



# DesignWare Cores Ethernet MAC Universal Databook

---

***DWC Ether MAC 10/100/1000 Universal***  
***DWC Ether MAC 10/100 Universal***

## Copyright Notice and Proprietary Information

Copyright © 2009 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

### Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

### Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

### Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, and Vera are registered trademarks of Synopsys, Inc.

### Trademarks (™)

Active Parasitics, AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BOA, BRT, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, DC Expert, DC Professional, DC Ultra, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, Direct RTL, Direct Silicon Access, Discovery, Dynamic-Macromodeling, Dynamic Model Switcher, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Formal Model Checker, FoundryModel, Frame Compiler, Galaxy, Gatran, HANEX, HDL Advisor, HDL Compiler, Hercules, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSIM<sup>plus</sup>, HSPICE-Link, iN-Tandem, Integrator, Interactive Waveform Viewer, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, Magellan, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Milkyway, ModelSource, Module Compiler, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Orion\_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Power Compiler, PowerCODE, PowerGate, ProFPGA, ProGen, Prospector, Raphael, Raphael-NES, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, Softwire, Source-Level Design, Star-RCXT, Star-SimXT, Taurus, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS Express, VCSI, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

### Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

Saber is a registered trademark of SabreMark Limited Partnership and is used under license.

All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.  
700 E. Middlefield Road  
Mountain View, CA 94043

[www.synopsys.com](http://www.synopsys.com)

# Contents

---

Revision History .....	11
<b>Preface .....</b>	<b>13</b>
Databook Organization .....	13
Reference Documentation .....	14
Authority .....	15
Customer Support .....	16
<b>Chapter 1</b>	
Product Overview .....	17
1.1 General Product Description .....	17
1.2 System Overview .....	18
1.2.1 System-Level Block Diagram .....	18
1.2.2 Interfaces .....	19
1.2.3 Transmit and Receive FIFOs .....	19
1.3 Features List .....	20
1.3.1 GMAC Core Features .....	20
1.3.2 DMA Block Features .....	21
1.3.3 Transaction Layer (MTL) Features .....	21
1.3.4 AMBA AHB Interface Features .....	22
1.3.5 AMBA AXI Interface Features .....	23
1.3.6 Monitoring, Test, and Debugging Support Features .....	23
1.4 Configurability .....	24
1.5 Deliverables .....	25
1.5.1 coreKit .....	25
1.5.2 Licensing Features .....	26
<b>Chapter 2</b>	
Building and Verifying Your Core .....	27
2.1 Setup Requirements .....	27
2.2 Basic Design Flow .....	27
2.3 Creating a Workspace .....	29
2.4 Configuration: Creating the RTL .....	30
2.5 Synthesis: Creating the Gate-Level Netlist .....	31
2.6 Simulation: Verifying DWC Ether MAC 10/100/1000 Universal .....	31
2.6.1 Generating a GTECH Model .....	31
2.6.2 Verifying DWC Ether MAC 10/100/1000 Universal .....	32

<b>Chapter 3</b>	
<b>Architecture</b>	.....
3.1 Introduction	.....
3.2 IEEE 1588-2002 Overview	.....
3.2.1 Reference Timing Source	.....
3.2.2 Transmit Path Functions	.....
3.2.3 Receive Path Functions	.....
3.2.4 Time Stamp Error Margin	.....
3.2.5 Frequency Range of Reference Timing Clock	.....
3.2.6 Advanced Time Stamp Feature Support	.....
3.3 AHB Application Host Interface	.....
3.3.1 AHB Slave Port Timing	.....
3.3.2 AHB Master Port Timing	.....
3.4 AXI Application Host Interface	.....
3.4.1 AXI Overview	.....
3.4.2 Burst Splitting and Burst Selection	.....
3.4.3 Outstanding Transactions	.....
3.4.4 Priority of AXI Requests	.....
3.4.5 Bursts Reordering and Data Interleaving	.....
3.4.6 AXI-ID Translation	.....
3.4.7 Endianess	.....
3.4.8 Posted Writes	.....
3.4.9 Error Response Handling	.....
3.4.10 AXI Memory Read	.....
3.4.11 AXI Memory Write	.....
3.4.12 AXI Early Burst Termination	.....
3.4.13 Slave Interface Support and Timing	.....
3.4.14 AXI Low Power Interface	.....
3.5 DMA Controller	.....
3.5.1 Initialization	.....
3.5.2 Transmission	.....
3.5.3 Reception	.....
3.5.4 Interrupts	.....
3.5.5 Error Response to DMA	.....
3.5.6 DMA Native Interface	.....
3.6 MAC Transaction Layer (MTL)	.....
3.6.1 Transmit Path	.....
3.6.2 Receive Path	.....
3.6.3 Interfacing With External Two-Port RAMs	.....
3.7 GMAC Core	.....
3.7.1 Transmission	.....
3.7.2 MAC Transmit Interface Protocol	.....
3.7.3 MAC Transmit Timing	.....
3.7.4 Reception	.....
3.7.5 MAC Receive Interface Protocol	.....
3.7.6 MAC Receive Timing	.....
3.7.7 MAC Control Interface Protocol	.....
3.7.8 System Time Register Module	.....

3.8 MAC Management Counters .....	130
3.8.1 Address Assignments .....	131
3.8.2 MMC Register Description .....	137
3.9 Power Management Block .....	146
3.9.1 PMT Block Description .....	147
3.9.2 Remote Wake-Up Frame Detection .....	149
3.9.3 Magic Packet Detection .....	149
3.9.4 System Considerations During Power-Down .....	150
3.10 Station Management Agent .....	151
3.10.1 Functions .....	151
3.10.2 GMII/MII Management Write Operation .....	152
3.10.3 GMII/MII Management Read Operation .....	153
3.11 Physical Coding Sublayer (PCS) .....	153
3.11.1 PCS Functions .....	154
3.12 Reduced Media Independent Interface .....	155
3.12.1 Block Diagram .....	156
3.12.2 Block Overview .....	156
3.12.3 Transmit Bit Ordering .....	157
3.12.4 RMII Transmit Timing Diagrams .....	158
3.13 Serial Media Independent Interface (SMII) .....	160
3.13.1 Block Diagram .....	161
3.13.2 Block Overview .....	161
3.13.3 SMII Timing .....	162
3.14 Reduced Gigabit Media Independent Interface .....	164
3.14.1 Block Diagram .....	165
3.14.2 Block Overview .....	165
3.14.3 RGMII Clocks .....	166
3.14.4 Signal Conversion .....	166
3.14.5 RGMII Transmit Timing .....	169
3.14.6 RGMII Receive Timing .....	169
3.15 Reduced Ten-Bit Interface .....	169
3.15.1 Block Diagram .....	170
3.15.2 Block Diagram Overview .....	170
3.15.3 RTBI Transmit Timing .....	171
3.16 Serial Gigabit Media Independent Interface .....	171
3.16.1 Block Diagram .....	171
3.16.2 Block Overview .....	171
3.16.3 Block Descriptions .....	172
3.17 Multiple PHY Interfaces .....	177
3.18 Interrupts From the GMAC Core .....	178
 Chapter 4	
Signal Descriptions .....	179
4.1 Top-Level I/O Diagram .....	179
4.2 Signal Descriptions .....	179
4.2.1 System Clock and Reset Signals (Default) .....	186
4.2.2 PHY Interface Signals (Default) .....	188
4.2.3 MAC Tx Interface Signals .....	193
4.2.4 MAC Rx Interface Signals .....	196
4.2.5 MAC Control Interface Signals .....	197

4.2.6 Application Clock and Reset Interface Signals .....	198
4.2.7 Application Transmit Interface (ATI) Signals .....	199
4.2.8 Application Receive Interface (ARI) Signals .....	202
4.2.9 MDC Interface Signals .....	204
4.2.10 AHB Master Interface Signals .....	207
4.2.11 AHB Slave Interface Signals .....	210
4.2.12 AXI Master Interface Signals .....	213
4.2.13 AXI Low Power Interface Signals .....	220
4.2.14 AXI Slave Interface Signals .....	221
4.2.15 External DPRAM Interface Signals .....	228
4.2.16 External DPRAM (Frame Length FIFO) Interface Signals .....	231
4.2.17 RGMII/RTBI Interface Signals (Optional) .....	233
4.2.18 RMII Interface Signals (Optional) .....	233
4.2.19 SMII Interface Signal (Optional) .....	234
4.2.20 PMT Interrupt Signal (Optional) .....	234
4.2.21 SGMII/TBI/RTBI Interface Signals (Optional) .....	235
4.2.22 SGMII Interface Signals (Optional) .....	237
4.2.23 TBI Interface Signals .....	237
4.2.24 SMA Interface Signals (Optional) .....	238
4.2.25 APB Interface Signals (Optional) .....	239
4.2.26 Sideband Signals (Optional) .....	240
4.2.27 IEEE 1588 Time Stamp Signals (Optional) .....	243
4.2.28 Test Mode .....	245
<b>Chapter 5</b>	
<b>Registers .....</b>	<b>247</b>
5.1 Register Maps .....	247
5.1.1 DMA Register Map .....	247
5.1.2 GMAC Register Map .....	248
5.2 Register Descriptions .....	256
5.2.1 DMA Register Description .....	257
5.2.2 GMAC Register Description .....	278
5.2.3 IEEE 1588 Time Stamp Registers .....	301
<b>Chapter 6</b>	
<b>Parameters .....</b>	<b>309</b>
6.1 Features .....	310
6.1.1 System Interface .....	310
6.1.2 General Features .....	314
6.1.3 PHY Line Interfaces (RGMII, RMII, SMII, SGMII, TBI, and RTBI) .....	318
6.2 Optional Modules .....	320
6.2.1 IEEE 1588 Time Stamp Block .....	320
6.2.2 Power Management Block .....	321
6.2.3 IP Checksum Block .....	321
6.2.4 GMAC Management (RMON) Counters .....	322
6.2.5 Station Management Block .....	335

<b>Chapter 7</b>	
Descriptors .....	337
7.1 Normal Descriptor Formats .....	337
7.1.1 Receive Descriptor .....	341
7.1.2 Transmit Descriptor .....	347
7.1.3 Descriptor Format With IEEE 1588 Time Stamping Enabled .....	352
7.2 Alternate or Enhanced Descriptors .....	356
7.2.1 Transmit Descriptor .....	356
7.2.2 Receive Descriptor .....	362
<b>Chapter 8</b>	
Implementation Guidelines .....	371
8.1 Clocks With GMII/MII .....	371
8.2 Clocks With TBI .....	372
8.3 Clocks With SGMII .....	373
8.4 Clocks With RMII .....	375
8.5 Clocks With SMII .....	376
8.5.1 Non Source Synchronous Mode .....	376
8.5.2 Source Synchronous Mode .....	377
8.6 Clocks With RGMII .....	379
8.7 Clocks With RTBI .....	380
8.8 Host (System Interface) Clock .....	381
8.9 Resets .....	381
8.10 SMA to PHY Interface .....	381
8.11 Timing Guidelines for Dual Data Rate RGMII/RTBI Outputs .....	383
8.12 Synthesis Guidelines .....	389
<b>Appendix A</b>	
Workspace Directory and Files .....	391
A.1 Workspace File Directory Structure .....	391
A.1.1 Directory Structures of Files (For Multiple Configurations) .....	391
A.1.2 Directory Structures for the Simulation Environment .....	393
A.2 Workspace Directory and File Descriptions .....	397
<b>Appendix B</b>	
GMAC-AHB Verification Environment Using AMBA and Ethernet VIPs .....	399
B.1 VTB Overview .....	399
B.2 Verification Flow .....	401
B.2.1 Transmission Verification Flow .....	401
B.2.2 Receive Verification Flow .....	406
B.3 Directory Structure .....	410
B.4 GPIO .....	411
B.4.1 GPIO_OUT (Tx) .....	411
B.4.2 GPIO_IN(Tx) .....	412
B.4.3 GPIO_OUT (Rx) .....	412
B.4.4 GPIO_IN (Rx) .....	413
B.5 SoC-Level Verification Guidelines .....	413
B.6 Running Simulations .....	414
B.7 Simulation PASS/FAIL .....	414

**Appendix C**

GMAC-AHB Verification Environment .....	415
C.1 VTB Overview .....	415
C.2 Initialization .....	417
C.3 Using VTB .....	417
C.4 Tool and Setup Requirements .....	420
C.5 Running Simulations .....	420
C.6 Simulation PASS/FAIL .....	420

**Appendix D**

GMAC-DMA Verification Environment .....	421
D.1 Testbench Overview .....	421
D.2 Test Flow .....	423
D.2.1 Frame Receive Flow .....	423
D.2.2 Frame Transmit Flow .....	423
D.3 Directory Structure .....	424
D.4 Running Simulations .....	425
D.5 Simulation PASS/FAIL .....	426

**Appendix E**

GMAC-MTL Verification Environment .....	427
E.1 Testbench Overview .....	427
E.2 Test Flow .....	429
E.3 Directory Structure .....	430
E.4 Running Simulations .....	431
E.5 Simulation PASS/FAIL .....	432

**Appendix F**

GMAC-Core Verification Environment .....	433
F.1 Testbench Overview .....	433
F.2 Test Flow .....	435
F.3 Directory Structure .....	436
F.4 Running Simulations .....	437
F.5 Simulation PASS/FAIL .....	438

**Appendix G**

GMAC-AXI Verification Environment .....	441
G.1 VTB Overview .....	441
G.2 Verification Flow .....	444
G.2.1 Transmission Verification Flow .....	444
G.2.2 Receive Verification Flow .....	448
G.2.3 Transmit-Receive Verification Flow .....	451
G.3 Directory Structure .....	456
G.4 GPIO .....	457
G.4.1 GPIO_OUT (Tx) .....	457
G.4.2 GPIO_IN (Tx) .....	457
G.4.3 GPIO_OUT (Rx) .....	458
G.4.4 GPIO_IN (Rx) .....	458
G.5 SoC-Level Verification Guidelines .....	459
G.6 Running Simulations .....	460
G.7 Simulation PASS/FAIL .....	460

<b>Appendix H</b>	
Area and Power .....	461
H.1 Area .....	461
H.1.1 Gate Count Summary .....	461
H.1.2 Area Differences Due to Scan Ready and Clock Gating .....	463
H.2 Power .....	465
<b>Appendix I</b>	
Programming Guide .....	467
I.1 DMA Initialization - Descriptors .....	467
I.2 MAC Initialization .....	468
I.3 Normal Receive and Transmit Operation .....	469
I.4 Stop and Start Operation .....	469
I.5 Programming Guideline for IEEE 1588 Time Stamping .....	470
I.5.1 Initialization Guideline for System Time Generation .....	470
I.5.2 System Time Correction .....	470
<b>Appendix J</b>	
Synchronizer Methods .....	473
J.1 Synchronizer 1: Simple Double Register Synchronizer (DWC_gmac_bcm21.v) .....	474
J.2 Synchronizer 2: Pulse Synchronizer (DWC_gmac_bcm22.v) .....	476
J.3 Synchronizer 3: Pulse Synchronizer with Acknowledge (DWC_gmac_bcm23.v) .....	477
J.4 Synchronizer 4: Gray Coded Synchronizer (DWC_gmac_bcm24.v) .....	478



# Revision History

Date	Version	Description
February, 2009	3.50a	<ul style="list-style-type: none"> <li>• Added support for Serial Media Independent Interface (SMII)</li> <li>• Added support for AMBA Advanced eXtensible Interface Bus (AXI) Application Host Interface</li> <li>• Updated support for IEEE 1588 standard</li> <li>• Fixed documentation errata <ul style="list-style-type: none"> <li>- Updated “Preface” on page <a href="#">13</a></li> <li>- Modified “Product Overview” on page <a href="#">17</a></li> <li>- Added “Auxiliary Snapshot” on page <a href="#">49</a></li> <li>- Added “AXI Application Host Interface” on page <a href="#">56</a></li> <li>- Added “AXI Low Power Interface” on page <a href="#">63</a></li> <li>- Added “ARI with Advanced Time Stamping” on page <a href="#">95</a></li> <li>- Added “Serial Media Independent Interface (SMII)” on page <a href="#">160</a></li> <li>- Added GMAC-AXI (GMII) Top-Level I/O Diagram <a href="#">Figure 4-5</a> on page <a href="#">184</a> and AXI Master and Slave signal descriptions (<a href="#">Table 4-12</a> on page <a href="#">213</a> and <a href="#">Table 4-14</a> on page <a href="#">221</a>, respectively)</li> <li>- Added SMII signal to <a href="#">Figure 4-6</a> on page <a href="#">185</a> and description to <a href="#">Table 4-19</a> on page <a href="#">234</a></li> <li>- Added AXI Bus Mode Register (modified “DMA Register Map” on page <a href="#">247</a> and added “Register 10 (AXI Bus Mode Register)” on page <a href="#">273</a>)</li> <li>- Added SMII and AXI configuration parameters to “System Interface” on page <a href="#">310</a></li> <li>- Added SMII register 54 to “DMA Register Map” on page <a href="#">247</a></li> <li>- Added SMII to register to “Register 54 (SGMII/RGMII/SMII Status Register)” on page <a href="#">300</a></li> <li>- Modified Time Stamp register descriptions (“IEEE 1588 Time Stamp Registers” on page <a href="#">301</a>)</li> <li>- Added SMII configuration parameters to “PHY Line Interfaces (RGMII, RMII, SMII, SGMII, TBI, and RTBI)” on page <a href="#">318</a></li> <li>- Added IEEE 1588 Time Stamp configuration parameters to <a href="#">Table 6-15</a> on page <a href="#">320</a></li> <li>- Modified Time Stamp descriptor descriptions “Descriptor Format With IEEE 1588 Time Stamping Enabled” on page <a href="#">352</a></li> <li>- Updated descriptor descriptions in “Alternate or Enhanced Descriptors” on page <a href="#">356</a></li> <li>- Added Implementation Guidelines for “Clocks With SMII” on page <a href="#">376</a></li> </ul> </li> </ul>

Date	Version	Description
(continued)		<ul style="list-style-type: none"> <li>- Added Implementation Guidelines for “<a href="#">Host (System Interface) Clock</a>” on page <a href="#">381</a></li> <li>- Added Implementation Guidelines for “<a href="#">Synthesis Guidelines</a>” on page <a href="#">389</a></li> <li>- Updated “<a href="#">Workspace Directory and Files</a>” on page <a href="#">391</a></li> <li>- Added “<a href="#">GMAC-AXI Verification Environment</a>” on page <a href="#">441</a></li> <li>- Added “<a href="#">Programming Guide</a>” on page <a href="#">467</a></li> <li>- Added “<a href="#">Synchronizer Methods</a>” on page <a href="#">473</a></li> </ul>
June 30, 2008	3.42a	Errata correction
February 7, 2008	3.41a	Errata correction
November 21, 2007	3.40a	<ul style="list-style-type: none"> <li>• Added IEEE 1588 support</li> <li>• Added SolvNet article citations</li> <li>• Added Appendix B, “<a href="#">GMAC-AHB Verification Environment Using AMBA and Ethernet VIPs</a>”</li> </ul>
March 22, 2007	3.30a	<ul style="list-style-type: none"> <li>• Added alternative descriptor structure</li> <li>• Added optional Transmit and Receive Checksum Offload engines and related RMON counters and interrupts registers</li> <li>• SMA-to-PHY interface updated and documented</li> <li>• Updated databook for minor new features</li> <li>• Fixed documentation errata</li> <li>• Updated gate count, area in “<a href="#">Area and Power</a>” on page <a href="#">461</a></li> </ul>
August 2006	3.20a	<ul style="list-style-type: none"> <li>• Fixed documentation errata</li> <li>• Edited selected text</li> <li>• Revised documentation of database file structures and user interface elements</li> <li>• Updated gate count, area in “<a href="#">Area and Power</a>” on page <a href="#">461</a></li> </ul>
October 2005	3.10a	<ul style="list-style-type: none"> <li>• Fixed documentation errata</li> <li>• Expanded architecture interface descriptions, adding timing diagrams to <ul style="list-style-type: none"> <li>- “<a href="#">AHB Application Host Interface</a>” on page <a href="#">50</a></li> <li>- “<a href="#">DMA Controller</a>” on page <a href="#">64</a></li> <li>- “<a href="#">MAC Transaction Layer (MTL)</a>” on page <a href="#">81</a></li> <li>- “<a href="#">GMAC Core</a>” on page <a href="#">102</a></li> </ul> </li> <li>• Added GMAC-DMA (GMII subsystem top-level I/O diagram (<a href="#">Figure 4-3</a>) to “<a href="#">Top-Level I/O Diagram</a>” on page <a href="#">179</a> and modified “<a href="#">Signal Descriptions</a>” on page <a href="#">179</a> accordingly (added <a href="#">Table 4-9</a> and <a href="#">Table 4-16</a>)</li> <li>• Added Appendixes B through F to describe the verification environments for the GMAC AHB, DMA, MTL, and Core interfaces.</li> </ul>
November 2005	3.10a	<ul style="list-style-type: none"> <li>• Added Revision History</li> <li>• Fixed documentation errata</li> <li>• Added notes for 10/100 Universal product applicability and operation</li> </ul>
June, 2005	3.0	First release

# Preface

---

This document describes Synopsys's DesignWare Cores Ethernet MAC Universal, release 3.50a. This product enables a host to transmit and receive data over Ethernet in compliance with the IEEE 802.3 specification. Ethernet MAC Universal (or GMAC-UNIV for short) is provided as two separate licenses, as follows:

- ❖ DWC Ether MAC 10/100/1000 Universal is the Media Access Controller for 10/100/1000 Ethernet.
- ❖ DWC Ether MAC 10/100 Universal is the Media Access Controller for 10/100 Ethernet.

Ethernet MAC Universal corresponds to either DWC Ether MAC 10/100/1000 Universal or DWC Ether MAC 10/100 Universal in the SolvNet database.

## Databook Organization

The chapters of this databook are organized as follows:

[Chapter 1, "Product Overview,"](#) describes GMAC-UNIV features and provides an overview of the architecture.

[Chapter 2, "Building and Verifying Your Core,"](#) provides an overview of setting up, configuring, and verifying the core.

[Chapter 3, "Architecture,"](#) describes the GMAC-UNIV interfaces, protocols, functionality, and implementation.

[Chapter 4, "Signal Descriptions,"](#) describe the GMAC-UNIV's top-level signals.

[Chapter 5, "Registers,"](#) maps and describes the function of the GMAC-UNIV's control and status registers.

[Chapter 6, "Parameters,"](#) describes the GMAC-UNIV configuration options and parameters.

[Chapter 7, "Descriptors,"](#) identifies and describes the GMAC UNIV descriptors.

[Chapter 8, "Implementation Guidelines,"](#) provides useful information on clocks and resets, scan, synthesis, and layout.

[Appendix A, "Workspace Directory and Files,"](#) describes the GMAC-UNIV database file structure.

[Appendix B, "GMAC-AHB Verification Environment Using AMBA and Ethernet VIPs"](#) describes effective use of AMBA and Ethernet Verification IP for verifying the GMAC-AHB.

[Appendix C, "GMAC-AHB Verification Environment,"](#) describes the Verilog Testbench (VTB) packaged in the coreKit.

[Appendix D, "GMAC-DMA Verification Environment,"](#) describes the example verification environment included in the coreKit for the GMAC-DMA configuration.

[Appendix E, "GMAC-MTL Verification Environment,"](#) describes the example verification environment included in the coreKit for the GMAC-MTL configuration.

Appendix F, “GMAC-Core Verification Environment,” describes the example verification environment included in the coreKit for the GMAC-CORE configuration.

Appendix G, “GMAC-AXI Verification Environment ,” describes effective use of AMBA and Ethernet Verification IP for verifying the GMAC-AXI.

Appendix H, “Area and Power,” summarizes area in terms of gate count, area differences due to scan ready and clock gating, and power dissipation estimates with and without clock gating.

Appendix I, “Programming Guide ,” provides instructions for initializing the DMA/MAC registers in the proper sequence.

Appendix J, “Synchronizer Methods,” documents the synchronizer methods (blocks of synchronizer functionality) used in DesignWare Ethernet Mac Universal IP to cross clock boundaries.

## Reference Documentation

The following references contain useful information concerning the protocols addressed by this IP core:

- ❖ *Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, Standard 802.3-2002, Institute of Electrical and Electronics Engineers (IEEE)
- ❖ *Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, Standard 1588-2002, IEEE
- ❖ *AMBA Specification* Revision 2.0, ARM Ltd. for AHB, APB interface
- ❖ *AMBA 3 Specification*, ARM Ltd for AXI interface
- ❖ *Reduced Gigabit Media Independent Interface (RGMII)*, Version 2.0, Broadcom Corp., Hewlett-Packard Co., Marvell Technology Group, Ltd.
- ❖ *Serial-GMII (SGMII) Specification*, Revision 1.8, Cisco Systems
- ❖ *Serial-MII (SMII) Specification*, Revision 2.1, Cisco Systems
- ❖ *RMII Specification*, Revision 1.2, RMII Consortium
- ❖ *RFC 768, User Datagram Protocol (UDP)*, DARPA Internet Program
- ❖ *RFC 791, Protocol Specification* (Internet Protocol, Version 4 (IPv4) Specification), DARPA Internet Program
- ❖ *RFC 792, Internet Control Message Protocol Specification*, DARPA Internet Program
- ❖ *RFC 793, Transmission Control Protocol (TCP)*, DARPA Internet Program
- ❖ *RFC 1071, Computing the Internet Checksum (memo)*, Network Working Group
- ❖ *RFC 2460, Internet Protocol, Version 6 (IPv6) Specification*, The Internet Society, Network Working Group
- ❖ *RFC 2819, Remote Network Monitoring Management Information Base*, The Internet Society, Network Working Group
- ❖ *RFC 2965, HTTP State Management Mechanism*, The Internet Society, Network Working Group
- ❖ *RFC 4443, Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, The Internet Society, Network Working Group
- ❖ 1588 Main Web Site, National Institute of Standards and Technology, <http://ieee1588.nist.gov/>
- ❖ *Hardware-Assisted IEEE 1588\* Implementation in the Intel® IXP46X Product Line* (white paper), Intel, Inc., March 2005

- ❖ *A Frequency Compensated Clock for Precision Synchronization using IEEE 1588 Protocol and its Application to Ethernet*, Sivaram Balasubramanian, Kendal R. Harris, and Anatoly Moldovansky, Rockwell Automation, November, 2003

## Authority

This databook is cited in the following SolvNet articles:

SolvNet Article No.	Title
019523	“DWC Ether MAC 10/100/1000 Mbps -Universal IP 3.x - software driver”
017998	“DWC-Ether_Mac 10/100/1000-Universal Behavior During a Transmit Underflow Event”
019407	“DWC Ether GMAC-Universal IP: Updating GMAC Configuration Register in RGMII Configuration”
017601	“FIFO size estimation for DWC Ether MAC 10100/1000 Universal IP”
019825	“DesignWare Cores Ethernet MAC 10/100/1000 Mbps Universal Core: Setting and Clearing the Own Bit Functionality in Transmit and Receive Descriptors”
021418	“Hash filtering in the GMAC core”
018112	“RMII, Transmit MII, and Receive MII Clock Balancing for GMAC-Universal 3.x IP”
018312	“Network Power management support in Synopsys' DWC Ether GMAC/EMAC Universal 3.x IP”
019417	“DWC Ether GMAC-Universal 3.x IP and Auto-Negotiation Support in PCS Mode of Operation”
017017	“VCI/PVCI Interface for 3.x releases of the DWC Ethernet MAC 10/100/1000-Universal and DWC Ethernet 10/100 MAC - Universal DMA Subsystem”
023329	“BYTE addressability on GMAC”
023331	“MAC addresses in GMAC core”
024530	“DWC GMAC 10/100/1000 Mbps: Ideal throughput with GMAC IP”
024534	“DWC GMAC 10/100/1000 Mbps: Type 1 and Type 2 checksum engines in GMAC”
023328	“Two consecutive writes to GMAC register”
023332	“Clock gating during power-down mode in GMAC core”
025602	“Speed control for MII in GMAC 10/100/1000Mbps”
025590	“Error Status Summary in DWC GMAC 10/100/1000Mbps”
025442	“PTP reference clock constraints in GMAC 10/100/1000”
025598	“Late collision before 64bytes in GMAC 10/100Mbps?”

## Customer Support

For customer support:

- ❖ Enter a call through SolvNet.
  - ◆ Go to <https://solvnet.synopsys.com/ManageCase?ccf=1> and provide the requested information, including:
    - ✧ Product: DesignWare Cores
    - ✧ Sub Product: Select 10/100 Ethernet or Gigabit Ethernet.
    - ✧ Version: 3.50a
    - ✧ Subject: Ethernet MAC Universal
- ❖ Send an e-mail message to [support\\_center@synopsys.com](mailto:support_center@synopsys.com).
  - ◆ Include the Product name, Sub Product name, and Version (product release number) in your e-mail so it can be routed correctly.
  - ◆ Provide the following additional information, if applicable:
    - ✧ For environment setup problems, run the debug\_info command in the coreConsultant GUI and include the text file generated.
    - ✧ For configuration failures, include the error messages that appear in the coreConsultant GUI console pane.
    - ✧ For simulation failures, include a text file with your specific configuration. Generate this text file using the coreConsultant GUI's write\_batch\_script command. Also include the log file from the <workspace>/simulation/ directory, the log file for the specific test, and a VPD/VCD waveform dump file.
    - ✧ For synthesis failures, include the log file from the <workspace>/syn/ directory.
- ❖ Telephone your local support center:
  - ◆ United States:  
Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
  - ◆ Canada:  
Call 1-650-584-4200 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
  - ◆ All other countries:  
<http://www.synopsys.com/Support/GlobalSupportCenters/Pages/default.aspx>

# Product Overview

## 1.1 General Product Description

The DWC Ether MAC 10/100/1000 Universal, commonly referred to as GMAC-UNIV in this document, enables a host to transmit and receive data over Ethernet in compliance with the IEEE 802.3-2002 standard. The GMAC-UNIV can have five major configurations: GMAC core only with native interface (GMAC-CORE), GMAC with transaction layer (GMAC-MTL), GMAC with native DMA (GMAC-DMA), and GMAC with AHB-interfaced DMA (GMAC-AHB) or GMAC with AXI-interfaced DMA (GMAC-AXI).

The GMAC-UNIV provides an optimized (with respect to gate count and latency), configurable, flexible product to meet the needs of various applications and customers, and supports a multitude of industry standard interfaces to the PHY, in addition to the default Gigabit Media Independent Interface (GMII)/Media Independent Interface (MII) defined in the IEEE 802.3 specifications. The GMAC-UNIV can be used in number of applications such as switches, network interface cards, etc. The GMAC-AHB is designed to interface to the industry standard AMBA High-Performance Bus (AHB) or the industry standard AMBA Advanced eXtensible Interface Bus (AXI) on the application side.

The GMAC-UNIV is compliant to the following standards:

- ❖ IEEE 802.3-2002 for Ethernet MAC, GMII, and TBI
- ❖ IEEE 1588-2008 standard for precision networked clock synchronization
- ❖ AMBA 2.0 for AHB Master/Slave ports
- ❖ AMBA 3.0 for AXI Master/Slave ports
- ❖ RGMII specification from HP/Marvell for RGMII
- ❖ SGMII specification from Cisco/Marvell for SGMII
- ❖ RMII specification from RMII consortium
- ❖ SMII specification from Cisco for SMII

## Note

Although the naming convention used in this databook and the source code indicate “GMAC,” the databook is also applicable to the 10/100 Universal product. 10/100 Universal users can ignore all descriptions with respect to Gigabit (1000 Mbps) operations. This mainly corresponds to the RGMII, TBI, RTB, and SGMII PHY interfaces, and Gigabit functions such as frame-bursting and carrier-extension.

The following sections are mainly applicable to Gigabit operations and can be ignored by 10/100 Universal users:

- Descriptions of frame bursting and carrier extension in [Sections 3.7.1.3](#) and [3.7.4.1](#).
- [Sections 3.11, 3.14, 3.15](#) and [3.16](#)
- [Sections 4.2.17, 4.2.21, 4.2.22](#), and [4.2.23](#)
- [Sections 5.2.2.17](#) through [5.2.2.23](#)

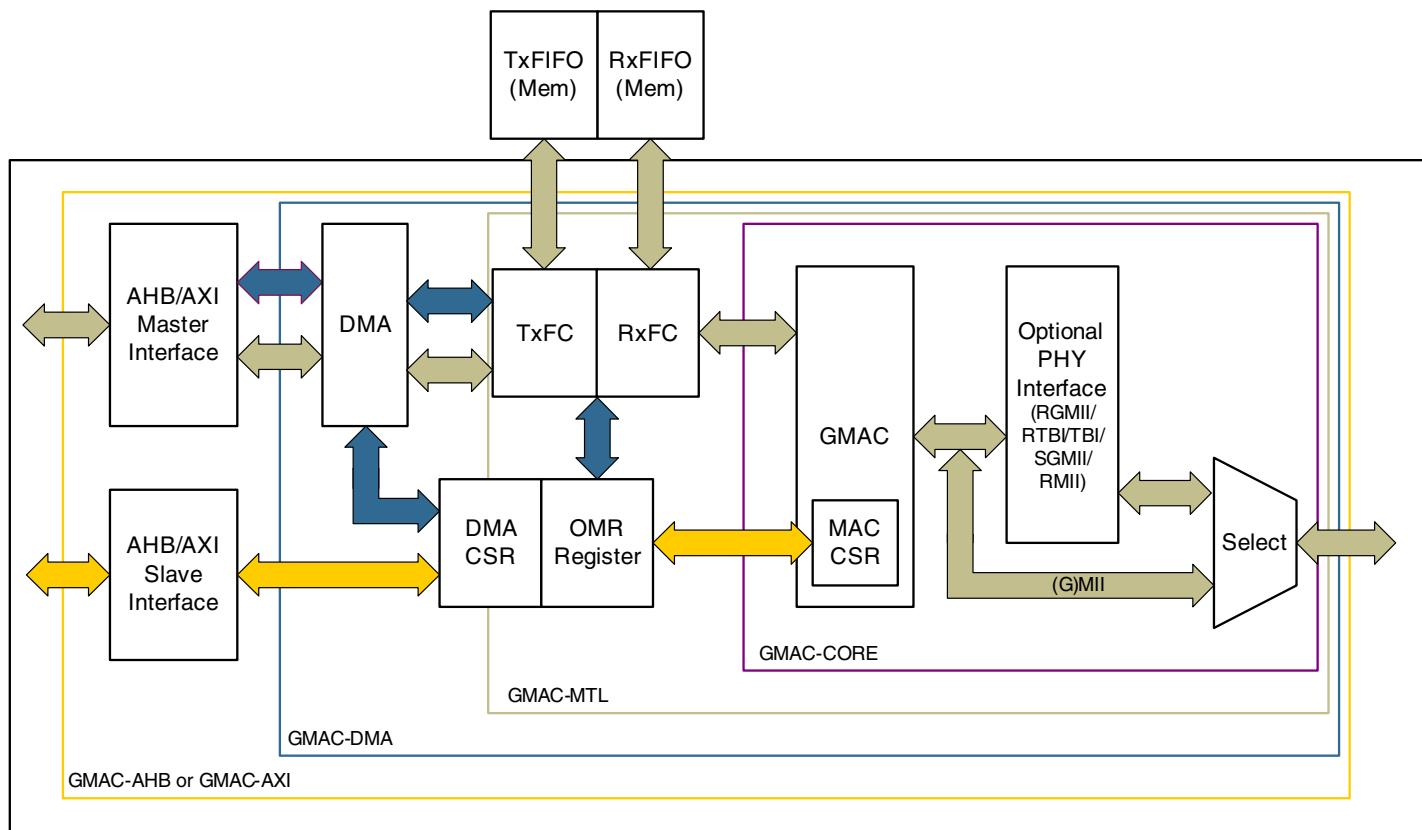
This core’s RTL fully complies with the RTL coding practices as specified in the *Reuse Methodology Manual*, third edition. Refer to the README.txt file for the latest product update information.

## 1.2 System Overview

### 1.2.1 System-Level Block Diagram

A system-level block diagram is shown in [Figure 1-1](#).

**Figure 1-1 GMAC-UNIV Block Diagram**



### 1.2.2 Interfaces

The GMAC-AHB or GMAC-AXI can be selected based on whether the system is designed with an AHB or AXI on-chip bus.

The GMAC-AHB transfers data to system memory through the AHB master interface. This interface can be removed for a non-AHB system, and the subsystem will have a direct native FIFO-type interface. The host CPU uses the default 32-bit AHB Slave interface to access the GMAC subsystem's Control and Status registers (CSRs), when GMAC-AHB is selected. There is an option to select an APB port for CSR access instead of the AHB Slave port. For non-AHB systems, the AHB/APB slave modules can be removed and the native 32-bit Read/Write bus is provided for CPU accesses.

The GMAC-AXI transfers data to system memory through the AXI master interface. The host CPU uses the AXI Slave interface to access the GMAC subsystem's Control and Status registers (CSRs), when GMAC-AXI is selected. There is an option to select an APB port for CSR access instead of the AXI Slave port.

The GMAC-UNIV supports any one or a combination of the following PHY interfaces:

- ❖ Gigabit Media Independent Interface (GMII)/Media Independent Interface (MII) [Default]
- ❖ Reduced GMII (RGMII)
- ❖ Serial GMII (SGMII)
- ❖ Ten Bit Interface (TBI)
- ❖ Reduced MII (RMII)
- ❖ Serial MII (SMII)
- ❖ Reduced TBI (RTBI)

### 1.2.3 Transmit and Receive FIFOs

The Transmit FIFO (Tx FIFO) buffers data read from system memory by the DMA before transmission by the GMAC Core. Similarly, the Receive FIFO (Rx FIFO) stores the Ethernet frames received from the line until they are transferred to system memory by the DMA. These are asynchronous FIFOs, as they also transfer the data between the application clock and the GMAC line clocks.

Both FIFOs are required to be two-ported RAM of configurable depth (35/68/133 bits wide for 32/64/128 data bus widths).

In the GMAC-MTL configuration, an additional two-port RAM (width configurable from 12 to 15 bits) is needed to store the frame lengths of received frames in the Rx FIFO. The depth of this frame-length FIFO depends on the maximum number of frames that can be stored in the Rx FIFO. This frame-length FIFO is optional, and is disabled by default.

## 1.3 Features List

The GMAC-UNIV includes the following features, listed by category.

### 1.3.1 GMAC Core Features

- ❖ Supports 10/100/1000-Mbps data transfer rates with the following PHY interfaces
  - ◆ IEEE 802.3-compliant GMII/MII (default) interface to communicate with an external Gigabit/Fast Ethernet PHY
  - ◆ IEEE 802.3z-compliant Ten-bit Interface (TBI), (optional), with auto-negotiation to communicate with an external PHY
  - ◆ RGMII/RTBI interface (optional) to communicate with an external gigabit PHY
  - ◆ SGMII interface (optional) to communicate with an external gigabit PHY
- ❖ Supports both full-duplex and half-duplex operation
  - ◆ Supports CSMA/CD Protocol for half-duplex operation
  - ◆ Supports packet bursting and frame extension in 1000 Mbps half-duplex operation
  - ◆ Supports IEEE 802.3x flow control for full-duplex operation
  - ◆ Optional forwarding of received pause control frames to the user application in full-duplex operation
  - ◆ Back-pressure support for half-duplex operation
  - ◆ Automatic transmission of zero-quanta pause frame on deassertion of flow control input in full-duplex operation
- ❖ Preamble and start-of-frame data (SFD) insertion in Transmit, and deletion in Receive paths
- ❖ Automatic CRC and pad generation controllable on a per-frame basis
- ❖ Options for Automatic Pad/CRC Stripping on receive frames
- ❖ Programmable frame length to support Standard or Jumbo Ethernet frames with sizes up to 16 KB
- ❖ Programmable InterFrameGap (40-96 bit times in steps of 8)
- ❖ Supports a variety of flexible address filtering modes:
  - ◆ Up to 31 additional 48-bit perfect (DA) address filters with masks for each byte
  - ◆ Up to 31 48-bit SA address comparison check with masks for each byte
  - ◆ 64-bit Hash filter (optional) for multicast and uni-cast (DA) addresses
  - ◆ Option to pass all multicast addressed frames
  - ◆ Promiscuous mode support to pass all frames without any filtering for network monitoring
  - ◆ Passes all incoming packets (as per filter) with a status report
- ❖ Separate 32-bit status returned for transmission and reception packets
- ❖ Supports IEEE 802.1Q VLAN tag detection for reception frames
- ❖ Separate transmission, reception, and control interfaces to the Application
- ❖ Configurable big endian and little endian support for transmission and reception data paths
- ❖ Supports 32/64/128-bit data transfer interface on the system-side

- ❖ Complete network statistics (optional) with RMON/MIB Counters (RFC2819/RFC2665)
- ❖ MDIO Master interface (optional) for PHY device configuration and management
- ❖ Optional module for detection of LAN wake-up frames and AMD Magic Packet frames
- ❖ Optional Receive module for checksum off-load for received IPv4 and TCP packets encapsulated by the Ethernet frame
- ❖ Optional Enhanced Receive module for checking IPv4 header checksum and TCP, UDP, or ICMP checksum encapsulated in IPv4 or IPv6 datagrams.
- ❖ Optional module to support Ethernet frame time stamping as described in IEEE 1588-2008. Sixty-four-bit time stamps are given in each frame's transmit or receive status.

### 1.3.2 DMA Block Features

The DMA block exchanges data between the MTL block and host memory. A set of registers (DMA CSR) to control DMA operation is accessible by the host.

DMA features include:

- ❖ 32/64/128-bit data transfers
- ❖ Single-channel Transmit and Receive engines
- ❖ Fully synchronous design operating on a single system clock (except for CSR module, when a separate CSR clock is configured)
- ❖ Optimization for packet-oriented DMA transfers with frame delimiters
- ❖ Byte-aligned addressing for data buffer support
- ❖ Dual-buffer (ring) or linked-list (chained) descriptor chaining
- ❖ Descriptor architecture, allowing large blocks of data transfer with minimum CPU intervention; each descriptor can transfer up to 8 KB of data
- ❖ Comprehensive status reporting for normal operation and transfers with errors
- ❖ Individual programmable burst size for Transmit and Receive DMA Engines for optimal host bus utilization
- ❖ Programmable interrupt options for different operational conditions
- ❖ Per-frame Transmit/Receive complete interrupt control
- ❖ Round-robin or fixed-priority arbitration between Receive and Transmit engines
- ❖ Start/Stop modes
- ❖ Separate ports for host CSR access and host data interface

### 1.3.3 Transaction Layer (MTL) Features

The MTL block consists of two sets of FIFOs: a Transmit FIFO with programmable threshold capability, and a Receive FIFO with a configurable threshold (default of 64 bytes).

MTL features include:

- ❖ 32-, 64-, or 128-bit Transaction Layer block providing a bridge between the application and the GMAC-CORE
- ❖ Single-channel Transmit and Receive engines

- ❖ Data transfers executed using simple FIFO-protocol
- ❖ Synchronization for all clocks in the design (Transmit, Receive and system clocks)
- ❖ Optimization for packet-oriented transfers with frame delimiters
- ❖ Four Separate ports for system-side and GMAC-CORE-side transmission and reception
- ❖ Two 2-port RAM-based asynchronous FIFOs with synchronous/asynchronous Read and Write operation with respect to the Read and Write clocks (one for transmission and one for reception)
- ❖ FIFO instantiation outside the top-level module to facilitate memory testing/instantiation
- ❖ Supports 128-, 256-, or 512-byte, or 1-, 2-, 4-, 8-, 16-, or 32-KB receive FIFO depths on reception.
- ❖ Optional interface to indicate the length of a received frame at the top of the MTL Rx FIFO in the GMAC-MTL configuration
- ❖ Programmable burst-length support for starting a burst up to half the size of the MTL Rx and Tx FIFO in the GMAC-MTL configuration
- ❖ Receive Status vectors inserted into the Receive FIFO after the EOF transfer enables multiple-frame storage in the Receive FIFO without requiring another FIFO to store those frames' Receive Status.
- ❖ Configurable Receive FIFO threshold (default fixed at 64 bytes) in Cut-Through mode
- ❖ Option to filter all error frames on reception and not forward them to the application in Store-and-Forward mode
- ❖ Option to forward under-sized good frames
- ❖ Supports statistics by generating pulses for frames dropped or corrupted (due to overflow) in the Receive FIFO
- ❖ Supports 256- or 512-byte, or 1-, 2-, 4-, 8-, or 16-KB FIFO depth on transmission
- ❖ Supports Store and Forward mechanism for transmission to the GMAC core
- ❖ Supports threshold control for transmit buffer management
- ❖ Supports configurable number of frames to be stored in FIFO at any time. The default is 2 frames (fixed) with internal DMA, and up to 8 frames in GMAC-MTL configuration.
- ❖ Automatic generation of PAUSE frame control or backpressure signal to the GMAC core based on Receive FIFO-fill (threshold configurable) level.
- ❖ Handles automatic retransmission of Collision frames for transmission
- ❖ Discards frames on late collision, excessive collisions, excessive deferral and underrun conditions
- ❖ Software control to flush Tx FIFO
- ❖ Data FIFO RAM chip-select disabled when inactive, to reduce power consumption
- ❖ Optional module to calculate and insert IPv4 header checksum and TCP, UDP, or ICMP checksum in frames transmitted in Store-and-Forward mode.

#### 1.3.4 AMBA AHB Interface Features

- ❖ Interfaces with the application via the AHB
- ❖ AHB Slave interface (32-, 64-, or 128-bit) for CSR access, in which only 32-bit or less (byte, half-word) accesses are possible
- ❖ Option for a 32-bit APB port for CSR access instead of an AHB Slave port
- ❖ Supports 32-, 64-, or 128-bit data on the AHB Master port

- ❖ Supports Split, Retry, and Error AHB responses in the AHB Master interface
- ❖ Does not generate Split, Retry, or Error responses in the AHB Slave interface (compatible with AHB-Lite)
- ❖ Configurable for Little- or Big-Endian modes
- ❖ Supports all AHB burst types in the AHB Slave Interface
- ❖ Software can select the type of AHB burst (fixed or indefinite burst) in the AHB Master interface.
- ❖ Option to select address-aligned bursts from AHB master port

### 1.3.5 AMBA AXI Interface Features

- ❖ Interfaces with the application via the AXI
- ❖ AXI Slave interface (32-, 64-, or 128-bit) for CSR access
- ❖ Supports 32-, 64-, or 128-bit data on the AXI Master port
- ❖ Supports 32-bit address width on AXI Master and Slave interfaces
- ❖ Supports OKAY, SLVERR and DECERR responses on the AXI Master
- ❖ Support Little Endian mode only for AXI Master and Slave Interface
- ❖ Supports all AXI burst types on the AXI Slave interface
- ❖ Software can select the type of AXI burst (fixed and variable length burst) in the AXI Master interface. Option to select extended length fixed bursts of 32, 64, 128, and 256
- ❖ Option to select select address-aligned bursts from AXI Master interface
- ❖ Make use of posted writes from AXI Master interface to maximize the bus utilization
- ❖ Supports AXI low-power interface
- ❖ Handles the AXI 4K boundary burst splitting on AXI Master interface
- ❖ Supports up to four read and write outstanding transactions on AXI Master interfaces
- ❖ Does not support burst interleaving and reordering on AXI Master or Slave interface
- ❖ Does not support Atomic, Locked, Cache and Protected accesses on AXI Master and Slave interfaces
- ❖ Does not support Exclusive access on Slave interface

### 1.3.6 Monitoring, Test, and Debugging Support Features

- ❖ Supports internal loopback on the GMII/MII for debugging
- ❖ DMA states (Tx and Rx) given as status bits
- ❖ Debug status register that gives status of FSMs in Transmit and Receive data-paths and FIFO fill-levels.
- ❖ Application Abort status bits
- ❖ MMC (RMON) module in the GMAC core
- ❖ Current Tx/Rx Buffer pointer as status registers
- ❖ Current Tx/Rx Descriptor pointer as status registers

## 1.4 Configurability

The DWC Ether MAC 10/100/1000 Universal can be configured to any of the following major architectures using CoreConsultant:

- ❖ GMAC Subsystem with DMA and AHB interface (default)
- ❖ GMAC Subsystem with DMA and AXI interface
- ❖ GMAC Subsystem with DMA and native interface (without AHB and AXI)
- ❖ GMAC Subsystem with transaction (FIFO) layer only (without DMA) and with native FIFO interface
- ❖ GMAC core with native FIFO interface (without DMA, FIFO layer, AHB and AXI)

In addition, the PHY can be configured to have any of the following interfaces (including an option to select them at reset), if multiple interfaces are configured:

- ❖ GMII/MII interface
- ❖ TBI interface
- ❖ RGMII/RTBI interface
- ❖ SGMII interface
- ❖ RMII interface (for 10/100 Mbps operations only)
- ❖ SMII interface (for 10/100 Mbps operations only)

The following features are configurable (using coreConsultant) to provide an optimized design with respect to area and timing.

- ❖ 10/100 Mbps operation only
- ❖ 1000 Mbps operation only
- ❖ Full-duplex operation only
- ❖ Depth of Receive FIFO in transaction layer
- ❖ Depth of Transmit FIFO in transaction layer
- ❖ Bus width of AHB/AXI (or application) interface to 32/64/128
- ❖ Address filtering option (Hash filter function)
- ❖ Configurable number of MAC Address registers (up to 32) for filtering
- ❖ RMON Counters (MMC) with individual registers selection
- ❖ Power Management Module (PMT) with Remote Wake-up and Magic Packet frame processing options
- ❖ TCP, UDP, or ICMP over IP Checksum Offload engine
- ❖ IEEE 1588 Ethernet frame time-stamping support
- ❖ Endianness for all application side data paths (except in GMAC-AXI)
- ❖ Station Management Agent (SMA) - MDIO module
- ❖ APB port instead of AHB/AXI Slave port
- ❖ Separate application clocks option for AHB/APB slave port

## 1.5 Deliverables

The GMAC-UNIV is packaged as a coreKit (similar to most Synopsys DesignWare IP cores). The GMAC-UNIV requires the Synopsys coreConsultant tool (described in detail in “[Building and Verifying Your Core](#)” on page 27) to install, unpack and configure the core.

The license file required to install, unpack and configure the coreKit is delivered separately.

### 1.5.1 coreKit

The GMAC-UNIV coreKit package includes the RTL source code, the Verification Environment with sample test cases to test the configured RTL, scripts such as Synopsys DC, TetraMax, and scripts for implementation. coreConsultant is the software tool Synopsys provides to simplify GMAC-UNIV installation, configuration, synthesis, and simulation. coreConsultant features and tasks include:

- ❖ A graphical user interface (GUI) to guide you through GMAC-UNIV design flow activities. All activities may also be performed through a command line interface using batch scripts generated from the coreConsultant.
- ❖ An interactive parameter selection to configure the Verilog RTL source code. coreConsultant checks for consistent settings and automatically derives any dependent parameters from your choices.
- ❖ Synthesis script generation for Synopsys DC and DC-FPGA
- ❖ Verification testbenches and test configurations derived from your parameter choices
  - ◆ Verilog-based system-level testbench environment and tests for GMAC-AHB configuration. The Vera-based Designware AMBA VIP is used as the verification model for the AHB bus in this testbench. For more information on DesignWare AMBA VIP, see the VIP documentation provided in the coreKit. For online VIP documentation, refer to DWIP\_install/doc/README.TXT.
  - ◆ Verilog-based testbench environment and test cases for GMAC-DMA, GMAC-MTL, and GMAC-CORE configurations.
  - ◆ Verilog-based testbench environment for GMAC-AHB and GMAC-AXI configuration using DW AMBA and DW Ethernet VIPs. You can use this testbench, along with the provided test cases, as a reference to build a verification environment for an SoC in which GMAC-AHB or GMAC-AXI integration can be tested.
  - ◆ Support for creating gate-level DUT models using a GTECH library
  - ◆ Verification scripts for running with three simulators: Synopsys VCS, ModelSim, and NC-Verilog simulators
- ❖ Useful example code, scripts and documents, including:
  - ◆ A software driver for NIC (PCI) card applications
  - ◆ Implementation flow guidelines and scripts
  - ◆ A configuration for attaching a PVCI interface to GMAC-DMA
  - ◆ C code for calculating CRC-32 and CRC-16



**Note** The software driver can be located after creating the workspace. Please see Table A-1 in the Appendix A Workspace Directory and Files section for the software driver location.

### 1.5.2 Licensing Features

The GMAC-UNIV license file includes

- ❖ One DW license for installation and configuration of the GMAC-UNIV
- ❖ Designware Ethernet VIP and Designware AMBA VIP licenses to run testcases for GMAC-AHB and GMAC-AXI configurations.

**Note**

New features such as the AXI system interface and the IEEE 1588-2008 Advanced Time Stamping support are available only when the user has the license for the DWC Ether MAC 10/100/1000 Universal product. With this license, you can still generate a 10/100 Mbps core with the above mentioned features.

In addition to the shipped licenses noted above, the following licenses are required:

For Synopsys synthesis tools:

- ❖ Design Compiler
- ❖ HDL-Compiler

For simulation tools:

- ❖ Synopsys VCS Simulator, or
- ❖ Any other simulator such as ModelSim-Verilog or NC-Verilog

# 2

## Building and Verifying Your Core

### 2.1 Setup Requirements

Before performing the procedures in this chapter, make sure that you have installed the DWC Ether MAC 10/100/1000 Universal database and set up your environment. In particular, it is important that you use the required versions of coreConsultant, Design Compiler, Vera, and your preferred simulation tool, and that all associated paths and environment variables are set correctly.



In this chapter, the DWC Ether MAC 10/100/1000 Universal product is offered as an example. These descriptions apply to the DWC Ether MAC 10/100 Universal product as well.



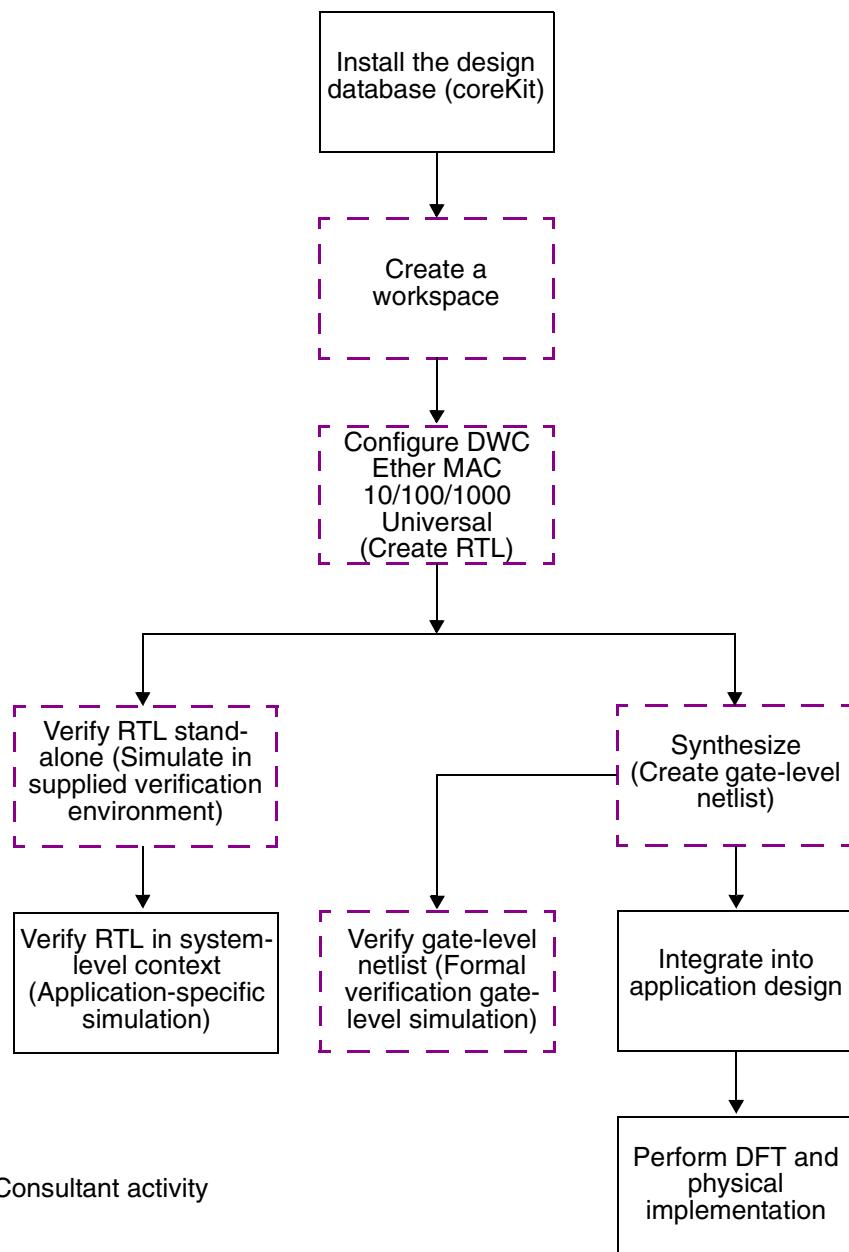
IP core products are subject to frequent enhancements and fixes. To ensure you have the latest Ethernet MAC Universal release, point your browser to [www.designware.com](http://www.designware.com) and enter Ethernet MAC Universal in the “Search for IP” box.

### 2.2 Basic Design Flow

coreConsultant is your user interface for configuration, standalone verification, and synthesis of the DWC Ether MAC 10/100/1000 Universal. [Figure 2-1](#) shows the basic sequence of tasks.



In coreConsultant terminology, the DWC Ether MAC 10/100/1000 Universal design database that you installed—the deliverables contained in the top-level DWC Ether MAC 10/100/1000 Universal directory—is referred to as the coreKit. A workspace is a working directory in which you work with a configurable copy of the coreKit. Though the coreKit is named DWC\_gmac, the top-level RTL module is named DWC\_gmac\_top and the top-level file is DWC\_gmac\_top.v.

**Figure 2-1 Design Flow**

## 2.3 Creating a Workspace

A workspace is a local copy of your DWC Ether MAC 10/100/1000 Universal coreKit installation in which you can configure, verify, and synthesize your own implementation of the DWC Ether MAC 10/100/1000 Universal. After you install the coreKit, you must create a workspace to begin working. You can create several workspaces so that you can experiment with different design alternatives, as shown in [Figure 2-2](#).

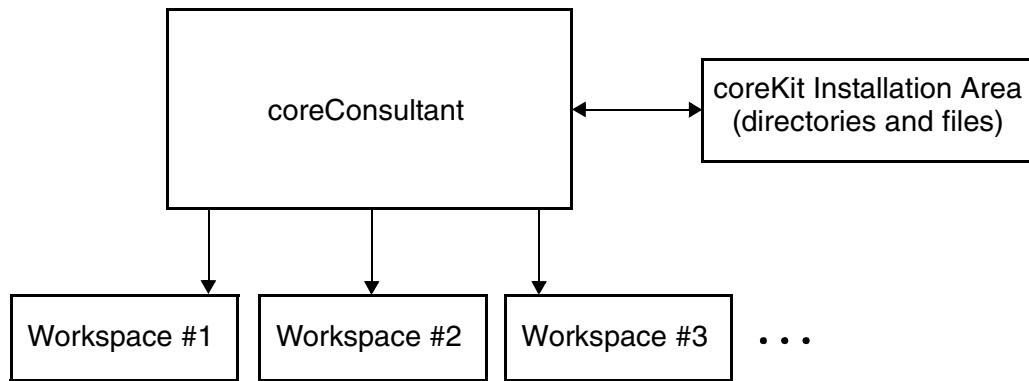
To create a workspace:

1. Invoke coreConsultant:

```
% coreConsultant
```

2. Select File > New Workspace in the coreConsultant console, then enter the requested information in the New Workspace dialog. For more information about answering the New Workspace options, refer to the coreConsultant online help.

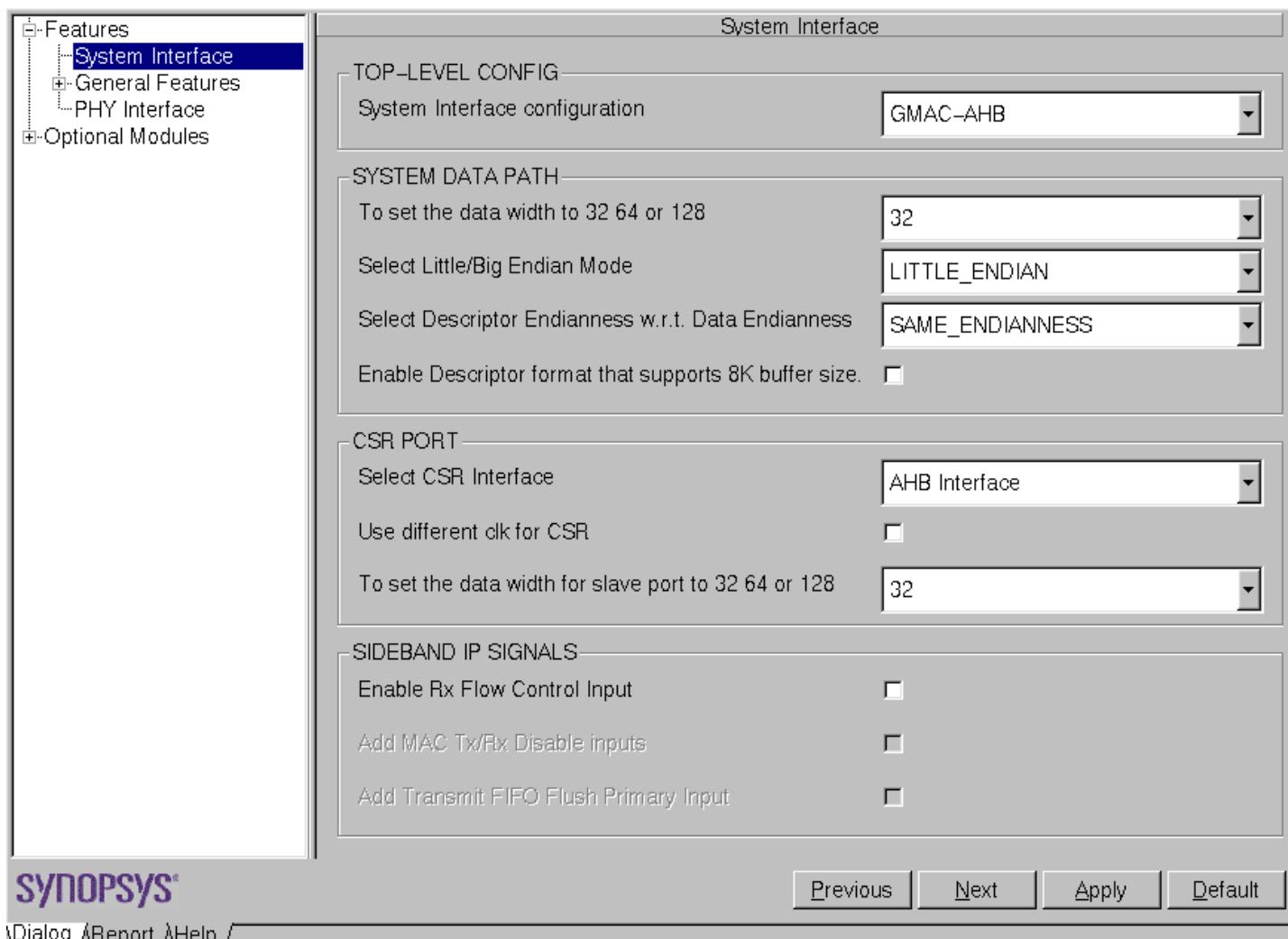
**Figure 2-2 coreKit and Workspaces**



At the completion of workspace creation, coreConsultant displays the Activity List ([Figure 2-3](#)). For DWC Ether MAC 10/100/1000 Universal, the first activity to execute is Specify Configuration. Go to ["Configuration: Creating the RTL"](#) on page 30 to continue.



**Note** Use the Set Design Prefix activity if you plan to instantiate DWC Ether MAC 10/100/1000 Universal more than once in your design. The default Set Design Prefix state is completed because it is not a required activity for most DWC Ether MAC 10/100/1000 Universal users.

**Figure 2-3 coreConsultant Specify Configuration Dialog**

## 2.4 Configuration: Creating the RTL

To configure DWC Ether MAC 10/100/1000 Universal interactively, go the Specify Configuration dialog and select your configuration options. There are options available to enable or disable certain features, select memory sizes and data bus width, and select the application and PHY interfaces. You can use the default values for an initial simulation and synthesis trial. Otherwise, you must select or enter the values required for your design.

Refer to “[Parameters](#) on page 309” for detailed descriptions of all configuration options. After you have specified values for all configuration options, click **Apply** to generate the configured RTL code (with `DWC_gmac_top.v` as the top-level file). `coreConsultant` then checks your parameter values, generates configured RTL code, and displays a configuration report.

You can reconfigure DWC Ether MAC 10/100/1000 Universal (create new RTL) at any time. If you do so, you will need to re-complete any downstream activities.

## 2.5 Synthesis: Creating the Gate-Level Netlist

coreConsultant is your interface to Synopsys synthesis tools for synthesizing DWC Ether MAC 10/100/1000 Universal. Follow the sequence of activities under the Create Gate-Level Netlist group in the coreConsultant GUI to choose your synthesis options, run the synthesis job, and view the results. For more information about specific dialog items, refer to the coreConsultant online help.

## 2.6 Simulation: Verifying DWC Ether MAC 10/100/1000 Universal

You can perform two types of verification from within coreConsultant:

- ❖ **Formal Verification:** Runs a formal verification (using the Synopsys Formality tool) to compare the configured RTL with your post-synthesis netlist.
  - ◆ To run a formal verification from coreConsultant, ensure that SVF is enabled and that a proper source and target are selected. By default, the latest synthesis database is taken for comparison.
  - ◆ To run a formal verification outside coreConsultant, look for the formal verification script, named xxxx\_fm.tcl (incr2\_fm.tcl, for example), in the <workspace>/syn/ directory. This script can run in the Formality shell as a standalone .
- ❖ **Simulate:** Simulates DWC Ether MAC 10/100/1000 Universal in the supplied test environment. You can simulate your configured RTL version of DWC Ether MAC 10/100/1000 Universal or your synthesized gate-level netlist.

For RTL simulation, if you are using an evaluation (encrypted RTL) version of DWC Ether MAC 10/100/1000 Universal, you may need to create a GTECH model for simulation:

- ◆ If you are using the Synopsys VCS simulator with an evaluation version of DWC Ether MAC 10/100/1000 Universal, you can set up and run your simulation directly on the encrypted RTL.
- ◆ If you are using another simulator with an evaluation version of DWC Ether MAC 10/100/1000 Universal, you must first generate a GTECH netlist of the RTL to use in your simulation.

If you are using the paid (source-code) version of DWC Ether MAC 10/100/1000 Universal, you can set up and run your simulation directly on the RTL using any of the supported simulators.



**Note** When using modelsim, make sure the LD\_LIBRARY\_PATH is pointed to the SystemC gcc 3.2.3 that contains the library libstdc++.so.5.

### 2.6.1 Generating a GTECH Model

You only need to generate a GTECH simulation model if you are using a non-Synopsys simulator with an encrypted RTL version of DWC Ether MAC 10/100/1000 Universal. To generate a GTECH simulation model:

1. Select your options for the GTECH synthesis job. For information about individual options, refer to the coreConsultant online help.
2. Click Apply to launch the GTECH synthesis job.
3. Go to the Generate GTECH Model activity.
4. When the GTECH model activity is complete, you can select the GTECH simulation model as the “Design view to simulate” in the View page of the Simulate dialog.

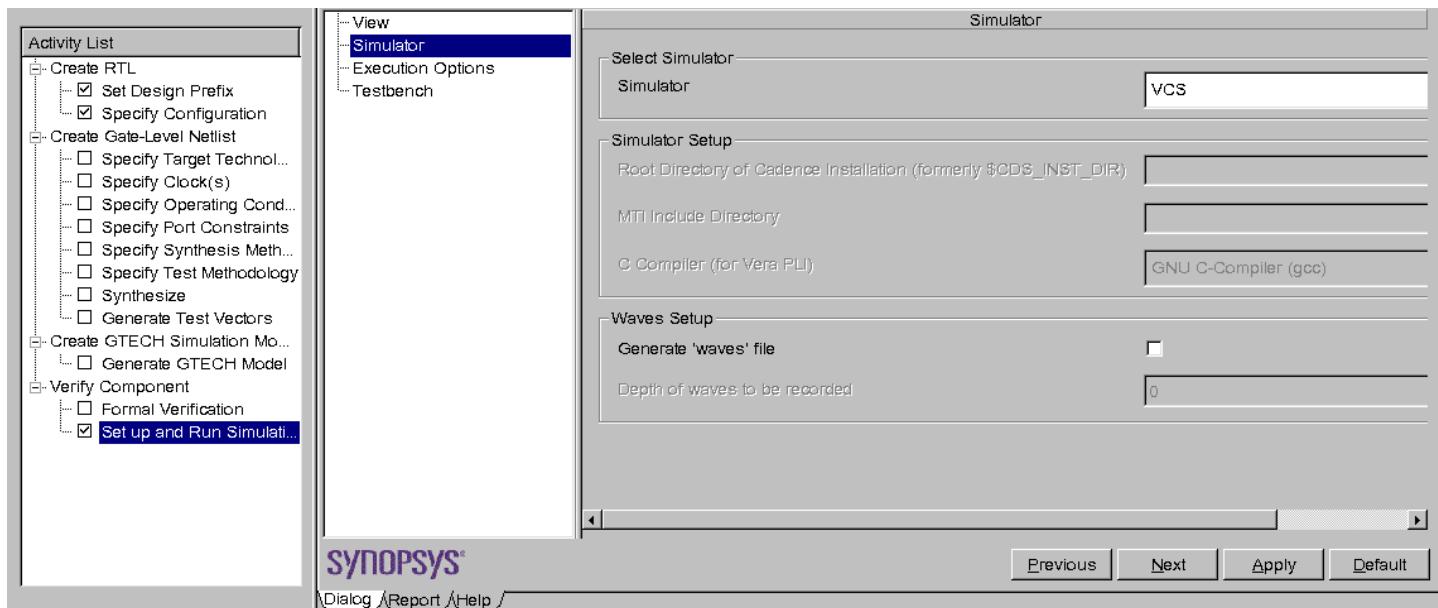
## 2.6.2 Verifying DWC Ether MAC 10/100/1000 Universal

The Set up and Run Simulations activity (Figure 2-4) is where you select your simulation options and run the simulation. When the simulation is complete, the results are available in the Report tab of this activity.



**Note** Clicking Apply in any page of the Simulate dialog applies all of your simulation options (on all pages) and starts the simulation. Do not click Apply until you have selected all of your simulation options on all pages of the dialog.

**Figure 2-4 Set Up and Run Simulations Dialog**



To run the simulation through coreConsultant, go to the Set up and Run Simulations dialog (Figure 2-4) and select from the following simulation options:

- ❖ **Simulator:** Options to select and set up your simulation tool and to generate waveforms. These options are common to most DesignWare coreKits. Use coreConsultant's online help to learn more about individual options.
- ❖ **View:** Options that select which view of DWC Ether MAC 10/100/1000 Universal you want to simulate. Choose RTL to simulate your configured RTL model of DWC Ether MAC 10/100/1000 Universal or GTECH to simulate the generated GTECH model. To simulate your technology-specific gate-level netlist, refer to the README file in the <config#>/resources/gatesim/ directory for detailed instructions to run simulation with gate-level netlist.
- ❖ **Execution Options:** Options related to simulation launch and execution. These options are common to most DesignWare coreKits. Use the coreConsultant online help for more information about individual options.
- ❖ **Testbench:** A list of tests that you can select to execute or not execute. By default, all tests applicable to the selected top-level configuration are selected. Test suites that do not apply to the selected configuration are grayed out. You can run a limited subset of the test suite for sanity check by selecting the Run User List Tests option.



After selecting your simulation options and tests, click Apply to launch the simulation. If you select the Do Not Launch Simulation option, coreConsultant does not invoke the simulation; go to your <workspace>/sim/ directory and execute run.scr to run the simulation.

When the simulation is complete, use the Report tab to view the simulation log.

If any test fails, contact [support\\_center@synopsys.com](mailto:support_center@synopsys.com) after checking the Known Problems list in the release notes. There is no known reason for any test to fail if you are using a supported simulation tool.



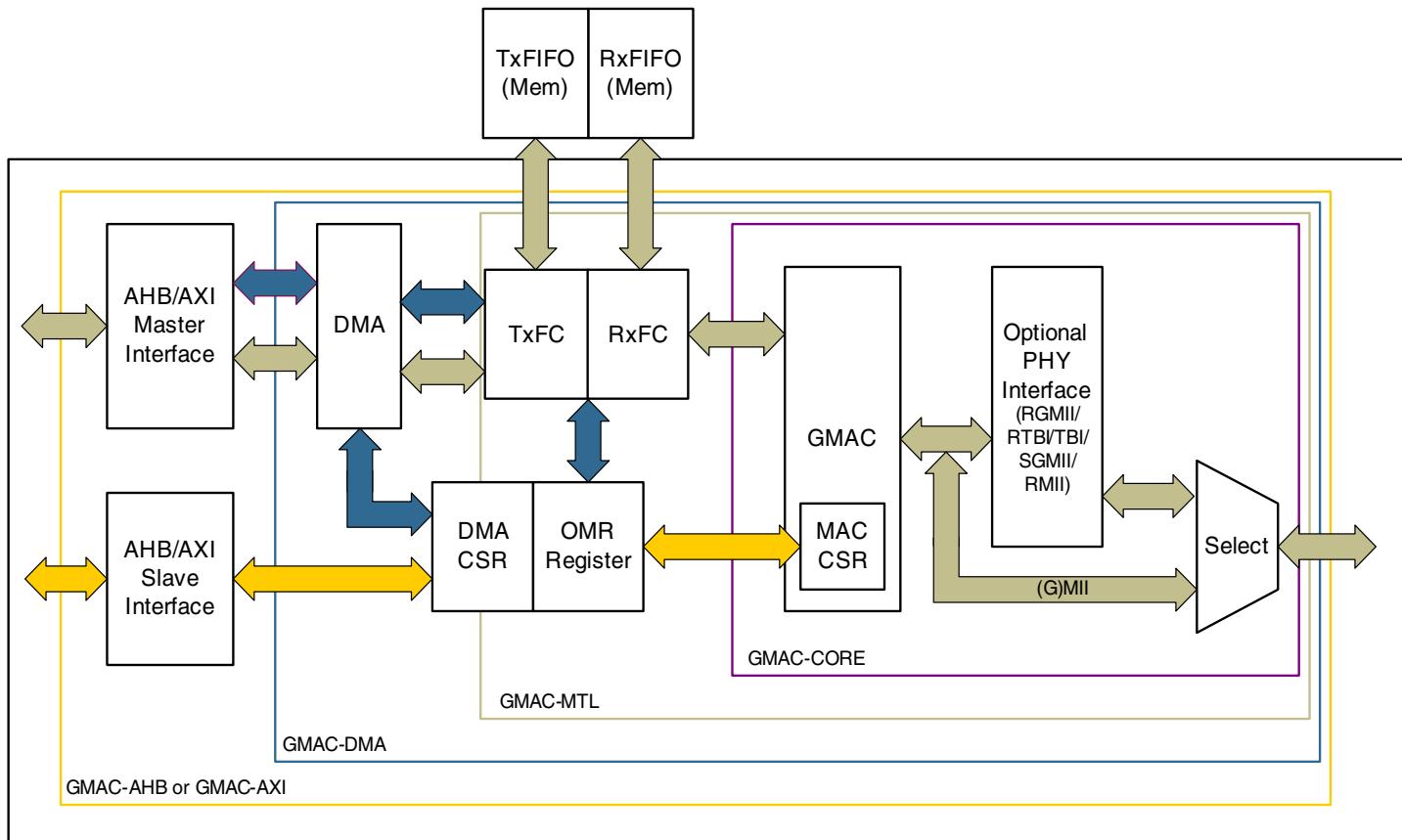
## 3

## Architecture

### 3.1 Introduction

The GMAC-UNIV is a highly configurable product with multiple interfaces to the system side or the PHY side. Major modules and functions are only selected when required for your design, ensuring that your design is optimized for area. A block diagram of the GMAC-UNIV's five major system configurations is provided in [Figure 3-1](#).

**Figure 3-1 GMAC-UNIV Block Diagram**



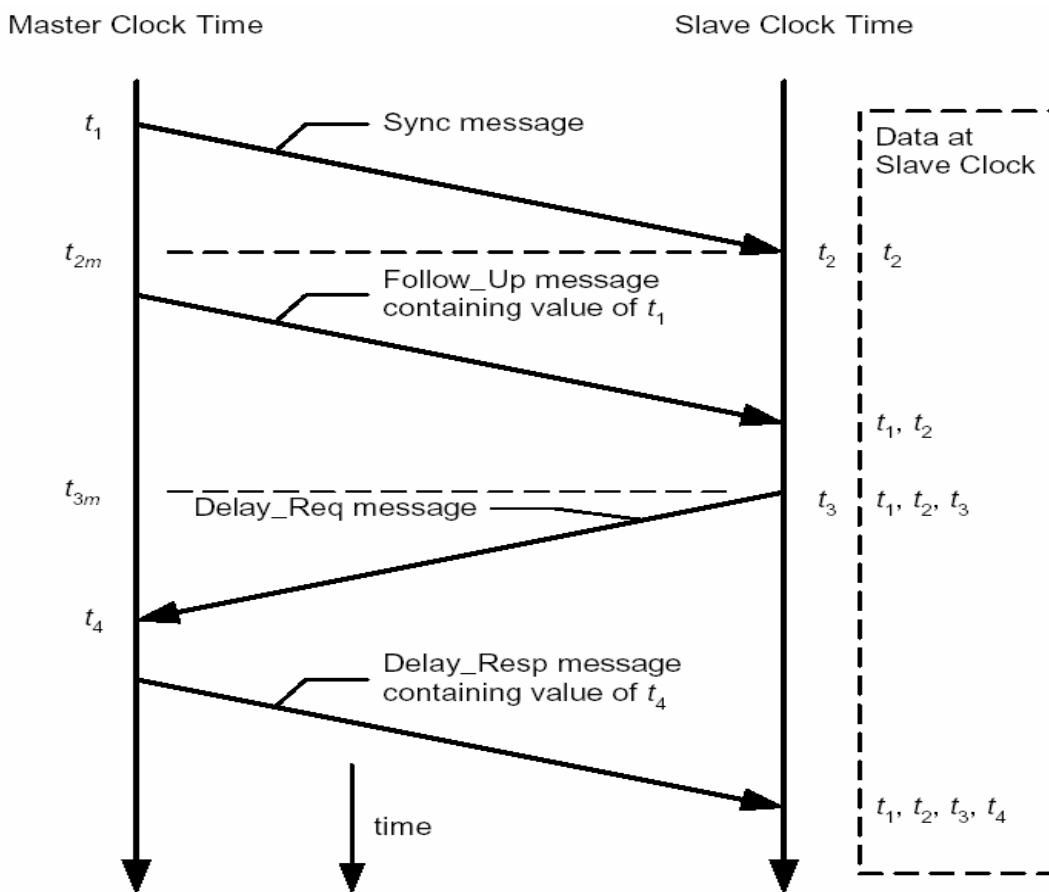
The following sections describe the functionality of the major blocks and timing behavior of the interfaces.

- ❖ [Section 3.2](#) provides an overview of precision network time synchronization using the Precision Time Protocol established under the IEEE 1588 standard.
- ❖ [Section 3.3](#) describes the functionality and timing behavior of the AHB interfaces in GMAC-AHB configuration
- ❖ [Section 3.4](#) describes the functionality and timing behavior of the AXI interfaces in GMAC-AXI configuration
- ❖ [Section 3.5](#) describes the functioning of the DMA and is applicable for both GMAC-DMA and GMAC-AHB configuration. It also describes the timing behavior of the native system-side interface in GMAC-DMA configuration.
- ❖ [Section 3.6](#) explains the functioning of the MAC Transaction Layer (MTL) module which controls the Receive and Transmit FIFO memories. This module is required in all configurations except GMAC\_CORE. The timing behavior of the system-side ATI and ARI interfaces in GMAC-MTL configuration is also described in this section.
- ❖ [Section 3.7](#) describes the functionality of the (G)MAC core transmitter and receiver. If you select the GMAC-CORE configuration, refer to the timing diagrams in this section to understand the system-side interface.
- ❖ [Sections 3.8, 3.9, and 3.10](#) describe the optional RMON, Power Management, and SMA interface modules.
- ❖ [Sections 3.11 through 3.16](#) describe the functionality and behavior of the various PHY interfaces available for your design. [Section 3.17](#) explains the architecture on the line-side when multiple PHY interfaces are selected.
- ❖ [Section 3.18](#) describes the Interrupt signal structure in the (G)MAC core.

## 3.2 IEEE 1588-2002 Overview

The IEEE 1588 standard defines a protocol enabling precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing, and distributed objects. The protocol applies to systems communicating by local area networks supporting multicast messaging, including (but not limited to) Ethernet. This protocol enables heterogeneous systems that include clocks of varying inherent precision, resolution, and stability to synchronize. The protocol supports system-wide synchronization accuracy in the sub-microsecond range with minimal network and local clock computing resources.

The messaged-based protocol, named Precision Time Protocol (PTP), is transported over UDP/IP. The system or network is classified into Master and Slave nodes for distributing the timing/clock information. The protocol's technique for synchronizing a slave node to a master node by exchanging PTP messages is depicted in [Figure 3-2](#).

**Figure 3-2 Networked Time Synchronization**

1. The master broadcasts PTP Sync messages to all its nodes. The Sync message contains the master's reference time information. The time at which this message leaves the master's system is  $t_1$  and must, for Ethernet ports, be captured at the GMII/MII.
2. A slave receives the Sync message and also captures the exact time,  $t_2$ , using its timing reference.
3. The master then sends the slave a Follow\_up message, which contains  $t_1$  information for later use.
4. The slave sends the master a Delay\_Req message, noting the exact time,  $t_3$ , at which this frame leaves the GMII/MII.
5. The master receives this message, capturing the exact time,  $t_4$ , at which it enters its system.
6. The master sends the  $t_4$  information to the slave in the Delay\_Resp message.
7. The slave uses the four values of  $t_1$ ,  $t_2$ ,  $t_3$ , and  $t_4$  to synchronize its local timing reference to the master's timing reference.

Most of the protocol implementation occurs in the software, above the UDP layer. As described above, however, hardware support is required to capture the exact time when specific PTP packets enter or leave the Ethernet port at the GMII/MII. This timing information must be captured and returned to the software for the proper implementation of PTP with high accuracy.

### 3.2.1 Reference Timing Source

To get a snapshot of the time, the core requires a reference time in 64-bit format (split into two 32-bit channels, with the upper 32 bits providing time in seconds, and the lower 32 bits indicating time in nanoseconds) as defined in the IEEE 1588 specification. The GMAC-UNIV provides two options for using the reference timing source in a node:

#### External Time Stamp Input Option

This option takes a 64-bit timing reference, along with its clock, required for synchronization into the Ethernet MAC clock domain, splits it into two 32-bit signals (the upper 32 bits provide the time in seconds; the lower 32 bits indicate nanoseconds), using the timing reference to time-stamp required frames.

#### Internal Reference Time Option

This option takes only the reference clock input and uses it to generate the Reference time (also called the System Time) internally and use it to capture time stamps. The generation, update, and modification of the System Time are described in “[System Time Register Module](#)” on page 128.

You can choose either of these options as described in “[Parameters](#)” on page 309.

### 3.2.2 Transmit Path Functions

When a frame’s SFD is output on the GMII/MII, a time stamp is captured. Frames for which capturing a time stamp is required are controllable on a per-frame basis. In other words, each transmit frame can be marked to indicate whether or not a time stamp must be captured for that frame.

No snooping or processing of the transmitted frames is performed to identify PTP frames. Framewise control is exercised either through control bits in the transmit descriptor (in GMAC-AHB/AI and GMAC-DMA configurations, as described in “[Descriptor Format With IEEE 1588 Time Stamping Enabled](#)” on page 352) or as input signals (in GMAC-MTL and GMAC-CORE configurations, as described in “[IEEE 1588 Time Stamp Signals \(Optional\)](#)” on page 243).

Captured time stamps are returned to the application in a manner similar to that in which status is provided for frames. A separate 64-bit bus is used for this purpose in the GMAC-CORE and GMAC-MTL configurations. The time stamp, along with the Transmit status of the frame, are given on this bus. In GMAC-AHB/AI and GMAC-DMA configurations the time stamp is returned to software inside the corresponding transmit descriptor, thus connecting the time stamp automatically to the specific PTP frame. The 64-bit time stamp information is written back to the TDES2 and TDES3 fields, with TDES2 holding the time stamp’s 32 least significant bits, except as described in “[Transmit Time Stamp Field](#)” on page 354.



**Note** When the alternate (enhanced) descriptor is selected, the 64-bit time-stamp is written in TDES6 and TDES7, respectively.

### 3.2.3 Receive Path Functions

When the IEEE 1588 time-stamping feature is selected and enabled, the Ethernet MAC captures the time stamp of all frames received on the GMII/MII. No snooping or processing of the received frames is performed to identify PTP frames in the default mode (Advanced Time Stamp feature is not selected).

In GMAC-CORE configuration, the time stamp and the corresponding status on the MRI are given when EOF data is presented.

In GMAC-MTL configuration, the core provides the time stamp on the data bus (`ari_data_o`) after the EOF data has been transferred. The core sends a separate signal (`ari_timestamp_val_o`) to validate the time stamp and to indicate the time stamp's availability. Once the time stamp is transferred, the Status Valid signal (`ari_rxstatus_val_o`) is asserted as soon as status data is present on the data bus (`ari_data_o`).

In GMAC-AHB/AXI and GMAC-DMA configurations, the core returns the time-stamp to the software in the corresponding receive descriptor. The 64-bit time stamp information is written back to the RDES2 and RDES3 fields, with RDES2 holding the time stamp's 32 least significant bits, except as mentioned in “[Receive Time Stamp](#)” on page 352. The time stamp is only written to the receive descriptor for which the Last Descriptor status field has been set to 1 (the EOF marker). When the time stamp is not available (for example, due to an RxFIFO overflow) an all-ones pattern is written to the descriptors (RDES2 and RDES3), indicating that time stamp is not correct. If the software uses a control register bit to disable time stamping, the DMA does not alter RDES2 or RDES3.



**Note** When the alternate (enhanced) descriptor is selected, the 64-bit time-stamp is written in RDES6 and RDES7, respectively. RDES0[7] will indicate whether the time-stamp is updated in RDES6/7 or not.

### 3.2.4 Time Stamp Error Margin

According to the IEEE 1588 specifications, the time stamp must be captured at the SFD of transmitted and received frames at the GMII/MII interface. Since the reference timing source (the PTP clock, `clk_ptp_ref_i`) is different from the GMII/MII clocks, a small error margin is introduced, due to the transfer of information across asynchronous clock domains.

In the transmit path, the captured and reported time stamp has a maximum error margin of 2 PTP clocks. In other words, the captured time stamp has the value of the reference time source given within 2 clocks after the SFD has been transmitted on the GMII.

Similarly, on the receive path, the error margin is 3 GMII/MII clocks, plus up to 2 PTP clocks. You can ignore the error margin due to the 3 GMII/MII clocks by assuming that this constant delay is present in the system (or link) before the SFD data reaches the GMAC's GMII/MII interface.

### 3.2.5 Frequency Range of Reference Timing Clock

Because asynchronous logic is in place for time stamp information transfers across clock domain, a minimum delay is required between two consecutive time stamp captures. This delay is 3 clock cycles of both the GMII/MII and PTP clocks. If the gap is shorter, the GMAC does not take a time stamp snapshot for the second frame.

The maximum PTP clock frequency is limited by the maximum resolution of the reference time (1 ns resulting in 1 GHz) and the timing constraints achievable for logic operating on the PTP clock. Another factor to consider is that the resolution, or granularity, of the reference time source determines the accuracy of the synchronization. Hence, a higher PTP clock frequency gives better system performance. The minimum PTP clock frequency depends on the time required between two consecutive SFD bytes. Because the GMII/MII clock frequency is fixed by IEEE specification, the minimum PTP clock frequency required for proper operation depends on the core's operating mode and operating speed.

For example, in 100-Mbps full-duplex operation, the minimum gap between two SFDs is 160 MII clocks (128 clocks for a 64-byte frame + 24 clocks of min IFG + 8 clocks of preamble), while for 1000-Mbps half-duplex transmit operation, the gap can be as low as 24 GMII clocks (4 for a JAM pattern sent just after SFD due to collision + 12 IFG + 8 preamble).

In the first example,  $(3 \times \text{PTP}) + (3 \times \text{MII}) \leq 160 \times \text{MII}$ ; thus, the minimum PTP clock frequency is about 0.5 MHz ( $(160 - 3) \times 40 \text{ ns} \div 3 = 2,093 \text{ ns period}$ )

Similarly, in the second example,  $3 \times \text{PTP} + 3 \times \text{GMII} \leq 24 \times \text{GMII}$ , resulting in a minimum PTP clock frequency of about 18 MHz.

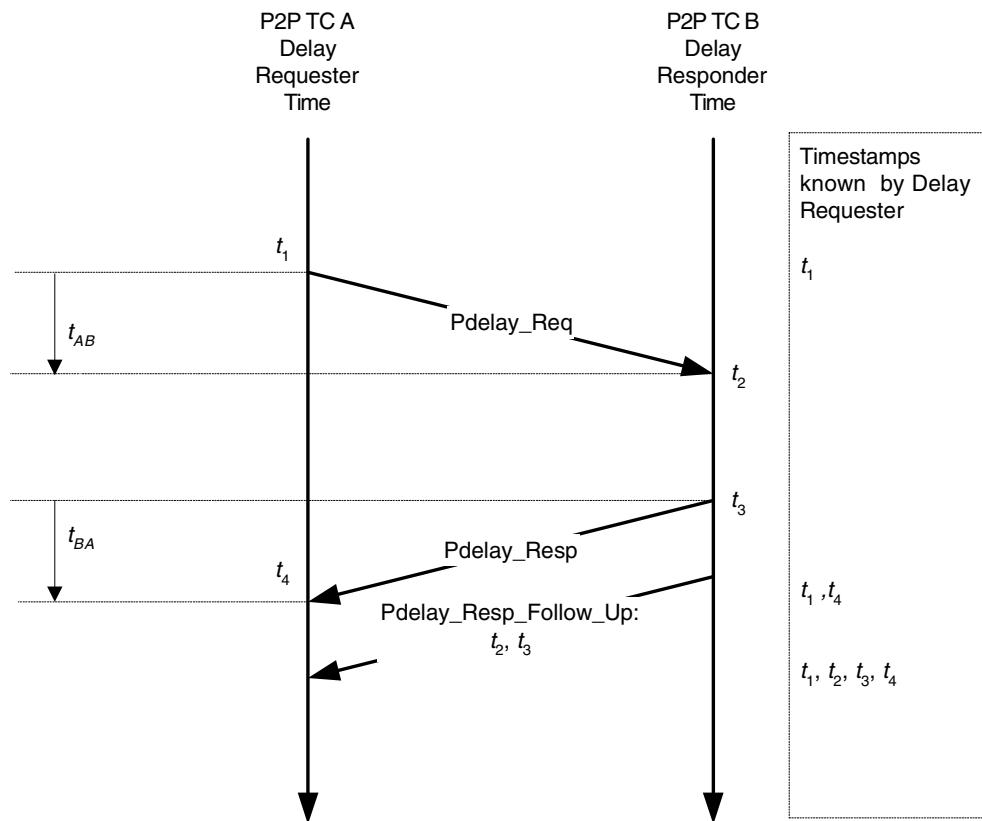
### 3.2.6 Advanced Time Stamp Feature Support

In addition to the basic features for time stamp mentioned in [Section 3.2](#), the advanced time stamp option has the following features.

- ❖ Support for the IEEE 1588-2008 (Version 2) timestamp format.
- ❖ Option to take snapshot for all frames or for PTP type frames.
- ❖ Option for taking snapshot for event messages only.
- ❖ Option to take the snapshot based on the clock type (ordinary, boundary, end-to-end and peer-to-peer)
- ❖ Option to select the node to be a Master or Slave for ordinary and boundary clock.
- ❖ Identification of PTP message type, version, and PTP payload sent directly over Ethernet given as status.
- ❖ Option to measure time in digital or binary format.

#### 3.2.6.1 Peer-to-Peer PTP (Pdelay) Transparent Clock (P2P TC) Message Support

The IEEE 1588-2008 version supports Pdelay message in addition to SYNC, Delay Request, Follow-up and Delay Response messages. [Figure 3-3](#) shows the method to calculate the propagation delay in clocks supporting peer-to-peer path correction.

**Figure 3-3 Propagation Delay Calculation in Clocks Supporting Peer-to-Peer Path Correction**

The link delay measurement starts with port-1 issuing a “Pdelay\_Req” message and generating a timestamp, for the Pdelay\_Req message. Port-2 receives the “Pdelay\_Req” message and generates a timestamp,  $t_2$ , for this message. Port-2 returns a Pdelay\_Resp message and generates a timestamp,  $t_3$ , for this message. To minimize errors due to any frequency offset between the two ports, Port-2 returns the Pdelay\_Resp message as quickly as possible after the receipt of the Pdelay\_Req message.

Port-2 either:

- ❖ Returns the difference between the timestamps  $t_2$  and  $t_3$  in the Pdelay\_Resp message,
- ❖ Returns the difference between the timestamps  $t_2$  and  $t_3$  in a Pdelay\_Resp\_Follow\_Up message, or
- ❖ Returns the timestamps  $t_2$  and  $t_3$  in the Pdelay\_Resp and Pdelay\_Resp\_Follow\_Up messages respectively.

Port-1 generates a timestamp,  $t_4$ , upon receiving the Pdelay\_Resp message. Port-1 then uses these four timestamps to compute the mean link delay.

### 3.2.6.2 Clock Types

The type of clock nodes supported in IEEE 1588-2008 is described in this section. The corresponding support provided by the advanced time stamp feature for each of the clock type is also mentioned.

1. **Ordinary clock support:** In this type the clock can be a grandmaster or a slave clock. This clock has a single PTP state. The following features are supported.

- a. Sends and receives PTP messages. The time stamp snapshot can be controlled as described in [“Register 448 \(Time Stamp Control Register\)”](#) on page [301](#).
- b. Maintains the data sets (e.g., time stamp values).

[Table 3-1](#) shows the messages for which time-stamp snapshot is taken on the receive side for Master and Slave nodes.

**Table 3-1 PTP Messages for which Snapshot is Taken on Receive Side for Ordinary Clock**

Master	Slave
Delay_Req	SYNC

The ordinary clock in the domain supports a single copy of the protocol and has a single PTP state and will typically be a single physical port. In typical industrial automation applications, an ordinary clock is associated with an application device such as sensors and actuators. In telecom applications, the ordinary clock may be associated with a timing demarcation device.

Typically for ordinary clock, you will need to take snapshot for only one type of PTP messages. For e.g. you will require supporting either version 1 or 2 PTP messages, not both.

2. **Boundary clock support:** This type of clock is similar to the ordinary clock except for the following.
  - a. The clock data sets are common to all ports of the boundary clock
  - b. The local clock is common to all ports of the boundary clock.

Hence the features of ordinary clock holds good for the boundary clock also.

The boundary clock typically has several physical ports communicating with the network. The messages related to synchronization, master-slave hierarchy and signaling terminate in the protocol engine of the boundary clock and are not forwarded. The PTP message type status given by the core (refer to [“Receive Path Functions”](#) on page [48](#)) will help you to quickly identify the type of message and take appropriate action.

3. **End to end transparent clock support:** The end-to-end transparent clock forwards all messages like normal bridge, router or repeater. The resident time needs to be computed to update the correctionField. Hence snapshot needs to be taken for the messages mentioned in [Table 3-2](#).

**Table 3-2 PTP Messages for which Snapshot is Taken for Transparent Clock Implementation**

SYNC
FOLLOW_UP

In the end-to-end transparent clock, the residence times are accumulated in a special field (correctionField) of the PTP event (SYNC) message or the associated Follow-up (FOLLOW\_UP) Message. Hence it is important to take a snapshot for these messages alone. This can be quickly done by setting the control bit (TSEVNTEA), which enables snapshot to be taken for event messages and also selecting the type of clock in the “Time Stamp Control Register”.

The residence time is also corrected for Delay\_Req messages (but snapshot of the timestamp is not required). The message type statuses provided helps you to quickly identify the message and update the correctionField.

The message type status provided will also help in taking appropriate action depending on the type of PTP message received.

4. **Peer to peer transparent clock support:** In this type of clock the computation of the link delay is based on an exchange of Pdelay\_Req, Pdelay\_Resp and Pdelay\_Resp\_Follow\_Up messages with the link peer. Hence support for taking snapshot for the event messages related to Pdelay is added. [Table 3-3](#).

**Table 3-3 PTP Messages for which Snapshot is Taken for Peer-to-Peer Transparent Clock Implementation**

SYNC
Pdelay_Req
Pdelay_Resp

The transparent clock corrects only the SYNC and Follow-up message. As discussed earlier this can be achieved using the message status provided.

The type of clock to be implemented will be configurable through control register, as mentioned in “[IEEE 1588 Time Stamp Registers](#)” on page [301](#). To ensure that the snapshot is taken only for the messages indicated in the table for the corresponding clock type, the “TSEVNNTENA: Enable Time Stamp Snapshot for Event Messages” bit has to be set.

### 3.2.6.3 PTP Processing and Control

The common message header for PTP messages is shown below. This format is taken from IEEE standard 1588-2008 (Revision of IEEE Std. 1588-2002).

**Table 3-4 Message Format Defined in IEEE 1588-2008**

<b>BITS</b>		<b>OCTETS</b>	<b>OFFSET</b>
transportSpecific	messageType	1	0
Reserved	versionPTP	1	1
	messageLength	2	2
	domainNumber	1	4
	Reserved	1	5
	flagField	2	6
	correctionField	8	8
	Reserved	4	16
	sourcePortIdentity	10	20
	sequenceId	2	30
	controlField (*)	1	32
	logMessageInterva	1	33

(\*) – controlField is used in version 1. For version 2, messageType field will be used for detecting different message types.

There are some fields in the PTP frame that are used to detect the type and control the snapshot to be taken. This is different for PTP frames sent directly over Ethernet, PTP frames sent over UDP / IPv4 and PTP frames that are sent over UDP / IPv6. The following sections provide information on the fields that are used to control taking the snapshot.

### 3.2.6.3.1 PTP Frame Over IPv4

[Table 3-5](#) gives the details of the fields that will be matched to control snapshot for PTP packets over UDP over IPv4 for IEEE 1588 version 1 and 2. Note that the octet positions for tagged frames will be offset by 4. This is based on Appendix-D of the IEEE 1588-2008 standard and the message format defined in [Table 3-4](#).

**Table 3-5 IPv4-UDP PTP Frame Fields Required for Control and Status**

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	0x0800	IPv4 datagram
IP version	14	0x45	IP version is IPv4
Layer 4 protocol	23	0x11	UDP
IP Multicast address (IEEE 1588 version 1)	30, 31, 32, 33	0xE0,0x00, 0x01,0x81 (or 0x82 or 0x83 or 0x84)	Multicast IPv4 addresses allowed. 224.0.1.129 224.0.1.130 224.0.1.131 224.0.1.132
IP Multicast address (IEEE 1588 version 2)	30, 31, 32, 33	0xE0, 0x00, 0x01, 0x81 (or 0x6B)	PTP-primary multicast address: 224.0.1.129 PTP-Pdelay multicast address: 224.0.0.107
UDP destination port	36, 37	0x013F, 0x0140	0x013F – PTP event message (*) 0x0140 – PTP general messages
PTP control field (IEEE version 1)	74	0x00/0x01/0x02/ 0x03/0x04	0x00 – SYNC, 0x01 – Delay_Req, 0x02 – Follow_Up 0x03 – Delay_Resp 0x04 – Management
PTP Message Type Field (IEEE version 2)	42 (nibble)	0x0/0x1/0x2/0x3/0x8/0x9/0 xB/ 0xC/0xD	0x0 – SYNC 0x1 – Delay_Req 0x2 – Pdelay_Req 0x3 – Pdelay_Resp 0x8 – Follow_Up 0x9 – Delay_Resp 0xA – Pdelay_Resp_Follow_Up 0xB – Announce 0xC – Signaling 0xD - Management
PTP version field	43 (nibble)	0x1 or 0x2	0x1 – Supports PTP version 1 0x2 – Supports PTP version 2

(\*) PTP event messages are SYNC, Delay\_Req (IEEE 1588 version 1 and 2) or Pdelay\_Req, Pdelay\_Resp (IEEE 1588 version 2 only).

### 3.2.6.3.2 PTP Frame Over IPv6

[Table 3-6](#) gives the details of the fields that will be matched to control snapshot for PTP packets over UDP over IPv6 for IEEE 1588 version 1 and 2. Note that the octet positions for tagged frames will be offset by 4. This is based on Appendix-E of the IEEE 1588-2008 standard and the message format defined in [Table 3-4](#).

**Table 3-6 IIPv6-UDP PTP Frame Fields Required for Control and Status**

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	0x86DD	IP datagram
IP version	14	0x06	IP version is IPv6
Layer 4 protocol	20 (*)	0x11	UDP
PTP Multicast address	38 – 53	FF0x:0:0:0:0:0:0:0:181 (Hex) FF02:0:0:0:0:0:0:0:6B (Hex)	PTP – primary multicast address: FF0x:0:0:0:0:0:0:0:181 (Hex) PTP – Pdelay multicast address: FF02:0:0:0:0:0:0:0:6B (Hex)
UDP destination port	56, 57 (*)	0x013F	0x013F – PTP event message 0x0140 – PTP general messages
PTP control field (IEEE 1588 Version 1)	93 (*)	0x00/0x01/0x02/0x03/0x04	0x00 – SYNC, 0x01 – Delay_Req, 0x02 – Follow_Up 0x03 – Delay_Resp 0x04 – Management (version1)
PTP Message Type Field (IEEE version 2)	74 (*) (nibble)	0x0/0x1/0x2/0x3/0x8/0x9/0xB/ 0xC/0xD	0x0 – SYNC 0x1 – Delay_Req 0x2 – Pdelay_Req 0x3 – Pdelay_Resp 0x8 – Follow_Up 0x9 – Delay_Resp 0xA – Pdelay_Resp_Follow_Up 0xB – Announce 0xC – Signaling 0xD - Management
PTP version field	75 (nibble)	0x1 or 0x2	0x1 – Supports PTP version 1 0x2 – Supports PTP version 2

(\*) The Extension Header is not defined for PTP packets.

### 3.2.6.3.3 PTP Frame Over Ethernet

[Table 3-7](#) gives the details of the fields that will be matched to control snapshot for PTP packets over Ethernet for IEEE 1588 version 1 and 2. Note that the octet positions for tagged frames will be offset by 4. This is based on Appendix-E of the IEEE 1588-2008 standard and the message format defined in [Table 3-4](#).

**Table 3-7** IEthernet PTP Frame Fields Required for Control And Status

Field Matched	Octet Position	Matched Value	Description
MAC Frame type	12, 13	0x88F7	PTP Ethernet frame.
PTP control field (IEEE Version 1)	45	0x00/0x01/0x02/ 0x03/0x04	0x00 – SYNC 0x01 – Delay_Req 0x02 – Follow_Up 0x03 – Delay_Resp 0x04 – Management
PTP Message Type Field (IEEE version 2)	14 (nibble)	0x0/0x1/0x2/0x3/0x8/0x9/0xB/ 0xC/0xD	0x0 – SYNC 0x1 – Delay_Req 0x2 – Pdelay_Req 0x3 – Pdelay_Resp 0x8 – Follow_Up 0x9 – Delay_Resp 0xA – Pdelay_Resp_Follow_Up 0xB – Announce 0xC – Signaling 0xD – Management
MAC Destination multicast address (*)	0-5	01-1B-19-00-00-00 01-80-C2-00-00-0E	All except peer delay messages - 01-1B-19-00-00-00 Pdelay messages - 01-80-C2-00-00-0E
PTP version field	15 (nibble)	0x1 or 0x2	0x1 – Supports PTP version 1 0x2 – Supports PTP version 2

(\*) In addition, the address match of destination addresses (DA) programmed in MAC address 1 to 31 will be used, if the control bit 18 (TSENMACADDR: Enable MAC address for PTP frame filtering) of the Time Stamp Control register is set.

### 3.2.6.4 Reference Timing Source (for Advance Timestamp Feature)

The updated functionality for advanced timestamp support is mentioned in the following points.

1. The IEEE 1588-2008 standard defines the seconds field of the time to be 48-bits wide. The fields to time-stamp will be the following.
  - ◆ UInteger48- seconds field
  - ◆ UInteger32-nanoseconds field

The “seconds” field is the integer portion of the timestamp in units of seconds. The “nanoseconds” field is the fractional portion of the timestamp in units of nanoseconds. E.g. 2.000000001 seconds is represented as secondsField = 0x0000\_0000\_0002 and nanoSeconds = 0x0000\_0001. Thus the maximum value in nanoseconds field in this format will be 0x3B9A\_C9FF value (i.e (10e9-1) nanoseconds). This is defined as digital rollover mode of operation. It will also support the older mode in which the nano-seconds field will roll-over and increment the seconds field after the value of 0x7FFF\_FFFF. (Accuracy is ~0.466 ns per bit). This is defined as the binary rollover mode. The modes can be controlled using the “TSCTRLSSR: Time Stamp Digital or Binary rollover control” bit of the time stamp control register.
2. When the Advanced IEEE 1588 time-stamp feature is selected time maintained in the core will still be 64-bit wide, as the overflow to the upper 16-bits of seconds register happens once in 130 years. The value of the upper 16-bits of the seconds field can only be obtained from the CSR register. This is optional and can be controlled by the coreKit parameter “IEEE1588 Higher Word Register Enable” described in Chapter 6. A CSR status bit which indicates if the 32-bit “seconds” field has overflowed is also provided.
3. There is also a pulse-per-second output given to indicate 1 second interval (default). Option is provided to change the interval. Refer section 5.2.3 “IEEE 1588 Time Stamp Registers” on page 301 for more information.
4. Option to take auxiliary time-stamp snapshot with an external event is supported. Refer to section “Auxiliary Snapshot” on page 49 for details.

