved
31	Clr	WO 0x0	Writing 0x1 to this field clears the <Count> field.

**Table 812: Analog Group Configuration Register**  
**Offset: 0x1007C**

Bit	Field	Type/InitVal	Description
2:0	Reserved	RSVD 0x0	Reserved
4:3	ANA_GRP0_BGR_SEL	RW 0x0	Band Gap Select Must be set to 0. 0 = TSMC BJT model 1 = Modified BJT model
31:5	Reserved	RSVD 0x3	Reserved

**Table 813: SSCG Configuration Register**  
**Offset: 0x100D8**

Bit	Field	Type/InitVal	Description
7:0	SSCG High Boundary	RW 0x80	<p>Notation:</p> <ul style="list-style-type: none"> <li>- H = SSCG_High_Boundary</li> <li>- L = SSCG_Low_Boundary</li> </ul> <p>Modulation frequency:</p> <ul style="list-style-type: none"> <li>- For up spread or down spread: Modulation frequency = <math>550\text{e}6 / [(H-L)*H^2]</math> Hz</li> <li>- For center spread: Modulation frequency = <math>550\text{e}6 / [(H-L)*H^4]</math> Hz</li> </ul> <p>Spread percentage:</p> <ul style="list-style-type: none"> <li>- For up spread or down spread: <math>0.96*(H-L)/H \%</math> pk to pk</li> <li>- For center spread: <math>0.96*(H-L)/H \%</math> pk to pk</li> </ul> <p>For example, to set ~33 KHz modulation frequency and ~0.5% down spread: H = 128 and L = 64.</p>



**Table 813: SSCG Configuration Register (Continued)**  
Offset: 0x100D8

Bit	Field	Type/InitVal	Description
15:8	SSCG Low Boundary	RW 0x0	See <SSCG_High_Boundary> for a description. Note: SSCG_High_Boundary must be greater than SSCG_Low_Boundary.
17:16	SSCG Mode	RW 0x0	SSCG Configuration 0 = Spread down 1 = Spread up 2 = Central spread 3 = Reserved
31:18	Reserved	RSVD 0x0	Reserved

**Table 814: PTP Clock Configuration Register**  
Offset: 0x100DC

Bit	Field	Type/InitVal	Description
1:0	Reserved	RSVD 0x0	Reserved
2	PTP_SEL	RW 0x0	When the PTP clock is active, this bit will be driven from one of MPP's . <b>NOTE:</b> In addition, the appropriate MPP Selector must be configured.
31:3	Reserved	RSVD 0x1FFE000	Reserved

**Table 815: IO Configuration 0 Register**  
Offset: 0x100E0

Bit	Field	Type/InitVal	Description
2:0	Reserved	RW 0x3	Reserved Must be 0x3.
5:3	Reserved	RW 0x3	Reserved Must be 0x3.
6	Reserved	RSVD 0x0	Reserved
7	RGMII PADS Voltage	RW 0x0	Configures the I/O voltage of the pads connected to Gigabit Ethernet interface. For the 88F6180, this is VDD_GE. For the 88F6190, 88F6192, and 88F6281, this is VDD_GE_A. 0 = 3.3V 1 = 1.8V

**Table 815: IO Configuration 0 Register (Continued)**  
Offset: 0x100E0

Bit	Field	Type/InitVal	Description
10:8	Reserved	RW 0x3	Reserved Must be 0x3.
13:11	Reserved	RW 0x3	Reserved Must be 0x3.
14	Reserved	RSVD 0x0	Reserved
15	MPP RGMII PADS Voltage	RW 0x0	Configures the I/O voltage of the pads connected to MPP[35:20]. For the 88F6180, this field is not applicable. For the 88F6190, 88F6192, and 88F6281, this is VDD_GE_B. 0 = 3.3V 1 = 1.8V
18:16	Reserved	RW 0x3	Reserved Must be 0x3.
21:19	Reserved	RW 0x3	Reserved Must be 0x3.
23:22	Reserved	RSVD 0x0	Reserved
26:24	Reserved	RW 0x3	Reserved Must be 0x3.
29:27	Reserved	RW 0x3	Reserved Must be 0x3.
31:30	Reserved	RSVD 0x0	Reserved

# B

## Revision History

**Table 816: Revision History**

Revision	Date	Comments
C	December 2, 2008	Revision
		<ol style="list-style-type: none"><li>1. Revised the specifications to indicate that the the 88F6180 supports two UART ports.</li><li>2. Revised <a href="#">Figure 2, 88F6190 Interface Block Diagram, on page 22</a>.</li><li>3. In <a href="#">Section 1.2, Overview of Functions and Interfaces, on page 25</a>, revised the descriptions of the: Gigabit Ethernet interface, TDM interface, and XOR engine.</li><li>4. In the introduction to <a href="#">Section 4, DDR SDRAM Controller, on page 44</a>, revised this sentence: The DRAM controller supports up to a 128-byte burst per single transaction from the Mbus port and up to a 32-byte burst from the Mbus-L port.</li><li>5. Revised the last sentence of <a href="#">Section 4.4, DRAM Burst, on page 48</a> to indicate that the redundant cycles are masked.</li><li>6. In <a href="#">Section 5.1, Functional Description, on page 56</a>, added the term SLAC to the bullet:<ul style="list-style-type: none"><li>• PCM bus interface for telephony SLIC/SLAC/codec devices.</li></ul></li><li>7. Deleted the PCI Express Clock Request (CLKREQ#) Signaling section, since this device does not support that feature.</li><li>8. Revised the introduction to <a href="#">Section 8, Gigabit Ethernet Controller, on page 113</a>.</li><li>9. At the end of <a href="#">Section 8.3.3.1, Transmit Operation, on page 117</a> added: If generation of IP or Layer4 checksum is required, then the maximum packet size is 1.6 KB.</li><li>10. Removed the Gigabit Ethernet <i>Port Loopback</i> section, since this device does not support that feature.</li><li>11. Added <a href="#">Section 8.20, Precise Time Protocol (PTP), on page 158</a>.</li><li>12. Revised <a href="#">Section 15.3.1, Direct Read from SPI, on page 238</a>.</li><li>13. In <a href="#">Section 18.3, Clocks, on page 266</a> added a description of the timestamp counter clock.</li><li>14. Revised <a href="#">Section 18.6.3.1, TS Output Descriptor Structure, on page 270</a> and <a href="#">Section 18.6.3.1, TS Output Descriptor Structure, on page 270</a> to indicate the differences between aggregate and non-aggregate modes.</li><li>15. In <a href="#">Section 15.3.4, DMA Based SPI, on page 239</a>, revised the procedure and added a note: Direct Write To SPI cannot be used for flash devices.</li><li>16. Added <a href="#">Section 23.1, Typical eFuse Applications, on page 286</a>.</li><li>17. Revised <a href="#">Section 23.3, eFuse Program and Lock, on page 286</a>.</li><li>18. In <a href="#">Table 77, Main Header Format, on page 293</a>, indicated that the UART boot is from UART port0.</li><li>19. In <a href="#">Section 24.2.5, BootROM Firmware Boot Sequence, on page 295</a>, indicated that the UART boot is from UART port0.</li><li>20. In step 5 of <a href="#">Section 24.2.5.5, Debug and Error Handling, on page 300</a>, indicated that the UART port is UART0.</li><li>21. In <a href="#">Table 79, Types of NAND Flash Read Commands Supported, on page 301</a>, revised rows 1 and 3.</li><li>22. In <a href="#">Table 80, Types of ECC Protocols Supported per Flash Type, on page 302</a>, revised row 1.</li><li>23. Revised the entire <a href="#">Section 24.3, Power Management, on page 303</a>.</li><li>24. Revised the register name and the description of the &lt;Base&gt; field in the following registers:<ul style="list-style-type: none"><li>• CPU CS Window0 Base Address Register (<a href="#">Table 160 p. 390</a>)</li><li>• CPU CS Window1 Base Address Register (<a href="#">Table 162 p. 390</a>)</li><li>• CPU CS Window2 Base Address Register (<a href="#">Table 164 p. 391</a>)</li><li>• CPU CS Window3 Base Address Register (<a href="#">Table 166 p. 392</a>)</li></ul></li></ol>