### 3.2.6.5 Transmit Path Functions

There are no changes in the transmit path functions for GMAC-CORE and GMAC-MTL configuration for the Advanced time stamp option.

In GMAC-AXI, GMAC-AHB, or GMAC-DMA configuration, the structure of the descriptor changes when Advanced IEEE 1588 version support is enabled. The IEEE 1588 timestamp feature is supported using Alternate (Enhanced) descriptors format only. The descriptor is 32-bytes long (8 DWORDS) and the snapshot of the timestamp is written in descriptor 6 and 7. The complete details on the descriptors when the Advanced IEEE 1588 time-stamping is enabled are mentioned in Chapter 8 (Alternate or Enhanced Descriptors).

### 3.2.6.6 Receive Path Functions

When the advanced time stamp feature is selected, processing of the received frames to identify valid PTP frames is done. The snapshot of the time to be sent to the application can be controlled.

The following options are provided in the time stamp control register to control the snapshot.

1. Option to enable snapshot for all frames.
2. Enable snapshot for IEEE 1588 version 2 or version 1 time stamp.
3. Enable snapshot for PTP frames transmitted directly over Ethernet or UDP-IP-Ethernet.
4. Enable time stamp snapshot for the received frame for IPv4 or IPv6.

5. Enable time stamp snapshot for EVENT messages (SYNC, DELAY\_REQ, PDELAY\_REQ or PDELAY\_RESP) only.
6. Enable the node to be a Master or Slave. This will control the type of messages for which snap-shot will be taken (this depends on the type of clock that is selected and is valid for ordinary or boundary clock only).

Note that PTP messages over VLAN frames are also supported.

In GMAC-CORE configuration, the time stamp is given on a 64-bit bus (`mri_timestamp_o`) on MRI along with EOF. Additional signal (`mri_timestamp_val_o`) shall validate the presence of timestamp for the receive frame. Additional status is provided when the advanced time stamp feature is enabled in the coreKit (Chapter 6). Refer to [Table 3-7](#) for the additional status bits.

In GMAC-MTL configuration, the time-stamp is provided on the data bus after the EOF data has been transferred. Separate signal is provided to validate the time stamp. This signal (`ari_timestamp_val_o`) is asserted to indicate the availability of time stamp. Once the time stamp is transferred, the status is provided on the data bus with status valid signal being asserted. When 32-bit data width is selected, additional status related to timestamp and IPC (Advanced Timestamp feature is enabled in the coreKit) will be provided on the data bus after the normal status is read out. The additional status is provided only when the bit-0 of the normal status is set and is validated by the assertion of (`ari_rxstatus_val_o`) signal. In 64/128 bit mode, the additional status is provided using the bits 32-45 of the normal status as in the case of GMAC-CORE.

GMAC-AHB/ AXI and GMAC-DMA configurations, the time stamp is returned to the software inside the corresponding Transmit and Receive Descriptor. The Advanced time stamp feature is supported using Alternate (Enhanced) descriptors only which is 32-bytes long. Extended status containing the time stamp message status and the IPC offload status is written back in descriptor 4 and the snapshot of the time stamp is written back in descriptor 6 and 7. The complete details on the descriptors when the advanced time stamping is enabled are provided in "[Alternate or Enhanced Descriptors](#)" on page [356](#).

### 3.2.6.7 Auxiliary Snapshot

The auxiliary snap shot feature allows you to store a snapshot of the system time based on an external event. The event is considered to be the rising edge of the sideband signal `ptp_aux_ts_trig_i`. This feature is independent of whether the system time is generated internally or given as input (on `ptp_timestamp_i` bus). Such snap-shots are stored in a 4-deep FIFO. Only 64-bit of the time will be stored in the FIFO. The upper 16-bits of the seconds register can be read from "Time Stamp Higher Word Register" when it is present.

The storing of the snapshot can be indicated to the host with an interrupt. The value of the snapshot is read out through a FIFO register access (see "[IEEE 1588 Time Stamp Registers](#)" on page [301](#)). When the FIFO becomes full and an external trigger to take the snapshot is asserted, then a snapshot trigger-missed status is set in the Time Stamp Status Register. This indicates that the latest auxiliary snapshot of the timestamp was not stored in the FIFO. The latest snapshot will not be written to the FIFO when it is full. When the host reads the 64-bit timestamp from the FIFO, space is available to store the next snapshot. This FIFO can be cleared using the control bit "ATSFC: Auxiliary Snapshot FIFO clear" defined in the Time Stamp Control Register. When multiple snapshots are present in the FIFO, the count is indicated in the "ATSNS: Auxiliary Time Stamp Number of Snapshots" bits of the Time Stamp Status Register.



This asynchronous input signal (`ptp_aux_ts_trig_i`) is first synchronized to `clk_ptp_ref_i` domain, the snap-shot taken and transferred to the `clk_csr` domain. Due to this latency, the minimum gap between two such events on "`ptp_aux_ts_trig_i`" must have at least 4 cycles of `clk_ptp_i` plus 3 cycles of `clk_csr`. Otherwise, the rising-edge of the trigger will be missed by the logic.

This optional feature is selected during coreKit configuration (see "[IEEE 1588 Time Stamp Block](#)" on page [320](#)) and is available only when Advanced Time Stamping feature is selected.

### 3.3 AHB Application Host Interface

In the GMAC-AHB core, the DMA Controller interfaces with the Host through the AMBA AHB Interface. The AHB Master Interface controls data transfers while the AHB Slave interface accesses CSR space. The DMA can be used in 32/64/128-bit embedded applications where DMA is required to optimize data transfer between the GMAC and system memory.

The AHB Master interface converts the internal DMA request cycles into AHB cycles.

Characteristics of this interface include the following:

- ❖ Fully AMBA 2.0-compliant AHB Master: no restrictions
- ❖ You can choose fixed burst length (SINGLE, INCR4, INCR8, INCR16) or unspecified burst length (SINGLE, INCR) transfers or a mix of fixed and unspecified burst transfers by programming the FB or MB bits in the DMA Bus Mode register (see “[DMA Register Description](#)” on page [257](#)).
- ◆ When fixed burst length is chosen, the AHB master always initiates a burst with SINGLE, INCR4, INCR8 or INCR16 type. But when such a burst is responded with SPLIT/RETRY/early burst termination, the AHB master will re-initiate the pending transfers of the burst with INCR or SINGLE burst-length type. It will terminate such INCR bursts when the original requested fixed-burst is transferred. In Fixed Burst-Length mode, if the DMA requests a burst transfer that is not equal to INCR4/8/16, the AHB Host interface splits the transfer into multiple burst transactions. For example, if the DMA requests a 15-beat burst transfer, the AHB interface splits it into multiple transfers of INCR8 and INCR4, and 3 SINGLE transactions.
- ◆ In unspecified burst mode, the AHB master will always initiate a transfer with INCR and complete the DMA requested burst in one go.
- ◆ In mixed burst mode, the AHB master will initiate bursts with fixed-size (INCRx) when the DMA requests transfers of size less than or equal to 16 beats. When DMA requests bursts of length more than 16, the AHB master will initiate such transfers with INCR and complete it in one go.
- ❖ The AHB slaves interfaced to the GMAC-AHB must support SINGLE and INCR transfers, at a minimum.
- ❖ Takes care of AHB SPLIT, RETRY, and ERROR conditions. Any ERROR response will halt all further transactions for that DMA, and indicate the error as fatal through the CSR and interrupt. The application must give a hard or soft reset to the module to restart the operation.
- ❖ Takes care of AHB 1K boundary breaking
- ❖ Handles all 32-, 64-, or 128-bit data transfers (according to the data bus width configuration), except for Descriptor Status Write accesses (which are always 32-bit). In any burst data transfer, the address bus value is always aligned to the data bus width and need not be aligned to the beat size.
- ❖ All AHB burst transfers can be aligned to an address value by enabling the DMA Bus Mode register’s AAL bit. If both the FB and AAL bits are set to 1, the AHB interface and the DMA together ensure that all initiated beats are aligned to the address, completing the frame transfer in the minimum number of required beats. For example, in 32-bit data bus mode, if a data buffer transfer’s start address is 0xF000\_0008 and the DMA is configured for a maximum beat size of 16, the AHB transfers occur in the following sequence:
  - ◆ 2 SINGLE transfers at addresses 0xF000\_0008 and 0xF000\_000C
  - ◆ 1 INCR4 transfer at address 0xF000\_0010
  - ◆ 1 INCR8 transfer at address 0xF000\_0020

- ◆ All subsequent beats are INCR16 transfers starting from address 0xF000\_0040. For an address-aligned INCR16 transfer, the 6 least-significant bits of the address must be 0.

Similarly, the AHB and DMA split such unaligned address beats for a 64-bit (or 128-bit) data bus and initiate INCR16 beats only when the address's seven (or eight) least-significant bits have a zero value.

- ❖ The DMA Controller requests an AHB Burst Read transfer only when it can accept the received burst data completely. Data read from the AHB is always pushed into the DMA without any delay or BUSY cycles.
- ❖ The DMA requests an AHB Burst Write transfer only when it has the sufficient data to transfer the burst completely. The AHB interface always assumes that it has data available to push into the AHB bus. However, the DMA can prematurely indicate end-of-valid data (due to the transfer of end-of-frame of an Ethernet frame) during the burst. In Fixed Burst Length mode, the AHB Master interface continues the burst with dummy data until the specified length is completed. In INCR mode, it takes steps to end the burst transfer prematurely.

The AHB 32-bit Slave interface provides access to the DMA and GMAC CSR space. Characteristics of this interface include the following:

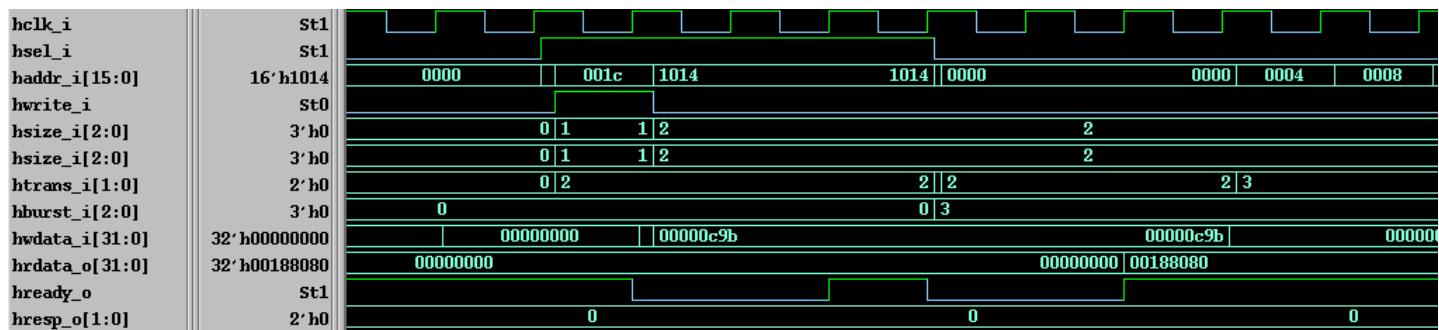
- ❖ Fully AMBA 2.0 Compliant AHB Slave – no restrictions
- ❖ Supports single and all burst transfers
- ❖ Supports busy and early terminations
- ❖ Supports 32-bit, 16-bit, and 8-bit write/read transfers to the CSR; 32-bit access to the CSR are recommended to avoid any SW synchronization problems.
- ❖ Generates OKAY only response; does not generate SPLIT, RETRY, or ERROR responses.

To interface the 32-bit AHB Slave port to a wider AMBA bus, extra logic described in Section 3-15 of the AMBA 2.0 specification is required. You can include this logic in the core at the time of RTL configuration (see “[Configuration: Creating the RTL](#)” on page [30](#)). In such configurations, even though the AHB slave port indicates a 64- or 128-bit data bus, all slave port accesses must still be 32 bits or less. If a 64- or 128-bit access is performed, the AHB slave port completes a 32-bit access internally and gives an OKAY response. The data on the higher lanes is ignored during write operations and the lower 32-bit lane is duplicated during AHB Slave Read transfers.

You can also enable an APB port to access DMA and GMAC CSR space. All register accesses are completed in 2 clock cycles to support an APB port. The GMAC-AHB default and optional signals are described in “[Signal Descriptions](#)” on page [179](#)

### 3.3.1 AHB Slave Port Timing

[Figure 3-4](#) shows a register write access (to address 0x001C) followed by a register read access (to address 0x1014) on the AHB Slave port. The waveform is as per the AMBA 2.0 specifications and self-explanatory. The AHB slave port completes all read and write transfers in 3 clocks and therefore, hready\_o is deasserted for 2 clocks for any register access. The AHB slave port supports burst operations and the timing response remains the same (3 clocks for completion of any read/write cycle).

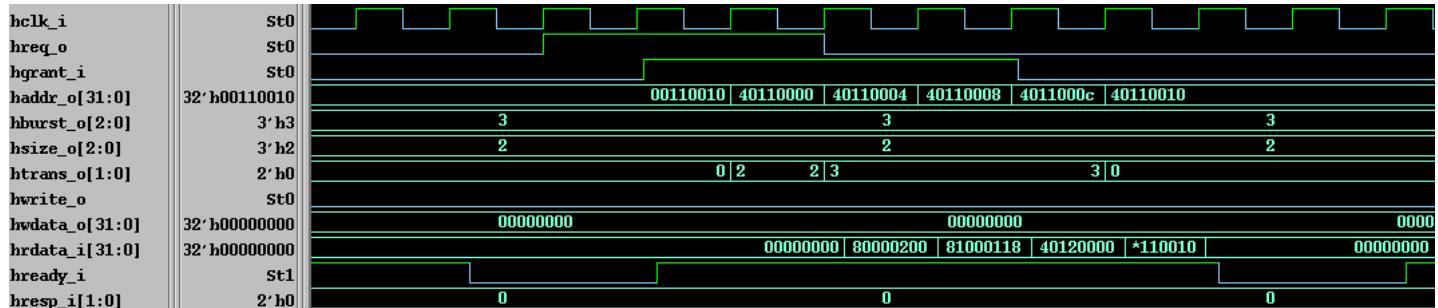
**Figure 3-4 Register Write Access Followed by a Register Read Access on the AHB Slave Port**

### 3.3.2 AHB Master Port Timing

#### 3.3.2.1 AHB Memory Read

Figure 3-5 shows the GMAC-AHB performing a memory read starting from address 0x40110000. The AHB master port uses the INCR4 burst on this occasion.

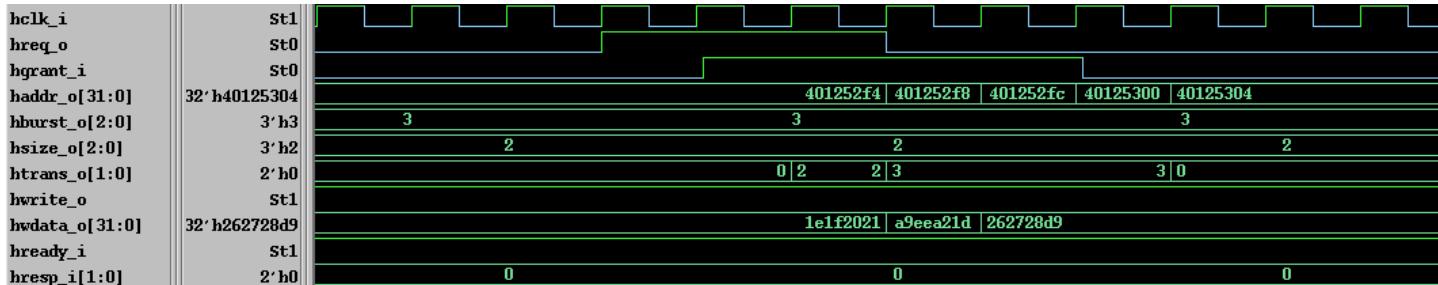
**Figure 3-5 AHB Memory Read Timing**



#### 3.3.2.2 AHB Memory Write

Figure 3-6 shows the GMAC-AHB performing a burst memory write transaction, starting from address 0x401252F4. This data transfer corresponds to an Ethernet frame being written to system memory by the Receive DMA (Please refer to “DMA Controller” on page 64 for DMA operations). The AHB master uses the INCR4 burst on this occasion. Notice that the data bus does not change values for the last two beats in the INCR4 transaction. This is because the EOF of the frame was given with the second beat transfer, by the DMA to the AHB master port internally. As the AHB Master had started a fixed INCR4 burst, it completes the burst with dummy data (same data) for the last 2 beats.

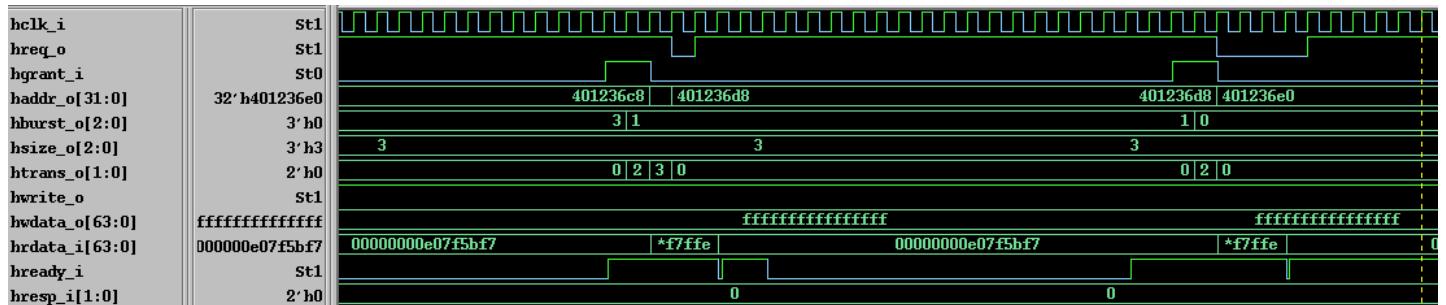
**Figure 3-6 AHB Memory Write Timing**



### 3.3.2.3 Early Burst Termination

Figure 3-7 shows an early burst termination occurring when the GMAC-AHB carries out an INCR (Write) transaction. This self-explanatory figure shows how the AHB master re-requests the bus grant and completes the transfer with a SINGLE transaction.

**Figure 3-7 Early Burst Termination Timing**



### 3.3.2.4 Error Response and Fatal Bus Error Interrupt

Figure 3-8 shows the behavior of the GMAC-AHB when an error response is received. Here, the DUT attempts to carry out the INCR4 transaction. However, it receives a two-cycle error response during the data phase of the first beat. Interrupt signal sbd\_intr\_o is asserted as a result of this response, as shown. See “Interrupts” on page 76 for the interrupt functions.

**Figure 3-8 Error Response and Fatal Bus Error Interrupt Timing**



### 3.3.2.5 Split/Retry Response

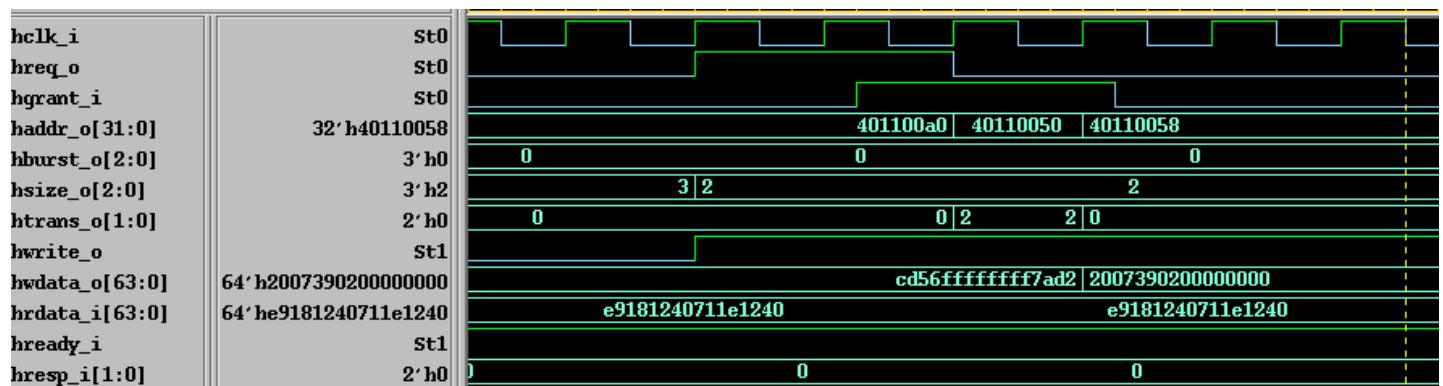
Figure 3-9 shows how the AHB Master port reacts to a split/retry response (to address 0x00120560) during a transaction. In this case, the GMAC-AHB receives a two-cycle retry response. Notice that the AHB Master immediately reconstructs the transaction and requests for bus by asserting hreq\_o again. hreq\_o is deasserted for only 1 cycle when the retry/split response is sampled in the first cycle of the 2-cycle response.

In case of a SPLIT response, the behavior remains the same, but the second request (hreq\_o) from this master will be masked in the AHB Arbiter until the corresponding AHB slave is ready with the data.

**Figure 3-9 Split/Retry Response Timing**

### 3.3.2.6 Descriptor Write-Back

Figure 3-10 shows a descriptor being closed on a 64-bit AHB at address 0x40110050 (See “[DMA Controller](#)” on page 64 and “[Descriptors](#)” on page 337 for descriptor operations). Note, that only 32-bit status is written back to memory and this appears on upper 32-bit of the hwdata\_o bus, as the AHB is configured as big-endian. For little-endian configuration, the 32-bit descriptor data appears on the lower 32 bits of the 64-bit AHB bus.

**Figure 3-10 Descriptor Write-Back Timing**

## 3.4 AXI Application Host Interface

### 3.4.1 AXI Overview

The AMBA 3 AXI bus interface provides the characteristics to support highly effective data traffic throughput. The System bus utilization is maximized by allowing simultaneous Read and Write transactions. The AXI Master interface allows multiple numbers of burst requests to be queued for Read/Write operation. System bus utilization can be further enhanced through by supporting Requests Re-Ordering and Data-Interleaving which allows Master the flexibility to select multiple slaves dynamically and independently to handle Priority Requests and Slow peripheral Slaves. Requests Re-ordering and Data Interleaving in AXI Write channel is not supported by this AXI Master in this release. It also supports low power interface defined in the AXI specifications.

### 3.4.2 Burst Splitting and Burst Selection

The GMAC-AXI splits the DMA requests into multiple bursts on the AXI System Bus. The splitting is based on DMA Count and software controllable burst enables bits UNDEF, BLEN256, BLEN128, BLEN64, BLEN32, BLEN16, BLEN8, BLEN4 and burst types (INCR and INCR\_ALIGNED) which is also controllable through the software. The GMAC-AXI also takes care of splitting the transfers that cross 4KB address boundary. Burst Length Select Priority is in the sequence: UNDEF, BLEN256, BLEN128, BLEN64, BLEN32, BLEN16, BLEN8, BLEN4

#### 3.4.2.1 INCR Burst Type

If the UNDEF is enabled, then GMAC-AXI always chooses the maximum allowed burst length based on bits BLEN256, BLEN128, BLEN64, BLEN32 (possible maximum burst lengths are 256, 128, 64, 32 or 16). In cases where the DMA requests are not multiples of the maximum allowed burst length or due to "4K address boundary crossing" condition, the AXI will choose a burst-length of any value less than the maximum enabled burst-length (all lesser burst-length enables are redundant). For example, when BLEN64, BLEN16, BLEN4 are only enabled and the DMA requests a burst transfer of size 102 beats, then the AXI will split it into two bursts of size 64 and 38 beats respectively.

If UNDEF is not enabled, then the burst length is based on the priority of the enabled bits in the following order – (BLEN256, BLEN128, BLEN64, BLEN32, BLEN16, BLEN8, BLEN4). When the DMA requests to transfer a burst, the AXI interface will split the requested bursts into multiple transfers using only the enabled burst lengths. This splitting can occur either because the requested burst is not a multiple of the maximum enabled burst or due to 4K address boundary crossing. If it can not choose any of the enabled burst length then it selects the burst length as 1. For example, when BLEN64, BLEN16, BLEN4 are only enabled and the DMA requests a burst transfer of size 102 beats, then the AXI will split it into multiple bursts of size 64, 16, 16, 4, 1 and 1 beats respectively. (The sequence is in decreasing burst sizes.)

#### 3.4.2.2 INCR\_ALIGNED Burst Type

When the Address-aligned burst-type is enabled, the AXI interface will split the DMA requested bursts such that each burst-size is aligned to the start address least significant bits in addition to the conditions for splitting bursts explained in previous section. It will initially generate smaller bursts such that the remaining transfers can be transferred with the maximum possible (enabled) fixed burst lengths.

For example in the same setting as explained above for UNDEF (BLEN64, BLEN16 and BLEN4 are enabled), DMA requests a burst of size 102 beats at the start address of 0x0000\_0164 (32-bit bus). The AXI will then start the first transfer with size 23 such that the address of the next burst will be aligned (0x0000\_0400) for a burst of 64. The sequence of bursts will thus be 23, 64 and 17 respectively.

When UNDEF is not set, then the sequence of transfers for the above scenario will be having burst lengths of 1, 1, 1, 4, 16, 64, 16 and 1 respectively. The sequence of smaller bursts at the beginning is to align the address to the next higher enabled burst-lengths programmed in the register.

### 3.4.3 Outstanding Transactions

The GMAC-AXI supports up to four or eight (user-configurable in coreConsultant) Read/Write outstanding requests on the AXI bus. This can also be controlled through software by programming the WR\_OSR\_LMT/RD\_OSR\_LMT bits in the AXI Bus Mode Register. This is a useful work around for any system level issues with the handling of multiple requests.



**Note** The maximum number of outstanding requests is divided equally between the requests generated by the two internal masters (RxDMA and TxDMA). i.e When the user selects 8 outstanding AXI requests during RTL configuration, then each DMA is limited to a maximum of 4 outstanding requests in each read or write channel. When the user reduces the outstanding request to say 4 by programming the respective control bits, then all the outstanding requests can be from the same DMA.

### 3.4.4 Priority of AXI Requests

The Descriptor transfers have higher priority than the Data Transfers. So, if there are two requests, for example, RX Descriptor Read and TX Data Read, then RX Descriptor Read is given higher priority so, that the next RX Data Write (subsequent to RX Descriptor Read) need not wait for TX Data Read transfer to complete. If there are descriptor read requests from both DMA, then they are serviced based on a first-come first-serve basis. RxDMA has higher priority if descriptor read requests are generated from both the DMAs in the same clock. Similarly, in the write channel, descriptor writes from any DMA have higher priority than the data-write transfers for the RxDMA.

### 3.4.5 Bursts Reordering and Data Interleaving

AXI protocol allows re-ordering of data transfers with different AXI-IDs with respect to the sequence of requests. It also allows data interleaving between transfers of different AXI-ID for both Read and Write Channels. In the GMAC-AXI, requests from each internal master (RxDMA and TxDMA) are generated with different AXI-ID. Hence re-ordering and data-interleaving can be performed as the two DMA operate independently and are allocated different address-space on the slave memory.

The GMAC-AXI supports reordering and interleaving from the AXI slaves on the Read channel. Each DMA internally ensures that a read request and write request to the same address (can occur for only descriptor accesses) is never generated at the same time. The TxDMA ensures that Tx Descriptor read, Tx Data read, and Tx Descriptor write-back requests are generated only after the previous requests are completed. And since the RxDMA generates only Descriptor reads on the read channel with a different ID, data re-ordering and interleaving on the read channel does not cause any problems in the core.

On the write channel, the data re-ordering or interleaving can happen between Tx Descriptor write-back and Rx Data write requests or between Tx Descriptor write-back and Rx Descriptor write-back requests.



**Note** In the current release of the GMAC-AXI, write requests reordering and data-interleaving on the write channel are NOT supported.

### 3.4.6 AXI-ID Translation

The GMAC-AXI selects separate ID data transfer requests from the TX and RX DMA. The GMAC-DMA has a 1-bit ID field with following encoding:

- ❖ 1'b0 – Rx DMA access (both for Read and Write)
- ❖ 1'b1 – Tx DMA access (both for Read and Write)

The selection of above ID encoding will take care of ordering requirement of GMAC-AXI requests.



**Note** The AXI system bus ID width is configurable. The Master default width is 4-bit. The upper 3-bits are hardwired to 0s. For the AXI Slave interface, the default ID width is 8.

### 3.4.7 Endianess

The GMAC-AXI only supports the Little-Endian mode on the AXI master and Slave ports.

### 3.4.8 Posted Writes

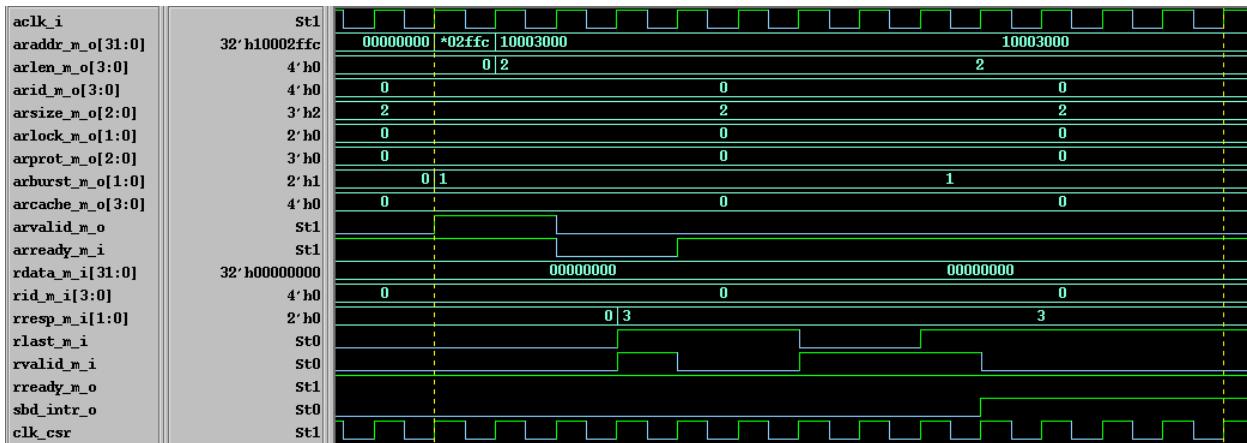
For Data transfers, the GMAC-AXI always issues posted write requests. (Without waiting for Slave acceptance after sending the data to AXI bus), to allow further pipelining of the requests. The RxDMA ensures that any further descriptor reads or descriptor writes will not happen until the posted write data transfer is completed successfully.

For Descriptor access GMAC-AXI always issues non-posted write requests. This is needed because the “transfer complete interrupt” generation is based on the descriptor writes for which it needs completion response from the memory slave. This guarantees no race condition, interrupt is guaranteed to be generated only after the data and descriptor is written in to the slave memory.

### 3.4.9 Error Response Handling

Whenever there is an error response from the AXI Slave (as shown in [Figure 3-11](#) with rresp\_m\_i[1:0] = 3), the respective DMA channel which generated the request gets disabled. The GMAC-AXI will assert the interrupt (sbd\_intr\_o in figure) when the corresponding “Fatal Bus Error” interrupt is enabled. The host will have to reset the core with hard-reset or soft-reset to re-start the DMA.

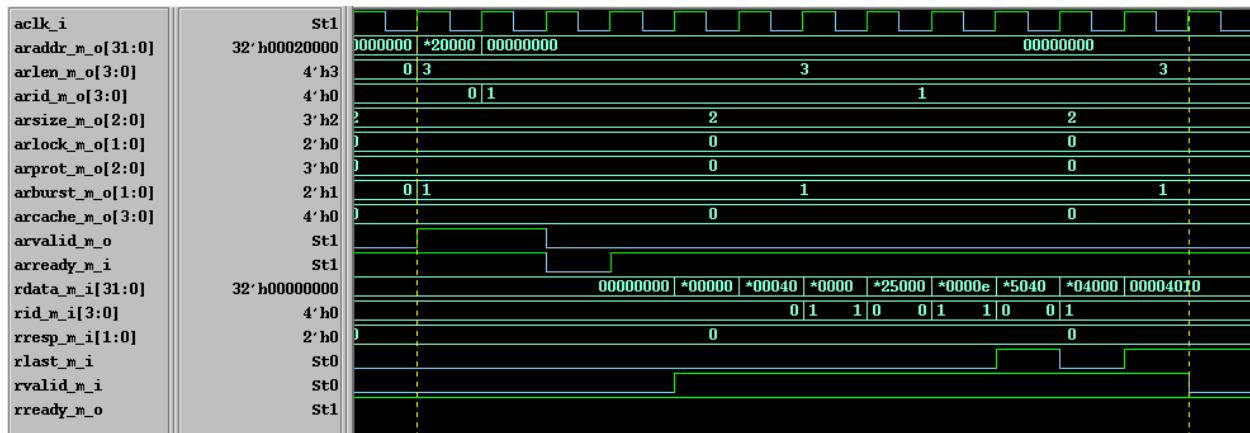
**Figure 3-11 Error Response Timing**



### 3.4.10 AXI Memory Read

Figure 3-12 shows the AXI Master read timings. Note that there are two requests placed for ID 0 and ID 1. The read data is interleaved by the interconnect (by way of ID 0 and ID 1) and received by the GMAC-AXI. As the TxDMA in GMAC-AXI always ensures that it has enough space in its FIFO to accept the requested burst of data, the r\_ready\_m\_o response will be always high.

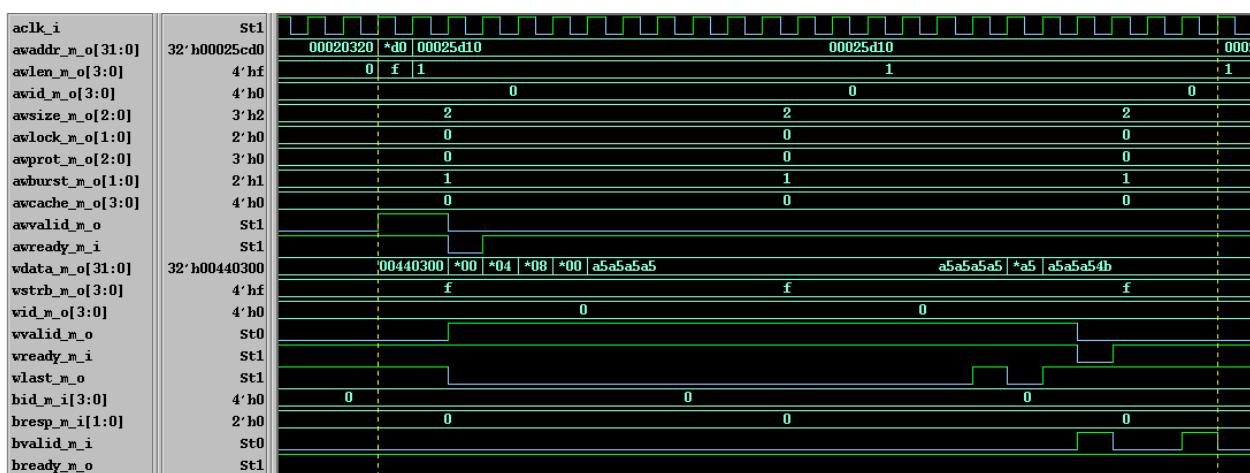
**Figure 3-12 AXI Memory Read Timing**



### 3.4.11 AXI Memory Write

Figure 3-13 shows the AXI write interface diagram. There are two requests for the ID 0 and are issued with burst lengths of 16 and 2 respectively. The latency between the first write request and the corresponding first data is two clocks. As the RxDMA ensures that it always has the request burst of data available in its FIFO before the request is generated, there is no further latencies or delays in the data transfer. As you can see in the diagram, the complete 18 beats of write data of the two requests is transferred in 18 clocks (provided there is no delay in the acceptance of that data by the AXI slave).

**Figure 3-13 AXI Memory Write Timing**



### 3.4.12 AXI Early Burst Termination

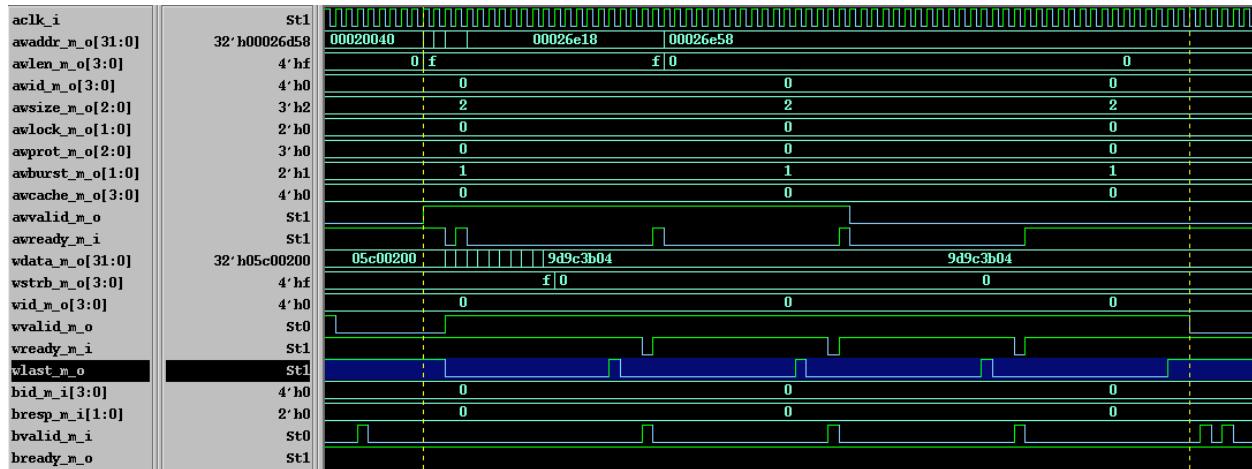
Figure 3-14 shows the early burst termination on the write channel. This can happen during data transfer when the end-of-frame is read out of the RxFIFO before the requested burst transfer is completed. There are 5 requests issued but the EOF gets transferred in middle of the first burst. The GMAC-AXI will still continue the data phases of the outstanding requested bursts but will not allow further data to be written to the slave memory by de-asserting the write-strobes (as indicated by wstrb\_m\_o[3:0] = 0).

You can observe that there are two extra request commands (@address 0x00026e18 and 0x00026e58) active after the write-strokes are de-asserted after EOF transfer. The first one had been already issued to slave and hence is not changed. The second one is a dummy request with single beat required to generate a last data transfer (with wlast high).



**Note** Considering the fact that there can be lot of wastage of AXI bandwidth with dummy writes after transfer of EOF data to host memory, it is recommended to have maximum of 2 or less outstanding requests on AXI master write channel by programming the AXI Bus mode register appropriately.

**Figure 3-14 AXI Early Burst Termination Timing**



### 3.4.13 Slave Interface Support and Timing

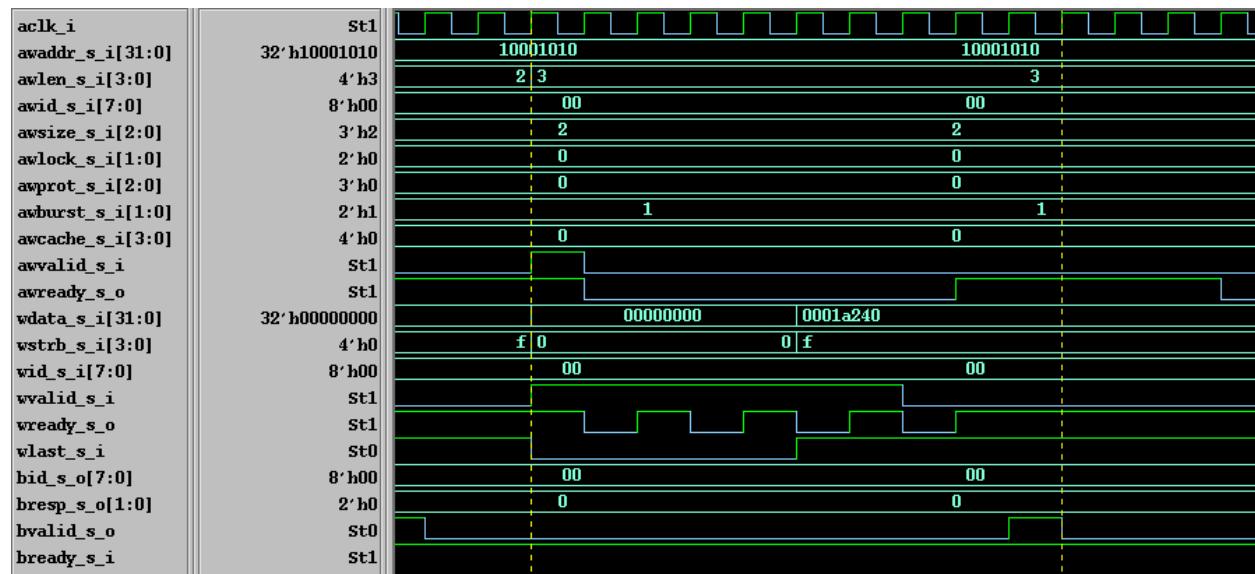
The GMAC-AXI supports an AXI-Slave Interface to access the CSR register sets. The features supported for the AXI slave interface are listed in “[AMBA AXI Interface Features](#)” on page 23.

The slave Interface has the following known limitations

- ❖ No support for data reordering
- ❖ No support for data interleaving
- ❖ No support for awcache/awprot or arcache/arprot
- ❖ No support for atomic accesses
- ❖ No support for Exclusive Access
- ❖ No support for WRAP/FIXED burst types; Only INCR burst type is supported.

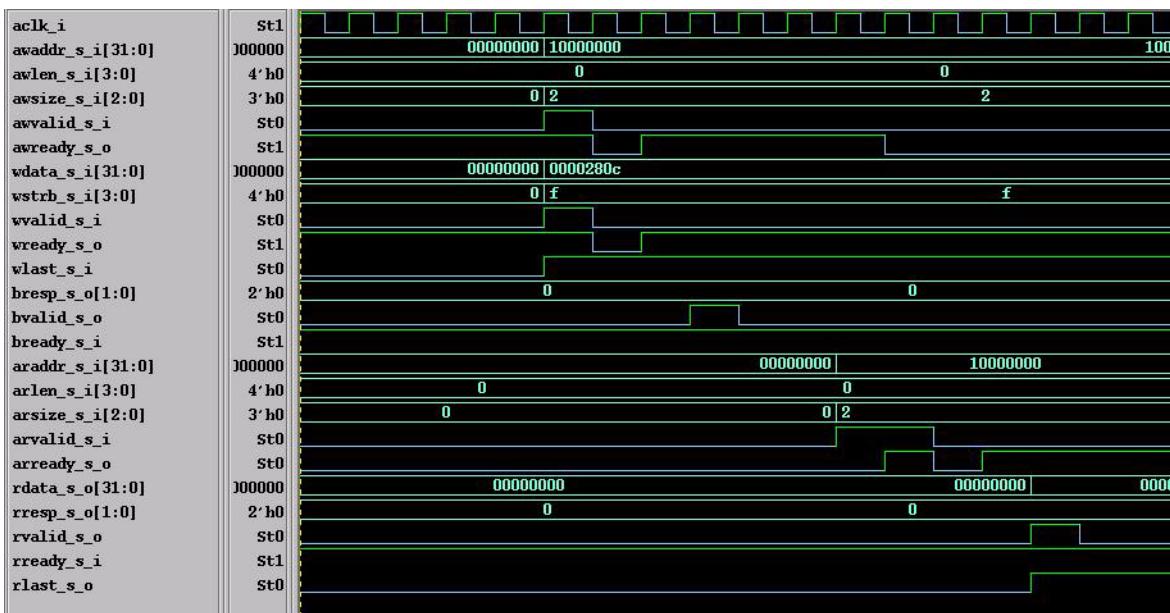
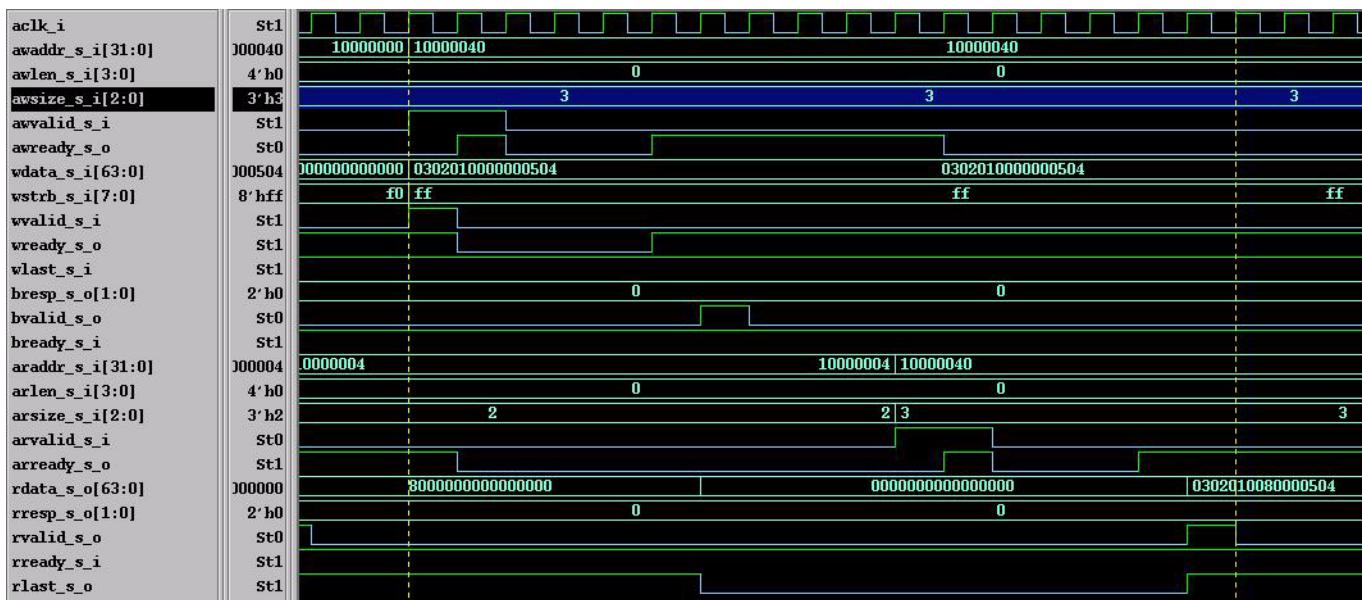
[Figure 3-15](#) shows the timing relationship on the AXI Slave interface for a burst-write transfer.

**Figure 3-15 32-bit AXI Slave Interface's Write Channel Timing**



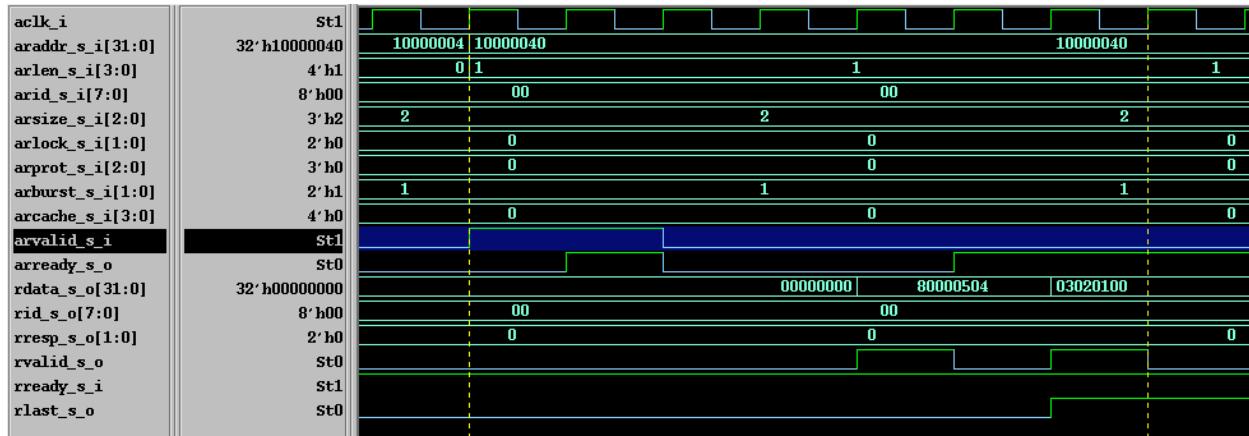
The AXI-slave interface receives a write burst of length 4 which gets accepted in 7 clocks. The respective write response takes an additional 3 clocks. Similarly a single write transfer will get completed in 4 clocks as shown in [Figure 3-16](#).

The AXI-slave interface supports all types of byte/half-word/word/etc burst-size transfers. The latencies may vary depending on the address offset and data-bus width, as shown in [Figure 3-17](#) for a 64-bit AXI Slave interface.

**Figure 3-16 Single Write and Read Transfer****Figure 3-17 Latencies for a 64-bit AXI Slave Interface**

In Figure 3-19, the AXI slave receives a read request of burst length 2. The latency for the first data to be output on the AXI bus is 4 clocks while the remaining beats are output 2 clocks after the previous beat is accepted by the AXI master. In the AXI-read channel, the latencies depend on the burst-size, data-bus width and the address offset.

**Figure 3-18 AXI Slave Receives a Read Request of Burst Length 2**



#### 3.4.14 AXI Low Power Interface

The GMAC-AXI core support the low-power interface defined by AXI specifications. Bits[31:30] of the AXI Bus Mode Register (see “[Register 10 \(AXI Bus Mode Register\)](#)” on page [273](#)) control the behavior of the GMAC-AXI in the low-power mode. When the EN\_LPI bit is enabled and the host requests a low-power initiation, the GMAC-AXI checks for the IDLE status of the data-paths before giving the response. In other words, when both the MAC transmitter and Receiver are in IDLE, both DMA engines are in idle and there is no data in the Transmit or Receive data-path/FIFOs for a continuous period of 16 clock-cycles (aclk\_i), then it will accept the low-power request and goes into low-power mode in which the host can remove the AXI-clock. If the GMAC goes to an inactive state within 15 clocks after the request, then the GMAC-AXI will wait for confirmation of the inactive state for a further period of 16 clocks before giving the response. If some activity is found in the GMAC during this period of 16 clock cycles, it will respond by denying the low-power request.

In the low-power mode, GMAC-AXI can initiate a request to come out of low-power mode when any packet is being received by the MAC Receiver or when a magic packet/remote-wake-up packet is received (as controlled by Bit[30] of the AXI Bus mode register). This request signal assertion (captive\_o) is synchronous to clk\_rx\_i in such cases, as aclk\_i is assumed to be absent.

## 3.5 DMA Controller

The DMA has independent Transmit and Receive engines, and a CSR space. The Transmit Engine transfers data from system memory to the device port (MTL), while the Receive Engine transfers data from the device port to system memory. The controller utilizes descriptors to efficiently move data from source to destination with minimal Host CPU intervention. The DMA is designed for packet-oriented data transfers such as frames in Ethernet. The controller can be programmed to interrupt the Host CPU for situations such as Frame Transmit and Receive transfer completion, and other normal/error conditions.

The DMA and the Host driver communicate through two data structures:

- ❖ Control and Status registers (CSR)
- ❖ Descriptor lists and data buffers

Control and Status registers are described in detail in “[Registers](#)” on page [247](#). Descriptors are described in detail in “[Descriptors](#)” on page [337](#).



**Note** Starting with Release 3.30a, you can select an alternative descriptor structure during RTL configuration. The control bits in this descriptor structure are reassigned so that the application can use a larger buffer size (8 KB). A detailed bit map of this alternative descriptor structure is provided in “[Descriptors](#)” on page [337](#). All descriptions in [Section 3.5](#) refer to the default descriptor structure, not this new alternative. If you are using the alternate descriptor structure, ignore the descriptor-specific mapping in [Section 3.5](#) and refer to the alternate descriptor-specific bit maps.

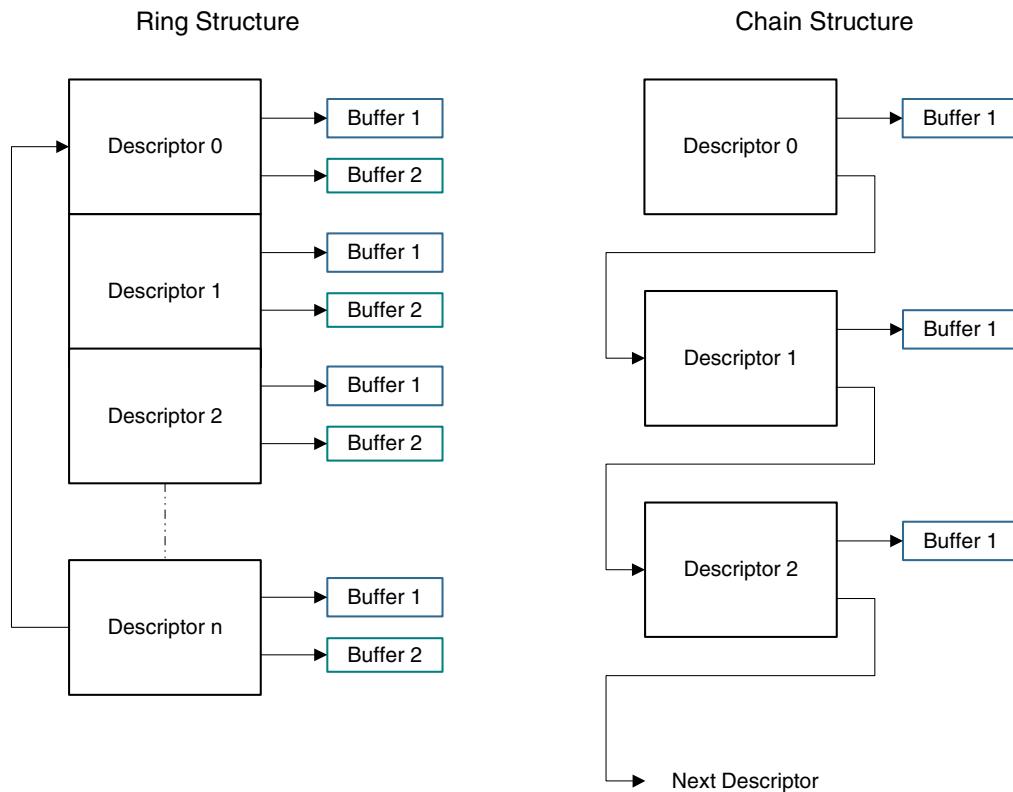
The DMA transfers data frames received by the core to the Receive Buffer in the Host memory, and Transmit data frames from the Transmit Buffer in the Host memory. Descriptors that reside in the Host memory act as pointers to these buffers.

There are two descriptor lists; one for reception, and one for transmission. The base address of each list is written into DMA Registers 3 and 4, respectively. A descriptor list is forward linked (either implicitly or explicitly). The last descriptor may point back to the first entry to create a ring structure. Explicit chaining of descriptors is accomplished by setting the second address chained in both Receive and Transmit descriptors (RDES1[24] and TDES1[24]). The descriptor lists resides in the Host physical memory address space. Each descriptor can point to a maximum of two buffers. This enables two buffers to be used, physically addressed, rather than contiguous buffers in memory.

A data buffer resides in the Host physical memory space, and consists of an entire frame or part of a frame, but cannot exceed a single frame. Buffers contain only data, buffer status is maintained in the descriptor. Data chaining refers to frames that span multiple data buffers. However, a single descriptor cannot span multiple frames. The DMA will skip to the next frame buffer when end-of-frame is detected. Data chaining can be enabled or disabled.

The descriptor ring and chain structure is shown in [Figure 3-19](#).

**Figure 3-19 Descriptor Ring and Chain Structure**



### 3.5.1 Initialization

Initialization for the GMAC is as follows.

1. Write to DMA Register 0 to set Host bus access parameters.
2. Write to DMA Register 7 to mask unnecessary interrupt causes.
3. The software driver creates the Transmit and Receive descriptor lists. Then it writes to both DMA Register 3 and DMA Register 4, providing the DMA with the starting address of each list.
4. Write to GMAC Registers 1, 2, and 3 for desired filtering options.
5. Write to GMAC Register 0 to configure and enable the Transmit and Receive operating modes. The PS and DM bits are set based on the auto-negotiation result (read from the PHY).
6. Write to DMA Register 6 to set bits 13 and 1 to start transmission and reception.
7. The Transmit and Receive engines enter the Running state and attempt to acquire descriptors from the respective descriptor lists. The Receive and Transmit engines then begin processing Receive and Transmit operations. The Transmit and Receive processes are independent of each other and can be started or stopped separately.

### 3.5.1.1 Host Bus Burst Access

The DMA will attempt to execute fixed-length Burst transfers on the AHB/AXI Master interface if configured to do so (FB bit of DMA Register 0). The maximum Burst length is indicated and limited by the PBL field (DMA Register 0[13:8]). The Receive and Transmit descriptors are always accessed in the maximum possible (limited by PBL or 16 \* 8 / bus width) burst-size for the 16-bytes to be read.

The Transmit DMA will initiate a data transfer only when sufficient space to accommodate the configured burst is available in MTL Transmit FIFO or the number of bytes till the end of frame (when it is less than the configured burst-length). The DMA will indicate the start address and the number of transfers required to the AHB/AXI Master Interface. When the AHB/AXI Interface is configured for fixed-length burst, then it will transfer data using the best combination of INCR4/8/16 and SINGLE transactions. Otherwise (no fixed-length burst), it will transfer data using INCR (undefined length) and SINGLE transactions.

The Receive DMA will initiate a data transfer only when sufficient data to accommodate the configured burst is available in MTL Receive FIFO or when the end of frame (when it is less than the configured burst-length) is detected in the Receive FIFO. The DMA will indicate the start address and the number of transfers required to the AHB/AXI Master Interface. When the AHB/AXI Interface is configured for fixed-length burst, then it will transfer data using the best combination of INCR4/8/16 and SINGLE transactions. If the end-of frame is reached before the fixed-burst ends on the AHB/AXI interface, then dummy transfers are performed in-order to complete the fixed-burst. Otherwise (FB bit of DMA Register 0 is reset), it will transfer data using INCR (undefined length) and SINGLE transactions.

When the AHB/AXI interface is configured for address-aligned beats, both DMA engines ensure that the first burst transfer the AHB/AXI initiates is less than or equal to the size of the configured PBL. Thus, all subsequent beats start at an address that is aligned to the configured PBL. The DMA can only align the address for beats up to size 16 (for PBL > 16), because the AHB/AXI interface does not support more than INCR16.

### 3.5.1.2 Host Data Buffer Alignment

The Transmit and Receive data buffers do not have any restrictions on start address alignment. For example, in systems with 32-bit memory, the start address for the buffers can be aligned to any of the four bytes. However, the DMA always initiates transfers with address aligned to the bus width with dummy data for the byte lanes not required. This typically happens during the transfer of the beginning or end of an Ethernet frame.

#### Example 3-1 Buffer Read

If the Transmit buffer address is 32'h00000FF2 (for 32-bit data bus), and 15 bytes need to be transferred, then the DMA will read five full words from address 32'h00000FF0, but when transferring data to the MTL Transmit FIFO, the extra bytes (the first two bytes) will be dropped or ignored. Similarly, the last 3 bytes of the last transfer will also be ignored. The DMA always ensures it transfers a full 32-bit data to the MTL Transmit FIFO, unless it is the end-of-frame.

#### Example 3-2 Buffer Write

If the Receive buffer address is 32'h0000FF2 (for 64-bit data bus) and 16 bytes of a received frame need to be transferred, then the DMA will write 3 full words from address 32'h00000FF0. But the first 2 bytes of first transfer and the last 6 bytes of the third transfer will have dummy data.

### 3.5.1.3 Buffer Size Calculations

The DMA does not update the size fields in the Transmit and Receive descriptors. The DMA updates only the status fields (RDES and TDES) of the descriptors. The driver has to perform the size calculations.

The transmit DMA transfers the exact number of bytes (indicated by buffer size field of TDES1) towards the GMAC core. If a descriptor is marked as first (FS bit of TDES1 is set), then the DMA marks the first transfer from the buffer as the start of frame. If a descriptor is marked as last (LS bit of TDES1), then the DMA marks the last transfer from that data buffer as the end-of frame to the MTL.

The Receive DMA transfers data to a buffer until the buffer is full or the end-of frame is received from the MTL. If a descriptor is not marked as last (LS bit of RDES0), then the descriptor's corresponding buffer(s) are full and the amount of valid data in a buffer is accurately indicated by its buffer size field minus the data buffer pointer offset when the FS bit of that descriptor is set. The offset is zero when the data buffer pointer is aligned to the data bus width. If a descriptor is marked as last, then the buffer may not be full (as indicated by the buffer size in RDES1). To compute the amount of valid data in this final buffer, the driver must read the frame length (FL bits of RDES0[29:16]) and subtract the sum of the buffer sizes of the preceding buffers in this frame. The Receive DMA always transfers the start of next frame with a new descriptor.



**Note** Even when the start address of a receive buffer is not aligned to the system bus's data width, the system should allocate a receive buffer of a size aligned to the system bus width. For example, if the system allocates a 1,024-byte (1 KB) receive buffer starting from address 0x1000, the software can program the buffer start address in the Receive descriptor to have a 0x1002 offset. The Receive DMA writes the frame to this buffer with dummy data in the first two locations (0x1000 and 0x1001). The actual frame is written from location 0x1002. Thus, the actual useful space in this buffer is 1,022 bytes, even though the buffer size is programmed as 1,024 bytes, due to the start address offset.

### 3.5.1.4 DMA Arbiter for GMAC-DMA and GMAC-AHB Cores

The arbiter inside the DMA module performs the arbitration between the Transmit and Receive channel accesses to the AHB Master interface. Two types of arbitrations are possible: round-robin, and fixed-priority.

When round-robin arbitration is selected (DA bit of DMA Register 0 is reset), the arbiter allocates the data bus in the ratio set by the PR bits of DMA Register 0, when both Transmit and Receive DMAs are requesting for access simultaneously. When the DA bit is set, the Receive DMA always gets priority over the Transmit DMA for data access.

## 3.5.2 Transmission

### 3.5.2.1 TxDMA Operation: Default (Non-OSF) Mode

The Transmit DMA engine in default mode proceeds as follows:

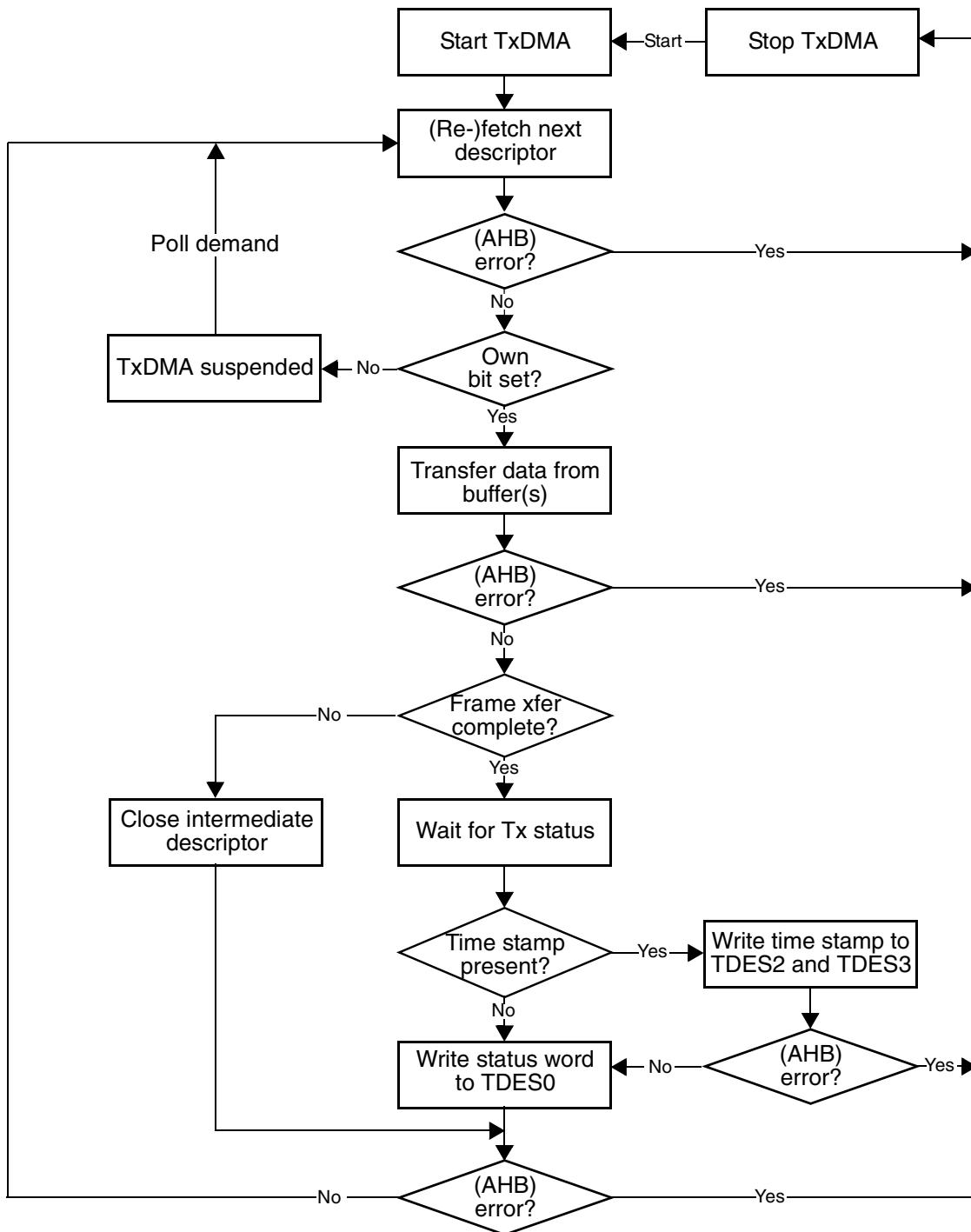
1. The Host sets up the transmit descriptor (TDES0-TDES3) and sets the Own bit (TDES0[31]) after setting up the corresponding data buffer(s) with Ethernet Frame data.
2. Once the ST bit (DMA Register 6[13]) is set, the DMA enters the Run state.
3. While in the Run state, the DMA polls the Transmit Descriptor list for frames requiring transmission. After polling starts, it continues in either sequential descriptor ring order or chained order. If the DMA detects a descriptor flagged as owned by the Host, or if an error condition occurs, transmission is suspended and both the Transmit Buffer Unavailable (DMA Register 5[2]) and

Normal Interrupt Summary (DMA Register 5[16]) bits are set. The Transmit Engine proceeds to [Step 9](#).

4. If the acquired descriptor is flagged as owned by DMA ( $TDES0[31] = 1'b1$ ), the DMA decodes the Transmit Data Buffer address from the acquired descriptor.
5. The DMA fetches the Transmit data from the Host memory and transfers the data to the MTL for transmission.
6. If an Ethernet frame is stored over data buffers in multiple descriptors, the DMA closes the intermediate descriptor and fetches the next descriptor. Steps [3](#), [4](#), and [5](#) are repeated until the end-of-Ethernet-frame data is transferred to the MTL.
7. When frame transmission is complete, if IEEE 1588 time stamping was enabled for the frame (as indicated in the transmit status) the time-stamp value obtained from MTL is written to the transmit descriptor ( $TDES2$  and  $TDES3$ ) that contains the end-of-frame buffer. The status information is then written to this transmit descriptor ( $TDES0$ ). Because the Own bit is cleared during this step, the Host now owns this descriptor. If time stamping was not enabled for this frame, the DMA does not alter the contents of  $TDES2$  and  $TDES3$ .
8. Transmit Interrupt (DMA Register 5[0]) is set after completing transmission of a frame that has Interrupt on Completion ( $TDES1[31]$ ) set in its Last Descriptor. The DMA engine then returns to [Step 3](#).
9. In the Suspend state, the DMA tries to re-acquire the descriptor (and thereby return to [Step 3](#)) when it receives a Transmit Poll demand and the Underflow Interrupt Status bit is cleared.

The TxDMA transmission flow in default mode is shown in [Figure 3-20](#).

**Figure 3-20 TxDMA Operation in Default Mode**



### 3.5.2.2 TxDMA Operation: OSF Mode

While in the Run state, the transmit process can simultaneously acquire two frames without closing the Status descriptor of the first (if the OSF bit is set in DMA Register 6[2]). As the transmit process finishes transferring the first frame, it immediately polls the Transmit Descriptor list for the second frame. If the second frame is valid, the transmit process transfers this frame before writing the first frame's status information.

In OSF mode, the Run state Transmit DMA operates in the following sequence:

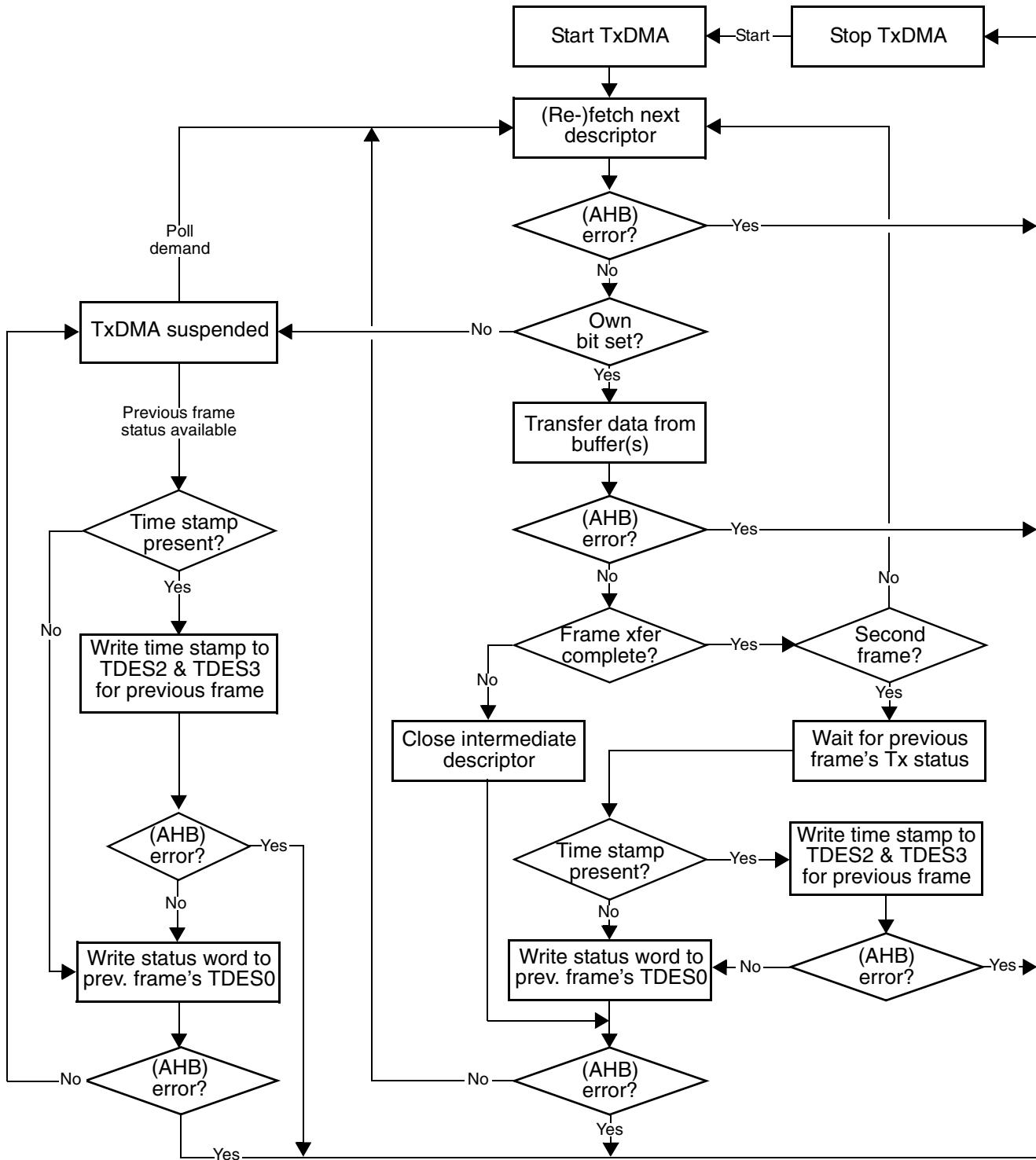
1. The DMA operates as described in [steps 1–6](#) of the TxDMA (default mode).
2. Without closing the previous frame's last descriptor, the DMA fetches the next descriptor.
3. If the DMA owns the acquired descriptor, the DMA decodes the transmit buffer address in this descriptor. If the DMA does not own the descriptor, the DMA goes into Suspend mode and skips to [Step 7](#).
4. The DMA fetches the Transmit frame from the Host memory and transfers the frame to the MTL until the End-of-Frame data is transferred, closing the intermediate descriptors if this frame is split across multiple descriptors.
5. The DMA waits for the previous frame's frame transmission status and time stamp. Once the status is available, the DMA writes the time stamp to TDES2 and TDES3, if such time stamp was captured (as indicated by a status bit). The DMA then writes the status, with a cleared Own bit, to the corresponding TDES0, thus closing the descriptor. If time stamping was not enabled for the previous frame, the DMA does not alter the contents of TDES2 and TDES3.
6. If enabled, the Transmit interrupt is set, the DMA fetches the next descriptor, then proceeds to [Step 3](#) (when Status is normal). If the previous transmission status shows an underflow error, the DMA goes into Suspend mode ([Step 7](#)).
7. In Suspend mode, if a pending status and time stamp are received from the MTL, the DMA writes the time stamp (if enabled for the current frame) to TDES2 and TDES3, then writes the status to the corresponding TDES0. It then sets relevant interrupts and returns to Suspend mode.
8. The DMA can exit Suspend mode and enter the Run state (go to [Step 1](#) or [Step 2](#) depending on pending status) only after receiving a Transmit Poll demand (DMA Register 1).



**Note** As the DMA fetches the next descriptor in advance before closing the current descriptor, the descriptor chain should have more than 2 different descriptors for correct and proper operation.

The basic flow is charted in [Figure 3-21](#).

**Figure 3-21 TxDMA Operation in OSF Mode**



### 3.5.2.3 Transmit Frame Processing

The Transmit DMA expects that the data buffers contain complete Ethernet frames, excluding preamble, pad bytes, and FCS fields. The DA, SA, and Type/Len fields contain valid data. If the Transmit Descriptor indicates that the MAC core must disable CRC or PAD insertion, the buffer must have complete Ethernet frames (excluding preamble), including the CRC bytes.

Frames can be data-chained and can span several buffers. Frames must be delimited by the First Descriptor (TDES1[29]) and the Last Descriptor (TDES1[30]), respectively.

As transmission starts, the First Descriptor must have (TDES1[29]) set. When this occurs, frame data transfers from the Host buffer to the MTL Transmit FIFO. Concurrently, if the current frame has the Last Descriptor (TDES1[30]) clear, the Transmit Process attempts to acquire the Next Descriptor. The Transmit Process expects this descriptor to have TDES1[29] clear. If TDES1[30] is clear, it indicates an intermediary buffer. If TDES1[30] is set, it indicates the last buffer of the frame.

After the last buffer of the frame has been transmitted, the DMA writes back the final status information to the Transmit Descriptor 0 (TDES0) word of the descriptor that has the last segment set in Transmit Descriptor 1 (TDES1[30]). At this time, if Interrupt on Completion (TDES1[31]) was set, Transmit Interrupt (DMA Register 5[0]) is set, the Next Descriptor is fetched, and the process repeats.

Actual frame transmission begins after the MTL Transmit FIFO has reached either a programmable transmit threshold (DMA Register 6[16:14]), or a full frame is contained in the FIFO. There is also an option for Store and Forward Mode (DMA Register 6[21]). Descriptors are released (Own bit TDES0[31] clears) when the DMA finishes transferring the frame.

### 3.5.2.4 Transmit Polling Suspended

Transmit polling can be suspended by either of the following conditions:

- ❖ The DMA detects a descriptor owned by the Host (TDES0[31]=0). To resume, the driver must give descriptor ownership to the DMA and then issue a Poll Demand command.
- ❖ A frame transmission is aborted when a transmit error due to underflow is detected. The appropriate Transmit Descriptor 0 (TDES0) bit is set.

If the second condition occur, both Abnormal Interrupt Summary (DMA Register 5[15]) and Transmit Underflow bits (DMA Register 5 [5]) are set, and the information is written to Transmit Descriptor 0, causing the suspension. If the DMA goes into SUSPEND state due to the first condition, then both Normal Interrupt Summary (DMA Register 5 [16]) and Transmit Buffer Unavailable (DMA Register 5 [2]) are set.

In both cases, the position in the Transmit List is retained. The retained position is that of the descriptor following the Last Descriptor closed by the DMA.

The driver must explicitly issue a Transmit Poll Demand command after rectifying the suspension cause.

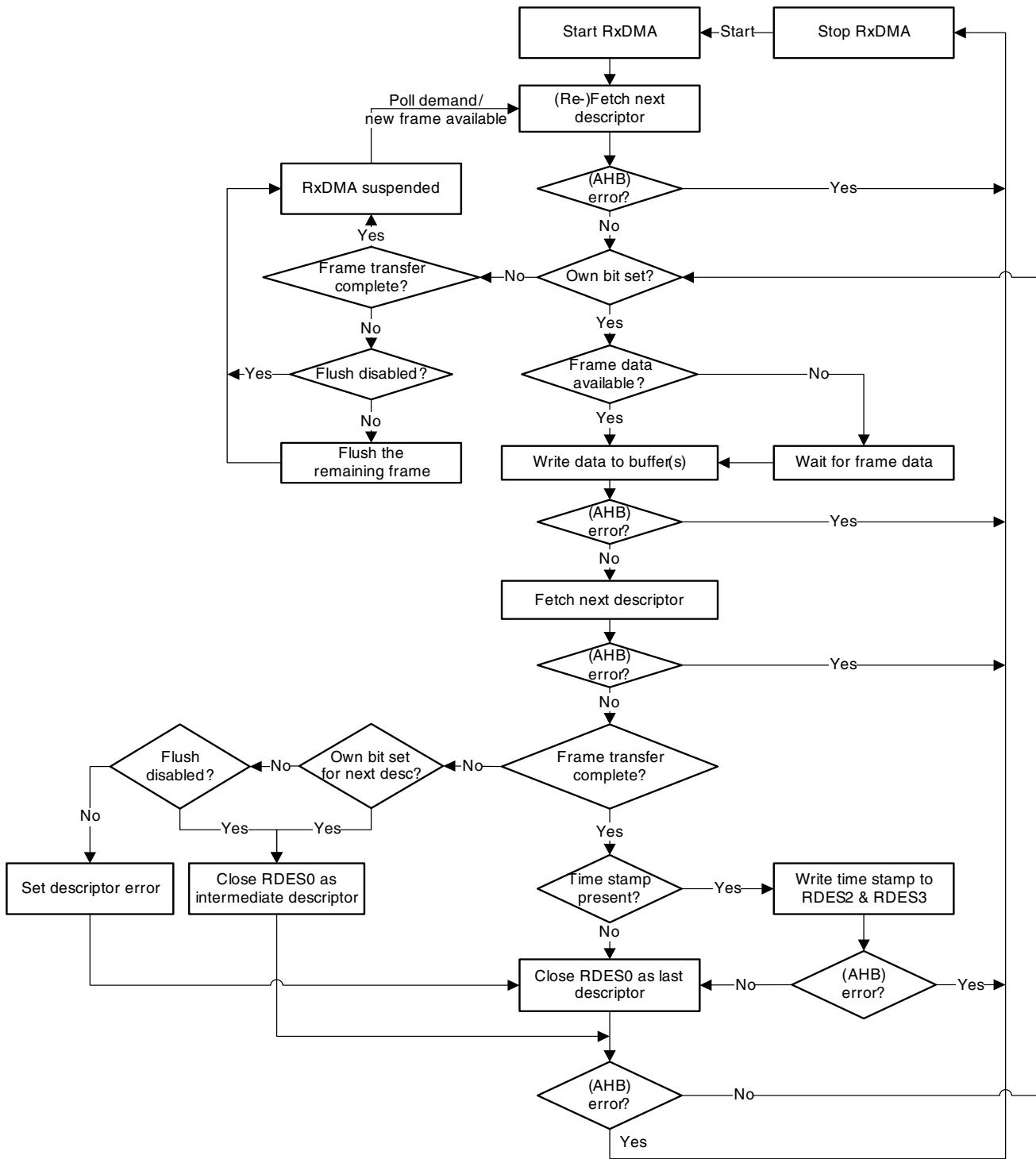
## 3.5.3 Reception

The Receive DMA engine's reception sequence is depicted in [Figure 3-22](#) and proceeds as follows:

1. The host sets up Receive descriptors (RDES0-RDES3) and sets the Own bit (RDES0[31]).
2. Once the SR (DMA Register 6[1]) bit is set, the DMA enters the Run state. While in the Run state, the DMA polls the Receive Descriptor list, attempting to acquire free descriptors. If the fetched descriptor is not free (is owned by the host), the DMA enters the Suspend state and jumps to [Step 9](#).
3. The DMA decodes the receive data buffer address from the acquired descriptors.
4. Incoming frames are processed and placed in the acquired descriptor's data buffers.

5. When the buffer is full or the frame transfer is complete, the Receive engine fetches the next descriptor.
6. If the current frame transfer is complete, the DMA proceeds to [Step 7](#). If the DMA does not own the next fetched descriptor and the frame transfer is not complete (EOF is not yet transferred), the DMA sets the Descriptor Error bit in the RDES0 (unless flushing is disabled). The DMA closes the current descriptor (clears the Own bit) and marks it as intermediate by clearing the Last Segment (LS) bit in the RDES0 value (marks it as Last Descriptor if flushing is not disabled), then proceeds to [Step 8](#). If the DMA does own the next descriptor but the current frame transfer is not complete, the DMA closes the current descriptor as intermediate and reverts to [Step 4](#).
7. If IEEE 1588 time stamping is enabled, the DMA writes the time stamp (if available) to the current descriptor's RDES2 and RDES3. It then takes the receive frame's status from the MTL and writes the status word to the current descriptor's RDES0, with the Own bit cleared and the Last Segment bit set.
8. The Receive engine checks the latest descriptor's Own bit. If the host owns the descriptor (Own bit is 1'b0) the Receive Buffer Unavailable bit (Register 5[7]) is set and the DMA Receive engine enters the Suspended state ([Step 9](#)). If the DMA owns the descriptor, the engine returns to [Step 4](#) and awaits the next frame.
9. Before the Receive engine enters the Suspend state, partial frames are flushed from the Receive FIFO (You can control flushing using Bit 24 of DMA Register 6).
10. The Receive DMA exits the Suspend state when a Receive Poll demand is given or the start of next frame is available from the MTL's Receive FIFO. The engine proceeds to [Step 2](#) and refetches the next descriptor.

**Figure 3-22 Receive DMA Operation**



The DMA does not acknowledge accepting the status from the MTL until it has completed the time stamp write-back and is ready to perform status write-back to the descriptor.

If software has enabled time stamping through CSR, when a valid time stamp value is not available for the frame (for example, because the receive FIFO was full before the time stamp could be written to it), the

DMA writes all-ones to RDES2 and RDES3. Otherwise (that is, if time stamping is not enabled), the RDES2 and RDES3 remain unchanged.

### 3.5.3.1 Receive Descriptor Acquisition

The Receive Engine always attempts to acquire an extra descriptor in anticipation of an incoming frame. Descriptor acquisition is attempted if any of the following conditions is satisfied:

- ❖ The receive Start/Stop bit (Register 6[1]) has been set immediately after being placed in the Run state.
- ❖ The data buffer of current descriptor is full before the frame ends for the current transfer.
- ❖ The controller has completed frame reception, but the current Receive Descriptor is not yet closed.
- ❖ The receive process has been suspended because of a host-owned buffer (RDES0[31] = 0) and a new frame is received.
- ❖ A Receive poll demand has been issued.

### 3.5.3.2 Receive Frame Processing

The GMAC transfers the received frames to the Host memory only when the frame passes the address filter and frame size is greater than or equal to configurable threshold bytes set for the Receive FIFO of MTL, or when the complete frame is written to the FIFO in Store-and-Forward mode.

If the frame fails the address filtering, it is dropped in the GMAC block itself (unless Receive All GMAC Register 1[31] bit is set). Frames that are shorter than 64 bytes, because of collision or premature termination, can be purged from the MTL Receive FIFO.

After 64 (configurable threshold) bytes have been received, the MTL block requests the DMA block to begin transferring the frame data to the Receive Buffer pointed to by the current descriptor. The DMA sets First Descriptor (RDES0[9]) after the DMA Host Interface (AHB/AXI or MDC) becomes ready to receive a data transfer (if DMA is not fetching transmit data from the host), to delimit the frame. The descriptors are released when the Own (RDES[31]) bit is reset to 1'b0, either as the Data buffer fills up or as the last segment of the frame is transferred to the Receive buffer. If the frame is contained in a single descriptor, both Last Descriptor (RDES[8]) and First Descriptor (RDES[9]) are set.

The DMA fetches the next descriptor, sets the Last Descriptor (RDES[8]) bit, and releases the RDES0 status bits in the previous frame descriptor. Then the DMA sets Receive Interrupt (Register 5[6]). The same process repeats unless the DMA encounters a descriptor flagged as being owned by the host. If this occurs, the Receive Process sets Receive Buffer Unavailable (Register 5[7]) and then enters the Suspend state. The position in the receive list is retained.

### 3.5.3.3 Receive Process Suspended

If a new Receive frame arrives while the Receive Process is in Suspend state, the DMA refetches the current descriptor in the Host memory. If the descriptor is now owned by the DMA, the Receive Process re-enters the Run state and starts frame reception. If the descriptor is still owned by the host, by default, the DMA discards the current frame at the top of the MTL Rx FIFO and increments the missed frame counter. If more than one frame is stored in the MTL Rx FIFO, the process repeats.

The discarding or flushing of the frame at the top of the MTL Rx FIFO can be avoided by setting Operation Mode register bit 24 (DFF). In such conditions, the receive process sets the Receive Buffer Unavailable status and returns to the Suspend state.

### 3.5.4 Interrupts

Interrupts can be generated as a result of various events. [DMA Register 5](#) contains all the bits that might cause an interrupt. [DMA Register 7](#) contains an enable bit for each of the events that can cause an interrupt.

There are two groups of interrupts, Normal and Abnormal, as described in DMA Register 5. Interrupts are cleared by writing a 1'b1 to the corresponding bit position. When all the enabled interrupts within a group are cleared, the corresponding summary bit is cleared. When both the summary bits are cleared, the interrupt signal `sbd_intr_o` is deasserted. If the GMAC core is the cause for assertion of the interrupt, then any of the GLI, GMI, or GPI bits of DMA Register 5 will be set high.



**Note** DMA Register 5 is the (interrupt) status register. The interrupt pin (`sbd_intr_o`) will be asserted due to any event in this status register only if the corresponding interrupt enable bit is set in DMA Register 7.

Interrupts are not queued and if the interrupt event occurs before the driver has responded to it, no additional interrupts are generated. For example, Receive Interrupt (DMA Register 5[6]) indicates that one or more frames was transferred to the Host buffer. The driver must scan all descriptors, from the last recorded position to the first one owned by the DMA.

An interrupt is generated only once for simultaneous, multiple events. The driver must scan DMA Register 5 for the cause of the interrupt. The interrupt is not generated again unless a new interrupting event occurs, after the driver has cleared the appropriate bit in DMA Register 5. For example, the controller generates a Receive interrupt (DMA Register 5[6]) and the driver begins reading DMA Register 5. Next, Receive Buffer Unavailable (DMA Register 5[7]) occurs. The driver clears the Receive interrupt. Even then, the `sbd_intr_o` signal is not deasserted, due to the active or pending Receive Buffer Unavailable interrupt.

An interrupt timer (see “[Register 9 \(Receive Interrupt Watchdog Timer Register\)](#)” on page [273](#)) is given for flexible control of Receive Interrupt (DMA register 5[6]). When this Interrupt timer is programmed with a non-zero value, it will get activated as soon as the RxDMA completes a transfer of a received frame to system memory without asserting the Receive Interrupt because it is not enabled in the corresponding Receive Descriptor (RDES1[31] in Table 7-3). When this timer runs out as per the programmed value, RI bit is set and the interrupt is asserted if the corresponding RI is enabled in DMA Register 7. This timer gets disabled before it runs out, when a frame is transferred to memory and the RI is set because it is enabled for that descriptor.

### 3.5.5 Error Response to DMA

For any data transfer initiated by a DMA channel, if the slave replies with an error response, that DMA stops all operations and updates the error bits and the Fatal Bus Error bit in the Status register (DMA Register 5). That DMA controller can resume operation only after soft resetting or hard resetting the core and re-initializing the DMA. This DMA behavior is true for non-AHB/AXI interfaced DMAs (“[Data Transfer With Error](#)” on page [80](#)) that receive an error response through the `mdc_error_i` signal.

### 3.5.6 DMA Native Interface

The Subsystem can be configured (GMAC-DMA) to have DMA with a native FIFO-like interface on the application side and an MCI (Optionally APB) interface for CSR port. [Figure 4-3](#) provides the top-level I/O diagram for this GMAC-DMA configuration. All of the functions and features of the DMA remain the same as described above.

On the native interface, the DMA initiates data or descriptor transfers using a simple hardware protocol. The DMA always starts a transaction with the proper values on the address bus, transfer-size, burst-length

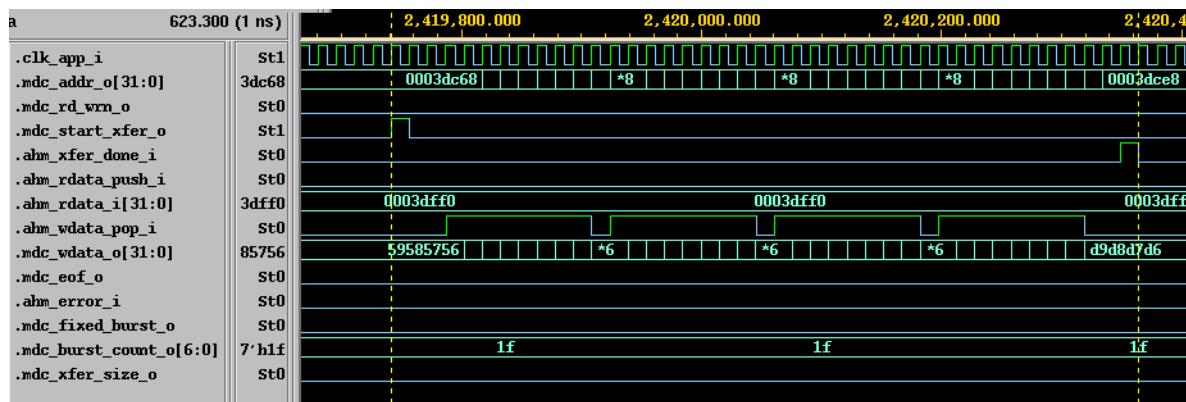
and type of transfer (whether read or write). If it is a read transfer, then the application can push valid data by asserting signal `mdc_rdata_push_i`. For write transfers, the application can accept the data from DMA by asserting signal `mdc_wdata_pop_i`. The application can pause the transfers by deasserting the respective push or pop control signals.

### Note

The values on the address bus, the burst-length, and the transfer size are not updated immediately on the receipt of push or pop signals. The address, burst-length, and transfer size are valid only at the start of the transfer.

In certain accesses, the burst-length may not even change during the transfers as shown in [Figure 3-23](#). The application must assert the `mdc_rdata_push_i` or `mdc_wdata_pop_i` for clocks equal to the burst length requested at the start of transfer (unless a premature write transfer is indicated by DMA as explained in next section) before asserting the `mdc_xfer_done_i` signal for proper transfer of data.

**Figure 3-23 mdc\_burst\_count\_o**



After transferring all of the requested data, application must terminate the DMA transaction with signal `mdc_xfer_done_i`. The application can indicate an error by activating signal `mdc_error_i`, and the respective DMA (Rx or Tx) will go to the Fatal-error state and all operations stop. The application must provide a software-reset to restart that DMA's operation.

#### 3.5.6.1 Write Data Transfer

[Figure 3-24](#) shows the transfer sequence of a data buffer being written to application memory.

1. The subsystem initiates the transaction by asserting `mdc_start_xfer_o` (pulse), along with the host memory address (`mdc_addr_o`), read/write control signal (`mdc_rd_wrn_o = 0`), number of beats (`mdc_burst_count_o`), and the transfer size (`mdc_xfer_size_o`). It also drives the first data beat of the burst on the `mdc_wdata_o` bus.
2. The application asserts signal `mdc_wdata_pop_i` to accept the data.
3. If the application is not ready to handle the data, it can deassert `mdc_wdata_pop_i` and delay the data transfer. The DMA will hold the values on the data bus, when the “pop” control is deasserted.
4. After all of the data has been transferred, as indicated by `mdc_burst_count_o`, the application must end the DMA transfer by asserting signal `mdc_xfer_done_i` (pulse).
5. The DMA can prematurely end the burst transfer due to an End-of-Frame being transferred to the application. The DMA marks the End-of-Frame data on `mdc_wdata_o` with the assertion of `mdc_eof_o`. The application must then complete the transaction by asserting signal `mdc_xfer_done_i`.

after accepting the End-of-Frame data even if the initial burst-length given at start of transaction is not completed.

**Figure 3-24 Write Data Transfer Timing (1 of 3)**

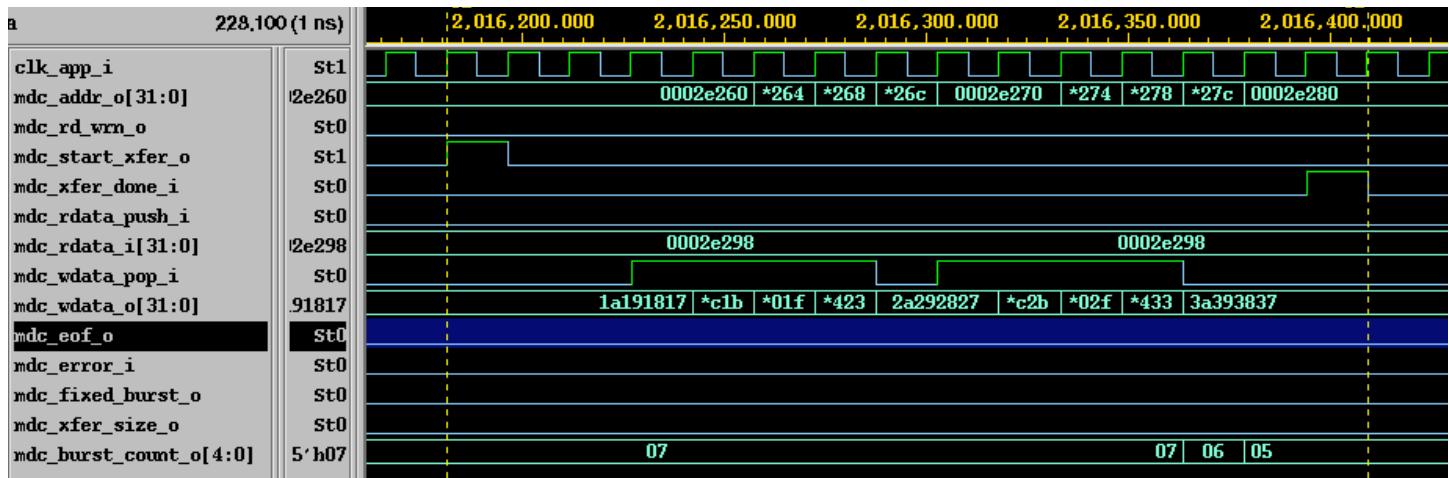


Figure 3-25 shows the transfer sequence when a data buffer is written to application memory with the burst transfer ending prematurely due to the assertion of mdc\_eof\_o. You can observe that the DMA terminates the burst by asserting mdc\_eof\_o after completing 3 cycles of the requested 4-cycle burst. The host has to then respond with the assertion of mdc\_xfer\_done\_i without asserting any more mdc\_wdata\_pop\_i.

**Figure 3-25 Write Data Transfer Timing (2 of 3)**

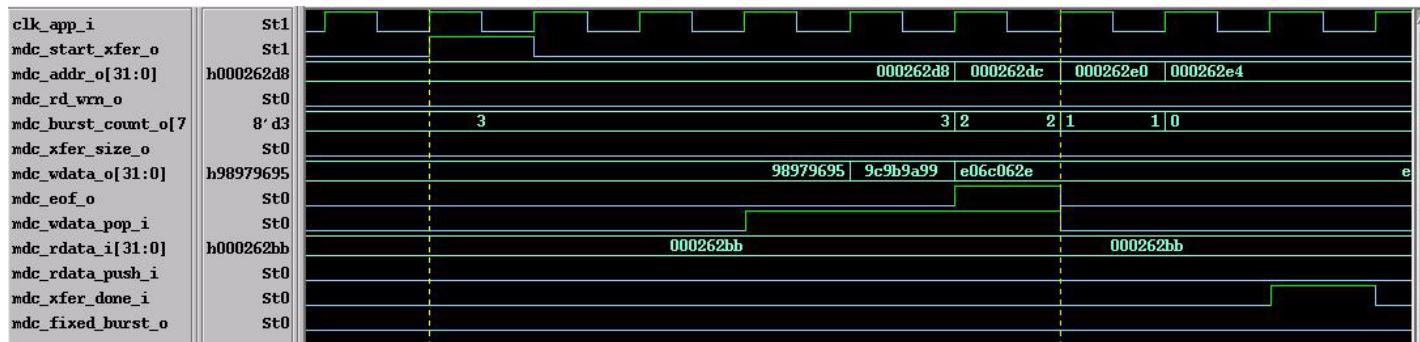
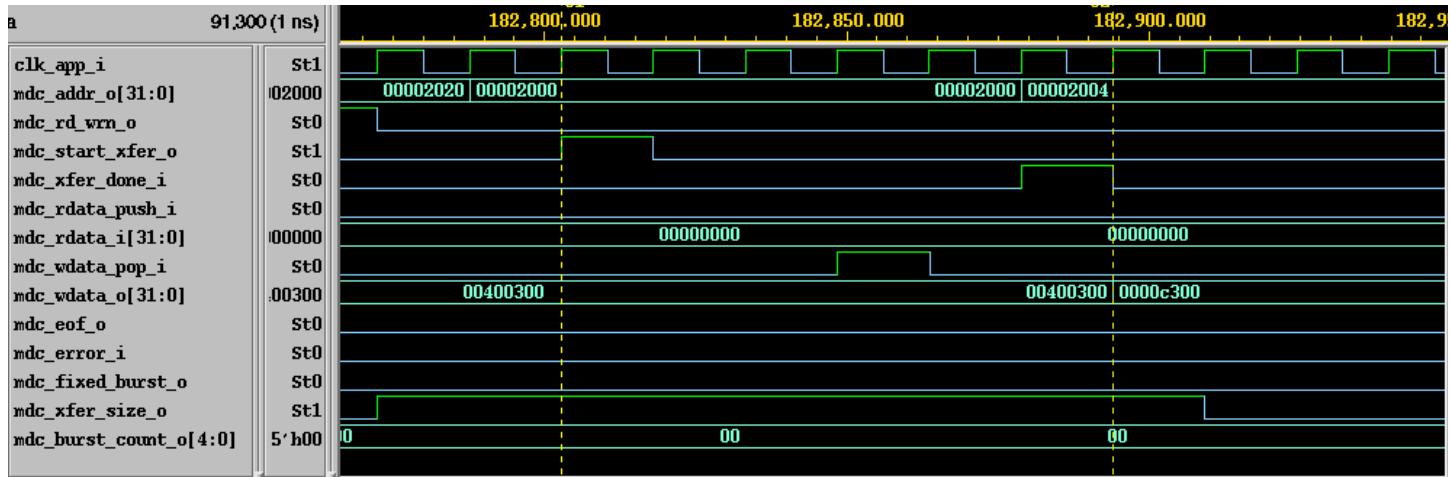


Figure 3-26 shows the transfer sequence when a descriptor is written to application memory.

**Figure 3-26 Write Data Transfer Timing (3 of 3)**



### 3.5.6.2 Read Data Transfer

Figure 3-27 shows the transfer sequence during data buffer being read from application memory.

1. The subsystem initiates the transaction by asserting mdc\_start\_xfer\_o (pulse), along with the host memory address (mdc\_addr\_o), read/write control signal (mdc\_rd\_wrn\_o = 1), number of beats (mdc\_burst\_count\_o), and the transfer size (mdc\_xfer\_size\_o).
2. The DMA accepts the data on the mdc\_rdata\_i bus whenever the application asserts signal mdc\_rdata\_push\_i.
3. If the application is not ready with the data, it can deassert mdc\_rdata\_push\_i and delay the data transfer.
4. After all of the data has been transferred as indicated by mdc\_burst\_count\_o, the application must end the DMA transfer by asserting signal mdc\_xfer\_done\_i (pulse).

**Figure 3-27 Read Data Transfer Timing (1 of 2)**

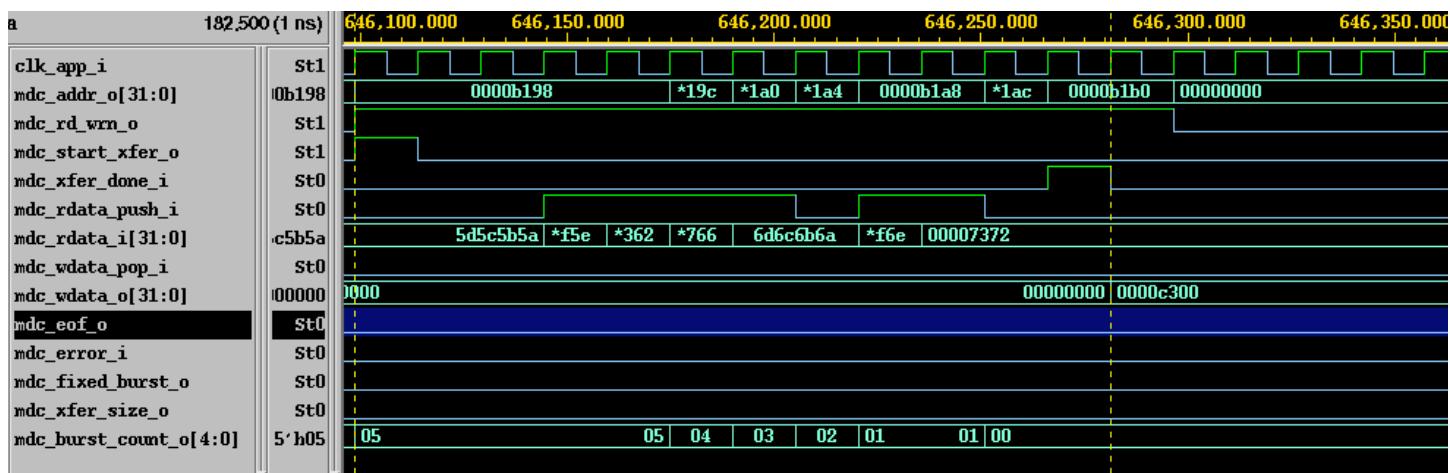
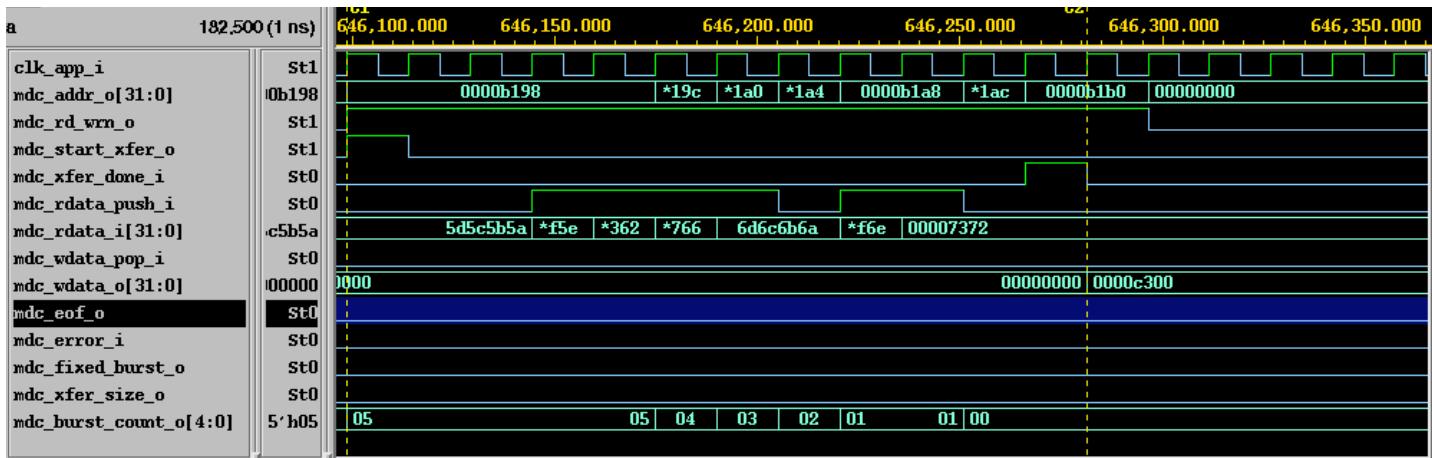


Figure 3-28 shows the transfer sequence during a descriptor being read from application memory.