**Table 816: Revision History**

Revision	Date	Comments
25.		Revised the register name and the descriptions of the <Win_CS> and <Size> fields in the following registers: • CPU CS Window0 Size Register ( <a href="#">Table 161 p. 390</a> ) • CPU CS Window1 Size Register ( <a href="#">Table 163 p. 391</a> ) • CPU CS Window2 Size Register ( <a href="#">Table 165 p. 392</a> ) • CPU CS Window3 Size Register ( <a href="#">Table 167 p. 392</a> )
26.		In the SDRAM Address Control Register ( <a href="#">Table 172 p. 398</a> ), revised the description of the following fields: <CS0Width>, <CS0Size>, <CS1Width>, <CS1Size>, <CS2Width>, <CS2Size>, <CS3Width>, and <CS3Size>.
27.		Revised the description of the <WR> field in the SDRAM Mode Register ( <a href="#">Table 175 p. 401</a> ).
28.		Revised the description of the <P2D Lat> field in the DDR Controller Control (High) Register ( <a href="#">Table 177 p. 403</a> ) and changed bit[7] to reserved, must be 0.
29.		Revised the register name and revised the initial values and descriptions of some of the field in the following registers: • Channel 0 Delay Control Register ( <a href="#">Table 205 p. 421</a> ) • Channel 0 Transmit Data Start Address Register ( <a href="#">Table 209 p. 423</a> ) • Channel 0 Receive Data Start Address Register ( <a href="#">Table 210 p. 424</a> ) • TDM Channel0 Wideband Delay Control Register ( <a href="#">Table 229 p. 431</a> )
30.		Added the following register tables: • Channel 1 Delay Control Register. • Channel 1 Transmit Data Start Address Register. • Channel 1 Receive Data Start Address Register. • TDM Channel 1 Wideband Delay Control Register.
31.		Revised the initial value of the <Size> field to 0x0FFF in the following registers: • Window0 Control Register ( <a href="#">Table 233 p. 433</a> )—In this register, also revised the description of the <WinEn> field. • Window1 Control Register ( <a href="#">Table 235 p. 434</a> ). • Window3 Control Register ( <a href="#">Table 239 p. 435</a> ).
32.		In the <IEMR> field in the Interrupt Event Mask Register ( <a href="#">Table 216 p. 426</a> ), revised the initial value from 0x1 to 0x3FFF. In the <Reserved> field of that register, revised the initial value from 0x1 to 0x0.
33.		In the <Reserved> field in the Interrupt Status Mask Register ( <a href="#">Table 217 p. 426</a> ), revised the initial value from 0x3FFF to 0x0.
34.		In PCI Express Link Capabilities Register ( <a href="#">Table 294 p. 465</a> ), changed bit [18] to reserved.
35.		In PCI Express Link Control Status Register ( <a href="#">Table 295 p. 466</a> ), changed bit [8] to reserved.
36.		In PCI Express Power Management Extended Register ( <a href="#">Table 313 p. 479</a> ), changed bit [16] to reserved.
37.		Revised the initial value of the <HOTPLUG_TIMER> field in the PHY Mode 4 Register ( <a href="#">Table 372 p. 530</a> ) from 0x0 to 0x6.
38.		In the PHY Mode 2 Register ( <a href="#">Table 374 p. 532</a> ), revised the initial value of the: • <TXIMP> field from 0x9 to 0x6. • <EXTRXIMP> field from 0x9 to 0x6.
39.		In the Port Serial Control0 (PSC0) Register ( <a href="#">Table 427 p. 566</a> ): • Revised the description of the <ForceFCMode> field. • Bits [15] and [16] where changed to: Reserved, must be 0x0.
40.		Revised the initial value of the <Reserved> field in the Port Serial Control1 (PSC1) Register ( <a href="#">Table 430 p. 572</a> ) (bit [13]) from 0x1 to 0x0.
41.		Revised the description of <PTKNRT> field in the Port Transmit Token-Bucket Rate Configuration (PTTBRC) Register ( <a href="#">Table 452 p. 591</a> ).
42.		Revised the initial value of the <PMTU> field in the Port Maximum Transmit Unit (PMTU) Register ( <a href="#">Table 454 p. 592</a> ) from 0x0 to 0x24.
43.		Revised the description of the <QTKNBKT> field in the Queue Transmit Token-Bucket Counter (QxTTBC) Register (n=0–7) ( <a href="#">Table 456 p. 593</a> ).
44.		Revised the description of the <QTKNRT> and the <QMTBS> field in the Transmit Queue Token Bucket Configuration (TQxTBC) Register (n=0–7) ( <a href="#">Table 457 p. 594</a> ).



Table 816: Revision History

Revision	Date	Comments
B	September 18, 2008	<p>First revision</p> <ol style="list-style-type: none"><li>Added device 88F6190 to the specification.</li><li>Revised all references to the Real-Time Clock (RTC), NAND flash, and SDIO to indicate that they apply to all devices.</li></ol>

**Table 816: Revision History**

Revision	Date	Comments
3.		Revised all references to the Audio unit (I2S / S/PDIF) to indicate that it also applies to the 88F6180.
4.		Revised all references to the Gigabit Ethernet interface to indicate that the 88F6180 supports MII/MMII, without COL, CRS, RX_ERR, TS_ERR, TX_CLK.
5.		Revised <a href="#">Figure 1, 88F6180 Interface Block Diagram, on page 21</a> and <a href="#">Figure 3, 88F6192 Interface Block Diagram, on page 23</a> , and added <a href="#">Figure 2, 88F6190 Interface Block Diagram, on page 22</a> and <a href="#">Figure 4, 88F6281 Interface Block Diagram, on page 24</a> .
6.		In <a href="#">Section 1.2, Overview of Functions and Interfaces, on page 25</a> , revised the description of the: CPU, SDRAM interface, Serial ATA interface, Gigabit Ethernet interface, XOR engine, TWSI interface, NAND flash interface, GPIO interface, and RTC.
7.		Revised <a href="#">Table 1, 88F6180, 88F619x, and 88F6281 Device Differences and Similarities, on page 31</a> and the notes after the table.
8.		In <a href="#">Table 2, Units IDs and Attributes—CPU, on page 35</a> and <a href="#">Table 3, Unit IDs and Attributes—PCI Express, on page 38</a> , revised the description of Attributes[7:0].
9.		In <a href="#">Section 2.3, PCI Express Address Decoding, on page 37</a> , revised the paragraph after the note.
10.		Revised the note in <a href="#">Section 2.12, Transport Stream (TS) Address Map (88F6192/88F6281 Only), on page 41</a> , to indicate that the SDRAM access from TS unit must not cross 32-byte boundary.
11.		Revised <a href="#">Section 3, Sheeva™ CPU Core</a> .
12.		Revised <a href="#">Section 4.14, SDRAM Read Data Sample, on page 53</a> .
13.		Revised <a href="#">Section 4.15, DDR2 On Die Termination (ODT), on page 53</a> , to reflect the number of ODT signals in each device.
14.		In <a href="#">Section 5, Time Division Multiplexing (TDM) Unit (88F6192 and 88F6281 Only), on page 56</a> , added reference to SLAC.
15.		Revised <a href="#">Table 10, Time Division Multiplexing (TDM) Interface Signals, on page 58</a> .
16.		Revised <a href="#">Section 5.3, TDM (SLIC/Codec) Registers Access via SPI, on page 71</a> .
17.		In <a href="#">Section 6, PCI Express Interface, on page 74</a> :
		• Revised the bullets in the introduction.
		• Revised:
		- <a href="#">Section 6.1.1, PHY Layer, on page 75</a>
		- <a href="#">Section 6.1.2, MAC Layer, on page 75</a>
		- <a href="#">Section 6.2, Link Initialization, on page 76</a>
		• Revised the power management description in <a href="#">Table 12, Supported Message Groups—Endpoint Mode, on page 80</a> .
		• Added:
		- <a href="#">Section 6.11, Message Signaled Interrupts (MSI), on page 81</a>
		- <a href="#">Section 6.16, Link Disable, on page 83</a>
		- <a href="#">Section 6.17, Power Management, on page 83</a>
		• Deleted the PHY Shutdown section.
18.		In <a href="#">Section 8, Gigabit Ethernet Controller, on page 113</a> , revised the introduction to include a description of the setting of the port modes. Also revised all references to Jumbo frames since checksum for Jumbo frames is not support on transmit.
19.		Revised <a href="#">Section 8.3.3.1, Transmit Operation, on page 117</a> .
20.		Revised the note in <a href="#">Section 8.3.3.3, Zero Padding of Short Frames, on page 120</a> .
21.		Revised the first two paragraphs of <a href="#">Section 8.3.3.6, TCP Checksum Generation, on page 121</a> .
22.		Revised <a href="#">Section 8.3.3.10, Transmit Descriptor Structure, on page 122</a> , to state that a descriptor may not be placed on a NAND Flash device.
23.		In <a href="#">Table 35, Transmit Descriptor Command/Status, on page 123</a> , revised the description of bits [9], [10], and [15].
24.		In <a href="#">Section 8.3.3.13, Transmit DMA Notes, on page 125</a> , revised the paragraphs starting with <i>A transmit underrun occurs and To stop DMA operation</i> .