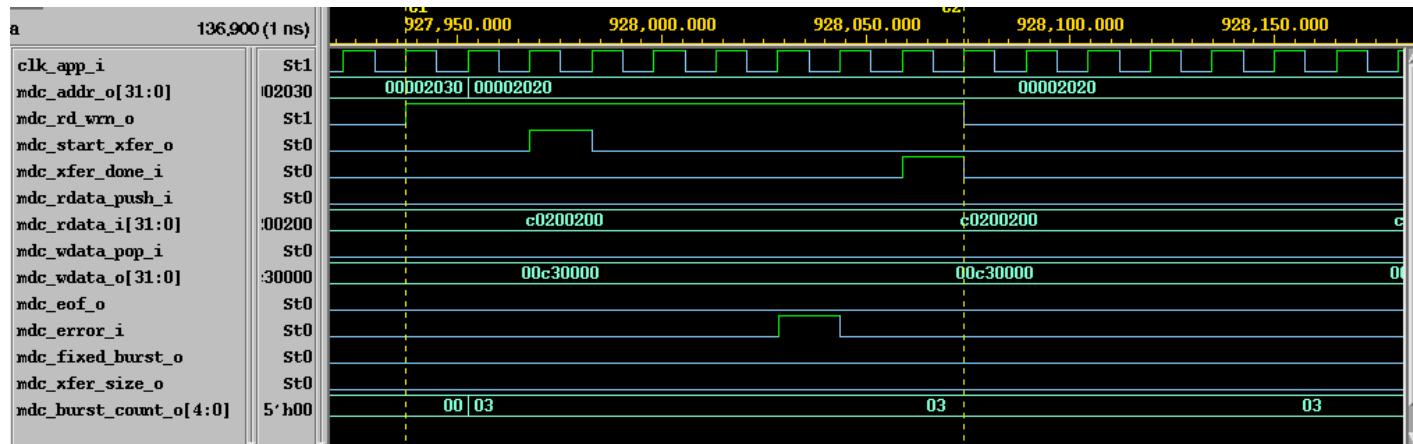
**Figure 3-28 Read Data Transfer Timing (2 of 2)**



### 3.5.6.3 Data Transfer With Error

For any write or read transfer started by the DMA, the application can assert mdc\_error\_i if it cannot complete the transaction (see Figure 3-29). The application should also complete the transfer by asserting mdc\_xfer\_done\_i in a different clock cycle. If the application asserts both the signals together, the DMA may restart the transfer to the same address and the behavior of the DMA is not predictable.

**Figure 3-29 Data Transfer With Error Timing**



## 3.6 MAC Transaction Layer (MTL)

The MAC Transaction Layer provides FIFO memory to buffer and regulate the frames between the application system memory and the GMAC core. It also enables the data to be transferred between the application clock domain and the GMAC clock domains. The MTL layer has 2 data paths, namely the Transmit path and the Receive Path. The data path for both directions is 32/64/128-bit wide and operates with a simple FIFO protocol.

The GMAC-MTL communicates with the application side with the Application Transmit Interface (ATI), Application Receive Interface (ARI), and the MAC Control Interface (MCI). The default and optional I/O signals are described in “[Signal Descriptions](#)” on page [179](#)

### 3.6.1 Transmit Path

In GMAC-AHB/AXI and GMAC-DMA configurations, the DMA controls all transactions for the transmit path through the ATI. Ethernet frames read from the system memory is pushed into the FIFO by the DMA. The frame is then popped out and transferred to the GMAC core when triggered. When the end-of-frame is transferred, the status of the transmission is taken from the GMAC core and transferred back to the DMA.

The Transmit FIFO has a default depth of 2K bytes. FIFO-fill level is indicated to the DMA so that it can initiate a data fetch in required bursts from the system memory, using the AHB/AXI interface. The data from the AHB/AXI Master interface is pushed into the FIFO with the appropriate byte lanes qualified by the DMA. The DMA also indicates the start-of-frame (SOF) and end-of-frame (EOF) transfers along with a few sideband signals controlling the pad-insertion/CRC generation for that frame in the GMAC core.

Per-frame control bits, such as Automatic Pad/CRC Stripping disable, time stamp capture, and so forth are taken as sideband control inputs on the ATI, stored in a separate register FIFO, and passed on to the core transmitter when the corresponding frame data is read from the Transmit FIFO.

There are two modes of operation for popping data towards the GMAC core. In Threshold mode, as soon as the number of bytes in the FIFO crosses the configured threshold level (or when the end-of-frame is written before the threshold is crossed), the data is ready to be popped out and forwarded to the GMAC core. The threshold level is configured using the TTC bits of DMA Register 0.

In Store-and-Forward mode, only after a complete frame is stored in the FIFO, the MTL pops the frame towards the GMAC core. If the Tx FIFO size is smaller than the Ethernet frame to be transmitted (such as Jumbo frame), then the MTL pops the frame towards the GMAC core when the Tx FIFO becomes almost full or when the ATI watermark becomes low. The watermark becomes low when the requested FIFO does not have space to accommodate the requested burst-length on the ATI. Therefore, the MTL never stalls in Store and Forward mode even if the Ethernet frame length is bigger than the Tx FIFO depth.

The application can flush the Transmit FIFO of all contents by setting the FTF (DMA Register 6[20]) bit. This bit is self-clearing and initializes the FIFO pointers to the default state. If the FTF bit is set during a frame transfer from the MTL to the GMAC core, then the MTL stops further transfer as the FIFO is considered to be empty. Hence an underflow event occurs at the GMAC transmitter and the corresponding Status word is forwarded to the DMA.

“[Initialization](#)” on page [81](#) through “[Transmit Status Word](#)” on page [83](#) detail initialization and transmit operations for the MTL Layer. Timing diagrams are provided in “[Transmit Path Timing](#)” on page [85](#).

#### 3.6.1.1 Initialization

Upon reset, the MTL is ready to manage the flow of data to and from the DMA and the GMAC.

There are no requirements for enabling the MTL. However, the GMAC block and the DMA controller must be enabled individually through their respective CSRs.

### 3.6.1.2 Single-Packet Transmit Operation

During a transmit operation, the MTL block is slaved to the DMA controller. The general sequence of events for a transmit operation is as follows.

1. If the system has data to be transferred, the DMA controller, if enabled, fetches data from the Host through the AHB/AXI Master interface and starts forwarding it to the MTL. The MTL pushes the data received from the DMA into the FIFO. It continues to receive the data until the end-of frame of the frame is transferred.
2. The data is taken out of the FIFO and sent to the MAC by the FIFO controller engine. When the threshold level is crossed or a full packet of data is received into the FIFO, the MTL pops out the frame data and drives them to the GMAC core. The engine continues to transfer data from the FIFO until a complete packet has been transferred to the MAC. Upon completion of the frame, the MTL receives the Status from the GMAC and then notifies the DMA controller

### 3.6.1.3 Transmit Operation—Two Packets in the Buffer

1. Because the DMA must update the descriptor status before releasing it to the Host, there can be at the most two frames inside a transmit FIFO. The second frame will be fetched by the DMA and put into the FIFO only if the OSF (Operate on Second Frame bit is set). If this bit is not set, the next frame will be fetched from the memory only after the MAC has completely processed the frame and the DMA has released the descriptors.
2. If the OSF bit is set, the DMA starts fetching the second frame immediately after completing the transfer of the first frame to the FIFO. It does not wait for the status to be updated. The MTL, in the meantime, receives the second frame into the FIFO while transmitting the first frame. As soon as the first frame has been transferred and the status is received from the MAC, the MTL pushes it to the DMA. If the DMA has already completed sending the second packet to the MTL, it must wait for the status of the first packet before proceeding to the next frame.

### 3.6.1.4 Transmit Operation—Multiple Packets in Buffer

In GMAC-MTL configuration, the transmit FIFO can be configured to accept more than 2 packets at a time. This option limits the number of status words that can be stored in the MTL before it is transferred to the DMA/host. By default, this number is limited to 2 but can be configured for 4 or 8 as well. Once the MTL FIFO accepts the number of frames equal to the status FIFO depth, it will stop accepting further frames unless the transmit Status that is given out and accepted by the host/DMA thus freeing up the space in this small FIFO.

### 3.6.1.5 Retransmission During Collision

While a frame is being transferred from the MTL to the GMAC, a collision event occurs on the GMAC line interface in Half-Duplex mode. The GMAC then indicates a retry attempt to the MTL by giving the status even before the end-of-frame is transferred from MTL. Then the MTL will enable the retransmission by popping out the frame again from the FIFO.

After more than 96 bytes (or 548 bytes in 1000-Mbps mode) are popped towards the GMAC core, the FIFO controller frees up that space and makes it available to the DMA to push in more data. This means that the retransmission is not possible after this threshold is crossed or when the GMAC core indicates a late-collision event.

### 3.6.1.6 Transmit FIFO Flush Operation

The GMAC provides a control to the software to flush the Transmit FIFO in the MTL layer through the use of Bit 20 of the Operation Mode register (see “[Register 6 \(Operation Mode Register\)](#)” on page [266](#)). The

Flush operation is immediate and the MTL clears the Tx FIFO and the corresponding pointers to the initial state even if it is in the middle of transferring a frame to the GMAC Core. The data which is already accepted by the MAC transmitter will not be flushed. It will be scheduled for transmission and will result in underflow as TxFIFO does not complete the transfer of rest of the frame. As in all underflow conditions, a runt frame will be transmitted and observed on the line. The status of such a frame will be marked with both Underflow and Frame Flush events (TDES0 bits 13 and 1).

The MTL layer also stops accepting any data from the application (DMA) during the Flush operation. It will generate and transfer Transmit Status Words to the application for the number of frames that is flushed inside the MTL (including partial frames). Frames that are completely flushed in the MTL will have the Frame Flush Status bit (TDES0 13) set. The MTL completes the Flush operation when the application (DMA) accepts all of the Status Words for the frames that were flushed, and then clears the Transmit FIFO Flush control register bit. At this point, the MTL starts accepting new frames from the application (DMA).

In GMAC-MTL configuration, an option is provided to have a sideband signal (`ati_txfifoflush_i`) initiate Flush operation. Whenever this input is sampled high, the `ati_rdy_o` signal is deasserted and the MTL Tx FIFO is flushed. When the Flush operation is complete, `ati_rdy_o` is asserted, indicating the MTL is ready to accept new frames. All data presented to the MTL after a Flush operation is discarded unless it starts with an SOF marker.

### 3.6.1.7 Transmit Status Word

At the end of transfer of the Ethernet frame to the GMAC core and after the core completes the transmission of the frame, the MTL outputs the transmit status (`ati_txstatus_o[23:0]`) to the application. This is indicated by an active `ati_txstatus_val_o` signal. The detailed description of the Transmit Status is the same as for bits [23:0] of TDES0, given in [Table 7-6](#).

If IEEE 1588 time stamping is enabled, the MTL returns specific frame's 64-bit time stamp, along with the ATI's transmit status. A valid time stamp is indicated by an active high on the `ati_txstatus_o[17]` bit.

### 3.6.1.8 Transmit Checksum Offload Engine

Communication protocols such as TCP and UDP implement checksum fields, which help determine the integrity of data transmitted over a network. Because the most widespread use of Ethernet is to encapsulate TCP and UDP over IP datagrams, the GMAC-UNIV has an optional Checksum Offload Engine (COE) to support checksum calculation and insertion in the transmit path, and error detection in the receive path. This engine is not present by default and is included when you select this module during RTL configuration (see ["Configuration: Creating the RTL" on page 30](#)). This section explains the operation of the Checksum Offload Engine for transmitted frames.



- The checksum for TCP, UDP, or ICMP is calculated over a complete frame, then inserted into its corresponding header field. Due to this requirement, this function is enabled only when the Transmit FIFO is configured for Store-and-Forward mode (that is, when the TSF bit is set in DMA Register 6). If the core is configured for Threshold (cut-through) mode, the Transmit COE is bypassed.
- You must make sure the Transmit FIFO is deep enough to store a complete frame before that frame is transferred to the GMAC Core transmitter. You must enable the checksum insertion only in frames of size less than FIFO depth - PBL (programmed burst-length in DMA Bus mode register or `ati_pbl_i` in GMAC-MTL) number of bytes even in Store-and-Forward mode. The reason is that the MTL TxFIFO will start reading once space is not available to accept the programmed burst-length of data, to avoid dead-lock. Once reading starts, then checksum insertion engine fails and consequently all succeeding frames may get corrupted due to improper recovery.

This module supports two types of checksum calculation and insertion. This checksum engine can be controlled for each frame by setting the CIC bits (Bits 28:27 of TDES1, described in “[Transmit Descriptor 1 \(TDES1\)](#)” on page 349) in GMAC-AHB or GMAC-DMA configurations. In GMAC-MTL configuration, the ati\_chksum\_ctrl\_i input pins ([Table 4-26](#)) controls the operation.



**Hint** See IETF specifications RFC 791, RFC 793, RFC 768, RFC 792, RFC 2460, and RFC 4443 for IPv4, TCP, UDP, ICMP, IPv6, and ICMPv6 packet header specifications, respectively.

### 3.6.1.8.1 IP Header Checksum Engine

In IPv4 datagrams, the integrity of the header fields is indicated by the 16-bit Header Checksum field (the eleventh and twelfth bytes of the IPv4 datagram). The COE detects an IPv4 datagram when the Ethernet frame’s Type field has the value 0x0800 and the IP datagram’s Version field has the value 0x4. The input frame’s checksum field is ignored during calculation and replaced with the calculated value.

IPv6 headers do not have a checksum field; thus, the COE does not modify IPv6 header fields.

The result of this IP header checksum calculation is indicated by the IP Header Error status bit in the Transmit status (Bit 16 in [Table 7-6](#)). This status bit is set whenever the values of the Ethernet Type field and the IP header’s Version field are not consistent, or when the Ethernet frame does not have enough data, as indicated by the IP header Length field.

In other words, this bit is set when an IP header error is asserted under the following circumstances:

**For IPv4 datagrams:**

- ❖ The received Ethernet type is 0x0800, but the IP header’s Version field does not equal 0x4
- ❖ The IPv4 Header Length field indicates a value less than 0x5 (20 bytes)
- ❖ The total frame length is less than the value given in the IPv4 Header Length field

**For IPv6 datagrams:**

- ❖ The Ethernet type is 0x86dd but the IP header Version field does not equal 0x6
- ❖ The frame ends before the IPv6 header (40 bytes) or extension header (as given in the corresponding Header Length field in an extension header) is completely received.

Even when the COE detects such an IP header error, it inserts an IPv4 header checksum if the Ethernet Type field indicates an IPv4 payload.

### 3.6.1.8.2 TCP/UDP/ICMP Checksum Engine

The TCP/UDP/ICMP Checksum Engine processes the IPv4 or IPv6 header (including extension headers) and determines whether the encapsulated payload is TCP, UDP, or ICMP.



- For non-TCP, -UDP, or -ICMP/ICMPv6 payloads, this checksum engine is bypassed and nothing further is modified in the frame.
- Fragmented IP frames (IPv4 or IPv6), IP frames with security features (such as an authentication header or encapsulated security payload), and IPv6 frames with routing headers are not processed by this engine, and therefore must be bypassed. In other words, payload checksum insertion must not be enabled for such frames.

The checksum is calculated for the TCP, UDP, or ICMP payload and inserted into its corresponding field in the header. This engine can work in the following two modes:

- ❖ In the first mode, the TCP, UDP, or ICMPv6 pseudo-header is not included in the checksum calculation and is assumed to be present in the input frame's Checksum field. This engine includes the Checksum field in the checksum calculation, then replaces the Checksum field with the final calculated checksum.
- ❖ In the second mode, the engine ignores the Checksum field, includes the TCP, UDP, or ICMPv6 pseudo-header data into the checksum calculation, and overwrites the checksum field with the final calculated value.



**Note** For ICMP-over-IPv4 packets, the Checksum field in the ICMP packet must always be 16'h0000 in both modes, because pseudo-headers are not defined for such packets. If it does not equal 16'h0000, an incorrect checksum may be inserted into the packet.

The result of this operation is indicated by the Payload Checksum Error status bit in the Transmit Status vector (Bit 12 in [Table 7-6](#)). This engine sets the Payload Checksum Error status bit when it detects that the frame has been forwarded to the MAC Transmitter engine in Store-and-Forward mode without the end-of-frame being written to the FIFO, or when the packet ends before the number of bytes indicated by the Payload Length field in the IP Header is received. When the packet is longer than the indicated payload length, the COE ignores them as stuff bytes, and no error is reported. When this engine detects the first type of error, it does not modify the TCP, UDP, or ICMP header. For the second error type, it still inserts the calculated checksum into the corresponding header field.

### 3.6.1.9 Transmit Path Timing

Timing specifications for the MAC Transaction layer (MTL) interface signals in the transmit path are illustrated in detail, in the following sections.

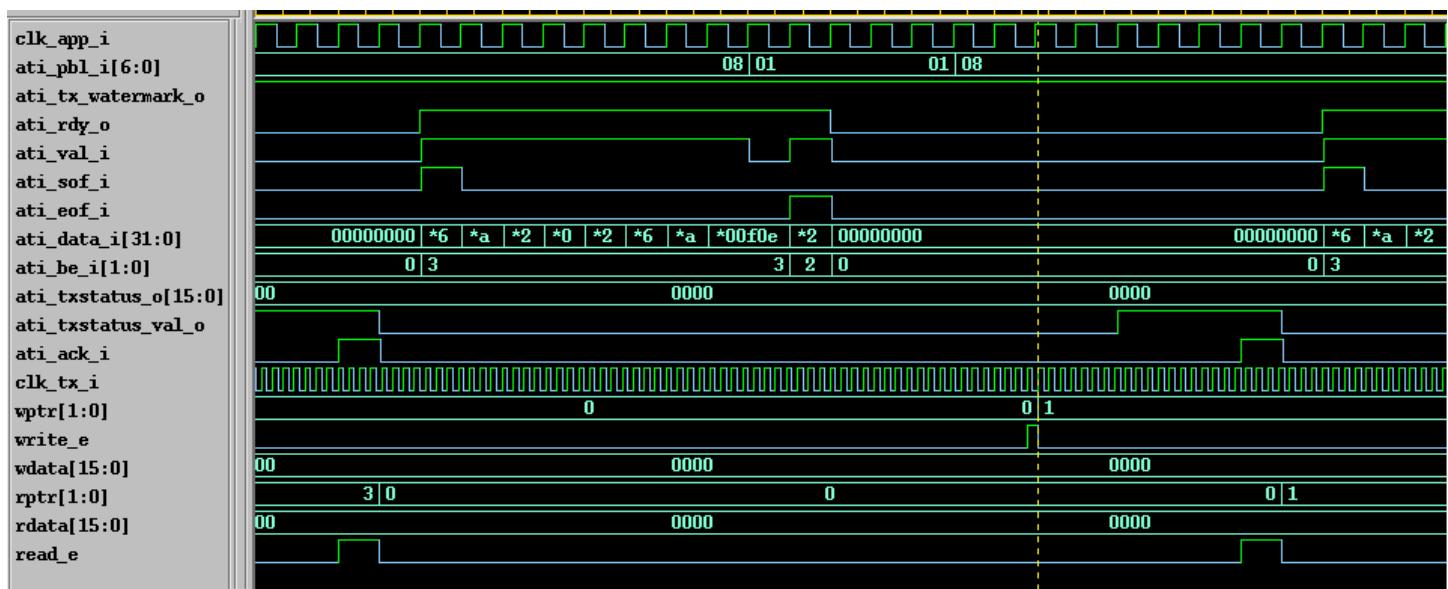
#### 3.6.1.9.1 Application Transmit Interface (ATI)

[Figure 3-30](#) illustrates the timing specifications for the Application Transmit Interface signals. The handshake mechanism involved in frame transmission is as follows:

1. The application, before starting a burst-transfer of data to ATI, should check whether the MTL Tx FIFO has space to accommodate the burst. The length of the proposed burst transfer is put on ati\_pbl\_i.
2. In response, the MTL asserts/deasserts ati\_txwatermark\_o one clock later, indicating whether it can accommodate the burst data transfer or not.
3. At any point during the transfer, the MTL can alert the application to stop the transfer by deasserting ati\_rdy\_o. Signal ati\_rdy\_o is deasserted only when FIFO is almost full, or when the Tx FIFO is being flushed. The Tx FIFO-full event will not occur if the application ensures that enough space is available before the start of data transfer, as described in [steps 1](#) and [2](#).
4. The application starts a frame transfer by asserting ati\_val\_i and ati\_sof\_i. The MTL accepts the data whenever ati\_rdy\_o and ati\_val\_i are asserted.
5. When the application wants to transfer the last bytes of a frame, ati\_eof\_i should be asserted along with the byte enables (ati\_be\_i), indicating the valid number of data bytes on the bus. It is mandatory that all data transfers except the EOF transfer have the byte-enables as all-ones.

6. After transferring the frame onto the Ethernet PHY, the GMAC gives the transmission status to the MTL (not shown in the diagram).
7. The MTL gives the transmit status of the frame back to the application through `ati_txstatus_o`, by asserting `ati_txstatus_val_o`. The application accepts the status by asserting `ati_ack_i`.
8. The application need not wait for steps 6 and 7 to start the transfer of next frame on the ATI. If `ati_rdy_o` is asserted, the application can transfer the next frame. In the figure, `wptr`, `rptr`, `write_e`, and `read_e` are the internal signals of the 2-deep asynchronous FIFO where the transmit status is stored. It can be seen that the status written in location 0x0 (`wptr` = 0) corresponds to the previous transmitted frame. The application reads that status and starts the transfer of next frame on the ATI.

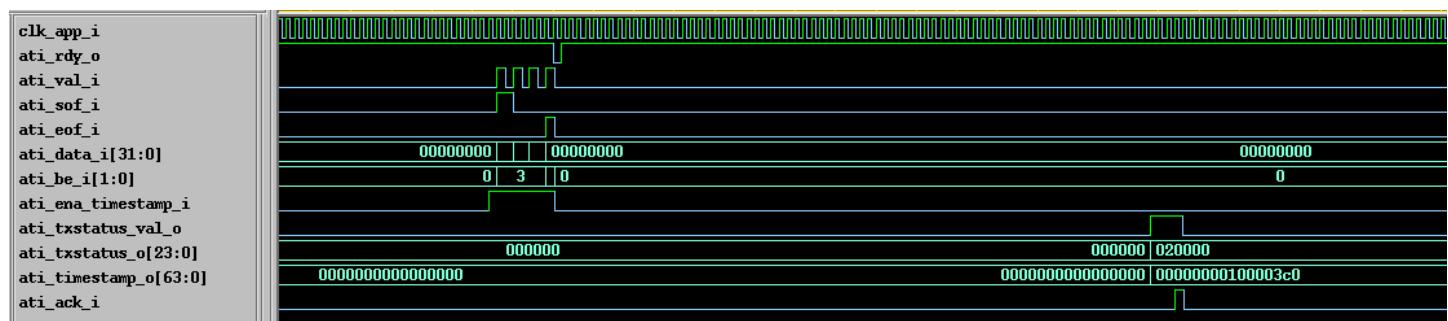
**Figure 3-30 Application Transmit Interface (ATI) Timing**



### 3.6.1.9.2 ATI With Time Stamping

Figure 3-31 shows the timing of the ATI when IEEE 1588 time stamping is enabled. The `ati_ena_timestamp_i` signal is sampled along with the `ati_sof_i` for an input frame. When that frame's status is given on `ati_txstatus_o`, the time stamp captured for that frame is output onto `ati_timestamp_o`. The time stamp's validity is indicated by the assertion of `ati_txstatus_o[17]` when the `ati_txstatus_val_o` is high.

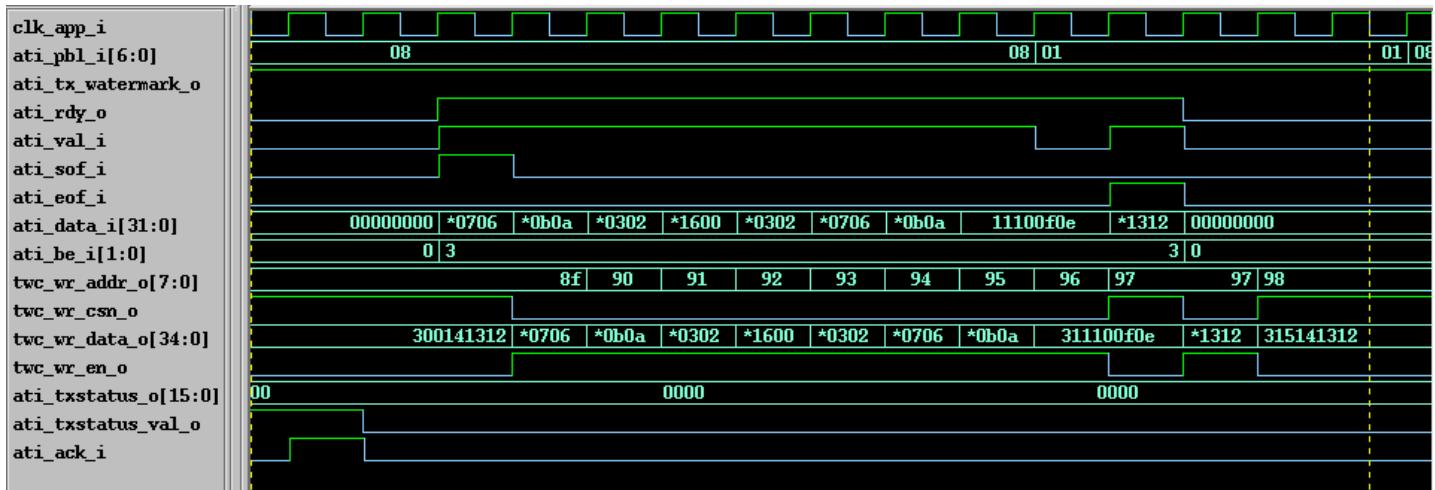
**Figure 3-31 ATI Time Stamp Timing**



### 3.6.1.9.3 Transmit FIFO Write Interface

The MTL Transmit FIFO controller transfers the data accepted by the ATI to the Tx FIFO implemented as external memory. Timing specifications for the memory interface signals (`tvc_wr_*` signals) are illustrated in [Figure 3-32](#), which also shows the timing relationship of the memory interface signals with respect to the ATI signals.

**Figure 3-32** Transmit FIFO Write Interface Timing



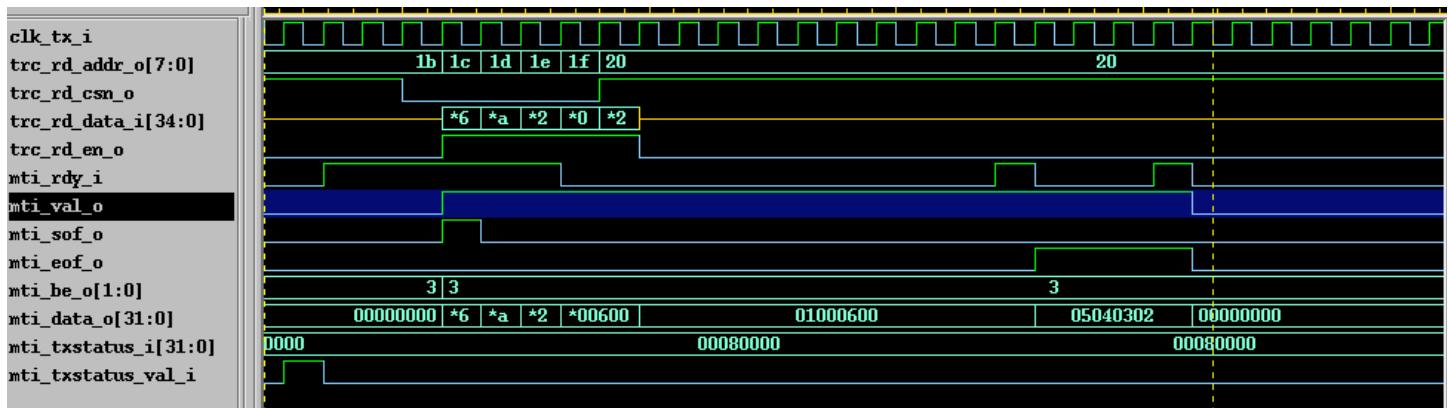
The external memory used for storing frame can be synchronous or asynchronous two-port RAM. The Transmit FIFO write interface timing shown above holds good for both synchronous and asynchronous two-port RAM.

### 3.6.1.9.4 Transmit FIFO Read Interface

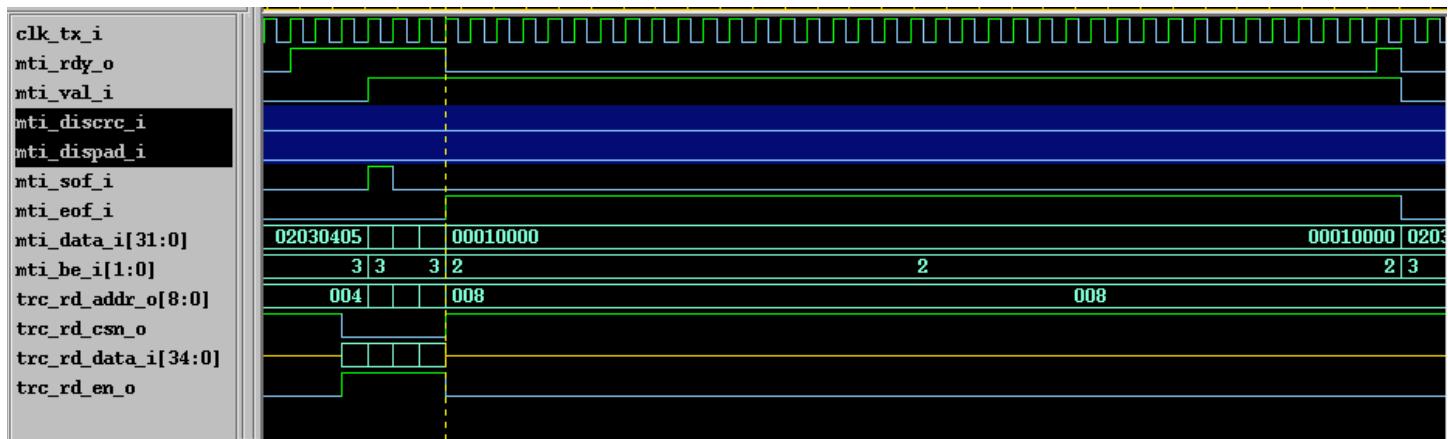
The Transmit FIFO Read controller reads the data from the Tx FIFO and transfers it to the GMAC Core on the MAC Transmit Interface (MTI). The timing relationship of the MTI signals with the Transmit FIFO Read Interface signals are shown in [Figure 3-33](#). The external memory of the Rx FIFO is configured as synchronous RAM in the figure.

The sequence of events during the transfer of a frame from Tx FIFO to the MTI is as follows:

1. When the Tx FIFO becomes non-empty (not shown in the diagram), the Read Controller starts the read transfers by asserting signals `trc_rd_addr_o` and `trc_rd_csn_o`. The assertion of `trc_rd_en_o` is delayed by 1 clock cycle as per synchronous RAM timing requirements.
2. The data is read on the `trc_rd_data_i` bus in the next clock cycle. As this is directly transferred to the `mti_data` bus to the GMAC Core (MTI), `mti_val_i` is also asserted. Signal `mti_sof_i` is also asserted along with `mti_val_i` in the figure, as the data corresponds to start of frame.
3. The Tx FIFO Read Controller can pre-fetch data from three locations (two in Asynchronous RAM port) so that it can sustain continuous transfers to the MTI interface without any delay.
4. If the MAC core accepts the data (active `mti_rdy_o`), more data is read from the FIFO. If the MAC deasserts `mti_rdy_o`, then the read operations are suspended in the next clock cycle and `trc_rd_csn_o` is deasserted.
5. Signals `mti_dispad_i` and `mti_discrc_i` are the sideband control signals used to direct the MAC to add pad-bytes and/or CRC. These signals are transferred from the ATI interface (`ati_dispad_i` and `ati_discrc_i`) to the MTI interface when `mti_sof_i` is active, through an internal asynchronous register FIFO.

**Figure 3-33 Transmit FIFO Read Interface Timing (1 of 2)**

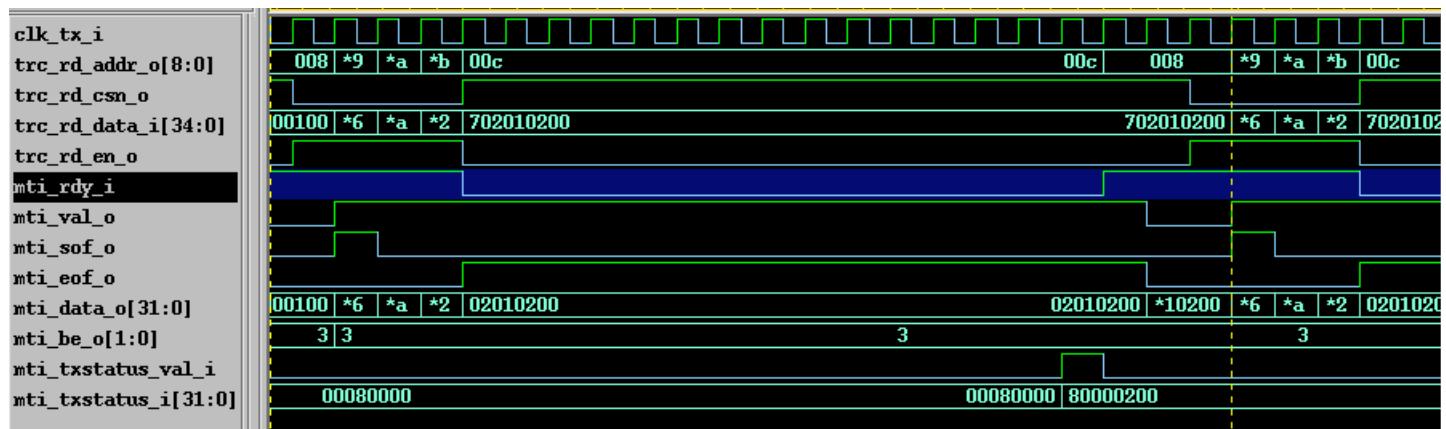
The asynchronous two-port RAM Tx FIFO read interface timing has subtle changes with respect to the synchronous RAM, and is shown in [Figure 3-34](#). In this mode, the read data (trc\_rd\_data\_i) is given out by the Tx FIFO on the assertion of the chip select and read enable signals (trc\_rd\_csn\_o and trc\_rd\_en\_o) in the same clock cycle. This data is registered and is given to the MTI interface in the next clock cycle.

**Figure 3-34 Transmit FIFO Read Interface Timing (2 of 2)**

### 3.6.1.9.5 Transmit Frame Retransmission

The MAC transmitter may abort the transmission of a frame due to collision, underflow, loss of carrier, and several other conditions. When frame transmission is aborted due to collision, the MAC requests retransmission of the frame. This scenario is illustrated with the following sequence of events, shown in [Figure 3-35](#).

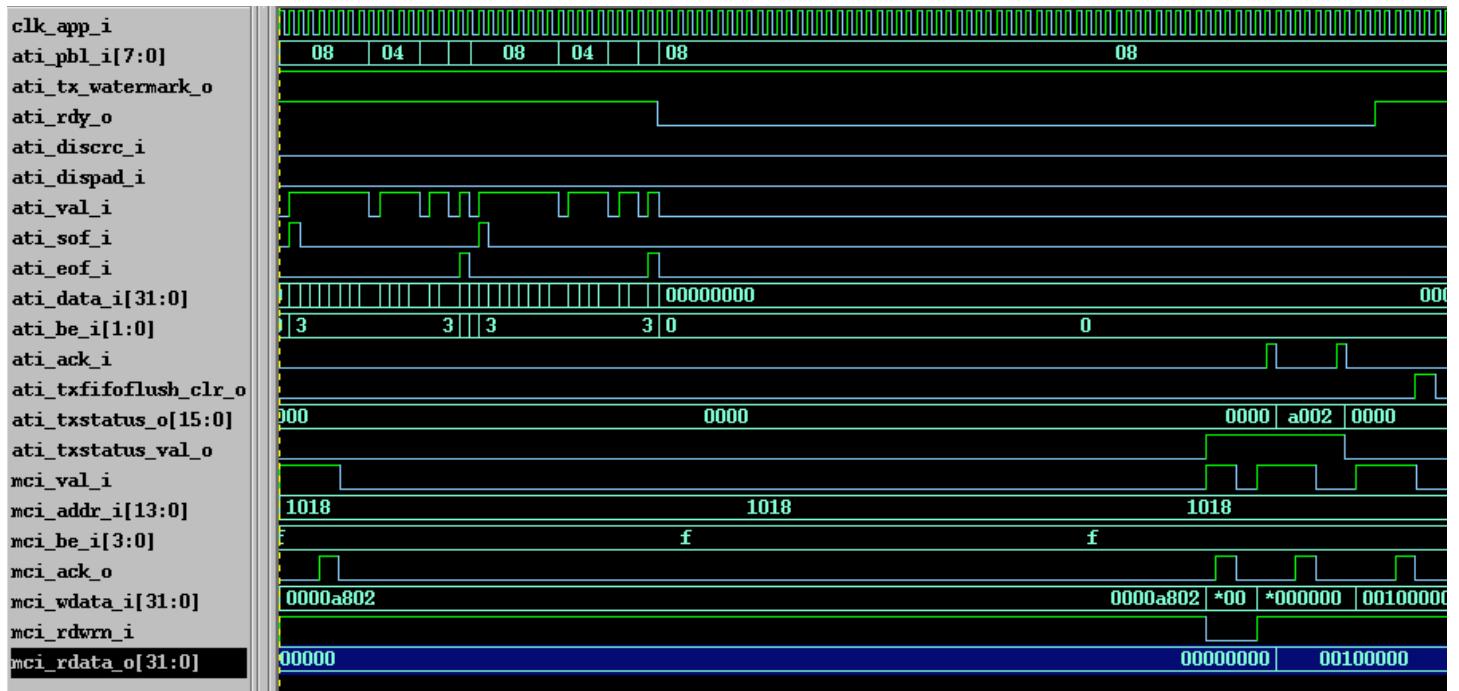
1. The MTL starts the transfer of a frame to the MAC by asserting mti\_val\_i and mti\_sof\_i.
2. Subsequently, the remaining data is sent upon the assertion of mti\_rdy\_o by the MAC.
3. When the MAC experiences a collision on the Ethernet bus (not shown in diagram), frame transmission is suspended first by keeping mti\_rdy\_o deasserted.
4. The MAC transfers the transmission status on mti\_txstatus, and requests retransmission by asserting Bit 31 of mti\_txstatus.
5. The MTL retransmits the frame. The Tx FIFO read address (trc\_rd\_addr) rewinds back to the start of frame address 0x008 for the retransmission.

**Figure 3-35** Transmit Frame Retransmission Timing

### 3.6.1.9.6 Transmit FIFO Flush Operation

The MTL allows the Tx FIFO to be flushed at any instance, through a CSR bit (Operation Mode Register[20]). [Figure 3-36](#) shows the scheme of things that occur after the FIFO Flush request from the application.

1. In the figure, two frames are pushed by the application into the Tx FIFO on the ATI interface.
2. The application performs a write to the Operation mode register (address = 1018) and sets the Flush Transmit FIFO bit.
3. The application reads back the same register to check the status of the Flush operation. mci\_rdata\_o[20] is asserted high during CSR read access, indicating the progress of the Flush operation.
4. Since the first frame is already read and transmitted by the MAC by this time, the MTL gives the transmit status of the first frame as normal. This is seen by the value of 0x0000 on the ati\_txstatus read with the first ati\_ack\_i pulse.
5. The MTL now flushes the second frame in the FIFO and subsequently, gives a Flush Frame status (0xa002) to the application on the ati\_txstatus bus.
6. The MTL indicates the completion of the Flush operation by asserting ati\_txifoflush\_clr (internal), which clears the Flush transmit FIFO bit in the OMR.

**Figure 3-36 Transmit FIFO Flush Timing**

### 3.6.2 Receive Path

This module receives the frames given out by the GMAC core and pushes them into the Rx FIFO. The status (fill level) of this FIFO is indicated to the DMA once it crosses the configured Receive threshold (RTC of DMA Register 6). The MTL also indicates the FIFO fill level so that the DMA can initiate pre-configured burst transfers towards the AHB interface.

[“Receive Operation” on page 90](#) through [“Receive Status Word” on page 92](#) detail receive operations for the MTL Layer. Timing diagrams are provided in [“Receive Path Timing Diagrams” on page 93](#).

#### 3.6.2.1 Receive Operation

During an Rx operation, the MTL is slaved to the GMAC. The general sequence of Receive operation events is as follows:

1. When the GMAC receives a frame, it pushes in data along with byte enables. The GMAC also indicates the SOF and EOF. The MTL accepts the data and pushes it into the Rx FIFO. After the EOF is transferred, the GMAC drives the status word, which is also pushed into the same Rx FIFO by the MTL.
2. When IEEE 1588 time stamping is enabled and the 64-bit time stamp is available along with the receive status, it is appended to the frame received from the GMAC and is pushed into the Rx FIFO before the corresponding receive status word is written. Thus, in 32-bit data bus mode, two additional locations per frame are taken for storing the time stamp in the Rx FIFO, while in 64-/128-bit mode, one additional location is taken.
3. The MTL\_RX engine takes the data out of the FIFO and sends it to the DMA. In the default Cut-Through mode, when 64 bytes (configured with RTC bits of DMA Register 6) or a full packet of data are received into the FIFO, the MTL\_RX engine pops out the data and indicates its availability to the DMA. Once the DMA initiates the transfer to the AHB/AI interface, the MTL\_RX engine continues to transfer data from the FIFO until a complete packet has been transferred. Upon completion of the

EOF frame transfer, the MTL pops out the status word and sends it to the DMA controller. If the 64-bit time stamp is read out before the receive status, it is marked by the assertion of the ari\_timestamp\_val\_o signal on the ARI. Otherwise, the status is given after the EOF data.



**Note** In 32-bit data bus mode, the time-stamp transfer takes two clock cycles and the lower 32-bit of the time-stamp is given out first. The status also may be extended to two cycles when Advanced Time-stamp feature is enabled.

4. In Rx FIFO Store-and-Forward mode (configured by the RSF bit of DMA Register 6), a frame is read out only after being written completely into the Receive FIFO. In this mode, all error frames are dropped (if the core is configured to do so) such that only valid frames are read out and forwarded to the application. In Cut-Through mode, some error frames are not dropped, because the error status is received at the end-of-frame, by which time the start of that frame has already been read out of the FIFO.

### 3.6.2.2 Receive Operation Multiframe Handling

Since the status is available immediately following the data, the MTL is capable of storing any number of frames into the FIFO, as long as it is not full.

### 3.6.2.3 GMAC Flow Control

The flow control operation of the GMAC can also be enabled (EFC bit of DMA Register 6) by the Rx FIFO control logic. The flow control signal to the GMAC is asserted whenever the Rx FIFO fill level crosses the configured threshold (RFA bits of DMA Register 6). This flow control signal is deasserted once the FIFO fill-level falls below the configured threshold (RFD bits). This operation is independent of whether the GMAC is configured for full-duplex or half-duplex.

The above hardware flow control operation is applicable only when the Rx FIFO is 4,096 or more bytes deep. A separate sideband flow control signal (sbd\_flowctrl\_i) is optionally provided at the top-level I/O for all GMAC-AHB, GMAC-AXI, GMAC-DMA and GMAC-MTL configurations. For 4,096-byte or larger Rx FIFOs, this sideband signal is logically ORed with the hardware flow control signal. Thus, flow control is enabled when either of these signals is asserted and disabled when both these signals are deasserted.

### 3.6.2.4 Error Handling

If the MTL Rx FIFO is full before it receives the EOF data from the GMAC, an overflow is declared, the whole frame (including the status word) is dropped, and the overflow counter in the DMA (Register 8) is incremented. This is true even if the Forward Error Frame (FEF bit of DMA Register 6) is set. If the start address of such a frame has already been transferred to the Read Controller, the rest of the frame is dropped and a dummy EOF is written to the FIFO along with the status word. The status will indicate a partial frame due to overflow. In such frames, the Frame Length field is invalid.

The MTL Rx Control logic can filter error and undersized frames, if enabled (using the DMA Register 6 FEF and FUF bits). If the start address of such a frame has already been transferred to the Rx FIFO Read Controller, that frame is not filtered. The start address of the frame is transferred to the Read Controller after the frame crosses the receive threshold (set by the DMA Register 6 RTC bits).

If the MTL Receive FIFO is configured to operate in Store-and-Forward mode, all error frames can be filtered and dropped.

If you enable the Frame Length FIFO in GMAC-MTL configuration (refer to “[Frame Length Interface](#)” on page [92](#)), then error frames can be filtered even after the start address of the frame is transferred to the Read Controller in the default Cut-Through mode. In this configuration, if a frame’s status and length are

available to the Read Controller using the Frame Length FIFO when that frame's SOF is read from the Rx FIFO, then the complete erroneous frame can be dropped indirectly in MTL.

For the DMA to flush the error frame being read from the FIFO, it must assert the `ari_frameflush_i` signal. The MTL then stops transferring data to the application (DMA). It will internally read out the rest of the frame and drop it. The MTL will then start the transfer of next frame, if it is available.

### 3.6.2.5 Receive Status Word

At the end of the transfer of the Ethernet frame to the host, the MTL outputs the receive status (`ari_data_o[31:0]`) to the Application. This is indicated by an active `ari_rxstatus_val_o` signal. The detailed description of the receive status is the same as for Bits[31:0] of RDES0, given in [Table 7-1](#), except that Bits 31, 14, 9, and 8 are reserved and have a reset of 1'b0 by default. When the status of a partial frame due to overflow is given out, the Frame Length field in the status word is not valid.



**Note** When Advanced Time Stamp feature is enabled, the status is composed of two parts - normal (default [31:0]), and extended. The extended status[63:32] gives the information about the received ethernet payload when it is carrying PTP packets or TCP/UDP/ICMP over IP packets. In 32-bit data-bus, these are transferred over two clock cycles. The detailed description of the receive status is the same as described in RDES0 and RDES4 in ["Receive Descriptor"](#) on page 362, except that bits 31, 14, 9, and 8 of normal status is reserved and have a reset value of 1'b0. When the status of a partial frame due to overflow is given out, the Frame Length field in the status word is not valid.

### 3.6.2.6 Frame Length Interface

In case of switch applications, data transmission and reception between the application and MTL happens as complete frame transfers. The application layer should be aware of the length of the frames received from the ingress port in-order to transfer the frame to the egress-port. The GMAC-MTL can be configured to provide a separate frame length interface to support this feature.

The GMAC core provides the frame length of each received frame inside the status at the end of each frame transferred to the MTL. The MTL stores the frame length in an asynchronous FIFO (Frame Length FIFO). The width of this FIFO is configurable from 12-15 (including 1 bit for frame-error tag) bits, while the depth of this FIFO depends on the configured size of the MTL Rx FIFO and the minimum size of the frame stored in the Rx FIFO. A two-port RAM is required to implement this asynchronous FIFO. To provide maximum flexibility in choosing the FIFO implementation, the corresponding input/output signals are made available at the top-level ports of the GMAC-MTL.

When this asynchronous FIFO is non-empty, the frame length is read by the MTL Rx FIFO Read controller and given to the application on the `ari_frame_len_o` bus and the validity is indicated by the assertion of signal `ari_frame_len_val_o`. An additional signal (`ari_rxfifo_frm_cnt_o`) indicating the number of frames present in the Rx FIFO is also driven on the ARI interface. When the corresponding frame is read out on the ARI by the application, the frame length of the next frame is read from the FIFO. The application can use this frame length to check if egress port buffer can accommodate the frame.



A frame length value of 0 is given for partial frames written into the Rx FIFO due to overflow.

### 3.6.2.7 Receive Path Timing Diagrams

Timing specifications for the MAC Transaction layer (MTL) interface signals in the receive path are illustrated in detail, in the following sections. The timing diagrams are organized to show the sequence of transactions that occur when a received frame is transferred from the MAC to the application.

#### 3.6.2.7.1 Receive FIFO Write Interface

A frame received by the MAC is transferred to the MTL over the MRI interface. The transferred data is accepted and written to the Rx FIFO by the write control logic. (See [Figure 3-37](#).)

1. The MAC starts transferring the received frame to the MTL by asserting mri\_val\_o and mri\_sof\_o along with the corresponding 128-bit data on mri\_data\_o.
2. This is directly transferred to the Rx FIFO in the next clock cycle. This can be seen by the assertion of rwc\_csn\_o and rwc\_wr\_en\_o along with the corresponding address and data. Note that the MSB 5 bits of rwc\_wr\_data correspond to the value on mri\_eof and mri\_be\_o signals accepted by the MTL.
3. Frame transfer is completed by the MAC by the assertion of mri\_eof\_o along with valid data. The receive frame status on mri\_rxstatus\_val\_o is also valid with mri\_eof\_o.
4. After writing the EOF data, the Rx FIFO write logic transfers the received status to the external RAM in the next clock cycle.

**Figure 3-37 Receive FIFO Write Interface Timing**



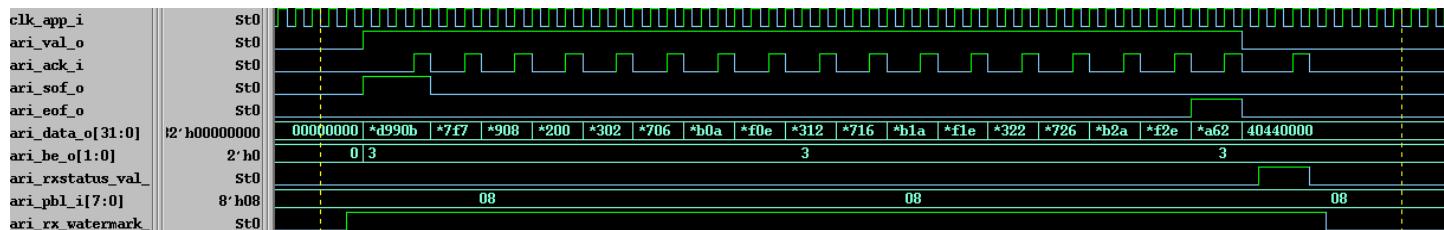
#### 3.6.2.7.2 Application Receive Interface (ARI)

The data received from the MAC and stored in the Rx FIFO is transferred to the Application Receive Interface with the following sequence of events, as shown in [Figure 3-38](#).

1. The ARI indicates the availability of data by asserting signal ari\_val\_o along with the data. The ARI transfers data in every clock cycle as long as data is available in the Rx FIFO.
2. The application can check whether the Rx FIFO has the amount of data (or the complete frame) to support the required burst length on ari\_pbl\_i.
3. The ARI updates ari\_watermark\_o in the next clock cycle. Asserting the watermark indicates that the MTL has enough data to sustain the burst transfer of length given on ari\_pbl\_i (in Step 2).
4. The application accepts the data from the MTL by asserting ari\_ack\_i. The MTL updates the data in the next clock cycle if it is continuing the burst transfer. The application can pause the transfer by deasserting ari\_ack\_i, as shown in [Figure 3-38](#).
5. The MTL transfers the remaining data by asserting ari\_val\_o and the corresponding data. When the last few bytes of frame (less bytes than the data bus width) are being transferred, ari\_eof\_o is also asserted along with ari\_be\_o, indicating a valid number of bytes in that data phase.

- The receive frame status (indicated with the assertion of ari\_rxstatus\_val\_o) is transferred back-to-back with the end-of-frame data. Frame transfer is complete only after ari\_ack\_i acknowledgement for status.

**Figure 3-38 Application Receive Interface (ARI) Timing**



### 3.6.2.7.3 ARI With Time Stamping (Default)

Figure 3-39 shows the ARI timing when IEEE 1588 time stamping is enabled. After the ARI outputs end-of-frame (EOF) data, as indicated by ari\_eof\_o high, it gives the time stamp value on ari\_data\_o. Asserting ari\_timestamp\_val\_o validates the time stamp. In this figure, the 64-bit time stamp is output in two cycles, because the data bus is only 32 bits wide. After the application accepts the time stamp (by asserting ari\_ack\_i for two clocks), the receive status is given on the data bus and validated with the assertion of ari\_rxstatus\_val\_o.

**Figure 3-39 ARI Timing With IEEE 1588 Time Stamping Enabled**

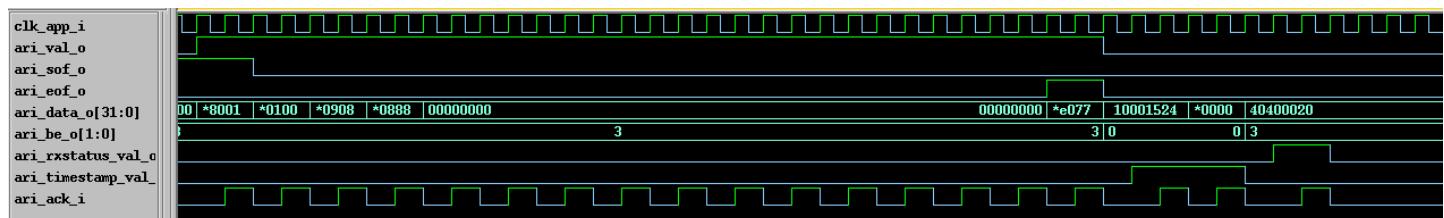
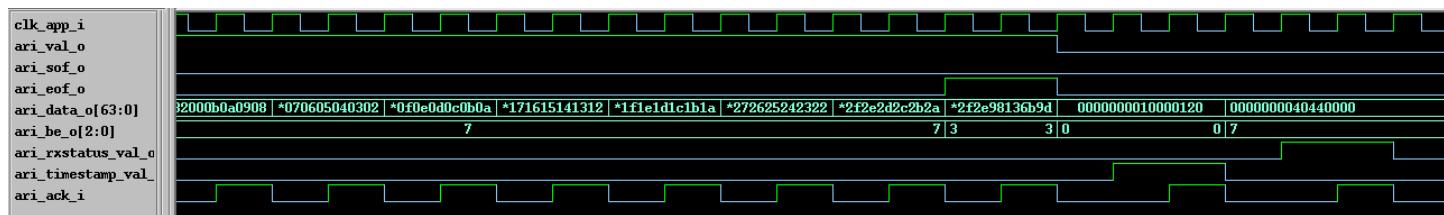


Figure 3-40 shows the timing of the time stamp transfer when the core is configured for a 64-bit data bus. The 64-bit time stamp is given in one cycle between the EOF data transfer and the Receive Status transfer.

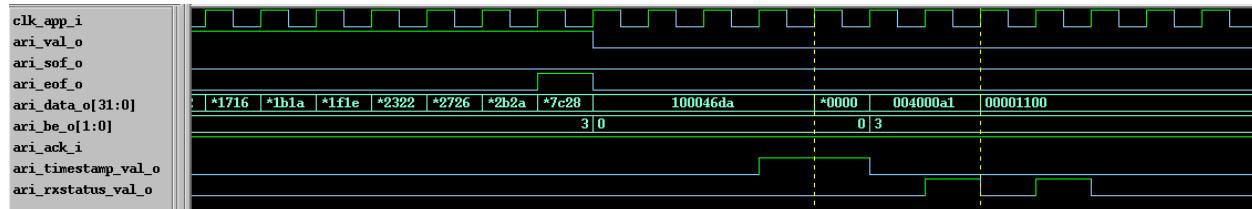
**Figure 3-40 ARI Timing for 64-Bit Bus, IEEE 1588 Time Stamping Enabled**



### 3.6.2.8 ARI with Advanced Time Stamping

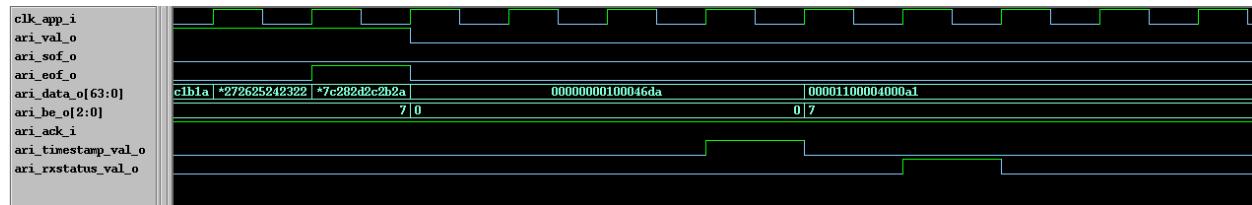
[Figure 3-41](#) shows the ARI timing when IEEE 1588 Advanced time stamping is enabled for a 32-bit interface. After the time stamp value is read out from the ari\_data\_o port by asserting ari\_ack\_i, the receive status is given on the data bus. The status is divided into two 32-bit status (normal and extended). The normal status is given first on ari\_data\_o validated with the assertion of ari\_rxstatus\_val\_o as shown in the figure. When the ari\_ack\_i is asserted for the normal status the extended status is given on ari\_data\_o validated with the assertion of ari\_rxstatus\_val\_o as shown in the figure. Bit 0 of the normal status indicates the availability of the extended status.

**Figure 3-41 ARI Timing for 32-Bit Interface, IEEE 1588 Advanced Time Stamping Enabled**



[Figure 3-42](#) shows the ARI timing when IEEE 1588 Advanced time stamping is enabled for 64-bit interface. After the time stamp value is read out from the ari\_data\_o port in one clock cycle, the receive status is given on the data bus. The normal and the extended status is given together on ari\_data\_o validated with the assertion of ari\_rxstatus\_val\_o as shown in the figure. The lower 32-bits [31:0] indicate the normal status and the higher 32-bits [63:32] indicate the extended status. Bit 0 indicates the validity of the extended status.

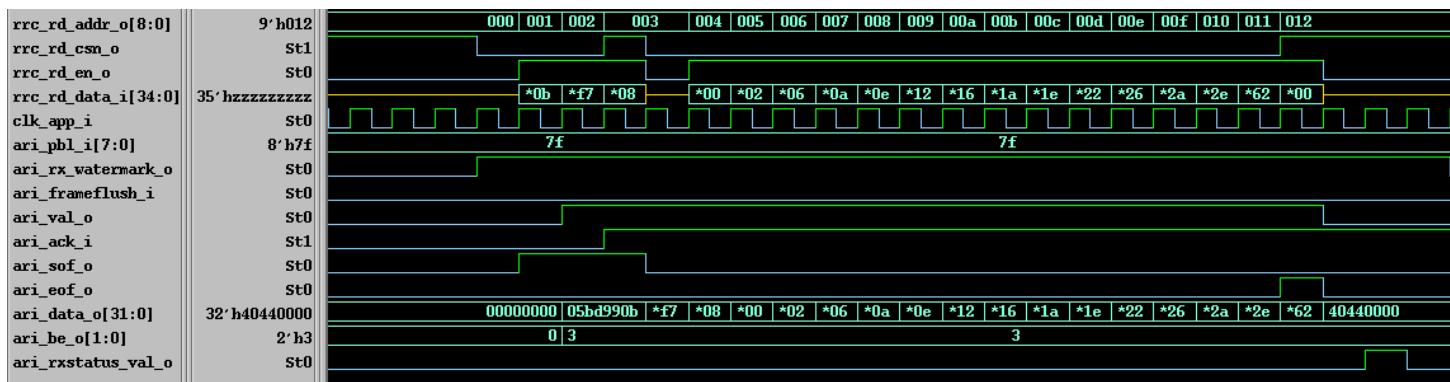
**Figure 3-42 ARI Timing for 64-Bit Interface, IEEE 1588 Advanced Time Stamping Enabled**



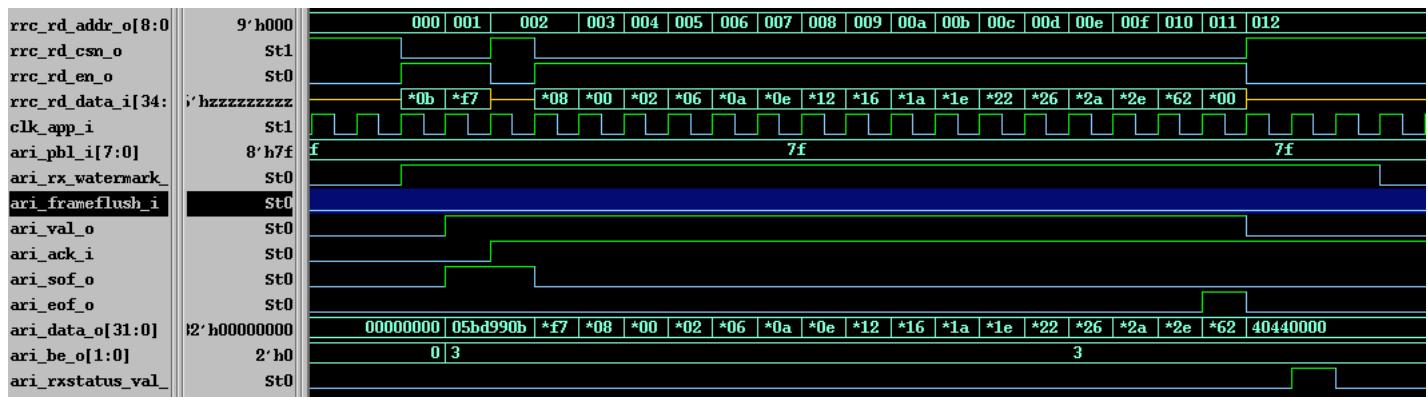
#### 3.6.2.8.1 Receive FIFO Read Interface

The FIFO read timing diagrams for the synchronous and asynchronous two-port RAM along with the timing relationship with signals on the ARI is shown in this section.

The MTL Rx FIFO read controller pre-fetches data (rrc\_rd\_\* signals) from the Rx FIFO when it finds the Rx FIFO is not empty. The pre-fetching corresponds to maximum of three locations for asynchronous RAM interface and two locations for synchronous RAM interface. This pre-fetching is done so as to sustain burst data transfers without any delay from ARI interface. The (pre-)fetched data is registered and given out on the ARI with the respective ari\_\* signals. When the application acknowledges the data with ari\_ack\_i, further data is read from the FIFO. [Figure 3-43](#) shows the FIFO read interface timing for synchronous SRAM. The timing relationship of the rrc\_rd\_\* signals is similar to that of trc\_rd\_\* signals explained in “[Transmit FIFO Read Interface](#)” on page 87.

**Figure 3-43 Receive FIFO Read Interface Timing (1 of 2)**

The timing relationship of the FIFO read interface signals (asynchronous SRAM) with the application receive signals is shown in [Figure 3-44](#). Note that in this figure burst frame transfers are sustained without any break, due to the prefetching of data.

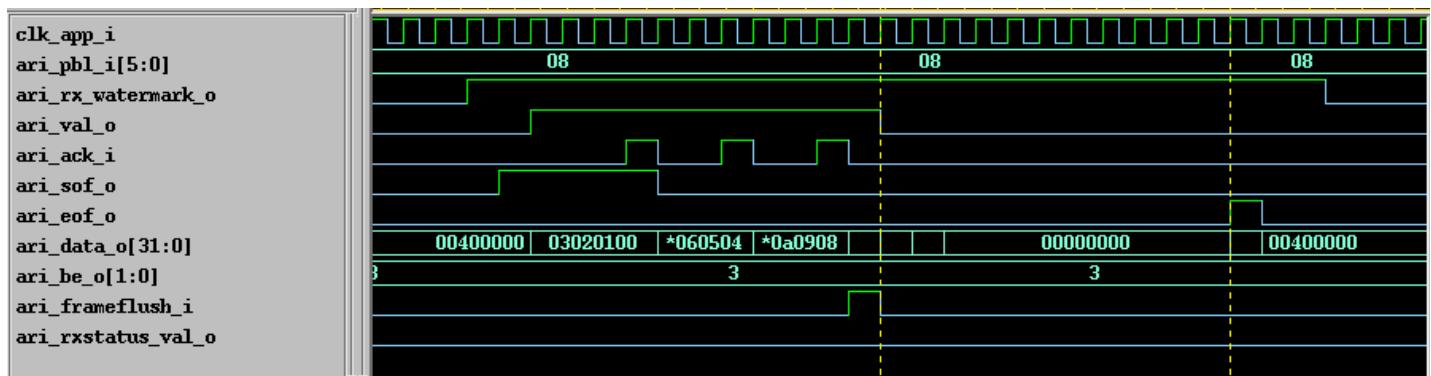
**Figure 3-44 Receive FIFO Read Interface Timing (2 of 2)**

### 3.6.2.8.2 Receive Frame Flush

When the application decides to drop the frame (at the top of the Rx FIFO) during the start, middle, or end of a frame transfer (`ari_val_o` is asserted), it can assert `ari_frameflush_i` for one clock. The MTL response to this is shown in [Figure 3-45](#).

The MTL first deasserts `ari_val_o` as soon as it sees an active pulse on `ari_frameflush_i`. It continues the read transfer from the Rx FIFO, but does not transfer it to the ARI. It thus flushes the entire frame, including the status. The MTL is now ready to transfer the next frame.

**Figure 3-45 Receive Frame Flush Timing**



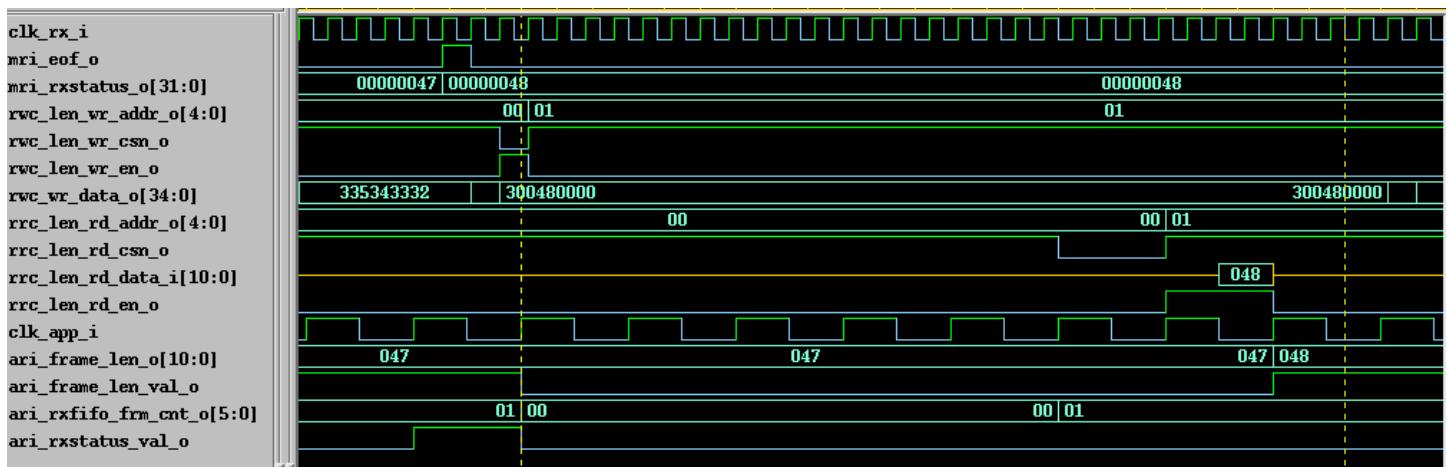
### 3.6.2.8.3 Receive Frame Length Interface (Optional)

[Figure 3-46](#) corresponds to a synchronous two-port RAM implementation for frame length FIFO. The sequence of events that occurs during the frame length transfer from the GMAC core to the application is as follows:

1. GMAC core issues the frame length of the received frame through `mri_rxstatus_o`.
2. Frame length is written with `rwc_wr_data_o[FIFO_WIDTH+16-1:16]` where `FIFO_WIDTH` is configured to any value between 12 to 15. `rwc_wr_data_o` is the data written to the Rx FIFO. The `rwc_err_frame_o` signal is written to the MSB of this frame-length FIFO.
3. When the Frame Length FIFO is not empty, the frame length is read out and issued to the application through `ari_frame_len_o` and an active `ari_frame_len_val_o` signal.
4. After a complete frame is read out (i.e. after the assertion of `ari_rxstatusval_o`), `ari_frame_len_val_o` is deasserted or held high, based on the availability of the next frame's length.



Signal `ari_rx fifo frm cnt o` indicates the number of frames whose EOF is present in the Rx FIFO.

**Figure 3-46 Receive Frame Length Interface Timing**

The write timing of the asynchronous two-port RAM is similar to that of the synchronous two-port RAM. The difference in the read timing is that the frame length from the FIFO (`rrc_len_rd_data_i`) is provided in the same clock as the asserted chip select (`rrc_len_rd_csn_o`).

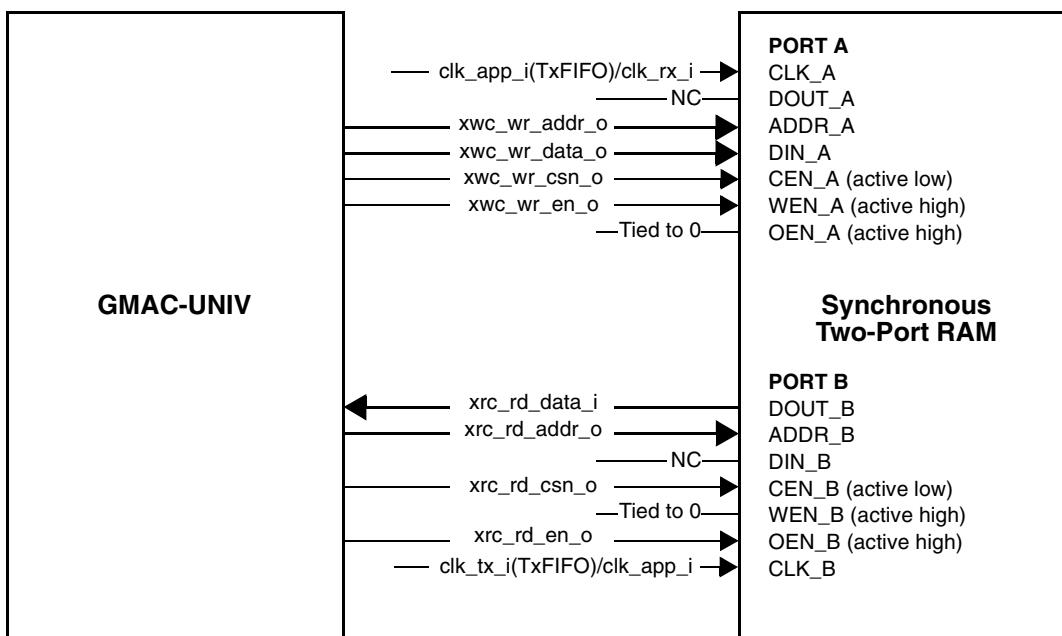
### 3.6.3 Interfacing With External Two-Port RAMs

The GMAC-UNIV core provides a generic memory interface for easy interconnection to any vendor-specific, two-port RAM for implementing the Tx FIFO and Rx FIFO. This section describes how GMAC-UNIV can be interconnected with different types of two-port RAMs.

The two memory interfaces in GMAC-UNIV provided for connectivity with Transmit and Receive two-port RAMs are identical. Hence, the connections with the Tx FIFO and Rx FIFO RAMs are illustrated in the following sections in a single diagram for each type of RAM. The “x” variable in the signal names can be replaced with a “t” for the Tx FIFO interface and an “r” for the Rx FIFO interface.

#### 3.6.3.1 Low-Power, Synchronous Two-Port SRAM Connection

The synchronous, two-port SRAM shown in [Figure 3-47](#) consists of two ports, Port A and Port B, which can be used for both read and write operations. The GMAC-UNIV (configured for synchronous RAM interface) uses this SRAM as an asynchronous FIFO for transferring data from one clock domain to another clock domain. Thus, a port (Port A, for example) is used to write the frame data and the other port is used to read the frame data, with the writes and reads happening in different clock domains. (Consequently, the Read data (DOUT\_A) from Port A and write data (DIN\_B) to Port B are left unconnected in [Figure 3-47](#).)

**Figure 3-47 Low Power, Synchronous Two-Port SRAM Connection**

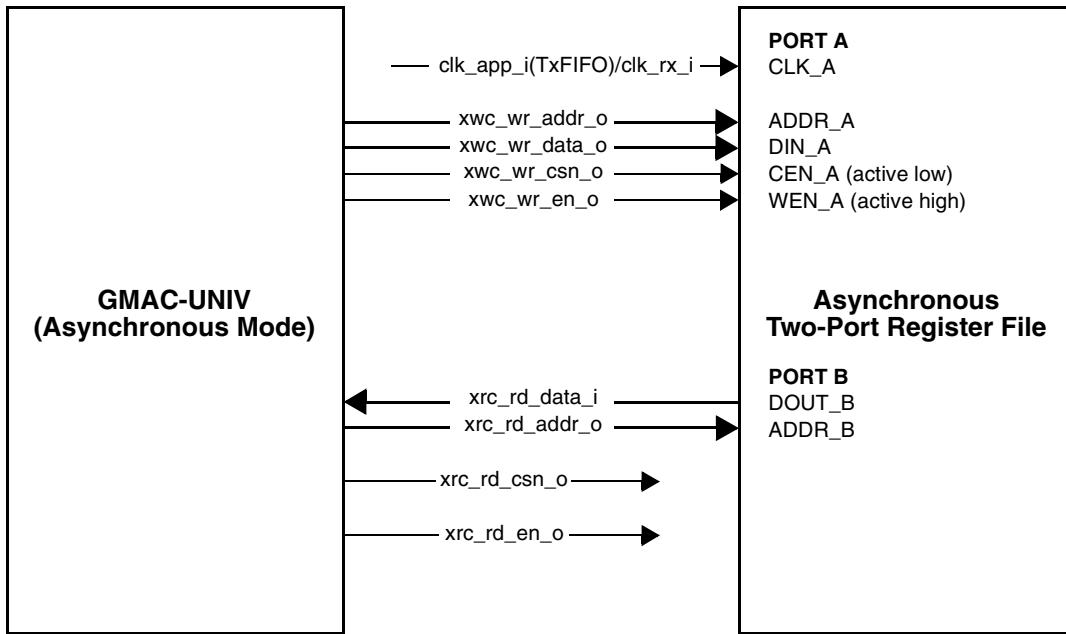
The SRAM chip enable (CEN) and write enable (WEN) are connected to chip select (\*\_csn\_o) and write enable (\*\_wr\_en\_o) signals from the GMAC-UNIV core. Based on the active states of CEN and WEN, GMAC-UNIV memory interface signals are directly connected or have inverted polarity. Connecting the chip select signal ensures that the SRAM I/O drivers are turned off when read/write operation is not scheduled and thereby reduces power consumption.

If the memory module (such as Xilinx FPGA Block Select RAM) has a single enable for the read port (No Output Enable), xrc\_rd\_csn\_o must be connected to the enable, with polarity taken into consideration.

### 3.6.3.2 Low Power, Asynchronous Two-Port Register File Connection

As shown in [Figure 3-48](#), an asynchronous, two-port register file has two ports: Port A for write operation and Port B for read operation. The write operation takes effect with regard to the Port A clock, but the read operation is asynchronous and governed only by the read address.

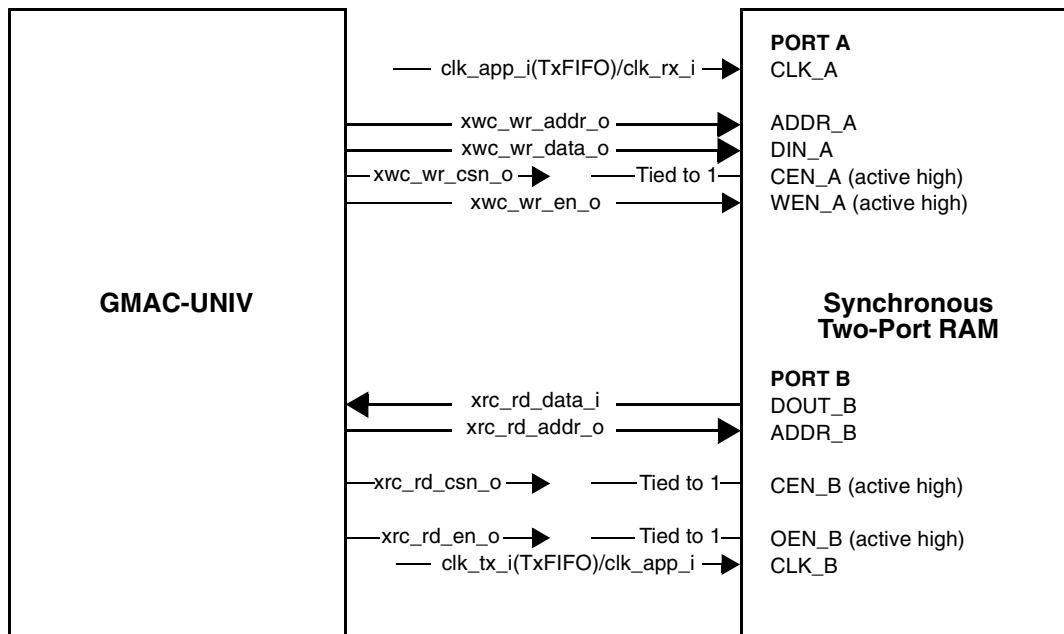
**Figure 3-48 Low Power, Asynchronous Two-Port Register File Connection**



### 3.6.3.3 High-Speed, Synchronous Two-Port SRAM Connection

In applications where power consumption is not a factor and memory access times are too critical, the chip enable and read enable signals can always be tied to active high inputs. In such applications, the generic memory interface signals can be connected to synchronous SRAM as shown in [Figure 3-49](#). The unused outputs of the generic memory interface are also shown as floating outputs.

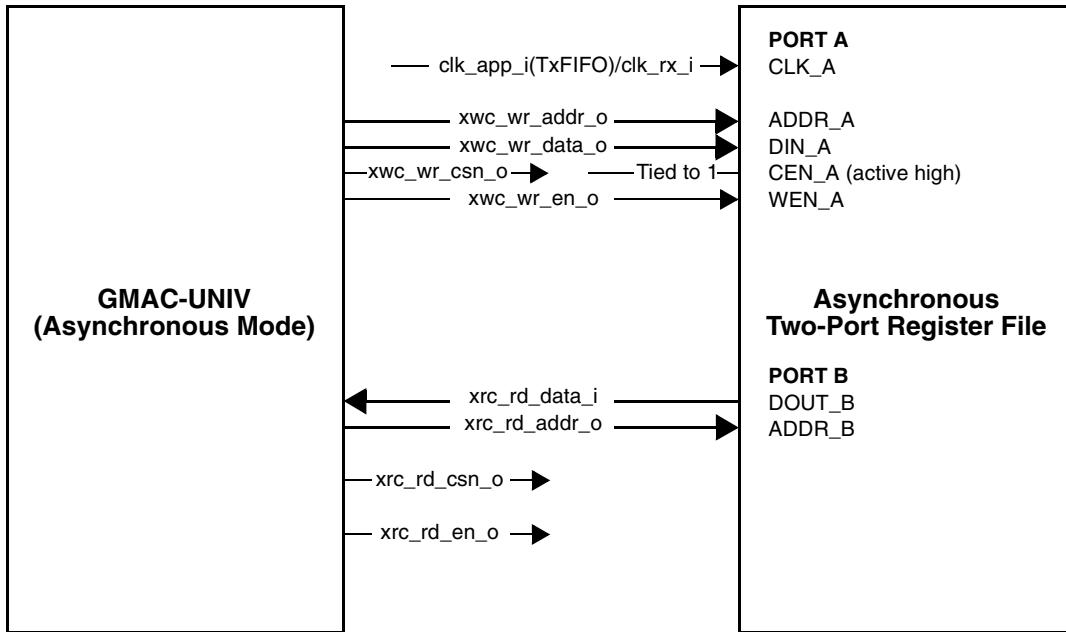
**Figure 3-49** High-Speed, Synchronous Two-Port SRAM Connection



### 3.6.3.4 High-Speed, Asynchronous Two-Port RAM Connection

Figure 3-50 shows the interconnection for achieving high-speed access times for two-port asynchronous RAMs/register files.

**Figure 3-50 High-Speed, Asynchronous Two-Port RAM Connection**



## 3.7 GMAC Core

The GMAC core supports many interfaces towards the PHY chip. The PHY interface can be selected only once after reset. The GMAC core communicates with the application side with the MAC Transmit Interface (MTI), MAC Receive Interface (MRI) and the MAC Control Interface (MCI). The core default and optional signals are described in “[Signal Descriptions](#)” on page [179](#).

### 3.7.1 Transmission

Transmission is initiated when the MTL Application pushes in data with the SOF (mti\_sof\_i) signal asserted. When the SOF signal is detected, the GMAC accepts the data and begins transmitting to the GMII/MII. The time required to transmit the frame data to the GMII/MII after the Application initiates transmission is variable, depending on delay factors like IFG delay, time to transmit preamble/SFD, and any back-off delays for Half-Duplex mode. Until then, the GMAC does not accept the data received from MTL by deasserting the mti\_rdy\_o signal.

After the EOF (mti\_eof\_i) is transferred to the GMAC Core, the core complete normal transmission and then gives the Status of Transmission back to the MTL. If a normal collision (in Half-duplex mode) occurs during transmission, the GMAC core makes valid the Transmit Status to the MTL. It then accepts and drops all further data until the next SOF is received. The MTL block should retransmit the same frame from SOF on observing a Retry request (in the Status) from the GMAC.

The GMAC issues an underflow status if the MTL is not able to provide the data continuously during the transmission. During the normal transfer of a frame from MTL, if the GMAC receives a SOF without getting an EOF for the previous frame, then it (the SOF) is ignored and the new frame is considered as continuation of the previous frame.

The following six modules constitute the transmission function of the GMAC:

- ❖ Transmit Bus Interface Module (TBU)
- ❖ Transmit Frame Controller Module (TFC)
- ❖ Transmit Protocol Engine Module (TPE)
- ❖ Transmit Scheduler Module (STX)
- ❖ Transmit CRC Generator Module (CTX)
- ❖ Transmit Flow Control Module (FTX)

### 3.7.1.1 Transmit Bus Interface Module

This module interfaces the transmit path of the GMAC core with the external frame with a FIFO interface. It accepts data in 32/64/128-bit-wide bus. This interface runs on the clock `clk_tx_i`, of the GMII interface.

This module also outputs the (32-bit) Transmit Status to the application at the end of normal transmission or collision. This module also does the Endian conversion of data bus by swapping the byte lanes and corresponding byte enables.

Additionally, this module outputs the Transmit Snapshot register value to the `mti_timestamp_o` signal and asserts the `mti_status_valid` signal.

### 3.7.1.2 Transmit Frame Controller Module

The Transmit Frame Controller (TFC) consists of two registers to hold data, byte enables, and the last data control received from the TBU. The register provides a buffer between the Application and the TPE to regulate data flow as well as converts the input data into an 8-bit bus towards the TPE.

When the number of bytes received from the Application falls below 60 (DA+SA+LT+DATA), the state machine that interfaces with the TBU automatically appends zeros to the transmitting frame to make the data length exactly 46 bytes (provided `mti_dispad_i` is LOW) to meet the minimum data field requirement of IEEE 802.3. The GMAC can be programmed not to append any padding through the sideband signal, `mti_dispad_i`, from the MTI.

The cyclic redundancy check (CRC) for the Frame Check Sequence (FCS) field is calculated before transmission to the TPE module. This value is computed by CTX module. The TFC module receives the computed CRC and appends it to the data being transmitted to the TPE module. When the GMAC is programmed (through a sideband signal `mti_discrc_i` from the Application interface) to not append the CRC value to the end of Ethernet frames, the TFC module ignores the computed CRC and transmits only the data received from the TBU module to the TPE module. An exception to this rule is that when the GMAC is programmed to append pads for frames (DA+SA+LT+DATA) less than 60 bytes sent by the TBU module, the TFC module will append the CRC at the end of padded frame irrespective of the value on the `mti_discrc_i` signal.

The TFC converts the data received on the 32/64/128-bit interface from the TBU into 8-bit data for the TPE module.

### 3.7.1.3 Transmit Protocol Engine Module

The Transmit Protocol Engine (TPE) module consists of a transmit state machine that controls the operation of Ethernet frame transmission. The module's transmit state machine performs the following functions to meet the IEEE 802.3/802.3z specifications.

- ❖ Generates preamble and SFD
- ❖ Generates jam pattern in Half-Duplex mode

- ❖ Generates carrier extension in Half-Duplex (only in GMII mode)
- ❖ Frame bursting in Half-Duplex (only in GMII mode)
- ❖ Jabber timeout
- ❖ Flow control for Half-Duplex mode (back pressure)
- ❖ Generates transmit frame status
- ❖ Contains time stamp snapshot logic for IEEE 1588 support

When a new frame transmission from the TFC is requested, the transmit state machine sends out the preamble and SFD, followed by the data received. The preamble is defined as 7 bytes of 8'b10101010 pattern, and the SFD is defined as 1 byte of 8b'10101011 pattern.

The collision window is defined as 1 slot time (512 bit times for 10/100 Mbps Ethernet and 4,096 bit times for 1,000 Mbps Ethernet). The jam pattern generation is applicable only to Half-Duplex mode, not to Full-Duplex mode. In Full-Duplex mode, the transmit state machine ignores the phy\_col\_i signal from the PHY.

In MII mode, if a collision occurs any time from the beginning of the frame to the end of the CRC field, the transmit state machine sends a 32-bit jam pattern of 32'h55555555 on the MII to inform all other stations that a collision has occurred. If the collision is seen during the preamble transmission phase, the transmit state machine completes the transmission of preamble and SFD and then sends the jam pattern.

In GMII mode, if a collision occurs any time between the beginning of the frame and the end of the extension field, the transmit state machine sends a 32-bit jam pattern of 32'h55555555 on the GMII to inform all other stations of the collision. If the collision is seen during the preamble transmission phase, the transmit state machine completes the transmission of preamble and SFD, then sends the jam pattern.

If the collision occurs after the collision window and before the end of the FCS field (or the end of Burst if the Frame Burst mode is enabled), the transmit state machine sends a 32-bit jam pattern and sets the late collision bit in the transmit frame status.



**Note** At the GMII/MII interface, collision signal (phy\_col\_i) being asynchronous is checked by the transmitter after it is double-synchronized to clk\_tx domain. Hence recognition of collision or late collision event is delayed by this additional latency.

In GMII Half-Duplex mode (1,000 Mbps), the Transmit state machine ensures that all valid carrier events exceed a slot time of 4,096 bit times. To accomplish this, any transmit frame shorter than 512 bytes from the TFC module is extended using a carrier extension. On the GMII, this is signaled to the PHY by asserting phy\_txer\_o, deasserting phy\_txen\_o, and setting phy\_txd\_o[7:0] to 8'0F.

When the Frame Burst mode is enabled, only the first frame of the Burst will be carrier extended, as necessary. The carrier extension is not applicable for MII Half-Duplex and GMII/MII Full-Duplex modes. When the Frame Burst mode is enabled, the transmit state machine transmits a burst of frames (as long as frames are available from the TFC module) without releasing the carrier of the PHY. To accomplish this, the state machine inserts the carrier extension for a minimum IFG period (96 bit times) between the frames. The transmit state machine continues to burst frames as long as additional frames are available from the TFC module and a burst limit of 8192 byte times has not been exceeded. If the additional frame is not available at the end of the IFG period in the middle of the Burst, the transmit state machine releases the carrier by deasserting the phy\_txer\_o and phy\_txen\_o signals on the GMII.

Frame bursting is applicable only for GMII Half-Duplex mode and is not applicable in MII and GMII/MII Full-Duplex modes.

The TPE module maintains a jabber timer (only in 10/100-Mbps mode) to cut off the transmission of Ethernet frames if the TFC module transfers more than 2,048 (default) bytes. The time-out is changed to 10,240 bytes when the Jumbo frame is enabled.

The Transmit state machine uses the deferral mechanism for the flow control (Back Pressure) in Half-Duplex mode. When the Application requests to stop receiving frames, the Transmit state machine sends a JAM pattern of 32 bytes whenever it senses a reception of a frame, provided the transmit flow control is enabled. This will result in a collision and the remote station will back off. The Application requests the flow control through a sideband signal mti\_flowctrl\_i (or by setting BPA bit of Register6). If the application requests a frame to be transmitted, then it will be scheduled and transmitted even when the backpressure is activated. Note that if the backpressure is kept activated for a long time (and more than 16 consecutive collision events occur) then the remote stations will abort their transmissions due to excessive collisions.

If IEEE 1588 time stamping is enabled for the transmit frame, this block takes a snapshot of the system time when the SFD is put onto the transmit GMII/MII bus. The system time source is either an external input or internally generated, according to the configuration selected.

### 3.7.1.4 Transmit Scheduler Module

The Transmit Scheduler (STX) module is responsible for scheduling the frame transmission on the GMII/MII. The two major functions of this module are to maintain the inter-frame gap between two transmitted frames and to follow the Truncated Binary Exponential Back-off algorithm for Half-Duplex mode. This module provides an enable signal to the TPE module after satisfying the IFG and Back-off delays.

The STX module maintains an idle period of the configured inter-frame gap (IFG bits of Register 0 between any two transmitted frames. If frames from the TFC arrive at the TPE module sooner than the configured IFG time, the TPE module waits for the enable signal from the STX module before starting the transmission on the GMII/MII. The STX module starts its IFG counter as soon as the carrier signal of the GMII/MII goes inactive. At the end of programmed IFG value, the module issues an enable signal to the TPE module in Full-Duplex mode. In Half-Duplex mode and when IFG is configured for 96 bit times, the STX module follows the rule of deference specified in Section 4.2.3.2.1 of the IEEE 802.3 specification. The module resets its IFG counter if a carrier is detected during the first two-thirds (64-bit times for all IFG values) of the IFG interval. If the carrier is detected during the final one third of the IFG interval, the STX module continues the IFG count and enables the transmitter after the IFG interval.

The STX module implements the Truncated Binary Exponential Back-off algorithm when it operates in Half-Duplex mode.

### 3.7.1.5 Transmit CRC Generator Module

The Transmit CRC Generator (CTX) module interfaces with the TFC module to generate CRC for the FCS field of the Ethernet frame. The TFC module sends the frame data and any necessary padding to the CTX module through an 8-bit interface.

This module calculates the 32-bit CRC for the FCS field of the Ethernet frame. The encoding is defined by the following generating polynomial.

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The module gets the Ethernet frame's byte data from the TFC module (DA + SA + LT + DATA + PAD) qualified with a Data Valid signal. The TFC also indicates to the CTX when to reset the previously calculated CRC and to start the new CRC calculation for the coming frame. The TFC module issues the start command before sending the new frame data for calculation. The calculated CRC is valid on the next clock after the data is received.

In the GMII mode, the Data Valid signal is valid for every clock, from the first data byte through the last data byte. In MII mode, this signal is valid every alternate clock.

### 3.7.1.6 Transmit Flow Control Module

The Transmit Flow Control (FTX) module generates Pause frames and transmit them to the TFC module as necessary, in Full-Duplex mode. The TFC module receives the Pause frame from the FTX module, appends the calculated CRC, and sends the frame to the TPE module. Pause frame generation can be initiated in two ways. The Application can request the FTX module to send a Pause frame either by setting the FCB bit in the Flow Control register Register 6 or by asserting the `mti_flowctrl_i` signal in response to the receive FIFO full conditions (packet buffer).

If the Application has requested the flow control by setting the FCB bit of Register 6, the FTX module will generate and transmit a single Pause frame to the TFC module. The value of the Pause Time in the generated frame contains the programmed Pause Time value in Register 6. To extend the pause or end the pause prior to the time specified in the previously transmitted Pause frame, the application must request another Pause frame transmission after programming the Pause Time register with appropriate value.

If the Application has requested the flow control by asserting the `mti_flowctrl_i` signal, the FTX module will generate and transmit a Pause frame to the TFC module. The value of the Pause Time in the generated frame contains the programmed Pause Time value in the Register 6. The FTX module monitors the `mti_flowctrl_i` signal. If it remains asserted at a configurable number of slot-times (PLT bits of GMAC Register 6) before this Pause-time runs-out, a second Pause frame will be transmitted to the TFC module. The process will be repeated as long as the `mti_flowctrl` signal remains asserted.

If the `mti_flowctrl_i` signal goes inactive prior to the sampling time, the FTX module will transmit a Pause frame with zero Pause Time to indicate to the remote end that the receive buffer is ready to receive new data frames.

### 3.7.2 MAC Transmit Interface Protocol

The MAC Transmit Interface (MTI) connects the application (MTL module in the GMAC) with the GMAC to provide the Ethernet data for transmission.

The application initiates the Ethernet frame transmission by writing the first data (`mti_sof_i`) of the frame to the GMAC, provided the GMAC is ready to accept data (indicated by `mti_rdy_o`). Each valid data transfer is indicated by an active `mti_valid_i` signal. The Application can push-in data as long as the GMAC core is ready to accept it. Thus the Application can assume a successful data transfer if the `mti_rdy_o` signal is asserted.

The application indicates the last data of the frame by asserting the end-of-packet `mti_eof_i` signal, along with the last data and byte enables (`mti_be_i`). The number of valid byte lanes for the last transfer is decoded with the `mti_be_i` signal (See [Table 3-8](#)). At the end of normal transmission of the Ethernet frame, the GMAC Core outputs the transmit status (`mti_txstatus_o`, described in [Table 3-11](#)) to the application. This is indicated by an active `mti_txstatus_val_o` signal.

**Table 3-8 mti\_be\_i Function for 128-Bit Bus**

<b>mti_be_i</b>	<b>Valid byte lanes (Little Endian)</b>	<b>Valid byte lanes (Big Endian)</b>
0000	<code>mti_data_i[7:0]</code> – Byte lane 0	<code>mti_data_i[127:120]</code> – Byte lane 15
0001	<code>mti_data_i[15:0]</code> – Byte lane 0-1	<code>mti_data_i[127:112]</code> – Byte lane 15-14
0010	<code>mti_data_i[23:0]</code> – Byte lane 0-2	<code>mti_data_i[127:104]</code> – Byte lane 15-13

**Table 3-8 mti\_be\_i Function for 128-Bit Bus**

<b>mti_be_i</b>	<b>Valid byte lanes (Little Endian)</b>	<b>Valid byte lanes (Big Endian)</b>
...		
1110	mti_data_i[119:0] – Byte lane 0-14	mti_data_i[127:8] – Byte lane 15-1
1111	mti_data_i[127:0] – Byte lane 0-15	mti_data_i[127:0] – Byte lane 15-0

**Table 3-9 mti\_be\_i Function for 64-Bit Bus**

<b>mti_be_i</b>	<b>Valid byte lanes (Little Endian)</b>	<b>Valid byte lanes (Big Endian)</b>
000	mti_data_i[7:0] – Byte lane 0	mti_data_i[63:56] – Byte lane 7
001	mti_data_i[15:0] – Byte lane 0-1	mti_data_i[63:48] – Byte lane 7-6
010	mti_data_i[23:0] – Byte lane 0-2	mti_data_i[63:40] – Byte lane 7-5
...		
110	mti_data_i[55:0] – Byte lane 0-6	mti_data_i[63:8] – Byte lane 7-1
111	mti_data_i[63:0] – Byte lane 0-7	mti_data_i[63:0] – Byte lane 7-0

**Table 3-10 mti\_be\_i Function for 32-Bit Bus**

<b>mti_be_i</b>	<b>Valid byte lanes (Little Endian)</b>	<b>Valid byte lanes (Big Endian)</b>
00	mti_data_i[7:0] – Byte lane 0	mti_data_i[31:24] – Byte lane 3
01	mti_data_i[15:0] – Byte lane 0-1	mti_data_i[31:16] – Byte lane 3-2
10	mti_data_i[23:0] – Byte lane 0-2	mti_data_i[31:8] – Byte lane 3-1
11	mti_data_i[31:0] – Byte lane 0-3	mti_data_i[31:0] – Byte lane 3-0

If the frame transmission is not successful (due to underflow, collision, jabber timeout, excessive deferral events), the GMAC core will assert the transmit status even before the EOF is received. The Application will have to take the appropriate action as per the status. The GMAC will drop all further data input to it until the next SOF.

**Table 3-11 Transmit Status at the GMAC Core Interface**

<b>Bit</b>	<b>Description</b>
31	<b>Frame Retry Requested</b> When set, this bit indicates a request from the MAC transmitter for retransmission of the entire frame from the Application. This bit has higher priority than the Frame Aborted (Bit 0). The GMAC core expects a retransmission of the frame when this bit is set irrespective of the value of Frame Aborted bit.

**Table 3-11 Transmit Status at the GMAC Core Interface (Continued)**

Bit	Description
30	TSS: Time Stamp Status When set, this bit indicates that the MAC transmitter has captured the transmitted frame's IEEE1588 time stamp. The captured time stamp is available, along with the transmit status, on the mti_timestamp_o[63:0] output. This bit is enabled only when IEEE1588 time stamping is enabled. Otherwise, it is reserved.
29	VLAN Frame When set, this bit indicates to the Application that the transmitted frame is a VLAN tagged frame (Type field equals 16'h8100).
28:27	Address Type This field indicates the type of destination address transmitted. <ul style="list-style-type: none"> <li>• 2'b00: Unicast</li> <li>• 2'b01: Multicast</li> <li>• 2'b10: Reserved</li> <li>• 2'b11: Broadcast</li> </ul>
26:13	Transmit Byte Count This 14-bit counter indicates the number of bytes transmitted.
12:9	Collision Count This 4-bit counter indicates the number of collisions that occurred before the frame was transmitted. The count is not valid when the Excessive Collisions bit is set.
8	Deferred When set, this bit indicates that the GMAC defers before transmission because of the presence of carrier.
7	Underflow When set, this bit indicates that the GMAC aborted the transmission because of data underflow. The Underflow error indicates that no more data is available for transmission in the middle of transmission.
6	Excessive Collisions When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If the DR (Disable Retry) bit in the Configuration register is set, this bit is set after the first collision and the transmission of the frame is aborted.
5	Late Collision When set, this bit indicates that the frame transmission was aborted due to a collision occurring after the collision window (64 bytes including Preamble in MII mode and 512 bytes including Preamble and Carrier Extension in GMII mode). Not valid if the Underflow error is set.
4	Excessive Deferral When set, this bit indicates that the transmission has ended because of excessive deferral of over 24,288 bit times (155,680 in 1000Mbps mode or if Jumbo-frame is enabled), if the deferral check (DC) bit is set high in the Control register.
3	Loss of Carrier When set, this bit indicates that the loss of carrier occurred during the frame's transmission (i.e., the gmii_crs_i signal was inactive for one or more transmission clock periods during frame transmission). This is valid only for the frames transmitted without collision and when the GMAC operates in Half-Duplex mode.

**Table 3-11 Transmit Status at the GMAC Core Interface (Continued)**

Bit	Description
2	No Carrier When set, this bit indicates that the carrier signal from the PHY was not present at the end of preamble transmission, and is valid only when the GMAC operates in Half-Duplex mode.
1	Jabber Timeout When set, this bit indicates the MAC transmitter has experienced a jabber timeout. This bit will be set only when the JB bit of the Configuration register is not disabled (deasserted).
0	Frame Aborted When set, this bit indicates that the transmission of the current frame has been aborted. The reason for the abort could be due to one or more of the following conditions. <ul style="list-style-type: none"><li>• Jabber Timeout</li><li>• No Carrier</li><li>• Loss of Carrier</li><li>• Excessive Deferral</li><li>• Late Collision</li><li>• Retry Count exceeds the attempt limit</li><li>• Data underrun</li></ul> When reset, this bit indicates that the current frame was successfully transmitted on the GMII/MII (provided Bit 31 is zero).

### 3.7.3 MAC Transmit Timing

Timing specifications for the MAC Core (GMAC-CORE) interface signals in the transmit path are illustrated in detail, in the following sections.

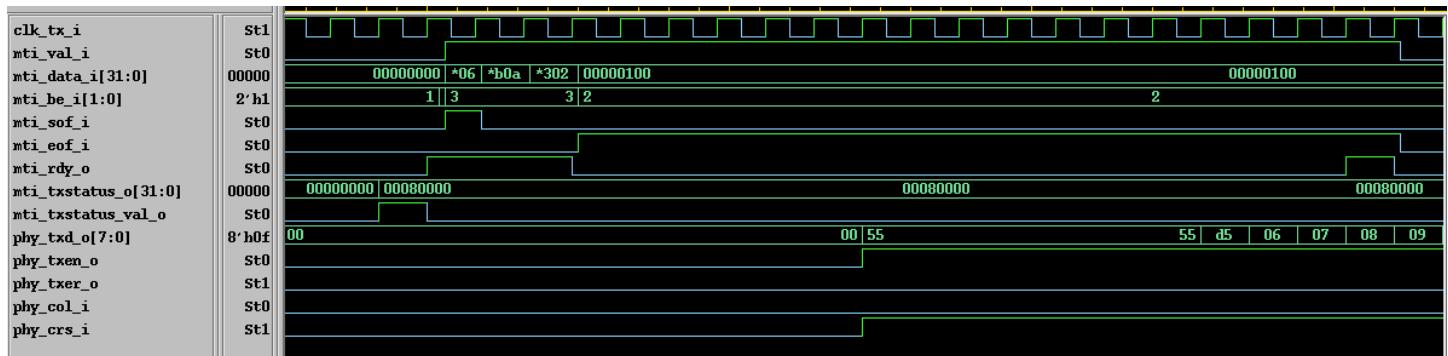
#### 3.7.3.1 MAC Transmit Interface (MTI)

Figure 3-51 shows the timing specifications for the MAC Transmit Interface signals. The frame transmission handshake mechanism is as follows:

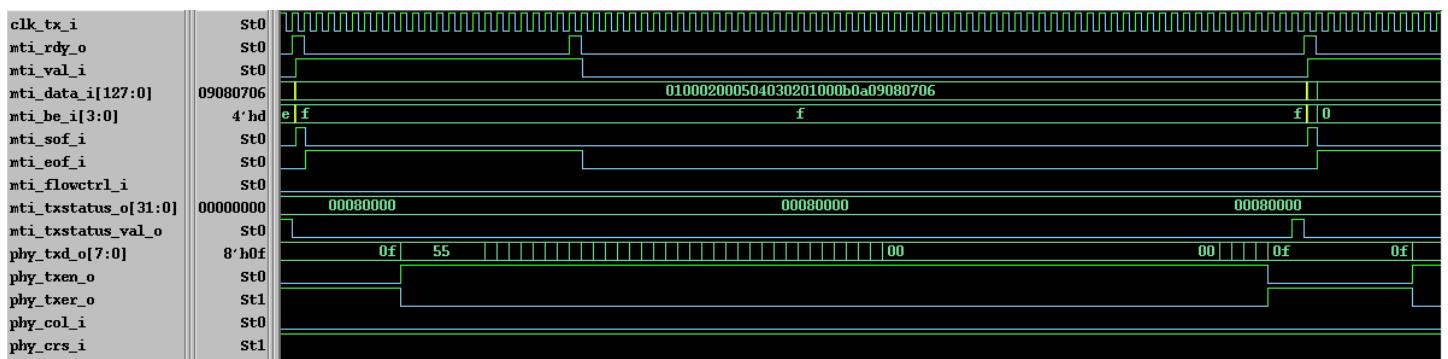
1. The application can initiate a frame transfer by asserting mti\_val\_i along with valid data on mti\_data\_i. Signal mti\_sof\_i should be asserted when the start of frame is being transferred.
2. The MAC core accepts the data as long as mti\_rdy\_o is active. The application should update the data (or deassert mti\_val\_i) on observing an active mti\_rdy\_o signal to transfer data in bursts.
3. The MAC starts transmitting the frame on the GMII as soon as it gets hold of the channel.
4. At any point during the transfer, the MAC can alert the application to stop the transfer by deasserting mti\_rdy\_o. This occurs when the MAC's internal FIFO is full.
5. When the application wants to transfer the last bytes of a frame, mti\_eof\_i should be asserted along with the byte enables (mti\_be\_i), indicating the valid number of data bytes on the bus. It is mandatory that all data transfers except the EOF transfer have the byte enables as all-ones.
6. After transferring the EOF, the application should wait until the MAC gives the transmission status of that frame. The MAC also deasserts mti\_rdy\_o after receiving the EOF, so that the application cannot transfer the next frame.
7. The MTI gives the transmit status of the frame back to the application through mti\_txstatus\_o by asserting mti\_txstatus\_val\_o (see Figure 3-52). The application has to accept it immediately.

8. The application can then start the transmission of the next frame.
9. Figure 3-52 also shows the timing of the GMII signals during the transmission of a frame, illustrating how the GMAC meets the IFG timing on the GMII during frame-burst transmission.

**Figure 3-51 MAC Transmit Interface (MTI) Timing (1 of 2)**



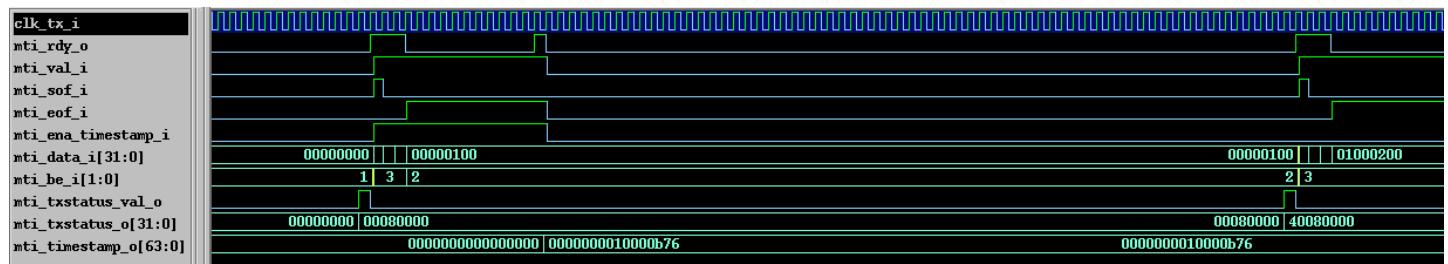
**Figure 3-52 MAC Transmit Interface (MTI) Timing (2 of 2)**



### 3.7.3.2 MTI With Time Stamping

Figure 3-53 shows the MTI's timing when IEEE 1588 time stamping is enabled. The mti\_ena\_timestamp\_i signal is sampled, along with the mti\_sof\_i, for an input frame. When that frame's status (0x40080000) is given on mti\_txstatus\_o, the time stamp captured for that frame is also output on mti\_timestamp\_o. The time stamp's validity is indicated by mti\_txstatus\_o[30] being asserted when mti\_txstatus\_val\_o is high.