**Table 816: Revision History**

Revision	Date	Comments
25.		Revised the fields and registers mentioned in the following sections: <ul style="list-style-type: none"><li>• <a href="#">Section 8.8.4, Transmit Queue Bandwidth Limitation, on page 146</a></li><li>• <a href="#">Section 8.8.5, Transmit Port Bandwidth Limitation, on page 146</a></li><li>• <a href="#">Section 8.8.6, Maximum Transmit Unit, on page 146</a></li></ul>
26.		Revised the beginning of <a href="#">Section 10.4.1, Using the Security Accelerator, on page 189</a> .
27.		Revised the first sentence in <a href="#">Section 11.1.4, Memory Initialization, on page 209</a> .
28.		In <a href="#">Section 12.2.6, TWSI Port Master Operation, on page 226</a> , revised step 1 to state that the CPU sets the <Start> bit.
29.		Revised all of <a href="#">Section 13, UART Interface, on page 228</a> .
30.		In <a href="#">Section 14.3.1, Guidelines for Access to NAND Flash, on page 232</a> : <ul style="list-style-type: none"><li>• Added a description of the Command phase.</li><li>• Added a note to the end of the section describing connection to the NAND Flash from the device bus.</li></ul>
31.		Revised <a href="#">Figure 60, 8-bit NAND Flash Read Parameters Example, on page 233</a> and <a href="#">Figure 61, 8-bit NAND Flash Write Parameters Example, on page 234</a> to used the correct pin name for the NF_IO[7:0] pins.
32.		Made extensive revisions to <a href="#">Section 15.3, Direct Mode, on page 237</a> .
33.		Revised <a href="#">Table 74, Transport Stream (TS) Interface Signal Assignment, on page 264</a> .
34.		Revised the first paragraph of <a href="#">Section 18.6.4, TS Output Direction DMA Flow, on page 271</a> .
35.		In <a href="#">Section 18.8.1, TS Input Mode, on page 273</a> and <a href="#">Section 18.8.2, TS Output Mode, on page 274</a> added notes stating when to use Type 1 and Type 2 Aggregation.
36.		Revised revised the introduction to <a href="#">Section 20, Real-Time Clock (RTC) Unit, on page 278</a> and added <a href="#">Section 20.2.4, RTC Alarm Operation, on page 279</a> .
37.		Revised <a href="#">Section 22.2, Watchdog Timer, on page 283</a> .
38.		Added <a href="#">Section 22.3, RTC Alarm, on page 283</a> .
39.		Added <a href="#">Section 22.4, SYSRSTn Duration Counter, on page 285</a> .
40.		Added <a href="#">Section 23, eFuse, on page 286</a> .
41.		Made major changes to <a href="#">Section 24.2, BootROM Firmware, on page 290</a> .
42.		In <a href="#">Section 24.2.6.3, Boot from NAND Flash, on page 301</a> , added a note indicating that the <NFActCEnBoot> field in the NAND Flash Control Register ( <a href="#">Table 613 p. 676</a> ) controls this mode. Also, revised the description of that field.
43.		In <a href="#">Table 80, Types of ECC Protocols Supported per Flash Type, on page 302</a> , revised the read command sequence for the 512 byte page NAND Flash type.
44.		In <a href="#">Table 87, Register Field Type Codes, on page 352</a> , changed type ITO to RW (ITO) and revised the description. Also added type RW (TO).
45.		In the CPU Configuration Register ( <a href="#">Table 115 p. 365</a> ), changed bits [16] and [17] to reserved, must write 0x0.
46.		Revised the descriptions of the <CPU2MbusLTickDrv> field and the <CPU2MbusLTickSample> field in the CPU Configuration Register ( <a href="#">Table 115 p. 367</a> ).
47.		Revised the field descriptions in the CPU RAM Management Control2 Register ( <a href="#">Table 130 p. 377</a> ).
48.		Revised DDR Controller Control (Low) Register ( <a href="#">Table 169 p. 394</a> ) and DDR Controller Control (High) Register ( <a href="#">Table 177 p. 403</a> ).
49.		Revised the <CS1Size> field, the <CS2Size> field, and the <CS3Size> field in the SDRAM Address Control Register ( <a href="#">Table 172 p. 399</a> ), to provide a 2 Gb option.
50.		For the <SPM> field in the SControl Register ( <a href="#">Table 369 p. 525</a> ), changed the field type to RW.
51.		Added <a href="#">Section A.3.5, L2 Non Cacheable Address, on page 381</a> , with the Window0 Base Address Register ( <a href="#">Table 143 p. 381</a> ) through Window3 Size Address Register ( <a href="#">Table 150 p. 383</a> ).
52.		Changed bit [15] in SDRAM Configuration Register ( <a href="#">Table 168 p. 393</a> ) to reserved, must be 0.
53.		Revised SDRAM ODT Control (Low) Register ( <a href="#">Table 187 p. 407</a> ) and SDRAM ODT Control (High) Register ( <a href="#">Table 188 p. 408</a> ).
54.		In <a href="#">Table 203, PCM Control Register, on page 418</a> , added descriptions for bits [6], [8], [9], [10], and [11].