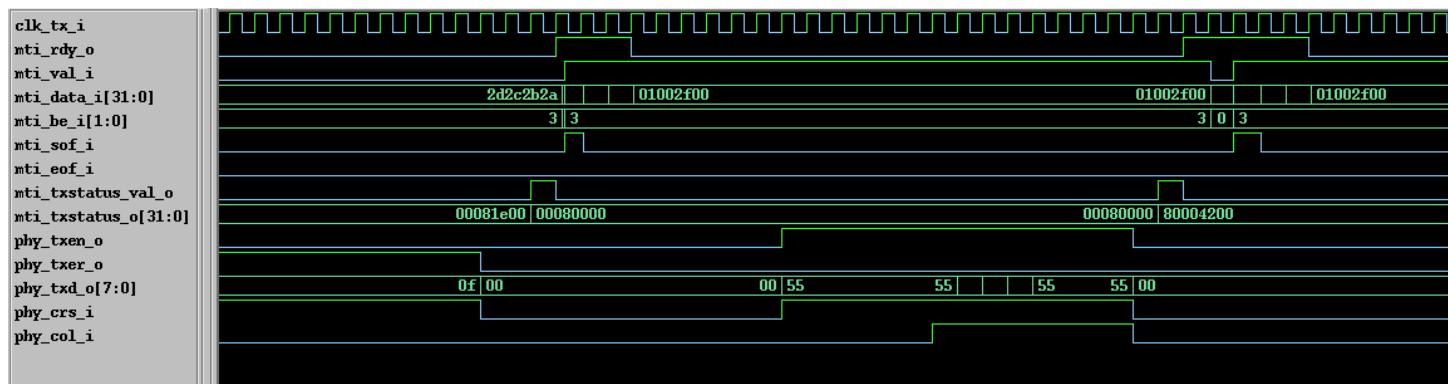
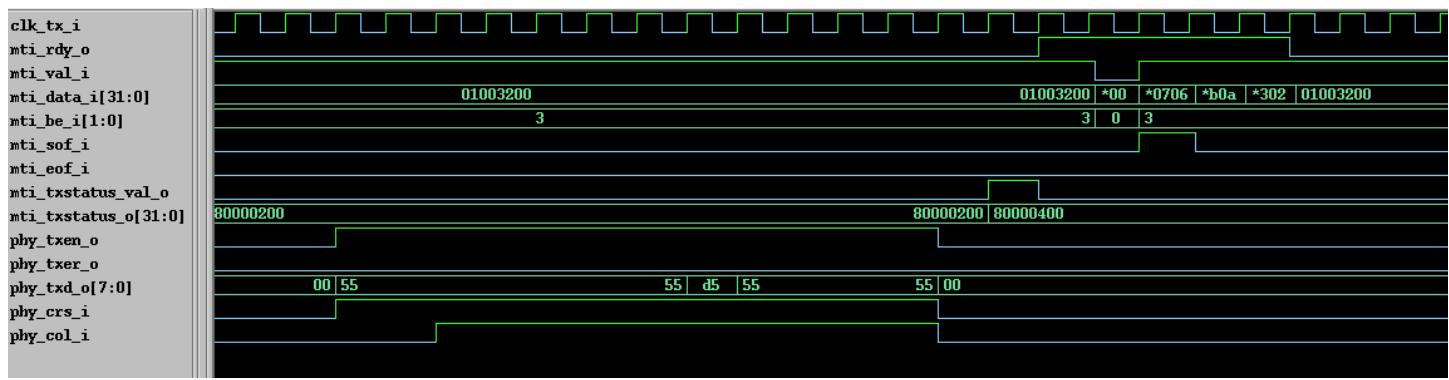
**Figure 3-53 MTI Timing With IEEE 1588 Time Stamping Enabled**



### 3.7.3.3 Collision During Transmission

Figure 3-54 and Figure 3-55 show how the MAC core behaves due to collisions occurring on the GMII interface.

1. The MAC accepts data from the application on the MTI and starts transmission in half-duplex mode.
2. When a collision is recognized, the MAC deasserts `mti_rdy_o` in order to stop taking in more data from the application on the MTI. In the figure, this signal is already low because the internal FIFOs were full.
3. After the transmission of the JAM pattern (4 bytes of 0x55), the MAC gives the transmission status (`mti_txstatus_o`) to the application by asserting `mti_txstatus_val_o`. It requests the application to retry the frame transfer by setting Bit31 of `mti_txstatus_o`.
4. In the next clock cycle, it asserts `mti_rdy_o`, indicating that it is ready to accept the next frame for transmission. The MAC (MTI) ignores or drops all data given after the assertion of `mti_txstatus_val_o` until the application indicates a start of frame transfer by asserting `mti_sof_i`.
5. [Figure 3-55](#) shows how the MAC transmitter starts a JAM pattern only after the transmission of the SFD even though the collision was recognized well in advance during the preamble transmission.

**Figure 3-54 Collision During Transmission (1 of 2)****Figure 3-55 Collision During Transmission (2 of 2)**

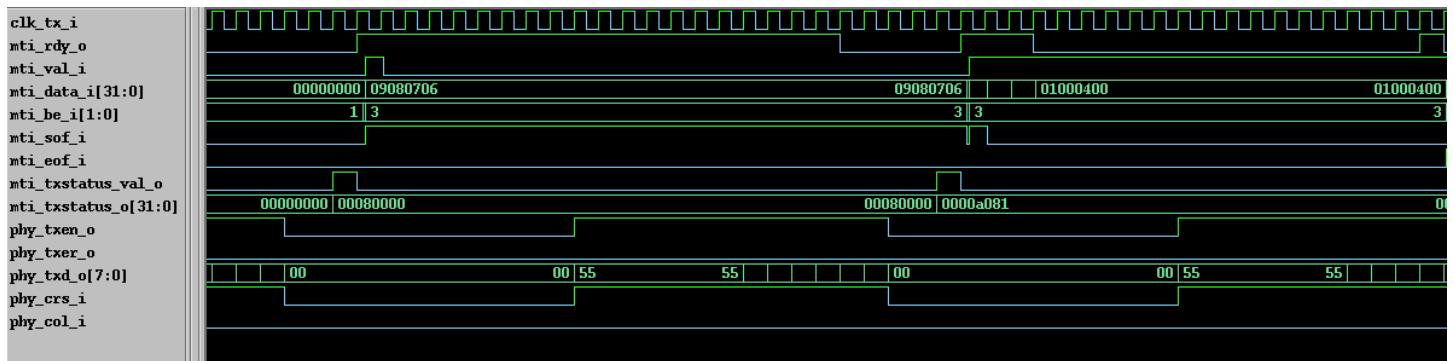
### 3.7.3.4 Underflow During Transmission

[Figure 3-56](#) shows the timing diagram for a frame abort due to insufficient data being input to the MAC.

1. The MAC schedules and starts a frame transmission as soon it receives the start of the frame on the MTI.
2. As the application does not input data on the MTI after the start-of-frame, the MAC finds that it does not have the end-of-frame data, and it deasserts `mti_rdy_o`.

3. The MAC aborts transmission after 5 bytes (the fifth byte is a dummy byte) in the figure.
4. It gives the transmit abort status (0x0000a081) to the application.
5. The MAC indicates that it is ready to accept new data in the next clock cycle by asserting mti\_rdy\_o.

**Figure 3-56 Underflow During Transmission Timing**



### 3.7.4 Reception

A receive operation is initiated when the GMAC detects an SFD on the GMII/MII. The core strips the preamble and SFD before proceeding to process the frame. The header fields are checked for the filtering and the FCS field used to verify the CRC for the frame. The received frame is stored in a shallow buffer until the address filtering is performed. The frame is dropped in the core if it fails the address filter.

The following are the functional blocks in the Receive path of the GMAC core.

- ❖ Receive Protocol Engine Module (RPE)
- ❖ Receive CRC Module (CRX)
- ❖ Receive Frame Controller Module (RFC)
- ❖ Receive Flow Control Module (FRX)
- ❖ Receive IP Checksum checker (IPC)
- ❖ Receive Bus Interface Unit Module (RBU)
- ❖ Address Filtering Module (AFM)

#### 3.7.4.1 Receive Protocol Engine Module

The RPE consists of the receive state machine which strips the preamble, SFD and carrier extension of the received Ethernet frame (in half-duplex 1000-Mbps mode). Once the phy\_rxdv\_i signal of the GMII/MII becomes active, the RPE's receive state machine begins hunting for the SFD field from the receive modifier logic. Until then, the state machine drops the receiving preambles. Once the SFD is detected, the state machine begins sending the data of the Ethernet frame to the RFC module, beginning with the first byte following the SFD (destination address).

If IEEE 1588 time stamping is enabled, the RPE takes a snapshot of the system time when any frame's SFD is detected on the GMII/MII. Unless the MAC filters out and drops the frame, this time stamp is passed on to the application.

In MII mode, the RPE converts the received nibble data into bytes, then forwards the valid frame data to the RFC module

The receive state machine of the RPE module decodes the Length/Type field of the receiving Ethernet frame. If the Length/Type field is less than 600 (hex) and if the MAC is programmed for the auto crc/pad stripping option, the state machine sends the data of the frame up to the count specified in the Length/Type field, then starts dropping bytes (including the FCS field). The state machine of the RPE module decodes the Length/Type field and checks for the Length interpretation.

If the Length/Type field is greater than or equal to 600 (hex), the RPE module will send all received Ethernet frame data to the RFC module, irrespective of the value on the programmed auto-CRC strip option.

As a default, the GMAC is programmed for watchdog timer to be enabled, that is, frames above 2,048 (10,240 if Jumbo Frame is enabled) bytes (DA + SA + LT + DATA + PAD + FCS) are cut off at the RPE module. This feature can be disabled by programming the GMAC Configuration register, Watchdog Disable. However even if the watchdog timer is disabled, frames greater than 16 KB in size are cut off and a watchdog time-out status is given.

The GMAC supports loopback of transmitted frames onto its receiver. As a default, the GMAC loopback function is disabled, but this feature can be enabled by programming the GMAC Configuration register, Loopback bit. The transmit and receive clocks can have an asynchronous timing relationship, so an asynchronous FIFO is used to make the loopback path of the phy\_txd\_o data onto the receive path. The asynchronous FIFO is 10 bits (6 bits in 10/100-Mbps mode) wide to accommodate phy\_txd\_o, phy\_txen\_o, and phy\_txer\_o. The FIFO is five deep in 1000-Mbps mode (nine deep in 10/100-Mbps mode) and free-running to write on the write clock (clk\_tx\_i) and read on every read clock (clk\_rx\_i).

The write and read pointers gets re-initialized to have an offset of 2 (4 in 10/100-Mbps mode) at the start of each frame read out of the FIFO. This helps to avoid overflow/underflow during the transfer of a frame, and ensures that the overflow/underflow occurs only during the IFG period between the frames. Please note that the FIFO depth of five/nine is sufficient to prevent data corruption for frame sizes up to 9,022 bytes with a difference of 200 ppm between the (G)MII Transmit and Receive clock frequencies. Hence, bigger frames should not be looped back, as they may get corrupted in this loopback FIFO.

At the end of every received frame, the RPE module generates received frame status and sends it to the RFC module. Control, missed frame, and filter fail status are added to the receive status in the RFC module.

### 3.7.4.2 Receive CRC Module

The Receive CRC (CRX) interfaces to the RPE module to check for any CRC error in the receiving frame.

This module calculates the 32-bit CRC for the received frame that includes the Destination address field through the FCS field. The encoding is defined by the following generating polynomial.

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The module gets the data from the RPE module (DA+SA+LT+DATA+PAD+FCS). The RPE module also sends a control signal that indicates the validity of the data. Irrespective of the auto pad/CRC strip, the CRX module receives the entire frame to compute the CRC check for received frame. As a note on the auto pad/CRC strip settings, the entire frame is not transferred between the RPE and RFC 8-bit interface.

### 3.7.4.3 Receive Checksum Offload Engine

The Checksum Offload engine is optional (not available in the default configuration) and selectable during coreConsultant configuration, as described in [Table 6-17](#). Two types of Receive Checksum Offload engine are available for configuration as described below. When you select only Enable IP Checksum for Received frames, the Type 1 engine is instantiated in the core. When you select the second option, Full Checksum Offload, the Type 2 engine is instantiated. The first configuration was made available in the initial 3.0 release of the core, while the second is available as of release 3.30a. Type 1 configuration is retained for backward compatibility, as the Type 2 engine is generally preferable.

### 3.7.4.3.1 Type 1

The application can enable IP header checksum checking and TCP/UDP checksum offload by setting the GMAC Configuration register's IPC bit. This module calculates the 16-bit ones' complement of the Ethernet frame's payload data's (DATA field) ones' complement sum. The payload data is assumed to start from byte 15 (19 for a VLAN-tagged frame) of the received Ethernet frame. This module only processes IPv4 datagrams, bypassing and not processing all other types (such as IPv6).

This module also compares the calculated IP checksum with the received frame's IPv4 header checksum. Bytes 25 and 26 of the received Ethernet frame (29 and 30 for a VLAN-tagged frame) are taken as the IP header checksum. The header checksum is calculated against the header length field (20 bytes minimum). The result of the comparison (pass or fail) is given to the RFC, which sets the appropriate bit in the receive status word. If the Header Length field value is less than 5 or if the IP Version field does not equal 4, an error is indicated for the IP header checksum.

The ones' complement sum of the IP datagram's 16-bit payload is also calculated. The start of the payload is considered to be the data after the IP header. If the data payload ends with a non-aligned halfword, then a pad byte is added for the sum calculation. The 16-bit ones' complement of the resultant sum is forwarded to the RFC module, which inserts it into the data stream (towards the application) right after the FCS bytes (MS byte first) of the Ethernet payload. This 16-bit sum helps the software check the TCP/UDP header checksums faster. Note that this 16-bit sum (which is always appended to the Ethernet frame in this mode) is invalid when the IP header checksum bit shows an Error status.

### 3.7.4.3.2 Type 2

In this mode, both IPv4 and IPv6 frames in the received Ethernet frames are detected and processed for data integrity. As with Type 1, you can enable this module by setting the IPC bit in the GMAC Configuration register. The GMAC receiver identifies IPv4 or IPv6 frames by checking for value 0x0800 or 0x86DD, respectively, in the received Ethernet frames' Type field. This identification applies to VLAN-tagged frames as well.

The Receive Checksum Offload engine calculates IPv4 header checksums and checks that they match the received IPv4 header checksums. The result of this operation (pass or fail) is given to the RFC module for insertion into the receive status word. The IP Header Error bit is set for any mismatch between the indicated payload type (Ethernet Type field) and the IP header version, or when the received frame does not have enough bytes, as indicated by the IPv4 header's Length field (or when fewer than 20 bytes are available in an IPv4 or IPv6 header).

This engine also identifies a TCP, UDP or ICMP payload in the received IP datagrams (IPv4 or IPv6) and calculates the checksum of such payloads properly, as defined in the TCP, UDP, or ICMP specifications. This engine includes the TCP/UDP/ICMPv6 pseudo-header bytes for checksum calculation and checks whether the received checksum field matches the calculated value. The result of this operation is given as a Payload Checksum Error bit in the receive status word. This status bit is also set if the length of the TCP, UDP, or ICMP payload does not tally to the expected payload length given in the IP header.

As mentioned in “[TCP/UDP/ICMP Checksum Engine](#)” on page 84, this engine bypasses the payload of fragmented IP datagrams, IP datagrams with security features, IPv6 routing headers, and payloads other than TCP, UDP or ICMP. This information (whether the checksum engine is bypassed or not) is given in the receive status, as described in [Table 3-14](#).

In this configuration, the core *does not* append any payload checksum bytes to the received Ethernet frames (as the Type 1 engine does).

### 3.7.4.4 Receive Frame Controller Module

The Receive Frame Controller (RFC) receives the Ethernet frame data and status from the RPE module. The RFC module consists of a FIFO of parameterized depth (default set to 4 deep and 37 bits wide) and two state machines for writing and reading the FIFO. The FIFO holds the received Ethernet frame data and byte enables, along with a control bit to indicate the last data. The state machines manage the FIFO and provide a frame buffering for the receiving Ethernet frame from the RPE module. The main functions of the RFC module are:

- ❖ Data path conversion, which converts the 8-bit data to 32-bit data to the RBU module.
- ❖ Frame filtering
- ❖ Attaching the calculated IP Checksum input from IPC.
- ❖ Update the Receive Status and forward to RBU.

If the RA bit of the GMAC CSR Frame Filter register is set, the RFC module initiates the data transfer to the RBU module as soon as 4 bytes of Ethernet data are received from the RPE module. At the end of the data transfer, the RFC module sends out the received frame status that includes the frame filter bits (SA Filterfail and DAFilterfail) and status from the RFC module. These bits are generated based on the filter-fail signals from the AFM module. This status bit indicates to the Application whether the received frame has passed the filter controls (both address filter and Frame Filter controls from CSR). The RFC module will not drop any frame on its own in this mode.

If the RA bit is reset, the RFC module performs frame filtering based on the destination/source address (the Application still needs to perform another level of filtering if it decides not to receive any bad frames like runt, CRC error frames, etc. The RFC module waits to receive the first 14 bytes of received data (type field) from the RPE module. Until then, the module will not initiate any transfers to the RBU module. After receiving the destination/source address bytes, the RFC checks the filter-fail signal from the AFM module for an address match. On detecting a filter-fail from AFB, the frame is dropped at the RFC module and not transferred to the Application.

On a delayed filter response from the AFM (this can only occur if you change the AFM logic), the RFC module waits until the FIFO is full, and then proceeds with the frame transfer to the RBU module. However, it will still take the delayed response from the AFM module and if it is a (DA/SA) filter failure, then it will drop the rest of the frame and send the Rx Status Word (with zero frame-length, CRC Error and Runt Error bits set) immediately indicating the filter-fail. If there is no response from the AFM until the end of frame is transmitted, the filter fail status in the Rx Status Word is updated accordingly.

When the optional PMT module is present and configured for power-down mode, all received frames are dropped by this block, and are not forwarded to the application.

### 3.7.4.5 Receive Flow Control Module

The Receive Flow Controller (FRX) detects the receiving Pause frame and pauses the frame transmission for the delay specified within the received Pause frame. The FRX module is enabled only in Full-Duplex mode. The Pause frame detection function can be enabled or disabled with the RFE bit of GMAC CSR Register 6.

Once the receive flow control is enabled, the FRX module begins monitoring the received frame destination address for any match with the multicast address of the control frame (48'h0180C2000001). If a match is detected, the FRX module indicates to the RFC module, that the destination address of the received frame matches the reserved control frame destination address. The RFC module then decides whether or not to transfer the received control frame to the Application, based on the (PCF) bit setting of GMAC CSR Register 1 (Filter register).

The FRX module also decodes the Type, Op-code, and Pause Timer field of the receiving control frame. At the end of received frame, the FRX module gets the received frame status from RPE. If the byte count of the status indicates 64 bytes, and if there is no CRC error, the FRX module requests the MAC transmitter to pause the transmission of any data frame for the duration of the decoded Pause Time value, multiplied by the slot time (64 byte times for both 10/100-Mbps mode and 1000-Mbps mode). Meanwhile, if another Pause frame is detected with a zero Pause Time value, the FRX module resets the Pause Time and gives another pause request to the Transmitter. If the received control frame matches neither the Type field (16'h8808), Opcode (16'h000001), nor byte length (64 bytes), or if there is a CRC error, the FRX module does not generate a Pause request to Transmitter.

In the case of a pause frame with a multicast destination address, the RFC filters the frame based on the address match from the FRX module. For a pause frame with a unicast destination address, the filtering in the FRX module depends on whether the DA matched the contents of the MAC Address Register 0 and the UP Bit of GMAC Core Register 6 is set (detecting a pause frame even with a unicast destination address). The PCF register bits (Bit [7:6] of GMAC Register 1) controls the filtering for control frames in addition to the Address filter module.

#### 3.7.4.6 Receive Bus Interface Unit Module

The Receive Bus Interface Unit (RBU) converts the 32-bit data received from the RFC module into a 32/64/128-bit FIFO protocol on the Application side. The RBU module interfaces with the Application through the MAC receive interface (MRI). This block also performs the endian conversion if the GMAC-CORE is configured for Big Endian mode.

If IEEE 1588 time stamping is enabled, the RBU also outputs the time stamp captured from the received frame, along with the status on the mri\_timestamp\_o bus in GMAC-CORE configuration. The value on this bus is valid when the mri\_eof\_o signal is asserted.

#### 3.7.4.7 Address Filtering Module

The Address Filtering (AFM) module performs the destination and source address checking function on all received frames and reports the address filtering status to the RFC module. The address checking is based on different parameters (Frame Filter register) chosen by the Application. These parameters are inputs to the AFM module as control signals, and the AFM module reports the status of the address filtering based on the combination of these inputs. The AFM module does not filter the receive frames by itself, but reports the status of the address filtering (whether to drop the frame or not) to the RFC module. The AFM module also reports whether the receiving frame is a multicast frame or a broadcast frame, as well as the address filter status.

The AFM module probes the 8-bit receive data path between the RPE module and the RFC module and checks the destination and source address field of each incoming packet. In GMII mode, the module takes 8/14 clocks (from the start of frame) to compare the destination/source address of the receiving frame. Similarly, in MII mode the module takes 14/26 clocks (from the start of frame) to compare the destination/source address of the receiving frame. The AFM module gets the station's physical (MAC) address and the Multicast Hash table from CSR module for address checking. The CSR module provides the Frame Filter register parameters to AFM.

#### Unicast Destination Address Filter

The AFM supports up to 32 MAC addresses for unicast perfect filtering. If perfect filtering is selected (HUC bit of Frame Filter register is reset), the AFM compares all 48 bits of the received unicast address with the programmed MAC address for any match. Default MacAddr0 is always enabled, other addresses MacAddr1-MacAddr31 are selected with an individual enable bit. Each byte of these other addresses (MacAddr1-MacAddr31) can be masked during comparison with the corresponding received DA byte by

setting the corresponding Mask Byte Control bit in the register. This helps group address filtering for the DA.

In Hash filtering mode (When HUC bit is set), the AFM performs imperfect filtering for unicast addresses using a 64-bit Hash table. For hash filtering, the AFM uses the upper 6 bits CRC of the received destination address to index the content of the Hash table. A value of 000000 selects Bit 0 of the selected register, and a value of 111111 selects Bit 63 of the Hash Table register. If the corresponding bit (indicated by the 6-bit CRC) is set to 1, the unicast frame is said to have passed the Hash filter; otherwise, the frame has failed the Hash filter.



**Note** A sample C routine that generates the 6-bit hash for an input DA is provided in your workspace's resources/ directory.

## Multicast Destination Address Filter

The GMAC can be programmed to pass all multicast frames by setting the PM bit in the Frame Filter register. If the PM bit is reset, the AFM performs the filtering for multicast addresses based on the HMC bit of Frame Filter register. In Perfect Filtering mode, the multicast address is compared with the programmed MAC Destination Address registers (1-31). Group address filtering is also supported.

In Hash filtering mode, the AFM performs imperfect filtering using a 64-bit Hash table. For hash filtering, the AFM uses the upper 6 bits CRC of the received multicast address to index the content of the Hash table. A value of 000000 selects Bit 0 of the selected register and a value of 111111 selects Bit 63 of the Hash Table register.

If the corresponding bit is set to 1, then the multicast frame is said to have passed the Hash filter; otherwise, the frame has failed the Hash filter.

## Hash or Perfect Address Filter

The DA filter can be configured to pass a frame when its DA matches either the Hash filter or the Perfect filter by setting the HPF bit of the Frame Filter register and setting the corresponding HUC or HMC bits. This configuration applies to both unicast and multicast frames. If the HPF bit is reset, only one of the filters (Hash or Perfect) is applied to the received frame.

## Broadcast Address Filter

The AFM doesn't filter any broadcast frames in the default mode. However, if the GMAC is programmed to reject all broadcast frames by setting the DBF bit in the Frame Filter register, the DAF module asserts the Filter fail signal to RFC, whenever a broadcast frame is received. This will tell the RFC module to drop the frame.

## Unicast Source Address Filter

The GMAC can also perform a perfect filtering based on the source address field of the received frames. By default, the AFM compares the SA field with the values programmed in the SA registers. The MAC Address registers [1:31] can be configured to contain SA instead of DA for comparison, by setting Bit 30 of the corresponding Register. Group filtering with SA is also supported. The frames that fail the SA Filter are dropped by the GMAC if the SAF bit of Frame Filter register is set. Otherwise, the result of the SA filter is given as a status bit in the Receive Status word (see [Table 3-14](#)).

When SAF bit is set, the result of SA Filter and DA filter is AND'ed to decide whether the frame needs to be forwarded. This means that either of the filter fail result will drop the frame and both filters have to pass in-order to forward the frame to the application.

### Inverse Filtering Operation

For both Destination and Source address filtering, there is an option to invert the filter-match result at the final output. These are controlled by the DAIF and SAIF bits of the Frame Filter register respectively. The DAIF bit is applicable for both Unicast and Multicast DA frames. The result of the unicast/multicast destination address filter is inverted in this mode. Similarly, when the SAIF bit is set, the result of unicast SA filter is reversed.

[Tables 3-12](#) and [3-13](#) summarize the Destination and Source Address filtering based on the type of frames received.

**Table 3-12 Destination Address Filtering Table**

Frame Type	PR	HPF	HUC	DAIF	HMC	PM	DB	DA Filter Operation
Broadcast	1	X	X	X	X	X	X	Pass
	0	X	X	X	X	X	0	Pass
	0	X	X	X	X	X	1	Fail
Unicast	1	X	X	X	X	X	X	Pass all frames.
	0	X	0	0	X	X	X	Pass on Perfect/Group filter match.
	0	X	0	1	X	X	X	Fail on Perfect/Group filter match.
	0	0	1	0	X	X	X	Pass on Hash filter match.
	0	0	1	1	X	X	X	Fail on Hash filter match.
	0	1	1	0	X	X	X	Pass on Hash or Perfect/Group filter match.
Multicast	1	X	X	X	X	X	X	Pass all frames.
	X	X	X	X	X	1	X	Pass all frames.
	0	X	X	0	0	0	X	Pass on Perfect/Group filter match and drop PAUSE control frames if PCF = 0x.
	0	0	X	0	1	0	X	Pass on Hash filter match and drop PAUSE control frames if PCF = 0x.
	0	1	X	0	1	0	X	Pass on Hash or Perfect/Group filter match and drop PAUSE control frames if PCF = 0x.
	0	X	X	1	0	0	X	Fail on Perfect/Group filter match and drop PAUSE control frames if PCF = 0x.

**Table 3-12 Destination Address Filtering Table (Continued)**

Frame Type	PR	HPF	HUC	DAIF	HMC	PM	DB	DA Filter Operation
	0	0	X	1	1	0	X	Fail on Hash filter match and drop PAUSE control frames if PCF = 0x.
	0	1	X	1	1	0	X	Fail on Hash or Perfect/Group filter match and drop PAUSE control frames if PCF = 0x.

**Table 3-13 Source Address Filtering Table**

Frame Type	PR	SAIF	SAF	SA Filter Operation
Unicast	1	X	X	Pass all frames.
	0	0	0	Pass status on Perfect/Group filter match but do not drop frames that fail.
	0	1	0	Fail status on Perfect/Group filter match but do not drop frame.
	0	0	1	Pass on Perfect/Group filter match and drop frames that fail.
	0	1	1	Fail on Perfect/Group filter match and drop frames that fail.

### 3.7.5 MAC Receive Interface Protocol

The MRI interface connects the application to the receive data path of the GMAC core with a simple FIFO-protocol interface. The RBU initiates an Ethernet frame transfer by asserting the mri\_sof\_o along with the data (mri\_data\_o). All valid data transfers are indicated by an active high on the mri\_val\_o signal. The RBU module transfers the last data of the frame by driving the signal mri\_eof\_o along with the data. The number of byte lanes having valid data in the last transfer (EOF) is indicated by the mri\_be\_o[3:0] signal. The value of mri\_be\_o is the same as for mti\_be\_i as given in [Table 3-8](#), [Table 3-9](#), and [Table 3-10](#). The 32-bit Receive status (mri\_rxstatus\_o, described in [Table 3-14](#)) is also valid along with the EOF data transfer.

The GMAC always assumes that the application accepts the data output by it in one clock cycle. Hence it does not have any acknowledgement from the application before performing the next data transfer.



Bits 47-32 of mri\_rxstatus\_o described in [Table 3-14](#) are valid and applicable only when Advanced Time-stamping feature is enabled.

**Table 3-14 Receive Status at the GMAC Core Interface**

Bit	Description
47-46	Reserved
45	PTP Version When set indicates the received PTP message is having the IEEE 1588 version 2 format in the version field. When reset it has the version 1 format. This is valid only if the Message Type is non-zero. This bit is available only when Advanced time stamp feature is selected.

**Table 3-14 Receive Status at the GMAC Core Interface (Continued)**

Bit	Description
44	<b>PTP Frame Type</b> When set, this bit indicates that the PTP message is sent directly over Ethernet. This bit is available only when Advanced time stamp feature is selected.
43:40	<b>Message Type</b> These bits are encoded to give the type of the message received. <ul style="list-style-type: none"> <li>• 0000 – No PTP message received</li> <li>• 0001 – SYNC (all clock types)</li> <li>• 0010 – Follow_Up (all clock types)</li> <li>• 0011 – Delay_Req (all clock types)</li> <li>• 0100 – Delay_Resp (all clock types)</li> <li>• 0101 – Pdelay_Req (in peer-to-peer transparent clock) or Announce (in ordinary or boundary clock)</li> <li>• 0110 – Pdelay_Resp (in peer-to-peer transparent clock) or Management (in ordinary or boundary clock)</li> <li>• 0111 – Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock) or Signaling (for ordinary or boundary clock)</li> <li>• 1xxx – Reserved</li> </ul> This bit is available only when Advanced time stamp feature is selected
39	<b>IPv6 Packet Received</b> When set, this bit indicates that the received packet is an IPv6 packet. This bit is available only when Advanced time stamp feature is selected
38	<b>IPv4 Packet Received.</b> When set, this bit indicates that the received packet is an IPv4 packet. This bit is available only when Advanced time stamp feature is selected
37	<b>IP Checksum Bypassed</b> When set, this bit indicates that the checksum offload engine is bypassed. This bit is available only when Advanced time stamp feature is selected.
36	<b>IP Payload Error</b> When set, this bit indicates that the 16-bit IP payload checksum (that is, the TCP, UDP, or ICMP checksum) that the core calculated does not match the corresponding checksum field in the received segment. It is also set when the TCP, UDP, or ICMP segment length does not match the payload length value in the IP Header field. This bit is available only when Advanced time stamp feature is selected.
35	<b>IP Header Error</b> When set, this bit indicates either that the 16-bit IPv4 header checksum calculated by the core does not match the received checksum bytes, or that the IP datagram version is not consistent with the Ethernet Type value. This bit is available only when Advanced time stamp feature is selected.

**Table 3-14 Receive Status at the GMAC Core Interface (Continued)**

Bit	Description
34-32	<p><b>IP Payload Type</b></p> <p>These bits indicate the type of payload encapsulated in the IP datagram processed by the Receive Checksum Offload Engine (COE). The COE also sets these bits to 2'b00 if it does not process the IP datagram's payload due to an IP header error or fragmented IP.</p> <ul style="list-style-type: none"> <li>• 3'b000: Unknown or did not process IP payload</li> <li>• 3'b001: UDP</li> <li>• 3'b010: TCP</li> <li>• 3'b011: ICMP</li> <li>• 3'b1xx : Reserved</li> </ul> <p>This bit is available only when Advanced time stamp feature is selected.</p>
31:28	<p><b>Rx MAC Address or Payload Checksum Error</b></p> <p>When the Advanced Time Stamp feature is enabled, these bits, along with bit 27, indicate which of the 0-31 MAC Address Registers matches the DA field. If more than one register matches the DA field, the lower register number is given.</p> <p>When Advanced Time-Stamp feature and Full Checksum Offload Engine (Type 2) are not present, these bits indicate which Rx MAC Address register (0–15) values matched the frame's DA field. If more than one register matches the DA field, the lower register number is given. For matches with MAC Address registers 16–31, the value is 0.</p> <p>When Advanced Time-Stamp feature is not selected and Full Checksum Offload Engine (Type 2) is enabled, Bits[31:29] are reserved. When Bit 28 is set, it indicates the TCP, UDP, or ICMP checksum calculated by the core does not match the received encapsulated TCP, UDP, or ICMP segment's Checksum field. Bit 28 is also set when the received number of payload bytes does not tally to the value indicated by the Length field of the encapsulated IPv4/IPv6 datagram in the received Ethernet frame. See <a href="#">Table 3-15</a> for more detail.</p>
27	<p><b>Rx MAC Address or IP Header Checksum Error</b></p> <p>When the Advanced Time Stamp feature is enabled, this bit is used for MAC Address match as described in bits 31-28 above. Otherwise, when this bit is set, it indicates that the 16-bit IP Header checksum calculated by the core did not match the received Header Checksum bytes.</p> <p>When Full Checksum Offload Engine (Type 2) is enabled, this bit indicates an error in the IPv4 or IPv6 header. This error can be due to: inconsistent values between the Ethernet Type and IP Header Version fields, an IPv4 header checksum mismatch, or an Ethernet frame that does not have the expected number of IP header bytes. See <a href="#">Table 3-15</a> for more detail.</p>
26	<p><b>Control Frame</b></p> <p>When this bit is set, the control frame is received.</p>
25	<p><b>SA Filter Fail</b></p> <p>When this bit is set, the Source Address filter failed. This means that this frame did not clear the SA filter.</p>
24	<p><b>DAfilter Fail</b></p> <p>When this bit is set, the Destination Address filter failed. This means that this frame did not clear the DA filter.</p>
23	<p><b>Length Error</b></p> <p>When set, this bit indicates that the current frame Length value is inconsistent with the total number of bytes received in the current frame. This is valid when the Frame Type is set to '0' (802.3z Frame) and Frame Length value is less than 1500. Length error bit is invalid when CRC error is present in the status.</p>

**Table 3-14 Receive Status at the GMAC Core Interface (Continued)**

Bit	Description
22	VLAN Frame When this bit is set, the current reception is tagged with a VLAN ID. The 13th and 14th bytes at the frame are compared with 16'h8100, and the following 2-byte data is compared with the VLAN Tag register. This bit is set in response to a match.
21	CRC Error When set, this bit indicates that a cyclic redundancy check (CRC) error occurred on the received frame
20	Dribbling Bit When set, this bit indicates that the frame contained a non-integer multiple of eight bits (valid only in MII mode). This bit is not valid if a collision is seen or if runt frame bits are set. If this bit is set and the CRC error is reset, then the packet is valid.
19	GMII Error When set, this bit indicates that the gmii_rxer_i was asserted during the reception of this frame. This error also includes carrier extension error in GMII and Half-duplex mode. Error can be of less/no extension, or error (rxd = 1f) during extension.
18	Frame Type When set, this bit indicates that the frame is an Ethernet-type frame (frame length field is greater than or equal to 0x600). When clear, it indicates that the frame is an IEEE 802.3z frame. This bit is not valid for runt frames of less than 14 bytes. See <a href="#">Table 3-15</a> for more detail.
17	Late Collision Seen When this bit is set, it indicates that the frame may be damaged by a late collision (active gmii_col_i signal seen after 64 bytes after SFD in Half-Duplex mode) during the reception of the frame.
16	Giant Frame When set, this bit indicates that the frame length exceeds the maximum Ethernet specified size of 1,518/1,522 bytes (9,018/9,022 bytes if jumbo frame enable is set). Giant frame is only a frame length indication, and it does not cause any frame truncation.
15	Runt Frame When this bit is set, it indicates that the GMAC has received frames of less than 64 bytes or a collision was observed before the receipt of 64th byte.
14	Watchdog Timeout When set, this bit indicates that the RPE module has received a frame with byte count greater than 2,048 (10,240 if Jumbo frame is enabled) bytes (DA + SA + LT + DATA + PAD + FCS). This bit is not set if the watchdog timer is disabled in the Configuration register. However, even if the watchdog timer is disabled, this bit is set when the frame is greater than 16 KB in size.
13:0	Frame Length Indicates the length, in bytes, of the received frame (including pad if applicable), and/or the FCS field when the Auto Pad/CRC Strip bit is deasserted. Otherwise, it indicates the length without pad and/or FCS field. It also includes the 2-byte IP Checksum appended after the FCS when the IP Checksum module (Type 1) is present and enabled by setting the IPC (Bit 10 of GMAC Register 0), and if the received frame is not a MAC control frame.

As mentioned in [Table 3-14](#), when Full Checksum Offload Engine (Type 2) is enabled, the meaning of certain register bits change. Their significance is mapped in [Table 3-15](#).

**Table 3-15 Frame Status with Full Checksum Offload Engine Enabled and Advanced Time-Stamping not Present**

Bit 18: Ethernet Frame	Bit 27: Header Checksum Error	Bit 28: Payload Checksum Error	Frame Status
0	0	0	The frame is an IEEE 802.3 frame (Length field value is less than 0x0600).
1	0	0	IPv4/IPv6 Type frame in which no checksum error is detected.
1	0	1	IPv4/IPv6 Type frame in which a payload checksum error (as described for PCE) is detected
1	1	0	IPv4/IPv6 Type frame in which IP header checksum error (as described for IPC HCE) is detected.
1	1	1	IPv4/IPv6 Type frame in which both PCE and IPC HCE is detected.
0	0	1	IPv4/IPv6 Type frame in which there is no IP HCE and the payload check is bypassed due to unsupported payload.
0	1	1	Type frame which is neither IPv4 or IPv6 (COE bypasses the checksum check completely)
0	1	0	Reserved



**Note** The first five conditions are backward-compatible to versions 3.30a and previous. The last two conditions (001, 011), which had been reserved, are not backward-compatible, because the FT bit is reset during bypasses, even for valid Type frames.

### 3.7.6 MAC Receive Timing

Timing specifications for the MAC Core (GMAC-CORE) interface signals in the receive path are illustrated in detail, in the following sections.

#### 3.7.6.1 MAC Receive Interface (MRI)

The timing diagrams in this section illustrate the timing specifications for the MAC receive interface signals and the GMII receive interface. The sequence of events is as follows:

1. The MAC receives frames on the (G)MII interface.
2. It stores the frame until it makes the decision to filter the frame, if configured so. When the frame is passed by the filter, the MAC puts the data on mri\_data\_o and indicates the validity by asserting mri\_val\_o. It also asserts mri\_sof\_o, if the data corresponds to the start of a frame.
3. The application must accept the data as soon as the MAC presents it because the MAC may output the next data (if available) in the next clock cycle itself.
4. The MAC indicates transfer of the end-of-frame data with the assertion of mri\_eof\_o. Signal mri\_be\_o may also have a non all-one value during the EOF, showing how many lanes on the data bus have valid data.
5. The MAC updates the receive status bits on mri\_rxstatus\_o during the transfer of the EOF.

**Figure 3-57 MAC Receive Interface Timing (1 of 4)**

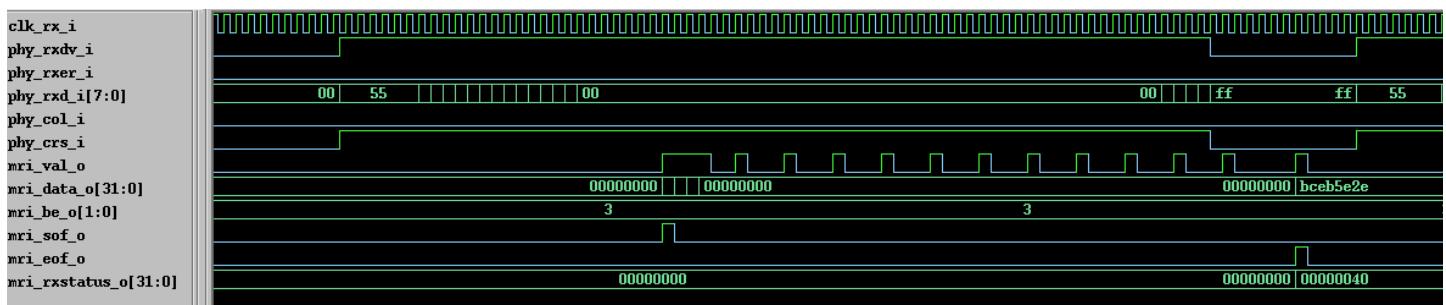
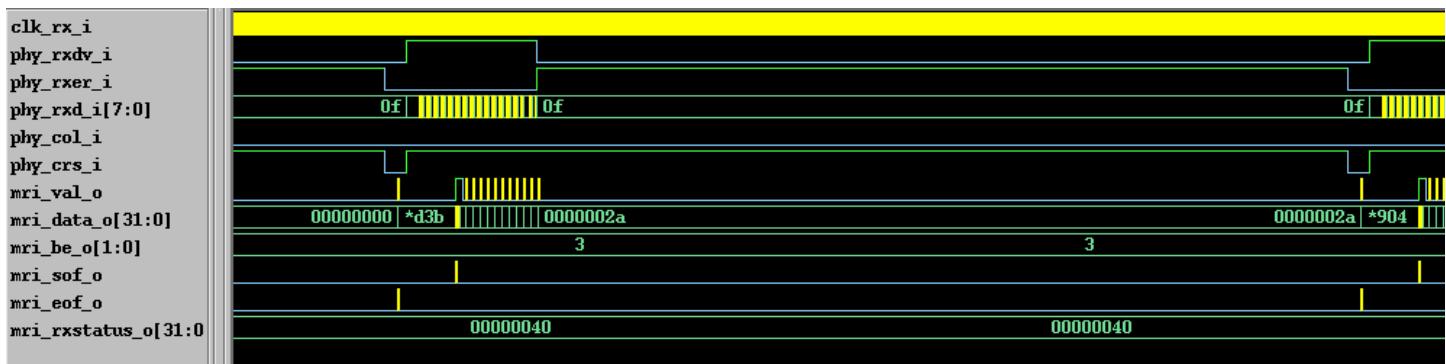
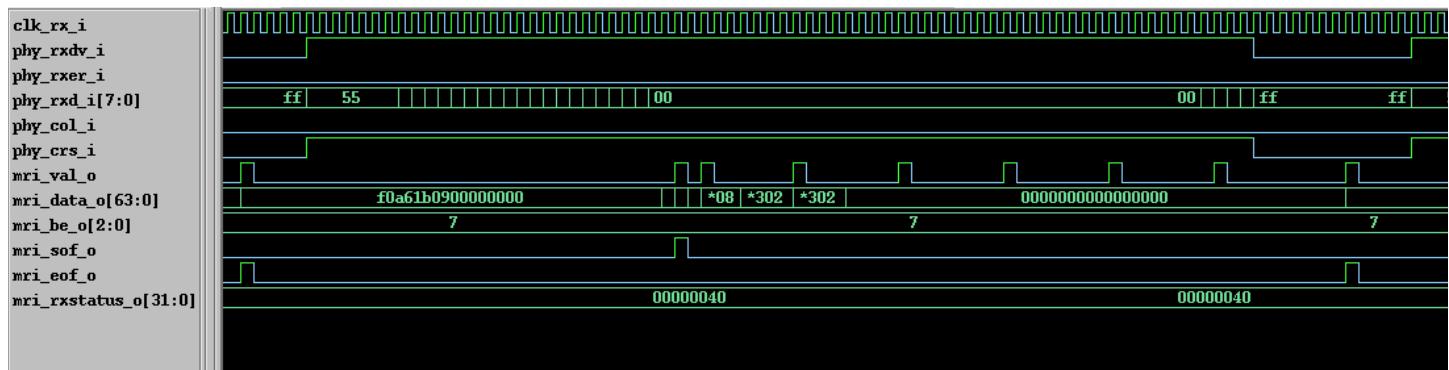
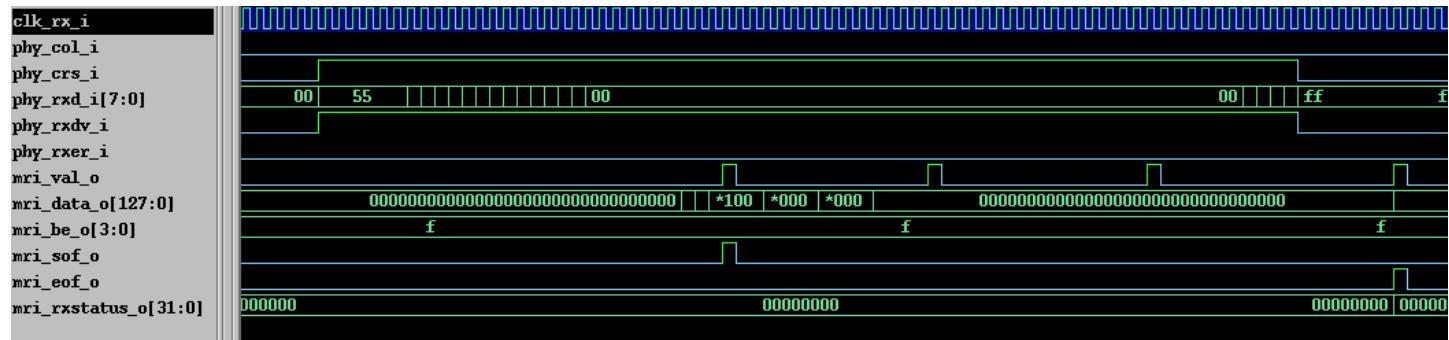


Figure 3-58 illustrates the reception and transfer of a frame in half-duplex mode/gigabit mode. Note that the EOF is not transferred until the completion of the frame-extension so that valid receive status can be output together.

**Figure 3-58 MAC Receive Interface Timing (2 of 4)**

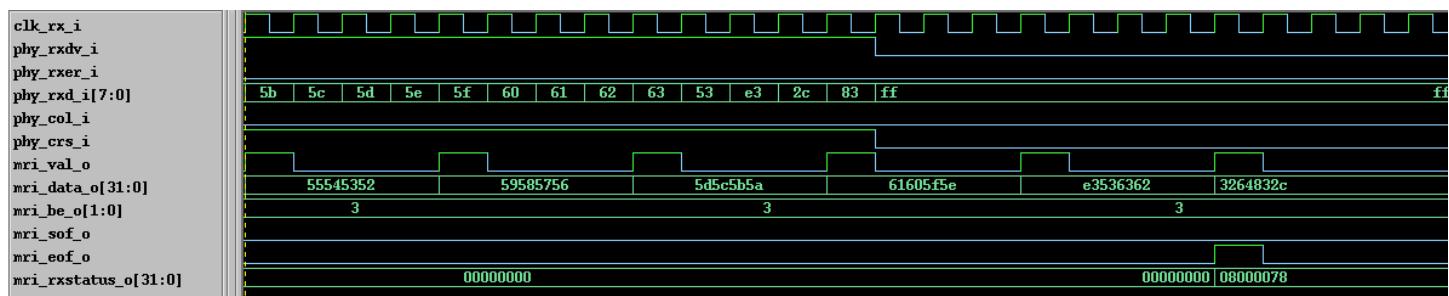


Figures 3-59 and 3-60 show the timing of the MRI, with 64-bit and 128-bit data bus configurations.

**Figure 3-59 MAC Receive Interface Timing (3 of 4)****Figure 3-60 MAC Receive Interface Timing (4 of 4)**

### 3.7.6.2 Reception of Frame With IP Payload Checksum (Type 1)

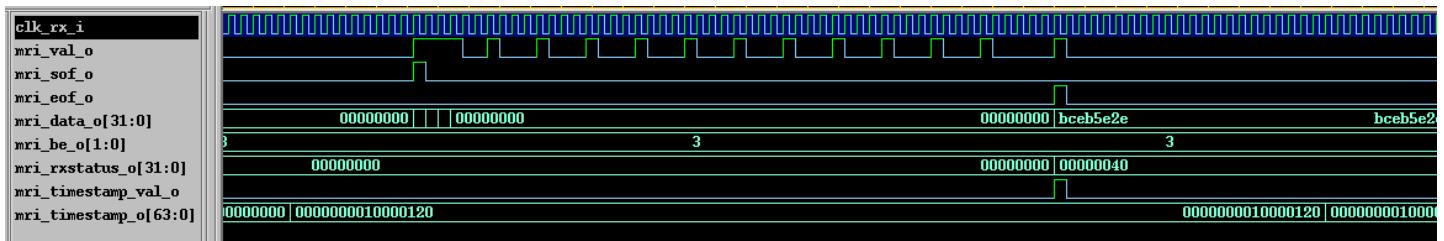
Figure 3-61 shows how the MAC receiver adds the 16-bit IP payload checksum to the frame transferred on the MRI. The normal end of a frame byte is 0x83 while the 16-bit IP payload checksum 0x6432 is attached to the frame data during the EOF. If the frame is such that the IP payload checksum spills over to the next transaction, then the EOF is indicated at that point.

**Figure 3-61 Reception of Frame Along With IP Payload Checksum Timing**

### 3.7.6.3 Frame Reception With Time-Stamping Enabled

Figure 3-62 shows the MRI timing when IEEE 1588 time stamping is enabled. When the MRI outputs the EOF data, as marked by mri\_eof\_o high, the time stamp value is also given on mri\_timestamp\_o. The time stamp is validated with the assertion of mri\_timestamp\_val\_o. When Advanced time-stamp feature is selected, the width of the status is 48-bits.

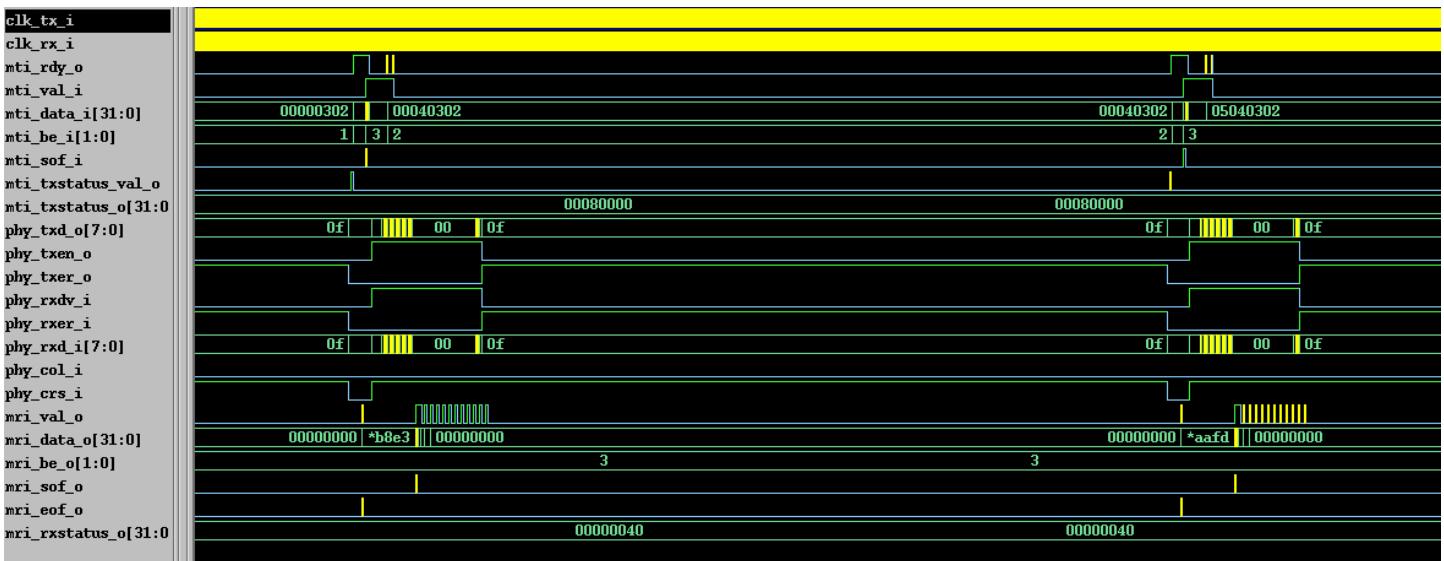
**Figure 3-62 MRI Timing with IEEE 1588 Time Stamping Enabled**



### 3.7.6.4 Frame Transmission and Reception

Figure 3-63 shows both the MTI and MRI interface signal timing along with the GMII signal timing for a frame transmission that is looped back to the GMII receiver in the testbench environment.

**Figure 3-63 Frame Transmission and Reception**



### 3.7.7 MAC Control Interface Protocol

This interface connects the Application with the Control and Status register (CSR) module of the Media Access Controller to provide the control path for programming the MAC registers. The Application initiates a Register Write/Read by asserting the command valid signal `mci_val_i` (with the valid data `mci_wdata_i[31:0]` and valid byte enables `mci_be_i[3:0]`), along with the register address on `mci_addr_i[12:0]`. The control signal `mci_rdwn_i` indicates a Write (1'b0) or Read (1'b1) operation. The Application assumes a successful data transfer when it receives an acknowledgement (`mci_ack_o`) asserted for 1 clock cycle. The CSR module drives the Read data on `mci_rdata[31:0]`. The Application can deassert the command valid once the acknowledgement is received.

If the Application continues to assert the command valid signal (on receiving an acknowledgement) then the GMAC considers it as the next access and will respond to it by asserting the `mci_ack_o` after 1 clock cycle. Accesses to reserved registers will be completed by the GMAC with dummy read or write operations. In other words, writes to reserved addresses have no effect and read operations are completed with an all-zero data output on the `mci_rdata_o` bus.

All read and write accesses to the CSR completes in 2 clock cycles. i.e, `mac_ack_o` will be asserted after a delay of 1 clock cycle after `mci_val_i` goes high. This enables the user to map the MCI signals to an APB port

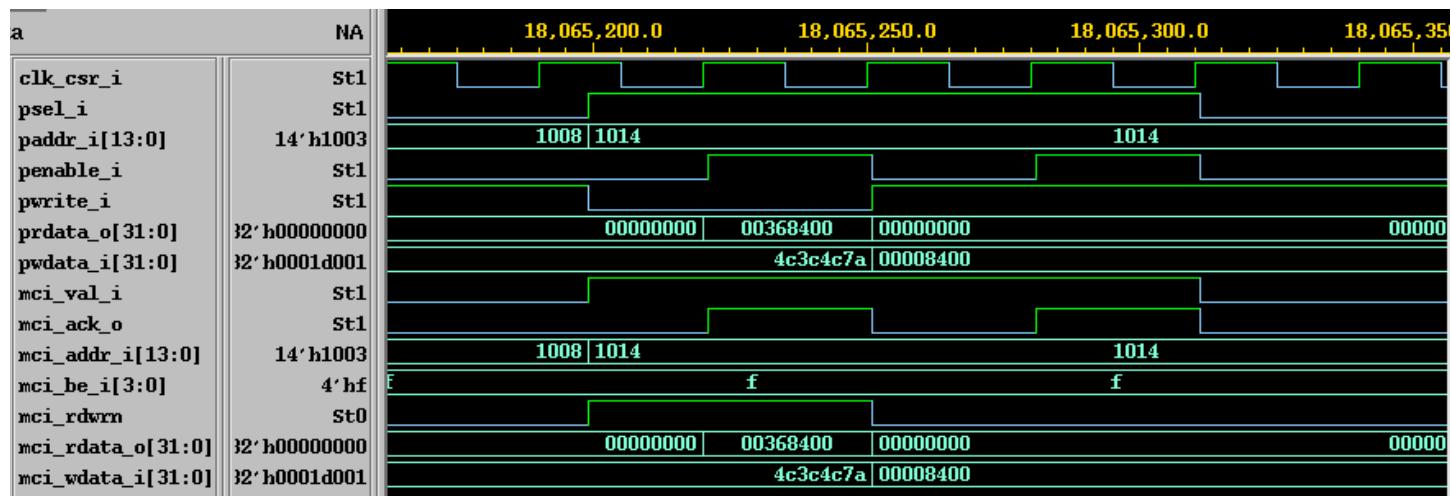
signals with minimal glue logic. For non-APB/AHB/AXI Slave interface configuration for CSR, mci\_rdata\_o is registered and hence there is an additional delay of 1 clock for the assertion of mci\_ack\_o. Thus, each read access will complete in 3 clock cycles for such configurations.

### 3.7.7.1 APB Port Timing

[Figure 3-64](#) shows a register read access followed by a register write access on the APB port. The flow is as follows:

1. The APB port signals behave as per the AMBA 2.0 standard specifications.
2. The mci\_\* signals shown in the figure are the internal MCI interface signals. Note that signals psel\_i, paddr\_i, prdata\_o, and pwdata\_i are the same as mci\_val\_i, mci\_addr\_i, mci\_rdata\_o and mci\_wdata\_i, respectively. Internally, the APB port signals are directly connected to the corresponding MCI signals. Signal mci\_rdwrn\_i is the invert of pwrite\_i. In APB port configuration, the internal mci\_ack\_o signal is an input tied to penable\_i.

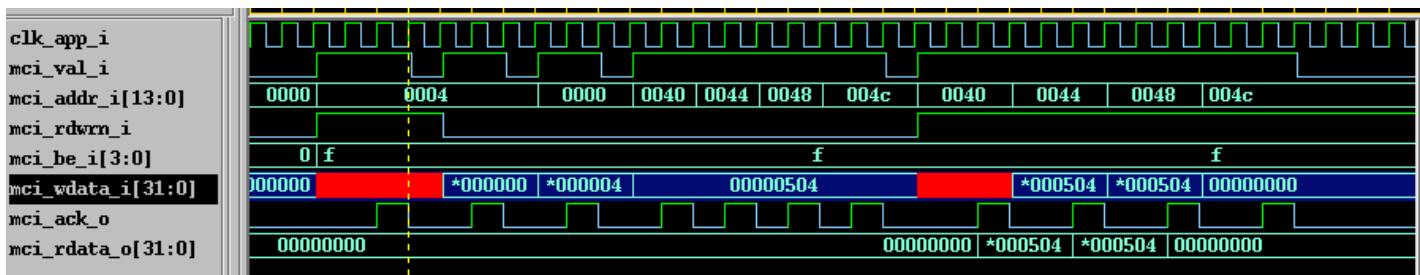
**Figure 3-64 APB Port Timing**



### 3.7.7.2 MCI Port Timing

[Figure 3-65](#) shows read and write accesses on the MCI port. The flow is as follows:

1. The first access to address 0x0004 is a single read cycle which completes in 3 clock cycles. Signal mci\_val\_i and the corresponding address and control signals should be kept asserted for the whole duration of the transaction.
2. The second and third access are single write operation cycles to address 0x0004 and 0x0000, respectively. Signal mci\_ack\_o is asserted 1 clock after mci\_val\_i is asserted. Thus the write operations are completed in 2 clock cycles.
3. The following accesses are burst write and burst read operations starting from address 0x0040. Note that each individual write/read transaction in the burst is identical to the “single” write/read transactions, with respect to timing.

**Figure 3-65 MCI Port Timing**

### 3.7.8 System Time Register Module

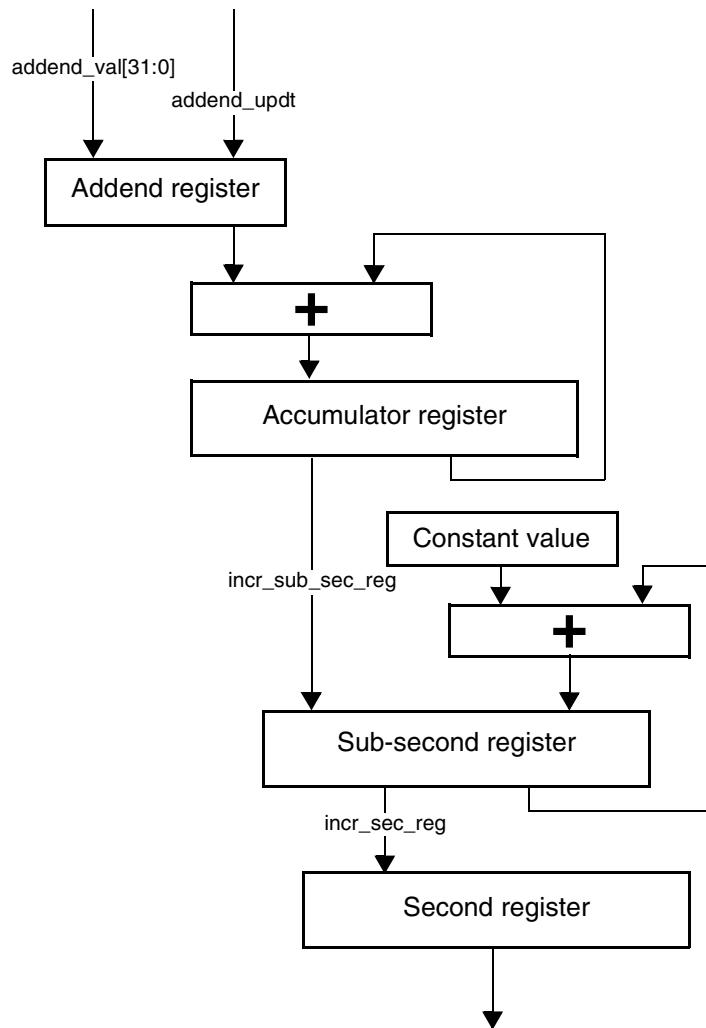
The System Time Generator module is optional and is not available if external time updating is enabled. 64-bit time is maintained in this module and updated using the input reference clock (clk\_ptp\_ref\_i). This time is the source for taking snapshots (time stamps) of Ethernet frames being transmitted or received at the GMII.

The System Time counter can be initialized or corrected using the *coarse correction method*. In this method, the initial value or the offset value is written to the Time Stamp Update register (“[IEEE 1588 Time Stamp Registers](#)” on page 301). For initialization, the System Time counter is written with the value in the Time Stamp Update registers, while for system time correction, the offset value is added to or subtracted from the system time.

In the *fine correction method*, a slave clock’s (clk\_ptp\_ref\_i) frequency drift with respect to the master clock (as defined in IEEE 1588) is corrected over a period of time instead of in one clock, as in coarse correction. This helps maintain linear time and does not introduce drastic changes (or a large jitter) in the reference time between PTP Sync message intervals. In this method, an accumulator sums up the contents of the Addend register, as shown in [Figure 3-66](#). The arithmetic carry that the accumulator generates is used as a pulse to increment the system time counter. The accumulator and the addend are 32-bit registers. Here, the accumulator acts as a high-precision frequency multiplier or divider.

This algorithm is depicted in [Figure 3-66](#):

**Figure 3-66 System Time Update Using Fine Method**



The System Time Update logic requires a 50-MHz clock frequency to achieve 20-ns accuracy. The frequency division is the ratio of the reference clock frequency to the required clock frequency. Hence, if the reference clock (clk\_ptp\_ref\_i) is, for example, 66 MHz, this ratio is calculated as  $66 \text{ MHz} / 50 \text{ MHz} = 1.32$ . Hence, the default addend value to be set in the register is  $2^{32} / 1.32$ , 0xC1F07C1F.

If the reference clock drifts lower, to 65 MHz for example, the ratio is  $65 / 50$ , or 1.3 and the value to set in the addend register is  $2^{32} / 1.30$ , or 0xC4EC4EC4. If the clock drifts higher, to 67 MHz for example, the addend register must be set to 0xBF0B7672. When the clock drift is nil, the default addend value of 0xC1F07C1F ( $2^{32} / 1.32$ ) must be programmed.

In [Figure 3-66](#), the constant value used to accumulate the sub-second register is decimal 43, which achieves an accuracy of 20 ns in the system time (in other words, it is incremented in 20-ns steps). The optional System Time module is unavailable when External Time Update is enabled. Two different methods are used to update the System Time register, depending on which configuration you choose (See “[Parameters](#)” on [page 309](#)).

The software must calculate the drift in frequency based on the Sync messages and update the Addend register accordingly.

Initially, the slave clock is set with FreqCompensationValue0 in the Addend register. This value is as follows:

$$\text{FreqCompensationValue}_0 = 2^{32} / \text{FreqDivisionRatio}$$

If MasterToSlaveDelay is initially assumed to be the same for consecutive Sync messages, the algorithm described below must be applied. After a few Sync cycles, frequency lock occurs. The slave clock can then determine a precise MasterToSlaveDelay value and re-synchronize with the master using the new value.

The algorithm is as follows:

- ❖ At time  $\text{MasterSyncTime}_n$  the master sends the slave clock a Sync message. The slave receives this message when its local clock is  $\text{SlaveClockTime}_n$  and computes  $\text{MasterClockTime}_n$  as:

$$\text{MasterClockTime}_n = \text{MasterSyncTime}_n + \text{MasterToSlaveDelay}_n$$

- ❖ The master clock count for current Sync cycle,  $\text{MasterClockCount}_n$  is given by:

$$\text{MasterClockCount}_n = \text{MasterClockTime}_n - \text{MasterClockTime}_{n-1} \text{ (assuming that MasterToSlaveDelay is the same for Sync cycles } n \text{ and } n-1\text{)}$$

- ❖ The slave clock count for current Sync cycle,  $\text{SlaveClockCount}_n$  is given by:

$$\text{SlaveClockCount}_n = \text{SlaveClockTime}_n - \text{SlaveClockTime}_{n-1}$$

- ❖ The difference between master and slave clock counts for current Sync cycle,  $\text{ClockDiffCount}_n$  is given by:

$$\text{ClockDiffCount}_n = \text{MasterClockCount}_n - \text{SlaveClockCount}_n$$

- ❖ The frequency-scaling factor for slave clock,  $\text{FreqScaleFactor}_n$  is given by:

$$\text{FreqScaleFactor}_n = (\text{MasterClockCount}_n + \text{ClockDiffCount}_n) / \text{SlaveClockCount}_n$$

- ❖ The frequency compensation value for Addend register,  $\text{FreqCompensationValue}_n$  is given by:

$$\text{FreqCompensationValue}_n = \text{FreqScaleFactor}_n * \text{FreqCompensationValue}_{n-1}$$

In theory, this algorithm achieves lock in one Sync cycle; however, it may take several cycles, due to changing network propagation delays and operating conditions.

This algorithm is self-correcting: if for any reason the slave clock is initially set to a value from the master that is incorrect, the algorithm will correct it at the cost of more Sync cycles.

## 3.8 MAC Management Counters

The counters in the MAC Management Counters (MMC) module can be viewed as an extension of the register address space of the CSR module. The MMC module maintains a set of registers for gathering statistics on the received and transmitted frames. These include a control register for controlling the behavior of the registers, two 32-bit registers containing interrupts generated (receive and transmit), and two 32-bit registers containing masks for the Interrupt register (receive and transmit). These registers are accessible from the Application through the MAC Control Interface (MCI). Each register is 32 bits wide. The write data is qualified with the corresponding mci\_be\_i signals. Thus, non-32-bit accesses are allowed as long as the address is word-aligned.

The organization of these registers is shown in [Table 3-16](#). The MMCs are accessed using transactions, in the same way the CSR address space is accessed. The following sections in the chapter describe the various counters and list the address for each of the statistics counters. This address will be used for Read/Write accesses to the desired transmit/receive counter.

The Receive MMC counters are updated for frames that are passed by the Address Filter (AFM) block. Statistics of frames that are dropped by the AFM module are not updated unless they are runt frames of less than 6 bytes (DA bytes are not received fully).

The MMC module gathers statistics on encapsulated IPv4, IPv6, TCP, UDP, or ICMP payloads in received Ethernet frames. This gathering is only enabled when Full Checksum Offload Engine is selected during RTL configuration. The address map of the corresponding registers, 0x0200–0x02FC, is given in [Table 3-16](#).

You can select the MMC counters individually during coreKit configuration. Addresses for counters that are not selected become reserved.

### 3.8.1 Address Assignments

The MMC register naming convention is as follows.

- ❖ “tx” as a prefix or suffix indicates counters associated with transmission
- ❖ “rx” as a prefix or suffix indicates counters associated with reception
- ❖ “\_g” as a suffix indicates registers that count good frames only
- ❖ “\_gb” as a suffix indicates registers that count frames regardless of whether they are good or bad

The following explanations pertain to terminology used in [Tables 3-16](#) through [3-23](#).

Transmitted frames are considered “good” if transmitted successfully. In other words, a transmitted frame is good if the frame transmission is not aborted due to any of the following errors:

- ❖ Jabber Timeout
- ❖ No Carrier/Loss of Carrier
- ❖ Late Collision
- ❖ Frame Underflow
- ❖ Excessive Deferral
- ❖ Excessive Collision

Received frames are considered “good” if none of the following errors exists:

- ❖ CRC error
- ❖ Runt Frame (shorter than 64 bytes)
- ❖ Alignment error (in 10/100 Mbps only)
- ❖ Length error (non-Type frames only)
- ❖ Out of Range (non-Type frames only, longer than maximum size)
- ❖ GMII\_RXER Input error

The maximum frame size depends on the frame type, as follows:

- ❖ Untagged frame maxsize = 1518
- ❖ VLAN Frame maxsize = 1522
- ❖ Jumbo Frame maxsize = 9018
- ❖ JumboVLAN Frame maxsize = 9022

**Table 3-16 MMC Register Map**

<b>GMAC CSR Register No.</b>	<b>Address Offset</b>	<b>Register Name</b>	<b>Register Description</b>
64	0x0100	mmc_cntrl	MMC Control establishes the operating mode of MMC. For more details, refer to <a href="#">Table 3-17</a> .
65	0x0104	mmc_intr_rx	MMC Receive Interrupt maintains the interrupt generated from all of the receive statistic counters. For more details, refer to <a href="#">Table 3-18</a> .
66	0x0108	mmc_intr_tx	MMC Transmit Interrupt maintains the interrupt generated from all of the transmit statistic counters. For more details, refer to <a href="#">Table 3-19</a> .
67	0x010C	mmc_intr_mask_rx	MMC Receive Interrupt mask maintains the mask for the interrupt generated from all of the receive statistic counters. For more details, refer to <a href="#">Table 3-20</a> .
68	0x0110	mmc_intr_mask_tx	MMC Transmit Interrupt Mask maintains the mask for the interrupt generated from all of the transmit statistic counters. For more details, refer to <a href="#">Table 3-21</a> .
69	0x0114	txoctetcount_gb	Number of bytes transmitted, exclusive of preamble and retried bytes, in good and bad frames.
70	0x0118	txframecount_gb	Number of good and bad frames transmitted, exclusive of retried frames.
71	0x011C	txbroadcastframes_g	Number of good broadcast frames transmitted.
72	0x0120	txmulticastframes_g	Number of good multicast frames transmitted.
73	0x0124	tx64octets_gb	Number of good and bad frames transmitted with length 64 bytes, exclusive of preamble and retried frames.
74	0x0128	tx65to127octets_gb	Number of good and bad frames transmitted with length between 65 and 127 (inclusive) bytes, exclusive of preamble and retried frames.
75	0x012C	tx128to255octets_gb	Number of good and bad frames transmitted with length between 128 and 255 (inclusive) bytes, exclusive of preamble and retried frames.
76	0x0130	tx256to511octets_gb	Number of good and bad frames transmitted with length between 256 and 511 (inclusive) bytes, exclusive of preamble and retried frames.
77	0x0134	tx512to1023octets_gb	Number of good and bad frames transmitted with length between 512 and 1,023 (inclusive) bytes, exclusive of preamble and retried frames.
78	0x0138	tx1024tomaxoctets_gb	Number of good and bad frames transmitted with length between 1,024 and maxsize (inclusive) bytes, exclusive of preamble and retried frames.

**Table 3-16 MMC Register Map (Continued)**

<b>GMAC CSR Register No.</b>	<b>Address Offset</b>	<b>Register Name</b>	<b>Register Description</b>
79	0x013C	txunicastframes_gb	Number of good and bad unicast frames transmitted.
80	0x0140	txmulticastframes_gb	Number of good and bad multicast frames transmitted.
81	0x0144	txbroadcastframes_gb	Number of good and bad broadcast frames transmitted.
82	0x0148	txunderflowerror	Number of frames aborted due to frame underflow error.
83	0x014C	txsinglecol_g	Number of successfully transmitted frames after a single collision in Half-duplex mode.
84	0x0150	txmulticol_g	Number of successfully transmitted frames after more than a single collision in Half-duplex mode.
85	0x0154	txdeferred	Number of successfully transmitted frames after a deferral in Half-duplex mode.
86	0x0158	txlatecol	Number of frames aborted due to late collision error.
87	0x015C	txexesscol	Number of frames aborted due to excessive (16) collision errors.
88	0x0160	txcarriererror	Number of frames aborted due to carrier sense error (no carrier or loss of carrier).
89	0x0164	txoctetcount_g	Number of bytes transmitted, exclusive of preamble, in good frames only.
90	0x0168	txframecount_g	Number of good frames transmitted.
91	0x016C	txexcessdef	Number of frames aborted due to excessive deferral error (deferred for more than two max-sized frame times).
92	0x0170	txpauseframes	Number of good PAUSE frames transmitted.
93	0x0174	txvlanframes_g	Number of good VLAN frames transmitted, exclusive of retried frames.
94	0x0178	Reserved	
95	0x017C	Reserved	
96	0x0180	rxframecount_gb	Number of good and bad frames received.
97	0x0184	rxoctetcount_gb	Number of bytes received, exclusive of preamble, in good and bad frames.
98	0x0188	rxoctetcount_g	Number of bytes received, exclusive of preamble, only in good frames.
99	0x018C	rbroadcastframes_g	Number of good broadcast frames received.
100	0x0190	rmulticastframes_g	Number of good multicast frames received.
101	0x0194	rxcrcerror	Number of frames received with CRC error.

**Table 3-16 MMC Register Map (Continued)**

<b>GMAC CSR Register No.</b>	<b>Address Offset</b>	<b>Register Name</b>	<b>Register Description</b>
102	0x0198	rxalignmenterror	Number of frames received with alignment (dribble) error. Valid only in 10/100 mode.
103	0x019C	rxrunterror	Number of frames received with runt (<64 bytes and CRC error) error.
104	0x01A0	rxjabbererror	Number of giant frames received with length (including CRC) greater than 1,518 bytes (1,522 bytes for VLAN tagged) and with CRC error. If Jumbo Frame mode is enabled, then frames of length greater than 9,018 bytes (9,022 for VLAN tagged) are considered as giant frames.
105	0x01A4	rxundersize_g	Number of frames received with length less than 64 bytes, without any errors.
106	0x01A8	rxoversize_g	Number of frames received with length greater than the maxsize (1,518 or 1,522 for VLAN tagged frames), without errors.
107	0x01AC	rx64octets_gb	Number of good and bad frames received with length 64 bytes, exclusive of preamble.
108	0x01B0	rx65to127octets_gb	Number of good and bad frames received with length between 65 and 127 (inclusive) bytes, exclusive of preamble.
109	0x01B4	rx128to255octets_gb	Number of good and bad frames received with length between 128 and 255 (inclusive) bytes, exclusive of preamble.
110	0x01B8	rx256to511octets_gb	Number of good and bad frames received with length between 256 and 511 (inclusive) bytes, exclusive of preamble.
111	0x01BC	rx512to1023octets_gb	Number of good and bad frames received with length between 512 and 1,023 (inclusive) bytes, exclusive of preamble.
112	0x01C0	rx1024tomaxoctets_gb	Number of good and bad frames received with length between 1,024 and maxsize (inclusive) bytes, exclusive of preamble and retried frames.
113	0x01C4	rxunicastframes_g	Number of good unicast frames received.
114	0x01C8	rxlengtherror	Number of frames received with length error (Length type field ≠ frame size), for all frames with valid length field.
115	0x01CC	rxoutoffragnotype	Number of frames received with length field not equal to the valid frame size (greater than 1,500 but less than 1,536).
116	0x01D0	rxpauseframes	Number of good and valid PAUSE frames received.
117	0x01D4	rxfifooverflow	Number of missed received frames due to FIFO overflow. This counter is not present in the GMAC-CORE configuration.
118	0x01D8	rxvlanframes_gb	Number of good and bad VLAN frames received.

**Table 3-16 MMC Register Map (Continued)**

<b>GMAC CSR Register No.</b>	<b>Address Offset</b>	<b>Register Name</b>	<b>Register Description</b>
119	0x01DC	rxwatchdogerror	Number of frames received with error due to watchdog timeout error (frames with a data load larger than 2,048 bytes).
120:127	0x01E0–0x01FC	Reserved	
128	0x0200	mmc_ipc_intr_mask_rx	MMC IPC Receive Checksum Offload Interrupt Mask maintains the mask for the interrupt generated from the receive IPC statistic counters. See <a href="#">Table 3-22</a> for further detail.
129	0x0204	Reserved	
130	0x0208	mmc_ipc_intr_rx	MMC Receive Checksum Offload Interrupt maintains the interrupt that the receive IPC statistic counters generate. See <a href="#">Table 3-23</a> for further detail.
131	0x020C	Reserved	
132	0x0210	rxipv4_gd_frms	Number of good IPv4 datagrams received with the TCP, UDP, or ICMP payload
133	0x0214	rxipv4_hdrerr_frms	Number of IPv4 datagrams received with header (checksum, length, or version mismatch) errors
134	0x0218	rxipv4_nopay_frms	Number of IPv4 datagram frames received that did not have a TCP, UDP, or ICMP payload processed by the Checksum engine
135	0x021C	rxipv4_frag_frms	Number of good IPv4 datagrams with fragmentation
136	0x0220	rxipv4_udsbl_frms	Number of good IPv4 datagrams received that had a UDP payload with checksum disabled
137	0x0224	rxipv6_gd_frms	Number of good IPv6 datagrams received with TCP, UDP, or ICMP payloads
138	0x0228	rxipv6_hdrerr_frms	Number of IPv6 datagrams received with header errors (length or version mismatch)
139	0x022C	rxipv6_nopay_frms	Number of IPv6 datagram frames received that did not have a TCP, UDP, or ICMP payload. This includes all IPv6 datagrams with fragmentation or security extension headers
140	0x0230	rxudp_gd_frms	Number of good IP datagrams with a good UDP payload. This counter is not updated when the rxipv4_udsbl_frms counter is incremented.
141	0x0234	rxudp_err_frms	Number of good IP datagrams whose UDP payload has a checksum error
142	0x0238	rxtcp_gd_frms	Number of good IP datagrams with a good TCP payload
143	0x023C	rxtcp_err_frms	Number of good IP datagrams whose TCP payload has a checksum error

**Table 3-16 MMC Register Map (Continued)**

<b>GMAC CSR Register No.</b>	<b>Address Offset</b>	<b>Register Name</b>	<b>Register Description</b>
144	0x0240	rxicmp_gd_frms	Number of good IP datagrams with a good ICMP payload
145	0x0244	rxicmp_err_frms	Number of good IP datagrams whose ICMP payload has a checksum error
146:147	0x0248– 0x024C	Reserved	
148	0x0250	rxipv4_gd_octets	Number of bytes received in good IPv4 datagrams encapsulating TCP, UDP, or ICMP data. (Ethernet header, FCS, pad, or IP pad bytes are not included in this counter or in the octet counters listed below).
149	0x0254	rxipv4_hdrerr_octets	Number of bytes received in IPv4 datagrams with header errors (checksum, length, version mismatch). The value in the Length field of IPv4 header is used to update this counter.
150	0x0258	rxipv4_nopay_octets	Number of bytes received in IPv4 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv4 header's Length field is used to update this counter.
151	0x025C	rxipv4_frag_octets	Number of bytes received in fragmented IPv4 datagrams. The value in the IPv4 header's Length field is used to update this counter.
152	0x0260	rxipv4_udsbl_octets	Number of bytes received in a UDP segment that had the UDP checksum disabled. This counter does not count IP Header bytes.
153	0x0264	rxipv6_gd_octets	Number of bytes received in good IPv6 datagrams encapsulating TCP, UDP or ICMPv6 data
154	0x0268	rxipv6_hdrerr_octets	Number of bytes received in IPv6 datagrams with header errors (length, version mismatch). The value in the IPv6 header's Length field is used to update this counter.
155	0x026C	rxipv6_nopay_octets	Number of bytes received in IPv6 datagrams that did not have a TCP, UDP, or ICMP payload. The value in the IPv6 header's Length field is used to update this counter.
156	0x0270	rxudp_gd_octets	Number of bytes received in a good UDP segment. This counter (and the counters below) does not count IP header bytes.
157	0x0274	rxudp_err_octets	Number of bytes received in a UDP segment that had checksum errors
158	0x0278	rxtcp_gd_octets	Number of bytes received in a good TCP segment
159	0x027C	rxtcp_err_octets	Number of bytes received in a TCP segment with checksum errors
160	0x0280	rxicmp_gd_octets	Number of bytes received in a good ICMP segment

**Table 3-16 MMC Register Map (Continued)**

<b>GMAC CSR Register No.</b>	<b>Address Offset</b>	<b>Register Name</b>	<b>Register Description</b>
161	0x0284	rxicmp_err_octets	Number of bytes received in an ICMP segment with checksum errors
162:191	0x0288– 0x02FC	Reserved	

## 3.8.2 MMC Register Description

### 3.8.2.1 MMC Control Register

The MMC Control register establishes the operating mode of the management counters.

**Table 3-17 MMC Control Register**

<b>Bit</b>	<b>Access Type</b>	<b>Reset Value</b>	<b>Description</b>
31:6	R	0000_000H	Reserved
5	R/W	0	<p>Full-Half preset When low and bit4 is set, all MMC counters get preset to almost-half value. All octet counters get preset to 0xFFFF_F800 (half - 2KBytes) and all frame-counters gets preset to 0xFFFF_FFF0 (half - 16)</p> <p>When high and bit4 is set, all MMC counters get preset to almost-full value. All octet counters get preset to 0xFFFF_F800 (full - 2KBytes) and all frame-counters gets preset to 0xFFFF_FFF0 (full - 16)</p>
4	R_W_SC	0	<p>Counters Preset When set, all counters will be initialized or preset to almost full or almost half as per Bit5 above. This bit will be cleared automatically after 1 clock cycle. This bit along with bit5 is useful for debugging and testing the assertion of interrupts due to MMC counter becoming half-full or full.</p>
3	R/W	0	<p>MMC Counter Freeze When set, this bit freezes all the MMC counters to their current value. (None of the MMC counters are updated due to any transmitted or received frame until this bit is reset to 0. If any MMC counter is read with the Reset on Read bit set, then that counter is also cleared in this mode.)</p>
2	R_W	0	<p>Reset on Read When set, the MMC counters will be reset to zero after Read (self-clearing after reset). The counters are cleared when the least significant byte lane (bits[7:0]) is read.</p>
1	R_W	0	<p>Counter Stop Rollover When set, counter after reaching maximum value will not roll over to zero.</p>
0	R_W_SC	0	<p>Counters Reset When set, all counters will be reset. This bit will be cleared automatically after 1 clock cycle</p>



When Bit 0 and Bit 4 are set in the same cycle, Counters get preset immediately after getting cleared.

### 3.8.2.2 MMC Receive Interrupt Register

The MMC Receive Interrupt register maintains the interrupts generated when the receive statistic counters reach half their maximum values (0x8000\_0000), and when they cross their maximum values (0xFFFF\_FFFF). When Counter Stop Rollover is set, then interrupts are set but the counter remains at all-ones. The MMC Receive Interrupt register is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (bits[7:0]) of the respective counter must be read in order to clear the interrupt bit.

**Table 3-18 MMC Receive Interrupt Register**

Bit	Access Type	Reset Value	Description
31:24	RO	00H	Reserved
23	R_SS_RC	0	The bit is set when the rxwatchdog error counter reaches half the maximum value, and also when it reaches the maximum value.
22	R_SS_RC	0	The bit is set when the rxvlanframes_gb counter reaches half the maximum value, and also when it reaches the maximum value.
21	R_SS_RC	0	The bit is set when the rxfifooverflow counter reaches half the maximum value, and also when it reaches the maximum value.
20	R_SS_RC	0	The bit is set when the rxpauseframes counter reaches half the maximum value, and also when it reaches the maximum value.
19	R_SS_RC	0	The bit is set when the rxoutofrange type counter reaches half the maximum value, and also when it reaches the maximum value.
18	R_SS_RC	0	The bit is set when the rxlengtherror counter reaches half the maximum value, and also when it reaches the maximum value.
17	R_SS_RC	0	The bit is set when the rxunicastframes_g counter reaches half the maximum value, and also when it reaches the maximum value.
16	R_SS_RC	0	The bit is set when the rx1024tomaxoctets_gb counter reaches half the maximum value, and also when it reaches the maximum value.
15	R_SS_RC	0	The bit is set when the rx512to1023octets_gb counter reaches half the maximum value, and also when it reaches the maximum value.
14	R_SS_RC	0	The bit is set when the rx256to511octets_gb counter reaches half the maximum value, and also when it reaches the maximum value.
13	R_SS_RC	0	The bit is set when the rx128to255octets_gb counter reaches half the maximum value, and also when it reaches the maximum value.
12	R_SS_RC	0	The bit is set when the rx65to127octets_gb counter reaches half the maximum value, and also when it reaches the maximum value.

**Table 3-18 MMC Receive Interrupt Register (Continued)**

Bit	Access Type	Reset Value	Description
11	R_SS_RC	0	The bit is set when the rx64octets_gb counter reaches half the maximum value, and also when it reaches the maximum value.
10	R_SS_RC	0	The bit is set when the rxoversize_g counter reaches half the maximum value, and also when it reaches the maximum value.
9	R_SS_RC	0	The bit is set when the rxundersize_g counter reaches half the maximum value, and also when it reaches the maximum value.
8	R_SS_RC	0	The bit is set when the rxjabbererror counter reaches half the maximum value, and also when it reaches the maximum value.
7	R_SS_RC	0	The bit is set when the rxrunterror counter reaches half the maximum value, and also when it reaches the maximum value.
6	R_SS_RC	0	The bit is set when the rxalignmenterror counter reaches half the maximum value, and also when it reaches the maximum value.
5	R_SS_RC	0	The bit is set when the rxcrcerror counter reaches half the maximum value, and also when it reaches the maximum value.
4	R_SS_RC	0	The bit is set when the rxmulticastframes_g counter reaches half the maximum value, and also when it reaches the maximum value.
3	R_SS_RC	0	The bit is set when the rxbroadcastframes_g counter reaches half the maximum value, and also when it reaches the maximum value.
2	R_SS_RC	0	The bit is set when the rxoctetcount_g counter reaches half the maximum value, and also when it reaches the maximum value.
1	R_SS_RC	0	The bit is set when the rxoctetcount_gb counter reaches half the maximum value, and also when it reaches the maximum value.
0	R_SS_RC	0	The bit is set when the rxframecount_gb counter reaches half the maximum value, and also when it reaches the maximum value.



**Note** R\_SS\_RC means that this register bit is set internally and is cleared when the Counter register is read.

### 3.8.2.3 MMC Transmit Interrupt Register

The MMC Transmit Interrupt register maintains the interrupts generated when transmit statistic counters reach half their maximum values (0x8000\_0000), and when they cross their maximum values (0xFFFF\_FFFF). When Counter Stop Rollover is set, then interrupts are set but the counter remains at all-ones. The MMC Transmit Interrupt register is a 32-bit wide register. An interrupt bit is cleared when the respective MMC counter that caused the interrupt is read. The least significant byte lane (bits[7:0]) of the respective counter must be read in order to clear the interrupt bit.

**Table 3-19 MMC Transmit Interrupt Register**

Bit	Access Type	Reset Value	Description
31:25	RO	00H	Reserved
24	R_SS_RC	0	The bit is set when the txvlanframes_g counter reaches half the maximum value, and also when it reaches the maximum value.
23	R_SS_RC	0	The bit is set when the txpauseframes error counter reaches half the maximum value, and also when it reaches the maximum value.
22	R_SS_RC	0	The bit is set when the txoexcessdef counter reaches half the maximum value, and also when it reaches the maximum value.
21	R_SS_RC	0	The bit is set when the txframecount_g counter reaches half the maximum value, and also when it reaches the maximum value.
20	R_SS_RC	0	The bit is set when the txoctetcount_g counter reaches half the maximum value, and also when it reaches the maximum value.
19	R_SS_RC	0	The bit is set when the txcarriererror counter reaches half the maximum value, and also when it reaches the maximum value.
18	R_SS_RC	0	The bit is set when the txexcesscol counter reaches half the maximum value, and also when it reaches the maximum value.
17	R_SS_RC	0	The bit is set when the txlatecol counter reaches half the maximum value, and also when it reaches the maximum value.
16	R_SS_RC	0	The bit is set when the txdeferred counter reaches half the maximum value, and also when it reaches the maximum value.
15	R_SS_RC	0	The bit is set when the txmulticol_g counter reaches half the maximum value, and also when it reaches the maximum value.
14	R_SS_RC	0	The bit is set when the txsinglecol_g counter reaches half the maximum value, and also when it reaches the maximum value.
13	R_SS_RC	0	The bit is set when the txunderflowerror counter reaches half the maximum value, and also when it reaches the maximum value.
12	R_SS_RC	0	The bit is set when the txbroadcastframes_gb counter reaches half the maximum value, and also when it reaches the maximum value.
11	R_SS_RC	0	The bit is set when the txmulticastframes_gb counter reaches half the maximum value, and also when it reaches the maximum value.
10	R_SS_RC	0	The bit is set when the txunicastframes_gb counter reaches half the maximum value, and also when it reaches the maximum value.
9	R_SS_RC	0	The bit is set when the tx1024tomaxoctets_gb counter reaches half the maximum value, and also when it reaches the maximum value.
8	R_SS_RC	0	The bit is set when the tx512to1023octets_gb counter reaches half the maximum value, and also when it reaches the maximum value.

**Table 3-19 MMC Transmit Interrupt Register (Continued)**

Bit	Access Type	Reset Value	Description
7	R_SS_RC	0	The bit is set when the tx256to511octets_gb counter reaches half the maximum value, and also when it reaches the maximum value.
6	R_SS_RC	0	The bit is set when the tx128to255octets_gb counter reaches half the maximum value, and also when it reaches the maximum value.
5	R_SS_RC	0	The bit is set when the tx65to127octets_gb counter reaches half the maximum value, and also when it reaches the maximum value.
4	R_SS_RC	0	The bit is set when the tx64to127octets_gb counter reaches half the maximum value, and also when it reaches the maximum value.
3	R_SS_RC	0	The bit is set when the txmulticastframes_g counter reaches half the maximum value, and also when it reaches the maximum value.
2	R_SS_RC	0	The bit is set when the txbroadcastframes_g counter reaches half the maximum value, and also when it reaches the maximum value.
1	R_SS_RC	0	The bit is set when the txframecount_gb counter reaches half the maximum value, and also when it reaches the maximum value.
0	R_SS_RC	0	The bit is set when the txoctetcount_gb counter reaches half the maximum value, and also when it reaches the maximum value.

### 3.8.2.4 MMC Receive Interrupt Mask Register

The MMC Receive Interrupt Mask register maintains the masks for the interrupts generated when receive statistic counters reach half their maximum value, and when they reach their maximum values. This register is 32 bits wide.

**Table 3-20 MMC Receive Interrupt Mask Register**

Bit	Access Type	Reset Value	Description
31:24	RO	00H	Reserved
23	R_W	0	Setting this bit masks the interrupt when the rxwatchdog counter reaches half the maximum value, and also when it reaches the maximum value.
22	R_W	0	Setting this bit masks the interrupt when the rxvlanframes_gb counter reaches half the maximum value, and also when it reaches the maximum value.
21	R_W	0	Setting this bit masks the interrupt when the rxfifooverflow counter reaches half the maximum value, and also when it reaches the maximum value.
20	R_W	0	Setting this bit masks the interrupt when the rxpauseframes counter reaches half the maximum value, and also when it reaches the maximum value.
....	R_W	...	... Same as above for corresponding counters in <a href="#">MMC Receive Interrupt Register</a>

**Table 3-20 MMC Receive Interrupt Mask Interrupt Register (Continued)**

Bit	Access Type	Reset Value	Description
1	R_W	0	Setting this bit masks the interrupt when the rxoctetcount_gb counter reaches half the maximum value, and also when it reaches the maximum value.
0	R_W	0	Setting this bit masks the interrupt when the rxframecount_gb counter reaches half the maximum value, and also when it reaches the maximum value.

### 3.8.2.5 MMC Transmit Interrupt Mask Register

The MMC Transmit Interrupt Mask register maintains the masks for the interrupts generated when transmit statistic counters reach half their maximum value, and when they reach their maximum values. This register is 32 bits wide.

**Table 3-21 MMC Transmit Interrupt Mask Register**

Bit	Access Type	Reset Value	Description
31:25	RO	00H	Reserved
24	R_W	0	Setting this bit masks the interrupt when the txvlanframes_g counter reaches half the maximum value, and also when it reaches the maximum value.
23	R_W	0	Setting this bit masks the interrupt when the txpauseframes counter reaches half the maximum value, and also when it reaches the maximum value.
22	R_W	0	Setting this bit masks the interrupt when the txoexcessdef counter reaches half the maximum value, and also when it reaches the maximum value.
21	R_W	0	Setting this bit masks the interrupt when the txframecount_g counter reaches half the maximum value, and also when it reaches the maximum value.
....	R_W	...	... Same as above for corresponding counters in <a href="#">MMC Transmit Interrupt Register</a>
1	R_W	0	Setting this bit masks the interrupt when the txframecount_gb counter reaches half the maximum value, and also when it reaches the maximum value.
0	R_W	0	Setting this bit masks the interrupt when the rxoctetcount_gb counter reaches half the maximum value, and also when it reaches the maximum value.

### 3.8.2.6 MMC Receive Checksum Offload Interrupt Mask Register

The MMC Receive Checksum Offload Interrupt Mask register maintains the masks for the interrupts generated when the receive IPC (Checksum Offload) statistic counters reach half their maximum value , and when they reach their maximum values. This register is 32 bits wide.

**Table 3-22 MMC Receive Checksum Offload Interrupt Mask Register**

Bit	Access Type	Reset Value	Description
31:30	RO	00	Reserved

**Table 3-22 MMC Receive Checksum Offload Interrupt Mask Register (Continued)**

Bit	Access Type	Reset Value	Description
29	R_W	0	Setting this bit masks the interrupt when the rxicmp_err_octets counter reaches half the maximum value, and also when it reaches the maximum value.
28	R_W	0	Setting this bit masks the interrupt when the rxicmp_gd_octets counter reaches half the maximum value, and also when it reaches the maximum value.
27	R_W	0	Setting this bit masks the interrupt when the rxtcp_err_octets counter reaches half the maximum value, and also when it reaches the maximum value.
26	R_W	0	Setting this bit masks the interrupt when the rxtcp_gd_octets counter reaches half the maximum value, and also when it reaches the maximum value.
25	R_W	0	Setting this bit masks the interrupt when the rxudp_err_octets counter reaches half the maximum value, and also when it reaches the maximum value.
24	R_W	0	Setting this bit masks the interrupt when the rxudp_gd_octets counter reaches half the maximum value, and also when it reaches the maximum value.
23	R_W	0	Setting this bit masks the interrupt when the rxipv6_nopay_octets counter reaches half the maximum value, and also when it reaches the maximum value.
22	R_W	0	Setting this bit masks the interrupt when the rxipv6_hdrerr_octets counter reaches half the maximum value, and also when it reaches the maximum value.
21	R_W	0	Setting this bit masks the interrupt when the rxipv6_gd_octets counter reaches half the maximum value, and also when it reaches the maximum value.
20	R_W	0	Setting this bit masks the interrupt when the rxipv4_udsl_octets counter reaches half the maximum value, and also when it reaches the maximum value.
19	R_W	0	Setting this bit masks the interrupt when the rxipv4_frag_octets counter reaches half the maximum value, and also when it reaches the maximum value.
18	R_W	0	Setting this bit masks the interrupt when the rxipv4_nopay_octets counter reaches half the maximum value, and also when it reaches the maximum value.
17	R_W	0	Setting this bit masks the interrupt when the rxipv4_hdrerr_octets counter reaches half the maximum value, and also when it reaches the maximum value.
16	R_W	0	Setting this bit masks the interrupt when the rxipv4_gd_octets counter reaches half the maximum value, and also when it reaches the maximum value.
15:14	RO	0	Reserved
13	R_W	0	Setting this bit masks the interrupt when the rxicmp_err_frms counter reaches half the maximum value, and also when it reaches the maximum value.
12	R_W	0	Setting this bit masks the interrupt when the rxicmp_gd_frms counter reaches half the maximum value, and also when it reaches the maximum value.
11	R_W	0	Setting this bit masks the interrupt when the rxtcp_err_frms counter reaches half the maximum value, and also when it reaches the maximum value.

**Table 3-22 MMC Receive Checksum Offload Interrupt Mask Register (Continued)**

Bit	Access Type	Reset Value	Description
10	R_W	0	Setting this bit masks the interrupt when the rxtcp_gd_frms counter reaches half the maximum value, and also when it reaches the maximum value.
9	R_W	0	Setting this bit masks the interrupt when the rxudp_err_frms counter reaches half the maximum value, and also when it reaches the maximum value.
8	R_W	0	Setting this bit masks the interrupt when the rxudp_gd_frms counter reaches half the maximum value, and also when it reaches the maximum value.
7	R_W	0	Setting this bit masks the interrupt when the rxipv6_nopay_frms counter reaches half the maximum value, and also when it reaches the maximum value.
6	R_W	0	Setting this bit masks the interrupt when the rxipv6_hdrerr_frms counter reaches half the maximum value, and also when it reaches the maximum value.
5	R_W	0	Setting this bit masks the interrupt when the rxipv6_gd_frms counter reaches half the maximum value, and also when it reaches the maximum value.
4	R_W	0	Setting this bit masks the interrupt when the rxipv4_udslb_frms counter reaches half the maximum value, and also when it reaches the maximum value.
3	R_W	0	Setting this bit masks the interrupt when the rxipv4_frag_frms counter reaches half the maximum value, and also when it reaches the maximum value.
2	R_W	0	Setting this bit masks the interrupt when the rxipv4_nopay_frms counter reaches half the maximum value, and also when it reaches the maximum value.
1	R_W	0	Setting this bit masks the interrupt when the rxipv4_hdrerr_frms counter reaches half the maximum value, and also when it reaches the maximum value.
0	R_W	0	Setting this bit masks the interrupt when the rxipv4_gd_frms counter reaches half the maximum value, and also when it reaches the maximum value.

### 3.8.2.7 MMC Receive Checksum Offload Interrupt Register

The MMC Receive Checksum Offload Interrupt register maintains the interrupts generated when receive IPC statistic counters reach half their maximum values (0x8000\_0000), and when they cross their maximum values (0xFFFF\_FFFF). When Counter Stop Rollover is set, then interrupts are set but the counter remains at all-ones. The MMC Receive Checksum Offload Interrupt register is 32 bits wide. When the MMC IPC counter that caused the interrupt is read, its corresponding interrupt bit is cleared. The counter's least-significant byte lane (bits[7:0]) must be read to clear the interrupt bit.

**Table 3-23 MMC Receive Checksum Offload Interrupt Register**

Bit	Access Type	Reset Value	Description
31:30	RO	00	Reserved
29	R_SS_RC	0	The bit is set when the rxicmp_err_octets counter reaches half the maximum value, and also when it reaches the maximum value.

**Table 3-23 MMC Receive Checksum Offload Interrupt Register (Continued)**

Bit	Access Type	Reset Value	Description
28	R_SS_RC	0	The bit is set when the rxicmp_gd_octets counter reaches half the maximum value, and also when it reaches the maximum value.
27	R_SS_RC	0	The bit is set when the rxtcp_err_octets counter reaches half the maximum value, and also when it reaches the maximum value.
26	R_SS_RC	0	The bit is set when the rxtcp_gd_octets counter reaches half the maximum value, and also when it reaches the maximum value.
25	R_SS_RC	0	The bit is set when the rxudp_err_octets counter reaches half the maximum value, and also when it reaches the maximum value.
24	R_SS_RC	0	The bit is set when the rxudp_gd_octets counter reaches half the maximum value, and also when it reaches the maximum value.
23	R_SS_RC	0	The bit is set when the rxipv6_nopay_octets counter reaches half the maximum value, and also when it reaches the maximum value.
22	R_SS_RC	0	The bit is set when the rxipv6_hdrerr_octets counter reaches half the maximum value, and also when it reaches the maximum value.
21	R_SS_RC	0	The bit is set when the rxipv6_gd_octets counter reaches half the maximum value, and also when it reaches the maximum value.
20	R_SS_RC	0	The bit is set when the rxipv4_udslbl_octets counter reaches half the maximum value, and also when it reaches the maximum value.
19	R_SS_RC	0	The bit is set when the rxipv4_frag_octets counter reaches half the maximum value, and also when it reaches the maximum value.
18	R_SS_RC	0	The bit is set when the rxipv4_nopay_octets counter reaches half the maximum value, and also when it reaches the maximum value.
17	R_SS_RC	0	The bit is set when the rxipv4_hdrerr_octets counter reaches half the maximum value, and also when it reaches the maximum value.
16	R_SS_RC	0	The bit is set when the rxipv4_gd_octets counter reaches half the maximum value, and also when it reaches the maximum value.
15:14	RO	00	Reserved
13	R_SS_RC	0	The bit is set when the rxicmp_err_frms counter reaches half the maximum value, and also when it reaches the maximum value.
12	R_SS_RC	0	The bit is set when the rxicmp_gd_frms counter reaches half the maximum value, and also when it reaches the maximum value.
11	R_SS_RC	0	The bit is set when the rxtcp_err_frms counter reaches half the maximum value, and also when it reaches the maximum value.
10	R_SS_RC	0	The bit is set when the rxtcp_gd_frms counter reaches half the maximum value, and also when it reaches the maximum value.

**Table 3-23 MMC Receive Checksum Offload Interrupt Register (Continued)**

Bit	Access Type	Reset Value	Description
9	R_SS_RC	0	The bit is set when the rxudp_err_frms counter reaches half the maximum value, and also when it reaches the maximum value.
8	R_SS_RC	0	The bit is set when the rxudp_gd_frms counter reaches half the maximum value, and also when it reaches the maximum value.
7	R_SS_RC	0	The bit is set when the rxipv6_nopay_frms counter reaches half the maximum value, and also when it reaches the maximum value.
6	R_SS_RC	0	The bit is set when the rxipv6_hdrerr_frms counter reaches half the maximum value, and also when it reaches the maximum value.
5	R_SS_RC	0	The bit is set when the rxipv6_gd_frms counter reaches half the maximum value, and also when it reaches the maximum value.
4	R_SS_RC	0	The bit is set when the rxipv4_udsl_frms counter reaches half the maximum value, and also when it reaches the maximum value.
3	R_SS_RC	0	The bit is set when the rxipv4_frag_frms counter reaches half the maximum value, and also when it reaches the maximum value.
2	R_SS_RC	0	The bit is set when the rxipv4_nopay_frms counter reaches half the maximum value, and also when it reaches the maximum value.
1	R_SS_RC	0	The bit is set when the rxipv4_hdrerr_frms counter reaches half the maximum value, and also when it reaches the maximum value.
0	R_SS_RC	0	The bit is set when the rxipv4_gd_frms counter reaches half the maximum value, and also when it reaches the maximum value.

## 3.9 Power Management Block

This section describes the power management (PMT) mechanisms supported by the GMAC. PMT supports the reception of network (remote) wake-up frames and Magic Packet frames. PMT does not perform the clock gate function, but generates interrupts for wake-up frames and Magic Packets received by the GMAC. The PMT block sits on the receiver path of the GMAC and is enabled with remote wake-up frame enable and Magic Packet enable. These enables are in the PMT Control and Status register and are programmed by the Application. You can include the optional PMT module by selecting it in the coreKit during configuration. You have the option to select either or both types of power-management frames (remote wake-up and Magic Packet).

PMT registers are accessed in the same manner as with GMAC-CSR registers. Refer to [Table 5-2](#), registers 10 and 11, for mapping information.

When the power down mode is enabled in the PMT, then all received frames are dropped by the core and they are not forwarded to the application. The core comes out of the power down mode only when either a Magic Packet or a Remote Wake-up frame is received and the corresponding detection is enabled.

### 3.9.1 PMT Block Description

#### 3.9.1.1 PMT Control and Status Register

The PMT CSR program the request wake-up events and monitor the wake-up events.

**Table 3-24 PMT Control and Status Register**

Bit	Access Type	Reset Value	Description
31	R_WS_SC	0	Wake-Up Frame Filter Register Pointer Reset When set, resets the Remote Wake-up Frame Filter register pointer to 3'b000. It is automatically cleared after 1 clock cycle.
30:10	R	0000_00H	Reserved
9	R_W	00	Global Unicast When set, enables any unicast packet filtered by the GMAC (DAF) address recognition to be a wake-up frame.
8:7	R	0	Reserved
6	R_SS_RC	0	Wake-Up Frame Received When set, this bit indicates the power management event was generated due to reception of a wake-up frame. This bit is cleared by a Read into this register.
5	R_SS_RC	0	Magic Packet Received When set, this bit indicates the power management event was generated by the reception of a Magic Packet. This bit is cleared by a Read into this register.
4:3	R	00	Reserved
2	R_W	0	Wake-Up Frame Enable When set, enables generation of a power management event due to wake-up frame reception.
1	R_W	0	Magic Packet Enable When set, enables generation of a power management event due to Magic Packet reception.
0	R_WS_SC	0	Power Down When set, all received frames will be dropped. This bit is cleared automatically when a magic packet or Wake-Up frame is received, and Power-Down mode is disabled. Frames received after this bit is cleared are forwarded to the application. This bit must only be set when either the Magic Packet Enable or Wake-Up Frame Enable bit is set high.