**Table 816: Revision History**

Revision	Date	Comments
55.		Revised the description of bits [31:16] in the: <ul style="list-style-type: none"><li>• PCI Express Window3 Control Register (<a href="#">Table 252 p. 443</a>)</li><li>• PCI Express Window4 Control Register (<a href="#">Table 255 p. 444</a>)</li><li>• PCI Express Window5 Control Register (<a href="#">Table 259 p. 446</a>)</li></ul>
56.		Revised the description of the <MultiEn> field in the PCI Express MSI Message Control Register ( <a href="#">Table 287 p. 460</a> ).
57.		In <a href="#">Table 292, PCI Express Device Capabilities Register, on page 462</a> , revised the bit descriptions.
58.		Revised the description of the <AspmSup> field and the <L0sExtLat> field in the PCI Express Link Capabilities Register ( <a href="#">Table 294 p. 466</a> ).
59.		Revised the description of the <Reserved> field in the PCI Express Link Capabilities Register ( <a href="#">Table 294 p. 466</a> ).
60.		Revised the description of the <AspmCnt> field and the <NegLnkWdth> field in the PCI Express Link Control Status Register ( <a href="#">Table 295 p. 467</a> ).
61.		In the PCI Express Control Register ( <a href="#">Table 307 p. 474</a> ), revised the description of the following fields: <ConfLinkX1>, <Conf_Training_Disable>, and <Crs_Enable>.
62.		Revised the description of the <PhyAddr> field in the PCI Express PHY Indirect Access Register ( <a href="#">Table 319 p. 484</a> ).
63.		In the Basic DMA Command Register ( <a href="#">Table 327 p. 494</a> ), revised the description of bit [3].
64.		In the EDMA Configuration Register ( <a href="#">Table 333 p. 498</a> ), revised the description of bits [8], [11], [17], and [18].
65.		Revised the description of the <SAICOALT> field in the SATAHC Interrupt Coalescing Threshold Register ( <a href="#">Table 351 p. 511</a> ).
66.		In the SATAHC Interrupt Cause Register ( <a href="#">Table 353 p. 512</a> ), added bits [1]and [9] and revised the description of the <SaCrpb0Done/DMA0Done> field and the <SaDevInterrupt1> field.
67.		Revised the description of the <Sata1Done> field and the <Sata1DmaDone> field in the SATAHC Main Interrupt Cause Register ( <a href="#">Table 354 p. 513</a> ).
68.		In the Window0 Base Register ( <a href="#">Table 358 p. 516</a> ), revised the description of field [31:16].
69.		Revised the description of the <RGMIIEn> field in the Port Serial Control1 (PSC1) Register ( <a href="#">Table 430 p. 571</a> ).
70.		Revised <a href="#">Section A.8.2.1, Tx Queues Arbiter, on page 590</a> .
71.		Added the <PTKNRT> field in the Port Transmit Token-Bucket Rate Configuration (PTTBRC) Register ( <a href="#">Table 452 p. 591</a> ).
72.		Added the <PTP_SYNC_ENB> field in the Transmit Queue Command1 (TQC1) Register ( <a href="#">Table 453 p. 591</a> ).
73.		Added the <HiTKNinAsyncPkt> field in the High Token in Asynchronous Packet (HITKNinASYNCPKT) Register ( <a href="#">Table 462 p. 595</a> ).
74.		Revised the description of the <Target> bit in the: <ul style="list-style-type: none"><li>• USB 2.0 Window0 Control Register (<a href="#">Table 501 p. 625</a>)</li><li>• USB 2.0 Window1 Control Register (<a href="#">Table 503 p. 626</a>)</li><li>• USB 2.0 Window2 Control Register (<a href="#">Table 505 p. 627</a>)</li><li>• USB 2.0 Window3 Control Register (<a href="#">Table 507 p. 628</a>)</li></ul>
75.		Revise the offset of the USB 2.0 Power Control Register ( <a href="#">Table 514 p. 631</a> ) to 0x50400.
76.		Added <a href="#">Section A.9.5, USB Controller Registers, on page 632</a> .
77.		In Security Accelerator Configuration Register ( <a href="#">Table 560 p. 648</a> ), changed bit [12] to reserved, must be 0.
78.		In Base Address Register (n=0–3) ( <a href="#">Table 571 p. 652</a> ), revised the offset to 0x3AA00 and revised the note in the <Base> field to state that the initial value for BARI1 is 0x1000. The note previously stated that the initial value was 0x1.