### 3.9.1.2 Remote Wake-Up Frame Filter Register

The register `wkupfmfilter_reg`, address (028H), loads the Wake-up Frame Filter register. To load values in a Wake-up Frame Filter register, the entire register (`wkupfmfilter_reg`) must be written. The `wkupfmfilter_reg` register is loaded by sequentially loading the eight register values in address (028) for `wkupfmfilter_reg0`, `wkupfmfilter_reg1`, ... `wkupfmfilter_reg7`, respectively. `wkupfmfilter_reg` is read in the same way.



**Note** The internal counter to access the appropriate `wkupfmfilter_reg` is incremented when lane3 (or lane 0 in big-endian) is accessed by the CPU. This should be kept in mind if you are accessing these registers in byte or half-word mode.

**Figure 3-67 Wake-Up Frame Filter Register**

<code>wkupfmfilter_reg0</code>	Filter 0 Byte Mask											
<code>wkupfmfilter_reg1</code>	Filter 1 Byte Mask											
<code>wkupfmfilter_reg2</code>	Filter 2 Byte Mask											
<code>wkupfmfilter_reg3</code>	Filter 3 Byte Mask											
<code>wkupfmfilter_reg4</code>	RSVD	Filter 3 Command	RSVD	Filter 2 Command	RSVD	Filter 1 Command	RSVD	Filter 0 Command				
<code>wkupfmfilter_reg5</code>	Filter 3 Offset		Filter 2 Offset		Filter 1 Offset		Filter 0 Offset					
<code>wkupfmfilter_reg6</code>	Filter 1 CRC - 16				Filter 0 CRC - 16							
<code>wkupfmfilter_reg7</code>	Filter 3 CRC - 16				Filter 2 CRC - 16							

#### Filter *i* Byte Mask

This register defines which bytes of the frame are examined by filter *i* (0, 1, 2, and 3) in order to determine whether or not the frame is a wake-up frame. The MSB (thirty-first bit) must be zero. Bit *j* [30:0] is the Byte Mask. If bit *j* (byte number) of the Byte Mask is set, then Filter *i* Offset + *j* of the incoming frame is processed by the CRC block; otherwise Filter *i* Offset + *j* is ignored.

#### Filter *i* Command

This 4-bit command controls the filter *i* operation. Bit 3 specifies the address type, defining the pattern's destination address type. When the bit is set, the pattern applies to only multicast frames; when the bit is reset, the pattern applies only to unicast frame. Bit 2 and Bit 1 are reserved. Bit 0 is the enable for filter *i*; if Bit 0 is not set, filter *i* is disabled.

#### Filter *i* Offset

This register defines the offset (within the frame) from which the frames are examined by filter *i*. This 8-bit pattern-offset is the offset for the filter *i* first byte to examined. The minimum allowed is 12, which refers to the 13th byte of the frame (offset value 0 refers to the first byte of the frame).

## Filter *i* CRC-16

This register contains the CRC\_16 value calculated from the pattern, as well as the byte mask programmed to the wake-up filter register block.

### 3.9.2 Remote Wake-Up Frame Detection

When the GMAC is in sleep mode and the remote wake-up bit is enabled in PMT Control and Status register (002C), normal operation is resumed after receiving a remote wake-up frame. The Application writes all eight wake-up filter registers, by performing a sequential Write to address (0028). The Application enables remote wake-up by writing a 1 to Bit 2 of the PMT Control and Status register.

PMT supports four programmable filters that allow support of different receive frame patterns. If the incoming frame passes the address filtering of Filter Command, and if Filter CRC-16 matches the incoming examined pattern, then the wake-up frame is received.

Filter\_offset (minimum value 12, which refers to the 13th byte of the frame) determines the offset from which the frame is to be examined. Filter Byte Mask determines which bytes of the frame must be examined. The thirty-first bit of Byte Mask must be set to zero.

The remote wake-up CRC block determines the CRC value that is compared with Filter CRC-16. The wake-up frame is checked only for length error, FCS error, dribble bit error, GMII error, collision, and to ensure that it is not a runt frame. Even if the wake-up frame is more than 512 bytes long, if the frame has a valid CRC value, it is considered valid. Wake-up frame detection is updated in the PMT Control and Status register for every remote Wake-up frame received. A PMT interrupt to the Application triggers a Read to the PMT Control and Status register to determine reception of a wake-up frame.



**Note** A sample C routine that generates CRC-16 for a sequence of data has been provided in the resources/ directory of your workspace.

### 3.9.3 Magic Packet Detection

The Magic Packet frame is based on a method that uses Advanced Micro Device's Magic Packet technology to power up the sleeping device on the network. The GMAC receives a specific packet of information, called a Magic Packet, addressed to the node on the network.

Only Magic Packets that are addressed to the device or a broadcast address will be checked to determine whether they meet the wake-up requirements. Magic Packets that pass the address filtering (unicast or broadcast) will be checked to determine whether they meet the remote Wake-on-LAN data format of 6 bytes of all ones followed by a GMAC Address appearing 16 times.

The application enables Magic Packet wake-up by writing a 1 to Bit 1 of the PMT Control and Status register. The PMT block constantly monitors each frame addressed to the node for a specific Magic Packet pattern. Each frame received is checked for a 48'hFF\_FF\_FF\_FF\_FF\_FF pattern following the destination and source address field. The PMT block then checks the frame for 16 repetitions of the GMAC address without any breaks or interruptions. In case of a break in the 16 repetitions of the address, the 48'hFF\_FF\_FF\_FF\_FF\_FF pattern is scanned for again in the incoming frame. The 16 repetitions can be anywhere in the frame, but must be preceded by the synchronization stream (48'hFF\_FF\_FF\_FF\_FF\_FF). The device will also accept a multicast frame, as long as the 16 duplications of the GMAC address are detected.

If the MAC address of a node is 48'h00\_11\_22\_33\_44\_55, then the GMAC scans for the data sequence:

Destination Address Source Address ..... FF FF FF FF FF FF

```
00 11 22 33 44 55 00 11 22 33 44 55 00 11 22 33 44 55  
00 11 22 33 44 55 00 11 22 33 44 55 00 11 22 33 44 55  
00 11 22 33 44 55 00 11 22 33 44 55 00 11 22 33 44 55  
00 11 22 33 44 55 00 11 22 33 44 55 00 11 22 33 44 55  
...CRC
```

Magic Packet detection is updated in the PMT Control and Status register for Magic Packet received. A PMT interrupt to the Application triggers a read to the PMT CSR to determine whether a Magic Packet frame has been received.

### 3.9.4 System Considerations During Power-Down

GMAC-UNIV neither gates nor stops clocks when Power-Down mode is enabled. Power saving by clock gating must be done outside the core by the application. The receive data path must be clocked with clk\_rx\_i during Power-Down mode, because it is involved in magic packet/wake-on-LAN frame detection. However, the transmit path and the application path clocks can be gated off during Power-Down mode.

The pmt\_intr\_o signal is asserted when a valid wake-up frame is received. This signal is generated in the clk\_rx\_i domain.

The recommended power-down and wake-up sequence is as follows.

1. Disable the Transmit DMA (if applicable) and wait for any previous frame transmissions to complete. These transmissions can be detected when Transmit Interrupt (TI-DMA Register 5[0]) is received.
2. Disable the MAC transmitter and MAC receiver by clearing the appropriate bits in the MAC Configuration register.
3. Wait until the Receive DMA empties all the frames from the Rx FIFO (a software timer may be required).
4. Enable Power-Down mode by appropriately configuring the PMT registers.
5. Enable the MAC Receiver and enter Power-Down mode.
6. Gate the application and transmit clock inputs to the core (and other relevant clocks in the system) to reduce power and enter Sleep mode.
7. On receiving a valid wake-up frame, the GMAC-UNIV asserts the pmt\_intr\_o signal and exits Power-Down mode.
8. On receiving the interrupt, the system must enable the application and transmit clock inputs to the core.
9. Read the PMT Status register to clear the interrupt, then enable the other modules in the system and resume normal operation.



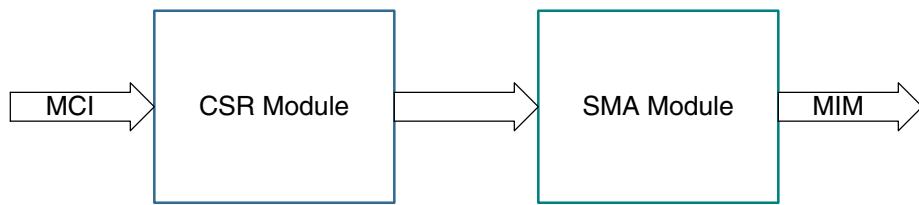
**Note** As the pmt\_intr\_o is generated in clk\_rx\_i domain, its clearing on a read to PMT Status register is not immediate. The resultant clear signal has to cross to the clk\_rx\_i domain and then clear the interrupt source. This delay will be at least 4 clock cycles of clk\_rx and can be significant when the core is operating in 10 Mbps mode.

## 3.10 Station Management Agent

The Station Management Agent (SMA) module allows the Application to access any PHY registers through a 2-wire Station Management interface (MIM). The interface supports accessing up to 32 PHYs.

The application can select one of the 32 PHYs and one of the 32 registers within any PHY and send control data or receive status information. Only one register in one PHY can be addressed at any given time. For more details on the communication from the Application to the PHYs, refer to the Reconciliation Sublayer and Media Independent Interface Specifications section of the IEEE 802.3z specification, 1000BASE Ethernet. The application sends the control data to the PHY and receives status information from the PHY through the SMA module, as shown in [Figure 3-68](#).

**Figure 3-68 SMA Interface Block**



### 3.10.1 Functions

The GMAC initiates the Management Write/Read operation. The clock gmii\_mdc\_o is a divided clock from the Application clock clk\_csr\_i. The divide factor depends on the clock range setting in the GMII Address register. Clock range is set as follows:

Selection	clk_csr_i	MDC Clock
0000	60-100 MHz	clk_csr_i/42
0001	100-150 MHz	clk_csr_i/62
0010	20-35 MHz	clk_csr_i/16
0011	35-60 MHz	clk_csr_i/26
0100	150-250 MHz	clk_csr_i/102
0101	250-300 MHz	clk_csr_i/124
0110, 0111	Reserved	

The gmii\_mdc\_o is the derivative of the Application clock clk\_csr\_i. The Management operation is performed through the gmii\_mdi\_i, gmii\_mdo\_o and gmii\_mdo\_o\_e signals. A three-state buffer is implemented outside the GMAC to interface with an external PHY. A detailed description is provided in ["SMA to PHY Interface" on page 381](#).

The frame structure on the MDIO line is shown below.

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
------	----------	-------	--------	----------	----------	----	------	------

IDLE	The mdio line is three-state; there is no clock on gmii_mdc_o
PREAMBLE	32 continuous bits of value 1
START	Start-of-frame is 2'01
OPCODE	2'b10 for Read and 2'b01 for Write
PHY ADDR	5-bit address select for one of 32 PHYs
REG ADDR	Register address in the selected PHY
TA	Turnaround is 2'bZ0 for Read and 2'b10 for Write
DATA	Any 16-bit value. In a Write operation, the GMAC drives mdio; in a Read operation, PHY drives it.

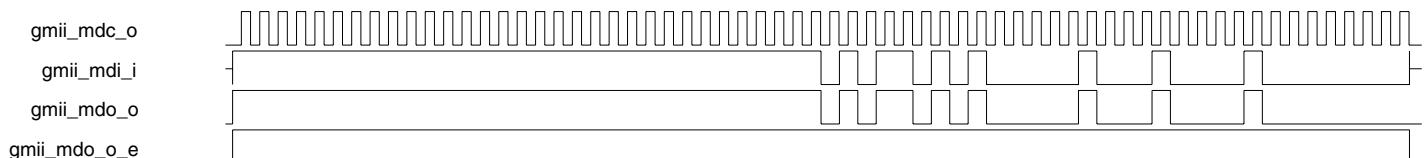
### 3.10.2 GMII/MII Management Write Operation

When the user sets the GMII Write and Busy bits (see GMII Address Register, “[Register 4 \(GMII Address Register\)](#)” on page 285), the GMAC CSR module transfers the PHY address, the register address in PHY, and the write data (GMII Data Register) to the SMA to initiate a Write operation into the PHY registers. At this point, the SMA module starts a Write operation on the GMII Management Interface using the Management Frame Format specified in the GMII specifications (Section 22.2.4.5 of IEEE Standard). The application should not change the GMII Address register contents or the GMII Data register while the transaction is ongoing. Write operations to the GMII Address register or the GMII Data Register during this period are ignored (the Busy bit is high), and the transaction is completed without any error on the MCI interface.

After the Write operation has completed, the SMA indicates this to the CSR which then resets the Busy bit. The SMA module divides the CSR (Application) clock with the clock divider programmed (CR bits of GMII Address Register) to generate the MDC clock for this interface. The GMAC drives the MDIO line for the complete duration of the frame. The frame format for the Write operation is as follows:

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
Z	1111...11	01	01	AAAAA	RRRRR	10	DDD . . . DDD	Z

**Figure 3-69 Management Write Operation**



**Figure 3-69** is a reference for the Write operation.

### 3.10.3 GMII/MII Management Read Operation

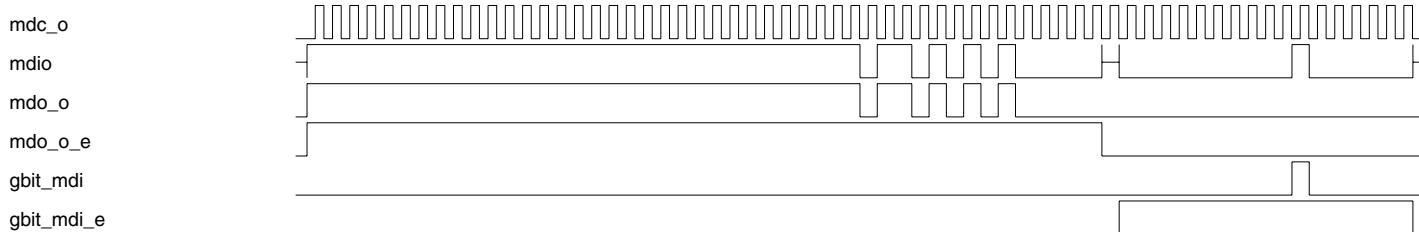
When the user sets the GMII Busy bit (see GMII Address Register, “[Register 4 \(GMII Address Register\)](#)” on page 285) with the GMII Write bit as 0, the GMAC CSR module transfers the PHY address and the register address in PHY to the SMA to initiate a Read operation in the PHY registers. At this point, the SMA module starts a Read operation on the GMII Management Interface using the Management Frame Format specified in the GMII specifications (Section 22.2.4.5 of IEEE Standard). The application should not change the GMII Address register contents or the GMII Data register while the transaction is ongoing. Write operations to the GMII Address register or GMII Data Register during this period are ignored (the Busy bit is high) and the transaction completed without any error on the MCI interface.

After the Read operation has completed, the SMA indicates this to the CSR, which then resets the Busy bit and updates the GMII Data register with the data read from the PHY. The SMA module divides the CSR (Application) clock with the clock divider programmed (CR bits of GMII Address Register) to generate the MDC clock for this interface. The GMAC drives the MDIO line for the complete duration of the frame except during the Data fields when the PHY is driving the MDIO line. The frame format for the Read operation is as follows:

IDLE	PREAMBLE	START	OPCODE	PHY ADDR	REG ADDR	TA	DATA	IDLE
Z	1111...11	01	10	AAAAA	RRRRR	Z0	DDD . . . . DDD	Z

[Figure 3-70](#) is a reference for the read operation.

**Figure 3-70 Management Read Operation**



## 3.11 Physical Coding Sublayer (PCS)

The GMAC provides an IEEE 802.3z compliant 10-bit Physical Coding Sublayer (PCS) interface used when the GMAC-UNIV is configured for TBI/RTBI/SGMII PHY interface. This interface will transmit 10-bit code groups with a 125-MHz transmit clock. In addition, the interface will receive 10-bit code groups with two out-of-phase 62.5-MHz clocks in TBI interface mode and with a single 125 MHz clock in SGMII/RTBI interface mode.

The auto-negotiation process is implemented per the IEEE 802.3z specification, Figure 37-6. Auto-negotiation provides an automatic process for two devices to share configuration information and resolve to operate at the highest of their common abilities. Manual configuration can also be accommodated to force a certain level of operation via the GMII Control register and Advertisement register, and auto-negotiation can be disabled via the GMII Control register.

The optional PCS module can be included in the GMAC controller by selecting the TBI/RTBI/SGMII interface option in the coreKit during configuration

### 3.11.1 PCS Functions

- ❖ 8B/10B encoding in transmit path
- ❖ 10B/8B decoding in the receive path
- ❖ Code-group synchronization in the receive path
- ❖ Auto-negotiation
- ❖ Start-of-packet (SOP) and end-of-packet (EOP) detection and generation
- ❖ Collision and Carrier sense generation

#### 3.11.1.1 Transmit

During the transmit process, the GMAC performs the encapsulation function according to the following rules.

- ❖ The first byte of the preamble is replaced with the /S/ code group.
- ❖ All data in the frame are encoded according to 8B/10B encoding standard.
- ❖ The GMAC inserts the /T/R/R/ or /T/R/ code group after the last byte of FCS.
- ❖ An idle code group /I/ is transmitted between frames.
- ❖ Collision and Carrier sense are generated.
- ❖ /V/ Error insertion is also supported.

#### 3.11.1.2 Receive

During the receive process, the GMAC performs the decapsulation function according to the following rules.

- ❖ An /I/S/ code group sequence will cause the internal data valid signal to be asserted.
- ❖ The /S/ code group is replaced by a preamble byte.
- ❖ All data of the received frame are decoded according to 10B/8B decoding standard.
- ❖ The /T/R/R/ or /T/R/ code group sequence causes the internal data valid signal to be deasserted.

#### 3.11.1.3 Synchronization

The synchronization process is responsible for determining whether the underlying receive channel is ready for operation. The GMAC implements synchronization of received code groups in compliance with IEEE 802.3z. After bit synchronization has been achieved by the PHY, and after the GMAC detects three consecutive valid /COMMA/D/ patterns sent by the PHY in the even code group position, the GMAC will acquire synchronization. With synchronization acquired, it is possible to begin auto-negotiation.

If four consecutive invalid code groups or commas in odd clock are received after acquiring synchronization, the synchronization process will restart automatically. A built-in hysteresis will prevent any restarting of the synchronization process that is due to reception of invalid code groups.

#### 3.11.1.4 Auto-Negotiation

A subset of the management registers is supported (Register 48-Register 53) in the GMAC. These registers can be accessed through MCI. The GMAC supports only base page exchange and does not support any next page exchanges.

Once auto-negotiation (AN) is enabled, any one the following events will trigger the AN process.

- ❖ Request from another device (transmitting configuration code groups)

- ❖ Loss of synchronization for more than 10ms
- ❖ Error condition detected while receiving /C/ or /I/ ordered set
- ❖ Auto-negotiation restart bit is enabled

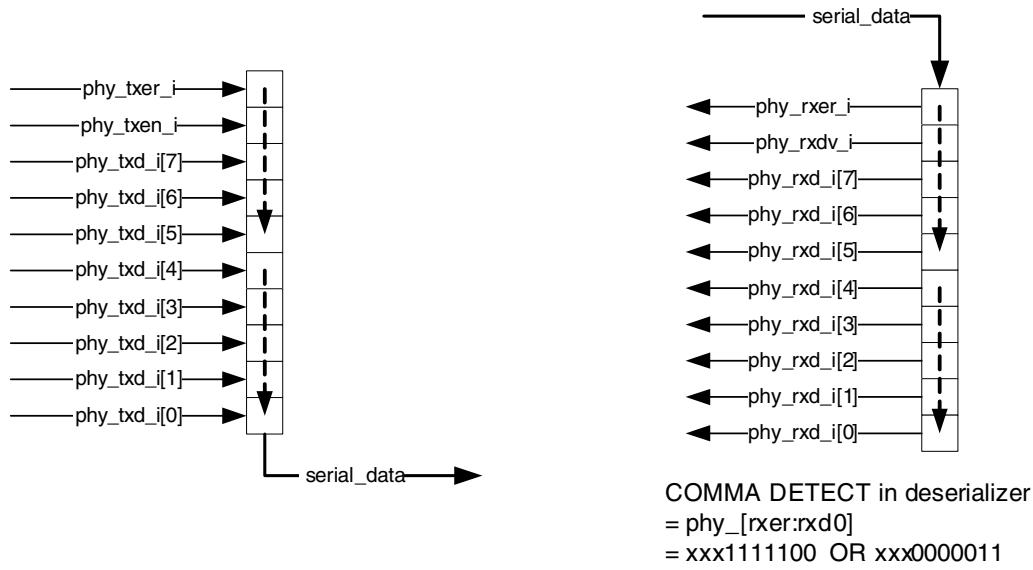
Once AN has been initiated, the GMAC initially sends configuration words with all zeros for a duration determined by the value in the link timer (10 ms). After the link timer duration expires, the contents of the AN Advertisement register are transmitted within the configuration words. The AN Advertisement register should be loaded prior to AN being enabled.

The AN process is completed after the base page exchange. Upon completion of the base page exchange, the ANC bit in the GMII Status register is set, and GMAC is programmed automatically for the Duplex and Flow Control Modes, based on the priority resolution. Once the AN is completed, the GMAC can transmit frames.

### 3.11.1.5 PMA SerDes

The 10-bit TBI signals from the PCS block are connected to the PMA layer of the PHY chip. The connection of the 10-bit transmit data signals with the serializer and the 10-bit receiver data signals from the de-serializer of the PMA are shown in [Figure 3-71](#).

**Figure 3-71 PMA Serializer and De-Serializer**



## 3.12 Reduced Media Independent Interface

The Reduced Media Independent Interface (RMII) specification reduces the pin count between Ethernet PHYs and Switch ASICs (only in 10/100 mode). According to the IEEE 802.3u standard, an MII contains 16 pins for data and control. In devices incorporating multiple MAC or PHY interfaces (such as switches), the number of pins adds significant cost with increase in port count. The RMII specification addresses this problem by reducing the pin count to 7 for each port – a 62.5% decrease in pin count.

- ❖ The RMII module is instantiated between the GMAC and the PHY. This helps translation of the MAC's MII into the RMII. The RMII block has the following characteristics:
- ❖ Supports 10-Mbps and 100-Mbps operating rates. It does not support 1000-Mbps operation.

- ❖ Two clock references are sourced externally, providing independent, 2-bit wide transmit and receive paths. Guidelines for clock connections and phase differences when the GMAC operates with an RMII are provided in “[Clocks With RMII](#)” on page 375.

The RMII interface can be included by selecting the RMII interface in the coreKit during the configuration.

### 3.12.1 Block Diagram

[Figure 3-72](#) shows the position of the RMII block relative to the DWC Ether MAC 10/100/1000 Universal and RMII PHY. The RMII block is placed in front of the DWC Ether MAC 10/100/1000 Universal to translate the MII signals to RMII signals.

**Figure 3-72 RMII Block Diagram**

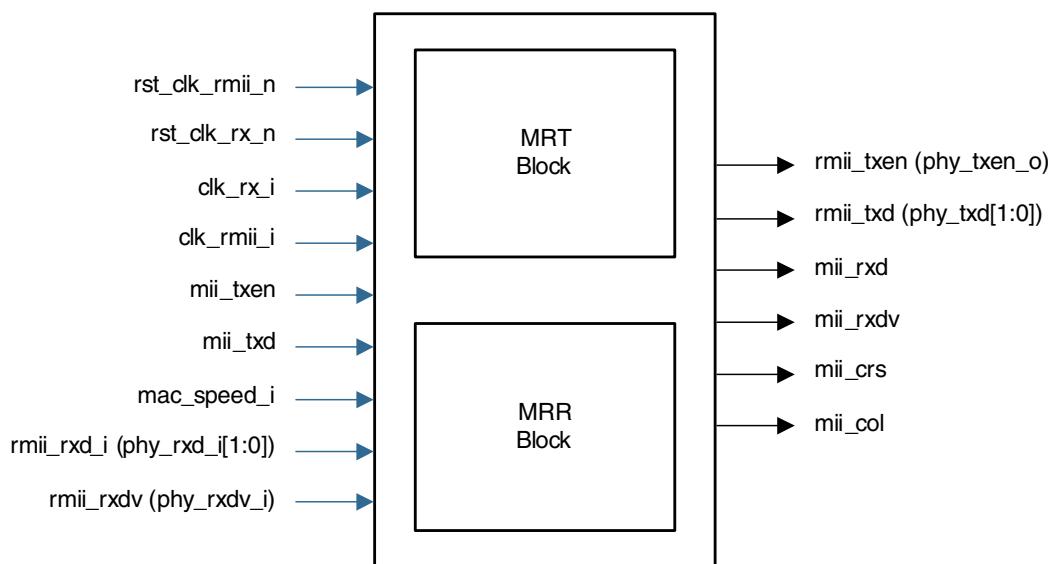


### 3.12.2 Block Overview

The following list describes the RMII’s hardware components, which are shown in [Figure 3-73](#). Each of these blocks is briefly described in the following sections.

**MII-RMII Transmit (MRT) Block:** This block translates all MII transmit signals to RMII transmit signals. All RMII signals are synchronous to clk\_rmii\_i.

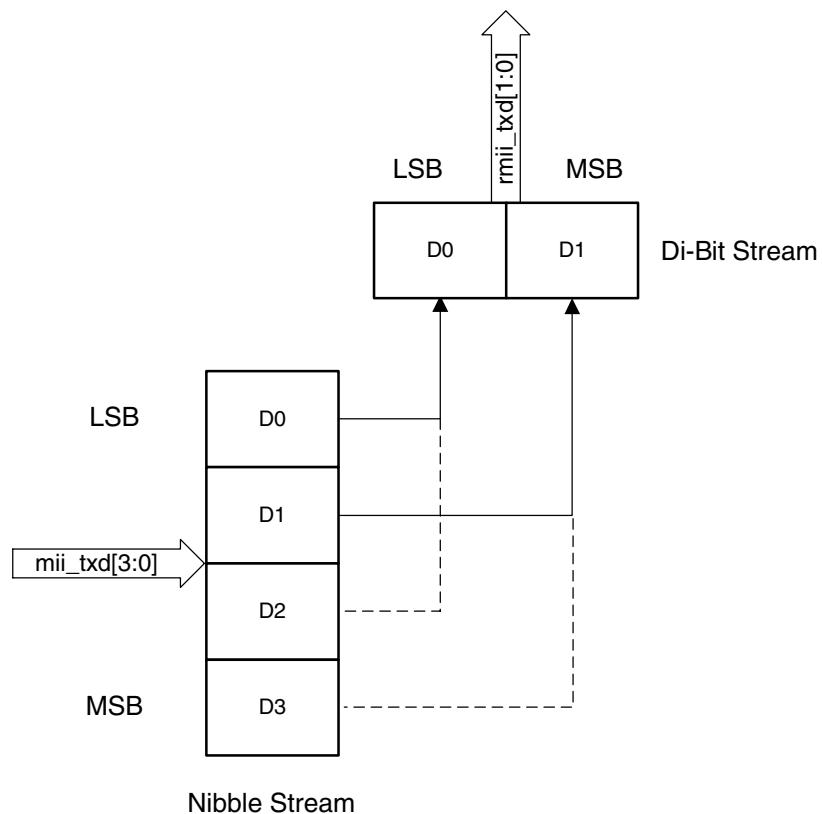
**MII-RMII Receive (MRR) Block:** This block translates all RMII receive signals to MII receive signals. All MII signals are synchronous to clk\_rx\_i. You must ensure that the same clock is connected to both clk\_tx\_i and clk\_rx\_i clock input ports in RMII mode. This is required because clock-MUXing logic is avoided inside the GMAC core.

**Figure 3-73 RMII Pinout**

**Note** The `mac_speed_i` signal configures the RMII to operate at 10 Mbps or 100 Mbps. This signal is either taken directly as a core input or driven from the MAC Configuration register's FES bit. See [Section 8.4](#) for details.

### 3.12.3 Transmit Bit Ordering

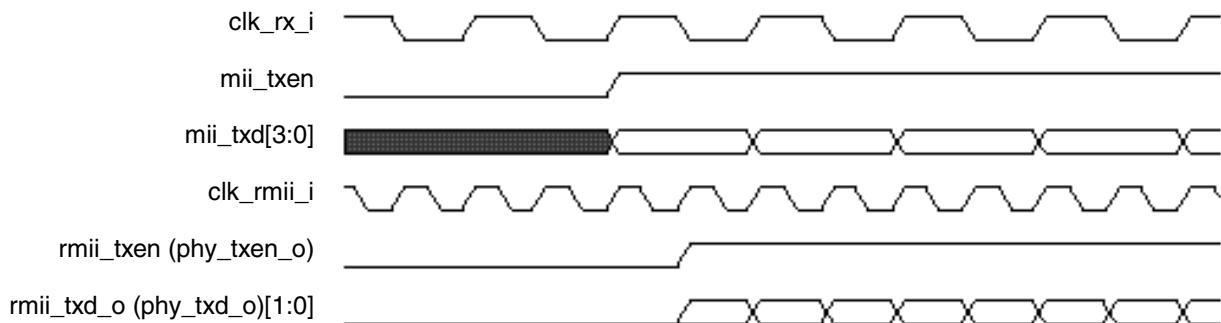
Each nibble from the MII must be transmitted on the RMII a di-bit at a time with the order of di-bit transmission shown in [Figure 3-74](#). The lower order bits (D1 and D0) are transmitted first followed by higher order bits (D2 and D3).

**Figure 3-74 Transmission Bit Ordering**

### 3.12.4 RMII Transmit Timing Diagrams

[Figures 3-75 through 3-78](#) show MII-to-RMII transaction timing.

[Figure 3-75](#) shows the start of MII transmission and the following RMII transmission in 100-Mbps mode.

**Figure 3-75 Start of MII and RMII Transmission in 100-Mbps Mode**

[Figure 3-76](#) shows the end of frame transmission for MII and RMII in 100-Mbps mode.

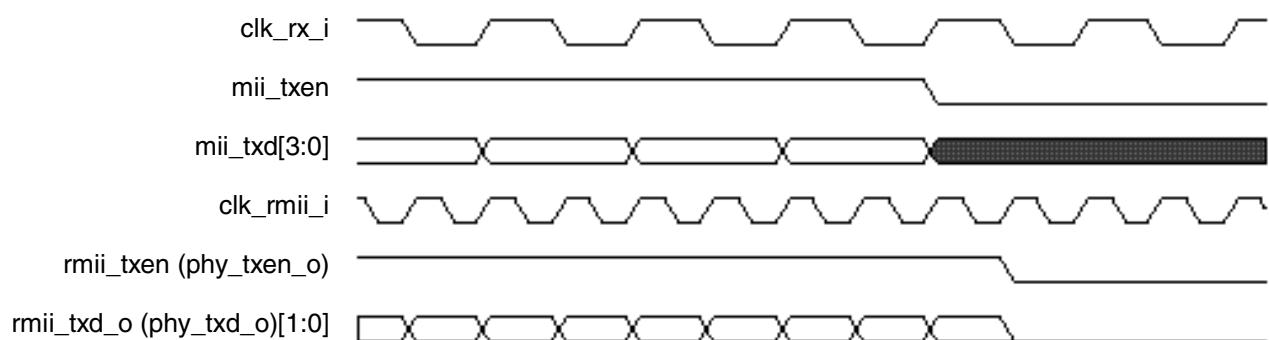
**Figure 3-76 End of MII and RMII Transmission in 100-Mbps Mode**

Figure 3-77 shows the start of MII transmission and the following RMII transmission in 10-Mbps mode.

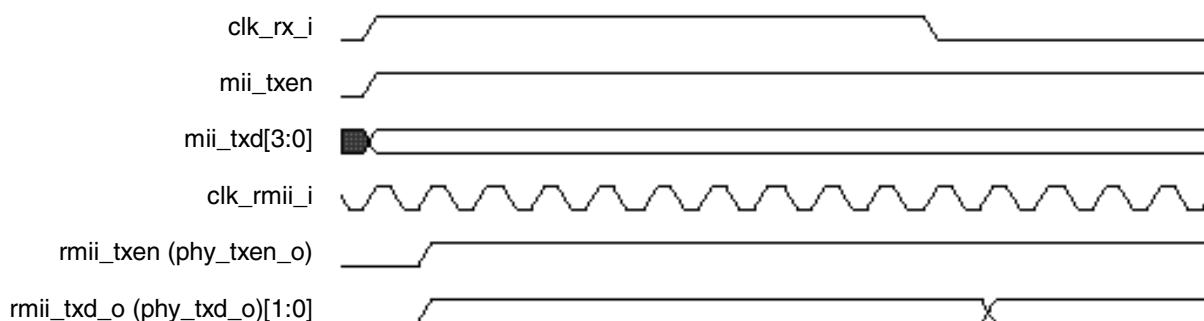
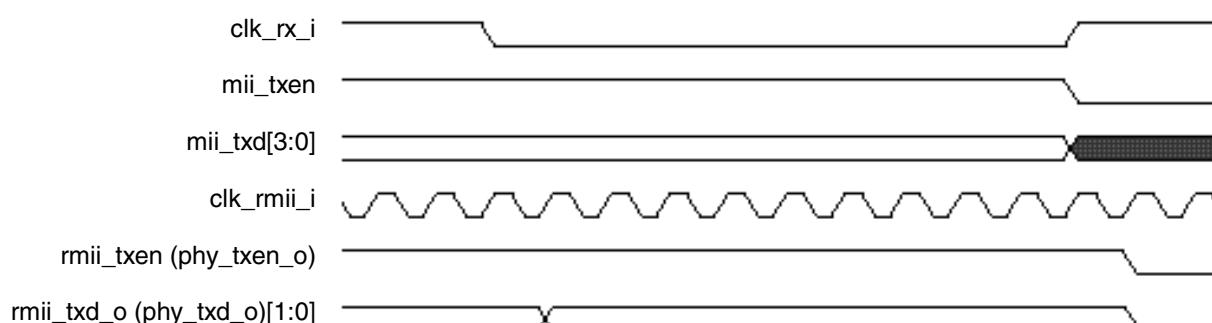
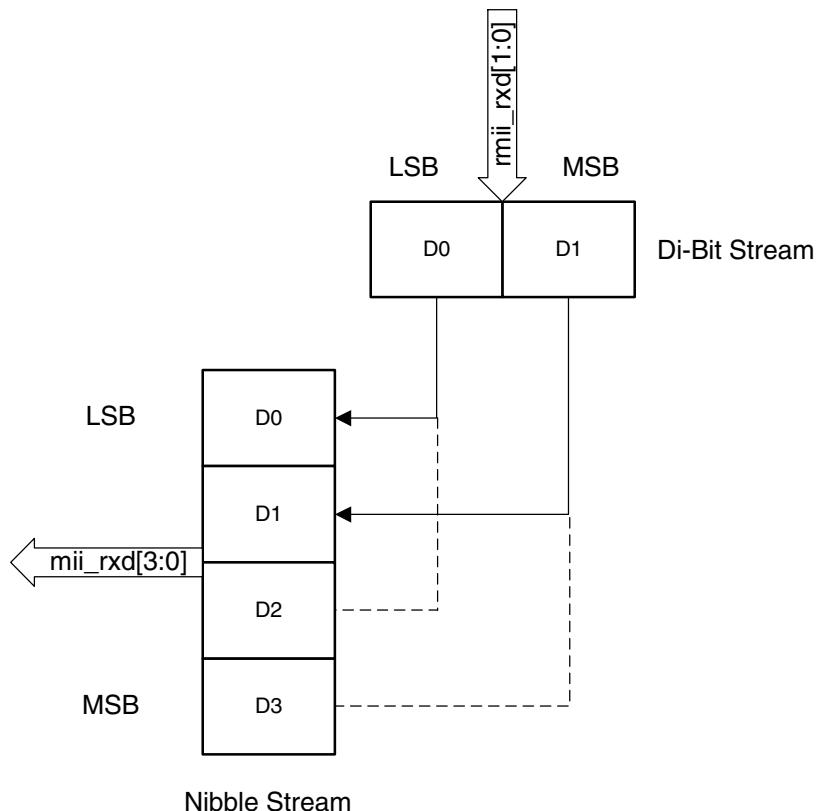
**Figure 3-77 Start of MII and RMII Transmission in 10-Mbps Mode**

Figure 3-78 shows the end of MII transmission and RMII transmission in 10-Mbps mode.

**Figure 3-78 End of MII and RMII Transmission in 10-Mbps Mode**

## Receive Bit Ordering

Each nibble is transmitted to the MII from the di-bit received from the RMII in the nibble transmission order shown in Figure 3-79. The lower order bits (D0 and D1) are received first, followed by the higher order bits (D2 and D3).

**Figure 3-79 Receive Bit Ordering**

### 3.13 Serial Media Independent Interface (SMII)

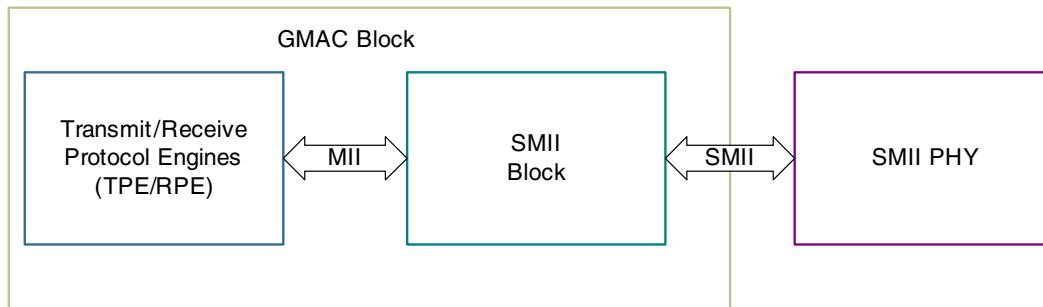
This section outlines the functional specifications of SMII support for the DWC Ethernet GMAC Universal core. The SMII block is designed to fully comply with the SMII specification, revision 2.1, from Cisco. The SMII has the following characteristics:

- ❖ Conveys complete MII information between a 10/100 PHY and MAC with two pins (TX and RX) per port, one global 125 MHz System clock (CLOCK) and one global Synchronization signal (SYNC). Guidelines for clock connections when the GMAC operates with an SMII are provided in “[Clocks With SMII](#)” on page 376.
- ❖ Allows a multi-port MAC/PHY communication with one system clock.
- ❖ Operates in both half and full duplex modes.
- ❖ Allows per packet switching between 10 MBit and 100 MBit data rates.
- ❖ Allows direct MAC to MAC communication.
- ❖ Optional source synchronous mode (four signals viz. TX\_CLK, TX\_SYNC, RX\_CLK and RX\_SYNC replaces SYNC signal).
- ❖ Optional selection of TX\_SYNC as input in source synchronous mode to synchronize the transmit data.
- ❖ The SMII block transmits status in between frames, when enabled. This is useful in MAC to MAC connection.

### 3.13.1 Block Diagram

Figure 3-80 shows the position of the SMII block relative to the GMAC and SMII PHY. The SMII block is between the DWC Ether MAC 10/100/1000 Universal's GMII and the PHY to translate the GMII signals to SMII signals. The pinout is shown in Figure 3-81.

**Figure 3-80 SMII Block Diagram**



### 3.13.2 Block Overview

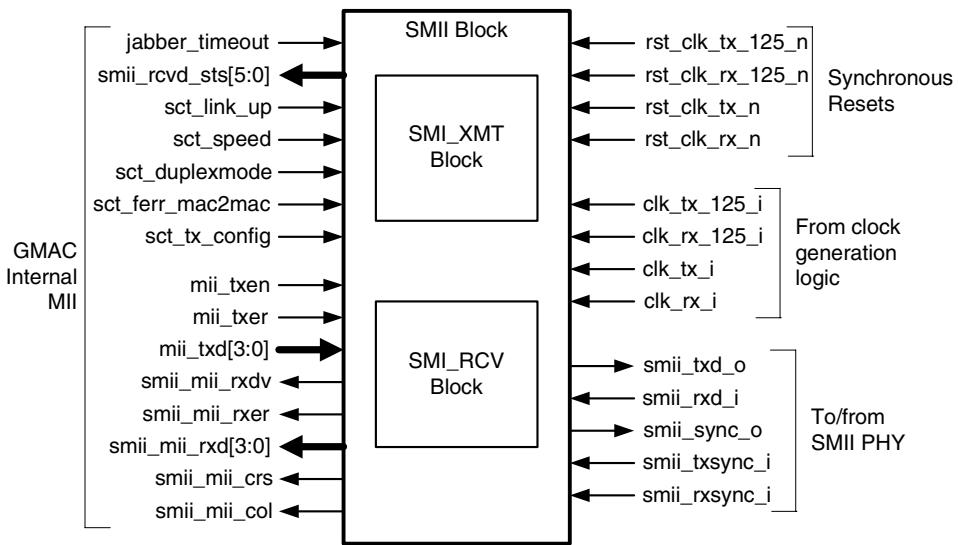
The SMII sub-blocks perform the signal translation.

#### 3.13.2.1 GMII-SMII Transmit (SMI\_XMT) Block

This block converts the two data nibbles and control signals received (at 2.5 MHz and 25 MHz for 10 Mbps and 100 Mbps respectively) from MII transmit interface into one ten bit segment transmitted serially at 125 MHz on the SMII transmit interface. The single ten bit segment is repeated 10 times in 10 Mbps mode. This block generates the SYNC signal (TX\_SYNC signal in source synchronous mode) once every 10 clocks of 125 MHz indicating the beginning of a ten bit segment. In source synchronous mode with TX\_SYNC as input, it synchronizes the transmit data to the received TX\_SYNC.

#### 3.13.2.2 GMII-SMII Receive (SMI\_RCV) Block

This block converts the ten bit segment received serially at 125 MHz from SMII receive interface into two data nibbles and control signals (at 2.5 MHz and 25 MHz for 10 Mbps and 100 Mbps respectively) on the MII receive interface. In 10 Mbps mode, although each ten bit segment is received ten times, only one sample is passed to the MII receive interface.

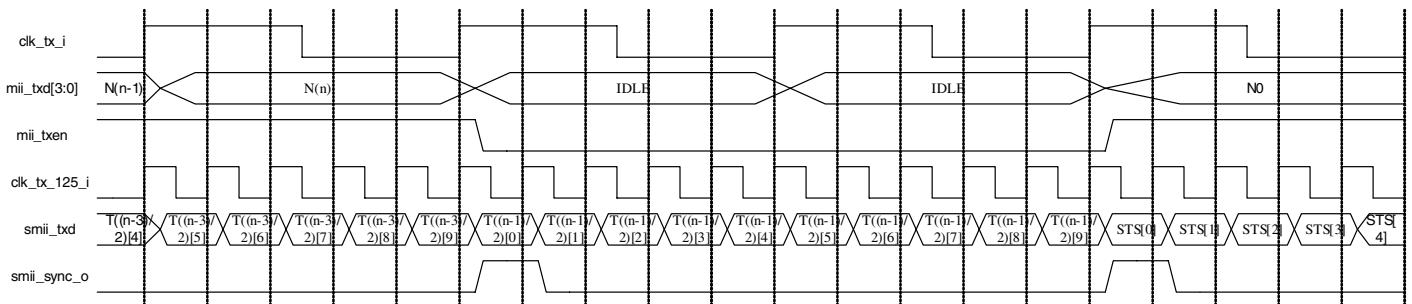
**Figure 3-81 SMII Pinout**

### 3.13.3 SMII Timing



**Note** “T” and “N” in the timing diagrams indicate ten bit segment and nibble, respectively. The relation between the T(m) and N(n) number is given by  $m = (n-1)/2$  for a frame, i.e., when there are 128 nibbles (minimum sized frame) to be transmitted, then N will be from N(0) to N(127). Correspondingly, the T will be from T(0) to T(63) in the waveform below.

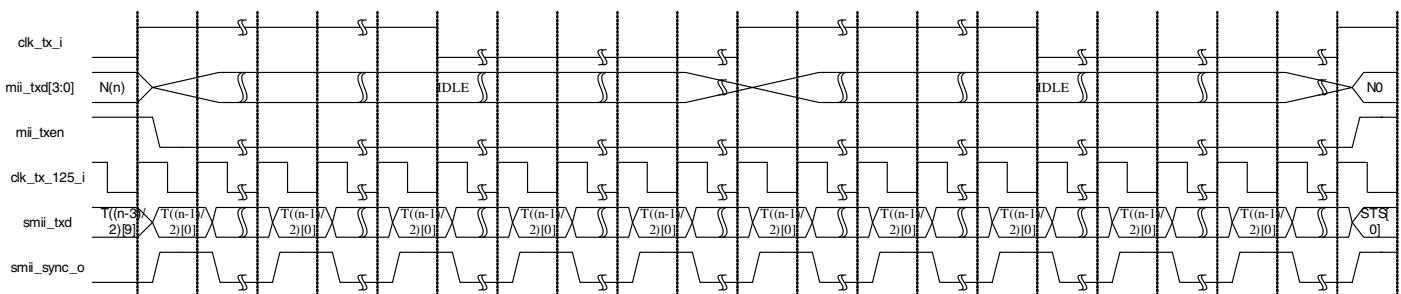
The SMII transmit (SMI\_XMT) block converts the two data nibbles and control signals received at 25 MHz from 100 Mbps MII transmit interface into one ten bit segment transmitted serially at 125 MHz clock on SMII transmit interface. The SMII transmit block sends ten bit status segments in the interpacket gap as defined in the SMII specification. This status is sent only if enabled by setting bit 24 (Transmit Configuration bit) of the MAC Configuration Register 0. This feature is useful for sending status in MAC to MAC connection only. The status includes the SFTERR, FES, DM and LUD bits from MAC Configuration Register 0 and Jabber timeout error signal from the TPE block.

**Figure 3-82 SMII transmit in 100 Mbps Mode**

The SMII transmit (SMI\_XMT) block converts the two data nibbles and control signals received at 2.5 MHz from 10 Mbps MII transmit interface into one ten bit segment transmitted serially at 125 MHz clock on SMII transmit interface. The single ten bit segment is repeated 10 times in 10 Mbps mode. The SMII transmit block sends ten bit status segments in the interpacket gap as defined in the SMII specification. This status is sent only if enabled by setting bit 24 (Transmit Configuration bit) of the MAC Configuration Register 0. This feature is useful for sending status in MAC to MAC connection only. The status includes the SFTERR, FES, DM and LUD bits from MAC Configuration Register 0 and Jabber timeout error signal from the TPE block.

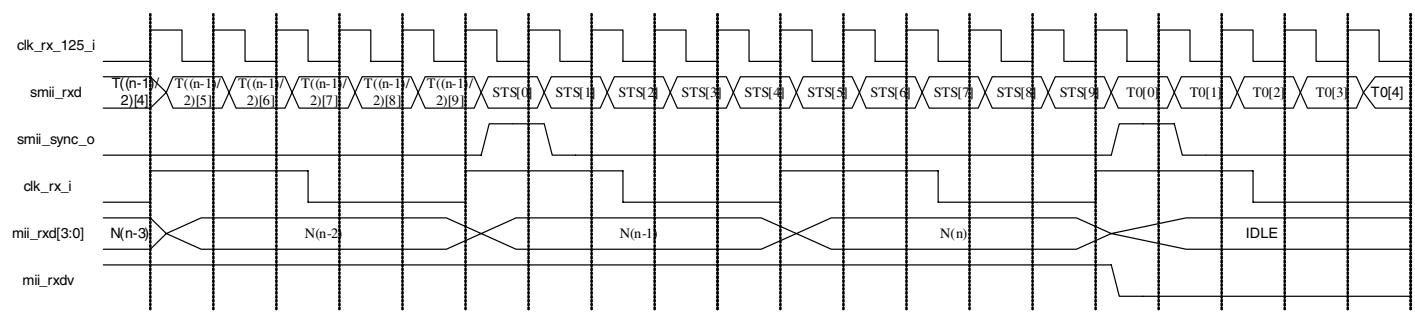
The SMI\_XMT block generates the SYNC signal (TX\_SYNC signal in source synchronous mode) once every 10 clocks of 125 MHz indicating the beginning of a ten bit segment. In source synchronous mode with TX\_SYNC as input, it synchronizes the transmit data to the received TX\_SYNC.

**Figure 3-83 SMII Transmit Block in 10 Mbps Mode**

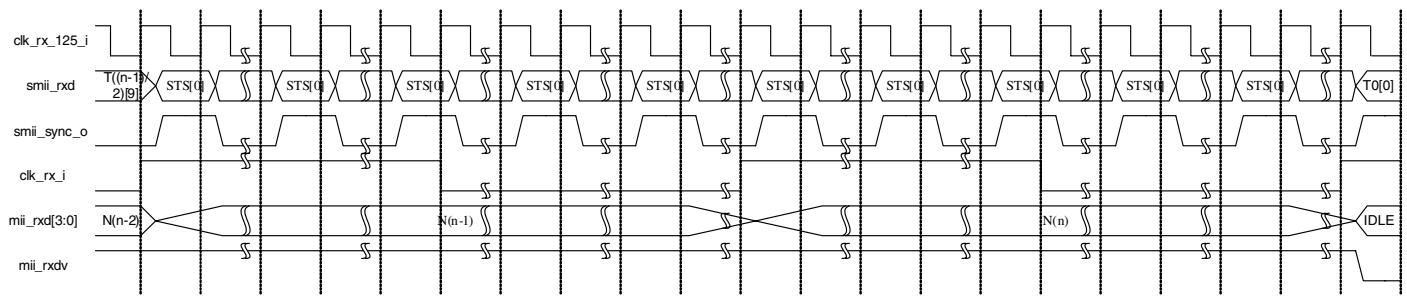


The SMII receive (SMI\_RCV) block converts the ten bit segment received serially at 125 MHz on SMII receive interface into two data nibbles and control signals at 2.5 MHz in 100 Mbps mode on the MII receive interface. The data nibble given to MII receive interface is delayed by one cycle so that receive error (if indicated in the ten bit status segment after the frame) can be signaled to the MAC with the last nibble.

**Figure 3-84 SMII Receive Block in 100 Mbps Mode**



The SMII receive (SMI\_RCV) block converts the ten bit segment received serially at 125 MHz on SMII receive interface into two data nibbles and control signals at 2.5 MHz in 10 Mbps mode on the MII receive interface. In 10 Mbps mode, although each ten bit segment is received ten times, only one sample is passed to the MII receive interface. The data nibble given to MII receive interface is delayed by one cycle so that receive error (if indicated in the ten bit status segment after the frame) can be signaled to the MAC with the last nibble.

**Figure 3-85 SMII Receive Block in 10 Mbps Mode**

The RGMII interrupt mechanism will be reused by the SMII as SGMII/RGMII Status Register 54 is reused by the SMII block.

When the TXSYNC signal is selected as input in Source Synchronous Mode, the smi\_xmt block uses it to synchronize the ten bit segments being transmitted on the SMII transmit interface.

### 3.14 Reduced Gigabit Media Independent Interface

The Reduced Gigabit Media Independent Interface (RGMII) specification reduces the pin count of the interconnection between the DWC Ether MAC 10/100/1000 Universal and the PHY for GMII and MII interfaces. To achieve this, the data path and control signals are reduced and multiplexed together with both the edges of the transmit and receive clocks. For gigabit operation the clocks operate at 125 MHz; for 10/100 operation, the clock rates are 2.5 MHz/25 MHz.

In the DWC Ether MAC 10/100/1000 Universal core, the optional RGMII module is instantiated between the GMAC core's GMII and the PHY to translate the control and data signals between the GMII and RGMII protocols.

The RGMII block has the following characteristics:

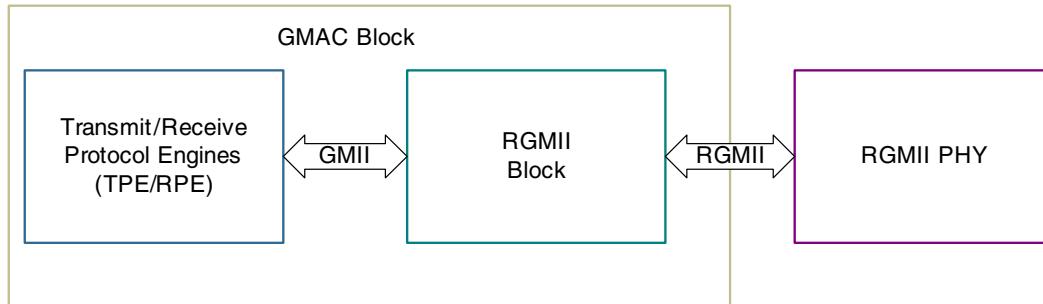
- ❖ Supports 10-Mbps, 100-Mbps, and 1000-Mbps operation rates.
- ❖ For the RGMII block, no extra clock is required because both the edges of the incoming clocks are used. To simplify back-end implementation of the DWC Ether MAC 10/100/1000 Universal, there are separate clock inputs (180 degrees out-of-phase with the associated transmit/receive clocks) for logic that uses the falling edges.
- ❖ Guidelines for clock connections when the GMAC operates with an RGMII are given in “[Clocks With RGMII](#)” on page 379.
- ❖ There is a synthesis option for the DWC Ether MAC 10/100/1000 Universal to inherently delay the RGMII transmit clock with respect to the data and control signals, as described in Version 2.0 of RGMII specification.
- ❖ The RGMII block extracts the in-band (link speed, duplex mode and link status) status signals from the PHY and provides them to the GMAC core logic for link detection.

The RGMII module can be included in the GMAC core controller by selecting the RGMII interface in the coreKit during configuration.

### 3.14.1 Block Diagram

Figure 3-86 shows the position of the RGMII block relative to the GMAC and RGMII PHY. The RGMII block is between the DWC Ether MAC 10/100/1000 Universal's GMII and the PHY to translate the GMII signals to RGMII signals.

**Figure 3-86 RGMII Block Diagram**



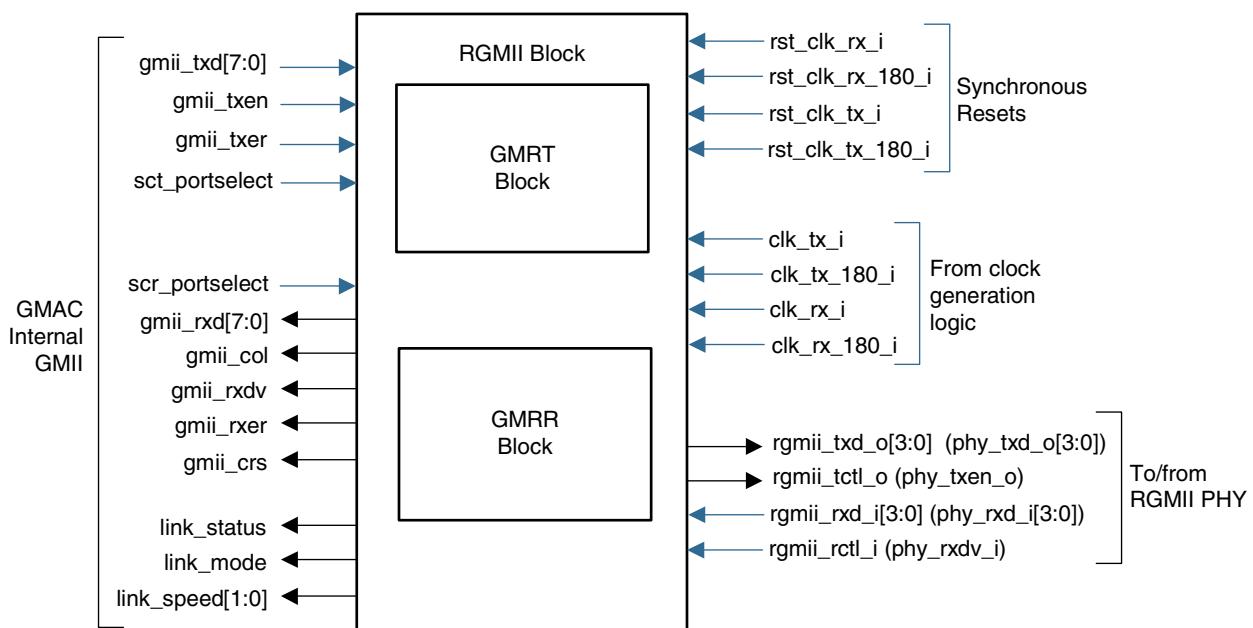
### 3.14.2 Block Overview

The RGMII sub-blocks perform the signal translation:

**GMII-RGMII Transmit (GMRT) Block:** Translates all GMII transmit signals to RGMII signals. The GMRT registers all GMII input signals at the rising edge of `clk_tx_i` and generates all RGMII transmit signals at the rising edge of either `clk_tx_i` or `clk_tx_180_i`.

**GMII-RGMII Receive (GMRR) Block:** Translates all RGMII receive signals to GMII receive signals. The GMRR registers all RGMII input signals at the rising edge of either `clk_rx_i` or `clk_rx_180_i` and generates all GMII receive signals at the rising edge of `clk_rx_i`.

**Figure 3-87 RGMII Block Pinout Diagram**



### 3.14.3 RGMII Clocks

RGMII operation uses both edges of the RGMII transmit and receive clocks (clk\_tx\_i and clk\_rx\_i), which run at 125 MHz for gigabit operation or 2.5 MHz/25 MHZ for 10/100 operation. These clocks come from a separate PLL logic block. To simplify implementation of the DWC Ether MAC 10/100/1000 Universal, the RGMII block provides two additional inputs to drive the logic that would use the falling edges of clk\_tx\_i and clk\_rx\_i:

- ❖ clk\_tx\_180\_i must be 180 degrees out-of-phase with respect to clk\_tx\_i
- ❖ clk\_rx\_180\_i must be 180 degrees out-of-phase with respect to clk\_rx\_i

Separate synchronous inputs are present to synchronously reset the logic driven by each of the four RGMII clocks.

### 3.14.4 Signal Conversion

#### 3.14.4.1 Transmit Data Conversion

As defined in the RGMII specification, multiplexing of data and control in gigabit mode is accomplished by using both edges of the transmit clock. The RGMII block accepts the 8-bit GMII transmit data from the DWC Ether MAC 10/100/1000 Universal GMII (gmii\_txd[7:0]) on the rising edge of clk\_tx\_i. For gigabit operation, the RGMII block then converts gmii\_txd[7:0] to two 4-bit nibbles and transmits the data on the RGMII transmit data port (rgmii\_txd\_o[3:0]) as follows:

- ❖ At the rising edge of clk\_tx\_i, rgmii\_txd\_o[3:0] contains gmii\_txd[3:0]
- ❖ At the falling edge of clk\_tx\_i (rising edge of clk\_tx\_180\_i), rgmii\_txd\_o[3:0] contains gmii\_txd[7:4]

For 10/100 mode, the transmit data to the PHY is 4 bits. So, for 10/100 mode, the RGMII block drives the gmii\_txd[3:0] data to the PHY on rgmii\_txd\_o[3:0] at the rising edge of clk\_tx\_i.

#### 3.14.4.2 Transmit Control Signal Conversion

The RGMII block accepts the GMII transmit control signals (gmii\_txen and gmii\_txer) from the DWC Ether MAC 10/100/1000 Universal GMII on the rising edge of clk\_tx\_i, then multiplexes gmii\_txen and gmii\_txer onto the rgmii\_tctl\_o output as required by the RGMII specification:

- ❖ At the rising edge of clk\_tx\_i, rgmii\_tctl\_o is gmii\_txen
- ❖ At the falling edge of clk\_tx\_i (rising edge of clk\_tx\_180\_i), rgmii\_tctl\_o is the logical XOR of gmii\_txen and gmii\_txer

Table 3-25 shows the mapping of the PLS\_DATA.request parameters from the DWC Ether MAC 10/100/1000 Universal to the corresponding signaling on the GMII and RGMII control and data signals.

**Table 3-25 Encoding of gmii\_txen, gmii\_txer, rgmii\_tctl\_o, and rgmii\_txd\_o**

gmii_txen	gmii_txer	gmii_txd <sup>a</sup>	rgmii_tctl_o <sup>b</sup>	Description	PLS_DATA.request parameter
0	0	0x00–0xFF	0,0	Normal inter-frame	TRANSMIT_COMPLETE
0	1	0x00–0x0E	0,1	Reserved	
0	1	0x0F	0,1	Carrier extend	EXTEND (8 bits)
0	1	0x10–0x1E	0,1	Reserved	

**Table 3-25 Encoding of gmii\_txen, gmii\_txer, rgmii\_tctl\_o, and rgmii\_txd\_o**

<b>gmii_txen</b>	<b>gmii_txer</b>	<b>gmii_txd<sup>a</sup></b>	<b>rgmii_tctl_o<sup>b</sup></b>	<b>Description</b>	<b>PLS_DATA.request parameter</b>
0	1	0x1F	0,1	Carrier extend error	EXTEND_ERROR (8 bits)
0	1	0x20–0xFF	0,1	Reserved	
1	0	0x00–0xFF	1,1	Normal data transmission	ZERO, ONE (8 bits)
1	1	0x00–0xFF	1,0	Transmit error propagation	No applicable parameter

a. When the RGMII interface is configured to transmit the configuration during the IFG, then rgmii\_txd[3:0] will reflect the Duplex Mode, Port Select, Speed (encoded as 00 for 10 Mbps, 01 for 100 Mbps and 10 for 1000 Mbps), and Link Up/Down bits of the MAC Configuration Register, during this period.

b. Values at rising, falling edges of clk\_tx\_i.

#### 3.14.4.3 Receive Data Conversion

For Gigabit mode, the RGMII block registers the 4-bit data on rgmii\_rxd\_i[3:0] at both edges of the receive clock (clk\_rx\_i) and transfers the resulting 8-bit data to the DWC Ether MAC 10/100/1000 Universal GMII on the rising edge of clk\_rx\_i, as follows:

- ❖ gmii\_rxd[3:0] is the value received on rgmii\_rxd\_i[3:0] at the rising edge of clk\_rx\_i
- ❖ gmii\_rxd[7:4] is the value received on rgmii\_rxd\_i[3:0] at the falling edge of clk\_rx\_i (rising edge of clk\_rx\_180\_i)

For 10/100 mode, the RGMII block registers the 4-bit data on rgmii\_rxd\_i[3:0] only at the rising edge of clk\_rx\_i and transfers the data to the DWC Ether MAC 10/100/1000 Universal GMII on the rising edge of clk\_rx\_i, as follows:

- ❖ gmii\_rxd[3:0] is the value received on rgmii\_rxd\_i[3:0] at the rising edge of clk\_rx\_i
- ❖ gmii\_rxd[7:4] is 0x0

#### 3.14.4.4 Receive Control Signal Conversion

The RGMII block samples the rgmii\_rctl\_i signal from the PHY at both edges of clk\_rx\_i and drives the resulting GMII receive control signals to the DWC Ether MAC 10/100/1000 Universal GMII as follows:

- ❖ gmii\_rxdv is the value received on rgmii\_rctl\_i at the rising edge of clk\_rx\_i.
- ❖ gmii\_rxer is the logical XOR of the following two signals:
  - ◆ The value received on rgmii\_rctl\_i at the rising edge of clk\_rx\_i (gmii\_rxdv)
  - ◆ The value received on rgmii\_rctl\_i at the falling edge of clk\_rx\_i (rising edge of clk\_rx\_180\_i)

#### 3.14.4.5 Receive Status Signal Conversion

To generate the GMII link status signals, the RGMII block decodes the rgmii\_rxd\_i[3:0] value sampled at the rising edge of clk\_rx\_i when the value of rgmii\_rctl\_i is 0 for both the rising and falling edges of clk\_rx\_i (normal inter-frame value). The decoded status signals are:

- ❖ link\_status is the value of rgmii\_rxd\_i[0]
- ❖ link\_speed is the value of rgmii\_rxd\_i[2:1]

- ❖ link\_mode is the value of rgmii\_rxd\_i[3]

[Table 3-26](#) shows the mapping of the PLS\_DATA.indicate parameters from the DWC Ether MAC 10/100/1000 Universal to the corresponding signalling on the GMII and RGMII control and data signals. In addition, the RGMII block asserts gmii\_crs to the DWC Ether MAC 10/100/1000 Universal GMII when any of the following conditions is true:

- ❖ gmii\_rxrdv is true.
- ❖ gmii\_rxrdv is false, gmii\_rxer is true, and the value of gmii\_rxd is 0xFF.
- ❖ A carrier extend, carrier extend error, or false carrier occurs in gigabit mode (see [Table 3-26](#)).
- ❖ A false carrier occurs in 10/100 mode.
- ❖ The RGMII is transmitting data, carrier extend, or carrier extend error on the RGMII transmit outputs.

The RGMII block asserts gmii\_col to the GMAC GMII when both of the following conditions is true:

- ❖ The RGMII is transmitting data, carrier extend, or carrier extend error on the RGMII transmit outputs.
- ❖ The RGMII is asserting either gmii\_crs or gmii\_rxrdv.

**Table 3-26 Decoding of rgmii\_rctl\_i and rgmii\_rxd\_i**

rgmii_rctl_i a	gmii_rxd[7:0] ([3:0] in 10/100 mode)	gmii_rxrdv	gmii_rxer	Description	PLS_DATA.indicate or PHY_status parameter
0,0	XXXXXXXX0 or XXXXXXXX1	0	0	Normal inter-frame	Indicates link status: <ul style="list-style-type: none"> <li>• 0: down</li> <li>• 1: up</li> </ul>
0,0	XXXXX00X or XXXXX01X or XXXXX10X or XXXXX11X	0	0	Normal inter-frame	Indicates receive clock frequency: <ul style="list-style-type: none"> <li>• 00: 2.5 MHz</li> <li>• 01: 25 MHz</li> <li>• 10: 125 MHz</li> <li>• 11: Reserved</li> </ul>
0,0	XXXX1XXX or XXXX0XXX	0	0	Normal inter-frame	Indicates duplex status: <ul style="list-style-type: none"> <li>• 0: half-duplex</li> <li>• 1: full-duplex</li> </ul>
0,1	0x0E (or 0xE in 10/100 mode)	0	1	False carrier indication	False carrier present
0,1	0x0F	0	1	Carrier extend	EXTEND (8 bits)
0,1	0x1F	0	1	Carrier extend error	ZERO, ONE (8 bits)
0,1	0xFF (or 0xF in 10/100 mode)	0	1	Carrier sense	PLS_Carrier.Indicate
1,1	0x00–0xFF	1	0	Normal data reception	ZERO, ONE (8 bits)

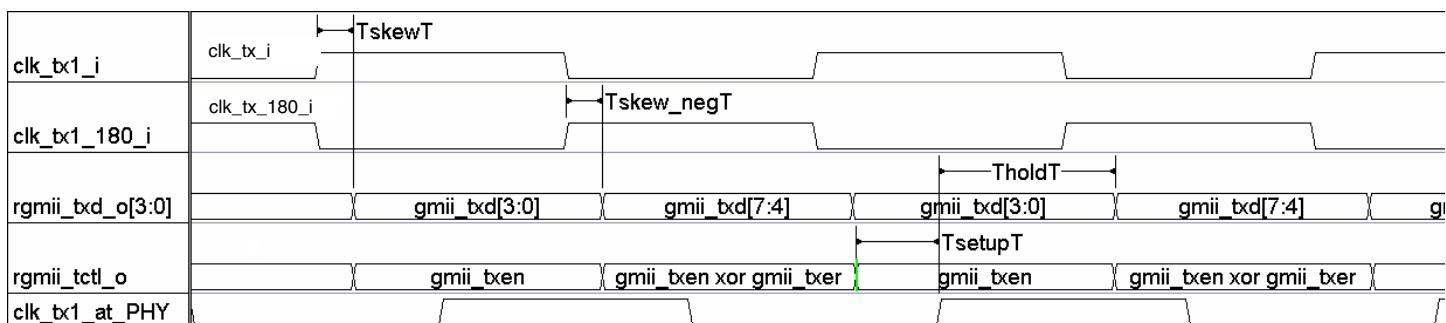
**Table 3-26 Decoding of rgmii\_rctl\_i and rgmii\_rxd\_i (Continued)**

rgmii_rctl_i a	gmii_rxd[7:0] ([3:0] in 10/100 mode)	gmii_rxrv	gmii_rxer	Description	PLS_DATA.indicate or PHY_status parameter
1,0	0x00–0xFF	1	1	Data reception error	ZERO, ONE (8 bits)

a. Values at rising, falling edges of clk\_rx\_i.

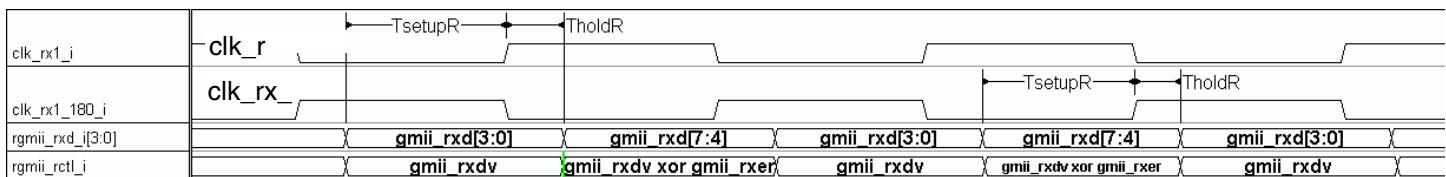
### 3.14.5 RGMII Transmit Timing

Figure 3-88 shows the timing for RGMII transmission. The RGMII specification defines a transmit skew (TskewT) of  $\pm 0.5$  ns. As shown in Figure 3-88, the trace delay on the PC board for clk\_tx\_i going to a receiving PHY must satisfy the set-up and hold time requirements of the RGMII transmit signals at the receiving PHY.

**Figure 3-88 RGMII Transmit Timing**

### 3.14.6 RGMII Receive Timing

Figure 3-89 shows the receive timing for RGMII. The TsetupR and TholdR values are required to be a minimum of 1 ns.

**Figure 3-89 RGMII Receive Timing**

## 3.15 Reduced Ten-Bit Interface

The Reduced Ten-Bit Interface (RTBI) specification reduces the pin count of the interconnection between the GMAC-UNIV and the PHY for TBI interface. To achieve this, the transmit and receive code-groups are multiplexed together with both the edges of the transmit and receive clocks.

In the GMAC-UNIV core, the optional RTBI module is instantiated between the GMAC's PCS module and the PHY to translate the code-group signals between the TBI and RTBI protocols. As the GMAC-UNIV's TBI supports only 1000BASE-X, the RTBI also supports only 1000BASE-X and is not valid for 10/100 Mbps operation.

The RTBI block has the following characteristics:

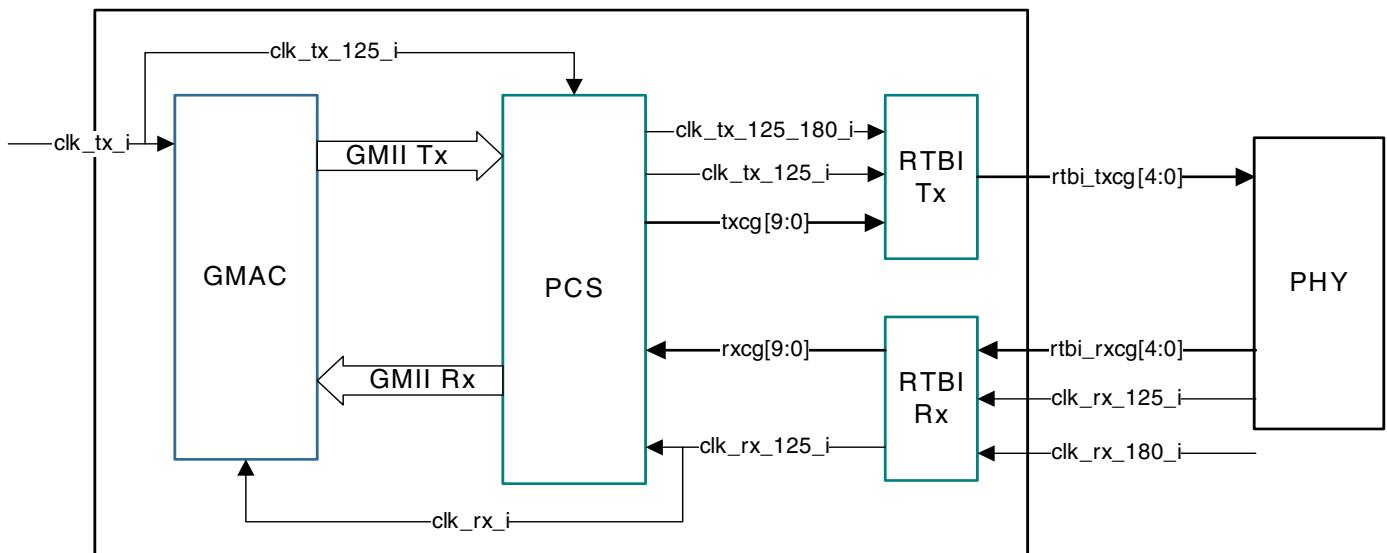
- ❖ For the RTBI block, no extra clock is required because both the edges of the incoming clocks are used. To simplify back-end implementation of the GMAC-UNIV, there are separate clock inputs (180 degrees out-of-phase with the associated transmit/receive clocks) for logic that uses the falling edges.
- ❖ Guidelines for clock connections when the GMAC operates with an RTBI are given in “[Clocks With RTBI](#)” on page [380](#).
- ❖ There is a synthesis option for the GMAC-UNIV to inherently delay the RTBI transmit clock with respect to the data and control signals, as described in Version 2.0 of RGMII/RTBI specification.

The RTBI module can be included in the GMAC-UNIV controller by selecting the RTBI interface in the coreKit during configuration.

### 3.15.1 Block Diagram

[Figure 3-90](#) shows the position of the RTBI block and its I/O signals relative to the GMAC and RTBI PHY.

**Figure 3-90 RTBI Block Diagram**



### 3.15.2 Block Diagram Overview

**Transmit (RTBI Tx) Block:** Translates the 10-bit Transmit code-group signal from PCS module to 5-bit transmit code-group. The RTBI Tx registers the input signals at the rising edge of `clk_tx_125_i` and generates the RTBI transmit signals at the rising edge of either `clk_tx_125_i` or `clk_tx_125_180_i`.

Note that `clk_tx_125_180_i` should be 180 degrees out-of-phase with respect to the `clk_tx_125_i` clock.

**Receive (RTBI Rx) Block:** Translates the 5-bit code-group signals received from the PHY to 10-bit code-group towards the PCS. The RTBI Rx registers the input signals at the rising edge of either `clk_rx_125_i` or `clk_rx_180_i` and generates the `rxcg[9:0]` signal output at the rising edge of `clk_rx_125_i`.

Note that `clk_rx_180_i` should be 180 degrees out-of-phase with respect to the `clk_rx_125_i` clock.

### 3.15.3 RTBI Transmit Timing

In the default mode, the RTBI transmitter outputs the lower 5 bits of the input txcg[9:0] (from PCS) at the rising edge of clk\_tx\_125\_i followed by the upper 5 bits of txcg[9:0] on the falling edge of clk\_tx\_125\_i.

## 3.16 Serial Gigabit Media Independent Interface

The Serial Gigabit Media Independent Interface (SGMII) defines the interface from the Gigabit MAC to the Gigabit PHY. Using the SGMII lowers the pin count required for the GMII. The interface definition supports all speed modes (10 Mbps, 100 Mbps, and 1 Gbps). The reduced pin count comes at the cost of higher power consumption, mainly because the SGMII interface maintains a constant 1250-MHz clock rate, regardless of the MAC's operating speed.

This SGMII block has the following characteristics:

- ❖ Supports 10-Mbps, 100-Mbps and 1000-Mbps operations.
- ❖ Provides in-band (link speed, duplex mode and link status) status signals from PHY provided to GMAC for link detection.
- ❖ The Synopsys RTL deliverable *does not include* the SerDes and high-speed I/O blocks. You must develop these modules to suit the technology and ASIC manufacture process.
- ❖ Requires separate 125-MHz clock inputs for transmit and receive paths.

Guidelines for clock connections when the GMAC operates with an SGMII are given in “[Clocks With SGMII](#)” on page [373](#).

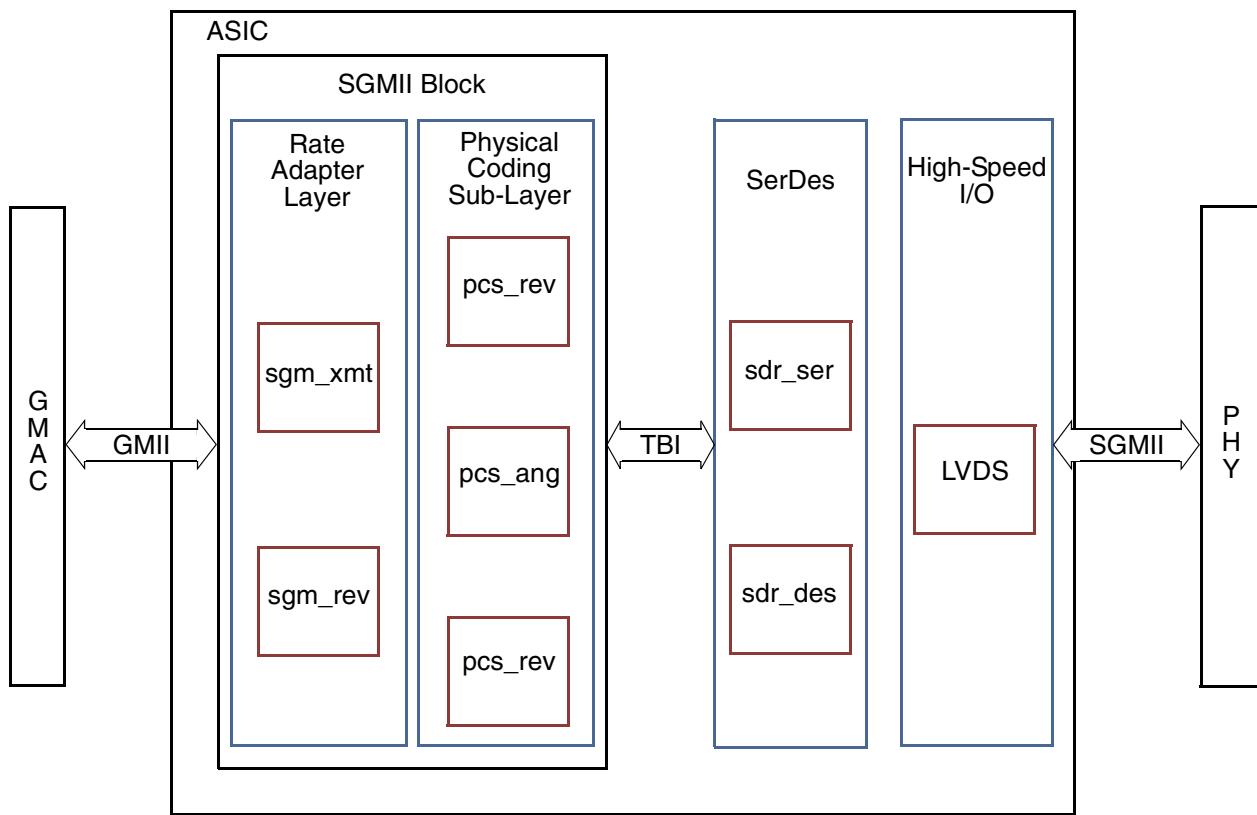
The SGMII module can be included in the DWC Ether MAC 10/100/1000 Universal controller by selecting the SGMII PHY interface in the coreKit during configuration.

### 3.16.1 Block Diagram

[Figure 3-91](#) shows the placement of the SGMII block with respect to the DWC Ether MAC 10/100/1000 Universal and PHY blocks.

### 3.16.2 Block Overview

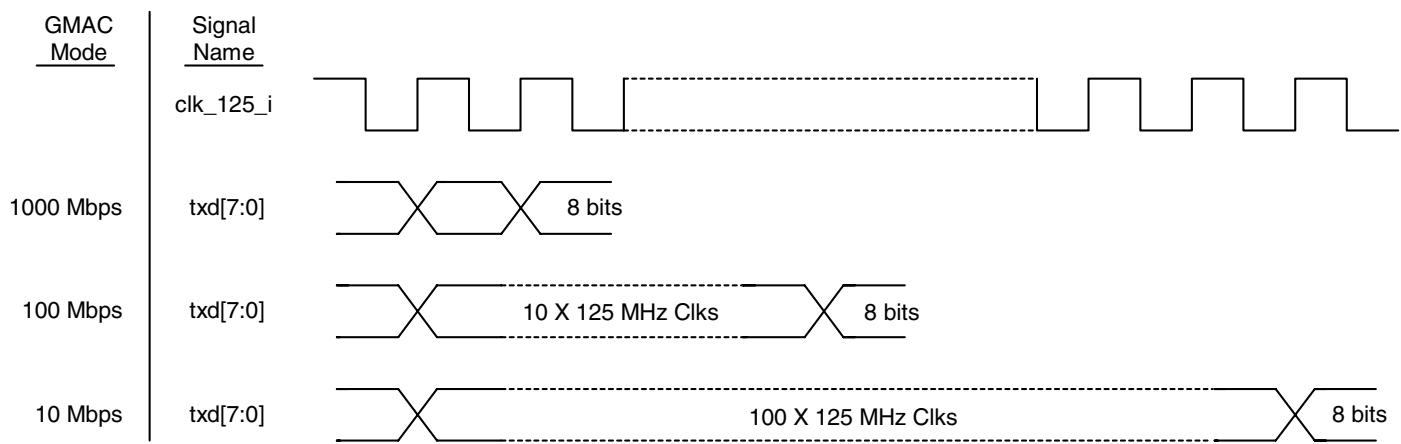
The SGMII block comprises the Rate Adapter Layer (RAL) and the PCS (Physical Coding Scheme) modules as shown in [Figure 3-91](#). You must implement the SerDes and high-speed I/O modules to implement the actual SGMII interface.

**Figure 3-91 SGMII Block Diagram**

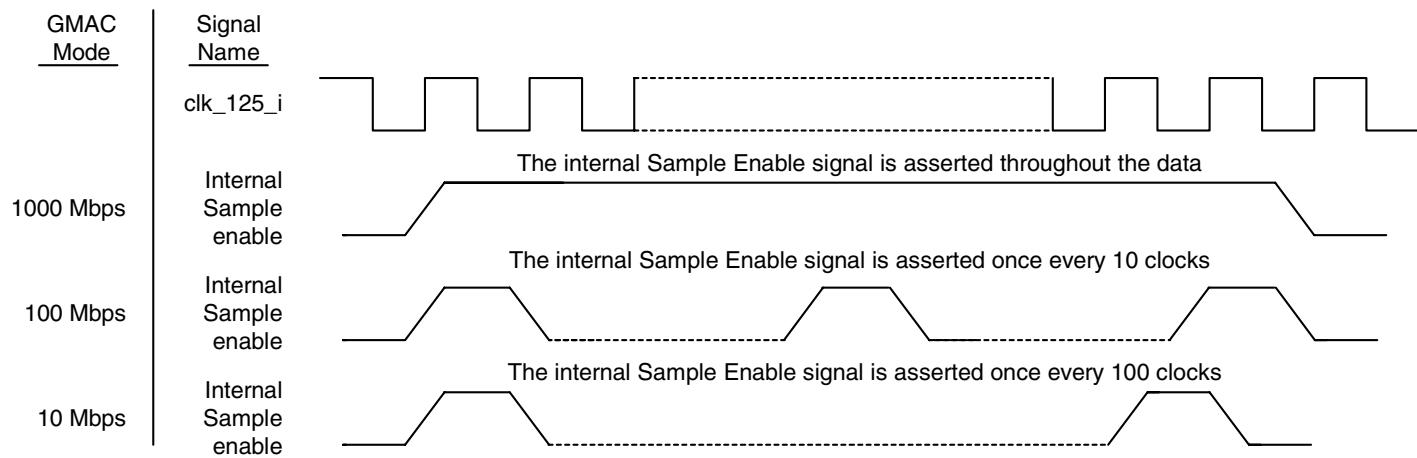
### 3.16.3 Block Descriptions

#### 3.16.3.1 Rate Adapter Layer (RAL)

Because the PCS layer runs on a 125-MHz clock while the GMII interface may run on a 2.5-MHz, 25-MHz, or 125-MHz clock, the data to/from the GMII must be stretched in the transmit path, based on the DWC Ether MAC 10/100/1000 Universal data rate. [Figure 3-92](#) shows how a single 8-bit data stream is stretched with respect to the 125-Mhz clock. If the DWC Ether MAC 10/100/1000 Universal is in the Gigabit mode, then the data is not stretched at all, and takes only one clock cycle. However, if the GMAC is in either the 100 Mbps or 10-Mbps mode, the data is repeated for 10 or 100 consecutive 125-MHz clock cycles, respectively. In [Figure 3-92](#), data is repeated 10 or 100 times in the RAL's transmit path to match the speed mode of the MAC layer. `txd[7:0]` is an internal signal.

**Figure 3-92 Data Stretched in the Tx Path With a 125-MHz Clock**

In the receive path, the SGMII PHY stretches the data according to the MAC's speed before it sends it out to the SGMII block. Thus, the RAL receives the data repeated 10 or 100 times, depending on what mode the MAC is in. Therefore, the RAL does not have to repeat that data, as in the case of the transmit path. It only samples the data once every 10 or 100 clocks, as shown in [Figure 3-93](#). (Refer to the Serial GMII Specification, Revision 1.7, for additional information.)

**Figure 3-93 Data Stretched in the Rx Path With a 125-MHz Clock**

### 3.16.3.2 Physical Coding Sub-Layer (PCS)

In the transmit path, the Physical Coding Sub-Layer takes the 8-bit-wide GMII data from the RAL and encodes it using the 8-bit/10-bit algorithm to supply the SerDes block with a 10-bit-wide data stream. In the receive block, the Physical Coding Sub-Layer takes 10-bit-wide data from the SerDes block and decodes it to supply the RAL block with an 8-bit-wide data stream.

The PCS module is configured with the changes specific to SGMII as given in the SGMII Specification. These changes are with respect to the Link Timer duration (1.6 ms instead of 10 ms) and the Configuration Register codes transmitted during auto-negotiation.

The tx\_config\_reg[15:0] bits sent by the MAC during Auto-negotiation depend on whether the Transmit Configuration register bit is enabled for the SGMII interface. [Table 3-27](#) describes this in detail.

**Table 3-27 tx\_config\_reg[15:0] Setting When Transmit Configuration Register Bit Is Enabled/Disabled**

Transmit Configuration Bit		
Bit	Disabled (Default)	Enabled
15	0	Link Up/Down (Bit 8 of MAC Configuration register)
14	1	1
13	0	0
12	0	Duplex (Bit 11 of MAC Configuration register)
11:10	00	<ul style="list-style-type: none"> <li>• 10 in 1000-Mbps mode</li> <li>• 01 in 100-Mbps mode</li> <li>• 00 in 10-Mbps mode</li> </ul> <p>Where the speed is determined by the Port Select and Speed Control bits in the MAC Configuration register</p>
9:1	00H	00H
0	1	1

### 3.16.3.3 SerDes

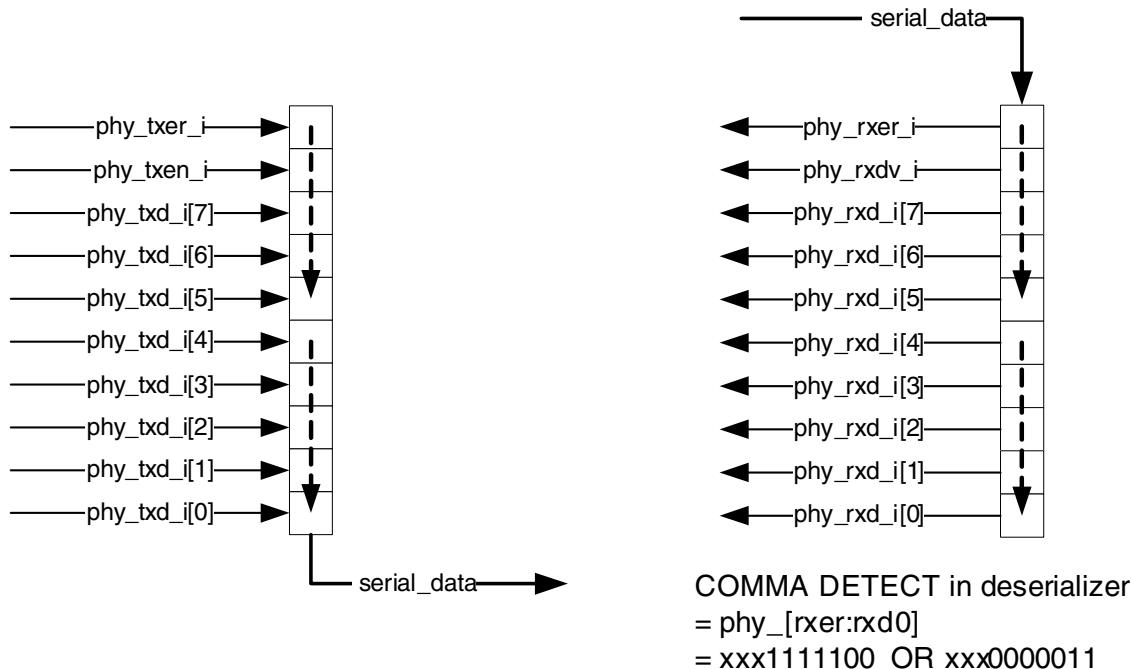
In the transmit path, the SerDes block takes the 10-bit-wide parallel data from the PCS and converts it to a serial format. In the receive path, the SerDes block takes the serial SGMII input and converts it to a parallel 10-bit-wide data format for the PCS layer.

The connection of the 10-bit transmit data signals with the serializer and the 10-bit receiver data signals from the de-serializer are shown in [Figure 3-94](#).



**Note** The provided RTL is supplied/used for simulation purposes only and is not to be used for synthesis. The SerDes and LVDS I/O blocks are custom blocks of technology-specific library parts. See [Figure 3-95](#) for a block diagram of the GMAC core (DUT) with support for an SGMII interface.

**Figure 3-94 SGMII Serializer and De-Serializer**



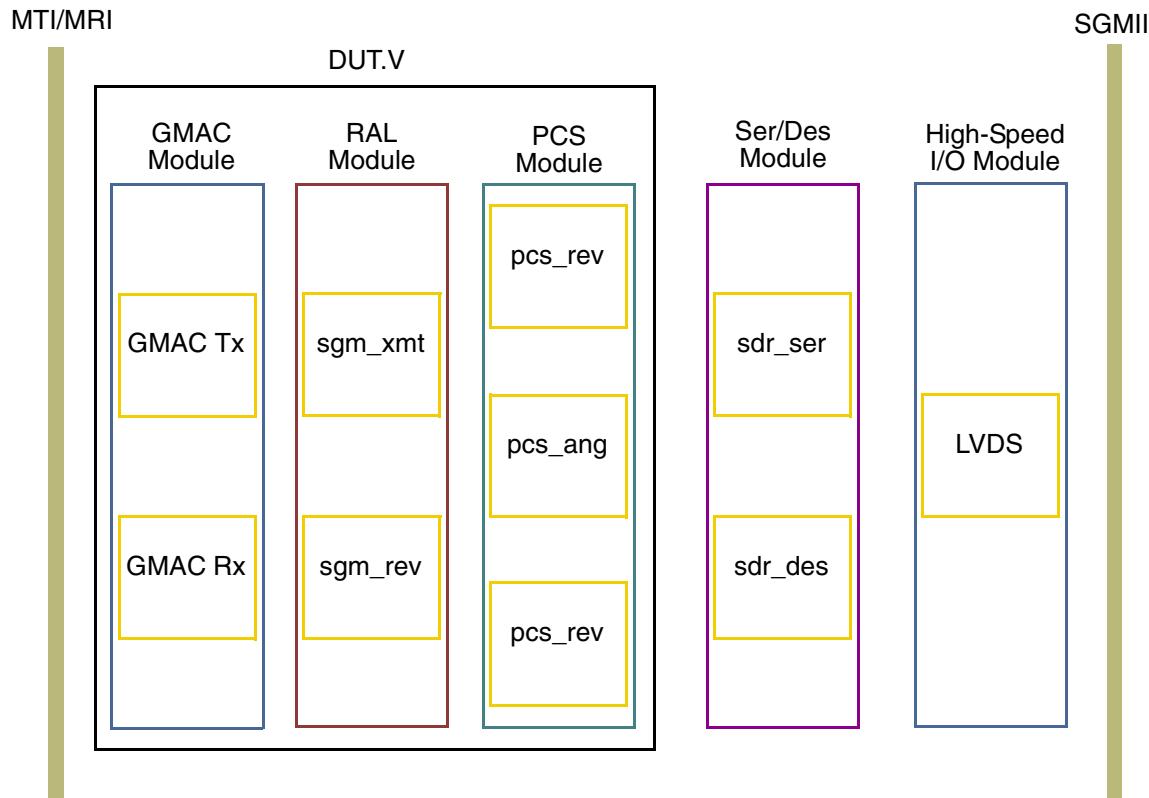
### 3.16.3.4 High-Speed I/O

The I/O block used in the SGMII is an LVDS type I/O. (Refer to the Serial GMII Specification, Revision 1.7, for additional information.)



The provided RTL is supplied/used for simulation purposes only. It should not be used for synthesis. The SerDes and LVDS I/O blocks are custom blocks of technology-specific library parts. See [Figure 3-95](#) for a block diagram of the GMAC core (DUT) with support for an SGMII interface.

**Figure 3-95 GMAC Core (DUT) With Support for an SGMII Interface**



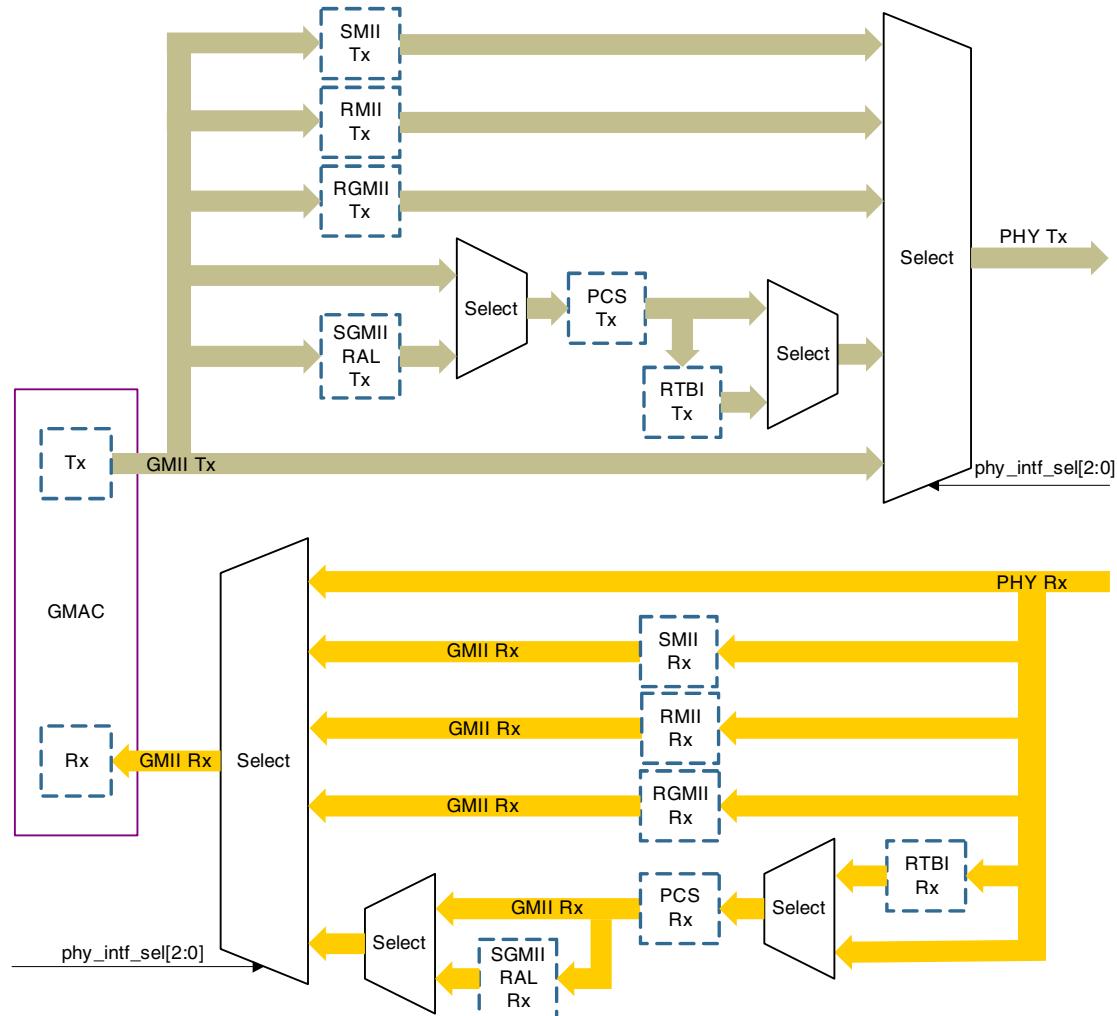
### 3.17 Multiple PHY Interfaces

**Figure 3-96** illustrates the multiplexing logic implemented to support multiple programmable PHY interfaces.



When the core is configured for a single PHY interface, all MUX logic is removed and the corresponding PHY interface is connected directly to the ports.

**Figure 3-96 Multiplexing Logic for Multiple Programmable PHY Interfaces**



### 3.18 Interrupts From the GMAC Core

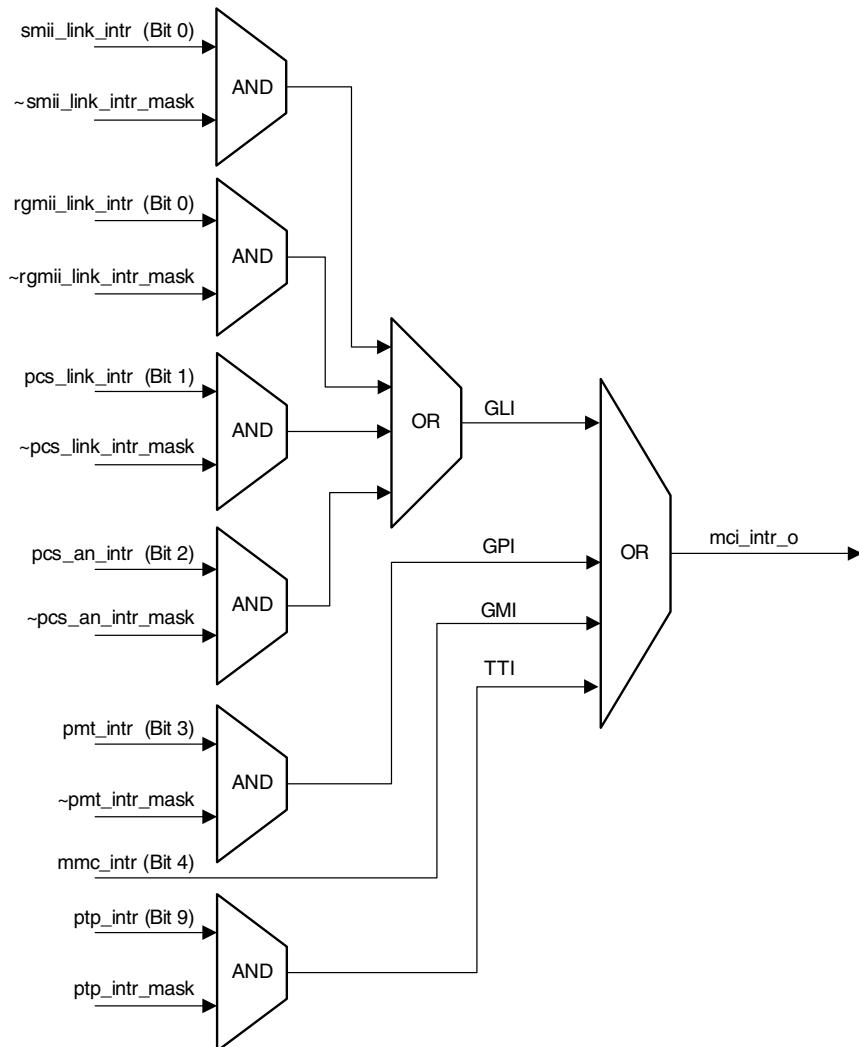
Interrupts can be generated from the GMAC core as a result of various events in the optional modules in it. mci\_intr\_o is the interrupt signal for the GMAC-CORE and GMAC-MTL configurations, while in other configurations (GMAC-DMA, GMAC-AHB), these interrupt events are combined with the events in the DMA on the sbd\_intr\_o signal.

The Interrupt Status register in the GMAC Register Map ([Table 5-2](#)) describes the events that can cause an interrupt from the GMAC core. Each event can be prevented from asserting the interrupt on the mci\_intr\_o or sbd\_intr\_o signals by setting the corresponding mask bits in the Interrupt Mask register.

The interrupt register bits only indicate the block from which the event is reported. You must read the corresponding status registers and other registers to clear the interrupt. For example, Bit 0 of the Interrupt register, set high, indicates that the link status on the RGMII/SMII interface has changed. You must read Register 54 (the RGMII/SMII Status register) to clear this interrupt event.

The interrupts from the RGMII/SMII and PCS blocks are combined (OR'ed) and given as the GLI bit in the DMA Status register. The TTI, GPI, and GMI signals in [Figure 3-97](#) refer to the bits that can be read in the DMA Status register.

**Figure 3-97 GMAC Core Interrupt Masking Scheme**



## 4

# Signal Descriptions

## 4.1 Top-Level I/O Diagram

[Figure 4-1](#) shows the top-level ports of the GMAC core only, with native interface (GMAC-CORE). [Figure 4-2](#) shows the top-level ports of the GMAC-MTL. [Figure 4-3](#) shows the top-level ports of the GMAC-DMA configuration. [Figure 4-4](#) shows the top-level ports of the GMAC with AHB interface (GMAC-AHB). [Figure 4-5](#) shows the top-level ports of the GMAC with AXI interface (GMAC-AXI). [Figure 4-6](#) shows the top-level ports of the optional interfaces. The signals are described in the tables that follow.

The signal interface group names in the figures are cross-referenced to the signal descriptions in [Tables 4-1](#) to [4-26](#). (For example, click on “PHY Interface Signals” in [Figure 4-1](#) to hyperlink to [Table 4-2](#) for a description of the PHY interface signals.

## 4.2 Signal Descriptions

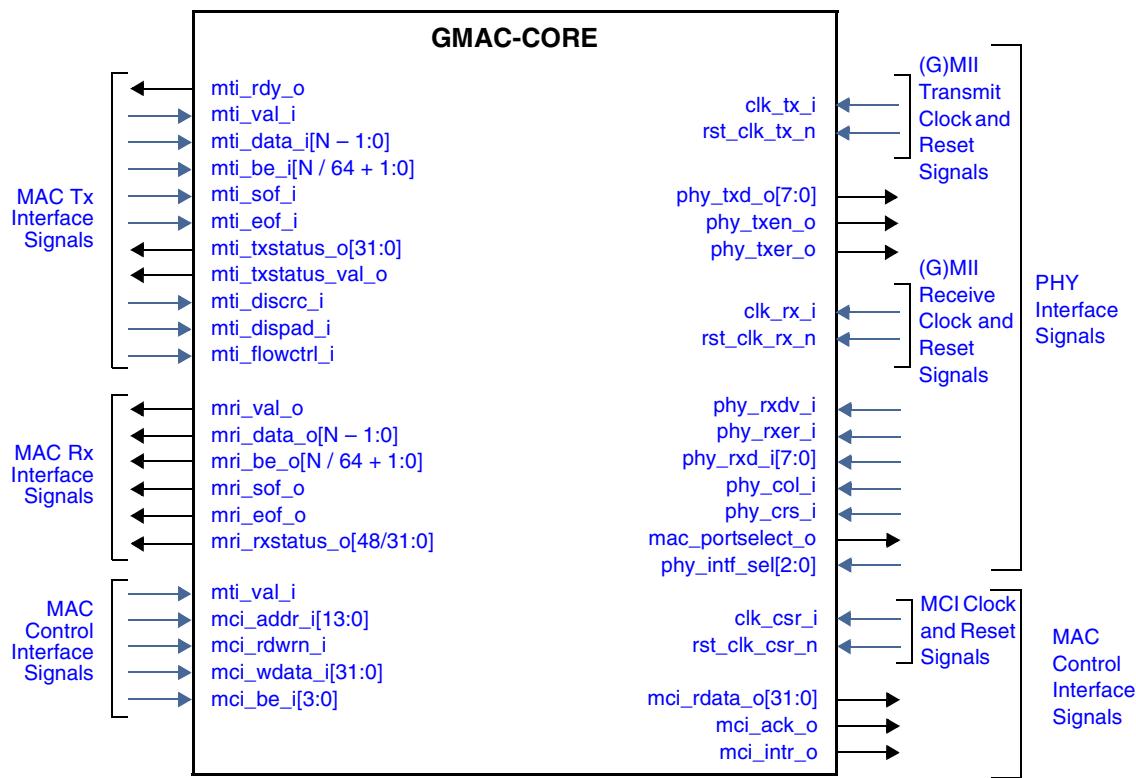
In addition to describing the function of each signal, the signal descriptions in this chapter include the following information:

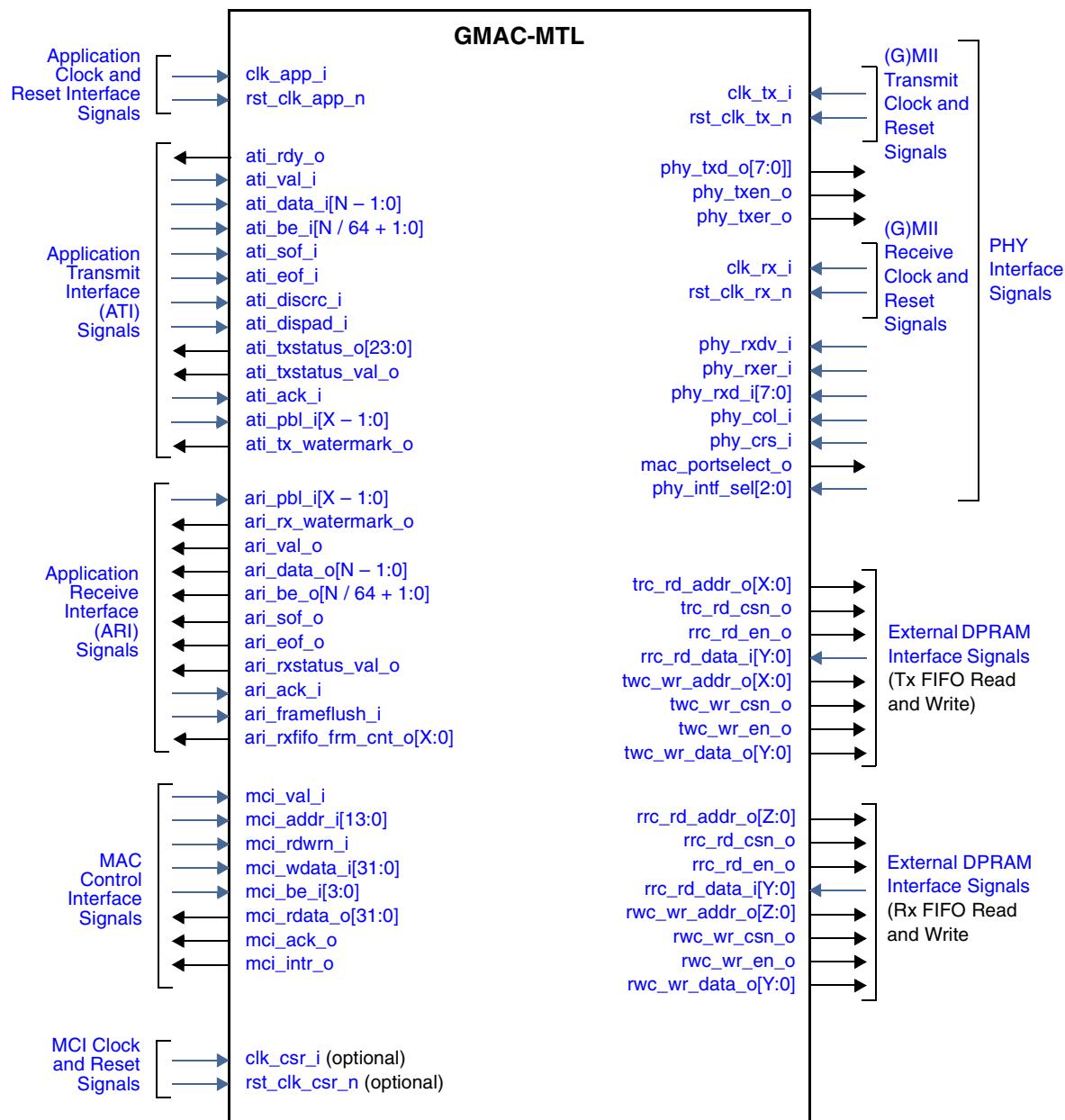
**Registered:** Indicates whether or not the signal is registered directly at the software-core boundary. A “No” value does not mean the signal is not synchronous, only that there is some combinatorial logic between the signal’s origin or destination register and the software-core boundary. A value of “Optionally” means that the signal is not registered at the core boundary by default, but you can choose to insert an input/output register by selecting the corresponding configuration option.

**Synchronous to:** Indicates to which clock (clk\_csr\_i or clk\_tx\_i) the signal is synchronous, or if the signal is asynchronous.

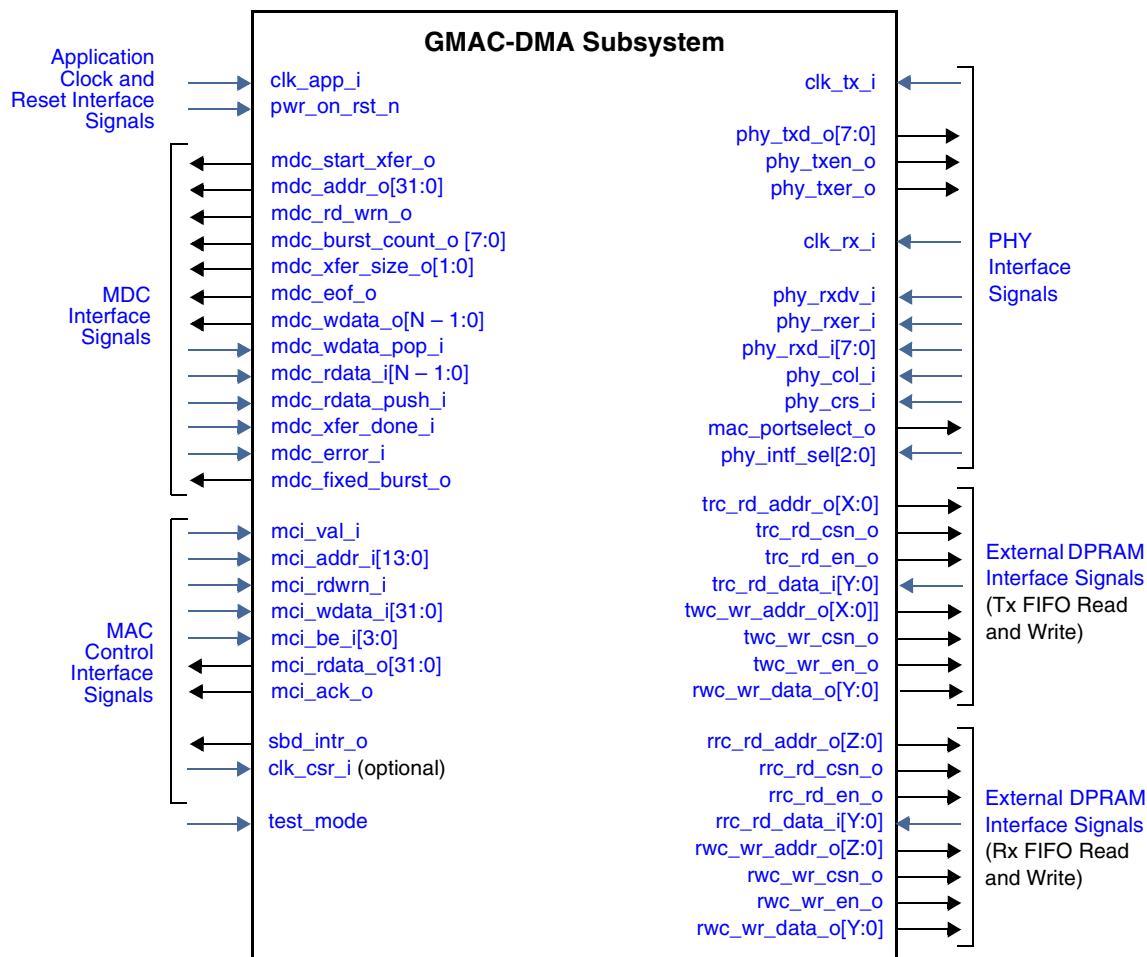


- All signals with “\_n” are active low. For example, signal hreset\_n is an active low signal; that is, asserted low.
- I/O indicates whether the signal is an input or an output. A signal with “\_i” is an input. A signal with “\_o” is an output. For example, signal clk\_tx\_i is an input.

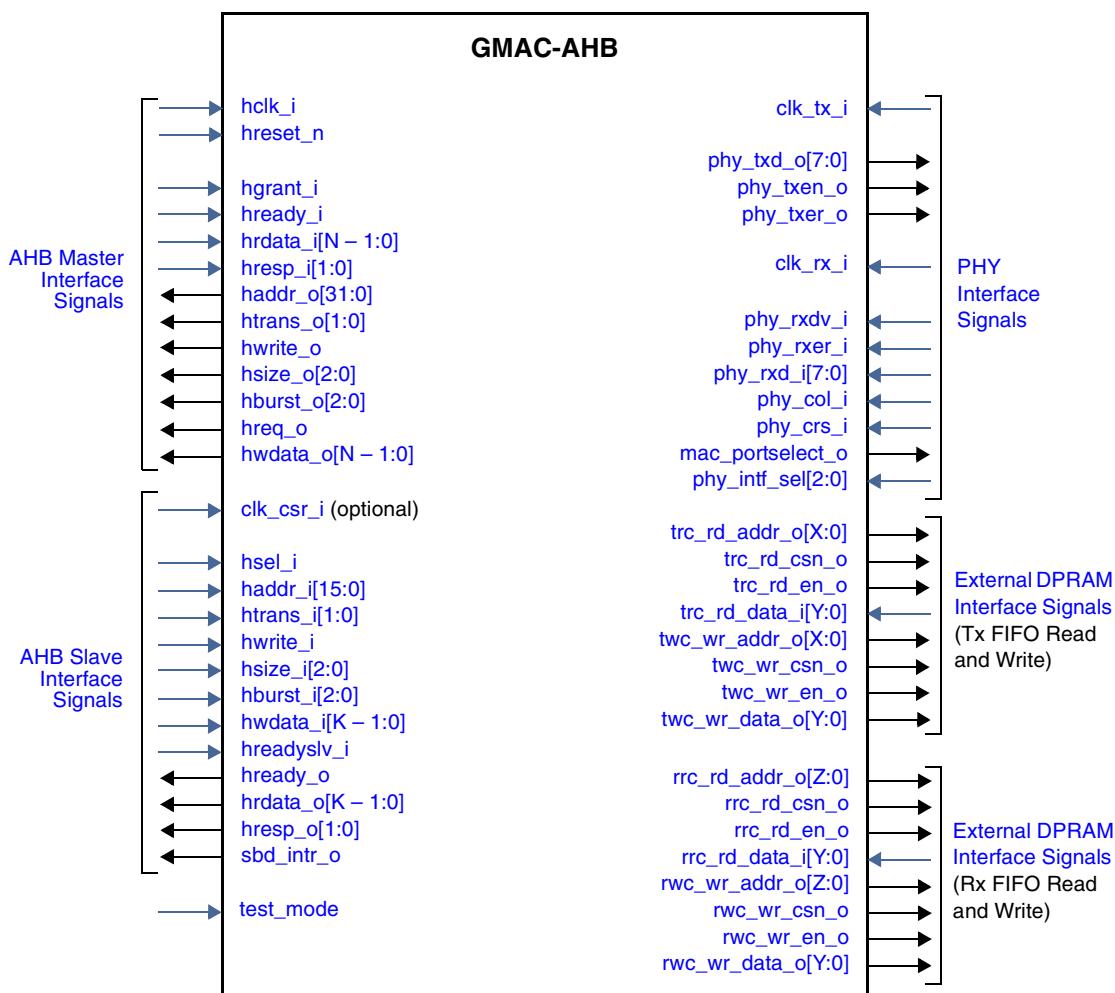
**Figure 4-1 GMAC-CORE Interface Top-Level I/O Diagram**

**Figure 4-2 GMAC-MTL Interface Top-Level I/O Diagram**

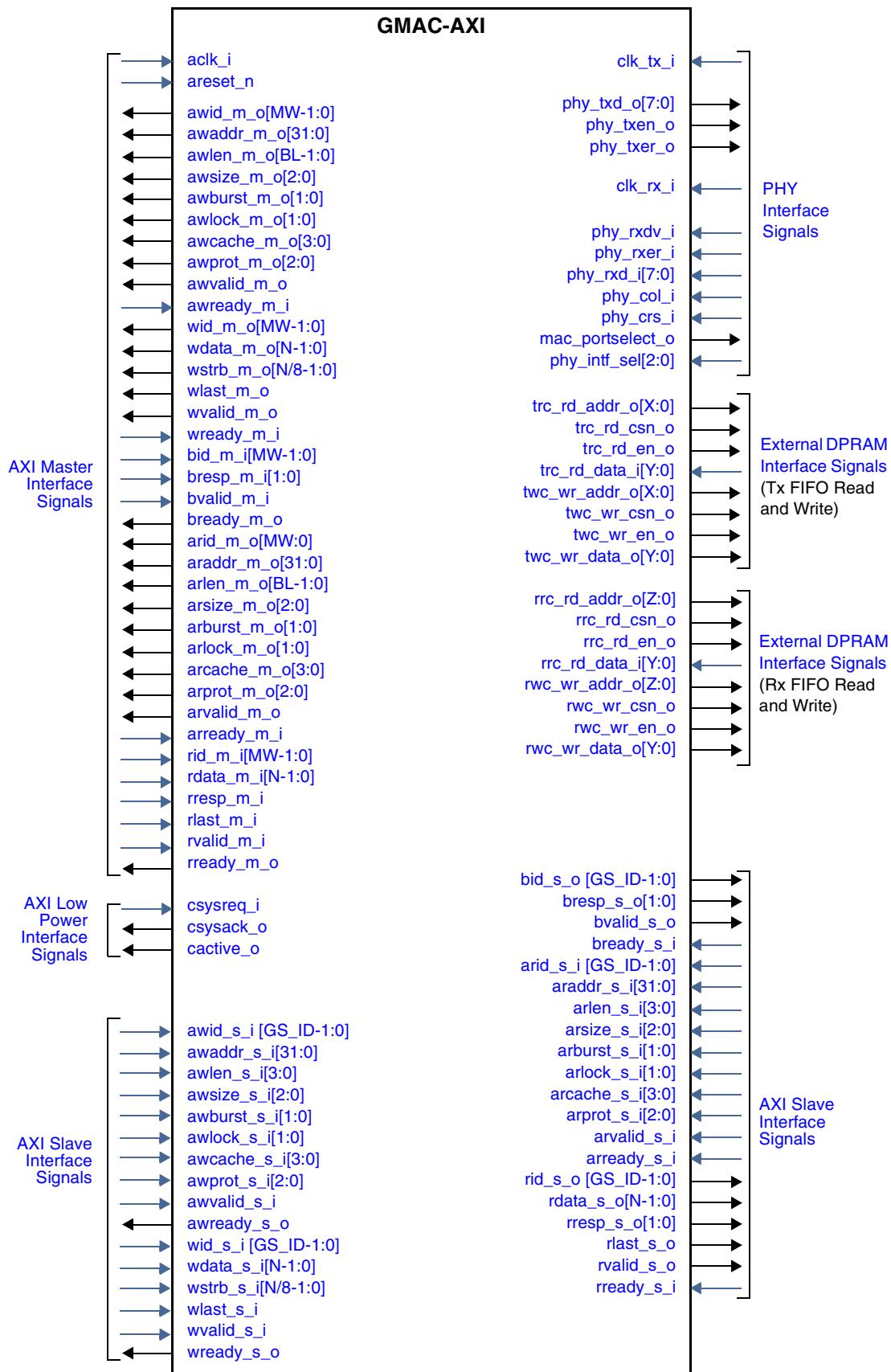
The address and data width of the memory interface (External DPRAM) depends on the configuration (FIFO depth and data width).

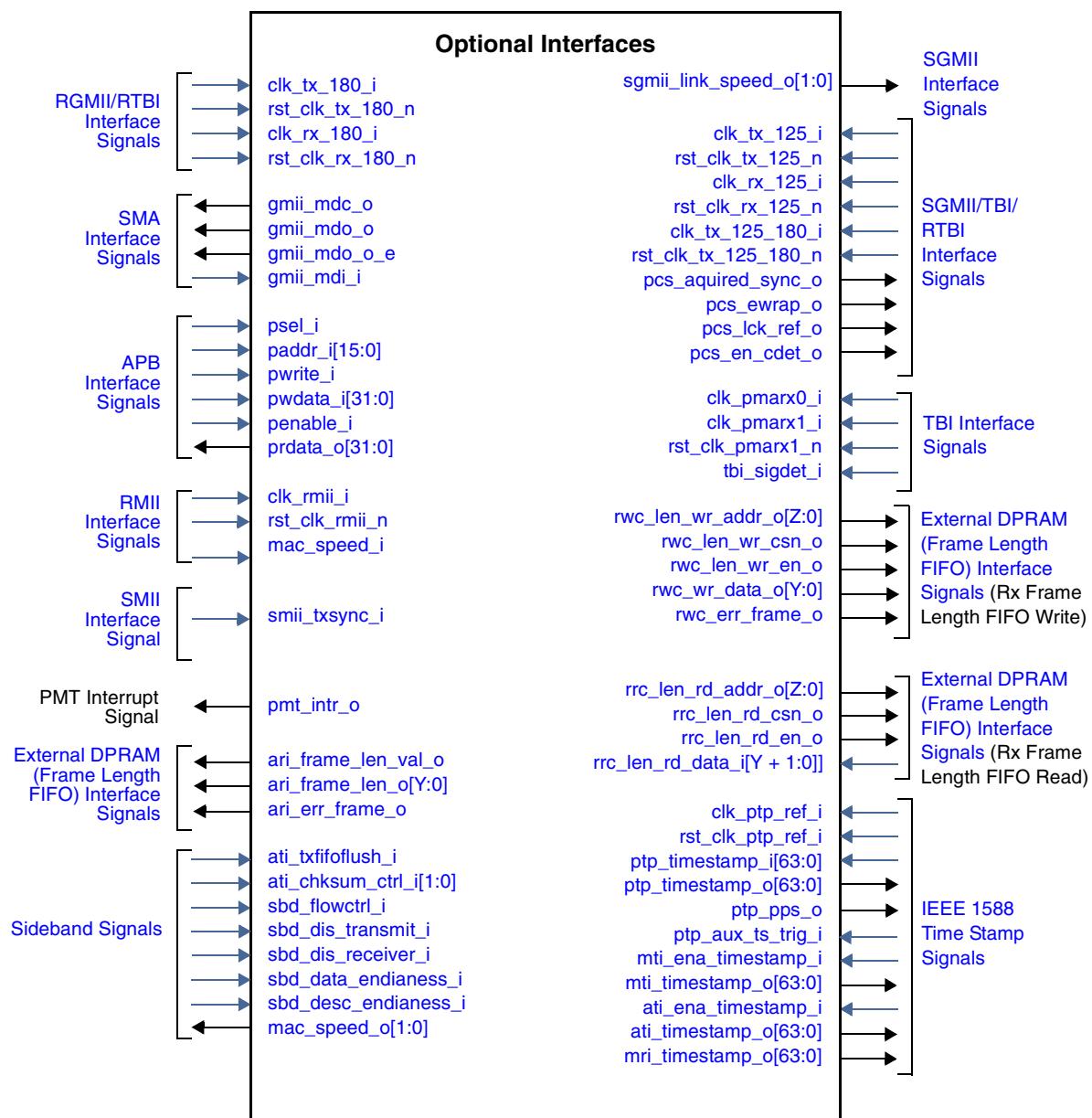
**Figure 4-3 GMAC-DMA (GMII) Configuration Top-Level I/O Diagram**

The address and data width of the memory interface (External DPRAM) depends on the configuration (FIFO depth and data width).

**Figure 4-4 GMAC-AHB (GMII) Top-Level I/O Diagram**

The address and data width of the memory interface (External DPRAM) depends on the configuration (FIFO depth and data width).

**Figure 4-5 GMAC-AXI (GMII) Top-Level I/O Diagram**

**Figure 4-6 Optional Interfaces Top-Level I/O Diagram**

All reset signals (rst\_\*) in Figure 4-6 are not required for GMAC-DMA or GMAC-AHB configurations.

### 4.2.1 System Clock and Reset Signals (Default)

The system clock and reset signals (default) are described in [Table 4-1](#).

**Table 4-1 System Clock and Reset Signals**

Signal	I/O	Description
<b>(G)MII Transmit Clock and Reset Signals</b>		
clk_tx_i	In	<p>Transmission Clock</p> <p><b>Function:</b> This is the transmission clock (125/25/2.5 MHz in 1G/100M/10Mbps) provided by the external PHY/oscillator for the RGMII/GMII/MII. All the RGMII/GMII/MII transmission signals generated by the GMAC-UNIV are synchronous to this clock. Even when any of the RMII/SGMII/TBI/RTBI interfaces are selected, this clock input is required.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> N/A</p>
<b>(G)MII Receive Clock and Reset Signals</b>		
clk_rx_i	In	<p>Reception Clock</p> <p><b>Function:</b> This is the reception clock (125/25/2.5 MHz in 1G/100M/10Mbps) provided by the external PHY for RGMII/GMII/MII and RMII. All the RGMII/GMII/MII receive signals received by the GMAC-UNIV are synchronous to this clock. Even when any of the RMII/SGMII/TBI/RTBI interfaces are selected, this clock input is required.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> N/A</p>
rst_clk_tx_n	In	<p>Transmit Clock Reset</p> <p><b>Function:</b> Reset signal. This input is not present in the GMAC-AHB and GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>
rst_clk_rx_n	In	<p>Receive Clock Reset</p> <p><b>Function:</b> Reset signal. This input is not present in GMAC-AHB and GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_rx_i</p>

**Table 4-1 System Clock and Reset Signals (Continued)**

Signal	I/O	Description
<b>MCI Clock and Reset Signals</b>		
clk_csr_i	In	<p>CSR Clock</p> <p><b>Function:</b> This is the free-running Application clock input provided by the Application. The MAC Control Interface, CSR, and Station Management Agent of the GMAC run on this clock. All signals generated by the CSR module are synchronous to clk_csr_i. The valid frequency range of this clock is 20–300 MHz (minimum frequency is 25 MHz when the MMC module is active in 1000-Mbps mode). Frequencies outside this range may result in incorrect operation. By default, this clock port is internally tied to hclk_i in the GMAC-AHB configuration, and tied to clk_app_i in the GMAC-MTL and GMAC-DMA configurations. If you select a separate clock for the CSR port in the GMAC-AHB, GMAC-DMA or GMAC-MTL configurations, then this input port is present. This clock input is required in the GMAC-CORE configuration.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> N/A</p>
rst_clk_csr_n	In	<p>CSR Clock Reset</p> <p><b>Function:</b> This is an active low reset signal. All registers and counters inside the CSR go to their default state when rst_clk_csr_n is asserted.</p> <p>This input is required in the GMAC-CORE configuration by default. It is an optional input for the GMAC-MTL configuration and is not present in the GMAC-AHB or GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_csr_i</p>

## 4.2.2 PHY Interface Signals (Default)

The default PHY Interface signals are described in [Table 4-2](#).

**Table 4-2** PHY Interface Signals

Signal	I/O	Description
phy_intf_sel[2:0]	In	<p>PHY Interface Select</p> <p><b>Function:</b> These pins select one of the GMAC's multiple PHY interfaces. This is sampled only during reset assertion (rst_clk_csr_n), and ignored after that. This input is removed from the port list when the core is configured for a single PHY interface.</p> <ul style="list-style-type: none"> <li>• 000: GMII/MII</li> <li>• 001: RGMII</li> <li>• 010: SGMII</li> <li>• 011: TBI</li> <li>• 100: RMII</li> <li>• 101: RTBI</li> <li>• 110: SMII</li> <li>• All others: Reserved</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_csr_i</p>
mac_portselect_o	Out	<p>MAC Port Select</p> <p><b>Function:</b> This signal mirrors the PS bit of the GMAC Configuration Register. It is low after reset and is used by the external clock circuitry to generate the GMII/MII clocks. A high indicates an MII interface, and a low a GMII interface.</p> <p>This output signal is removed from the port list when the optional mac_speed_o signal output is enabled during coreKit configuration.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_csr_i</p>

**Table 4-2** PHY Interface Signals (Continued)

Signal	I/O	Description
phy_txd_o[7:0]	Out	<p>PHY Transmit Data</p> <p><b>Function:</b> This is a bundle of eight transmit data signals driven by the GMAC. It has multiple functions depending on which PHY interface is selected, as given below. Unused bits in the RGMII/RTBI/RMII/MII interface configurations are tied to low.</p> <ul style="list-style-type: none"> <li>• GMII: All 8 bits provide the GMII transmit data byte. The validity of the data is qualified with phy_txen_o and phy_txer_o. <b>Synchronous to:</b> clk_tx_i</li> <li>• MII: Bits [3:0] provide the MII transmit data nibble. The validity of the data is qualified with phy_txen_o and phy_txer_o. <b>Synchronous to:</b> clk_tx_i</li> <li>• RGMII: Bits [3:0] provide the RGMII transmit data. The data bus changes with both rising and falling edges of the transmit clock (clk_tx_i). The validity of the data is qualified with phy_txen_o. <b>Synchronous to:</b> clk_tx_i, clk_tx_180_i</li> <li>• RMII: Bits [1:0] provide the RMII transmit data. The validity of the data is qualified with phy_txen_o. <b>Synchronous to:</b> clk_rmii_i</li> <li>• SMII: Bit[0] provides the SMII transmit data. <b>Synchronous to:</b> clk_tx_125_i</li> <li>• SGMII/TBI: The 8 bits correspond to the transmit code group [7:0]. <b>Synchronous to:</b> clk_tx_125_i</li> <li>• RTBI: Bits[3:0] provide the 8 bits (bits[3:0] and bits[8:5]) of the RTBI 10-bit transmit code group. The data bus changes with both edges of the transmit clock. <b>Synchronous to:</b> clk_tx_125_i and clk_tx_125_180_i</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> This signal is a registered output only when a single PHY interface (except RGMII or RTBI) is selected. When MUX logic is present, it is unregistered.</p> <p><b>Synchronous to:</b> Depends on the selected PHY interface.</p> <p><b>Note 1:</b> If you have configured the core for 10/100 Mbps-only operation, phy_txd_o is only 4 bits wide.</p> <p><b>Note 2:</b> if you have configured the core for Single PHY interface, then the width of the signal corresponds to the specified width. i.e 4, 2, 1 for RGMII/RMII/SMII respectively</p>

**Table 4-2 PHY Interface Signals (Continued)**

Signal	I/O	Description
phy_txen_o	Out	<p>PHY Transmit Data Enable</p> <p><b>Function:</b> This signal is driven by the GMAC and has multiple functions depending on which PHY interface is selected, as given below.</p> <ul style="list-style-type: none"> <li>• GMII/MII: When high, indicates that valid data is being transmitted on the phy_txd bus. <b>Synchronous to:</b> clk_tx_i</li> <li>• RMII: When high, indicates that valid data is being transmitted on the phy_txd bus. <b>Synchronous to:</b> clk_rmii_i</li> <li>• SMII: This is the Global synchronization signal for SMII. This signal is not present/valid when global sync signal is an input in SSSMII configuration. <b>Synchronous to:</b> clk_tx_125_i</li> <li>• RGMII: This signal is the control signal (rgmii_tctl) for the transmit data, and is driven on both edges of the clock. <b>Synchronous to:</b> clk_tx_i, clk_tx_180_i</li> <li>• SGMII/TBI: Contains Bit 8 of the transmit code group. <b>Synchronous to:</b> clk_tx_125_i</li> <li>• RTBI: This signal contains 2 bits (Bit[4] and Bit[9]) of the 10-bit RTBI transmit code-group and is driven on both edges of the clock. <b>Synchronous to:</b> clk_tx_125_i, clk_tx_125_180_i.</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> This signal is a registered output only when a single PHY interface (except RGMII or RTBI) is selected. When MUX logic is present, it is unregistered.</p> <p><b>Synchronous to:</b> Depends on the selected PHY interface.</p>
phy_txer_o	Out	<p>PHY Transmit Error</p> <p><b>Function:</b> This signal is driven by the GMAC and has multiple functions depending on which PHY interface is selected, as given below.</p> <ul style="list-style-type: none"> <li>• GMII: When high, indicates a transmit error or carrier extension on the phy_txd bus. <b>Synchronous to:</b> clk_tx_i</li> <li>• MII, RMII, RGMII, SMII, or RTBI: Not used. Tied low in some configurations; driven low in others.</li> <li>• SGMII/TBI: Contains Bit 9 of the transmit code group. <b>Synchronous to:</b> clk_tx_125_i</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> This signal is a registered output only when a single PHY interface is selected. When MUX logic is present, it is unregistered.</p> <p><b>Synchronous to:</b> Depends on the selected PHY interface.</p> <p><b>Note:</b> If you have configured the core for Single PHY interface, then this output is present only for the required interface. i.e only in GMII/SGMII/TBI.</p>

**Table 4-2** PHY Interface Signals (Continued)

Signal	I/O	Description
phy_rxd_i[7:0]	In	<p>PHY Receive Data</p> <p><b>Function:</b> This is a bundle of eight data signals received from the PHY. It has multiple functions depending on which PHY interface is selected, as given below.</p> <ul style="list-style-type: none"> <li>• GMII: All 8 bits provide the GMII receive data byte. The validity of the data is qualified with phy_rxrv_i and phy_rxer_i. <b>Synchronous to:</b> clk_rx_i</li> <li>• MII: Bits [3:0] provide the MII receive data nibble. The validity of the data is qualified with phy_rxrv_i and phy_rxer_i. <b>Synchronous to:</b> clk_rx_i</li> <li>• RGMII: Bits [3:0] provide the RGMII receive data. The data bus is sampled with both rising and falling edges of the receive clock (clk_rx_i). The validity of the data is qualified with phy_rxrv_i. <b>Synchronous to:</b> clk_rx_i, clk_rx_180_i</li> <li>• RMII: Bits [1:0] provide the RMII receive data. The validity of the data is qualified with phy_rxrv_o. <b>Synchronous to:</b> clk_rmii_i</li> <li>• SMII: Bit[0] provides the SMII receive data. <b>Synchronous to:</b> clk_rx_125_i</li> <li>• SGMII: The 8 bits correspond to the receive code group [7:0]. Synchronous to: clk_rx_125_i</li> <li>• TBI: The 8 bits correspond to the receive code group [7:0]. <b>Synchronous to:</b> clk_pmarx0_i, clk_pmarx1_i</li> <li>• RTBI: Bits [3:0] provides the 8 bits (Bits[3:0] and Bits[8:5]) of the 10-bit receive code-group and is sampled at both edges of the clock. <b>Synchronous to:</b> clk_rx_125_i and clk_rx_180_i</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes, except when RTBI is selected along with TBI/SGMII</p> <p><b>Synchronous to:</b> Depends on the selected PHY interface.</p> <p><b>Note 1:</b> If you have configured the core for 10/100 Mbps-only operation, phy_rxd_i is only 4 bits wide.</p> <p><b>Note 2:</b> If you have configured the core for Single PHY interface, then the width of the signal corresponds to the specified width. i.e 4, 2, 1 for RGMII/RMII/SMII respectively.</p>

**Table 4-2 PHY Interface Signals (Continued)**

Signal	I/O	Description
phy_rxrv_i	In	<p>PHY Receive Data Valid</p> <p><b>Function:</b> This signal is driven by PHY and has multiple functions depending on which PHY interface is selected as given below.</p> <ul style="list-style-type: none"> <li>GMII/MII: When high, indicates that data on the phy_rxd bus is valid. It remains asserted continuously from the first recovered byte/nibble of the frame through the final recovered byte/nibble. <b>Synchronous to:</b> clk_rx_i</li> <li>RGMII: This is the receive control signal used to qualify the data received on phy_rxd. This signal is sampled on both edges of the clock. <b>Synchronous to:</b> clk_rx_i, clk_rx_180_i</li> <li>RMII: Contains the crs and data valid information of the receive interface. <b>Synchronous to:</b> clk_rmii_i</li> <li>SMII: This is the Receive Global synchronization signal input for SSSMII. This signal is not present/valid when SMII is not configured for SSSMII. <b>Synchronous to:</b> clk_rx_125_i</li> <li>SGMII: Contains Bit 8 of the receive code group. <b>Synchronous to:</b> clk_rx_125_i</li> <li>TBI: Contains Bit 8 of the receive code group. <b>Synchronous to:</b> clk_pmarx0_i, clk_pmarx1_i</li> <li>RTBI: This contains the 2 bits (Bit[4] and Bit[9]) of the 10-bit receive code-group and is sampled at both edges of the clock. <b>Synchronous to:</b> clk_rx_125_i and clk_rx_180_i</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes, except when RTBI is selected along with TBI/SGMII</p> <p><b>Synchronous to:</b> Depends on the selected PHY interface.</p>
phy_rxer_i	In	<p>PHY Receive Error</p> <p><b>Function:</b> This signal is driven by the PHY and has multiple functions depending on which PHY interface is selected as given below.</p> <ul style="list-style-type: none"> <li>GMII/MII: When high, indicates an error or carrier extension (in GMII) in the received frame on the phy_rxd bus. <b>Synchronous to:</b> clk_rx_i</li> <li>RGMII/RMII/SMII/RTBI: Not used.</li> <li>SGMII: Contains Bit 9 of the receive code group. <b>Synchronous to:</b> clk_rx_125_i</li> <li>TBI: Contains Bit 9 of the receive code group. <b>Synchronous to:</b> clk_pmarx0_i, clk_pmarx1_i</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> Depends on the selected PHY interface.</p> <p><b>Note:</b> If you have configured the core for Single PHY interface, then this input is present only for the required interface. i.e only in GMII/MII/SGMII/TBI.</p>

**Table 4-2** PHY Interface Signals (Continued)

Signal	I/O	Description
phy_crs_i	In	<p>PHY CRS</p> <p><b>Function:</b> This signal, valid only in GMII/MII mode, is asserted by the PHY when either the transmit or receive medium is not idle. The PHY deasserts this signal when both transmit and receive medium are idle. This signal is not synchronous to any clock.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> N/A</p> <p><b>Note:</b> If you have configured the core for Single PHY interface, then this input is present only for GMII or MII and not present in other PHY interfaces.</p>
phy_col_i	In	<p>PHY Collision</p> <p><b>Function:</b> This signal, valid only in GMII/MII mode, is asserted by the PHY when a collision is detected on the medium. This signal is not synchronous to any clock.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> N/A</p> <p><b>Note:</b> If you have configured the core for Single PHY interface, then this input is present only for GMII or MII and not present in other PHY interfaces.</p>

#### 4.2.3 MAC Tx Interface Signals

The MAC Tx Interface (MTI) signals are described in [Table 4-3](#).

**Table 4-3** MAC Tx Interface Signals

Signal	I/O	Description
mti_rdy_o	Out	<p>Transmit Data Ready</p> <p><b>Function:</b> When high, indicates that the GMAC-CORE is ready to accept data input to it on mti_data_i bus.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>
mti_val_i	In	<p>Transmit Data Valid</p> <p><b>Function:</b> This signal is asserted by the Application to indicate to the GMAC-CORE that data is available for transmission on mti_data_i bus. This signal also qualifies the validity of mti_be_i, mti_sof_i, mti_eof_i, mti_discrc_i and mti_dispad_i signals.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>

**Table 4-3 MAC Tx Interface Signals (Continued)**

Signal	I/O	Description
mti_data_i[N – 1:0]	In	<p>Transmit Data Bus</p> <p><b>Function:</b> The data for transmission is input to the core on this bus. The width of the bus can be configured to (N=) 32/64/128. The GMAC core can be configured to process the data in little-endian or big-endian format</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_tx_i</p>
mti_be_i[N / 64 + 1:0]	In	<p>Transmit Byte Lanes</p> <p><b>Function:</b> This signal indicates the number of byte lanes valid on mti_data_i when mti_eof_i is high. The width of this bus depends on the data bus width (N). If N = 32, the width is 2, for N = 64, the width is 3, for N = 128, the width is 4. When mti_eof_i is low, the GMAC Core always considers all byte lanes to have valid data.</p> <p>When the width is 2, a value of</p> <ul style="list-style-type: none"> <li>• 11: all 4 byte lanes have data</li> <li>• 10: LS 3 byte lanes have data</li> <li>• 01: LS 2 byte lanes have data</li> <li>• 00: LS 1 byte lane have data</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_tx_i</p>
mti_sof_i	In	<p>Transmit Start of Frame</p> <p><b>Function:</b> This signal indicates that the start of an Ethernet frame is being transferred to the GMAC core.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>
mti_eof_i	In	<p>Transmit End of Frame</p> <p><b>Function:</b> This signal indicates that the end of an Ethernet frame is being transferred to the GMAC core.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>
mti_txstatus_o[31:0]	Out	<p>Transmit Status</p> <p><b>Function:</b> This bus gives the status of transmission of the last Ethernet frame input to the GMAC core on mti_data_i.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_tx_i</p>

**Table 4-3 MAC Tx Interface Signals (Continued)**

Signal	I/O	Description
mti_txstatus_val_o	Out	<p>Transmit Status Valid</p> <p><b>Function:</b> This signal is set high for 1 clock cycle to indicate that the mti_txstatus_o gives valid status for the last transmitted frame</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>
mti_discrc_i	In	<p>Transmit Disable Calculation</p> <p><b>Function:</b> When set high, instructs the GMAC-CORE to disable the calculation and insertion of the FCS bytes into the frame being transmitted. By asserting this signal, the Application assures the GMAC that the proper CRC will be appended prior to sending. The GMAC will not append the CRC for any frames received from the Application with a byte count greater than or equal to 60.</p> <p>However, the GMAC appends both pads and CRC for all frames from the Application with a byte count less than 60 when mti_dispad_i is reset to low, even if mti_discrc_i is set high.</p> <p>This signal is sampled only when the mti_sof_i is high.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>
mti_dispad_i	In	<p>Transmit Disable Padding</p> <p><b>Function:</b> When set high, instructs the GMAC to disable the padding function. By asserting this signal, the Application assures the GMAC that proper padding will be appended prior to sending. The GMAC appends only the CRC for the transmitting Ethernet frame. This signal is sampled only when mti_sof_i is high</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>
mti_flowctrl_i	In	<p>Transmit Flow Control</p> <p><b>Function:</b> When set high, instructs the GMAC to transmit PAUSE Control frames in Full-duplex mode. In Half-duplex mode, the GMAC enables the Back-pressure function until this signal is made low again.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>

#### 4.2.4 MAC Rx Interface Signals

The MAC Rx Interface signals are described in [Table 4-4](#).

**Table 4-4 MAC Rx Interface Signals**

Signal	I/O	Description
mri_val_o	Out	<p>Receive Data Valid</p> <p><b>Function:</b> The GMAC-CORE asserts this signal to indicate to the Application (MTL) that received frame data is available on mri_data_o. This signal also qualifies mri_be_o, mri_sof_o and mri_eof_o.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>
mri_data_o[N – 1:0]	Out	<p>Receive Data Bus</p> <p><b>Function:</b> This signal carries the received data of the frame from the GMAC-CORE to the MTL. The width of the bus can be configured to (N=) 32/64/128. The validity of mri_data_o is qualified with mri_val_o.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>
mri_be_o[N / 64 + 1:0]	Out	<p>Receive Byte Lanes</p> <p><b>Function:</b> This signal indicates the number of byte lanes valid on mri_data_o when mri_eof_o is high. The width of this bus depends on the data bus width (N). When N = 32, the width is 2; for N = 64, the width is 3, for N = 128, the width is 4. When mri_eof_i is low, the GMAC core always drives all-ones on mri_be_o.</p> <ul style="list-style-type: none"> <li>• When mri_be_o width is 2, a value of</li> <li>• 11: all 4 byte lanes have data</li> <li>• 10: LS 3 byte lanes have data</li> <li>• 01: LS 2 byte lanes have data</li> <li>• 00: LS 1 byte lane have data</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>
mri_sof_o	Out	<p>Receive Start of Frame</p> <p><b>Function:</b> This signal indicates that the start of an Ethernet frame is being transferred from the GMAC core.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>
mri_eof_o	Out	<p>Receive End of Frame</p> <p><b>Function:</b> This signal indicates that the end of an Ethernet frame is being transferred from the GMAC core.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>

**Table 4-4 MAC Rx Interface Signals (Continued)**

Signal	I/O	Description
mri_rxstatus_o[48/31:0]	Out	<p>Receive Status</p> <p><b>Function:</b> This bus gives the status of the received Ethernet frame output from the GMAC on mri_data_i. The validity of this signal is qualified with mri_eof_o. The status is 48-bits when the advance time stamp feature is selected.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>

#### 4.2.5 MAC Control Interface Signals

The MAC Control Interface signals are described in [Table 4-5](#).

**Table 4-5 MAC Control Interface Signals**

Signal	I/O	Description
mci_val_i	In	<p>Data Valid</p> <p><b>Function:</b> The application asserts mci_val_i to access the GMAC's CSR module, indicating to the GMAC that the address, Read/Write and, in the case of a Write, that the data are all valid. The assertion of this signal is the start of transaction.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_csr_i</p>
mci_addr_i[13:0]	In	<p>Address</p> <p><b>Function:</b> This signal carries the register address of the CSR module from the Application. The address is valid only when mci_val_i is asserted.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_csr_i</p>
mci_rdwrn_i	In	<p>Read/Write</p> <p><b>Function:</b> This signal alerts the GMAC to a Read/Write operation. The Application writes the data into the GMAC Control registers by deasserting this signal. Asserting this signal indicates the Read operation.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_csr_i</p>
mci_wdata_i[31:0]	In	<p>Write Data</p> <p><b>Function:</b> This signal carries the Write data from the Application for the Control registers of the CSR module. The validity of each byte in mci_wdata_i[31:0] is qualified with the byte enables on mci_be_i[3:0].</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_csr_i</p>

**Table 4-5 MAC Control Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
mci_be_i[3:0]	In	Bytes Enabled <b>Function:</b> This signal indicates which bytes are valid on the mci_wdata_i[31:0] signal. This is valid when mci_val_i is asserted. <b>Active State:</b> N/A <b>Registered:</b> No <b>Synchronous to:</b> clk_csr_i
mci_ack_o	Out	Data Acknowledge <b>Function:</b> GMAC-issued acknowledgement for all data transfers. <b>Active State:</b> High <b>Registered:</b> Yes <b>Synchronous to:</b> clk_csr_i
mci_rdata_o[31:0]	Out	Read Data <b>Function:</b> This signal carries the Read data from the GMAC's Control registers to the Application. The GMAC will supply all the bytes irrespective of the byte enable signal. <b>Active State:</b> N/A <b>Registered:</b> Yes <b>Synchronous to:</b> clk_csr_i
mci_intr_o	Out	MCI Interrupt <b>Function:</b> When set high, indicates an interrupt event in the core's optional MMC, PCS, RGMII, or PMT module. This interrupt pin is present only GMAC-CORE and GMAC-MTL configurations. <b>Active State:</b> High <b>Registered:</b> No <b>Synchronous to:</b> clk_csr_i

#### 4.2.6 Application Clock and Reset Interface Signals

The Application Clock and Reset Interface signals are described in [Table 4-6](#).

**Table 4-6 Application Clock and Reset Interface Signals**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
clk_app_i	In	Application Clock <b>Function:</b> A free-running input clock from the application In the GMAC-DMA configuration, the DMA interface signals are synchronous to this clock. In GMAC-MTL configuration, the MTL application interface operates synchronous to this clock. The valid frequency range is 25 Mhz to 300 Mhz. The Transmit Write Control (TWC) and Receive Read Control (RRC) blocks operate with this clock. <b>Active State:</b> N/A <b>Registered:</b> N/A <b>Synchronous to:</b> N/A

**Table 4-6 Application Clock and Reset Interface Signals (Continued)**

Signal	I/O	Description
rst_clk_app_n	In	<p>Application Clock Reset</p> <p><b>Function:</b> This is the reset signal synchronous to the application clock domain in the GMAC-MTL configuration.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> clk_app_i</p>
pwr_on_rst_n	In	<p>Application Clock Reset</p> <p><b>Function:</b> This is the reset signal synchronous to the application clock domain in the GMAC-DMA configuration.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> clk_app_i</p>

#### 4.2.7 Application Transmit Interface (ATI) Signals

The Application Transmit Interface (ATI) signals are described in [Table 4-7](#).

**Table 4-7 Application Transmit Interface (ATI) Signals**

Signal	I/O	Description
ati_sof_i	In	<p>Start of frame</p> <p><b>Function:</b> Indicates that application is ready to transfer the first data of the frame. This signal is qualified by ati_val_i.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>
ati_discrc_i	In	<p>Disable CRC</p> <p><b>Function:</b> When asserted, disables the addition of FCS to the frame by the GMAC core. This signal value is sampled only when ati_sof_i is active. This signal is qualified by ati_val_o.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
ati_dispad_i	In	<p>Disable Pad</p> <p><b>Function:</b> When asserted disables the addition of pads onto the frame by the GMAC core. This signal value is sampled only when ati_sof_i is active. This signal is qualified by ati_val_i.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>

**Table 4-7 Application Transmit Interface (ATI) Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>								
ati_eof_i	In	<p>End of Frame</p> <p><b>Function:</b> Indicates that the current data transferred to then MTL is the last data phase. This signal is qualified by ati_val_i.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>								
ati_be_i[N / 64 + 1:0]	In	<p>Number of Valid Byte Lanes</p> <p><b>Function:</b> Indicates the number of valid byte lanes during the last data phase transfer to MTL.</p> <p>Application data width determines the ati_be_i signal width:</p> <table> <tr> <td>N</td> <td>(N / 64 + 1)</td> </tr> <tr> <td>32</td> <td>1</td> </tr> <tr> <td>64</td> <td>2</td> </tr> <tr> <td>128</td> <td>3</td> </tr> </table> <p>This signal is qualified by ati_val_i and ati_eof_i. When ati_eof_i is low, ati_be_i is taken as all-ones by the MTL.</p> <p>When ati_be_i width is 2, a value of</p> <ul style="list-style-type: none"> <li>• 11: all 4 byte lanes have data</li> <li>• 10: LS 3 byte lanes have data</li> <li>• 01: LS 2 byte lanes have data</li> <li>• 00: LS 1 byte lane have data</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>	N	(N / 64 + 1)	32	1	64	2	128	3
N	(N / 64 + 1)									
32	1									
64	2									
128	3									
ati_val_i	In	<p>ATI Valid</p> <p><b>Function:</b> This signal qualifies all the frame transmission control signals and data on application transmit interface</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>								
ati_data_i[N – 1:0]	In	<p>Application Data</p> <p><b>Function:</b> Frame data from the application. Width of the data bus can be configured to either 32/64/128-bit.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>								

**Table 4-7 Application Transmit Interface (ATI) Signals (Continued)**

Signal	I/O	Description
ati_rdy_o	Out	<p>ATI Ready</p> <p><b>Function:</b> When asserted, indicates that the MTL is ready to accept additional data from the application. This signal will be deasserted when Tx FIFO is being flushed or when Tx FIFO is full.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
ati_txstatus_val_o	Out	<p>Valid Transmit Status</p> <p><b>Function:</b> Signal qualifying the status word on ati_txstatus_o. This signal will be asserted when a valid status word is available in the status FIFO.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
ati_txstatus_o[23:0]	Out	<p>Transmit Status Word</p> <p><b>Function:</b> Transmit frame status word transferred to the application. The eight most significant bits are tied low in default mode. Bit 16 is required when a checksum offload engine is included in the core. Bit 17 is required when IEEE1588 time stamping is included.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
ati_ack_i	In	<p>ATI Acknowledgement</p> <p><b>Function:</b> Acknowledgement signal from the application indicating the transmit status word has been accepted.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>
ati_tx_watermark_o	Out	<p>Transmit Watermark</p> <p><b>Function:</b> When asserted, indicates that Tx FIFO in MTL has enough space to accommodate the number of data beats as indicated by ati_pbl_i.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
ati_pbl_i[X – 1:0]	In	<p>Number of Beats</p> <p><b>Function:</b> Indicates the space requested by the application in MTL (in terms of number of beats: 4, 8, or 16 bytes, respectively, in 32-, 64-, or 128-bit data bus mode) Transmit FIFO. The maximum limit of the number of beats is equal to half the Tx FIFO depth. Thus the width of this signal depends on the data bus width and Tx FIFO depth (X = 8 by default; Ranges from 3 to 10).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>

#### 4.2.8 Application Receive Interface (ARI) Signals

The Application Receive Interface (ARI) signals are described in [Table 4-8](#).

**Table 4-8 Application Receive Interface (ARI) Signals**

Signal	I/O	Description
ari_sof_o	Out	<p>Start of Frame</p> <p><b>Function:</b> Indicates that the current data transferred to the application corresponds to the first data phase of the frame.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
ari_eof_o	Out	<p>End of Frame</p> <p><b>Function:</b> Indicates that the current data transferred to the application corresponds to the last data phase of the frame.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>
ari_val_o	Out	<p>ARI Valid</p> <p><b>Function:</b> When asserted, qualifies the data and other control signals on MTL receive application interface.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
ari_data_o[N – 1:0]	Out	<p>Frame Data</p> <p><b>Function:</b> Frame data from the Rx FIFO to the application.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>

**Table 4-8 Application Receive Interface (ARI) Signals (Continued)**

Signal	I/O	Description								
ari_be_o[N / 64 + 1:0]	Out	<p>Number of Valid Byte Lanes</p> <p><b>Function:</b> Indicates the number of valid bytes lanes in the last data phase transfer on the MTL receive application interface.</p> <p>Application data width determines the ari_be_i signal width:</p> <table> <tr><td>N</td><td>(N / 64 + 1)</td></tr> <tr><td>32</td><td>1</td></tr> <tr><td>64</td><td>2</td></tr> <tr><td>128</td><td>3</td></tr> </table> <p>This signal is qualified by ari_val_i and ari_eof_i. When ari_eof_i is low, ari_be_i is always driven with all-ones by the MTL.</p> <p>When ari_be_o width is 2, a value of</p> <ul style="list-style-type: none"> <li>• 11: all 4 byte lanes have data</li> <li>• 10: LS 3 byte lanes have data</li> <li>• 01: LS 2 byte lanes have data</li> <li>• 00: LS 1 byte lane have data</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>	N	(N / 64 + 1)	32	1	64	2	128	3
N	(N / 64 + 1)									
32	1									
64	2									
128	3									
ari_ack_i	In	<p>ARI Acknowledgement</p> <p><b>Function:</b> Acknowledgement from the application indicating that the current data (ari_data_o) including Receive status has been accepted by the host.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>								
ari_rxstatus_val_o	Out	<p>Valid Receive Status</p> <p><b>Function:</b> Indicates that the ari_data_o bus contains the receive status to the application.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>								
ari_rx_watermark_o	Out	<p>Receive Watermark</p> <p><b>Function:</b> When asserted, indicates that the Rx FIFO in the MTL has enough data to generate the number of beats (4/8/16 bytes in 32/64/128 bit data bus mode) as requested through ari_pbl_i.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>								

**Table 4-8 Application Receive Interface (ARI) Signals (Continued)**

Signal	I/O	Description
ari_rxfifo_frm_cnt_o[X:0]	Out	Rx FIFO Frames Counter  <b>Function:</b> This gives the number of frames (with status) present in the Rx FIFO. The width of the counter depends on the maximum number of frames that can be stored in the Rx FIFO. For example, for an Rx FIFO depth of 2,048 bytes and a minimum receive frame size of 16 bytes, the maximum number of frames that can be stored in the Rx FIFO is 128. Thus X = 7 in this example.  <b>Active State:</b> N/A  <b>Registered:</b> Yes  <b>Synchronous to:</b> clk_app_i
ari_pbl_i[X – 1:0]	In	Number of Beats  <b>Function:</b> Indicates the number of beats of data requested by application from the MTL Receive FIFO. The maximum limit of the number of beats is equal to half the Rx FIFO depth. Thus the width of this signal depends on the data bus width and Rx FIFO depth (X = 8 by default; Ranges from 2 to 11).  <b>Active State:</b> N/A  <b>Registered:</b> No  <b>Synchronous to:</b> clk_app_i
ari_frameflush_i	In	Frame Flush Request  <b>Function:</b> Frame Flush request from the application to flush the current frame that is being transferred to the application.  <b>Active State:</b> High  <b>Registered:</b> No  <b>Synchronous to:</b> clk_app_i

#### 4.2.9 MDC Interface Signals

The MDC signals are described in [Table 4-9](#).

**Table 4-9 MDC Interface Signals**

Signal	I/O	Description
mdc_start_xfer_o	Out	Start Transfer  <b>Function:</b> This signal, when asserted (for 1 clock), indicates the start of a transaction by the DMA. This signal also indicates the validity of the values on mdc_rd_wrn, mdc_burst_count, mdc_xfer_size, and mdc_fixed_burst signal outputs.  <b>Active State:</b> High  <b>Registered:</b> Yes  <b>Synchronous to:</b> clk_app_i

**Table 4-9 MDC Interface Signals (Continued)**

Signal	I/O	Description
mdc_addr_o[31:0]	Out	<p>Address</p> <p><b>Function:</b> These bits give the address of the data transfer. The start address of a burst transfer is given along with the assertion of mdc_start_xfer_o, and the address is incremented at the end of each read/write cycle, indicated by the assertion of mdc_wdata_pop_i or mdc_rdata_push_is.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
mdc_rd_wrn_o	Out	<p>Read/Write</p> <p><b>Function:</b> When high, indicates a read operation command; when low, requests a write transfer.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
mdc_burst_count_o [7:0]	Out	<p>Burst Count</p> <p><b>Function:</b> These bits indicate the length of the burst transfer initiated by the DMA.</p> <ul style="list-style-type: none"> <li>• 0000_0000: 1 beat</li> <li>• 0000_0001: 2 beats</li> <li>...</li> <li>• 1111_1100: 255 beats</li> <li>• 1111_1111: 256 beats</li> </ul> <p>Note that each beat is 4, 8, or 16 bytes, respectively, for 32-, 64-, or 128-bit-wide buses.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
mdc_xfer_size_o[1:0]	Out	<p>Transfer Size</p> <p><b>Function:</b> This signal is a single bit by default, and 2 bits wide when the core is configured for 128-bit data bus width and IEEE1588 time stamping functions. In such configurations, the signal indicates that the DMA has initiated the following transfer sizes:</p> <ul style="list-style-type: none"> <li>• 2'b00 128-bit transfer</li> <li>• 2'b01 32-bit transfer</li> <li>• 2'b10 64-bit transfer</li> <li>• 2'b11 Reserved</li> </ul> <p>In single-bit configurations, this bit indicates a 32/64/128-bit transfer (same as data bus width) when low, and when high, indicates that a 32-bit transfer must be performed. This signal is used only for 32-bit descriptor write operations in a 64-/128-bit-wide bus configuration.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>

**Table 4-9 MDC Interface Signals (Continued)**

Signal	I/O	Description
mdc_wdata_o[N – 1:0]	Out	<p>Write Data</p> <p><b>Function:</b> This bus contains the data output for write transfers. The data on this bus will change at the clock cycle at which the mdc_wdata_pop_i is sampled high.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
mdc_wdata_pop_i	In	<p>Write Data Pop</p> <p><b>Function:</b> This signal is asserted to get the next valid data on the mdc_wdata_o bus in the next clock cycle. The DMA updates the data bus with next valid data when this signal is asserted.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>
mdc_rdata_i[N – 1:0]	In	<p>Read Data</p> <p><b>Function:</b> This bus contains the data read from the application. The DMA samples the read data bus whenever mdc_rdata_push_i is asserted.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>
mdc_rdata_push_i	In	<p>Read Data Valid</p> <p><b>Function:</b> This signal is asserted to indicate that the application has valid data on mdc_rdata_i bus. The DMA starts a burst read transfer only when it is capable of accepting the complete burst data. Hence the DMA will accept the read data whenever mdc_rdata_push_i is sampled as high.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>
mdc_xfer_done_i	In	<p>Transfer Done</p> <p><b>Function:</b> This signal, when asserted (for 1 clock), indicates that the data transfer started with the assertion of mdc_start_xfer_o is complete. This signal should be asserted 1 clock after the last read data is pushed in the case of read transfer. In the case of write transfer, this signal will be asserted after the last data is accepted by the slave on the bus.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>

**Table 4-9 MDC Interface Signals (Continued)**

Signal	I/O	Description
mdc_error_i	In	<p>Transfer Error</p> <p><b>Function:</b> This signal, when high (for 1 clock), indicates that an error occurred during application data transfer. The application must also terminate the transfer by asserting mdc_xfer_done_i in the next clock cycle.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>
mdc_eof_o	Out	<p>End of Frame</p> <p><b>Function:</b> The signal indicates that the current data on the mdc_wdata bus is an EOF data and the DMA will not be able to supply more data as per the requested burst length. The application must terminate the burst transfer on the receipt of this signal. The application must not assert mdc_wdata_pop_i after the transaction terminates.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
mdc_fixed_burst_o	Out	<p>Fixed Burst Length</p> <p><b>Function:</b> This signal reflects the value of the FB (fixed burst) bit of the DMA Bus Mode register.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>

#### 4.2.10 AHB Master Interface Signals

The AHB Master interface signals are described in [Table 4-10](#).

**Table 4-10 AHB Master Interface Signals**

Signal	I/O	Description
hclk_i	In	<p>AMBA AHB System Clock</p> <p><b>Function:</b> This is the free-running AHB clock input provided by the Application. The DMA and the MTL modules also operate with this clock.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> N/A</p>
hreset_n	In	<p>AMBA AHB Power On System Reset</p> <p><b>Function:</b> AMBA AHB power on system reset signal</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>

**Table 4-10 AHB Master Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
hreq_o	Out	<p>AHB Request</p> <p><b>Function:</b> This signal is set high by the GMAC-AHB subsystem to indicate that it requires the bus. This signal is processed by the arbiter to grant the bus.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i</p>
hgrant_i	In	<p>AHB Grant</p> <p><b>Function:</b> Indicates that the GMAC-AHB Subsystem is currently the highest priority Master. Ownership of address and control signals changes at the end of a transfer, when hready_i is high. Thus, the Subsystem gets access when both hgrant_i and hready_i are high.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>
haddr_o[31:0]	Out	<p>AHB Address</p> <p><b>Function:</b> AHB 32-bit address for the current transaction</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i</p>
htrans_o[1:0]	Out	<p>AHB Transfer Type</p> <p><b>Function:</b></p> <p>The AHB Master interface uses the following values</p> <ul style="list-style-type: none"> <li>• 00: IDLE</li> <li>• 10: NONSEQUENTIAL</li> <li>• 11: SEQUENTIAL</li> </ul> <p>All other encodings are not used.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i</p>
hsize_o[2:0]	Out	<p>AHB Data Transfer Size</p> <p><b>Function:</b> Typically values used are:</p> <ul style="list-style-type: none"> <li>• 010: Word (32 bits)</li> <li>• 011: DWord (64 bits)</li> <li>• 100: LWord (128 bits)</li> </ul> <p>All other encodings are not used.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i</p>

**Table 4-10 AHB Master Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
hwrite_o	Out	<p>AHB Read/Write Signal</p> <p><b>Function:</b> A high hwrite_o signal indicates a write transfer, a low indicates a read transfer.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i</p>
hburst_o[2:0]	Out	<p>AHB Burst</p> <p><b>Function:</b> Indicates the transfer part of an AHB Burst:</p> <ul style="list-style-type: none"> <li>• 000: SINGLE (Single Transfer)</li> <li>• 001: INCR (Incrementing burst of undefined length)</li> <li>• 011: INCR4 (4-beat incrementing burst)</li> <li>• 101: INCR8 (8-beat incrementing burst)</li> <li>• 111: INCR16 (16-beat incrementing burst)</li> </ul> <p>All other encodings are not used.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i</p>
hwdata_o[N – 1:0]	Out	<p>AHB Write Data Bus</p> <p><b>Function:</b> hwdata_o transfers data from the GMAC-AHB subsystem to the AHB Slaves. The width of the data bus is configurable to (N=) 128, 64, or 32 (default)</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i</p>
hready_i	In	<p>AHB Slave Ready</p> <p><b>Function:</b> hready_i indicates that a transfer has finished on the bus. It may be driven low by the Slave being addressed to extend a transfer.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>
hresp_i[1:0]	In	<p>AHB Slave Response</p> <p><b>Function:</b> hresp_i provides information on the transfer status:</p> <ul style="list-style-type: none"> <li>• 00: OKAY (Transfer completed OK)</li> <li>• 01: ERROR (Error in current transfer)</li> <li>• 10: RETRY (Slave is busy and wants Master to retry the transfer)</li> <li>• 11: SPLIT (Slave accepted request, but Master shall back off from bus until the Slave is ready to serve)</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>

**Table 4-10 AHB Master Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
hrdata_i[N – 1:0]	In	<p>AHB Slave Read Data</p> <p><b>Function:</b> hrdata_i transfers data from the AHB Slaves to the GMAC-AHB during read operations. The width of the data bus is configurable to (N =) 128, 64, or 32 (default).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i</p>

#### 4.2.11 AHB Slave Interface Signals

The AHB Slave interface signals are described in [Table 4-11](#).



All signals of this interface are synchronous to hclk\_i by default. They are synchronous to clk\_csr\_i when the optional clk\_csr\_i clock input is selected for the Slave interface.

**Table 4-11 AHB Slave Interface Signals**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
hsel_i	In	<p>AHB Slave Select</p> <p><b>Function:</b> hsel_i is a combinatorial decode of the address bus, indicating that the GMAC-AHB Subsystem is the target for the current transfer. All reads are completed without split response for this address region.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>
haddr_i[15:0]	In	<p>AHB Address</p> <p><b>Function:</b> AHB 16-bit address for the current transaction</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>
htrans_i[1:0]	In	<p>AHB Transfer</p> <p><b>Function:</b> Current AHB transfer type:</p> <ul style="list-style-type: none"> <li>• 00: IDLE</li> <li>• 01: BUSY</li> <li>• 10: NONSEQUENTIAL</li> <li>• 11: SEQUENTIAL</li> </ul> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>

**Table 4-11 AHB Slave Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
hsize_i[2:0]	In	<p>AHB Transfer Data Size</p> <p><b>Function:</b> Acceptable values are:</p> <ul style="list-style-type: none"> <li>• 000: Byte (8 bits)</li> <li>• 001: Halfword (16 bits)</li> <li>• 010: Word (32 bits)</li> </ul> <p>All other encodings are considered to be Word.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>
hwrite_i	In	<p>AHB Read/Write</p> <p><b>Function:</b> A high hwrite_i signal indicates a write transfer, a low indicates a read transfer</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>
hburst_i[2:0]	In	<p>AHB Burst Length</p> <p><b>Function:</b> Indicates the transfer part of an AHB burst:</p> <ul style="list-style-type: none"> <li>• 000: SINGLE (Single Transfer)</li> <li>• 001: INCR (Incrementing burst of undefined length)</li> <li>• 010: WRAP4 (4-beat wrapping burst)</li> <li>• 011: INCR4 (4-beat incrementing burst)</li> <li>• 100: WRAP8 (8-beat wrapping burst)</li> <li>• 101: INCR8 (8-beat incrementing burst)</li> <li>• 110: WRAP16 (16-beat wrapping burst)</li> <li>• 111: INCR16 (16-beat incrementing burst)</li> </ul> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>
hwdata_i[K – 1:0]	In	<p>AHB Write Data</p> <p><b>Function:</b> AHB write data input to the Slave port. The data bus width is configurable (K = 32, 64 or 128).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i</p>
hreadyslv_i	In	<p>AHB Master/Slave Ready</p> <p><b>Function:</b> The hready of the AHB bus input to all master and slave ports</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i</p>

**Table 4-11 AHB Slave Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
hready_o	Out	AHB Slave Ready  <b>Function:</b> The GMAC-AHB Subsystem Slave indicates that the current transfer has finished.  <b>Active State:</b> High <b>Registered:</b> Yes <b>Synchronous to:</b> hclk_i
hresp_o[1:0]	Out	AHB Slave Response  <b>Function:</b> hresp_o provides transfer status information. Valid responses are: <ul style="list-style-type: none"><li>• 00: OKAY: Transfer completed OK</li><li>• 01: ERROR: Error in current transfer</li></ul> All other encodings are not used.  <b>Active State:</b> N/A <b>Registered:</b> Yes <b>Synchronous to:</b> hclk_i
hrdata_o[K – 1:0]	Out	AHB Slave Read Data  <b>Function:</b> AHB slave read data output from the Slave port. The data bus width is configurable (K = 32, 64 or 128).  <b>Active State:</b> N/A <b>Registered:</b> Yes <b>Synchronous to:</b> hclk_i
sbd_intr_o	Out	GMAC-AHB Subsystem Interrupt  <b>Function:</b> This signal, when high, indicates an interrupt event to the application.  <b>Active State:</b> High <b>Registered:</b> No <b>Synchronous to:</b> hclk_i

## 4.2.12 AXI Master Interface Signals

The AHB Master interface signals are described in [Table 4-12](#).

**Table 4-12 AXI Master Interface Signals**

Signal	I/O	Description
aclk_i	In	<p>AMBA AXI System Clock</p> <p><b>Function:</b> This is the free-running AXI clock input provided by the Application. The DMA and the MTL modules also operate with this clock.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> N/A</p>
areset_n	In	<p>AMBA AXI Power On System Reset</p> <p><b>Function:</b> AMBA AXI power on system reset signal</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
awid_m_o[MW-1:0]	Out	<p>Master Write Address ID</p> <p><b>Function:</b> This signal is the identification tag for the write address group of signals. The width is configured during RTL configuration</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
awaddr_m_o[31:0]	Out	<p>Master Write Address</p> <p><b>Function:</b> The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
awlen_m_o[BL-1:0]	Out	<p>Master Burst Length</p> <p><b>Function:</b> The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. The default width is 4 but can be increased to support upto 256-beat transfers during RTL configuration. The value of all-zeros indicates a single transfer.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-12 AXI Master Interface Signals (Continued)**

Signal	I/O	Description
awsize_m_o[2:0]	Out	<p>Master Burst Size</p> <p><b>Function:</b> This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update. This is always driven to have 'System Datawidth' transfers. 3'b010 for 32-bit, 3'b011 for 64-bit and 3'b100 for 128-bit interface.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
awburst_m_o[1:0]	Out	<p>Master Burst Type</p> <p><b>Function:</b> The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. This is always hard wired to 2'b01 to indicate INCR burst type.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
awlock_m_o[1:0]	Out	<p>Master Lock Type</p> <p><b>Function:</b> This signal provides additional information about the atomic characteristics of the transfer. This is always hard wired to 2'b00 to indicate normal access.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
awcache_m_o[3:0]	Out	<p>Master Cache Type</p> <p><b>Function:</b> This signal indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction. This is always hard wired to 4'b0000 to indicate Non-cacheable and Non-bufferable.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
awprot_m_o[2:0]	Out	<p>Master Protection Type</p> <p><b>Function:</b> This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access. This is always hard wired to 3'b000 to indicate Normal, Non-secure, Data access.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-12 AXI Master Interface Signals (Continued)**

Signal	I/O	Description
awvalid_m_o	Out	<p>Master Write Address Valid</p> <p><b>Function:</b> This signal indicates that valid write address and control information are available:</p> <ul style="list-style-type: none"> <li>• 1: address and control information available</li> <li>• 0: address and control information not available</li> </ul> <p>The address and control information remain stable until the address acknowledge signal, awready, goes high.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
awready_m_i	In	<p>Master Write Address Ready</p> <p><b>Function:</b> This signal indicates that the AXI slave is ready to accept an address and associated control signals:</p> <ul style="list-style-type: none"> <li>• 1: Slave ready</li> <li>• 0: Slave not ready</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
wid_m_o[MW-1:0]	Out	<p>Master Write ID Tag</p> <p><b>Function:</b> This signal is the ID tag of the write data transfer. The WID value must match the AWID value of the write transaction. MW takes the value AXI_GM_AXI_ID_WIDTH user-configurable parameter.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
wdata_m_o[N-1:0]	Out	<p>Master Write Data</p> <p><b>Function:</b> The write data bus can be 32, 64 or 128 -bit wide, and takes the value of parameter "System Data Width" (DATAWIDTH).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
wstrb_m_o[N/8-1:0]	Out	<p>Master Write Strobes</p> <p><b>Function:</b> This signal indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-12 AXI Master Interface Signals (Continued)**

Signal	I/O	Description
wlast_m_o	Out	<p>Master Write Last</p> <p><b>Function:</b> This signal indicates the last transfer in a write burst.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
wvalid_m_o	Out	<p>Master Write Valid</p> <p><b>Function:</b> This signal indicates that valid write data and strobes are available:</p> <ul style="list-style-type: none"> <li>• 1: write data and strobes available</li> <li>• 0: write data and strobes not available</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
wready_m_i	In	<p>Master Write Ready</p> <p><b>Function:</b> This signal indicates that the Slave accepted the write data:</p> <ul style="list-style-type: none"> <li>• 1: Slave accepted</li> <li>• 0: Slave has not accepted</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
bid_m_i[MW-1:0]	In	<p>Master Response ID</p> <p><b>Function:</b> The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding. The width of this signal is configured during RTL configuration.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
bresp_m_i[1:0]	In	<p>Master Write Response</p> <p><b>Function:</b> This signal indicates the status of the write transaction. The allowable responses are OKAY(2'b00), EXOKAY(2'b01), SLVERR(2'b10), and DECERR(2'b11).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
bvalid_m_i	In	<p>Master Write Response Valid</p> <p><b>Function:</b> This signal indicates that a valid write response is available:</p> <ul style="list-style-type: none"> <li>• 1: write response available</li> <li>• 0: write response not available</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-12 AXI Master Interface Signals (Continued)**

Signal	I/O	Description
bready_m_o	Out	<p>Master Response Ready</p> <p><b>Function:</b> This signal indicates that the master can accept the response information.</p> <ul style="list-style-type: none"> <li>• 1: master ready</li> <li>• 0: master not ready</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
arid_m_o[MW:0]	Out	<p>Master Read Address ID</p> <p><b>Function:</b> This signal is the identification tag for the read address group of signals. The width of this signal is configured during RTL configuration.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
araddr_m_o[31:0]	Out	<p>Master Read Address</p> <p><b>Function:</b> The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
arlen_m_o[BL-1:0]	Out	<p>Master Burst Length</p> <p><b>Function:</b> The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address. The default width is 4 but width of this signal can be increased during RTL configuration to support bigger bursts.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
arsize_m_o[2:0]	Out	<p>Master Burst Size</p> <p><b>Function:</b> This signal indicates the size of each transfer in the burst. This is always driven to have 'System Datawidth' transfers. 3'b010 for 32-bit, 3'b011 for 64-bit and 3'b100 for 128-bit interface.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-12 AXI Master Interface Signals (Continued)**

Signal	I/O	Description
arburst_m_o[1:0]	Out	<p>Master Burst Type</p> <p><b>Function:</b> The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. This is always hard wired to 2'b01 to indicate INCR burst type.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
arlock_m_o[1:0]	Out	<p>Lock Type</p> <p><b>Function:</b> This signal provides additional information about the atomic characteristics of the transfer. This is always hard wired to 2'b00 to indicate normal access.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
arcache_m_o[3:0]	Out	<p>Master Cache Type</p> <p><b>Function:</b> This signal provides additional information about the cacheable characteristics of the transfer. This is always hard wired to 4'b0000 to indicate Noncacheable and Nonbufferable.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
arprot_m_o[2:0]	Out	<p>Master Protection Type</p> <p><b>Function:</b> This signal provides protection unit information for the transaction. This is always hard wired to 3'b000 to indicate Normal, Non-secure, Data access.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
arvalid_m_o	Out	<p>Master Read Address Valid</p> <p><b>Function:</b> This signal indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledge signal,<b>ARREADY</b>, is high.</p> <ul style="list-style-type: none"> <li>• 1: address and control information valid</li> <li>• 0: address and control information not valid</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-12 AXI Master Interface Signals (Continued)**

Signal	I/O	Description
arready_m_i	In	<p>Master Read Ready</p> <p>This signal indicates that the Slave can accept the read request:</p> <ul style="list-style-type: none"> <li>• 1: Slave ready</li> <li>• 0: Slave not ready</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
rid_m_i[MW-1:0]	In	<p>Master Read ID Tag</p> <p><b>Function:</b> This signal is the ID tag of the read data group of signals. The rid value is generated by the Slave and must match the arid value of the read transaction to which it is responding.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
rdata_m_i[N-1:0]	In	<p>Master Read Data</p> <p><b>Function:</b> The read data bus can be 32, 64 or 128-bit wide and takes the value of parameter "System Data width" (DATAWIDTH).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
rresp_m_i	In	<p>Master Read Response</p> <p><b>Function:</b> This signal indicates the status of the read transfer. The allowable responses are OKAY(2'b00), EXOKAY(2'b01), SLVERR(2'b10) and DECERR(2'b11).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
rlast_m_i	In	<p>Master Read Last</p> <p><b>Function:</b> This signal indicates the last transfer in a read burst.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
rvalid_m_i	In	<p>Master Read Valid</p> <p><b>Function:</b> This signal indicates that the required read data is available and the read transfer can complete:</p> <ul style="list-style-type: none"> <li>• 1: read data available</li> <li>• 0: read data not available</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-12 AXI Master Interface Signals (Continued)**

Signal	I/O	Description
rready_m_o	Out	<p>Master Read Ready</p> <p>This signal indicates that the master can accept the read data and response information:</p> <ul style="list-style-type: none"> <li>• 1: master ready</li> <li>• 0: master not ready</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>

#### 4.2.13 AXI Low Power Interface Signals

The AXI Low Power Interface interface signals are described in [Table 4-13](#).

**Table 4-13 RGMII/RTBI Interface Signals (Optional)**

Signal	I/O	Description
csysreq_i	In	<p>Clock Controller System low-power request.</p> <p>This signal is a request from the system clock controller for the GMAC-AXI to enter a low-power state.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
csysack_o	Out	<p>Low-Power Request Acknowledgement</p> <p>This signal is the acknowledgement from the GMAC-AXI to the system low-power request.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
cactive_o	Out	<p>Clock Active</p> <p>This signal indicates that the GMAC-AXI requires its clock signal:</p> <ul style="list-style-type: none"> <li>• 1: GMAC-AXI clock required</li> <li>• 0: GMAC-AXI clock not required</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b></p> <ul style="list-style-type: none"> <li>- clk_rx_i when self-initiating request to come out of low-power mode</li> <li>- aclk_i in all other cases.</li> </ul>

## 4.2.14 AXI Slave Interface Signals

The AXI Slave interface signals are described in [Table 4-14](#).

**Table 4-14 AXI Slave Interface Signals**

Signal	I/O	Description
awid_s_i [GS_ID-1:0]	In	<p>Slave Write Address ID</p> <p><b>Function:</b> This signal is the identification tag for the write address group of signals.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
awaddr_s_i[31:0]	In	<p>Slave Write Address</p> <p><b>Function:</b> The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
awlen_s_i[3:0]	In	<p>Slave Burst Length</p> <p><b>Function:</b> The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
awsize_s_i[2:0]	In	<p>Slave Burst Size</p> <p><b>Function:</b> This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
awburst_s_i[1:0]	In	<p>Slave Burst Type</p> <p><b>Function:</b> The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
awlock_s_i[1:0]	In	<p>Slave Lock Type</p> <p><b>Function:</b> This signal provides additional information about the atomic characteristics of the transfer. GMAC AXI Slave does not support Exclusive access.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-14 AXI Slave Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
awcache_s_i[3:0]	In	<p>Slave Cache Type</p> <p><b>Function:</b> This signal indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
awprot_s_i[2:0]	In	<p>Slave Protection Type</p> <p><b>Function:</b> This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
awvalid_s_i	In	<p>Slave Write Address Valid</p> <p><b>Function:</b> This signal indicates that valid write address and control information are available:</p> <ul style="list-style-type: none"> <li>• 1: address and control information available</li> <li>• 0: address and control information not available</li> </ul> <p>The address and control information remain stable until the address acknowledge signal, awready_s_o, goes high.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
awready_s_o	Out	<p>Slave Write Address Ready</p> <p><b>Function:</b> This signal indicates that the slave is ready to accept an address and associated control signals:</p> <ul style="list-style-type: none"> <li>• 1: slave ready</li> <li>• 0: slave not ready</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
wid_s_i [GS_ID-1:0]	In	<p>Slave Write ID Tag</p> <p><b>Function:</b> This signal is the ID tag of the write data transfer. The wid value must match the awid value of the write transaction.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-14 AXI Slave Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
wdata_s_i[N-1:0]	In	<p>Slave Write Data</p> <p><b>Function:</b> The write data bus can be N takes the value 'System Bus Width" parameter value (DATAWIDTH).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
wstrb_s_i[N/8-1:0]	In	<p>Slave Write Strobes</p> <p><b>Function:</b> This signal indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
wlast_s_i	In	<p>Slave Write Last</p> <p><b>Function:</b> This signal indicates the last transfer in a write burst.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
wvalid_s_i	In	<p>Slave Write Valid</p> <p><b>Function:</b> This signal indicates that valid write data and strobes are available:</p> <ul style="list-style-type: none"> <li>• 1: write data and strobes available</li> <li>• 0: write data and strobes not available</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
wready_s_o	Out	<p>Slave Write Ready</p> <p><b>Function:</b> This signal indicates that the slave can accept the write data:</p> <ul style="list-style-type: none"> <li>• 1: slave ready</li> <li>• 0: slave not ready</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
bid_s_o [GS_ID-1:0]	Out	<p>Slave Response ID</p> <p><b>Function:</b> The identification tag of the write response. The bid value must match the awid value of the write transaction to which the slave is responding.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-14 AXI Slave Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
bresp_s_o[1:0]	Out	<p>Slave Write Response</p> <p><b>Function:</b> This signal indicates the status of the write transaction. The allowable responses are OKAY(2'b00), EXOKAY(2'b01), SLVERR(2'b10) and DECERR(2'b11).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
bvalid_s_o	Out	<p>Slave Write Response valid</p> <p><b>Function:</b> This signal indicates that a valid write response is available:</p> <ul style="list-style-type: none"> <li>• 1: write response available</li> <li>• 0: write response not available</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
bready_s_i	In	<p>Slave Response Ready</p> <p><b>Function:</b> This signal indicates that the master can accept the response information.</p> <ul style="list-style-type: none"> <li>• 1: slave ready</li> <li>• 0: slave not ready</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
arid_s_i [GS_ID-1:0]	In	<p>Slave Read Address ID</p> <p><b>Function:</b> This signal is the identification tag for the read address group of signals.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
araddr_s_i[31:0]	In	<p>Slave Read Address</p> <p><b>Function:</b> The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
arlen_s_i[3:0]	In	<p>Slave Burst Length</p> <p><b>Function:</b> The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-14 AXI Slave Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
arsize_s_i[2:0]	In	<p>Slave Burst Size</p> <p><b>Function:</b> This signal indicates the size of each transfer in the burst.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
arburst_s_i[1:0]	In	<p>Slave Burst Type</p> <p><b>Function:</b> The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
arlock_s_i[1:0]	In	<p>Lock Type</p> <p><b>Function:</b> This signal provides additional information about the atomic characteristics of the transfer.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
arcache_s_i[3:0]	In	<p>Master Cache Type</p> <p><b>Function:</b> This signal provides additional information about the cacheable characteristics of the transfer.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
arprot_s_i[2:0]	In	<p>Slave Protection Type</p> <p><b>Function:</b> This signal provides protection unit information for the transaction. GMAC AXI Slave does not support Exclusive access.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
arvalid_s_i	In	<p>Slave Read Address Valid</p> <p><b>Function:</b> This signal indicates, when high, that the read address and control information is valid and will remain stable until the address acknowledge signal, arready, is high.</p> <ul style="list-style-type: none"> <li>• 1: address and control information valid</li> <li>• 0: address and control information not valid</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-14 AXI Slave Interface Signals (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
arready_s_i	In	<p>Slave Read Address Ready</p> <p><b>Function:</b> This signal indicates that the slave is ready to accept an address and associated control signals:</p> <ul style="list-style-type: none"> <li>• 1: slave ready</li> <li>• 0: slave not ready</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>
rid_s_o [GS_ID-1:0]	Out	<p>Slave Read ID Tag</p> <p><b>Function:</b> This signal is the ID tag of the read data group of signals. The rid value is generated by the slave and must match the arid value of the read transaction to which it is responding.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
rdata_s_o[N-1:0]	Out	<p>Slave Read Data</p> <p><b>Function:</b> The read data bus can be 32, 64 or 128-bit wide and takes the value 'System Data Bus width" parameter (DATAWIDTH).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
rresp_s_o[1:0]	Out	<p>Slave Read Response</p> <p><b>Function:</b> This signal indicates the status of the read transfer. The allowable responses are OKAY(2'b00), EXOKAY(2'b01), SLVERR(2'b10) and DECERR(2'b11).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
rlast_s_o	Out	<p>Slave Read Last</p> <p><b>Function:</b> This signal indicates the last transfer in a read burst.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> aclk_i</p>
rvalid_s_o	Out	<p>Slave Read Valid</p> <p><b>Function:</b> This signal indicates that the required read data is available and the read transfer can complete:</p> <ul style="list-style-type: none"> <li>• 1: read data available</li> <li>• 0: read data not available</li> </ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>

**Table 4-14 AXI Slave Interface Signals (Continued)**

Signal	I/O	Description
rready_s_i	In	<p>Slave Read Ready</p> <p>This signal indicates that the slave can accept the read data and response information:</p> <ul style="list-style-type: none"><li>• 1: AXI Master ready</li><li>• 0: Master not ready</li></ul> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> aclk_i</p>

#### 4.2.15 External DPRAM Interface Signals

The External DPRAM interface signals are described in [Table 4-15](#).



Signals on the application side are synchronous to hclk\_i in the GMAC-AHB configuration and synchronous to clk\_app\_i in the GMAC-MTL and GMAC-DMA configurations.

**Table 4-15 External DPRAM Interface Signals**

Signal	I/O	Description
twc_wr_addr_o[X:0]	Out	<p>Transmit FIFO Write Address</p> <p><b>Function:</b> This signal gives the address to the Tx FIFO write port. It is valid when twc_wr_csn_o is asserted. The width of the bus depends on the Tx FIFO depth and width. For example, for a Tx FIFO depth of 2,048 bytes and a data bus width of 64 bits, X = 6.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i</p>
twc_wr_data_o[Y:0]	Out	<p>Transmit FIFO Write Data</p> <p><b>Function:</b> This bus carries the data to be written in the Tx FIFO. It is valid when twc_wr_csn_o and tfc_wr_en_o are asserted. The width of this data bus (Y = 34, 67, or 132) depends on the data bus width (32/64/128 respectively) selected during coreKit configuration.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i</p>
twc_wr_en_o	Out	<p>Transmit FIFO Write Enable</p> <p><b>Function:</b> When high, indicates that twc_wr_data_o should be written into memory</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i</p>
twc_wr_csn_o	Out	<p>Transmit FIFO Write Port Chip Select</p> <p><b>Function:</b> Qualifies the twc_wr_addr_o, twc_wr_data_o, and twc_wr_en_o signals.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i</p>
trc_rd_addr_o[X:0]	Out	<p>Transmit FIFO Read Address</p> <p><b>Function:</b> This gives the address to the Tx FIFO read port. It is valid when trc_rd_csn_o is asserted. The width of address bus depends on the Tx FIFO Depth and the data bus width.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_tx_i</p>

**Table 4-15 External DPRAM Interface Signals (Continued)**

Signal	I/O	Description
trc_rd_data_i[Y:0]	In	<p>Transmit FIFO Read Data</p> <p><b>Function:</b> This bus carries the data read from the Tx FIFO. It is valid when trc_rd_csn_o and trc_rd_en_o are asserted. The width of the data bus (<math>Y = 34, 67, \text{ or } 132</math>) depends on the data bus width (32, 64, or 128, respectively) selected during coreKit configuration.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>
trc_rd_en_o	Out	<p>Transmit FIFO Read Enable</p> <p><b>Function:</b> When high, indicates that the trc_rd_data_i should contain valid data.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_tx_i</p>
trc_rd_csn_o	Out	<p>Transmit FIFO Read Port Chip Select</p> <p><b>Function:</b> Qualifies the trc_rd_addr_o, trc_rd_data_i, and trc_rd_en_o signals.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>
rwc_wr_addr_o[Z:0]	Out	<p>Receive FIFO Write Address</p> <p><b>Function:</b> This gives the address to the Rx FIFO write port. It is valid when rwc_wr_csn_o is asserted. The width of address bus depends on the Rx FIFO depth and the data bus width. For example, for an Rx FIFO depth of 2,048 bytes and a data bus width of 32-bits, <math>Z = 8</math>. For an Rx FIFO depth of 8192 bytes and a data bus width of 64-bits, <math>Z = 9</math>.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>
rwc_wr_data_o[Y:0]	Out	<p>Receive FIFO Write Data</p> <p><b>Function:</b> This bus carries the data to be written in the Rx FIFO. It is valid when rwc_wr_csn_o and rwc_wr_en_o are asserted. The width of the data bus (<math>Y = 34, 67, \text{ or } 132</math>) depends on the data bus width (32, 64, or 128, respectively) selected during coreKit configuration.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>
rwc_wr_en_o	Out	<p>Receive FIFO Write Enable</p> <p><b>Function:</b> When high, indicates that the rwc_wr_data_o should be written into memory</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_rx_i</p>

**Table 4-15 External DPRAM Interface Signals (Continued)**

Signal	I/O	Description
rwc_wr_csn_o	Out	<p>Receive FIFO Write Port Chip Select</p> <p><b>Function:</b> Qualifies the rwc_wr_addr_o, rwc_wr_data_o, and rwc_wr_en_o signals.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_rx_i</p>
rrc_rd_addr_o[Z:0]	Out	<p>Receive FIFO Read Address</p> <p><b>Function:</b> This gives the address to the Rx FIFO read port. It is valid when rrc_rd_csn_o is asserted. The width of address bus depends on the Rx FIFO Depth and the data bus width.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i</p>
rrc_rd_data_i[Y:0]	In	<p>Receive FIFO Read Data</p> <p><b>Function:</b> This bus carries the data read from the Rx FIFO. It is valid when rrc_rd_csn_o and rrc_rd_en_o are asserted. The width of the data bus (<math>Y = 34, 67, \text{ or } 132</math>) depends on the data bus width (32, 64, or 128, respectively) selected during coreKit configuration.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i</p>
rrc_rd_en_o	Out	<p>Receive Read Enable</p> <p><b>Function:</b> When high, indicates that rrc_rd_data_i should contain valid data.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i</p>
rrc_rd_csn_o	Out	<p>Receive Read Qualification</p> <p><b>Function:</b> Qualifies the rrc_rd_addr_o, rrc_rd_data_i, and rx_rd_en_o signals.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i</p>

## 4.2.16 External DPRAM (Frame Length FIFO) Interface Signals

The External DPRAM Frame Length FIFO interface signals are described in [Table 4-16](#). These signals can be configured to be active only in the GMAC-MTL configuration (refer to [Table 6-7](#).)

**Table 4-16 External DPRAM Interface Signals**

Signal	I/O	Description
rwc_len_wr_addr_o[Z:0]	Out	<p>Receive Frame Length FIFO Write Address</p> <p><b>Function:</b> Gives the address to the Rx Frame Length FIFO write port. It is valid when rwc_len_wr_csn_o is asserted. The width of the address bus depends on the maximum number of frames stored in the Rx FIFO. For example, for an Rx FIFO depth of 2,048 bytes and a minimum receive frame size of 16 bytes, the maximum number of frames that can be stored in the Rx FIFO is 128. Thus, Z = 6.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>
rwc_len_wr_en_o	Out	<p>Receive Frame Length FIFO Write Enable</p> <p><b>Function:</b> When high, indicates that rwc_wr_data_o[Y + 16:16] should be written into DPRAM, where Y = Width of the Frame Length FIFO. Note that the rwc_wr_data_o is the same data bus written to the MTL Receive FIFO.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_rx_i</p>
rwc_err_frame_o	Out	<p>Frame Error Status</p> <p><b>Function:</b> When high, indicates that the frame whose status is being written is an error frame and the frame should be dropped by the Rx FIFO Read Controller. This bit should be concatenated as MSB to the frame length data (rwc_wr_data_o[Y+16:16]), written to the Rx Frame Length FIFO.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>
rwc_len_wr_csn_o	Out	<p>Receive Frame Length FIFO Write Port Chip Select</p> <p><b>Function:</b> Indicates that the rwc_len_wr_addr_o, rwc_wr_data_o, and rwc_len_wr_en_o signals have valid values.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_rx_i</p>
rrc_len_rd_addr_o[Z:0]	Out	<p>Receive Frame Length FIFO Read Address</p> <p><b>Function:</b> This gives the address to the Rx Frame Length FIFO read port. It is valid when rrc_len_rd_csn_o is asserted. The width of this port is same as rwc_len_wr_addr_o.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>

**Table 4-16 External DPRAM Interface Signals (Continued)**

Signal	I/O	Description
rrc_len_rd_data_i[Y + 1:0]	In	<p>Receive FIFO Read Data</p> <p><b>Function:</b> This bus carries the data read from the Frame Length FIFO. It is valid when rrc_len_rd_csn_o and rrc_len_rd_en_o are asserted. The width of the data bus is configurable and ranges from 12 to 15 bits (Y = 10 to 13). The MS bit gives the frame-error status while the other bits indicate the length of the frame. Stored in the top of Rx FIFO.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>
rrc_len_rd_en_o	Out	<p>Receive Read Enable</p> <p><b>Function:</b> When high, indicates that rrc_rd_data_i contains valid data.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
rrc_len_rd_csn_o	Out	<p>Receive Read Qualification</p> <p><b>Function:</b> Qualifies the rrc_len_rd_addr_o, rrc_len_rd_data_i, and rrc_len_rd_en_o signals.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>
ari_frame_len_o[Y:0]	Out	<p>Receive Frame Length</p> <p><b>Function:</b> This bus carries the frame length of the received frame at the top of the MTL Rx FIFO. Frame length is valid when ari_frame_len_val_o is asserted.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
ari_frame_len_val_o	Out	<p>Receive Frame Length Qualifier</p> <p><b>Function:</b> Qualifies the ari_frame_len_o signal.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
ari_err_frame_o	Out	<p>Frame Error Status</p> <p><b>Function:</b> When high, this signal indicates that the frame being transferred on the ARI is an error frame and can be dropped by the application. This signal has valid value when ari_frame_len_val_o is asserted.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>

#### 4.2.17 RGMII/RTBI Interface Signals (Optional)

The optional RGMII interface signals are described in [Table 4-17](#).

**Table 4-17 RGMII/RTBI Interface Signals (Optional)**

Signal	I/O	Description
clk_tx_180_i	In	<p>Transmit Clock</p> <p><b>Function:</b> The RGMII transmit interface uses the rising edge of clk_tx_180_i to represent the falling edge of clk_tx_i. This clock input is not used in the RTBI interface</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> clk_tx_i</p>
clk_rx_180_i	In	<p>Receive Clock</p> <p><b>Function:</b> The RGMII receive interface uses the positive edge of clk_rx_180_i to represent the falling edge of clk_rx_i. The RTBI receive interface uses the positive edge of clk_rx_180_i to represent the falling edge of clk_rx_125_i.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> clk_rx_i</p>
rst_clk_tx_180_n	In	<p>Transmit Clock Reset</p> <p><b>Function:</b> Reset signal. This input is not present in the GMAC-AHB or GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_180_i</p>
rst_clk_rx_180_n	In	<p>Receive Clock Reset</p> <p><b>Function:</b> Reset signal. This input is not present in the GMAC-AHB or GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_rx_180_i</p>

#### 4.2.18 RMII Interface Signals (Optional)

The optional RMII interface signals are described in [Table 4-18](#).

**Table 4-18 RMII Interface Signals (Optional)**

Signal	I/O	Description
clk_rmii_i	In	<p>RMII Clock</p> <p><b>Function:</b> This is the 50-MHz clock used by the RMII interface. clk_rx_i should be 25/2.5 MHz, derived from this clock when the RMII interface is selected.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> N/A</p>

**Table 4-18 RMII Interface Signals (Optional) (Continued)**

Signal	I/O	Description
rst_clk_rmii_n	In	<p>RMII Clock Reset</p> <p><b>Function:</b> Reset signal. This input is not present in the GMAC-AHB or GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_rmii_i</p>
mac_speed_i	In	<p>MAC Speed</p> <p><b>Function:</b> When high, indicates 100-Mbps operation. When low, indicates 10-Mbps operation. If you choose the output signal (mac_speed_o) during configuration, this input pin is removed.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_rmii_i</p>

#### 4.2.19 SMII Interface Signal (Optional)

The optional SMII sync signal is described in [Table 4-19](#).

**Table 4-19 SMII Interface Signal (Optional)**

Signal	I/O	Description
smii_txsync_i	In	<p>SMII Transmit Synchronization - input signal in source synchronous (SSSMII) mode</p> <p><b>Function:</b> This signal will exist when the SSSMII_TXSYNC_IN parameter is enabled during coreKit configuration.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_tx_125_i</p>

#### 4.2.20 PMT Interrupt Signal (Optional)

The optional PMT Interrupt signal is described in [Table 4-20](#).

**Table 4-20 PMT Interrupt Signal (Optional)**

Signal	I/O	Description
pmt_intr_o	Out	<p>PMT Interrupt</p> <p><b>Function:</b> When high, indicates an interrupt event in the optional PMT module of the core. This interrupt pin is always present when the optional PMT is included and can be used to wake the system from Sleep mode. This interrupt has no masking control.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_rx_i</p>

## 4.2.21 SGMII/TBI/RTBI Interface Signals (Optional)

The optional SGMII/TBI/RTBI interface signals are described in [Table 4-21](#).

**Table 4-21 SGMII/TBI/RTBI Interface Signals (Optional)**

Signal	I/O	Description
clk_tx_125_i	In	<p>125-MHz Transmit Clock</p> <p><b>Function:</b> This is the 125 MHz clock input for the SGMII/TBI/RTBI Transmit interface. The clk_tx_i clock (125/25/2.5 MHz) should be synchronous with this clock.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> N/A</p>
clk_rx_125_i	In	<p>125-MHz Receive Clock</p> <p><b>Function:</b> This is the 125-MHz clock input for the SGMII/TBI/RTBI Receive interface. The clk_rx_i clock (125/25/2.5 MHz) must be synchronous with this clock</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> N/A</p>
clk_tx_125_180_i	In	<p>125-MHz Transmit Clock (180)</p> <p><b>Function:</b> The SGMII/TBI/RTBI transmit interface uses the rising edge of clk_tx_125_180_i to represent the falling edge of clk_tx_125_i.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> clk_tx_125_i</p>
rst_clk_tx_125_n	In	<p>125-MHz Transmit Clock Reset</p> <p><b>Function:</b> Reset signal. This input is not present in the GMAC-AHB or GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_125_i</p>
rst_clk_rx_125_n	In	<p>125-MHz Receive Clock Reset</p> <p><b>Function:</b> Reset signal. This input is not present in the GMAC-AHB or GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_rx_125_i</p>
rst_clk_tx_125_180_n	In	<p>125-MHz Transmit Clock (180) Reset</p> <p><b>Function:</b> Reset signal. This input is not present in the GMAC-AHB or GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_125_180_i</p>

**Table 4-21 SGMII/TBI/RTBI Interface Signals (Optional) (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
pcs_aquired_sync_o	Out	<p>PCS Acquired Synchronization</p> <p><b>Function:</b> When high, indicates that the PCS interface has synchronized to the running disparity of the receive code-groups</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_125_i /clk_rx_i</p>
pcs_ewrap_o	Out	<p>PCS Enable Wrap</p> <p><b>Function:</b> This signal is the enable for the PHY loop back. When asserted high, the PHY should loop back the transmit serialized data to the receive path. This bit mirrors the ELE bit of the PCS Control register.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_csr_i</p>
pcs_lck_ref_o	Out	<p>PCS Lock Reference</p> <p><b>Function:</b> This signal mirrors the LR bit of the PCS Control register. It is low after reset.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_csr_i</p>
pcs_en_cdet_o	Out	<p>PCS Enable Detect</p> <p><b>Function:</b> This signal mirrors the ECD bit of the PCS Control register. It is low after reset.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_csr_i</p>

## 4.2.22 SGMII Interface Signals (Optional)

The optional SGMII interface signals are described in [Table 4-22](#).

**Table 4-22 SGMII Interface Signals (Optional)**

Signal	I/O	Description
sgmii_link_speed_o[1:0]	Out	<p>SGMII Link Speed</p> <p><b>Function:</b> Indicates link speed and should be used to change the frequency of clk_tx_i and clk_rx_i:</p> <ul style="list-style-type: none"> <li>• 00: 10 Mbps</li> <li>• 01: 100 Mbps</li> <li>• 10: 1000 Mbps</li> <li>• 11: Reserved:</li> </ul> <p><b>Note:</b> If auto-negotiation is disabled in SGMII, this signal retains the value and does not change. The default value after reset is 10.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_csr_i</p>

## 4.2.23 TBI Interface Signals

The additional TBI interface signals are described in [Table 4-23](#).

**Table 4-23 TBI Interface Signals**

Signal	I/O	Description
clk_pmarx0_i	In	<p>Recovered Receive Clock</p> <p><b>Function:</b> 62.5-MHz recovered receive clock used by the GMAC to register the received data odd code groups.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> N/A</p>
clk_pmarx1_i	In	<p>Recovered Receive Clock (180)</p> <p><b>Function:</b> The 62.5-MHz recovered receive clock. This clock is 180 degrees out of phase with clk_pmarx0_i and is used by the GMAC to register the even code groups of the received data</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> clk_pmarx0_i</p>
rst_clk_pmarx1_n	In	<p>Recovered Receive Clock (180) Reset</p> <p><b>Function:</b> Reset signal. This input is not present in the GMAC-AHB or GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_pmarx0_i</p>

**Table 4-23 TBI Interface Signals (Continued)**

Signal	I/O	Description
tbi_sigdet_i	In	TBI Signal Detect <b>Function:</b> When high, informs the GMAC that a signal has been detected at the PMD device. As TBI is generally used with Fibre media, it indicates the detection of light. <b>Active State:</b> High <b>Registered:</b> No <b>Synchronous to:</b> clk_rx_125_i

**4.2.24 SMA Interface Signals (Optional)**

The optional SMA interface signals are described in [Table 4-24](#).

**Table 4-24 SMA Interface Signals (Optional)**

Signal	I/O	Description
gmii_mdo_o	Out	Management Data Out <b>Function:</b> The GMAC uses this signal to transfer control and data information to the PHY. <b>Active State:</b> N/A <b>Registered:</b> Yes <b>Synchronous to:</b> clk_csr_i
gmii_mdi_i	In	Management Data In <b>Function:</b> The PHY generates this signal to transfer register data during a read operation. This signal is driven synchronously with the gmii_mdc_o clock. <b>Active State:</b> N/A <b>Registered:</b> Yes <b>Synchronous to:</b> clk_csr_i
gmii_mdo_o_e	Out	Management Data Output Enable <b>Function:</b> This enable signal drives the gmii_mdo_o signal from an external three-state I/O buffer. This signal is asserted whenever valid data is driven on the gmii_mdo_o signal. <b>Active State:</b> High <b>Registered:</b> Yes <b>Synchronous to:</b> clk_csr_i
gmii_mdc_o	Out	Management Data Clock <b>Function:</b> The GMAC provides timing reference for the gmii_mdi_i and gmii_mdo_o signals on the GMII/MII through this aperiodic clock. The maximum frequency of this clock is 2.5 MHz. This clock is generated from the application clock via a clock divider. <b>Active State:</b> N/A <b>Registered:</b> Yes <b>Synchronous to:</b> clk_csr_i

## 4.2.25 APB Interface Signals (Optional)

The optional APB interface signals (instead of the AHB Slave Interface in the GMAC-AHB configuration or the MAC Control Interface in other configurations) are described in [Table 4-25](#).



**Note** All signals in this interface are synchronous to hclk\_i (for GMAC-AHB configuration), clk\_app\_i (for GMAC-DMA and GMAC-MTL configurations) by default, and are synchronous to clk\_csr\_i clock when the optional clk\_csr\_i clock input is selected. In the GMAC-CORE configuration, all signals are synchronous to clk\_csr\_i.

**Table 4-25 APB Interface Signals (Optional)**

Signal	I/O	Description
psel_i	In	<p>APB Slave Select</p> <p><b>Function:</b> The assertion of this signal is the start of transaction and indicates the validity of the control signals and pwdata.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i/clk_csr_i</p>
paddr_i[15:0]	In	<p>APB Address</p> <p><b>Function:</b> This signal carries the register address of the CSR module. The address is valid only when signal psel_i is asserted.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i/clk_csr_i</p>
penable_i	In	<p>APB Enable</p> <p><b>Function:</b> This signal, when high, completes the read or write transaction cycle.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i/clk_csr_i</p>
pwrite_i	In	<p>APB Read/Write</p> <p><b>Function:</b> When high, indicates a write operation, when low, a read operation</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i/clk_csr_i</p>
pwdata_i[31:0]	In	<p>Write Data</p> <p><b>Function:</b> This signal carries the write data from the Application for the CSR module Control registers.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i/clk_csr_i</p>

**Table 4-25 APB Interface Signals (Optional) (Continued)**

Signal	I/O	Description
prdata_o[31:0]	Out	<p>Read Data</p> <p><b>Function:</b> This signal outputs the read data from the CSR module to the APB bus. Valid data is output 1 clock cycle after psel_i is asserted. If psel_i continues to be asserted for burst transfers, then valid data is output every 2 clock cycles during this period.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> hclk_i/clk_app_i/clk_csr_i</p>

#### 4.2.26 Sideband Signals (Optional)

These signals are optional and applicable to specific configurations as mentioned in its description. By default configuration, these signals are not enabled.

**Table 4-26 Sideband Signals (Optional)**

Signal	I/O	Description
sbd_flowctrl_i	In	<p>Sideband Flow Control</p> <p><b>Function:</b> When set high, instructs the GMAC to transmit PAUSE Control frames in Full-duplex mode. In Half-duplex mode, the GMAC enables the back-pressure function until this signal is made low again. This signal is an optional port applicable to only GMAC-AHB, GMAC-DMA and GMAC-MTL configuration.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> Asynchronous</p>
sbd_dis_transmit_i	In	<p>Sideband MAC Transmitter Disable Control</p> <p><b>Function:</b> When set high, this signal instructs the GMAC transmitter to stop transmitting frames after the completion of any current frame. The GMAC transmitter restarts transmission only if this signal is reset to low and the MAC Configuration register Transmitter Enable bit [3] is set high.</p> <p>This signal is an optional port applicable to GMAC-CORE and GMAC-MTL configurations only.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> Asynchronous</p>
sbd_dis_receiver_i	In	<p>Sideband MAC Receiver Disable Control</p> <p><b>Function:</b> When set high, this signal instructs the GMAC receiver to stop receiving frames after the completion of any current frame. The GMAC receiver restarts reception only if this signal is reset to low and the MAC Configuration Register Receiver Enable bit [2] is set high.</p> <p>This signal is an optional port applicable to GMAC-CORE and GMAC-MTL configurations only.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> Asynchronous</p>

**Table 4-26 Sideband Signals (Optional) (Continued)**

Signal	I/O	Description
sbd_data_endianess_i	In	<p>Sideband Data Endianness Control</p> <p><b>Function:</b> When set high, this signal configures the DMA to transfer data in big-endian format. When low (by default), the data is transferred in little-endian format. This signal is sampled during active reset (including soft-reset) only and ignored when reset is deasserted.</p> <p>This signal is an optional port applicable to GMAC-DMA and GMAC-AHB configurations only and enabled when Endianness is configured for “BOTH_OPTIONS” (see <a href="#">Table 6-2</a> for parameter details).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i/hclk_i/clk_csr_i (depending on which is the slave interface clock)</p>
sbd_desc_endianess_i	In	<p>Sideband Descriptor Endianness Control</p> <p><b>Function:</b> When set high, this signal configures the DMA to transfer descriptors in reverse endianness of the data format. When low (by default), the descriptors are transferred in the same endian format as the data. This signal is sampled during active reset (including soft-reset) only and ignored when reset is deasserted.</p> <p>This signal is an optional port applicable to GMAC-DMA and GMAC-AHB configurations only and enabled when the Descriptor Endianness is configured to BOTH_OPTIONS (<a href="#">Table 6-2</a>).</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i/hclk_i</p>
ati_txfifoflush_i	In	<p>Tx FIFO Flush</p> <p><b>Function:</b> When there is a pulse on this input, the Tx FIFO is flushed completely. The ati_rdy_o signal is deasserted immediately on sampling a high on this input. When the Tx FIFO Flush operation completes, ati_rdy_o is asserted. The core starts storing a frame only on receiving an SOF after a successful Flush operation. The ati_txfifoflush_i signal should be deasserted after 1 clock cycle.</p> <p>This signal is an optional port applicable to GMAC-MTL configuration.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i</p>

**Table 4-26 Sideband Signals (Optional) (Continued)**

Signal	I/O	Description
ati_chksum_ctrl_i[1:0]	In	<p>Sideband Checksum Offload Control</p> <p><b>Function:</b> These input signals control the insertion of checksums in Ethernet frames that encapsulate TCP, UDP, or ICMP over IPv4 or IPv6, as follows:</p> <ul style="list-style-type: none"> <li>• 2'b00: Do nothing. Checksum engine is bypassed</li> <li>• 2'b01: Insert IPv4 Header checksum. Use this value to insert IPv4 header checksum when the frame encapsulates an IPv4 datagram.</li> <li>• 2'b10: Insert TCP, UDP, or ICMP checksum with pseudo-header checksum available in Checksum field. An IPv4 header checksum is also inserted if the encapsulated datagram is IPv4.</li> <li>• 2'b11: Insert TCP, UDP, or ICMP checksum. This checksum, including pseudo-header bytes, is fully calculated in the Checksum Offload engine. An IPv4 header checksum is also inserted if the encapsulated datagram is IPv4.</li> </ul> <p>This signal is sampled only when ati_val_i and ati_sof_i are high.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_app_i, hclk_i</p>
mac_speed_o[1:0]	Out	<p>Sideband MAC Speed Control</p> <p><b>Function:</b> This signal indicates the 10-, 100-, or 1000-Mbps operating speed and is driven by the GMAC Configuration register's PS (Port Select) and FES (Speed) bits (<a href="#">Table 5-20</a>).</p> <p>Speeds are indicated as follows:</p> <ul style="list-style-type: none"> <li>• 2'b0x 1000 Mbps (GMII)</li> <li>• 2'b10 10 Mbps (MII)</li> <li>• 2'b11 100 Mbps (MII)</li> </ul> <p>You can use this optional signal (selectable during coreKit configuration) to control the clock divider required to generate the GMAC transmit clock (clk_tx_i) when the GMAC is operating with an RMII, SMII, SGMII, or RGMII PHY interface.</p> <p><b>Active state:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_csr_i</p>

## 4.2.27 IEEE 1588 Time Stamp Signals (Optional)

Optional signals added to the core when IEEE 1588 time stamping is enabled during configuration are described in [Table 4-27](#). Some of these signals also depend on the GMAC-UNIV configuration, as described in the table.