Table 816: Revision History

Revision	Date	Comments
A	February 4, 2008	<p>79. Revised the offsets of the following registers to 0x3Axx instead of 0x30Axx:</p> <ul style="list-style-type: none"><li>• Window Control Register (n=0–3) (<a href="#">Table 572 p. 652</a>)</li><li>• Control Register (<a href="#">Table 573 p. 653</a>)</li><li>• TDMA Byte Count Register (<a href="#">Table 574 p. 654</a>)</li><li>• TDMA Source Address Register (<a href="#">Table 575 p. 655</a>)</li><li>• TDMA Destination Address Register (<a href="#">Table 576 p. 655</a>)</li><li>• Next Descriptor Pointer Register (<a href="#">Table 577 p. 655</a>)</li><li>• Current Descriptor Pointer Register (<a href="#">Table 578 p. 655</a>)</li><li>• TDMA Error Cause Register (<a href="#">Table 579 p. 656</a>)</li><li>• TDMA Error Mask Register (<a href="#">Table 580 p. 656</a>)</li></ul> <p>80. Revised the initial values from 0x0 to 0x3 in &lt;DstBurstLimit&gt; and &lt;SrcBurstLimit&gt; field in the Control Register (<a href="#">Table 573 p. 654</a>).</p> <p>81. Added the TWSI Initialization Last Data Register (<a href="#">Table 609 p. 674</a>).</p> <p>82. In the NAND Flash Control Register (<a href="#">Table 613 p. 676</a>), revised the bit descriptions and values.</p> <p>83. Added the following SPI registers:</p> <ul style="list-style-type: none"><li>• Added Serial Memory Direct Write Configuration Register (<a href="#">Table 634 p. 688</a>)</li><li>• Revised Serial Memory Direct Write Header Register (<a href="#">Table 635 p. 688</a>)</li></ul> <p>84. Revised the description of the &lt;ClkDvdrMValue&gt; field in the Clock Divider Value Register (<a href="#">Table 726 p. 743</a>).</p> <p>85. Revised the description of the &lt;TSCK88&gt; field in the TSU Modes Register (<a href="#">Table 730 p. 744</a>).</p> <p>86. In <a href="#">Table 741, TS Interface Configuration Register, on page 750</a>, changed the initial value of bit [16] to 0x1 and of bit [17] to 0x0 and revised the descriptions of bits [7:2] and [17].</p> <p>87. Revised the description of the &lt;SampleAtReset&gt; field in the Sample at Reset Register (<a href="#">Table 800 p. 778</a>).</p> <p>88. Revise the description for the &lt;TS DMA Length&gt; field and the &lt;TS Data Watermark&gt; field in the TS DMA Parameter Register (<a href="#">Table 742 p. 752</a>).</p> <p>89. In TSU Enable Access Register (<a href="#">Table 752 p. 755</a>), revised the description of all of the fields.</p> <p>90. In TSU Test Register (<a href="#">Table 756 p. 757</a>), revised the description of bits [0] and [1].</p> <p>91. In TSU Interrupt Source Register (<a href="#">Table 757 p. 758</a>), revised the description of all of the fields.</p> <p>92. In TSU Aggregation Control Register (<a href="#">Table 764 p. 760</a>), revised the values in the description of bits [31:26].</p> <p>93. Added <a href="#">Section A.23, eFuse Registers, on page 779</a>.</p> <p>94. Revised the description of the &lt;SampleAtReset&gt; field in the Sample at Reset Register (<a href="#">Table 800 p. 778</a>).</p> <p>95. Revised <a href="#">Table 809, Device ID Register, on page 782</a> to show the function of bits [1:0].</p> <p>96. Revised the description of the &lt;DCO_REF_CLK_SEL&gt; field in the Clock Control Register (<a href="#">Table 810 p. 782</a>) to list: 0x0 = REF_CLK_XIN and 0x1 = PTP_IN.</p> <p>97. Added the SYSRSTn Length Counter Register (<a href="#">Table 811 p. 783</a>).</p> <p>98. Added the SSCG Configuration Register (<a href="#">Table 813 p. 783</a>).</p>





Marvell Semiconductor, Inc.  
5488 Marvell Lane  
Santa Clara, CA 95054, USA

Tel: 1.408.222.2500  
Fax: 1.408.752.9028

[www.marvell.com](http://www.marvell.com)

**Marvell. Moving Forward Faster**