**Table 4-27 EEE 1588 Time Stamp Signals (Optional)**

Signal	I/O	Description
clk_ptp_ref_i	In	<p>Reference Clock for the Time Stamp Update Logic</p> <p><b>Function:</b> This is the reference clock, provided by an external oscillator or timing source, that is used for the time stamp logic. See “<a href="#">Frequency Range of Reference Timing Clock</a>” on page <a href="#">39</a> to decide the frequency of clk_ptp_ref_i.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> N/A</p> <p><b>Synchronous to:</b> NA</p>
rst_clk_ptp_ref_i	In	<p>Time Stamp Logic Reset</p> <p><b>Function:</b> Reset signal. This input is not present in GMAC-AHB or GMAC-DMA configurations.</p> <p><b>Active State:</b> Low</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_ptp_ref_i</p>
ptp_timestamp_i[63:0]	In	<p>64-Bit Time Stamp Input</p> <p><b>Function:</b> This bus provides the system time used to time-stamp the specific transmit packets or received frames. This is available only when the External Time Stamp Input Enable option is chosen during configuration</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_ptp_ref_i</p>
ptp_timestamp_o[63:0]	Out	<p>64-Bit Reference Time Stamp Output</p> <p><b>Function:</b> This bus provides the system time stamp output when the time stamp is generated internally. This output pin is available only when the External Time Stamp Input option is not chosen during configuration.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_ptp_ref_i</p>
ptp_pps_o	Out	<p>Pulse Per Second</p> <p><b>Function:</b> This signal is asserted every time the Seconds counter is incremented. This output pin is available only when the External Time Stamp Input option is not chosen during configuration. When Advanced Time-Stamping is enabled, the frequency of this signal is controlled by the PPS Control register.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_ptp_ref_i</p>

**Table 4-27 EEE 1588 Time Stamp Signals (Optional) (Continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
ptp_aux_ts_trig_i	In	<p>Auxiliary Time Stamp Trigger</p> <p><b>Function:</b> This signal is asserted to take an auxiliary snapshot of the time and store it in the auxiliary time stamp FIFO. A rising edge on this port is used to trigger the auxiliary snapshot.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> Asynchronous</p>
mti_ena_timestamp_i	In	<p>Transmit Enable Time Stamping</p> <p><b>Function:</b> When set high, this signal directs the GMAC to capture the time stamp for the transmit frame. This signal only applies in the GMAC-CORE configuration; the application should only assert it for PTP frames that require time stamps. This signal is only sampled when mti_sof_i and mti_val_i are high.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> clk_tx_i</p>
mti_timestamp_o[63:0]	Out	<p>Transmit Frame Time Stamping</p> <p><b>Function:</b> This bus, applicable only for GMAC-CORE configuration, provides the time stamp of the frame transmitted by the GMAC core. This value on this bus is valid only when mti_txstatus_val_o signal is high.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_tx_i</p>
ati_ena_timestamp_i	In	<p>Transmit Enable Time Stamping</p> <p><b>Function:</b> When set high, instructs the GMAC core to capture the time stamp for the transmit frame. This signal is applies only in GMAC-MTL configuration; the application should only assert it for PTP frames that require time stamping. This signal is only sampled when ati_sof_i and ati_val_i are high.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>
ati_timestamp_o[63:0]	Out	<p>Transmit Frame Time Stamp</p> <p><b>Function:</b> This bus provides the time stamp of the transmitted frame, which frame's status is available on the ati_txstatus_o bus. This value on this bus is thus qualified by ati_txstatus_val_o signal. This signal applies only in the GMAC-MTL configuration.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>

**Table 4-27 EEE 1588 Time Stamp Signals (Optional) (Continued)**

Signal	I/O	Description
mri_timestamp_o[63:0]	Out	<p>Receive Time Stamp Value</p> <p><b>Function:</b> This bus provides the received frame's time stamp. This signal only applies in GMAC-CORE configuration, and is valid only when mri_eof_o asserted.</p> <p><b>Active State:</b> N/A</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_rx_i</p>
ari_timestamp_val_o	Out	<p>Receive Time Stamp Valid</p> <p><b>Function:</b> When asserted, this signal indicates that a valid time stamp value is available on ARI data bus (ari_data_o). This signal is available only in GMAC-MTL configuration.</p> <p>In 32-bit configuration, this signal is asserted for the two clock cycles required to transfer two 32-bit words. The lower 32 bits of the time stamp (the Nanoseconds field) is given first on ari_data_o, followed by the upper word (the Seconds field).</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> Yes</p> <p><b>Synchronous to:</b> clk_app_i</p>

#### 4.2.28 Test Mode

**Table 4-28 Test Mode Signal**

Signal	I/O	Description
test_mode		<p>Test Mode</p> <p><b>Function:</b> This input (present only in GMAC-AHB and GMAC-DMA configurations) enables bypassing of the internally generated resets and directly connecting these resets to the external reset input (hreset_n or pwr_on_rst_n respectively). This input should normally be tied to zero and set only during scan testing.</p> <p><b>Active State:</b> High</p> <p><b>Registered:</b> No</p> <p><b>Synchronous to:</b> N/A</p>



## 5

# Registers

## 5.1 Register Maps

The register maps in this section provide high-level summaries of each register or group of registers. The register name in the register maps (tables) are cross-referenced to the detailed register descriptions in “Register Descriptions” on page 256 (double-click on the register name to link to the detailed description).

### 5.1.1 DMA Register Map



**Note** Register 6 (Operation Mode register) is also valid and present in the GMAC-MTL configuration.

**Table 5-1 DMA Register Map**

Register No.	Offset Address	Register Name and Description
0	0x1000	Register 0 (Bus Mode Register) Controls the Host Interface Mode.
1	0x1004	Register 1 (Transmit Poll Demand Register) Used by the host to instruct the DMA to poll the Transmit Descriptor List.
2	0x1008	Register 2 (Receive Poll Demand Register) Used by the Host to instruct the DMA to poll the Receive Descriptor list.
3	0x100C	Register 3 (Receive Descriptor List Address Register) Points the DMA to the start of the Receive Descriptor list.
4	0x1010	Register 4 (Transmit Descriptor List Address Register) Points the DMA to the start of the Transmit Descriptor List.
5	0x1014	Register 5 (Status Register) The Software driver (application) reads this register during interrupt service routine or polling to determine the status of the DMA.
6	0x1018	Register 6 (Operation Mode Register) Establishes the Receive and Transmit operating modes and command.

**Table 5-1 DMA Register Map (Continued)**

Register No.	Offset Address	Register Name and Description
7	0x101C	<a href="#">Register 7 (Interrupt Enable Register)</a> Enables the interrupts reported by the Status Register.
8	0x1020	<a href="#">Register 8 (Missed Frame and Buffer Overflow Counter Register)</a> Contains the counters for discarded frames because no host Receive Descriptor was available, and discarded frames because of Receive FIFO Overflow.
9	0x1024	<a href="#">Register 9 (Receive Interrupt Watchdog Timer Register)</a> Watchdog time-out for Receive Interrupt (RI) from DMA
10	0x1028	<a href="#">Register 10 (AXI Bus Mode Register)</a> Controls AXI Master behavior (mainly controls burst splitting and number of outstanding requests).
11	0x102C	<a href="#">Register 11 (AXI Status Register)</a> Gives the idle status of the AXI master's read/write channels.
12–17	0x1030–0x1044	Reserved
18	0x1048	<a href="#">Register 18 (Current Host Transmit Descriptor Register)</a> Points to the start of current Transmit Descriptor read by the DMA.
19	0x104C	<a href="#">Register 19 (Current Host Receive Descriptor Register)</a> Points to the start of current Receive Descriptor read by the DMA.
20	0x1050	<a href="#">Register 20 (Current Host Transmit Buffer Address Register)</a> Points to the current Transmit Buffer address read by the DMA.
21	0x1054	<a href="#">Register 21 (Current Host Receive Buffer Address Register)</a> Points to the current Receive Buffer address read by the DMA.
22	0x1058	<a href="#">Register 22 (HW Feature Register)</a> Indicates the presence of the optional features of the core.

### 5.1.2 GMAC Register Map

Table 5-2 provides the address map of the GMAC core registers. Most of the registers are optional and present in the source code only if selected during coreKit configuration. If a register is not configured, then that address is reserved.

**Table 5-2 GMAC Register Map**

Register No.	Offset Address	Register Name and Description
0	0x0000	<a href="#">Register 0 (MAC Configuration Register)</a> This is the operation mode register for the MAC.

**Table 5-2 GMAC Register Map (Continued)**

Register No.	Offset Address	Register Name and Description
1	0x0004	<b>Register 1 (MAC Frame Filter)</b> Contains the frame filtering controls.
2	0x0008	<b>Register 2 (Hash Table High Register)</b> Contains the higher 32 bits of the Multicast Hash table. This register is present only when the Hash filter function is selected in coreConsultant. (See <a href="#">Table 6-9</a> .)
3	0x000C	<b>Register 3 (Hash Table Low Register)</b> Contains the lower 32 bits of the Multicast Hash table. This register is present only when the Hash filter function is selected in coreConsultant. (See <a href="#">Table 6-9</a> .)
4	0x0010	<b>Register 4 (GMII Address Register)</b> Controls the management cycles to an external PHY. This register is present only when the Station Management (MDIO) feature is selected in coreConsultant. (See <a href="#">Table 6-22</a> .)
5	0x0014	<b>Register 5 (GMII Data Register)</b> Contains the data to be written to or read from the PHY register. This register is present only when the Station Management (MDIO) feature is selected in coreConsultant. (See <a href="#">Table 6-22</a> .)
6	0x0018	<b>Register 6 (Flow Control Register)</b> Controls the generation of control frames.
7	0x001C	<b>Register 7 (VLAN Tag Register)</b> Identifies IEEE 802.1Q VLAN type frames.
8	0x0020	<b>Register 8 (Version Register)</b> Identifies the version of the Core
9	0x0024	<b>Register 9 (Debug Register)</b> Gives the status of various internal blocks for debugging
10	0x0028	Remote Wake-Up Frame Filter This is the address through which the remote Wake-up Frame Filter registers (wkupfmfilter_reg) are written/read by the Application. wkupfmfilter_reg is actually a pointer to eight (not transparent) such wkupfmfilter_reg registers. Eight sequential Writes to this address (028) will write all wkupfmfilter_reg registers. Eight sequential Reads from this address (028) will read all wkupfmfilter_reg registers. This register contains the higher 16 bits of the 7th MAC address. This register is present only when the PMT module Remote Wake-up feature is selected in coreConsultant. (See <a href="#">Table 6-16</a> .) Refer to “ <a href="#">Remote Wake-Up Frame Filter Register</a> ” on page <a href="#">148</a> for additional information.
11	0x002C	PMT Control and Status This register is present only when the PMT module is selected in coreConsultant. (See <a href="#">Table 6-16</a> .) Refer to “ <a href="#">PMT Control and Status Register</a> ” on page <a href="#">147</a> for additional information.

**Table 5-2 GMAC Register Map (Continued)**

Register No.	Offset Address	Register Name and Description
12–13	0x0030–0x0034	Reserved
14	0x0038	<a href="#">Register 14 (Interrupt Status Register)</a> Contains the interrupt status.
15	0x003C	<a href="#">Register 15 (Interrupt Mask Register)</a> Contains the masks for generating the interrupts.
16	0x0040	<a href="#">Register 16 (MAC Address0 High Register)</a> Contains the higher 16 bits of the first MAC address.
17	0x0044	<a href="#">Register 17 (MAC Address0 Low Register)</a> Contains the lower 32 bits of the first MAC address.
18	0x0048	<a href="#">Register 18 (MAC Address1 High Register)</a> Contains the higher 16 bits of the second MAC address. This register is present only when Enable MAC Address1 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
19	0x004C	<a href="#">Register 19 (MAC Address1 Low Register)</a> Contains the lower 32 bits of the second MAC address. This register is present only when Enable MAC Address1 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
20	0x0050	MAC Address2 High Register Contains the lower 32 bits of the third MAC address. This register is present only when Enable MAC Address2 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
21	0x0054	MAC Address2 Low Register Contains the lower 32 bits of the third MAC address. This register is present only when Enable MAC Address2 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
22	0x0058	MAC Address3 High Register Contains the higher 16 bits of the fourth MAC address. This register is present only when Enable MAC Address3 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
23	0x005C	MAC Address3 Low Register Contains the lower 32 bits of the fourth MAC address. This register is present only when Enable MAC Address3 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
24	0x0060	MAC Address4 High Register Contains the higher 16 bits of the fifth MAC address. This register is present only when Enable MAC Address4 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
25	0x0064	MAC Address4 Low Register Contains the lower 32 bits of the fifth MAC address. This register is present only when Enable MAC Address4 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
26	0x0068	MAC Address5 High Register Contains the higher 16 bits of the sixth MAC address. This register is present only when Enable MAC Address5 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).

**Table 5-2 GMAC Register Map (Continued)**

<b>Register No.</b>	<b>Offset Address</b>	<b>Register Name and Description</b>
27	0x006C	MAC Address5 Low Register Contains the lower 32 bits of the sixth MAC address. This register is present only when Enable MAC Address5 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
28	0x0070	MAC Address6 High Register Contains the higher 16 bits of the seventh MAC address. This register is present only when Enable MAC Address6 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
29	0x0074	MAC Address6 Low Register Contains the lower 32 bits of the seventh MAC address. This register is present only when Enable MAC Address6 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
30	0x0078	MAC Address7 High Register Contains the higher 16 bits of the eighth MAC address. This register is present only when Enable MAC Address7 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
31	0x007C	MAC Address7 Low Register Contains the lower 32 bits of the eighth MAC address. This register is present only when Enable MAC Address7 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
32	0x0080	MAC Address8 High Register Contains the higher 16 bits of the ninth MAC address. This register is present only when Enable MAC Address8 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
33	0x0084	MAC Address8 Low Register Contains the lower 32 bits of the ninth MAC address. This register is present only when Enable MAC Address8 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
34	0x0088	MAC Address9 High Register Contains the higher 16 bits of the tenth MAC address. This register is present only when Enable MAC Address9 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
35	0x008C	MAC Address9 Low Register Contains the lower 32 bits of the tenth MAC address. This register is present only when Enable MAC Address9 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
36	0x0090	MAC Address10 High Register Contains the higher 16 bits of the eleventh MAC address. This register is present only when Enable MAC Address10 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
37	0x0094	MAC Address10 Low Register Contains the lower 32 bits of the eleventh MAC address. This register is present only when Enable MAC Address10 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
38	0x0098	MAC Address11 High Register This register contains the higher 16 bits of the twelfth MAC address. This register is present only when Enable MAC Address11 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).

**Table 5-2 GMAC Register Map (Continued)**

<b>Register No.</b>	<b>Offset Address</b>	<b>Register Name and Description</b>
39	0x009C	MAC Address11 Low Register Contains the lower 32 bits of the twelfth MAC address. This register is present only when Enable MAC Address11 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
40	0x00A0	MAC Address12 High Register Contains the higher 16 bits of the thirteenth MAC address. This register is present only when Enable MAC Address12 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
41	0x00A4	MAC Address12 Low Register Contains the lower 32 bits of the thirteenth MAC address. This register is present only when Enable MAC Address12 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
42	0x00A8	MAC Address13 High Register Contains the higher 16 bits of the fourteenth MAC address. This register is present only when Enable MAC Address13 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
43	0x00AC	MAC Address13 Low Register Contains the lower 32 bits of the fourteenth MAC address. This register is present only when Enable MAC Address13 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
44	0x00B0	MAC Address14 High Register Contains the higher 16 bits of the fifteenth MAC address. This register is present only when Enable MAC Address14 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
45	0x00B4	MAC Address14 Low Register Contains the lower 32 bits of the fifteenth MAC address. This register is present only when Enable MAC Address14 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
46	0x00B8	MAC Address15 High Register Contains the higher 16 bits of the sixteenth MAC address. This register is present only when Enable MAC Address15 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
47	0x00BC	MAC Address15 Low Register Contains the lower 32 bits of the sixteenth MAC address. This register is present only when Enable MAC Address15 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
48	0x00C0	<a href="#">Register 48 (AN Control Register)</a> Enables and/or restarts auto-negotiation. It also enables PCS loopback.
49	0x00C4	<a href="#">Register 49 (AN Status Register)</a> Indicates the link and auto-negotiation status.
50	0x00C8	<a href="#">Register 50 (Auto-Negotiation Advertisement Register)</a> This register is configured before auto-negotiation begins. It contains the advertised ability of the GMAC.
51	0x00CC	<a href="#">Register 51 (Auto-Negotiation Link Partner Ability Register)</a> Contains the advertised ability of the link partner. Its value is valid after successful completion of auto-negotiation or when a new base page has been received (indicated in the Auto-Negotiation Expansion Register).

**Table 5-2 GMAC Register Map (Continued)**

Register No.	Offset Address	Register Name and Description
52	0x00D0	<a href="#">Register 52 (Auto-Negotiation Expansion Register)</a> Indicates whether a new base page has been received from the link partner.
53	0x00D4	<a href="#">Register 53 (TBI Extended Status Register)</a> Indicates all modes of operation of the GMAC.
54	0x00D8	<a href="#">Register 54 (SGMII/RGMII/SMII Status Register)</a> Indicates the status signals received from the PHY through the SGMII/RGMII/SMII interface.
55–63	0x00DC–0x00FC	Reserved
64–191	0x0100–0x02FC	MMC Register Map See <a href="#">Table 3-16</a> .
192–447	0x0300–0x06FC	Reserved
448	0x0700	<a href="#">Register 448 (Time Stamp Control Register)</a> Controls the time stamp generation and update logic. This register is only present when IEEE1588 time stamping is enabled during coreKit configuration.
449	0x0704	<a href="#">Register 449 (Sub-Second Increment Register)</a> Contains the 8-bit value by which the Sub-Second register is incremented. This register is only present when IEEE1588 time stamping is enabled without an external time stamp input.
450	0x0708	<a href="#">Register 450 (System Time - Seconds Register)</a> Contains the lower 32 bits of the seconds field of the system time. This register is only present when IEEE1588 time stamping is enabled without an external time stamp input.
451	0x070C	<a href="#">Register 451 (System Time - Nanoseconds Register)</a> Contains 32 bits of the nano-seconds field of the system time. This register is only present when IEEE1588 time stamping is enabled without an external time stamp input.
452	0x0710	<a href="#">Register 452 (System Time - Seconds Update Register)</a> Contains the lower 32 bits of the seconds field to be written to, added to, or subtracted from the System Time value. This register is only present when IEEE1588 time stamping is enabled without an external time stamp input.
453	0x0714	<a href="#">Register 453 (System Time - Nanoseconds Update Register)</a> Contains 32 bits of the nano-seconds field to be written to, added to, or subtracted from the System Time value.. This register is only present when IEEE1588 time stamping is enabled without an external time stamp input.

**Table 5-2 GMAC Register Map (Continued)**

Register No.	Offset Address	Register Name and Description
454	0x0718	<b>Register 454 (Time Stamp Addend Register)</b> This register is used by the software to readjust the clock frequency linearly to match the master clock frequency. This register is only present when IEEE1588 time stamping is enabled without an external time stamp input.
455	0x071C	<b>Register 455 (Target Time Seconds Register)</b> Contains the higher 32 bits of time to be compared with the system time for interrupt event generation. This register is only present when IEEE1588 time stamping is enabled without an external time stamp input.
456	0x0720	<b>Register 456 (Target Time Nanoseconds Register)</b> Contains the lower 32 bits of time to be compared with the system time for interrupt event generation. This register is only present when IEEE1588 time stamping is enabled without an external time stamp input.
457	0x0724	<b>Register 457 (System Time - Higher Word Seconds Register)</b> Contains the most significant 16-bits of the time stamp seconds value. This register is optional and can be selected using the coreKit parameter identified in “ <a href="#">IEEE 1588 Time Stamp Block</a> ” on page <a href="#">320</a> .
458	0x0728	<b>Register 458 (Time Stamp Status Register)</b> Contains the PTP status. This register is available only when the advanced IEEE 1588 timestamp feature is selected.
459	0x072C	<b>Register 459 (PPS Control Register)</b> This register is used to control the interval of the PPS signal output, such that a duration of less than 1 second can be achieved between pulses.
460	0x0730	<b>Register 460 (Auxiliary Time Stamp - Nanoseconds Register)</b> Contains the lower 32 bits (nano-seconds field) of the auxiliary time stamp register.
461	0x0731	<b>Register 461 (Auxiliary Time Stamp - Seconds Register)</b> Contains the lower 32 bits of the Seconds field of the auxiliary time stamp register.
462–511	0x0738–0x7FC	Reserved
512	0x0800	<b>MAC Address 16 High Register</b> Contains the higher 16 bits of the seventeenth MAC address. This register is present only when Enable MAC Address16 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
513	0x0804	<b>MAC Address 16 Low Register</b> Contains the lower 32 bits of the seventeenth MAC address. This register is present only when Enable MAC Address16 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).

**Table 5-2 GMAC Register Map (Continued)**

<b>Register No.</b>	<b>Offset Address</b>	<b>Register Name and Description</b>
514	0x0808	MAC Address 17 High Register Contains the higher 16 bits of the eighteenth MAC address. This register is present only when Enable MAC Address17 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
.....		
543	0x087C	MAC Address 31 Low Register Contains the lower 32 bits of the thirty-second MAC address. This register is present only when Enable MAC Address 31 is selected in coreConsultant. (See <a href="#">Table 6-8</a> ).
.....		Reserved
	0x0FFC	Reserved

## 5.2 Register Descriptions

The Access column of each register description that follows specifies how the application and the core can access the register fields of the CSRs.

The Access column uses the following conventions:

Read Only (RO)	Register field can only be read by the application. Writes to read-only fields have no effect.
Write Only (WO)	Register field can only be written by the application.
Read and Write (R_W)	Register field can be read and written by the application. The application can set this field by writing 1'b1 and can clear it by writing 1'b0.
Read, Write, and Self Clear (R_W_SC)	Register field can be read and written by the application (Read and Write), and is cleared to 1'b0 by the core (Self Clear). The conditions under which the core clears this field are explained in detail in the field's description.
Read, Self Set, and Write Clear (R_SS_WC)	Register field can be read by the application (Read), can be set to 1'b1 by the core on a certain internal event (Self Set), and can be cleared to 1'b0 by the application with a register write of 1'b1 (Write Clear). A register write of 1'b0 has no effect on this field. The conditions under which the core sets this field are explained in detail in the field's description. (Example, interrupt bits.)
Read, Write Set, and Self Clear (R_WS_SC)	Register field can be read by the application (Read), can be set to 1'b1 by the application with a register write of 1'b1 (Write Set), and is cleared to 1'b0 by the core (Self Clear). The application cannot clear this type of field, and a register write of 1'b0 to this bit has no effect on this field. The conditions under which the core clears this field are explained in detail in the field's description. (Example, reset signals)
Read, Self Set, and Self Clear or Write Clear (R_SS_SC_WC)	Register field can be read by the application (Read), can be set to 1'b1 by the core on a certain internal event (Self Set), and can be cleared to 1'b0 either by the core itself (Self Clear) or by the application with a register write of 1'b0 (Write Clear). A register write of 1'b1 to this bit has no effect on this field. The conditions under which the core sets or clears this field are explained in detail in the field's description.
Read Only and Write Trigger (RO_WT)	Register field can be read by the application, and when a write operation is performed with any data value, an event is triggered, as explained in the field's description. (Example: Tx Poll Demand register)
Read, Self Set, and Read Clear (R_SS_RC)	Register field can be read by the application (Read), can be set to 1'b1 by the core on a certain internal event (Self Set), and is automatically cleared to 1'b0 on a register read. A register write of 1'b0 has no effect on this field. The conditions under which the core sets this field are explained in detail in the field's description. (Example: Overflow counter.)



- All CSRs are implemented as 32-bit registers and can be accessed in 1 read/write cycle with a 32-bit MCI interface (or 32-bit AHB slave port). If you initiate a non-32-bit register access, the byte enable signals (`mci_be_i[3:0]`) qualify the corresponding register bytes. Only write operations are qualified with byte enables, while read operations are always 32-bit, unless the register is affected by a read operation. Byte enable signals qualify reads to such registers as, for example, the MMC counter registers.
- Register bits[7:0] correspond to address[1:0] = 00 in Little Endian mode, while register bits[31:24] correspond to address[1:0] = 00 in Big Endian mode.
- The transfer of particular register contents (e.g., MAC DA address register) to the Transmit or Receive clock domains are initiated when a particular byte is written. This is indicated in the register descriptions, where applicable.
- When any Register content is being transferred to a different clock domain after a write operation, there should not be any further writes to the same location until the first write is updated. Otherwise, the second write operation will not get updated to the destination clock domain. Thus the delay between two writes to the same register location should be at least 4 cycles of the destination clock (PHY receive clock or PHY transmit clock).

## 5.2.1 DMA Register Description

This section defines the bits for each DMA register. The write data inputs to the DMA registers are qualified with the corresponding `mci_be_i` signal inputs (MCI interface). Thus, non-32 bit accesses are allowed as long as the address is Word-aligned. For the APB interface, only 32-bit accesses are possible, while for AHB slave interfaces, byte, half-word or word accesses are possible.

### 5.2.1.1 Register 0 (Bus Mode Register)

The Bus Mode register establishes the bus operating modes for the DMA.

**Table 5-3 Register 0 (Bus Mode Register)**

Field	Description	Reset	Access
31:27	Reserved	00H	RO
26	MB: Mixed Burst  When this bit is set high and FB bit is low, the AHB master interface will start all bursts of length more than 16 with INCR (undefined burst) whereas it will revert to fixed burst transfers (INCRx and SINGLE) for burst-length of 16 and below.  This bit is valid only in GMAC-AHB configuration and reserved in all others.	0	R_W
25	AAL: Address-Aligned Beats  When this bit is set high and the FB bit equals 1, the AHB/AXI interface generates all bursts aligned to the start address LS bits. If the FB bit equals 0, the first burst (accessing the data buffer's start address) is not aligned, but subsequent bursts are aligned to the address.  This bit is valid only in GMAC-AHB/AXI configuration, and reserved (RO with default value 0) in all other configurations.	0	R_W
24	8xPBL Mode  When set high, this bit multiplies the PBL value programmed (bits [22:17] and bits [13:8]) eight times. Thus the DMA will transfer data in to a maximum of 8, 16, 32, 64, 128, and 256 beats depending on the PBL value.  <b>Note:</b> This bit function is not backward compatible. Before version 3.50a, this bit was 4xPBL.	0	R_W

**Table 5-3 Register 0 (Bus Mode Register) (Continued)**

Field	Description	Reset	Access
23	USP: Use Separate PBL When set high, it configures the RxDMA to use the value configured in bits [22:17] as PBL while the PBL value in bits [13:8] is applicable to TxDMA operations only. When reset to low, the PBL value in bits [13:8] is applicable for both DMA engines.	0	R_W
22:17	RPBL: RxDMA PBL These bits indicate the maximum number of beats to be transferred in one RxDMA transaction. This will be the maximum value that is used in a single block Read/Write. The RxDMA will always attempt to burst as specified in RPBL each time it starts a Burst transfer on the host bus. RPBL can be programmed with permissible values of 1, 2, 4, 8, 16, and 32. Any other value will result in undefined behavior. These bits are valid and applicable only when USP is set high.	01H	R_W
16	FB: Fixed Burst This bit controls whether the AHB/AXI Master interface performs fixed burst transfers or not. When set, the AHB will use only SINGLE, INCR4, INCR8 or INCR16 during start of normal burst transfers. When reset, the AHB/AXI will use SINGLE and INCR burst transfer operations. In GMAC-AXI configuration, refer to Bit 0 (UNDEF) of AXI Bus Mode register for detailed description.	0	R_W
15:14	PR: Rx:Tx priority ratio. RxDMA requests given priority over TxDMA requests in the following ratio. This is valid only when the DA bit is reset. <ul style="list-style-type: none"><li>• 00: 1:1</li><li>• 01: 2:1</li><li>• 10: 3:1</li><li>• 11: 4:1</li></ul> These bits are reserved and RO in GMAC-AXI configuration.	00	R_W

**Table 5-3 Register 0 (Bus Mode Register) (Continued)**

Field	Description	Reset	Access																					
13:8	<p>PBL: Programmable Burst Length</p> <p>These bits indicate the maximum number of beats to be transferred in one DMA transaction. This will be the maximum value that is used in a single block Read/Write. The DMA will always attempt to burst as specified in PBL each time it starts a Burst transfer on the host bus. PBL can be programmed with permissible values of 1, 2, 4, 8, 16, and 32. Any other value will result in undefined behavior. When USP is set high, this PBL value is applicable for TxDMA transactions only.</p> <p>The PBL values have the following limitations.</p> <p>The maximum number of beats (PBL) possible is limited by the size of the Tx FIFO and Rx FIFO in the MTL layer and the data bus width on the DMA. The FIFO has a constraint that the maximum beat supported is half the depth of the FIFO, except when specified (as given below). For different data bus widths and FIFO sizes, the valid PBL range (including x8 mode) is provided in the following table. If the PBL is common for both transmit and receive DMA, the minimum Rx FIFO and Tx FIFO depths must be considered. Do not program out-of-range PBL values, because the system may not behave properly.</p>	01H	R_W																					
<b>Data Bus Width</b> <b>FIFO Depth</b> <b>Valid PBL Range in Full Duplex Mode</b> <b>Valid PBL Range (for TxFIFO only) in Half Duplex Mode</b>																								
32	<table> <tr> <td>128 bytes</td><td>16 or less</td><td>8 or less (10/100 Mbps only)</td></tr> <tr> <td>256 bytes</td><td>32 or less</td><td>32 or less (10/100 Mbps only)</td></tr> <tr> <td>512 bytes</td><td>64 or less</td><td>64 or less (10/100 Mbps only)</td></tr> <tr> <td>1 KB</td><td>128 or less</td><td>128 or less in 10/100 Mbps, 64 or less in 1000-Mbps,</td></tr> <tr> <td>2 KB and above</td><td>All</td><td>All</td></tr> </table>	128 bytes	16 or less	8 or less (10/100 Mbps only)	256 bytes	32 or less	32 or less (10/100 Mbps only)	512 bytes	64 or less	64 or less (10/100 Mbps only)	1 KB	128 or less	128 or less in 10/100 Mbps, 64 or less in 1000-Mbps,	2 KB and above	All	All								
128 bytes	16 or less	8 or less (10/100 Mbps only)																						
256 bytes	32 or less	32 or less (10/100 Mbps only)																						
512 bytes	64 or less	64 or less (10/100 Mbps only)																						
1 KB	128 or less	128 or less in 10/100 Mbps, 64 or less in 1000-Mbps,																						
2 KB and above	All	All																						
64	<table> <tr> <td>128 bytes</td><td>8 or less</td><td>4 or less (10/100 Mbps only)</td></tr> <tr> <td>256 bytes</td><td>16 or less</td><td>16 or less(10/100 Mbps only)</td></tr> <tr> <td>512 bytes</td><td>32 or less</td><td>32 or less (10/100 Mbps only)</td></tr> <tr> <td>1 KB</td><td>64 or less</td><td>64 in 10/100-Mbps operation, 32 or less in 1000-Mbps.</td></tr> <tr> <td>2 KB</td><td>128 or less</td><td>128 or less</td></tr> <tr> <td>4 KB and above</td><td>All</td><td>All</td></tr> </table>	128 bytes	8 or less	4 or less (10/100 Mbps only)	256 bytes	16 or less	16 or less(10/100 Mbps only)	512 bytes	32 or less	32 or less (10/100 Mbps only)	1 KB	64 or less	64 in 10/100-Mbps operation, 32 or less in 1000-Mbps.	2 KB	128 or less	128 or less	4 KB and above	All	All					
128 bytes	8 or less	4 or less (10/100 Mbps only)																						
256 bytes	16 or less	16 or less(10/100 Mbps only)																						
512 bytes	32 or less	32 or less (10/100 Mbps only)																						
1 KB	64 or less	64 in 10/100-Mbps operation, 32 or less in 1000-Mbps.																						
2 KB	128 or less	128 or less																						
4 KB and above	All	All																						
128	<table> <tr> <td>128 bytes</td><td>4 or less</td><td>2 or less(10/100 Mbps only)</td></tr> <tr> <td>256 bytes</td><td>8 or less</td><td>8 or less(10/100 Mbps only)</td></tr> <tr> <td>512 bytes</td><td>16 or less</td><td>16 or less(10/100 Mbps only)</td></tr> <tr> <td>1KB</td><td>32 or less</td><td>32 in 10/100-Mbps operation, 16 or less in 1000-Mbps.</td></tr> <tr> <td>2KB</td><td>64 or less</td><td>64 or less</td></tr> <tr> <td>4KB</td><td>128 or less</td><td>128 or less</td></tr> <tr> <td>8KB and above</td><td>All</td><td>All</td></tr> </table>	128 bytes	4 or less	2 or less(10/100 Mbps only)	256 bytes	8 or less	8 or less(10/100 Mbps only)	512 bytes	16 or less	16 or less(10/100 Mbps only)	1KB	32 or less	32 in 10/100-Mbps operation, 16 or less in 1000-Mbps.	2KB	64 or less	64 or less	4KB	128 or less	128 or less	8KB and above	All	All		
128 bytes	4 or less	2 or less(10/100 Mbps only)																						
256 bytes	8 or less	8 or less(10/100 Mbps only)																						
512 bytes	16 or less	16 or less(10/100 Mbps only)																						
1KB	32 or less	32 in 10/100-Mbps operation, 16 or less in 1000-Mbps.																						
2KB	64 or less	64 or less																						
4KB	128 or less	128 or less																						
8KB and above	All	All																						

**Table 5-3 Register 0 (Bus Mode Register) (Continued)**

Field	Description	Reset	Access
7	ATDS: Alternate Descriptor Size When set, the alternate descriptor (described in “ <a href="#">Alternate or Enhanced Descriptors</a> ” on page 356) size is increased to 32bytes (8 DWORDS). This is required when the Advanced Time-Stamp feature or Full IPC Offload Engine is enabled in the receiver. When reset, the descriptor size reverts back to 4 DWORDs (16 bytes). This bit is present only when Alternate Descriptor feature is selected and either Advanced Time Stamp or IPC Full Checksum Offload (type 2) feature is selected during configuration. Otherwise, this bit is reserved and read-only.	0	RW
6:2	DSL: Descriptor Skip Length This bit specifies the number of Word/Dword/Lword (depending on 32/64/128-bit bus) to skip between two unchained descriptors. The address skipping starts from the end of current descriptor to the start of next descriptor. When DSL value equals zero, then the descriptor table is taken as contiguous by the DMA, in Ring mode.	00H	R_W
1	DA: DMA Arbitration scheme <ul style="list-style-type: none"><li>• 0: Round-robin with Rx:Tx priority given in bits [15:14]</li><li>• 1: Rx has priority over Tx</li></ul> This bit is reserved and RO in GMAC-AXI configuration.	0	R_W
0	SWR: Software Reset When this bit is set, the MAC DMA Controller resets all GMAC Subsystem internal registers and logic. It is cleared automatically after the reset operation has completed in all of the core clock domains. Read a 0 value in this bit before re-programming any register of the core. <b>Note:</b> The reset operation is completed only when all the resets in all the active clock domains are de-asserted. Hence it is essential that all the PHY inputs clocks (applicable for the selected PHY interface) are present for software reset completion.	1	R_WS_SC

### 5.2.1.2 Register 1 (Transmit Poll Demand Register)

The Transmit Poll Demand register enables the Transmit DMA to check whether or not the current descriptor is owned by DMA. The Transmit Poll Demand command is given to wake up the TxDMA if it is in Suspend mode. The TxDMA can go into Suspend mode due to an Underflow error in a transmitted frame or due to the unavailability of descriptors owned by Transmit DMA. You can give this command anytime and the TxDMA will reset this command once it starts re-fetching the current descriptor from host memory.

**Table 5-4 Register 1 (Transmit Poll Demand Register)**

Field	Description	Reset	Access
31:0	TPD: Transmit Poll Demand When these bits are written with any value, the DMA reads the current descriptor pointed to by Register 18. If that descriptor is not available (owned by Host), transmission returns to the Suspend state and DMA Register 5[2] is asserted. If the descriptor is available, transmission resumes.	0000_0000H	RO_WT

### 5.2.1.3 Register 2 (Receive Poll Demand Register)

The Receive Poll Demand register enables the receive DMA to check for new descriptors. This command is given to wake up the RxDMA from SUSPEND state. The RxDMA can go into SUSPEND state only due to the unavailability of descriptors owned by it.

**Table 5-5 Register 2 (Receive Poll Demand Register)**

Field	Description	Reset	Access
31:0	RPD: Receive Poll Demand When these bits are written with any value, the DMA reads the current descriptor pointed to by Register 19. If that descriptor is not available (owned by Host), reception returns to the Suspended state and Register 5[7] is not asserted. If the descriptor is available, the Receive DMA returns to active state.	0000_0000H	RO_WT

### 5.2.1.4 Register 3 (Receive Descriptor List Address Register)

The Receive Descriptor List Address register points to the start of the Receive Descriptor List. The descriptor lists reside in the host's physical memory space and must be Word/Dword/Lword-aligned (for 32/64/128-bit data bus). The DMA internally converts it to bus width aligned address by making the corresponding LS bits low. Writing to Register 3 is permitted only when reception is stopped. When stopped, Register 3 must be written to before the receive Start command is given.

**Table 5-6 Register 3 (Receive Descriptor List Address Register)**

Field	Description	Reset	Access
31:0	Start of Receive List This field contains the base address of the First Descriptor in the Receive Descriptor list. The LSB bits [1/2/3:0] for 32/64/128-bit bus width) will be ignored and taken as all-zero by the DMA internally. Hence these LSB bits are Read Only.	0000_0000H	R_W

### 5.2.1.5 Register 4 (Transmit Descriptor List Address Register)

The Transmit Descriptor List Address register points to the start of the Transmit Descriptor List. The descriptor lists reside in the host's physical memory space and must be Word/DWORD/LWORD-aligned (for 32/64/128-bit data bus). The DMA internally converts it to bus width aligned address by making the corresponding LSB to low. Writing to Register 4 is permitted only when transmission has stopped. When stopped, Register 4 can be written before the transmission Start command is given.

**Table 5-7 Register 4 (Transmit Descriptor List Address Register)**

Field	Description	Reset	Access
31:0	Start of Transmit List This field contains the base address of the First Descriptor in the Transmit Descriptor list. The LSB bits [1/2/3:0] for 32/64/128-bit bus width) will be ignored and taken as all-zero by the DMA internally. Hence these LSB bits are Read Only.	0000_0000H	R_W

### 5.2.1.6 Register 5 (Status Register)

The Status register contains all the status bits that the DMA reports to the host. Register 5 and is usually read by the Software driver during an interrupt service routine or polling. Most of the fields in this register cause the host to be interrupted. Register 5 bits are not cleared when read. Writing 1'b1 to (unreserved) bits in Register 5[16:0] clears them and writing 1'b0 has no effect. Each field (bits[16:0]) can be masked by masking the appropriate bit in Register 7.

**Table 5-8 Register 5 (Status Register)**

Field	Description	Reset	Access
31:30	Reserved	0H	RO
29	TTI: Time-Stamp Trigger Interrupt This bit indicates an interrupt event in the GMAC core's Time Stamp Generator block. The software must read the corresponding registers in the GMAC core to get the exact cause of interrupt and clear its source to reset this bit to 1'b0. When this bit is high, the interrupt signal from the GMAC subsystem (sbd_intr_o) is high.	0	RO
28	GPI: GMAC PMT Interrupt This bit indicates an interrupt event in the GMAC core's PMT module. The software must read the corresponding registers in the GMAC core to get the exact cause of interrupt and clear its source to reset this bit to 1'b0. The interrupt signal from the GMAC subsystem (sbd_intr_o) is high when this bit is high.	0	RO
27	GMI: GMAC MMC Interrupt This bit reflects an interrupt event in the MMC module of the GMAC core. The software must read the corresponding registers in the GMAC core to get the exact cause of interrupt and clear the source of interrupt to make this bit as 1'b0. The interrupt signal from the GMAC subsystem (sbd_intr_o) is high when this bit is high.	0	RO
26	GLI: GMAC Line interface Interrupt This bit reflects an interrupt event in the GMAC Core's PCS or RGMII interface block. The software must read the corresponding registers in the GMAC core to get the exact cause of interrupt and clear the source of interrupt to make this bit as 1'b0. The interrupt signal from the GMAC subsystem (sbd_intr_o) is high when this bit is high.	0	RO
25:23	EB: Error Bits These bits indicate the type of error that caused a Bus Error (e.g., error response on the AHB/AXI interface). Valid only with Fatal Bus Error bit (Register 5[13]) set. This field does not generate an interrupt. Bit 23      1'b1      Error during data transfer by TxDMA 1'b0      Error during data transfer by RxDMA Bit 24      1'b1      Error during read transfer 1'b0      Error during write transfer Bit 25      1'b1      Error during descriptor access 1'b0      Error during data buffer access	000	RO

**Table 5-8 Register 5 (Status Register) (Continued)**

Field	Description	Reset	Access
22:20	<p>TS: Transmit Process State</p> <p>These bits indicate the Transmit DMA FSM state. This field does not generate an interrupt.</p> <ul style="list-style-type: none"> <li>• 3'b000: Stopped; Reset or Stop Transmit Command issued.</li> <li>• 3'b001: Running; Fetching Transmit Transfer Descriptor.</li> <li>• 3'b010: Running; Waiting for status.</li> <li>• 3'b011: Running; Reading Data from host memory buffer and queuing it to transmit buffer (Tx FIFO).</li> <li>• 3'b100: TIME_STAMP write state.</li> <li>• 3'b101: Reserved for future use.</li> <li>• 3'b110: Suspended; Transmit Descriptor Unavailable or Transmit Buffer Underflow.</li> <li>• 3'b111: Running; Closing Transmit Descriptor.</li> </ul>	000	RO
19:17	<p>RS: Receive Process State</p> <p>These bits indicate the Receive DMA FSM state. This field does not generate an interrupt.</p> <ul style="list-style-type: none"> <li>• 3'b000: Stopped: Reset or Stop Receive Command issued.</li> <li>• 3'b001: Running: Fetching Receive Transfer Descriptor.</li> <li>• 3'b010: Reserved for future use.</li> <li>• 3'b011: Running: Waiting for receive packet.</li> <li>• 3'b100: Suspended: Receive Descriptor Unavailable.</li> <li>• 3'b101: Running: Closing Receive Descriptor.</li> <li>• 3'b110: TIME_STAMP write state.</li> <li>• 3'b111: Running: Transferring the receive packet data from receive buffer to host memory.</li> </ul>	000	RO
16	<p>NIS: Normal Interrupt Summary</p> <p>Normal Interrupt Summary bit value is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Register 7:</p> <ul style="list-style-type: none"> <li>• Register 5[0]: Transmit Interrupt</li> <li>• Register 5[2]: Transmit Buffer Unavailable</li> <li>• Register 5[6]: Receive Interrupt</li> <li>• Register 5[14]: Early Receive Interrupt</li> </ul> <p>Only unmasked bits affect the Normal Interrupt Summary bit.</p> <p>This is a sticky bit and must be cleared (by writing a 1 to this bit) each time a corresponding bit that causes NIS to be set is cleared.</p>	0	R_SS_WC

**Table 5-8 Register 5 (Status Register) (Continued)**

Field	Description	Reset	Access
15	<p>AIS: Abnormal Interrupt Summary</p> <p>Abnormal Interrupt Summary bit value is the logical OR of the following when the corresponding interrupt bits are enabled in DMA Register 7:</p> <ul style="list-style-type: none"> <li>• Register 5[1]:Transmit Process Stopped</li> <li>• Register 5[3]:Transmit Jabber Timeout</li> <li>• Register 5[4]: Receive FIFO Overflow</li> <li>• Register 5[5]: Transmit Underflow</li> <li>• Register 5[7]: Receive Buffer Unavailable</li> <li>• Register 5[8]: Receive Process Stopped</li> <li>• Register 5[9]: Receive Watchdog Timeout</li> <li>• Register 5[10]: Early Transmit Interrupt</li> <li>• Register 5[13]: Fatal Bus Error</li> </ul> <p>Only unmasked bits affect the Abnormal Interrupt Summary bit.</p> <p>This is a sticky bit and must be cleared each time a corresponding bit that causes AIS to be set is cleared.</p>	0	R_SS_WC
14	<p>ERI: Early Receive Interrupt</p> <p>This bit indicates that the DMA had filled the first data buffer of the packet. Receive Interrupt Register 5[6] automatically clears this bit.</p>	0	R_SS_WC
13	<p>FBI: Fatal Bus Error Interrupt</p> <p>This bit indicates that a bus error occurred, as detailed in [25:23]. When this bit is set, the corresponding DMA engine disables all its bus accesses.</p>	0	R_SS_WC
12:11	Reserved	00	RO
10	<p>ETI: Early Transmit Interrupt</p> <p>This bit indicates that the frame to be transmitted was fully transferred to the MTL Transmit FIFO.</p>	0	R_SS_WC
9	<p>RWT: Receive Watchdog Timeout</p> <p>This bit is asserted when a frame with a length greater than 2,048 bytes is received (10,240 when Jumbo Frame mode is enabled).</p>	0	R_SS_WC
8	<p>RPS: Receive Process Stopped</p> <p>This bit is asserted when the Receive Process enters the Stopped state.</p>	0	R_SS_WC
7	<p>RU: Receive Buffer Unavailable</p> <p>This bit indicates that the Next Descriptor in the Receive List is owned by the host and cannot be acquired by the DMA. Receive Process is suspended. To resume processing Receive descriptors, the host should change the ownership of the descriptor and issue a Receive Poll Demand command. If no Receive Poll Demand is issued, Receive Process resumes when the next recognized incoming frame is received. Register 5[7] is set only when the previous Receive Descriptor was owned by the DMA.</p>	0	R_SS_WC

**Table 5-8 Register 5 (Status Register) (Continued)**

Field	Description	Reset	Access
6	RI: Receive Interrupt This bit indicates the completion of frame reception. Specific frame status information has been posted in the descriptor. Reception remains in the Running state.	0	R_SS_WC
5	UNF: Transmit Underflow This bit indicates that the Transmit Buffer had an Underflow during frame transmission. Transmission is suspended and an Underflow Error TDES0[1] is set.	0	R_SS_WC
4	OVF: Receive Overflow This bit indicates that the Receive Buffer had an Overflow during frame reception. If the partial frame is transferred to application, the overflow status is set in RDES0[11].	0	R_SS_WC
3	TJT: Transmit Jabber Timeout This bit indicates that the Transmit Jabber Timer expired, meaning that the transmitter had been excessively active. The transmission process is aborted and placed in the Stopped state. This causes the Transmit Jabber Timeout TDES0[14] flag to assert.	0	R_SS_WC
2	TU: Transmit Buffer Unavailable This bit indicates that the Next Descriptor in the Transmit List is owned by the host and cannot be acquired by the DMA. Transmission is suspended. Bits[22:20] explain the Transmit Process state transitions. To resume processing transmit descriptors, the host should change the ownership of the bit of the descriptor and then issue a Transmit Poll Demand command.	0	R_SS_WC
1	TPS: Transmit Process Stopped This bit is set when the transmission is stopped.	0	R_SS_WC
0	TI: Transmit Interrupt This bit indicates that frame transmission is finished and TDES1[31] is set in the First Descriptor.	0	R_SS_WC

### 5.2.1.7 Register 6 (Operation Mode Register)

The Operation Mode register establishes the Transmit and Receive operating modes and commands. Register 6 should be the last CSR to be written as part of DMA initialization. This register is also present in the GMAC-MTL configuration with bits 24, 13, 2, and 1 unused and reserved.

**Table 5-9 Register 6 (Operation Mode Register)**

Field	Description	Reset	Access
31:27	Reserved	000H	RO
26	DT: Disable Dropping of TCP/IP Checksum Error Frames  When this bit is set, the core does not drop frames that only have errors detected by the Receive Checksum Offload engine. Such frames do not have any errors (including FCS error) in the Ethernet frame received by the MAC but have errors in the encapsulated payload only. When this bit is reset, all error frames are dropped if the FEF bit is reset.  If the Full Checksum Offload engine (Type 2) is disabled, this bit is reserved (RO with value 1'b0).	0	R_W
25	RSF: Receive Store and Forward  When this bit is set, the MTL only reads a frame from the Rx FIFO after the complete frame has been written to it, ignoring RTC bits. When this bit is reset, the Rx FIFO operates in Cut-Through mode, subject to the threshold specified by the RTC bits.	0	R_W
24	DFF: Disable Flushing of Received Frames  When this bit is set, the RxDMA does not flush any frames due to the unavailability of receive descriptors/buffers as it does normally when this bit is reset. (See “ <a href="#">Receive Process Suspended</a> ” on page 75.)  This bit is reserved (and RO) in GMAC-MTL configuration.	0	R_W
23	RFA[2]: MSB of Threshold for Activating Flow Control  If the GMAC-UNIV is configured for an Rx FIFO depth of 8 KB or more, this bit (when set) provides additional threshold levels for activating the Flow Control in both Half-Duplex and Full-Duplex modes. This bit (as Most Significant Bit) along with the RFA (bits [10:9]) give the following thresholds for activating flow control. <ul style="list-style-type: none"> <li>• 100: Full minus 5 KB</li> <li>• 101: Full minus 6 KB</li> <li>• 110: Full minus 7 KB</li> <li>• 111: Reserved</li> </ul> This bit is reserved (and RO) if the Rx FIFO is 4 KB or less deep.	0	R_W
22	RFD[2]: MSB of Threshold for Deactivating Flow Control  If the GMAC-UNIV is configured for an Rx FIFO depth of 8 KB or more, this bit (when set) provides additional threshold levels for deactivating the Flow Control in both Half-Duplex and Full-Duplex modes. This bit (as Most Significant Bit) along with the RFD (bits [12:11]) give the following thresholds for deactivating flow control. <ul style="list-style-type: none"> <li>• 100: Full minus 5 KB</li> <li>• 101: Full minus 6 KB</li> <li>• 110: Full minus 7 KB</li> <li>• 111: Reserved</li> </ul> This bit is reserved (and RO) if the Rx FIFO is 4 KB or less deep.	0	R_W

**Table 5-9 Register 6 (Operation Mode Register) (Continued)**

Field	Description	Reset	Access
21	TSF: Transmit Store and Forward When this bit is set, transmission starts when a full frame resides in the MTL Transmit FIFO. When this bit is set, the TTC values specified in Register 6[16:14] are ignored. This bit should be changed only when transmission is stopped.	0	R_W
20	FTF: Flush Transmit FIFO When this bit is set, the transmit FIFO controller logic is reset to its default values and thus all data in the Tx FIFO is lost/flushed. This bit is cleared internally when the flushing operation is completed fully. The Operation Mode register should not be written to until this bit is cleared. The data which is already accepted by the MAC transmitter will not be flushed. It will be scheduled for transmission and will result in underflow and runt frame transmission. <b>Note:</b> The flush operation completes only after emptying the TxFIFO of its contents and all the pending Transmit Status of the transmitted frames are accepted by the host. In order to complete this flush operation, the PHY transmit clock (clk_tx_i) is required to be active.	0	R_WS_SC
19:17	Reserved	000H	RO
16:14	TTC: Transmit Threshold Control These three bits control the threshold level of the MTL Transmit FIFO. Transmission starts when the frame size within the MTL Transmit FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are also transmitted. These bits are used only when the TSF bit (Bit 21) is reset. <ul style="list-style-type: none"><li>• 000: 64</li><li>• 001: 128</li><li>• 010: 192</li><li>• 011: 256</li><li>• 100: 40</li><li>• 101: 32</li><li>• 110: 24</li><li>• 111: 16</li></ul>	000	R_W
13	ST: Start/Stop Transmission Command When this bit is set, transmission is placed in the Running state, and the DMA checks the Transmit List at the current position for a frame to be transmitted. Descriptor acquisition is attempted either from the current position in the list, which is the Transmit List Base Address set by Register 4, or from the position retained when transmission was stopped previously. If the current descriptor is not owned by the DMA, transmission enters the Suspended state and Transmit Buffer Unavailable (Register 5[2]) is set. The Start Transmission command is effective only when transmission is stopped. If the command is issued before setting DMA Register 4, then the DMA behavior is unpredictable. When this bit is reset, the transmission process is placed in the Stopped state after completing the transmission of the current frame. The Next Descriptor position in the Transmit List is saved, and becomes the current position when transmission is restarted. The stop transmission command is effective only the transmission of the current frame is complete or when the transmission is in the Suspended state.	0	R_W

**Table 5-9 Register 6 (Operation Mode Register) (Continued)**

Field	Description	Reset	Access
12:11	RFD: Threshold for deactivating flow control (in both HD and FD)  These bits control the threshold (Fill-level of Rx FIFO) at which the flow-control is deasserted after activation. <ul style="list-style-type: none"><li>• 00: Full minus 1 KB</li><li>• 01: Full minus 2 KB</li><li>• 10: Full minus 3 KB</li><li>• 11: Full minus 4 KB</li></ul> Note that the deassertion is effective only after flow control is asserted. If the Rx FIFO is 8 KB or more, an additional bit (RFD[2]) is used for more threshold levels as described in bit [22]. These bits are reserved and read-only when RxFIFO depth is less than 4 KB.	00	R_W
10:9	RFA: Threshold for activating flow control (in both HD and FD)  These bits control the threshold (Fill level of Rx FIFO) at which flow control is activated. <ul style="list-style-type: none"><li>• 00: Full minus 1 KB</li><li>• 01: Full minus 2 KB</li><li>• 10: Full minus 3 KB</li><li>• 11: Full minus 4 KB</li></ul> Note that the above only applies to Rx FIFOs of 4 KB or more when the EFC bit is set high. If the Rx FIFO is 8 KB or more, an additional bit (RFA[2]) is used for more threshold levels as described in bit [23]. These bits are reserved and read-only when RxFIFO depth is less than 4 KB.	00	R_W
8	EFC: Enable HW flow control  When this bit is set, the flow control signal operation based on fill-level of Rx FIFO is enabled. When reset, the flow control operation is disabled. This bit is not used (reserved and always reset) when the Rx FIFO is less than 4 KB.	0	R_W
7	FEF: Forward Error Frames  When this bit is reset, the Rx FIFO drops frames with error status (CRC error, collision error, GMII_ER, giant frame, watchdog timeout, overflow). However, if the frame's start byte (write) pointer is already transferred to the read controller side (in Threshold mode), then the frames are not dropped. Note that in GMAC-MTL configuration in which the Frame Length FIFO is also enabled during coreKit configuration, the Rx FIFO drops the error frames if that frame's start byte is not transferred (output) on the ARI bus. When FEF is set, all frames except runt error frames are forwarded to the DMA. But when RxFIFO overflows when a partial frame is written, then such frames are dropped even when FEF is set.	0	R_W
6	FUF: Forward Undersized Good Frames  When set, the Rx FIFO will forward Undersized frames (frames with no Error and length less than 64 bytes) including pad-bytes and CRC). When reset, the Rx FIFO will drop all frames of less than 64 bytes, unless it is already transferred due to lower value of Receive Threshold (e.g., RTC = 01).	0	R_W
5	Reserved	0	RO

**Table 5-9 Register 6 (Operation Mode Register) (Continued)**

Field	Description	Reset	Access
4:3	RTC: Receive Threshold Control These two bits control the threshold level of the MTL Receive FIFO. Transfer (request) to DMA starts when the frame size within the MTL Receive FIFO is larger than the threshold. In addition, full frames with a length less than the threshold are transferred automatically. Note that value of 11 is not applicable if the configured Receive FIFO size is 128 bytes. These bits are valid only when the RSF bit is zero, and are ignored when the RSF bit is set to 1. <ul style="list-style-type: none"><li>• 00: 64</li><li>• 01: 32</li><li>• 10: 96</li><li>• 11: 128</li></ul>	00	R_W
2	OSF: Operate on Second Frame When this bit is set, this bit instructs the DMA to process a second frame of Transmit data even before status for first frame is obtained.	0	R_W
1	SR: Start/Stop Receive When this bit is set, the Receive process is placed in the Running state. The DMA attempts to acquire the descriptor from the Receive list and processes incoming frames. Descriptor acquisition is attempted from the current position in the list, which is the address set by DMA Register 3 or the position retained when the Receive process was previously stopped. If no descriptor is owned by the DMA, reception is suspended and Receive Buffer Unavailable (Register 5[7]) is set. The Start Receive command is effective only when reception has stopped. If the command was issued before setting DMA Register 3, DMA behavior is unpredictable. When this bit is cleared, RxDMA operation is stopped after the transfer of the current frame. The next descriptor position in the Receive list is saved and becomes the current position after the Receive process is restarted. The Stop Receive command is effective only when the Receive process is in either the Running (waiting for receive packet) or in the Suspended state.	0	R_W
0	Reserved	0	RO

### 5.2.1.8 Register 7 (Interrupt Enable Register)

The Interrupt Enable register enables the interrupts reported by Register 5. Setting a bit to 1'b1 enables a corresponding interrupt. After a hardware or software reset, all interrupts are disabled.

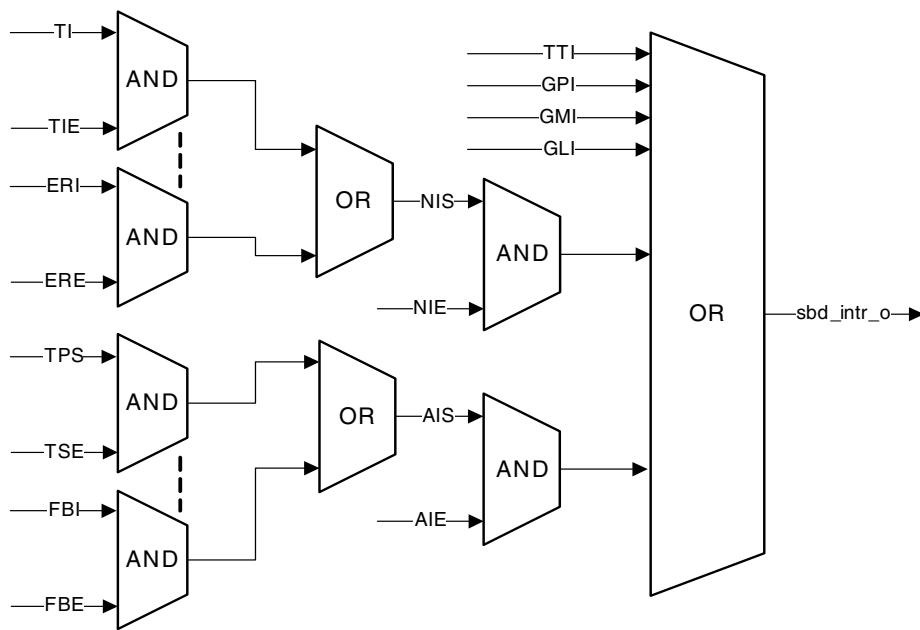
**Table 5-10 Register 7 (Interrupt Enable Register)**

Field	Description	Reset	Access
31:17	Reserved	00H	RO
16	NIE: Normal Interrupt Summary Enable When this bit is set, a normal interrupt is enabled. When this bit is reset, a normal interrupt is disabled. This bit enables the following bits: <ul style="list-style-type: none"><li>• Register 5[0]: Transmit Interrupt</li><li>• Register 5[2]: Transmit Buffer Unavailable</li><li>• Register 5[6]: Receive Interrupt</li><li>• Register 5[14]: Early Receive Interrupt</li></ul>	0	R_W
15	AIE: Abnormal Interrupt Summary Enable When this bit is set, an Abnormal Interrupt is enabled. When this bit is reset, an Abnormal Interrupt is disabled. This bit enables the following bits <ul style="list-style-type: none"><li>• Register 5[1]: Transmit Process Stopped</li><li>• Register 5[3]: Transmit Jabber Timeout</li><li>• Register 5[4]: Receive Overflow</li><li>• Register 5[5]: Transmit Underflow</li><li>• Register 5[7]: Receive Buffer Unavailable</li><li>• Register 5[8]: Receive Process Stopped</li><li>• Register 5[9]: Receive Watchdog Timeout</li><li>• Register 5[10]: Early Transmit Interrupt</li><li>• Register 5[13]: Fatal Bus Error</li></ul>	0	R_W
14	ERE: Early Receive Interrupt Enable When this bit is set with Normal Interrupt Summary Enable (Register 7[16]), Early Receive Interrupt is enabled. When this bit is reset, Early Receive Interrupt is disabled.	0	R_W
13	FBE: Fatal Bus Error Enable When this bit is set with Abnormal Interrupt Summary Enable (Register 7[15]), the Fatal Bus Error Interrupt is enabled. When this bit is reset, Fatal Bus Error Enable Interrupt is disabled.	0	R_W
12:11	Reserved	00	RO
10	ETE: Early Transmit Interrupt Enable When this bit is set with an Abnormal Interrupt Summary Enable (Register 7[15]), Early Transmit Interrupt is enabled. When this bit is reset, Early Transmit Interrupt is disabled.	0	R_W
9	RWE: Receive Watchdog Timeout Enable When this bit is set with Abnormal Interrupt Summary Enable (Register 7[15]), the Receive Watchdog Timeout Interrupt is enabled. When this bit is reset, Receive Watchdog Timeout Interrupt is disabled.	0	R_W

**Table 5-10 Register 7 (Interrupt Enable Register) (Continued)**

Field	Description	Reset	Access
8	RSE: Receive Stopped Enable When this bit is set with Abnormal Interrupt Summary Enable (Register 7[15]), Receive Stopped Interrupt is enabled. When this bit is reset, Receive Stopped Interrupt is disabled.	0	R_W
7	RUE: Receive Buffer Unavailable Enable When this bit is set with Abnormal Interrupt Summary Enable (Register 7[15]), Receive Buffer Unavailable Interrupt is enabled. When this bit is reset, the Receive Buffer Unavailable Interrupt is disabled.	0	R_W
6	RIE: Receive Interrupt Enable When this bit is set with Normal Interrupt Summary Enable (Register 7[16]), Receive Interrupt is enabled. When this bit is reset, Receive Interrupt is disabled.	0	R_W
5	UNE: Underflow Interrupt Enable When this bit is set with Abnormal Interrupt Summary Enable (Register 7[15]), Transmit Underflow Interrupt is enabled. When this bit is reset, Underflow Interrupt is disabled.	0	R_W
4	OVE: Overflow Interrupt Enable When this bit is set with Abnormal Interrupt Summary Enable (Register 7[15]), Receive Overflow Interrupt is enabled. When this bit is reset, Overflow Interrupt is disabled	0	R_W
3	TJE: Transmit Jabber Timeout Enable When this bit is set with Abnormal Interrupt Summary Enable (Register 7[15]), Transmit Jabber Timeout Interrupt is enabled. When this bit is reset, Transmit Jabber Timeout Interrupt is disabled.	0	R_W
2	TUE: Transmit Buffer Unavailable Enable When this bit is set with Normal Interrupt Summary Enable (Register 7[16]), Transmit Buffer Unavailable Interrupt is enabled. When this bit is reset, Transmit Buffer Unavailable Interrupt is disabled.	0	R_W
1	TSE: Transmit Stopped Enable When this bit is set with Abnormal Interrupt Summary Enable (Register 7[15]), Transmission Stopped Interrupt is enabled. When this bit is reset, Transmission Stopped Interrupt is disabled.	0	R_W
0	TIE: Transmit Interrupt Enable When this bit is set with Normal Interrupt Summary Enable (Register 7[16]), Transmit Interrupt is enabled. When this bit is reset, Transmit Interrupt is disabled.	0	R_W

The sbd\_intr\_o interrupt is generated as shown in [Figure 5-1](#). It is asserted only when the TTI, GPI, GMI, or GLI bits of the DMA Status register is asserted, or when the NIS/AIS Status bit is asserted and the corresponding Interrupt Enable bits (NIE/AIE) are enabled.

**Figure 5-1 Sbd\_intr\_o Generation**

**Note:** Signals NIS and

### 5.2.1.9 Register 8 (Missed Frame and Buffer Overflow Counter Register)

The DMA maintains two counters to track the number of missed frames during reception. This register reports the current value of the counter. The counter is used for diagnostic purposes. Bits[15:0] indicate missed frames due to the host buffer being unavailable. Bits[27:17] indicate missed frames due to buffer overflow conditions (MTL and GMAC) and runt frames (good frames of less than 64 bytes) dropped by the MTL.

**Table 5-11 Register 8 (Missed Frame and Buffer Overflow Counter Register)**

Field	Description	Reset	Access
31:29	Reserved	000	RO
28	Overflow bit for FIFO Overflow Counter	0	R_SS_RC
27:17	Indicates the number of frames missed by the application. This counter is incremented each time the MTL asserts the sideband signal mtl_rxoverflow_o. The counter is cleared when this register is read with mci_be_i[2] at 1'b1.	000H	R_SS_RC
16	Overflow bit for Missed Frame Counter	0	R_SS_RC
15:0	Indicates the number of frames missed by the controller due to the Host Receive Buffer being unavailable. This counter is incremented each time the DMA discards an incoming frame. The counter is cleared when this register is read with mci_be_i[0] at 1'b1.	0000H	R_SS_RC

### 5.2.1.10 Register 9 (Receive Interrupt Watchdog Timer Register)

This register, when written with non-zero value, will enable the watchdog timer for RI (DMA Register 5[6]).

**Table 5-12 Register 9 (RI Watchdog Timer Register)**

Field	Description	Reset	Access
31:8	Reserved	000000H	RO
7:0	RIWT: RI Watchdog Timer count  Indicates the number of system clock cycles multiplied by 256 for which the watchdog timer is set. The watchdog timer gets triggered with the programmed value after the RxDMA completes the transfer of a frame for which the RI status bit is not set due to the setting in the corresponding descriptor RDES1[31]. When the watch-dog timer runs out, the RI bit is set and the timer is stopped. The watchdog timer is reset when RI bit is set high due to automatic setting of RI as per RDES1[31] of any received frame.	00H	R_W

### 5.2.1.11 Register 10 (AXI Bus Mode Register)

The AXI Bus Mode Register Controls AXI master behavior. It is mainly used to control the burst splitting and the number of outstanding requests. This register is present and valid only in GMAC-AXI configuration.

**Table 5-13 Register 10 (AXI Bus Mode Register)**

Field	Description	Reset	Access
31	EN_LPI: Enable LPI (Low Power Interface)  When set to 1, enable the LPI (Low Power Interface) supported by the GMAC-AXI and accepts the LPI request from the AXI System Clock controller.  When set to 0, disables the Low Power Mode and always denies the LPI request from the AXI System Clock controller.	0	RW
30	UNLCK_ON_MGK_RWK: Unlock on Magic Packet or Remote Wake Up  When set to 1, enables GMAC-AXI to request coming out of Low Power mode only when Magic Packet or Remote Wake Up Packet is received.  When set to 0, enables GMAC-AXI requests to come out of Low Power mode when any frame is received.	0	RW
29:23	Reserved	7'h00	RO
22/ 21:20	WR_OSR_LMT: AXI Maximum Write Out Standing Request Limit  This value limits the maximum outstanding request on the AXI write interface. Maximum outstanding requests = WR_OSR_LMT+1  <b>Note:</b> The 22 <sup>nd</sup> bit is reserved if AXI_GM_MAX_WR_REQUESTS = 4	'h1	RW
19	Reserved	0	RO
18/ 17:16	RD_OSR_LMT: AXI Maximum Read Out Standing Request Limit  This value limits the maximum outstanding request on the AXI read interface. Maximum outstanding requests = RD_OSR_LMT+1  <b>Note:</b> The 18 <sup>nd</sup> bit is reserved if AXI_GM_MAX_RD_REQUESTS = 4,	'h1	RW

**Table 5-13 Register 10 (AXI Bus Mode Register) (Continued)**

Field	Description	Reset	Access
15:13	Reserved	0	RO
12	AXI_AAL: Address-Aligned Beats  This bit is read-only bit and reflects the AAL bit Register0 (Bus Mode Register[25]).  When this bit set to 1, GMAC-AXI performs address-aligned burst transfers on both read and write channels. Refer to section 3.4.2 for details.	0	RO
11:8	Reserved		
7	BLEN256: AXI Burst Length 256  This bit is present only when the configuration parameter ‘AXI_BURST_LENGTH’ is 256. Otherwise, this bit is Reserved.  When this bit is set to 1, the GMAC-AXI is allowed to select a burst length of 256.	0	See Desc.
6	BLEN128: AXI Burst Length 128  This bit is present only when the configuration parameter ‘AXI_BURST_LENGTH’ is 128 or more. Otherwise, this bit is Reserved.  When this bit is set to 1, the GMAC-AXI is allowed to select a burst length of 128.	0	See Desc.
5	BLEN64: AXI Burst Length 64  This bit is present only when the configuration parameter ‘AXI_BURST_LENGTH’ is 64 or more. Otherwise, this bit is Reserved.  When this bit is set to 1, the GMAC-AXI is allowed to select a burst length of 64.	0	See Desc.
4	BLEN32: AXI Burst Length 32  This bit is present only when the configuration parameter ‘AXI_BURST_LENGTH’ is 32 or more. Otherwise, this bit is Reserved.  When this bit is set to 1, the GMAC-AXI is allowed to select a burst length of 32.	0	See Desc.
3	BLEN16: AXI Burst Length 16  When this bit is set to 1, or when UNDEF is set to 1, the GMAC-AXI is allowed to select a burst length of 16.	0	R/W
2	BLEN8: AXI Burst Length 8  When this bit is set to 1, or when UNDEF is set to 1, the GMAC-AXI is allowed to select a burst length of 8.	0	R/W
1	BLEN4: AXI Burst Length 4  When this bit is set to 1, or when UNDEF is set to 1, the GMAC-AXI is allowed to select a burst length of 4.	0	R/W
0	UNDEF: AXI Undefined Burst Length  This bit is read-only bit and indicates the complement (invert) value of FB bit in Register0 (Bus Mode Register[16]). <ul style="list-style-type: none"><li>• When this bit is set to 1, the GMAC-AXI is allowed to perform any burst length equal to or below the maximum allowed burst length as programmed in bits[7:1]</li><li>• When this bit is set to 0, the GMAC-AXI is allowed to perform only fixed burst lengths as indicated by BLEN256/128/64/32/16/8/4, or a burst length of 1.</li></ul>	1	RO

### 5.2.1.12 Register 11 (AXI Status Register)

The active status of the AXI interface's read and write channel are given in this register. This register is present and valid only in GMAC-AXI configuration and useful for debug purposes.

**Table 5-14 Register 11 (AXI Status Register)**

Field	Description	Reset	Access
31:2	Reserved	0000_0000H	RO
1	When high, it indicates that AXI Master's read channel is active and transferring data.	0	RO
0	When high, it indicates that AXI Master's write channel is active and transferring data.	0	RO

### 5.2.1.13 Register 18 (Current Host Transmit Descriptor Register)

The Current Host Transmit Descriptor register points to the start address of the current Transmit Descriptor read by the DMA.

**Table 5-15 Register 18 (Current Host Transmit Descriptor Register)**

Field	Description	Reset	Access
31:0	Host Transmit Descriptor Address Pointer Cleared on Reset. Pointer updated by DMA during operation.	0000_0000H	RO

### 5.2.1.14 Register 19 (Current Host Receive Descriptor Register)

The Current Host Receive Descriptor register points to the start address of the current Receive Descriptor read by the DMA.

**Table 5-16 Register 19 (Current Host Receive Descriptor Register)**

Field	Description	Reset	Access
31:0	Host Receive Descriptor Address Pointer Cleared on Reset. Pointer updated by DMA during operation.	0000_0000H	RO

### 5.2.1.15 Register 20 (Current Host Transmit Buffer Address Register)

The Current Host Transmit Buffer Address register points to the current Transmit Buffer Address being read by the DMA.

**Table 5-17 Register 20 (Current Host Transmit Buffer Address Register)**

Field	Description	Reset	Access
31:0	Host Transmit Buffer Address Pointer Cleared on Reset. Pointer updated by DMA during operation.	0000_0000H	RO

### 5.2.1.16 Register 21 (Current Host Receive Buffer Address Register)

The Current Host Receive Buffer Address register points to the current Receive Buffer address being read by the DMA.

**Table 5-18 Register 21 (Current Host Receive Buffer Address Register)**

Field	Description	Reset	Access
31:0	Host Receive Buffer Address Pointer Cleared on Reset. Pointer updated by DMA during operation.	0000_0000H	RO

### 5.2.1.17 Register 22 (HW Feature Register)

This register indicates the presence of the optional features/functions of the core. It is useful for SW driver to dynamically enable or disable the programs related to the optional blocks.

**Table 5-19 Register 22 (HW Feature Register)**

Field	Description	Reset	Access
<b>Note:</b> All bits are read-only and are set or reset as per the selection of features during RTL configuration.			
31:25	Reserved		
24	Alternate (Enhanced Descriptor)		
23:20	Reserved		
19	RxFIFO > 2048 Bytes		
18	IP Checksum Offload (Type 2) in Rx		
17	IP Checksum Offload (Type 1) in Rx		
16	Checksum Offload in Tx		
15:14	Reserved		
13	IEEE 1588-2008 Advanced Time-Stamp		
12	IEEE 1588-2002 Time-Stamp		
11	RMON module		
10	PMT Magic Packet		
9	PMT Remote Wakeup		
8	SMA (MDIO) Interface		
7	Reserved		
6	PCS registers (TBI/SGMII/RTBI PHY interface)		
5	Multiple MAC Address Registers		
4	HASH Filter		

**Table 5-19 Register 22 (HW Feature Register) (Continued)**

Field	Description	Reset	Access
3	Reserved		
2	Half-Duplex support		
1	1000 Mbps support		
0	10/100 Mbps support		

## 5.2.2 GMAC Register Description

### 5.2.2.1 Register 0 (MAC Configuration Register)

The MAC Configuration register establishes receive and transmit operating modes.

**Table 5-20 Register 0 (MAC Configuration Register)**

Field	Description	Reset	Access
31:27	Reserved	00H	RO
26	SFTERR: SMII Force Transmit Error When set, indicates to the PHY to force transmit error in the SMII frame being transmitted. This bit is reserved if SMII PHY port is not selected during core configuration.	0	R_W
25	CST: CRC stripping for Type frames When set, the last 4 bytes (FCS) of all frames of Ether type (type field greater than 0x0600) will be stripped and dropped before forwarding the frame to the application. This function is not valid when the MAC receiver has IP Checksum Engine (Type 1) enabled.	0	R_W
24	TC: Transmit Configuration in RGMII/SGMII/SMII When set, this bit enables the transmission of duplex mode, link speed, and link up/down information to the PHY in the RGMII/SMII/SGMII ports. The details of this feature are explained in “ <a href="#">Reduced Gigabit Media Independent Interface</a> ” on page 164, “ <a href="#">Serial Media Independent Interface (SMII)</a> ” on page 160, and “ <a href="#">Serial Gigabit Media Independent Interface</a> ” on page 171. When this bit is reset, no such information is driven to the PHY. This bit is reserved (and RO) if neither RGMII nor SMII nor SGMII PHY port is selected during core configuration.	0	R_W
23	WD: Watchdog Disable When this bit is set, the GMAC disables the watchdog timer on the receiver, and can receive frames of up to 16,384 bytes. When this bit is reset, the GMAC allows no more than 2,048 bytes (10,240 if JE is set high) of the frame being received and cuts off any bytes received after that.	0	R_W
22	JD: Jabber Disable When this bit is set, the GMAC disables the jabber timer on the transmitter, and can transfer frames of up to 16,384 bytes. When this bit is reset, the GMAC cuts off the transmitter if the application sends out more than 2,048 bytes of data (10,240 if JE is set high) during transmission.	0	R_W
21	BE: Frame Burst Enable When this bit is set, the GMAC allows frame bursting during transmission in GMII Half-Duplex mode. This bit is reserved (and RO) in 10/100 Mbps only or Full-Duplex-only configurations.	0	R_W
20	JE: Jumbo Frame Enable When this bit is set, GMAC allows Jumbo frames of 9,018 bytes (9,022 bytes for VLAN tagged frames) without reporting a giant frame error in the receive frame status.	0	R_W

**Table 5-20 Register 0 (MAC Configuration Register) (Continued)**

Field	Description	Reset	Access
19:17	<p>IFG: Inter-Frame Gap These bits control the minimum IFG between frames during transmission.</p> <ul style="list-style-type: none"> <li>• 000: 96 bit times</li> <li>• 001: 88 bit times</li> <li>• 010: 80 bit times</li> <li>....</li> <li>• 111: 40 bit times</li> </ul> <p>Note that in Half-Duplex mode, the minimum IFG can be configured for 64 bit times (IFG = 100) only. Lower values are not considered. In 1000-Mbps mode, the minimum IFG supported is 64 bit times (and above) in the GMAC-CORE configuration and 80 bit times (and above) in other system configurations.</p>	000	R_W
16	<p>DCRS: Disable Carrier Sense During Transmission When set high, this bit makes the MAC transmitter ignore the (G)MII CRS signal during frame transmission in Half-Duplex mode. This request results in no errors generated due to Loss of Carrier or No Carrier during such transmission. When this bit is low, the MAC transmitter generates such errors due to Carrier Sense and will even abort the transmissions.</p> <p>This bit is reserved (and RO) when the core is selected for Full-Duplex-only operation during core configuration.</p>	0	R_W
15	<p>PS: Port Select Selects between GMII and MII:</p> <ul style="list-style-type: none"> <li>• 0: GMII (1000 Mbps)</li> <li>• 1: MII (10/100 Mbps)</li> </ul> <p>This bit is read-only with the appropriate value in the 10/100 Mbps-only (always 1) or 1000 Mbps-only (always 0) configurations, and R_W in the default 10/100/1000 Mbps configuration.</p>	0	R_W
14	<p>FES: Speed Indicates the speed in Fast Ethernet (MII) mode:</p> <ul style="list-style-type: none"> <li>• 0: 10 Mbps</li> <li>• 1: 100 Mbps</li> </ul> <p>This bit is reserved (RO) by default and is enabled only when RMII/SMII/RGMII/SGMII is enabled during configuration. This bit generates link speed encoding when TC (Bit 24) is set in RGMII/SMII/SGMII mode. This signal can optionally be driven as an output signal (mac_speed_o[0]) if the core is configured with an RMII, SMII, SGMII, or RGMII PHY interface selected. This bit is reserved when RMII is enabled during configuration without enabling the "Output Port for speed selection" option.</p>	0	R_W

**Table 5-20 Register 0 (MAC Configuration Register) (Continued)**

Field	Description	Reset	Access
13	DO: Disable Receive Own When this bit is set, the GMAC disables the reception of frames when the gmii_txen_o is asserted in Half-Duplex mode. When this bit is reset, the GMAC receives all packets that are given by the PHY while transmitting. This bit is not applicable if the GMAC is operating in Full-Duplex mode. This bit is reserved (RO with default value) if the GMAC is configured for Full-Duplex-only operation during configuration.	0	R_W
12	LM: Loopback Mode When this bit is set, the GMAC operates in loopback mode at GMII/MII. The (G)MII Receive clock input (clk_rx_i) is required for the loopback to work properly, as the Transmit clock is not looped-back internally.	0	R_W
11	DM: Duplex Mode When this bit is set, the GMAC operates in a Full-Duplex mode where it can transmit and receive simultaneously. This bit is RO with default value of 1'b1 in Full-Duplex-only configuration.	0	R_W
10	IPC: Checksum Offload When this bit is set, the GMAC calculates the 16-bit one's complement of the one's complement sum of all received Ethernet frame payloads. It also checks whether the IPv4 Header checksum (assumed to be bytes 25–26 or 29–30 (VLAN-tagged) of the received Ethernet frame) is correct for the received frame and gives the status in the receive status word. The GMAC core also appends the 16-bit checksum calculated for the IP header datagram payload (bytes after the IPv4 header) and appends it to the Ethernet frame transferred to the application (when Type 2 COE is deselected). When this bit is reset, this function is disabled. When Type 2 COE is selected, this bit, when set, enables IPv4 checksum checking for received frame payloads' TCP/UDP/ICMP headers. When this bit is reset, the COE function in the receiver is disabled and the corresponding PCE and IP HCE status bits (see <a href="#">Table 3-15</a> on page <a href="#">123</a> ) are always cleared. If the IP Checksum Offload feature is not enabled during coreKit configuration, this bit is reserved (RO with default value).	0	R_W
9	DR: Disable Retry When this bit is set, the GMAC will attempt only 1 transmission. When a collision occurs on the GMII/MII, the GMAC will ignore the current frame transmission and report a Frame Abort with excessive collision error in the transmit frame status. When this bit is reset, the GMAC will attempt retries based on the settings of BL. This bit is applicable only to Half-Duplex mode and is reserved (RO with default value) in Full-Duplex-only configuration.	0	R_W

**Table 5-20 Register 0 (MAC Configuration Register) (Continued)**

Field	Description	Reset	Access
8	LUD: Link Up/Down  Indicates whether the link is up or down during the transmission of configuration in RGMII/SGMII/SMII interface: <ul style="list-style-type: none"><li>• 0: Link Down</li><li>• 1: Link Up</li></ul> This bit is reserved (RO with default value) and is enabled when RGMII/SGMII/SMII is enabled during configuration.	0	R_W
7	ACS: Automatic Pad/CRC Stripping  When this bit is set, the GMAC strips the Pad/FCS field on incoming frames only if the length's field value is less than or equal to 1,500 bytes. All received frames with length field greater than or equal to 1,501 bytes are passed to the application without stripping the Pad/FCS field.  When this bit is reset, the GMAC will pass all incoming frames to the Host unmodified.	0	R_W
6:5	BL: Back-Off Limit  The Back-Off limit determines the random integer number ( $r$ ) of slot time delays (4,096 bit times for 1000 Mbps and 512 bit times for 10/100 Mbps) the GMAC waits before rescheduling a transmission attempt during retries after a collision. This bit is applicable only to Half-Duplex mode and is reserved (RO) in Full-Duplex-only configuration. <ul style="list-style-type: none"><li>• 00: <math>k = \min(n, 10)</math></li><li>• 01: <math>k = \min(n, 8)</math></li><li>• 10: <math>k = \min(n, 4)</math></li><li>• 11: <math>k = \min(n, 1)</math>,</li></ul> where $n$ = retransmission attempt. The random integer $r$ takes the value in the range $0 \leq r < 2^k$	00	R_W
4	DC: Deferral Check  When this bit is set, the deferral check function is enabled in the GMAC. The GMAC will issue a Frame Abort status, along with the excessive deferral error bit set in the transmit frame status when the transmit state machine is deferred for more than 24,288 bit times in 10/100-Mbps mode. If the Core is configured for 1000 Mbps operation, or if the Jumbo frame mode is enabled in 10/100-Mbps mode, the threshold for deferral is 155,680 bits times. Deferral begins when the transmitter is ready to transmit, but is prevented because of an active CRS (carrier sense) signal on the GMII/MII. Defer time is not cumulative. If the transmitter defers for 10,000 bit times, then transmits, collides, backs off, and then has to defer again after completion of back-off, the deferral timer resets to 0 and restarts.  When this bit is reset, the deferral check function is disabled and the GMAC defers until the CRS signal goes inactive. This bit is applicable only in Half-Duplex mode and is reserved (RO) in Full-Duplex-only configuration.	0	R_W
3	TE: Transmitter Enable  When this bit is set, the transmit state machine of the GMAC is enabled for transmission on the GMII/MII. When this bit is reset, the GMAC transmit state machine is disabled after the completion of the transmission of the current frame, and will not transmit any further frames.	0	R_W

**Table 5-20 Register 0 (MAC Configuration Register) (Continued)**

Field	Description	Reset	Access
2	RE: Receiver Enable When this bit is set, the receiver state machine of the GMAC is enabled for receiving frames from the GMII/MII. When this bit is reset, the GMAC receive state machine is disabled after the completion of the reception of the current frame, and will not receive any further frames from the GMII/MII.	0	R_W
1:0	Reserved	00	RO

**5.2.2.2 Register 1 (MAC Frame Filter)**

The MAC Frame Filter register contains the filter controls for receiving frames. Some of the controls from this register go to the address check block of the MAC, which performs the first level of address filtering. The second level of filtering is performed on the incoming frame, based on other controls such as Pass Bad Frames and Pass Control Frames.

**Table 5-21 Register 1 (MAC Frame Filter)**

Field	Description	Reset	Access
31	RA: Receive All When this bit is set, the GMAC Receiver module passes to the Application all frames received irrespective of whether they pass the address filter. The result of the SA/DA filtering is updated (pass or fail) in the corresponding bits in the Receive Status Word. When this bit is reset, the Receiver module passes to the Application only those frames that pass the SA/DA address filter.	0	R_W
30:11	Reserved	0000_000H	RO
10	HPF: Hash or Perfect Filter When set, this bit configures the address filter to pass a frame if it matches either the perfect filtering or the hash filtering as set by HMC or HUC bits. When low and if the HUC/HMC bit is set, the frame is passed only if it matches the Hash filter. This bit is reserved (and RO) if the Hash filter is not selected during core configuration.	0	R_W
9	SAF: Source Address Filter Enable The GMAC core compares the SA field of the received frames with the values programmed in the enabled SA registers. If the comparison matches, then the SAMatch bit of RxStatus Word is set high. When this bit is set high and the SA filter fails, the GMAC drops the frame. When this bit is reset, then the GMAC Core forwards the received frame to the application and with the updated SA Match bit of the RxStatus depending on the SA address comparison.	0	R_W
8	SAIF: SA Inverse Filtering When this bit is set, the Address Check block operates in inverse filtering mode for the SA address comparison. The frames whose SA matches the SA registers will be marked as failing the SA Address filter. When this bit is reset, frames whose SA does not match the SA registers will be marked as failing the SA Address filter.	0	R_W

**Table 5-21 Register 1 (MAC Frame Filter) (Continued)**

Field	Description	Reset	Access
7:6	<p><b>PCF:</b> Pass Control Frames</p> <p>These bits control the forwarding of all control frames (including unicast and multicast PAUSE frames). Note that the processing of PAUSE control frames depends only on RFE of Flow Control Register[2].</p> <ul style="list-style-type: none"> <li>• 00: GMAC filters all control frames from reaching the application.</li> <li>• 01: GMAC forwards all control frames except PAUSE control frames to application even if they fail the Address filter.</li> <li>• 10: GMAC forwards all control frames to application even if they fail the Address Filter.</li> <li>• 11: GMAC forwards control frames that pass the Address Filter.</li> </ul>	0	R_W
5	<p><b>DBF:</b> Disable Broadcast Frames</p> <p>When this bit is set, the AFM module filters all incoming broadcast frames.</p> <p>When this bit is reset, the AFM module passes all received broadcast frames.</p>	0	R_W
4	<p><b>PM:</b> Pass All Multicast</p> <p>When set, this bit indicates that all received frames with a multicast destination address (first bit in the destination address field is '1') are passed.</p> <p>When reset, filtering of multicast frame depends on HMC bit.</p>	0	R_W
3	<p><b>DAIF:</b> DA Inverse Filtering</p> <p>When this bit is set, the Address Check block operates in inverse filtering mode for the DA address comparison for both unicast and multicast frames.</p> <p>When reset, normal filtering of frames is performed.</p>	0	R_W
2	<p><b>HMC:</b> Hash Multicast</p> <p>When set, MAC performs destination address filtering of received multicast frames according to the hash table.</p> <p>When reset, the MAC performs a perfect destination address filtering for multicast frames, that is, it compares the DA field with the values programmed in DA registers.</p> <p>If Hash Filter is not selected during coreKit configuration, this bit is reserved (and RO).</p>	0	R_W
1	<p><b>HUC:</b> Hash Unicast</p> <p>When set, MAC performs destination address filtering of unicast frames according to the hash table.</p> <p>When reset, the MAC performs a perfect destination address filtering for unicast frames, that is, it compares the DA field with the values programmed in DA registers.</p> <p>If Hash Filter is not selected during coreKit configuration, this bit is reserved (and RO).</p>	0	R_W
0	<p><b>PR:</b> Promiscuous Mode</p> <p>When this bit is set, the Address Filter module passes all incoming frames regardless of its destination or source address. The SA/DA Filter Fails status bits of the Receive Status Word will always be cleared when PR is set.</p>	0	R_W

### 5.2.2.3 Register 2 (Hash Table High Register)

The 64-bit Hash table is used for group address filtering. For hash filtering, the contents of the destination address in the incoming frame is passed through the CRC logic, and the upper 6 bits of the CRC register are used to index the contents of the Hash table. The most significant bit determines the register to be used (Hash Table High/Hash Table Low), and the other 5 bits determine which bit within the register. A hash value of 5b'00000 selects Bit 0 of the selected register, and a value of 5b'11111 selects Bit 31 of the selected register.

For example, if the DA of the incoming frame is received as 0x1F52419CB6AF (0x1F is the first byte received on GMII interface), then the internally calculated 6-bit Hash value is 0x2C and the HTH register bit[12] is checked for filtering. If the DA of the incoming frame is received as 0xA00A98000045, then the calculated 6-bit Hash value is 0x07 and the HTL register bit[7] is checked for filtering.



**Note** To help you program the hash table, a sample C routine that generates a DA's 6-bit hash has been included in your workspace's resources/ directory.

If the corresponding bit value of the register is 1'b1, the frame is accepted. Otherwise, it is rejected. If the PM (Pass All Multicast) bit is set in Register 1, then all multicast frames are accepted regardless of the multicast hash values.

If the Hash Table register is configured to be double-synchronized to the (G)MII clock domain, the synchronization is triggered only when Bits[31:24] (in Little-Endian mode) or Bits[7:0] (in Big-Endian mode) of the Hash Table High/Low registers are written to. Please note that consecutive writes to these register should be performed only after at least 4 clock cycles in the destination clock domain when double-synchronization is enabled.

The Hash Table High register contains the higher 32 bits of the Hash table.

**Table 5-22 Register 2 (Hash Table High Register)**

Field	Description	Reset	Access
31:0	HTH: Hash Table High This field contains the upper 32 bits of Hash table.	0000_0000H	R_W

### 5.2.2.4 Register 3 (Hash Table Low Register)

The Hash Table Low register contains the lower 32 bits of the Hash table. Both Register 2 and Register 3 and corresponding HMC and HUC bits in Filter Register are reserved if the Hash Filter Function is disabled during coreKit configuration.

**Table 5-23 Register 3 (Hash Table Low Register)**

Field	Description	Reset	Access
31:0	HTL: Hash Table Low This field contains the lower 32 bits of Hash table.	0000_0000H	R_W

### 5.2.2.5 Register 4 (GMII Address Register)

The GMII Address register controls the management cycles to the external PHY through the management interface.

**Table 5-24 Register 4 (GMII Address Register)**

Field	Description	Reset	Access																																																		
31:16	Reserved	0000H	RO																																																		
15:11	PA: Physical Layer Address This field tells which of the 32 possible PHY devices are being accessed.	00H	R_W																																																		
10:6	GR: GMII Register These bits select the desired GMII register in the selected PHY device.	00H	R_W																																																		
5:2	CR: CSR Clock Range The CSR Clock Range selection determines the frequency of the MDC clock as per the clk_csr_i frequency used in your design. The suggested range of clk_csr_i frequency applicable for each value below (when Bit[5] = 0) ensures that the MDC clock is approximately between the frequency range 1.0 MHz - 2.5 MHz.	0000	R_W																																																		
<table> <thead> <tr> <th>Selection</th> <th>clk_csr_i</th> <th>MDC Clock</th> <th></th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>60-100 MHz</td> <td>clk_csr_i/42</td> <td></td> </tr> <tr> <td>0001</td> <td>100-150 MHz</td> <td>clk_csr_i/62</td> <td></td> </tr> <tr> <td>0010</td> <td>20-35 MHz</td> <td>clk_csr_i/16</td> <td></td> </tr> <tr> <td>0011</td> <td>35-60 MHz</td> <td>clk_csr_i/26</td> <td></td> </tr> <tr> <td>0100</td> <td>150-250 MHz</td> <td>clk_csr_i/102</td> <td></td> </tr> <tr> <td>0101</td> <td>250-300 MHz</td> <td>clk_csr_i/124</td> <td></td> </tr> <tr> <td>0110, 0111</td> <td>Reserved</td> <td></td> <td></td> </tr> </tbody> </table> <p>When bit 5 is set, you can achieve MDC clock of frequency higher than the IEEE 802.3 specified frequency limit of 2.5 MHz and program a clock divider of lower value. For example, when clk_csr_i is of frequency 100 Mhz and you program these bits as "1010", then the resultant MDC clock will be of 12.5 Mhz which is outside the limit of IEEE 802.3 specified range. Please program the values given below only if the interfacing chips supports faster MDC clocks.</p> <table> <thead> <tr> <th>Selection</th> <th>MDC Clock</th> </tr> </thead> <tbody> <tr> <td>1000</td> <td>clk_csr_i/4</td> </tr> <tr> <td>1001</td> <td>clk_csr_i/6</td> </tr> <tr> <td>1010</td> <td>clk_csr_i/8</td> </tr> <tr> <td>1011</td> <td>clk_csr_i/10</td> </tr> <tr> <td>1100</td> <td>clk_csr_i/12</td> </tr> <tr> <td>1101</td> <td>clk_csr_i/14</td> </tr> <tr> <td>1110</td> <td>clk_csr_i/16</td> </tr> <tr> <td>1111</td> <td>clk_csr_i/18</td> </tr> </tbody> </table>	Selection	clk_csr_i	MDC Clock		0000	60-100 MHz	clk_csr_i/42		0001	100-150 MHz	clk_csr_i/62		0010	20-35 MHz	clk_csr_i/16		0011	35-60 MHz	clk_csr_i/26		0100	150-250 MHz	clk_csr_i/102		0101	250-300 MHz	clk_csr_i/124		0110, 0111	Reserved			Selection	MDC Clock	1000	clk_csr_i/4	1001	clk_csr_i/6	1010	clk_csr_i/8	1011	clk_csr_i/10	1100	clk_csr_i/12	1101	clk_csr_i/14	1110	clk_csr_i/16	1111	clk_csr_i/18			
Selection	clk_csr_i	MDC Clock																																																			
0000	60-100 MHz	clk_csr_i/42																																																			
0001	100-150 MHz	clk_csr_i/62																																																			
0010	20-35 MHz	clk_csr_i/16																																																			
0011	35-60 MHz	clk_csr_i/26																																																			
0100	150-250 MHz	clk_csr_i/102																																																			
0101	250-300 MHz	clk_csr_i/124																																																			
0110, 0111	Reserved																																																				
Selection	MDC Clock																																																				
1000	clk_csr_i/4																																																				
1001	clk_csr_i/6																																																				
1010	clk_csr_i/8																																																				
1011	clk_csr_i/10																																																				
1100	clk_csr_i/12																																																				
1101	clk_csr_i/14																																																				
1110	clk_csr_i/16																																																				
1111	clk_csr_i/18																																																				
1	GW: GMII Write When set, this bit tells the PHY that this will be a Write operation using the GMII Data register. If this bit is not set, this will be a Read operation, placing the data in the GMII Data register.	0	R_W																																																		

**Table 5-24 Register 4 (GMII Address Register) (Continued)**

Field	Description	Reset	Access
0	GB: GMII Busy This bit should read a logic 0 before writing to Register 4 and Register 5. This bit must also be set to 0 during a Write to Register 4. During a PHY register access, this bit will be set to 1'b1 by the Application to indicate that a Read or Write access is in progress. Register 5 (GMII Data) should be kept valid until this bit is cleared by the GMAC during a PHY Write operation. The Register 5 is invalid until this bit is cleared by the GMAC during a PHY Read operation. The Register 4 (GMII Address) should not be written to until this bit is cleared.	0	R_WS_SC

**5.2.2.6 Register 5 (GMII Data Register)**

The GMII Data register stores Write data to be written to the PHY register located at the address specified in Register 4. Register 5 also stores Read data from the PHY register located at the address specified by Register 4.

**Table 5-25 Register 5 (GMII Data Register)**

Field	Description	Reset	Access
31:16	Reserved	0000H	RO
15:0	GD: GMII Data This contains the 16-bit data value read from the PHY after a Management Read operation or the 16-bit data value to be written to the PHY before a Management Write operation.	0000H	R_W

**5.2.2.7 Register 6 (Flow Control Register)**

The Flow Control register controls the generation and reception of the Control (Pause Command) frames by the GMAC's Flow control module. A Write to a register with the Busy bit set to '1' triggers the Flow Control block to generate a Pause Control frame. The fields of the control frame are selected as specified in the 802.3x specification, and the Pause Time value from this register is used in the Pause Time field of the control frame. The Busy bit remains set until the control frame is transferred onto the cable. The Host must make sure that the Busy bit is cleared before writing to the register.

**Table 5-26 Register 6 (Flow Control Register)**

Field	Description	Reset	Access
31:16	PT: Pause Time This field holds the value to be used in the Pause Time field in the transmit control frame. If the Pause Time bits is configured to be double-synchronized to the (G)MII clock domain, then consecutive writes to this register should be performed only after at least 4 clock cycles in the destination clock domain.	0000H	R_W
15:8	Reserved	000H	RO

**Table 5-26 Register 6 (Flow Control Register) (Continued)**

Field	Description	Reset	Access
7	DZPQ: Disable Zero-Quanta Pause When set, this bit disables the automatic generation of Zero-Quanta Pause Control frames on the deassertion of the flow-control signal from the FIFO layer (MTL or external sideband flow control signal sdb_flowctrl_i/mti_flowctrl_i). When this bit is reset, normal operation with automatic Zero-Quanta Pause Control frame generation is enabled.	0	R_W
6	Reserved	0	RO
5:4	PLT: Pause Low Threshold This field configures the threshold of the PAUSE timer at which the input flow control signal mti_flowctrl_i (or sdb_flowctrl_i) is checked for automatic retransmission of PAUSE Frame. The threshold values should be always less than the Pause Time configured in Bits[31:16]. For example, if PT = 100H (256 slot-times), and PLT = 01, then a second PAUSE frame is automatically transmitted if the mti_flowctrl_i signal is asserted at 228 (256 – 28) slot-times after the first PAUSE frame is transmitted. <b>Selection Threshold</b> 00 Pause time minus 4 slot times 01 Pause time minus 28 slot times 10 Pause time minus 144 slot times 11 Pause time minus 256 slot times Slot time is defined as time taken to transmit 512 bits (64 bytes) on the GMII/MII interface.	00	R_W
3	UP: Unicast Pause Frame Detect When this bit is set, the GMAC will detect the Pause frames with the station's unicast address specified in MAC Address0 High Register and MAC Address0 Low Register, in addition to the detecting Pause frames with the unique multicast address. When this bit is reset, the GMAC will detect only a Pause frame with the unique multicast address specified in the 802.3x standard.	0	R_W
2	RFE: Receive Flow Control Enable When this bit is set, the GMAC will decode the received Pause frame and disable its transmitter for a specified (Pause Time) time. When this bit is reset, the decode function of the Pause frame is disabled.	0	R_W
1	TFE: Transmit Flow Control Enable In Full-Duplex mode, when this bit is set, the GMAC enables the flow control operation to transmit Pause frames. When this bit is reset, the flow control operation in the GMAC is disabled, and the GMAC will not transmit any Pause frames. In Half-Duplex mode, when this bit is set, the GMAC enables the back-pressure operation. When this bit is reset, the backpressure feature is disabled.	0	R_W

**Table 5-26 Register 6 (Flow Control Register) (Continued)**

Field	Description	Reset	Access
0	<p>FCB/BPA: Flow Control Busy/Backpressure Activate</p> <p>This bit initiates a Pause Control frame in Full-Duplex mode and activates the backpressure function in Half-Duplex mode if TFE bit is set.</p> <p>In Full-Duplex mode, this bit should be read as 1'b0 before writing to the Flow Control register. To initiate a Pause control frame, the Application must set this bit to 1'b1. During a transfer of the Control Frame, this bit will continue to be set to signify that a frame transmission is in progress. After the completion of Pause control frame transmission, the GMAC will reset this bit to 1'b0. The Flow Control register should not be written to until this bit is cleared.</p> <p>In Half-Duplex mode, when this bit is set (and TFE is set), then backpressure is asserted by the GMAC Core. During backpressure, when the GMAC receives a new frame, the transmitter starts sending a JAM pattern resulting in a collision. This control register bit is logically OR'ed with the mti_flowctrl_i input signal for the backpressure function. When the GMAC is configured to Full-Duplex mode, the BPA is automatically disabled.</p>	0	R_WS_SC(FCB) R_W (BPA)

### 5.2.2.8 Register 7 (VLAN Tag Register)

The VLAN Tag register contains the IEEE 802.1Q VLAN Tag to identify the VLAN frames. The MAC compares the 13th and 14th bytes of the receiving frame (Length/Type) with 16'h8100, and the following 2 bytes are compared with the VLAN tag; if a match occurs, it sets the received VLAN bit in the receive frame status. The legal length of the frame is increased from 1518 bytes to 1522 bytes.

If the VLAN Tag register is configured to be double-synchronized to the (G)MII clock domain, then consecutive writes to these register should be performed only after at least 4 clock cycles in the destination clock domain.

**Table 5-27 Register 7 (VLAN Tag Register)**

Field	Description	Reset	Access
31:17	Reserved	0000H	RO
16	<p>ETV: Enable 12-Bit VLAN Tag Comparison</p> <p>When this bit is set, a 12-bit VLAN identifier, rather than the complete 16-bit VLAN tag, is used for comparison and filtering. Bits[11:0] of the VLAN tag are compared with the corresponding field in the received VLAN-tagged frame.</p> <p>When this bit is reset, all 16 bits of the received VLAN frame's fifteenth and sixteenth bytes are used for comparison.</p>	0	R/W
15:0	<p>VL: VLAN Tag Identifier for Receive Frames</p> <p>This contains the 802.1Q VLAN tag to identify VLAN frames, and is compared to the fifteenth and sixteenth bytes of the frames being received for VLAN frames. Bits[15:13] are the User Priority, Bit[12] is the Canonical Format Indicator (CFI) and bits[11:0] are the VLAN tag's VLAN Identifier (VID) field. When the ETV bit is set, only the VID (Bits[11:0]) is used for comparison.</p> <p>If VL (VL[11:0] if ETV is set) is all zeros, the GMAC does not check the fifteenth and sixteenth bytes for VLAN tag comparison, and declares all frames with a Type field value of 0x8100 to be VLAN frames.</p>	0000H	R_W

### 5.2.2.9 Register 8 (Version Register)

The Version register's contents identify the version of the core. This register contains two bytes, one of which Synopsys uses to identify the core release number, and the other of which you set during coreKit configuration.

**Table 5-28 Register 8 (Version Register)**

Field	Description	Reset	Access
15:8	User-defined version (configured with coreKit)	xxH	RO
7:0	Synopsys-defined version (3.5)	35H	RO

### 5.2.2.10 Register 9 (Debug Register)

This debug register gives the status of all the main modules of the transmit and receive data-paths and the FIFOs. An all-zero status indicates that the MAC core is in idle state (and FIFOs are empty) and no activity is going on in the data-paths.

**Table 5-29 Register 9 (Debug Register)**

Field	Description	Reset	Access
31:26	Reserved		
25	When high, it indicates that the MTL TxStatus FIFO is full and hence the MTL will not be accepting any more frames for transmission.		
24	When high, it indicates that the MTL TxFIFO is not empty and has some data left for transmission.		
23	Reserved		
22	When high, it indicates that the MTL TxFIFO Write Controller is active and transferring data to the TxFIFO.		
21:20	This indicates the state of the TxFIFO read Controller:		
	<ul style="list-style-type: none"> <li>• 00: IDLE state</li> <li>• 01: READ state (transferring data to MAC transmitter)</li> <li>• 10: Waiting for TxStatus from MAC transmitter</li> <li>• 11: Writing the received TxStatus or flushing the TxFIFO</li> </ul>		
19	When high, it indicates that the MAC transmitter is in PAUSE condition (in full-duplex only) and hence will not schedule any frame for transmission		
18:17	This indicates the state of the MAC Transmit Frame Controller module (section 3.7.1.2):		
	<ul style="list-style-type: none"> <li>• 00: IDLE</li> <li>• 01: Waiting for Status of previous frame or IFG/backoff period to be over</li> <li>• 10: Generating and transmitting a PAUSE control frame (in full duplex mode)</li> <li>• 11: Transferring input frame for transmission</li> </ul>		
16	When high, it indicates that the MAC GMII/MII transmit protocol engine is actively transmitting data and not in IDLE state.		

**Table 5-29 Register 9 (Debug Register) (Continued)**

Field	Description	Reset	Access
15:10	Reserved		
9:8	This gives the status of the RxFIFO Fill-level: • 00: RxFIFO Empty • 01: RxFIFO fill-level below flow-control de-activate threshold • 10: RxFIFO fill-level above flow-control activate threshold • 11: RxFIFO Full		
7	Reserved		
6:5	It gives the state of the RxFIFO read Controller: • 00: IDLE state • 01: Reading frame data • 10: Reading frame status (or time-stamp) • 11: Flushing the frame data and Status		
4	When high, it indicates that the MTL RxFIFO Write Controller is active and transferring a received frame to the FIFO.		
3	Reserved		
2:1	When high, it indicates the active state of the small FIFO Read and Write controllers respectively of the MAC receive Frame Controller module (section 3.7.4.4)		
0	When high, it indicates that the MAC GMII/MII receive protocol engine is actively receiving data and not in IDLE state.		

### 5.2.2.11 Register 14 (Interrupt Status Register)

The Interrupt Status register contents identify the events in the GMAC-CORE that can generate interrupt. Note that all the interrupt events are generated only when the corresponding optional feature is selected during coreKit configuration and enabled during operation. Hence, these bits are reserved when the corresponding features is not present in the core.

**Table 5-30 Register 14 (Interrupt Status Register)**

Field	Description	Reset	Access
15:10	Reserved	000H	RO

**Table 5-30 Register 14 (Interrupt Status Register) (Continued)**

Field	Description	Reset	Access
9	<p>Time Stamp Interrupt Status When Advanced Time Stamping feature is enabled, this bit is set when</p> <ol style="list-style-type: none"> <li>1. The system time value equals or exceeds the value specified in the Target Time High and Low registers, or</li> <li>2. There is an overflow in the seconds register, or</li> <li>3. When the Auxiliary snapshot trigger is asserted.</li> </ol> <p>This bit is cleared on reading the byte 0 of the “Time Stamp Status register (see <a href="#">“Register 458 (Time Stamp Status Register)” on page 306</a>). Otherwise, when default Time-Stamping is enabled, this bit when set indicates that the system time value equals or exceeds the value specified in the Target Time registers. In this mode, this bit is cleared after the completion of the read of this Interrupt Status Register[9].</p> <p>In all other modes, this bit is reserved.</p>	0	RO/ R_SS_RC
8	Reserved	000H	RO
7	<p>MMC Receive Checksum Offload Interrupt Status This bit is set high whenever an interrupt is generated in the <a href="#">MMC Receive Checksum Offload Interrupt Register</a>. This bit is cleared when all the bits in this interrupt register are cleared.</p> <p>This bit is only valid when the optional MMC module and Checksum Offload Engine (Type 2) are selected during configuration.</p>	0	RO
6	<p>MMC Transmit Interrupt Status This bit is set high whenever an interrupt is generated in the <a href="#">MMC Transmit Interrupt Register</a>. This bit is cleared when all the bits in this interrupt register are cleared.</p> <p>This bit is only valid when the optional MMC module is selected during configuration.</p>	0	RO
5	<p>MMC Receive Interrupt Status This bit is set high whenever an interrupt is generated in the <a href="#">MMC Receive Interrupt Register</a>. This bit is cleared when all the bits in this interrupt register are cleared.</p> <p>This bit is only valid when the optional MMC module is selected during configuration.</p>	0	RO
4	<p>MMC Interrupt Status This bit is set high whenever any of bits 7:5 is set high and cleared only when all of these bits are low. This bit is valid only when the optional MMC module is selected during configuration.</p>	0	RO
3	<p>PMT Interrupt Status This bit is set whenever a Magic packet or Wake-on-LAN frame is received in Power-Down mode (See bits 5 and 6 in the PMT Control and Status register <a href="#">“PMT Control and Status Register” on page 147</a>). This bit is cleared when both bits[6:5] are cleared due to a read operation to the PMT Control and Status register.</p> <p>This bit is valid only when the optional PMT module is selected during configuration</p>	0	RO

**Table 5-30 Register 14 (Interrupt Status Register) (Continued)**

Field	Description	Reset	Access
2	<p>PCS Auto-Negotiation Complete</p> <p>This bit is set when the Auto-negotiation is completed in the TBI/RTBI/SGMII PHY interface (Bit 5 in the AN Status Register in <a href="#">"Register 49 (AN Status Register)"</a> on page <a href="#">297</a>). This bit is cleared when the user makes a read operation to the AN Status register.</p> <p>This bit is valid only when the optional TBI/RTBI/SGMII PHY interface is selected during configuration and operation</p>	0	RO
1	<p>PCS Link Status Changed</p> <p>This bit is set due to any change in Link Status in the TBI/RTBI/SGMII PHY interface (Bit 2 in the AN Status Register in <a href="#">"Register 49 (AN Status Register)"</a> on page <a href="#">297</a>). This bit is cleared when the user makes a read operation the AN Status register.</p> <p>This bit is valid only when the optional TBI/RTBI/SGMII PHY interface is selected during configuration and operation</p>	0	RO
0	<p>RGMII/SMII Interrupt Status</p> <p>This bit is set due to any change in value of the Link Status of RGMII/SMII interface (Bit 3 of RGMII/SMII Status Register in <a href="#">"Register 54 (SGMII/RGMII/SMII Status Register)"</a> on page <a href="#">300</a>). This bit is cleared when the user makes a read operation the SGMII/RGMII/SMII Status register.</p> <p>This bit is valid only when the optional RGMII/SMII PHY interface is selected during configuration and operation.</p>	0	RO

### 5.2.2.12 Register 15 (Interrupt Mask Register)

The Interrupt Mask Register bits enables the user to mask the interrupt signal due to the corresponding event in the Interrupt Status Register. The interrupt signal is sbd\_intr\_o in GMAC-AHB and GMAC-DMA configuration while the interrupt signal is mci\_intr\_o in the GMAC-MTL and GMAC-CORE configuration.

**Table 5-31 Register 15 (Interrupt Mask Register)**

Field	Description	Reset	Access
15:10	Reserved	000H	RO
9	Time Stamp Interrupt Mask When set, this bit disables the assertion of the interrupt signal due to the setting of Time Stamp Interrupt Status bit in Register 14. This bit is only active when IEEE1588 time stamping is enabled. In all other modes, this bit is reserved.	0	R_W
8:4	Reserved	000H	RO
3	PMT Interrupt Mask This bit when set, will disable the assertion of the interrupt signal due to the setting of PMT Interrupt Status bit in Register 14.	0	R_W
2	PCS AN Completion Interrupt Mask This bit when set, will disable the assertion of the interrupt signal due to the setting of PCS Auto-negotiation complete bit in Register 14 caused due to the completion of Auto-negotiation event.	0	R_W
1	PCS Link Status Interrupt Mask This bit when set, will disable the assertion of the interrupt signal due to the setting of PCS Link-status changed bit in Register 14 caused due to change in link-status event.	0	R_W
0	RGMII/SMII Interrupt Mask This bit when set, will disable the assertion of the interrupt signal due to the setting of RGMII/SMII Interrupt Status bit in Register 14.	0	R_W

### 5.2.2.13 Register 16 (MAC Address0 High Register)

The MAC Address0 High register holds the upper 16 bits of the 6-byte first MAC address of the station. Note that the first DA byte that is received on the (G)MII interface corresponds to the LS Byte (Bits [7:0]) of the MAC Address Low register. For example, if 0x112233445566 is received (0x11 is the first byte) on the (G)MII as the destination address, then the MacAddress0 Register [47:0] is compared with 0x665544332211.

If the MAC address registers are configured to be double-synchronized to the (G)MII clock domains, then the synchronization is triggered only when Bits[31:24] (in Little-Endian mode) or Bits[7:0] (in Big-Endian mode) of the MAC Address Low Register (Register 17) are written to. Please note that consecutive writes to this Address Low Register should be performed only after at least 4 clock cycles in the destination clock domain for proper synchronization updates.

**Table 5-32 Register 16 (MAC Address0 High Register)**

Field	Description	Reset	Access
31	MO: Always 1.	1	RO
30:16	Reserved	0000H	RO
15:0	A[47:32]: MAC Address0 [47:32]  This field contains the upper 16 bits (47:32) of the 6-byte first MAC address. This is used by the MAC for filtering for received frames and for inserting the MAC address in the Transmit Flow Control (PAUSE) Frames.	FFFFH	R_W

### 5.2.2.14 Register 17 (MAC Address0 Low Register)

The MAC Address0 Low register holds the lower 32 bits of the 6-byte first MAC address of the station.

**Table 5-33 Register 17 (MAC Address0 Low Register)**

Field	Description	Reset	Access
31:0	A[31:0]: MAC Address0 [31:0]  This field contains the lower 32 bits of the 6-byte first MAC address. This is used by the MAC for filtering for received frames and for inserting the MAC address in the Transmit Flow Control (PAUSE) Frames.	FFFF_FFFFH	R_W

### 5.2.2.15 Register 18 (MAC Address1 High Register)

The MAC Address1 High register holds the upper 16 bits of the 6-byte second MAC address of the station.

If the MAC address registers are configured to be double-synchronized to the (G)MII clock domains, then the synchronization is triggered only when Bits[31:24] (in Little Endian mode) or Bits[7:0] (in Big Endian mode) of the MAC Address Low Register (Register 19) are written to. Consecutive writes to this Address Low Register must be performed only after at least 4 clock cycles in the destination clock domain for proper synchronization updates.

**Table 5-34 Register 18 (MAC Address1 High Register)**

Field	Description	Reset	Access
31	AE: Address Enable When this bit is set, the Address filter module uses the second MAC address for perfect filtering. When reset, the address filter module will ignore the address for filtering.	0	R_W
30	SA: Source Address When this bit is set, the MAC Address1[47:0] is used to compare with the SA fields of the received frame. When this bit is reset, the MAC Address1[47:0] is used to compare with the DA fields of the received frame.	0	R_W
29:24	MBC: Mask Byte Control These bits are mask control bits for comparison of each of the MAC Address bytes. When set high, the GMAC core does not compare the corresponding byte of received DA/SA with the contents of MAC Address1 registers. Each bit controls the masking of the bytes as follows: <ul style="list-style-type: none"><li>• Bit 29: Register 18[15:8]</li><li>• Bit 28: Register 18[7:0]</li><li>• Bit 27: Register 19[31:24]</li><li>...</li><li>• Bit 24: Register 19[7:0]</li></ul>	000000	R_W
23:16	Reserved	00H	RO
15:0	A[47:32]: MAC Address1 [47:32] This field contains the upper 16 bits (47:32) of the 6-byte second MAC address.	FFFFH	R_W

### 5.2.2.16 Register 19 (MAC Address1 Low Register)

The MAC Address1 Low register holds the lower 32 bits of the 6-byte second MAC address of the station.

**Table 5-35 Register 17 (MAC Address0 Low Register)**

Field	Description	Reset	Access
31:0	A[31:0]: MAC Address1 [31:0] This field contains the lower 32 bits of the 6-byte second MAC address. The content of this field is undefined until loaded by the Application after the initialization process.	FFFF_FFFFH	R_W


**Note**

- The descriptions for registers 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, and 46 (MAC Address2 High Register through MAC Address15 High Register) are the same as for the Register 18 MAC Address1 High Register.
- The descriptions for registers 21, 23, 25, 27, 29, 31, 33, 35, 37, 38, 41, 43, 45, and 47 (MAC Address2 Low Register through MAC Address15 Low Register) are the same as for the Register 19 MAC Address1 Low Register.
- The descriptions for registers 512, 514, 516, 518, 520, 522, 524, 526, 528, 530, 532, 534, 536, 538, 540, and 542 (MAC Address16 High Register through MAC Address31 High Register) are the same as for the Register 18 MAC Address1 High Register.
- The descriptions for registers 513, 515, 517, 519, 521, 523, 525, 527, 529, 531, 533, 535, 537, 539, 541, and 543 (MAC Address16 Low Register through MAC Address31 Low Register) is the same as for the Register 19 MAC Address1 Low Register.

### 5.2.2.17 Register 48 (AN Control Register)

The AN Control register enables and/or restarts auto-negotiation. It also enables PCS loopback. This register is optional and is present only when the core is configured for TBI/SGMII/RTBI PHY interfaces.

**Table 5-36 Register 48 (AN Control Register)**

Field	Description	Reset	Access
31:19	Reserved	0000H	RO
18	<b>SGMII RAL Control</b> When set, this bit forces the SGMII RAL block to operate in the speed configured in the MAC Configuration register's Speed and Port Select bits. This is useful when the SGMII interface is used in a direct MAC-MAC connection (without a PHY) and any MAC must reconfigure the speed. When reset, the SGMII RAL block operates according to the link speed status received on the SGMII (from the PHY). This bit is reserved (and RO) if the SGMII PHY interface is not selected during coreKit configuration.	0	R_W
17	<b>LR: Lock to Reference</b> When set, enables the PHY to lock its PLL to the 125 MHz reference clock. This bit controls the <code>pcs_lck_ref_o</code> signal on the TBI/RTBI/SGMII.	0	R_W
16	<b>ECD: Enable Comma Detect</b> When set, enables the PHY for comma detect and word resynchronization. This bit controls the <code>pcs_en_cdet_o</code> signal on the TBI/RTBI/SGMII.	0	R_W
15	Reserved	0	RO
14	<b>ELE: External Loopback Enable</b> When set, will cause the PHY to loopback the transmit data into the receive path. The <code>pcs_ewrap_o</code> signal will be asserted high when this bit is set.	0	R_W
13	Reserved	0	RO
12	<b>ANE: Auto-Negotiation Enable</b> When set, will enable the GMAC to perform auto-negotiation with the link partner. Clearing this bit will disable auto-negotiation.	0	R_W

**Table 5-36 Register 48 (AN Control Register) (Continued)**

Field	Description	Reset	Access
11:10	Reserved	00	RO
9	RAN: Restart Auto-Negotiation When set, will cause auto-negotiation to restart if the ANE is set. This bit is self-clearing after auto-negotiation starts. This bit should be cleared for normal operation.	0	R_W_SC
8:0	Reserved	00H	RO

**5.2.2.18 Register 49 (AN Status Register)**

The AN Status register indicates the link and the auto-negotiation status. This register is optional and is present only when the core is configured for TBI/SGMII/RTBI PHY interfaces.

**Table 5-37 Register 49 (AN Status Register)**

Field	Description	Reset	Access
31:9	Reserved	00_0000H	RO
8	ES: Extended Status This bit is tied to high, because the GMAC supports extended status information in Register 53.	1	RO
7:6	Reserved	00	RO
5	ANC: Auto-Negotiation Complete When set, this bit indicates that the auto-negotiation process is completed. This bit is cleared when auto-negotiation is reinitiated.	0	RO
4	Reserved	0	RO
3	ANA: Auto-Negotiation Ability This bit is always high, because the GMAC supports auto-negotiation.	1	RO
2	LS: Link Status When set, this bit indicates that the link is up. When cleared, this bit indicates that the link is down.	0	R_SS_SC_LLO
1:0	Reserved	00	RO



LLO indicates that when the bit is reset to 1'b0, its value is registered until the application reads the register. This bit is only updated after the read.

### 5.2.2.19 Register 50 (Auto-Negotiation Advertisement Register)

The Auto-Negotiation Advertisement register indicates the link and the auto-negotiation status. This register is optional and is present only when the core is configured for TBI/RTBI PHY interfaces.

**Table 5-38 Register 50 (Auto-Negotiation Advertisement Register)**

Field	Description	Reset	Access
31:16	Reserved	0000H	RO
15	NP: Next Page Support This bit is tied to low, because the GMAC does not support the next page.	0	RO
14	Reserved	0	RO
13:12	RFE: Remote Fault Encoding These 2 bits provide a remote fault encoding, indicating to a link partner that a fault or error condition has occurred. The encoding of these 2 bits is defined in IEEE 802.3z, Section 37.2.1.5.	00	R_W
11:9	Reserved	000	RO
8:7	PSE: Pause Encoding These 2 bits provide an encoding for the PAUSE bits, indicating that the GMAC is capable of configuring the PAUSE function as defined in IEEE 802.3x. The encoding of these 2 bits is defined in IEEE 802.3z, Section 37.2.1.4.	11	R_W
6	HD: Half-Duplex This bit, when set high, indicates that the GMAC supports Half-Duplex. This bit is tied to low (and RO) when the GMAC is configured for Full-Duplex-only operation.	1	R_W
5	FD: Full-Duplex This bit, when set high, indicates that the GMAC supports Full-Duplex.	1	R_W
4:0	Reserved	00000	RO

### 5.2.2.20 Register 51 (Auto-Negotiation Link Partner Ability Register)

The Auto-Negotiation Link Partner Ability register contains the advertised ability of the link partner. This register is optional and is present only when the core is configured for TBI/RTBI PHY interfaces.

**Table 5-39 Register 51 (Auto-Negotiation Link Partner Ability Register)**

Field	Description	Reset	Access
31:16	Reserved	0000H	RO
15	NP: Next Page Support When set, this bit indicates that more next page information is available. When cleared, this bit indicates that next page exchange is not desired.	0	RO
14	ACK: Acknowledge When set, this bit is used by the auto-negotiation function to indicate that the link partner has successfully received the GMAC's base page. When cleared, it indicates that a successful receipt of the base page has not been achieved.	0	RO

**Table 5-39 Register 51 (Auto-Negotiation Link Partner Ability Register) (Continued)**

Field	Description	Reset	Access
13:12	RFE: Remote Fault Encoding These 2 bits provide a remote fault encoding, indicating a fault or error condition of the link partner. Their encoding is defined in IEEE 802.3z, Section 37.2.1.5.	00	RO
11:9	Reserved	000	RO
8:7	PSE: Pause Encoding These 2 bits provide an encoding for the PAUSE bits, indicating that the link partner's capability of configuring the PAUSE function as defined in IEEE 802.3x. The encoding of these 2 bits is defined in IEEE 802.3z, Section 37.2.1.4.	00	RO
6	HD: Half-Duplex When set, this bit indicates that the link partner has the ability to operate in Half-Duplex mode. When cleared, the link partner does not have the ability to operate in Half-Duplex mode.	0	RO
5	FD: Full-Duplex When set, this bit indicates that the link partner has the ability to operate in Full-Duplex mode. When cleared, the link partner does not have the ability to operate in Full-Duplex mode.	0	RO
4:0	Reserved	00000	RO

### 5.2.2.21 Register 52 (Auto-Negotiation Expansion Register)

The Auto-Negotiation Expansion register indicates whether a new base page from the link partner has been received. This register is optional and is present only when the core is configured for TBI/RTBI PHY interfaces.

**Table 5-40 Register 52 (Auto-Negotiation Expansion Register)**

Field	Description	Reset	Access
31:3	Reserved	0000_0000H	RO
2	NPA: Next Page Ability This bit is tied to low, because the GMAC does not support next page function.	0	RO
1	NPR: New Page Received When set, this bit indicates that a new page has been received by the GMAC. This bit will be cleared when read.	0	RO
0	Reserved	0	RO

### 5.2.2.22 Register 53 (TBI Extended Status Register)

The TBI Extended Status register indicates all modes of operation of the GMAC. This register is optional and is present only when the core is configured for TBI/RTBI PHY interfaces.

**Table 5-41 Register 53 (TBI Extended Status Register)**

Field	Description	Reset	Access
31:16	Reserved	0000H	RO
15	GFD: 1000BASE-X Full-Duplex Capable This bit indicates that the GMAC is able to perform Full-Duplex, 1000BASE-X operations.	1	RO
14	GHD: 1000BASE-X Half-Duplex Capable This bit indicates that the GMAC is able to perform Half-Duplex, 1000BASE-X operations. This bit is tied to low when the GMAC is configured for Full-Duplex-only operation during coreKit configuration.	1	RO
13:0	Reserved	0000H	RO

### 5.2.2.23 Register 54 (SGMII/RGMII/SMII Status Register)

The SGMII/RGMII/SMII Status register indicates the status signals received by the SGMII/RGMII/SMII (whichever is selected at reset) from the PHY. This register is optional and is present only when the core is configured for RGMII/SGMII PHY interfaces.

**Table 5-42 Register 54 (SGMII/RGMII/SMII Status Register)**

Field	Description	Reset	Access
31:6	Reserved	000_0000H	RO
5	False Carrier Detected Indicates whether SMII PHY detected false carrier (1'b1). Reserved when the core is configured for SGMII/RGMII PHY interfaces.	0	RO
4	Jabber Timeout. Indicates whether there was jabber timeout error (1'b1) in received frame. Reserved when the core is configured for SGMII/RGMII PHY interfaces.	0	RO
3	Link Status. Indicates whether the link is up (1'b1) or down (1'b0).	0	RO
2:1	Link Speed. Indicates the current speed of the link: <ul style="list-style-type: none"> <li>• 00: 2.5 MHz</li> <li>• 01: 25 MHz</li> <li>• 10: 125 MHz</li> </ul> Bit 2 is reserved when the core is configured for SMII PHY interface.	<ul style="list-style-type: none"> <li>• 10 for SGMII</li> <li>• 00 for RGMII/SMII</li> </ul>	RO
0	Link Mode. Indicates the current mode of operation of the link: <ul style="list-style-type: none"> <li>• 1'b0: Half-Duplex mode</li> <li>• 1'b1: Full-Duplex mode</li> </ul>	0	RO

### 5.2.3 IEEE 1588 Time Stamp Registers

This section describes the updated registers required to support the IEEE 1588 functions. These registers, described at a high level in [Table 5-2](#), are described in detail in [Tables 5-43](#) through [5-51](#).

#### 5.2.3.1 Register 448 (Time Stamp Control Register)

This register controls the operation of the System Time generator and the snooping of PTP packets for time-stamping in the Receiver.



**Note** Bits [5:1] are reserved when External Time Stamp Input is selected during configuration. Bits[19:8] are reserved and read-only when Advanced Time Stamping is NOT enabled.

**Table 5-43 Register 448 (Time Stamp Control Register)**

Field	Description	Reset	Access
31:20	Reserved	000H	RO
19	ATSFC: Auxiliary Snapshot FIFO Clear When set, it resets the pointers of the Auxiliary Snapshot FIFO. This bit is cleared when the pointers are reset and the FIFO is empty. When this bit is high, auxiliary snapshots will get stored in the FIFO. This bit is reserved when "Add IEEE 1588 Auxiliary Snapshot" is not selected.	0	R_W_SC
18	TSENMACADDR: Enable MAC address for PTP frame filtering When set, uses the DA MAC address (that matches any MAC Address register except the default MAC address 0) to filter the PTP frames when PTP is sent directly over Ethernet.	0	R_W
17:16	TSCLKTYPE: Select the type of clock node The following are the options to select the type of clock node: <ul style="list-style-type: none"><li>• 00: Ordinary clock</li><li>• 01: Boundary clock</li><li>• 10: End-to-End Transparent clock</li><li>• 11: Peer-to-peer Transparent clock</li></ul>	0	R_W
15	TSMSTRENA: Enable Snapshot for Messages Relevant to Master When set, the snapshot is taken for messages relevant to master node only else snapshot is taken for messages relevant to slave node. This is valid only for ordinary clock and boundary clock node.	0	R_W
14	TSEVNNTENA: Enable Time Stamp Snapshot for Event Messages When set, the time stamp snapshot is taken for event messages only (SYNC, Delay_Req, Pdelay_Req or Pdelay_Resp). When reset snapshot is taken for all other messages except Announce, Management and Signaling.	0	R_W
13	TSIPV4ENA: Enable Time Stamp Snapshot for IPv4 frames When set, the time stamp snapshot is taken for IPv4 frames.	1	R_W
12	TSIPV6ENA: Enable Time Stamp Snapshot for IPv6 frames When set, the time stamp snapshot is taken for IPv6 frames.	0	R_W

**Table 5-43 Register 448 (Time Stamp Control Register) (Continued)**

Field	Description	Reset	Access
11	TSIPENA: Enable Time Stamp Snapshot for PTP over Ethernet frames When set, the time stamp snapshot is taken for frames which have PTP messages in Ethernet frames (PTP over Ethernet) also. By default snapshots are taken for UDP-IP-Ethernet PTP packets.	0	R_W
10	TSVER2ENA: Enable PTP packet snooping for version 2 format When set, the PTP packets are snooped using the 1588 version 2 format else snooped using the version 1 format (IEEE 1588 Version 1 and Version 2 format are described in “ <a href="#">PTP Processing and Control</a> ” on page 44).	0	R_W
9	TSCTRLSSR: Time Stamp Digital or Binary rollover control When set, the Time Stamp Low register rolls over after 0x3B9A_C9FF value (i.e., 1 nanosecond accuracy) and increments the Time Stamp (High) seconds. When reset, the rollover value of sub-second register is 0x7FFF_FFFF. The sub-second increment has to be programmed correctly depending on the PTP reference clock frequency and this bit value.	0	R_W
8	TSENALL: Enable Time Stamp for All Frames When set, the time stamp snapshot is enabled for all frames received by the core.	0	R_W
7:6	Reserved		
5	TSADDREG: Addend Reg Update When set, the contents of the Time Stamp Addend register is updated in the PTP block for fine correction. This is cleared when the update is completed. This register bit should be zero before setting it. This is a reserved bit when only coarse correction option is selected.	0	R_WSC
4	TSTRIG: Time Stamp Interrupt Trigger Enable When set, the Time Stamp interrupt is generated when the System Time becomes greater than the value written in Target Time register. This bit is reset after the generation of Time Stamp Trigger Interrupt.	0	R_WSC
3	TSUPDT: Time Stamp Update When set, the system time is updated (added/subtracted) with the value specified in the Time Stamp High Update and Time Stamp Low Update registers. This register bit should be read zero before updating it. This bit is reset once the update is completed in hardware. The “Time Stamp Higher Word” register (if enabled using coreKit configuration) is not updated.	0	R_WSC
2	TSINIT: Time Stamp Initialize When set, the system time is initialized (over-written) with the value specified in the Time Stamp High Update and Time Stamp Low Update registers. This register bit should be read zero before updating it. This bit is reset once the initialize is complete. The “Time Stamp Higher Word” register (if enabled using coreKit configuration) can only be initialized.	0	R_WSC

**Table 5-43 Register 448 (Time Stamp Control Register) (Continued)**

Field	Description	Reset	Access
1	TSCFUPDT: Time Stamp Fine or Coarse Update When set, indicates that the system times update to be done using fine update method. When reset it indicates the system time stamp update to be done using Coarse method. This bit is reserved if the fine correction option is not enabled.	0	R_W
0	TSENA: Time Stamp Enable When this bit, is set the timestamping is enabled for transmit and receive frames. When disabled timestamp is not added for transmit and receive frames and the TimeStamp Generator is also suspended. User has to always initialize theTimeStamp (system time) after enabling this mode.	0	R_W

[Table 5-44](#) indicates the messages, for which a snapshot is taken depending on the clock, enable master and enable snapshot for event message register settings.

**Table 5-44 Time-Snapshot Dependency on Register Bits**

TSCLKTYPE	TSMSTRENA	TSEVNTEA	Messages for which snapshot is taken
00 or 01	X	0	SYNC, Follow_Up, Delay_Req, Delay_Resp
00 or 01	1	1	Delay_Req
00 or 01	0	1	SYNC
10	N/A	0	SYNC, Follow_Up, Delay_Req, Delay_Resp
10	N/A	1	SYNC, Follow_Up
11	N/A	0	SYNC, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp
11	N/A	1	SYNC, Pdelay_Req, Pdelay_Resp

### 5.2.3.2 Register 449 (Sub-Second Increment Register)

This register is present only when the IEEE 1588 time stamping feature is selected without an external time stamp input. In Coarse Update mode (TSCFUPDT bit in Register 448), the value in this register is added to the system time every clock cycle of clk\_ptp\_ref\_i. In Fine Update mode, the value in this register is added to the system time whenever the Accumulator gets an overflow.

**Table 5-45 Register 449 (Sub-Second Increment Register)**

Field	Description	Reset	Access
31:8	Reserved	0000000H	RO
7:0	SSINC: Sub-second increment value The value programmed in this register is accumulated with the contents of the sub-second register. For example, to achieve an accuracy of 20 ns, the value to be programmed is 20. (14H with a 50 MHz reference clock if 1 ns accuracy is selected.)	00H	R_W

### 5.2.3.3 Register 450 (System Time - Seconds Register)

The System Time - Seconds register, along with System Time - Nanoseconds register, indicates the current value of the system time maintained by the core. Though it is updated on a continuous basis, there is some delay from the actual time due to clock domain transfer latencies (from clk\_ptp\_ref\_i to clk\_csr\_i).

These registers (450 and 451) are present only when the IEEE 1588 Time Stamp feature is selected without external time-stamp input.

**Table 5-46 Register 450 (System Time - Seconds Register)**

Field	Description	Reset	Access
31:0	TSS: Time Stamp Second The value in this field indicates the current value in seconds of the System Time maintained by the core.	0000000H	RO

### 5.2.3.4 Register 451 (System Time - Nanoseconds Register)

**Table 5-47 Register 451 (System Time - Nanoseconds Register)**

Field	Description	Reset	Access
31	PSNT: Positive or Negative Time This bit indicates positive or negative time value. If the bit is reset, it indicates that the time representation is positive, and if it is set, it indicates negative time value. (This bit represents the 32nd bit of the nanoseconds value when the Advance Time Stamp feature is enabled).	0	RO
30:0	TSSS: Time Stamp Sub Seconds The value in this field has the sub second representation of time, with an accuracy of 0.46 nano-second. (When TSCTRLSSR is set, each bit represents 1 ns and the maximum value will be 0x3B9A_C9FF, after which it rolls-over to zero).	0000000H	RO

### 5.2.3.5 Register 452 (System Time - Seconds Update Register)

The System Time - Seconds Update register, along with the System Time - Nanoseconds Update register, initialize or update the system time maintained by the core. You must write both of these registers before setting the TSINIT or TSUPDT bits in the Time Stamp Control register. This register is present only when the IEEE 1588 Time Stamp feature is selected without external time-stamp input.

**Table 5-48 Register 452 (System Time - Seconds Update Register)**

Field	Description	Reset	Access
31:0	TSS: Time Stamp Second The value in this field indicates the time, in seconds, to be initialized or added to the system time.	0000000H	R_W

### 5.2.3.6 Register 453 (System Time - Nanoseconds Update Register)

This register is present only when IEEE 1588 time stamp feature is selected without external time stamp input.

**Table 5-49 Register 453 (System Time - Nanoseconds Update Register)**

Field	Description	Reset	Access
31	ADDSUB: Add or subtract time When this bit is set, the time value is subtracted with the contents of the update register. When this bit is reset, the time value is added with the contents of the update register.	0	R_W
30:0	TSSS: Time Stamp Sub Seconds The value in this field has the sub second representation of time, with an accuracy of 0.46 nano-second. (When TSCTRLSSR is set in the time stamp control register, each bit represents 1 ns and the programmed value should not exceed 0x3B9A_C9FF.)	0000000H	R_W

### 5.2.3.7 Register 454 (Time Stamp Addend Register)

This register is present only when the IEEE 1588 time stamp feature is selected without external time stamp input. This register value is used only when the system time is configured for Fine Update mode (TSCFUPDT bit in Register 448). This register content is added to a 32-bit accumulator in every clock cycle (of clk\_ptp\_ref\_i) and the system time is updated whenever the accumulator overflows.

**Table 5-50 Register 454 (Time Stamp Addend Register)**

Field	Description	Reset	Access
31:0	TSAR: Time Stamp Addend Register This register indicates the 32-bit time value to be added to the Accumulator register to achieve time synchronization.	0000000H	R_W

### 5.2.3.8 Register 455 (Target Time Seconds Register)

The Target Time Seconds register, along with Target Time Nanoseconds register, are used to schedule an interrupt event (TSTARTGT bit in register 458 when Advanced Timestamping is enabled, or otherwise, TS interrupt bit in Register14[9]) when the system time exceeds the value programmed in these registers.

This register is present only when the IEEE 1588 time stamp feature is selected without external time stamp input.

**Table 5-51 Register 455 (Target Time Seconds Register)**

Field	Description	Reset	Access
31:0	TSTR: Target Time Seconds Register This register stores the time in seconds. When the time stamp value matches or exceeds both Target Time Stamp registers, the MAC, if enabled, generates an interrupt.	0000000H	R_W

### 5.2.3.9 Register 456 (Target Time Nanoseconds Register)

This register is present only when the IEEE 1588 Time Stamp feature is selected without external time stamp input.

**Table 5-52 Register 456 (Target Time Nanoseconds Register)**

Field	Description	Reset	Access
31	Reserved	0	RO
30:0	TSTR: Target Time Stamp Low Register  This register stores the time in (signed) nanoseconds. When the value of the Time Stamp matches the Target time stamp registers (both), the MAC will generate an interrupt if enabled. (This value should not exceed 0x3B9A_C9FF when TSCTRLSSR is set in the time stamp control register.)	00000000H	R_W

### 5.2.3.10 Register 457 (System Time - Higher Word Seconds Register)

This register is present only when the IEEE 1588 Advanced Time Stamp feature is selected without an external timestamp input.

**Table 5-53 Register 457 (System Time - Higher Word Seconds Register)**

Field	Description	Reset	Access
31:16	Reserved	0H	RO
15:0	TSHWR: Time Stamp Higher Word Register  Contains the most significant 16-bits of the time stamp seconds value. This register is optional and can be selected using the coreKit parameter mentioned in chapter 6. The register is directly written to initialize the value. This register is incremented when there is an overflow from the 32-bits of the System Time - Seconds register.	0000H	R_W_SU

### 5.2.3.11 Register 458 (Time Stamp Status Register)

This register is present only when Advanced IEEE 1588 time stamp feature is selected. All bits except Bits[27:25] gets cleared after this register is read by the host.

**Table 5-54 Register 458 (Time Stamp Status Register)**

Field	Description	Reset	Access
31:28	Reserved	0	RO
27:25	ATSNS: Auxiliary Time Stamp Number of Snapshots  These bits indicate the number of Snapshots available in the FIFO. A value of 4 (100) indicates that the Auxiliary Snapshot FIFO is full. These bits are cleared (to 000) when the Auxiliary snapshot FIFO clear bit is set. This bit is valid only if the Auxiliary Snapshot coreKit parameter is enabled.	000	RO
24	ATSSTM: Auxiliary Time Stamp Snapshot Trigger Missed  This bit is set when the Auxiliary time stamp snapshot FIFO is full and external trigger was set. This indicates that the latest snapshot was not stored in the FIFO. This bit is valid only if the Auxiliary Snapshot coreKit parameter is enabled.	0	R_SS_RC

**Table 5-54 Register 458 (Time Stamp Status Register) (Continued)**

Field	Description	Reset	Access
23:3	Reserved	0	RO
2	Auxiliary Time Stamp Trigger Snapshot This bit is set high when the auxiliary snapshot is written to the FIFO. This bit is valid only if the Auxiliary Snapshot coreKit parameter is enabled	0	R_SS_RC
1	TSTARGET: Time Stamp Target Time Reached When set, indicates the value of system time is greater or equal to the value specified in the Target Time High and Low registers	0	R_SS_RC
0	TSSOVF: Time Stamp Seconds Overflow When set, indicates that the seconds value of the time stamp (when supporting version 2 format) has overflowed beyond 32'hFFFF_FFFF.	0	R_SS_RC

### 5.2.3.12 Register 459 (PPS Control Register)

This register is present only when Advanced Time Stamp feature is selected and External Time-stamping is not enabled.

**Table 5-55 Register 459 (PPS Control Register)**

Field	Description	Reset	Access
31:4	Reserved	0H	RO
3:0	PPSCTRL: Control the duration between 2-pulses of PPS signal output This controls the duration between 2 pulses of the PPS output signal. By default the control is set to 1 second between pulses. The duration of the pulses for valid settings is mentioned below. <ul style="list-style-type: none"><li>• 0000: 1.0 sec (Binary &amp; Digital Rollover)</li><li>• 0001: 0.5 sec (Binary Rollover), 0.536 sec (Digital Rollover)</li><li>• 0010: 0.25 sec (Binary Rollover), 0.26 sec (Digital Rollover)</li><li>• 0011: 0.125 sec (Binary Rollover), 0.13 sec (Digital Rollover)</li><li>....</li><li>• 1111: 15.28 µsec (Binary Rollover), 32.77 µsec (Digital Rollover)</li></ul>	0H	RW

### 5.2.3.13 Register 460 (Auxiliary Time Stamp - Nanoseconds Register)

This register along with Register 461 gives the 64-bit timestamp stored as auxiliary snapshot. The two registers together form the read port of a 4-deep 64-bit wide FIFO. Multiple snapshots can be stored in this FIFO and the fill-level of this FIFO is indicated by ATSNS bits in the Time Stamp Status register. The top of the FIFO is removed only when the last byte of Register 461 is read. In Little-endian mode, this means when Bits[31:24] is read while in Big-endian mode it corresponds to the reading of Bits[7:0] of Register 461.

Both these registers are present only when "Auxiliary Snapshot Enable" is selected during coreKit configuration..

**Table 5-56 Register 459 (PPS Control Register)**

Field	Description	Reset	Access
31:0	Contains the lower 32 bits (nano-seconds field) of the auxiliary time stamp	0H	RO

### 5.2.3.14 Register 461 (Auxiliary Time Stamp - Seconds Register)

**Table 5-57 Register 461 (Auxiliary Time Stamp - Seconds Register)**

Field	Description	Reset	Access
31:0	Contains the lower 32 bits of the Seconds field of the auxiliary time stamp	0H	RO

# 6

## Parameters

---

This chapter provides detailed descriptions for all configuration options and parameters. Options are grouped as follows:

- ❖ “Features” on page 310
  - ◆ “System Interface” on page 310
  - ◆ “General Features” on page 314
    - ✧ “Buffer Management” on page 314
    - ✧ “MAC Address” on page 316
    - ✧ “Miscellaneous” on page 316
  - ◆ “PHY Line Interfaces (RGMII, RMII, SMII, SGMII, TBI, and RTBI)” on page 318
- ❖ “Optional Modules” on page 320
  - ◆ “IEEE 1588 Time Stamp Block” on page 320
  - ◆ “Power Management Block” on page 321
  - ◆ “IP Checksum Block” on page 321
  - ◆ “GMAC Management (RMON) Counters” on page 322
    - ✧ “GMAC Management (RMON) Receive Counters” on page 322
    - ✧ “GMAC Management (RMON) Transmit Counters” on page 326
    - ✧ “GMAC Management (RMON) Receive IPC Counters” on page 330
  - ◆ “Station Management Block” on page 335

## 6.1 Features

### 6.1.1 System Interface

**Table 6-1 Top-Level Configuration**

Option	Definition
Top-Level Configuration	<p><b>Description:</b> Enables the core with FIFO, DMA, AHB or AXI interface.</p> <p>Enable one of the following options from the drop-down menu:</p> <ul style="list-style-type: none"> <li>• GMAC-AXI</li> <li>• GMAC-AHB</li> <li>• GMAC-CORE</li> <li>• GMAC-MTL</li> <li>• GMAC-DMA</li> </ul> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> GMAC-AHB</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b></p> <ul style="list-style-type: none"> <li>• GMAC_AXI_SUBSYS</li> <li>• GMAC_AHB_SUBSYS</li> <li>• GMAC_CORE</li> <li>• GMAC_MTL_SUBSYS</li> <li>• GMAC_DMA_SUBSYS</li> </ul>

**Table 6-2 System Data Path**

Option	Definition
Datawidth	<p><b>Description:</b> Configures the width of the data (32, 64 or 128 only)</p> <p><b>Value Range:</b> 32, 64, or 128</p> <p><b>Default Value:</b> 32</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> DATAWIDTH</p>
Endianness	<p><b>Description:</b> Configures the GMAC-UNIV to operate in either Little Endian or Big Endian mode, or to have an input pin (sbd_data_endianess_i) to configure either. Select one of the following options from the drop-down menu.</p> <ul style="list-style-type: none"> <li>• LITTLE_ENDIAN</li> <li>• BIG_ENDIAN</li> <li>• BOTH_OPTIONS</li> </ul> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> LITTLE_ENDIAN</p> <p><b>Dependencies:</b> Not enabled in GMAC-AXI configuration.</p> <p><b>HDL Parameter Name:</b> ENDIAN_NESSION</p>

**Table 6-2 System Data Path (Continued)**

Option	Definition
Descriptor Endianness	<p><b>Description:</b> Configures the format (endianness) of the DMA Descriptors with respect to the data bus endianness. The DMA Descriptor format is given in “<a href="#">Normal Descriptor Formats</a>” on page 337 with different options. Select one of the following options from the drop-down menu:</p> <ul style="list-style-type: none"> <li>• SAME_ENDIANESS</li> <li>• REVERSE_ENDIANESS</li> <li>• BOTH_OPTIONS</li> </ul> <p>Selecting SAME_ENDIANESS configures the DMA descriptor as little endian or big endian, as selected by the Endianness parameter above. Selecting the Reverse will configure the DMA Descriptors to the reverse of the endianness of the data bus. BOTH_OPTIONS results in an input pin (sbd_desc_endianess_i) for the core.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> BOTH_OPTIONS when Data Endianness = BOTH_OPTIONS, else SAME_ENDIANESS</p> <p><b>Dependencies:</b> This is enabled only in GMAC-AHB or GMAC-DMA configurations and when Data Endianness is not set to BOTH_OPTIONS</p> <p><b>HDL Parameter Name:</b> DESC_ENDIANESS</p>
Alternate Descriptor Format	<p><b>Description:</b> Selects the alternate descriptor format for the DMA. This enables the DMA to transfer data to and from buffers of size up to 8 KB. This descriptor format is given in “<a href="#">Alternate or Enhanced Descriptors</a>” on page 356</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> This is enabled only in GMAC-AXI, GMAC-AHB or GMAC-DMA configurations.</p> <p><b>HDL Parameter Name:</b> DESC_ENHANCED_FORMAT</p>



The Alternate or Enhanced Descriptor format is not supported with Normal Time Stamp mode. Either the Advanced Time Stamp mode has to be enabled, or the Enhanced Descriptor mode needs to be de-selected.

**Table 6-3 CSR Port**

Option	Definition
CSR Port	<p><b>Description:</b> Enables the CSR port to have the MCI or APB or AHB or AXI Slave interface</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> AHB interface in GMAC-AHB configuration, AXI interface in GMAC-AXI configuration, else MCI interface</p> <p><b>Dependencies:</b> MCI interface cannot be selected in GMAC-AHB or GMAC-AXI configuration, and AHB interface cannot be selected in non-GMAC-AHB configuration, AXI interface cannot be selected in non-GMAC-AXI configuration.</p> <p><b>HDL Parameter Name:</b> CSR_PORT</p>

**Table 6-3 CSR Port (Continued)**

Option	Definition
Clock	<p><b>Description:</b> Enables a separate clock port instead of the default application clock for the CSR interface.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only if the top-level configuration is anything other than GMAC_CORE or GMAC_AXI with AXI Slave interface.</p> <p><b>HDL Parameter Name:</b> CSR_SLV_CLK</p>
CSR Port Datawidth	<p><b>Description:</b> Configures the width of the data (32, 64 or 128 only)</p> <p><b>Value Range:</b> 32, 64, or 128</p> <p><b>Default Value:</b> 32</p> <p><b>Dependencies:</b> Parameter selection is enabled only when the CSR port is configured as an AHB slave interface.</p> <p><b>HDL Parameter Name:</b> CSR_DATAWIDTH</p>

**Table 6-4 Sideband Input Signals**

Option	Definition
Rx Flow Control	<p><b>Description:</b> Adds the sideband sbd_flowctrl_i as input port for triggering flow control (back pressure or PAUSE frames).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled when the top-level configuration is not GMAC-CORE</p> <p><b>HDL Parameter Name:</b> SBD_FC_EN</p>
MAC Transmitter /Receiver Disable	<p><b>Description:</b> Adds the sideband signals sbd_dis_transmit_i and sbd_dis_receive_i input ports for Disable control of MAC transmitter and receiver</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only in GMAC-CORE and GMAC-MTL configurations</p> <p><b>HDL Parameter Name:</b> ADD_TXRX_DIS_IO</p>
Transmit FIFO Flush	<p><b>Description:</b> Adds the sideband signal ati_txfifoflush_i input port for MTL Tx FIFO Flush control</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only in GMAC-MTL configurations</p> <p><b>HDL Parameter Name:</b> ADD_TXFIFO_FLUSH_IO</p>

**Table 6-5 AXI Interface**

Option	Definition
AXI Maximum Burst Length	<p><b>Description:</b> Selects the maximum burst length allowed on the AXI bus.</p> <p><b>Value Range:</b> 16, 32, 64, 128, or 256</p> <p><b>Default Value:</b> 16</p> <p><b>Dependencies:</b> Parameter selection is only enabled for GMAC-AXI configuration.</p> <p><b>HDL Parameter Name:</b> AXI_BL</p>
AXI Master ID Width	<p><b>Description:</b> Selects the AXI Master bus ID width.</p> <p><b>Value Range:</b> 1-12</p> <p><b>Default Value:</b> 4</p> <p><b>Dependencies:</b> Parameter selection is only enabled for GMAC-AXI configuration.</p> <p><b>HDL Parameter Name:</b> AXI_GM_AXI_ID_WIDTH</p>
AXI Slave ID Width	<p><b>Description:</b> Selects the AXI Slave bus ID width.</p> <p><b>Value Range:</b> 2-16</p> <p><b>Default Value:</b> 8</p> <p><b>Dependencies:</b> Parameter selection is only enabled for GMAC-AXI configuration.</p> <p><b>HDL Parameter Name:</b> GS_ID</p>
AXI Maximum Write Requests	<p><b>Description:</b> Maximum number of outstanding Write Requests on AXI interface.</p> <p><b>Value Range:</b> 4, 8</p> <p><b>Default Value:</b> 4</p> <p><b>Dependencies:</b> Parameter selection is only enabled for GMAC-AXI configuration</p> <p><b>HDL Parameter Name:</b> AXI_GM_MAX_WR_REQUESTS</p>
AXI Maximum Read Requests	<p><b>Description:</b> Maximum number of outstanding Read Requests on AXI interface.</p> <p><b>Value Range:</b> 4, 8</p> <p><b>Default Value:</b> 4</p> <p><b>Dependencies:</b> Parameter selection is only enabled for GMAC-AXI configuration</p> <p><b>HDL Parameter Name:</b> AXI_GM_MAX_RD_REQUESTS</p>

## 6.1.2 General Features

**Table 6-6 General Features**

Option	Definition
Operating Mode	<p><b>Description:</b> Configures the core to work in 10/100/1000-Mbps mode. Select 10/100/1000 Mbps for enabling both Fast Ethernet as well as Gigabit operations, 10/100 Mbps for Fast Ethernet-only operations, and 1000 Mbps for Gigabit-only operations.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> 10/100/1000 Mbps with Gigabit License, 10/100 with Fast Ethernet license</p> <p><b>Dependencies:</b> This is not configurable with Fast Ethernet License</p> <p><b>HDL Parameter Name:</b> OP_MODE</p>
User Defined Version	<p><b>Description:</b> Configurable 8-bit value that is hard-coded into Bits[15:8] of the GMAC Version Register (for example, 8'h10 for version 1.0).</p> <p><b>Value Range:</b> 0x00 to 0xFF</p> <p><b>Default Value:</b> 0x10</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> USER_VER</p>
Half-/Full-Duplex	<p><b>Description:</b> Configures the core to work in Full-Duplex mode only.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> FDUPLEX_ONLY</p>
Reset Mode	<p><b>Description:</b> Enables the asynchronous reset for all flip-flops. When deselected, all register flip-flops have synchronous resets.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Enabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> ASYNC_RST</p>

### 6.1.2.1 Buffer Management

**Table 6-7 Buffer Management**

Option	Definition
Rx FIFO Size	<p><b>Description:</b> Selects the Rx FIFO size.</p> <p><b>Value Range:</b> 128 bytes to 32 KB in powers of 2</p> <p><b>Default Value:</b> 2 KB</p> <p><b>Dependencies:</b> Parameter selection is enabled only if top-level configuration is anything other than GMAC-CORE.</p> <p><b>HDL Parameter Name:</b> RXFIFO_SIZE</p>

**Table 6-7 Buffer Management (Continued)**

Option	Definition
Tx FIFO Size	<p><b>Description:</b> Selects the Tx FIFO size.</p> <p><b>Value Range:</b> 256 bytes to 16 KB in powers of 2</p> <p><b>Default Value:</b> 2 KB</p> <p><b>Dependencies:</b> The parameter does not apply if the top-level configuration is GMAC-CORE. 256- and 512-byte values can only be selected in 10/100-only mode, or if the core is configured for Full-Duplex-only mode.</p> <p><b>HDL Parameter Name:</b> TXFIFO_SIZE</p>
Maximum Number of Frames in the Tx FIFO	<p><b>Description:</b> Selects the maximum number of frames that can be pushed into the Tx FIFO at a time without any transmit status being read out on the ATI interface.</p> <p><b>Value Range:</b> 2, 4, or 8</p> <p><b>Default Value:</b> 2</p> <p><b>Dependencies:</b> Parameter selection is only enabled in GMAC-MTL configuration.</p> <p><b>HDL Parameter Name:</b> MAX_FRAME_IN_TXFIFO</p>
DPRAM TYPE	<p><b>Description:</b> Selects the use of synchronous or asynchronous DPRAM type.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only if top-level configuration is anything other than GMAC-CORE.</p> <p><b>HDL Parameter Name:</b> ASYNC_RAM</p>
Min Size of Rx Frames	<p><b>Description:</b> Configures the minimum size of frame that is being stored in the MTL Rx FIFO. This will configure the width of the counter used for storing the number of frames present in Rx FIFO</p> <p><b>Value Range:</b> 16, 32, 64</p> <p><b>Default Value:</b> 16</p> <p><b>Dependencies:</b> Parameter selection is enabled only when the top-level configuration is GMAC-MTL and the RX_FRAME_LEN_IF is enabled.</p> <p><b>HDL Parameter Name:</b> MIN_FRAME_SIZE</p>
Rx Frame Length FIFO	<p><b>Description:</b> Enables the Frame Length FIFO-related logic in the MTL Receive path.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only when the top-level configuration is GMAC-MTL.</p> <p><b>Parameter Name:</b> RX_FRAME_LEN_IF</p>
Rx Frame Length FIFO Width	<p><b>Description:</b> Configures the width of MTL Rx Frame Length FIFO.</p> <p><b>Value Range:</b> 12 to 15</p> <p><b>Default Value:</b> 15</p> <p><b>Dependencies:</b> Parameter selection is enabled only when the top-level configuration is GMAC-MTL and the RX_FRAME_LEN_IF is enabled.</p> <p><b>HDL Parameter Name:</b> FRAME_LEN_FIFO_WIDTH</p>

### 6.1.2.2 MAC Address

**Table 6-8 MAC Address Registers**

Option	Definition
Enable all MAC Address Registers	<p><b>Description:</b> Enables all MAC Address registers (1 to 31) for the Address Filter block.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> ALL_MAC_ADDR_EN</p>
Enable MAC Address Register 1	<p><b>Description:</b> Enable MAC Address register 1 for the Address Filter block.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only if All MAC Addresses is disabled.</p> <p><b>HDL Parameter Name:</b> MAC_ADDR1</p> <p><b>Note:</b> MAC Address registers 2 through 31 can be enabled in the same manner.</p>
Enable MAC Address Register <i>N</i>	<p><b>Description:</b> Enable MAC Address registers 2–30 for the Address filter block.</p>
Enable MAC Address Register 31	<p><b>Description:</b> Enables MAC Address register 31 for the Address Filter block.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only if All MAC Addresses is disabled.</p> <p><b>HDL Parameter Name:</b> MAC_ADDR31</p>

### 6.1.2.3 Miscellaneous

**Table 6-9 Destination Address Filter**

Option	Definition
Disable address filter Hash Table	<p><b>Description:</b> Removes the Hash Table registers and related address filter logic.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> HASH_DIS</p>

**Table 6-10 Synchronization**

Option	Definition
Sync CSR MAC Address to Tx/Rx clock domain	<p><b>Description:</b> Enable the synchronization of MAC address registers to the transmit/receive clock domain. Otherwise, the CSR MAC Address Register value written with the application clock is directly used in the transmit/receive clock domains.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> SYNC_MAC_ADDR</p>
Sync Pause Time to Tx clock domain	<p><b>Description:</b> Enable the synchronization of the Pause Timer register to the transmit clock domain.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> SYNC_PAUSETIME</p>
Sync VLAN Tag to Rx clock domain	<p><b>Description:</b> Enables the synchronization of the VLAN Tag register to the receive clock domain.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> SYNC_VLANTAG</p>
Sync Hash table to Rx clock domain	<p><b>Description:</b> Enable the synchronization of the Hash Table register to the receive clock domain.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> SYNC_HASHTABLE</p>

### 6.1.3 PHY Line Interfaces (RGMII, RMII, SMII, SGMII, TBI, and RTBI)

The GMII/MII interface is the default selection, and is always available. The interfaces you select from [Table 6-11](#) are added to the core as depicted in [Figure 3-96](#).

**Table 6-11 Line Interface**

Option	Definition
Reduced GMI Interface (RGMII)	<p><b>Description:</b> Enables the Reduced GMI Interface (RGMII).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Not enabled in 10/100 Mbps-only mode</p> <p><b>HDL Parameter Name:</b> RGMII_EN</p>
Reduced MI Interface (RMII)	<p><b>Description:</b> Enables the Reduced MI Interface (RMII).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> This interface is only applicable for the 10/100-Mbps mode only and will not be available in 1000 Mbps-only mode</p> <p><b>HDL Parameter Name:</b> HDL Parameter name: RMII_EN</p>
Serial MI Interface (SMII)	<p><b>Description:</b> Enables the Serial MI Interface (SMII).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> This interface is only applicable for the 10/100-Mbps and 10/100/1000-Mbps modes only, and will not be available in 1000 Mbps-only mode.</p> <p><b>HDL Parameter Name:</b> HDL Parameter name: SMII_EN</p>
Serial GMI Interface (SGMII)	<p><b>Description:</b> Enables the Serial GMI Interface (SGMII).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Not enabled in 10/100 Mbps-only mode</p> <p><b>HDL Parameter Name:</b> HDL Parameter name: SGMII_EN</p>
Ten Bit Interface (TBI)	<p><b>Description:</b> Enables the Ten Bit Interface (TBI).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Not enabled in 10/100 Mbps-only mode</p> <p><b>HDL Parameter Name:</b> HDL Parameter name: TBI_EN</p>
Reduced Ten Bit Interface (RTBI)	<p><b>Description:</b> Enables the Reduced Ten Bit Interface (TBI).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Not enabled in 10/100 Mbps-only mode</p> <p><b>HDL Parameter Name:</b> HDL Parameter name: RTBI_EN</p>

**Table 6-12 Miscellaneous Features for RGMII/RTBI Line Interface**

Option	Definition
Dual edge flip-flop	<p><b>Description:</b> Selects the dual edge flip-flop from Xilinx Virtex 2V FPGA Library for the RGMII or RTBI only (for synthesis).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only when RGMII or RTBI interface is selected.</p> <p><b>HDL Parameter Name:</b> XILINX</p>

**Table 6-13 Miscellaneous Features for RGMII/SGMII/RMII/SMII Line Interface**

Option	Definition
Add output port to indicate speed (10, 100, or 1000 Mbps)	<p><b>Description:</b> Additional output port for speed selection (10, 100, or 1000 Mbps) for RGMII, SGMII, or RMII.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only when RGMII, SGMII, or RMII interface is selected.</p> <p><b>HDL Parameter Name:</b> SPEED_SELECT</p>
Source Synchronous SMII (SSSMII)	<p><b>Description:</b> Enables the Source Synchronous Serial MI Interface (SSSMII).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> This option is only applicable when SMII_EN is enabled</p> <p><b>HDL Parameter Name:</b> SSSMII_EN</p>
TXSYNC as Input in SSSMII	<p><b>Description:</b> Converts the TXSYNC as input signal in SSSMII.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> This option is only applicable when SSSMII_EN is enabled</p> <p><b>HDL Parameter Name:</b> SSSMII_TXSYNC_IN</p>

**Table 6-14 Option for Single PHY Interface**

Option	Definition
Enable single PHY interface	<p><b>Description:</b> A single PHY interface is selected without any MUX logic at the input or output. When this option is enabled, the phy_intf_sel_i input port is removed.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only when one of the PHY interfaces or GMII interface is selected</p> <p><b>HDL Parameter Name:</b> SINGLE_PHY_INTF</p>

## 6.2 Optional Modules

### 6.2.1 IEEE 1588 Time Stamp Block

Parameters that have been updated to support Advanced IEEE 1588 features are indicated by italics in [Table 6-15](#).

**Table 6-15 IEEE 1588 Time Stamp Block Parameters**

Option	Definition
IEEE1588 Time Stamping	<p><b>Description:</b> Adds logic for supporting IEEE1588 time stamping protocol. Time stamp is captured for all the receive frames. Time stamp is also captured for transmit frames marked as PTP frames.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> TIME_STAMPING</p>
IEEE1588 External Time Stamp Input Enable	<p><b>Description:</b> This removes the time stamp updating logic in the core, and the 64-bit time stamp value which is assigned through external input is directly provided along with the status.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Enabled when IEEE1588 Time Stamping is enabled.</p> <p><b>HDL Parameter Name:</b> EXT_TIME_STAMPING</p>
IEEE 1588 Advanced Time Stamp Support	<p><b>Description:</b> This enables the advanced time stamp features like snooping PTP packets along with support for IEEE 1588-2008 specifications.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Enabled when IEEE1588 Time Stamping is enabled.</p> <p><b>HDL Parameter Name:</b> ADV_TIME_STAMPING</p> <p><b>Note:</b> When Advanced Time Stamp Support is selected, Enable Full Receive Checksum Offload and Alternate (Enhanced) Descriptor are enabled, by default.</p>
IEEE1588 Higher Word Register Enable	<p><b>Description:</b> This adds a register to store the most significant 16 bits [47:32] of the seconds value if version 2 format of time stamp is used.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Enabled when IEEE1588 Advanced Time Stamping is enabled.</p> <p><b>HDL Parameter Name:</b> ADV_TIME_HIGH_WORD</p>
IEEE 1588 Auxiliary Snapshot Enable	<p><b>Description:</b> This adds the capability of storing the snapshot in a FIFO based on an external trigger.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Enabled when IEEE1588 Advanced Time Stamping is enabled.</p> <p><b>HDL Parameter Name:</b> ADV_TIME_AUX_SNAP</p>

## 6.2.2 Power Management Block

**Table 6-16 Power Management Block**

Option	Definition
Enable Power Management block (PMT)	<p><b>Description:</b> Enables the Power Management block in the core.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> PMT_EN</p>
Enable detection of remote wake-up frame	<p><b>Description:</b> Enables the detection of a remote wake-up frame.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only if PMT is enabled.</p> <p><b>HDL Parameter Name:</b> PMT_RWK_EN</p>
Enable detection of magic packet frame	<p><b>Description:</b> Enables the detection of a magic packet frame.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only if PMT is enabled.</p> <p><b>HDL Parameter Name:</b> PMT_MGK_EN</p>

## 6.2.3 IP Checksum Block

**Table 6-17 MDC Features Configuration Options**

Option	Definition
Enable IP Checksum for received frames	<p><b>Description:</b> Enables the appending of the IP checksum (Type 1) for received frames.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> IPCHKSUM_EN</p>
Enable Full Checksum Offload	<p><b>Description:</b> Enables the enhanced Checksum Offload engine (Type 2) in the receive path. In this mode, the Checksum Offload engine checks and detects IPv4 header checksum errors and TCP, UDP, or ICMP checksum errors encapsulated in IPv4 or IPv6 datagrams payload of the received frames.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> This option is only enabled when IP Checksum (IPCHKSUM_EN above) is selected.</p> <p><b>HDL Parameter Name:</b> IPC_FULL_OFFLOAD</p>

**Table 6-17 MDC Features Configuration Options (Continued)**

Option	Definition
Enable Transmit COE	<p><b>Description:</b> Enabled the Checksum Offload Engine in the transmit path. In this mode, the core can calculate and insert the checksums of TCP, UDP, or ICMP segments encapsulated in IPv4 or IPv6 datagrams, and can insert IPv4 header checksums as well in the transmitted frames.</p> <p><b>Value range:</b> N/A</p> <p><b>Default value:</b> Disabled</p> <p><b>Dependencies:</b> Applicable only in non-GMAC-CORE configuration</p> <p><b>HDL parameter Name:</b> TX_COE</p>

## 6.2.4 GMAC Management (RMON) Counters

**Table 6-18 RMON Counters**

Option	Definition
Enable GMAC Management (RMON) Counters	<p><b>Description:</b> Enables the GMAC Management (RMON) counters.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> MMC_EN</p>
Enable GMAC Management (RMON) IPC Counters	<p><b>Description:</b> Enables the GMAC Management (RMON) counters for the Checksum Offload module.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Enabled only when RMON counters (above) and Full Checksum Offload engine (Type 2) are selected.</p> <p><b>HDL Parameter Name:</b> MMC_IPC_EN</p>

### 6.2.4.1 GMAC Management (RMON) Receive Counters

**Table 6-19 RMON Receive Counters**

Option	Definition
Enable all Rx frame counters	<p><b>Description:</b> Enables all of the Rx frame counters.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only if GMAC Management (RMON) Counters are enabled.</p> <p><b>HDL Parameter Name:</b> MMC_EN_RX_ALL</p>

**Table 6-19 RMON Receive Counters (Continued)**

Option	Definition
Rx frame counter for good and bad frames	<p><b>Description:</b> Enables Rx frame counter for good or bad frames received.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXADDR_RNG_CNT_EN</p>
Rx counter for octets in good frames	<p><b>Description:</b> Enables the Rx counter for octets in good frames received.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXOCTG_CNT_EN</p>
Rx counter for octets in good and bad frames	<p><b>Description:</b> Enables the Rx counter for octets in good and bad frames received</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXOCTGB_CNT_EN</p>
Rx frame counter for good broadcast frames	<p><b>Description:</b> Enables the Rx frame counter for good broadcast frames received.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXBCASTG_CNT_EN</p>
Rx frame counter for good multicast frames	<p><b>Description:</b> Enables the Rx frame counter for good multicast frames received.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXMCASTG_CNT_EN</p>
Rx frame counter for frames with CRC errors	<p><b>Description:</b> Enables the Rx frame counter for frames received with CRC errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXCRCERR_CNT_EN</p>

**Table 6-19 RMON Receive Counters (Continued)**

Option	Definition
Rx frame counter for frames with Alignment errors	<p><b>Description:</b> Enables the Rx frame counter for frames received with Alignment errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled. It is not enabled for 1000 Mbps-only configuration</p> <p><b>HDL Parameter Name:</b> MMC_RXALGNERR_CNT_EN</p>
Rx frame counter for frames with Runt errors	<p><b>Description:</b> Enables the Rx frame counter for frames received with runt errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXRUNTERR_CNT_EN</p>
Rx frame counter for frames with Jabber error	<p><b>Description:</b> Enables the Rx frame counter for frames received with Jabber error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXJABERR_CNT_EN</p>
Rx frame counter for undersized good frames	<p><b>Description:</b> Enables the Rx frame counter for undersized good frames received.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXUNDERSZG_CNT_EN</p>
Rx frame counter for giant frames	<p><b>Description:</b> Enables the Rx frame counter for giant frames</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXOVERSZG_CNT_EN</p>
Rx Octet counters for various frame sizes	<p><b>Description:</b> Enables the octet counters for received frames. Separate octet counters for (good and bad) received frames sized equal to 64, 65–127, 128–255, 256–511, 512–1,023 and 1024_to_max bytes are provided.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXADDR RNG_CNT_EN</p>

**Table 6-19 RMON Receive Counters (Continued)**

Option	Definition
Rx frame counter for good unicast frames	<p><b>Description:</b> Enables the Rx frame counter for good unicast frames received.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXUCASTG_CNT_EN</p>
Rx frame counter for frames with Length error	<p><b>Description:</b> Enables the Rx frame counter for frames received with length errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXLENERR_CNT_EN</p>
Rx frame counter for out-of-range length frames	<p><b>Description:</b> Enables the Rx frame counter for out-of-range length frames (with length field values greater than 1500 and less than 1536) received.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXOUTOFRNG_TYP_CNT_EN</p>
Rx frame counter for pause frames	<p><b>Description:</b> Enables the Rx frame counter for pause frames received.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXPAUSE_CNT_EN</p>
Rx frame counter for number of FIFO overflow conditions	<p><b>Description:</b> Enables the Rx frame counter for the number of FIFO overflow conditions.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled and when the top-level configuration is anything other than GMAC-CORE.</p> <p><b>HDL Parameter Name:</b> MMC_RXFIFOVFL_CNT_EN</p>
Rx frame counter for VLAN good and bad frames	<p><b>Description:</b> Enables the Rx frame counter for VLAN good and bad frames received.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXVLANGB_CNT_EN</p>

**Table 6-19 RMON Receive Counters (Continued)**

Option	Definition
Rx frame counter for frames with watchdog error	<p><b>Description:</b> Enables the Rx frame counter for frames received with watchdog errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and Enable all Rx frame counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RXWDGERR_CNT_EN</p>

#### 6.2.4.2 GMAC Management (RMON) Transmit Counters

**Table 6-20 RMON Transmit Counters**

Option	Definition
Enable all Tx frame counters	<p><b>Description:</b> Enables all of the Tx frame counters.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only if GMAC Management (RMON) counters are enabled.</p> <p><b>HDL Parameter Name:</b> MMC_EN_TX_ALL</p>
Tx octet counters for various frame sizes	<p><b>Description:</b> Enables the octet counters for transmitted frames. Separate octet counters for (good and bad) frames sized equal to 64, 65-127, 128-255, 256-511, 512-1,023 and 1024_to_max bytes are provided.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXADDR_RNG_CNT_EN</p>
Tx octet counter for good and bad frames	<p><b>Description:</b> Enables the Tx octet counter for good and bad frames transmitted.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXOCTGB_CNT_EN</p>
Tx frame counter for good and bad frames	<p><b>Description:</b> Enables the Tx frame counter for good and bad frames transmitted.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXFRMGB_CNT_EN</p>

**Table 6-20 RMON Transmit Counters (Continued)**

Option	Definition
Tx frame counter for good broadcast frames	<p><b>Description:</b> Enables the Tx frame counter for good broadcast frames.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXBCASTG_CNT_EN</p>
Tx frame counter for good multicast frames	<p><b>Description:</b> Enables the Tx frame counter for good multicast frames transmitted.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXMCASTG_CNT_EN</p>
Tx frame counter for good and bad unicast frames	<p><b>Description:</b> Enables the Tx frame counter for good and bad unicast frames.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXUCASTGB_CNT_EN</p>
Tx frame counter for good and bad multicast frames	<p><b>Description:</b> Enables the Tx frame counter for good and bad multicast frames transmitted.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXMCASTGB_CNT_EN</p>
Tx frame counter for good and bad broadcast frames	<p><b>Description:</b> Enables the Tx frame counter for good and bad broadcast frames transmitted.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXBCASTGB_CNT_EN</p>
Tx frame counter for frames with underflow errors	<p><b>Description:</b> Enables the Tx frame counter for frames transmitted with underflow errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXUNRFLW_CNT_EN</p>

**Table 6-20 RMON Transmit Counters (Continued)**

Option	Definition
Tx frame counter for good frames with single collision	<p><b>Description:</b> Enables the Tx frame counter for good frames transmitted with single collisions.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXSNGLCOLG_CNT_EN</p>
Tx frame counter for good frames with multiple collision	<p><b>Description:</b> Enables the Tx frame counter for good frames transmitted with multiple collisions.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXMULTCOLG_CNT_EN</p>
Tx frame counter for frames deferred before transmit	<p><b>Description:</b> Enables the Tx frame counter for frames that were deferred before being transmitted.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXDEFRD_CNT_EN</p>
Tx frame counter for frames with late collision	<p><b>Description:</b> Enables the Tx frame counter for frames aborted due to late collisions.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXLATECOL_CNT_EN</p>
Tx frame counter for frames aborted after excessive collision	<p><b>Description:</b> Enables the Tx frame counter for frames aborted after excessive collisions.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXEXSCOL_CNT_EN</p>
Tx frame counter for frames with carrier error	<p><b>Description:</b> Enables the Tx frame counter for frames aborted with carrier errors (loss of carrier or no carrier).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled, “Enable all Tx frame counters” is disabled, and Half-Duplex mode is enabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXCARR_CNT_EN</p>

**Table 6-20 RMON Transmit Counters (Continued)**

Option	Definition
Tx octet counter for good frames	<p><b>Description:</b> Enables the Tx octet counter for good frames.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXOCTG_CNT_EN</p>
Tx frame counter for good frames	<p><b>Description:</b> Enables the Tx frame counter for good frames transmitted.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXFRMG_CNT_EN</p>
Tx frame counter for frames aborted due to excessive deferral	<p><b>Description:</b> Enables the Tx frame counter for frames that were aborted due to excessive deferral.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXEXSDEF_CNT_EN</p>
Tx frame counter for pause frames	<p><b>Description:</b> Enables the Tx frame counter for pause frames transmitted.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXPAUSE_CNT_EN</p>
Tx frame counter for VLAN frames	<p><b>Description:</b> Enables the Tx frame counter for VLAN frames transmitted.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters are enabled and “Enable all Tx frame counters” is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_TXVLAN_CNT_EN</p>

### 6.2.4.3 GMAC Management (RMON) Receive IPC Counters

**Table 6-21 GMAC Management (RMON) Receive IPC Counters**

Option	Definition
Enable all Rx IPC Counters	<p><b>Description:</b> Enables all Rx IPC counters.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled only if GMAC Management (RMON) counters and RMON IPC counters are enabled and Full Checksum Offload Engine (Type 2) is selected</p> <p><b>HDL Parameter Name:</b> MMC_IPC_EN_RX_ALL</p>
Enable counter for Rx IPv4 Good frame	<p><b>Description:</b> Enables Rx frame counter for good IPv4 datagrams received without any header or checksum errors and having TCP, UDP, or ICMP payload.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC counters, and Full Checksum Offload Engine (Type 2) are enabled and Enable all Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV4_GD_FRMS_CNT_EN</p>
Enable counter for Rx IPv4 datagrams with header error	<p><b>Description:</b> Enables Rx frame counter for all IPv4 datagrams received with header errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, and Full Checksum Offload Engine (Type 2) are enabled and Enable all Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV4_HDRERR_FRMS_CNT_EN</p>
Enable counter for Rx IPv4 datagrams not having TCP/UDP/ ICMP	<p><b>Description:</b> Enables Rx frame counter for all IPv4 datagrams received with payload other than TCP, UDP, or ICMP.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV4_NOPAY_FRMS_CNT_EN</p>
Enable counter for Rx IPv4 datagrams having fragmentation	<p><b>Description:</b> Enables Rx frame counter for all IPv4 datagrams received with fragments.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, and Full Checksum Offload Engine (Type 2) are enabled and Enable all Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV4_FRAG_FRMS_CNT_EN</p>

**Table 6-21 GMAC Management (RMON) Receive IPC Counters (Continued)**

Option	Definition
Enable counter for Rx UDP over IPv4 datagrams having UDP checksum disabled	<p><b>Description:</b> Enables Rx frame counter for all IPv4 datagrams received that have checksum-disabled UDP segments.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, and Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV4_UDSBL_FRMS_CNT_EN</p>
Enable counter for Rx IPv6 Good frame	<p><b>Description:</b> Enables Rx frame counter for good IPv6 datagrams that encapsulate TCP, UDP, or ICMP data and are received without header or checksum errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, and Full Checksum Offload Engine (Type 2) are enabled and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV6_GD_FRMS_CNT_EN</p>
Enable counter for Rx IPv6 datagrams with header error	<p><b>Description:</b> Enables an Rx frame counter for all IPv6 datagrams received with header errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, and Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV6_HDRERR_FRMS_CNT_EN</p>
Enable counter for Rx IPv6 datagrams not having TCP/UDP/ ICMP	<p><b>Description:</b> Enables an Rx frame counter for all IPv6 datagrams received with payload other than TCP, UDP, or ICMP.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV6_NOPAY_FRMS_CNT_EN</p>
Enable counter for Rx UDP segments with good checksum	<p><b>Description:</b> Enables an Rx frame counter for all UDP segments received and with no checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_UDP_GD_FRMS_CNT_EN</p>

**Table 6-21 GMAC Management (RMON) Receive IPC Counters (Continued)**

Option	Definition
Enable counter for Rx UDP segments with checksum error	<p><b>Description:</b> Enables an Rx frame counter for all UDP segments received with a checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_UDP_ERR_FRMS_CNT_EN</p>
Enable counter for Rx TCP segments with good checksum	<p><b>Description:</b> Enables an Rx frame counter for all TCP segments received with no checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_TCP_GD_FRMS_CNT_EN</p>
Enable counter for Rx TCP segments with checksum error	<p><b>Description:</b> Enables an Rx frame counter for all TCP segments received with a checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_TCP_ERR_FRMS_CNT_EN</p>
Enable counter for Rx ICMP segments with good checksum	<p><b>Description:</b> Enables an Rx frame counter for all ICMP segments received with no checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_ICMP_GD_FRMS_CNT_EN</p>
Enable counter for Rx ICMP segments with checksum error	<p><b>Description:</b> Enables an Rx frame counter for all ICMP segments received with a checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_ICMP_ERR_FRMS_CNT_EN</p>

**Table 6-21 GMAC Management (RMON) Receive IPC Counters (Continued)**

Option	Definition
Enable byte counter for Rx IPv4 Good frame	<p><b>Description:</b> Enables an Rx octet counter for good IPv4 datagrams that encapsulate TCP, UDP or ICMP data that are received with no errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV4_GD_OCTET_CNT_EN</p>
Enable byte counter for Rx IPv4 datagrams with header error	<p><b>Description:</b> Enables an Rx octet counter for all IPv4 datagrams received with a header error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV4_HDRERR_OCTET_CNT_EN</p>
Enable byte counter for Rx IPv4 datagrams not having TCP/UDP/ ICMP	<p><b>Description:</b> Enables an Rx octet counter for all IPv4 datagrams received with a payload other than TCP, UDP, or ICMP.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV4_NOPAY_OCTET_CNT_EN</p>
Enable byte counter for Rx IPv4 datagrams having fragmentation	<p><b>Description:</b> Enables an Rx octet counter for all IPv4 datagrams received with fragments.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV4_FRAG_OCTET_CNT_EN</p>
Enable byte counter for Rx UDP over IPv4 datagrams having UDP checksum disabled	<p><b>Description:</b> Enables an Rx octet counter for all IPv4 datagrams with UDP segments received with checksum disabled.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV4_UDSBL_OCTET_CNT_EN</p>

**Table 6-21 GMAC Management (RMON) Receive IPC Counters (Continued)**

Option	Definition
Enable byte counter for Rx IPv6 Good frame	<p><b>Description:</b> Enables an Rx octet counter for good IPv6 datagrams that encapsulate TCP, UDP, or ICMP data that are received without any errors.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV6_GD_OCTET_CNT_EN</p>
Enable byte counter for Rx IPv6 datagrams with header error	<p><b>Description:</b> Enables an Rx octet counter for all IPv6 datagrams received with a header error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV6_HDRERR_OCTET_CNT_EN</p>
Enable byte counter for Rx IPv6 datagrams not having TCP/UDP/ ICMP	<p><b>Description:</b> Enables an Rx octet counter for all IPv6 datagrams received with a payload other than TCP, UDP, or ICMP.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_IPV6_NOPAY_OCTET_CNT_EN</p>
Enable byte counter for Rx UDP segments with good checksum	<p><b>Description:</b> Enables an Rx octet counter for all UDP segments received with no checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_UDP_GD_OCTET_CNT_EN</p>
Enable byte counter for Rx UDP segments with checksum error	<p><b>Description:</b> Enables an Rx octet counter for all UDP segments received with a checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_UDP_ERR_OCTET_CNT_EN</p>

**Table 6-21 GMAC Management (RMON) Receive IPC Counters (Continued)**

Option	Definition
Enable byte counter for Rx TCP segments with good checksum	<p><b>Description:</b> Enables an Rx octet counter for all TCP segments received with no checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_TCP_GD_OCTET_CNT_EN</p>
Enable byte counter for Rx TCP segments with checksum error	<p><b>Description:</b> Enables an Rx octet counter for all TCP segments received with checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_TCP_ERR_OCTET_CNT_EN</p>
Enable byte counter for Rx ICMP segments with good checksum	<p><b>Description:</b> Enables an Rx octet counter for all ICMP segments received and having no checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_ICMP_GD_OCTET_CNT_EN</p>
Enable byte counter for Rx ICMP segments with checksum error	<p><b>Description:</b> Enables an Rx octet counter for all ICMP segments received with checksum error.</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Disabled</p> <p><b>Dependencies:</b> Parameter selection is enabled if GMAC Management (RMON) Counters, RMON IPC Counters, Full Checksum Offload Engine (Type 2) are enabled, and Enable All Rx IPC Counters is disabled.</p> <p><b>HDL Parameter Name:</b> MMC_RX_ICMP_ERR_OCTET_CNT_EN</p>

## 6.2.5 Station Management Block

**Table 6-22 Station Management Block**

Option	Definition
Enable Station Management block (MDIO)	<p><b>Description:</b> Enables the Station Management block (MDIO Interface).</p> <p><b>Value Range:</b> N/A</p> <p><b>Default Value:</b> Enabled</p> <p><b>Dependencies:</b> None</p> <p><b>HDL Parameter Name:</b> SMA_EN</p>



## 7

# Descriptors

This chapter comprises the following subsections:

- ❖ “Normal Descriptor Formats”
- ❖ “Alternate or Enhanced Descriptors”

## 7.1 Normal Descriptor Formats

The DMA in the Ethernet subsystem transfers data based on a linked list of descriptors, as explained in “[DMA Controller](#)” on page [64](#). The default descriptor formats (common for both Receive and Transmit Descriptors) are shown in [Figures 7-1 to 7-10](#), and field descriptions are provided in [Sections 7.1.1–7.1.2](#).



- All descriptions in [Sections 7.1.1](#) and [Sections 7.1.2](#) correspond to the default descriptor format. The alternate enhanced descriptor format is discussed in “[Alternate or Enhanced Descriptors](#)” on page [356](#).
- Changes to the default descriptor format when IEEE1588 time stamping is enabled are described in “[Descriptor Format With IEEE 1588 Time Stamping Enabled](#)” on page [352](#)

Each descriptor contains two buffers, two byte-count buffers, and two address pointers, which enable the adapter port to be compatible with various types of memory management schemes.

The descriptor addresses must be aligned to the bus width used (Word/Dword/Lword for 32/64/128-bit buses). The descriptor can be configured for Same Endianness as the data bus or Reverse Endianness to the data bus (refer to “[System Interface](#)” on page [310](#)). The data bus can be configured for either little endian or big endian format.

[Figures 7-1](#) and [7-2](#) show the descriptor format in a 32-bit data bus.

**Figure 7-1 Rx/Tx Descriptors in Same-Endian Mode for 32-Bit, Little-Endian Format; Rx/Tx Descriptors in Reverse-Endian Mode for 32-Bit, Big-Endian Format Data Bus**

	31	23	15	7	0		
DES0	O W N	Status [30:0]					
DES1	Control Bits [9:0]		Byte Count Buffer2 [10:0]		Byte Count Buffer1[10:0]		
DES2	Buffer1 Address[31:0]						
DES3	Buffer2 Address [31:0] / Next Descriptor Address [31:0]						

**Figure 7-2 Rx/Tx Descriptors in Same-Endian Mode for 32-Bit, Big-Endian Format; Rx/Tx Descriptors in Reverse-Endian Mode for 32-Bit, Little-Endian Format Data Bus**

	31	23	15	7	0
DES0	Status [15:8]	Status [15:8]	Status [23:16]	O W N	Status [30:24]
DES1	BCB1 [7:0]	BCB1 [10:8]	BCB2 [4:0]	CB [1:0]	BCB2 [10:5] Control Bits [9:2]
DES2	Buffer1 Addr [7:0]	Buffer1 Addr [15:8]		Buffer1 Addr [23:16]	
DES3	Buffer2 Addr / Next Desc Addr [7:0]	Buffer2 Addr / Next Desc Addr [15:8]		Buffer2 Addr / Next Desc Addr [23:16]	
					Buffer2 Addr / Next Desc Addr [31:24]

[Figures 7-3](#) through [7-6](#) show the same descriptors with a 64-bit data bus.

**Figure 7-3 Rx/Tx Descriptors in Same-Endian Mode for 64-Bit, Little-Endian Format Data Bus**

	63	55	47	39	31	23	15	7	0
DES1-DES0	Control Bits [9:0]		Byte Count Buffer2 [10:0]		Byte Count Buffer1[10:0]		O W N	Status [30:0]	
DES3-DES2	Buffer2 Address [31:0] / Next Descriptor Address [31:0]					Buffer1 Address[31:0]			

**Figure 7-4 Rx/Tx Descriptors in Reverse-Endian Mode for 64-Bit, Little-Endian Format Data Bus**

	63	55	47	39	31	23	15	7	0	
DES1-DES0	BCB1 [7:0]	BCB1 [10:8] [10:8]	BCB2 [4:0] [4:0]	CB [1:0] [1:0]	BCB2 [10:5] [10:5]	Control Bits [9:2] [9:2]	Status [7:0] [7:0]	Status [15:8] [15:8]	Status [23:16] [23:16]	O W N [30:24] [30:24]
DES3-DES2	Buffer2 Addr / Next Desc Addr [7:0] [7:0]	Buffer2 Addr / Next Desc Addr [15:8] [15:8]	Buffer2 Addr / Next Desc Addr [23:16] [23:16]	Buffer2 Addr / Next Desc Addr [31:24] [31:24]	Buffer1 Addr [7:0] [7:0]	Buffer1 Addr [15:8] [15:8]	Buffer1 Addr [23:16] [23:16]	Buffer1 Addr [31:24] [31:24]		

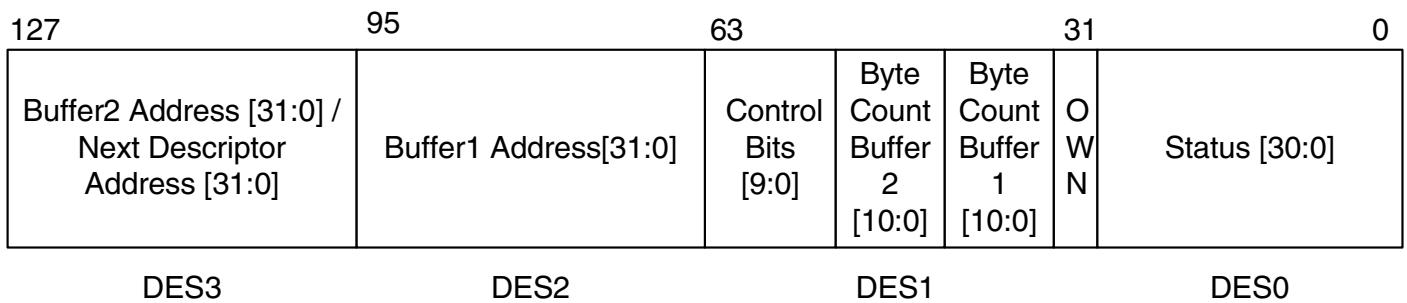
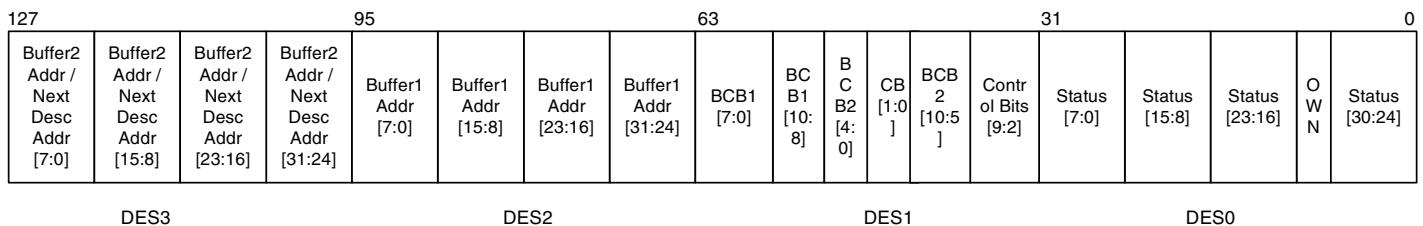
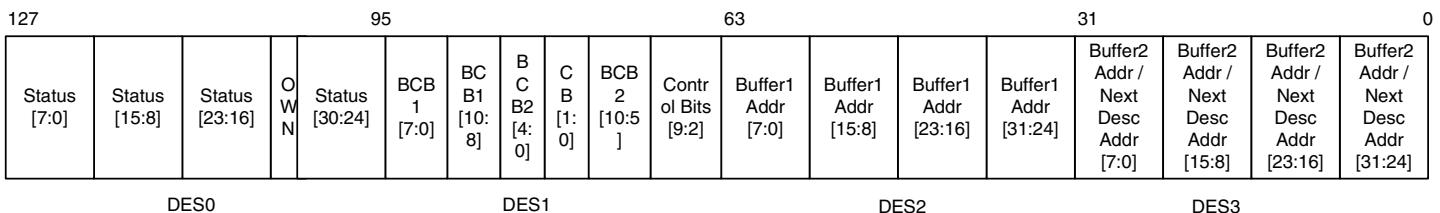
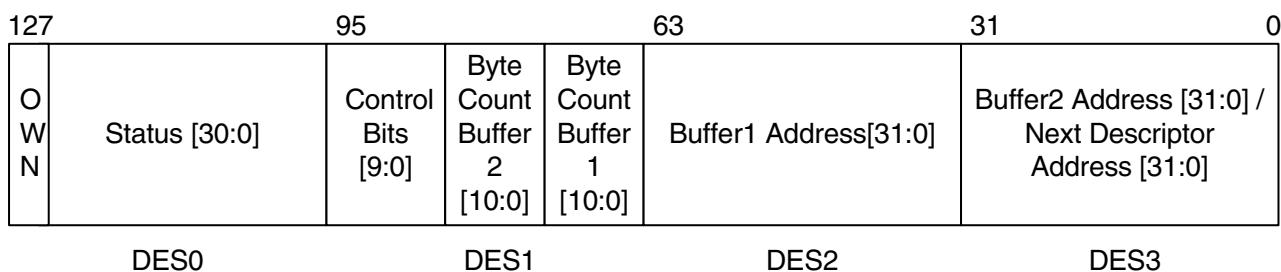
**Figure 7-5 Rx/Tx Descriptors in Same-Endian Mode for 64-Bit, Big-Endian Format Data Bus**

	63	55	47	39	31	23	15	7	0		
DES0-DES1	Status [7:0] [7:0]	Status [15:8] [15:8]	Status [23:16] [23:16]	O W N [30:24] [30:24]	Status [30:24] [30:24]	BCB1 [7:0] [7:0]	BCB1 [10:8] [10:8]	BCB2 [4:0] [4:0]	CB [1:0] [1:0]	BCB2 [10:5] [10:5]	Control Bits [9:2] [9:2]
DES2-DES3	Buffer1 Addr [7:0] [7:0]	Buffer1 Addr [15:8] [15:8]	Buffer1 Addr [23:16] [23:16]	Buffer1 Addr [31:24] [31:24]	Buffer2 Addr / Next Desc Addr [7:0] [7:0]	Buffer2 Addr / Next Desc Addr [15:8] [15:8]	Buffer2 Addr / Next Desc Addr [23:16] [23:16]	Buffer2 Addr / Next Desc Addr [31:24] [31:24]			

**Figure 7-6 Rx/Tx Descriptors in Reverse-Endian Mode for 64-Bit, Big-Endian Format Data Bus**

	63	55	47	39	31	23	15	7	0
DES0-DES1	O W N [30:0]	Status [30:0] [30:0]			Control Bits [9:0] [9:0]		Byte Count Buffer2 [10:0] [10:0]	Byte Count Buffer1 [10:0] [10:0]	
DES2-DES3	Buffer1 Address [31:0] [31:0]			Buffer2 Address [31:0] / Next Descriptor Address [31:0] [31:0]					

Figures 7-7 through 7-10 show the same descriptors with a 128-bit data bus.

**Figure 7-7 Rx/Tx Descriptors in Same-Endian Mode for 128-Bit, Little-Endian Format Data Bus****Figure 7-8 Rx/Tx Descriptors in Reverse-Endian Mode for 128-Bit, Little-Endian Format Data Bus****Figure 7-9 Rx/Tx Descriptors in Same-Endian Mode for 128-Bit, Big-Endian Format Data Bus****Figure 7-10 Rx/Tx Descriptors in Reverse-Endian Mode for 128-Bit, Big-Endian Format Data Bus**

In Big-Endian data bus format with the Same Endianness descriptor format, the descriptor bytes are swapped internally. If the DUT is configured for a 32-bit data bus (Figure 7-2), then byte lanes 3 and 0 are swapped while byte lanes 1 and 2 are swapped. That is, bits [31:24] will be available on data bus [7:0], and vice-versa. Hence, you must ensure that the descriptors are created accordingly in system memory. This is true for Little-Endian data bus with Reverse Endianness for descriptors.

Similarly, byte swapping is illustrated in Figures 7-4 and 7-5 for a 64-bit data bus and in Figures 7-8 and 7-9 for 128-bit data bus.

In 64-bit Big-Endian data bus systems, when the descriptors are configured for Reverse Endianness ([Figure 7-6](#)), the order of the descriptor subset is reversed from the default ([Figure 7-3](#)). For example, bits [63:32] are DES0 while bits [31:0] are DES1.

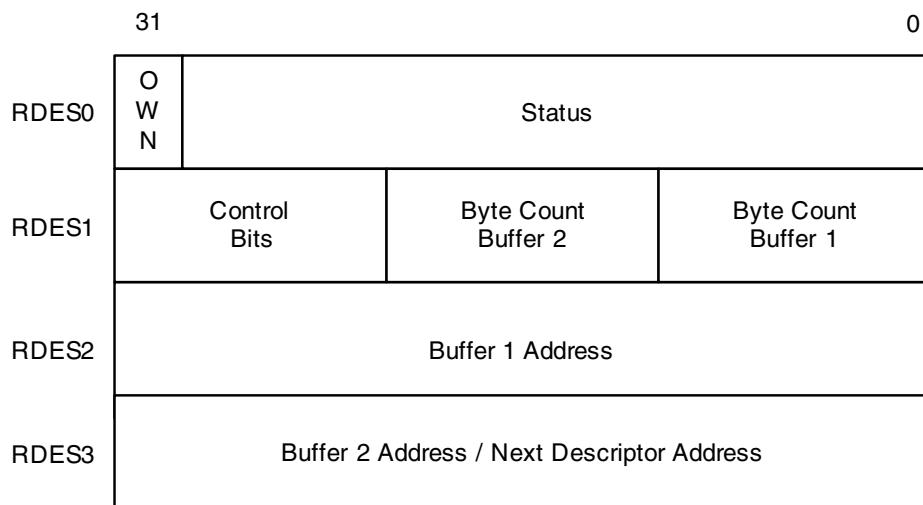
[Figure 7-10](#) illustrates similar phenomenon for 128-bit Big-Endian data bus with Reverse Endian descriptors.

### 7.1.1 Receive Descriptor

The GMAC Subsystem requires at least two descriptors when receiving a frame. The Receive state machine of the DMA (in the GMAC Subsystem) always attempts to acquire an extra descriptor in anticipation of an incoming frame. (The size of the incoming frame is unknown). Before the RxDMA closes a descriptor, it will attempt to acquire the next descriptor even if no frames are received.

In a single descriptor (receive) system, the subsystem will generate a descriptor error if the receive buffer is unable to accommodate the incoming frame and the next descriptor is not owned by the DMA. Thus, the Host is forced to increase either its descriptor pool or the buffer size. Otherwise, the subsystem starts dropping all incoming frames.

**Figure 7-11 Receive Descriptor Format in Little-Endian Mode With a 32-bit Data Bus**



### 7.1.1.1 Receive Descriptor 0 (RDES0)

RDES0 contains the received frame status, the frame length, and the descriptor ownership information. The descriptions in the following tables (Tables 7-1 through 7-9) are for the default mode of a Little-Endian 32-bit data bus with Same Endian descriptors, or Big-Endian data bus with Reverse-Endian descriptors. If the DUT is configured for Big-Endian mode, 32-bit data bus with Same Endian descriptors or Little-Endian data bus with Reverse Endian descriptors, then byte lanes 3 and 0 are swapped while byte lanes 1 and 2 are swapped, that is, bits [31:24] will be available on data bus [7:0], and vice-versa.

**Table 7-1 Receive Descriptor 0**

Bit	Description
31	<b>OWN: Own Bit</b> When set, this bit indicates that the descriptor is owned by the DMA of the GMAC Subsystem. When this bit is reset, this bit indicates that the descriptor is owned by the Host. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.
30	<b>AFM: Destination Address Filter Fail</b> When set, this bit indicates a frame that failed in the DA Filter in the GMAC Core.
29:16	<b>FL: Frame Length</b> These bits indicate the byte length of the received frame that was transferred to host memory (including CRC). This field is valid when Last Descriptor (RDES0[8]) is set and either the Descriptor Error (RDES0[14]) or Overflow Error bits are reset. The frame length also includes the two bytes appended to the Ethernet frame when IP checksum calculation (Type 1) is enabled and the received frame is not a MAC control frame. This field is valid when Last Descriptor (RDES0[8]) is set. When the Last Descriptor and Error Summary bits are not set, this field indicates the accumulated number of bytes that have been transferred for the current frame.
15	<b>ES: Error Summary</b> Indicates the logical OR of the following bits: <ul style="list-style-type: none"> <li>• RDES0[0]: Payload Checksum Error</li> <li>• RDES0[1]: CRC Error</li> <li>• RDES0[3]: Receive Error</li> <li>• RDES0[4]: Watchdog Timeout</li> <li>• RDES0[6]: Late Collision</li> <li>• RDES0[7]: IPC Checksum (Type 2) / Giant Frame</li> <li>• RDES0[11]: Overflow Error</li> <li>• RDES0[14]: Descriptor Error</li> </ul> This field is valid only when the Last Descriptor (RDES0[8]) is set.
14	<b>DE: Descriptor Error</b> When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers, and that the DMA does not own the Next Descriptor. The frame is truncated. This field is valid only when the Last Descriptor (RDES0[8]) is set.
13	<b>SAF: Source Address Filter Fail</b> When set, this bit indicates that the SA field of frame failed the SA Filter in the GMAC Core.

**Table 7-1 Receive Descriptor 0 (Continued)**

Bit	Description
12	<b>LE:</b> Length Error When set, this bit indicates that the actual length of the frame received and that the Length/ Type field does not match. This bit is valid only when the Frame Type (RDES0[5]) bit is reset. Length error status is not valid when CRC error is present.
11	<b>OE:</b> Overflow Error When set, this bit indicates that the received frame was damaged due to buffer overflow in MTL.
10	<b>VLAN:</b> VLAN Tag When set, this bit indicates that the frame pointed to by this descriptor is a VLAN frame tagged by the GMAC Core.
9	<b>FS:</b> First Descriptor When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next Descriptor contains the beginning of the frame.
8	<b>LS:</b> Last Descriptor When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame
7	<b>IPC:</b> Checksum Error/Giant Frame When IP Checksum Engine (Type 1) is enabled, this bit, when set, indicates that the 16-bit IPv4 Header checksum calculated by the core did not match the received checksum bytes. The Error Summary bit[15] is NOT set when this bit is set in this mode. If this bit is set when Full Checksum Offload Engine (Type 2) is enabled, it indicates an error in the IPv4 or IPv6 header. This error can be due to inconsistent Ethernet Type field and IP header Version field values, a header checksum mismatch in IPv4, or an Ethernet frame lacking the expected number of IP header bytes. Refer to <a href="#">Table 7-2</a> for more details. If you do not select IP Checksum Module during core configuration, this bit, when set, indicates that the received frame was a Giant Frame. Giant frames are larger-than-1,518-byte (or 1,522-byte for VLAN) normal frames and larger-than-9,018-byte (9,022-byte for VLAN) frame when Jumbo Frame processing is enabled.
6	<b>LC:</b> Late Collision When set, this bit indicates that a late collision has occurred while receiving the frame in Half-Duplex mode.
5	<b>FT:</b> Frame Type When set, this bit indicates that the Receive Frame is an Ethernet-type frame (the LT field is greater than or equal to 16'h0600). When this bit is reset, it indicates that the received frame is an IEEE802.3 frame. This bit is not valid for Runt frames less than 14 bytes. Refer to <a href="#">Table 7-2</a> for more details.
4	<b>RWT:</b> Receive Watchdog Timeout When set, this bit indicates that the Receive Watchdog Timer has expired while receiving the current frame and the current frame is truncated after the Watchdog Timeout.
3	<b>RE:</b> Receive Error When set, this bit indicates that the gmii_rxer_i signal is asserted while gmii_rxdv_i is asserted during frame reception. This error also includes carrier extension error in GMII and Half-duplex mode. Error can be of less/no extension, or error ( $rxd \neq 0f$ ) during extension.

**Table 7-1** Receive Descriptor 0 (Continued)

Bit	Description
2	DE: Dribble Bit Error When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles). This bit is valid only in MII Mode.
1	CE: CRC Error When set, this bit indicates that a Cyclic Redundancy Check (CRC) Error occurred on the received frame. This field is valid only when the Last Descriptor (RDES0[8]) is set.
0	Rx MAC Address/Payload Checksum Error When set, this bit indicates that the Rx MAC Address registers value (1 to 15) matched the frame's DA field. When reset, this bit indicates that the Rx MAC Address Register 0 value matched the DA field. If Full Checksum Offload Engine is enabled, this bit, when set, indicates the TCP, UDP, or ICMP checksum the core calculated does not match the received encapsulated TCP, UDP, or ICMP segment's Checksum field. This bit is also set when the received number of payload bytes does not match the value indicated in the Length field of the encapsulated IPv4 or IPv6 datagram in the received Ethernet frame. Refer to <a href="#">Table 7-2</a> for more details.

When the Full Checksum Offload Engine (Type 2) is enabled, the permutations of bits 5, 7, and 0 reflect the conditions discussed in [Table 7-2](#).

**Table 7-2** Receive Descriptor 0 When COE (Type 2) Is Enabled

Bit 5: Frame Type	Bit 7: IPC Checksum Error	Bit 0: Payload Checksum Error	Frame Status
0	0	0	IEEE 802.3 Type frame (Length field value is less than 0x0600.)
1	0	0	IPv4/IPv6 Type frame, no checksum error detected
1	0	1	IPv4/IPv6 Type frame with a payload checksum error (as described for PCE) detected
1	1	0	IPv4/IPv6 Type frame with an IP header checksum error (as described for IPC CE) detected
1	1	1	IPv4/IPv6 Type frame with both IP header and payload checksum errors detected
0	0	1	IPv4/IPv6 Type frame with no IP header checksum error and the payload check bypassed, due to an unsupported payload
0	1	1	A Type frame that is neither IPv4 or IPv6 (the Checksum Offload engine bypasses checksum completely.)
0	1	0	Reserved



**Note** The first five conditions are backward-compatible to versions 3.30a and previous. The last two conditions (001, 011), which had been reserved, are not backward-compatible, because the Frame Type (FT) bit is reset during bypasses, even for valid Type frames.

### 7.1.1.2 Receive Descriptor 1 (RDES1)

RDES1 contains the buffer sizes and other bits that control the descriptor chain/ring.



See “[Buffer Size Calculations](#)” on page [67](#) for further detail on calculating buffer sizes.

**Table 7-3 Receive Descriptor 1**

Bit	Description
31	Disable Interrupt on Completion When set, this bit will prevent the setting of the RI (CSR5[6]) bit of the Status Register for the received frame that ends in the buffer pointed to by this descriptor. This, in turn, will disable the assertion of the interrupt to Host due to RI for that frame.
30:26	Reserved
25	RER: Receive End of Ring When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a Descriptor Ring.
24	RCH: Second Address Chained When set, this bit indicates that the second address in the descriptor is the Next Descriptor address rather than the second buffer address. When RDES1[24] is set, RBS2 (RDES1[21-11]) is a “don’t care” value. RDES1[25] takes precedence over RDES1[24].
23:22	Reserved
21:11	RBS2: Receive Buffer 2 Size These bits indicate the second data buffer size in bytes. The buffer size must be a multiple of 4/8/16 depending upon the bus widths (32/64/128), even if the value of RDES3 (buffer2 address pointer) is not aligned to bus width. In the case where the buffer size is not a multiple of 4/8/16, the resulting behavior is undefined. This field is not valid if RDES1[24] is set.
10:0	RBS1: Receive Buffer 1 Size Indicates the first data buffer size in bytes. The buffer size must be a multiple of 4/8/16 depending upon the bus widths (32/64/128), even if the value of RDES2 (buffer1 address pointer) is not aligned. In the case where the buffer size is not a multiple of 4/8/16, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or next descriptor depending on the value of RCH (Bit 24).

### 7.1.1.3 Receive Descriptor 2 (RDES2)

RDES2 contains the address pointer to the first data buffer in the descriptor.



See “[Host Data Buffer Alignment](#)” on page [66](#) for further detail on buffer address alignment.

**Table 7-4 Receive Descriptor 2 (Default Operation)**

Bit	Description
31:0	<p>Buffer 1 Address Pointer</p> <p>These bits indicate the physical address of Buffer 1. There are no limitations on the buffer address alignment except for the following condition: The DMA uses the configured value for its address generation when the RDES2 value is used to store the start of frame. Note that the DMA performs a write operation with the RDES2[3/2/1:0] bits as 0 during the transfer of the start of frame but the frame data is shifted as per the actual Buffer address pointer. The DMA ignores RDES2[3/2/1:0] (corresponding to bus width of 128/64/32) if the address pointer is to a buffer where the middle or last part of the frame is stored.</p>

### 7.1.1.4 Receive Descriptor 3 (RDES3)

RDES3 contains the address pointer either to the second data buffer in the descriptor or to the next descriptor.

**Table 7-5 Receive Descriptor 3**

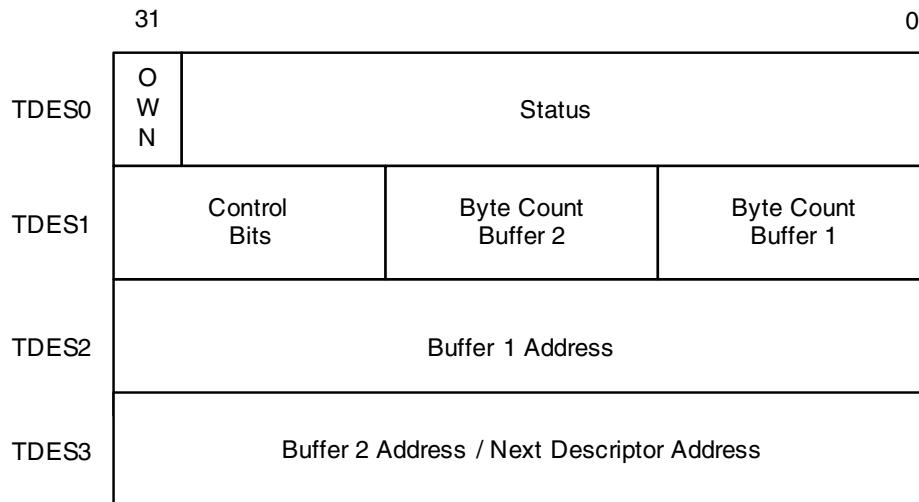
Bit	Description
31:0	<p>Buffer 2 Address Pointer (Next Descriptor Address)</p> <p>These bits indicate the physical address of Buffer 2 when a descriptor ring structure is used. If the Second Address Chained (RDES1[24]) bit is set, this address contains the pointer to the physical memory where the Next Descriptor is present.</p> <p>If RDES1[24] is set, the buffer (Next Descriptor) address pointer must be bus width-aligned (RDES3[3, 2, or 1:0] = 0, corresponding to a bus width of 128, 64, or 32. LSBs are ignored internally.) However, when RDES1[24] is reset, there are no limitations on the RDES3 value, except for the following condition: The DMA uses the configured value for its buffer address generation when the RDES3 value is used to store the start of frame. The DMA ignores RDES3[3, 2, or 1:0] (corresponding to a bus width of 128, 64, or 32) if the address pointer is to a buffer where the middle or last part of the frame is stored.</p>

## 7.1.2 Transmit Descriptor

The descriptor addresses must be aligned to the bus width used (32/64/128). Figure 7-12 shows the transmit descriptor format in Little-Endian mode with a 32-bit data bus. Refer to Figures 7-3 through 7-10 to see the expected descriptor format in other configurations.

Each descriptor is provided with two buffers, two byte-count buffers, and two address pointers, which enable the adapter port to be compatible with various types of memory-management schemes.

**Figure 7-12** Transmit Descriptor Format in Little-Endian Mode With a 32-bit Data Bus



### 7.1.2.1 Transmit Descriptor 0 (TDES0)

TDES0 contains the transmitted frame status and the descriptor ownership information.

**Table 7-6** Transmit Descriptor 0

Bit	Description
31	OWN: Own Bit When set, this bit indicates that the descriptor is owned by the DMA. When this bit is reset, this bit indicates that the descriptor is owned by the Host. The DMA clears this bit either when it completes the frame transmission or when the buffers allocated in the descriptor are empty. The ownership bit of the First Descriptor of the frame should be set after all subsequent descriptors belonging to the same frame have been set. This avoids a possible race condition between fetching a descriptor and the driver setting an ownership bit.
30:18	Reserved
17	TTSS: Tx Time Stamp Status This status bit indicates that a time stamp has been captured for the corresponding transmit frame. When this bit is set, TDES2 and TDES3 have time stamp values that were captured for the transmit frame. This field is valid only when the Last Segment control bit (TDES1[30]) in a descriptor is set. This bit is valid only when IEEE1588 time stamping feature is enabled; otherwise, it is reserved.
16	IHE: IP Header Error When set, this bit indicates that the Checksum Offload engine detected an IP header error and consequently did not modify the transmitted frame for any checksum insertion. This bit is valid only when Full Checksum Offload is enabled; otherwise, it is reserved.

**Table 7-6    Transmit Descriptor 0 (Continued)**

<b>Bit</b>	<b>Description</b>
15	<p><b>ES:</b> Error Summary Indicates the logical OR of the following bits:</p> <ul style="list-style-type: none"> <li>• TDES0[14]: Jabber Timeout</li> <li>• TDES0[13]: Frame Flush</li> <li>• TDES0[11]: Loss of Carrier</li> <li>• TDES0[10]: No Carrier</li> <li>• TDES0[9]: Late Collision</li> <li>• TDES0[8]: Excessive Collision</li> <li>• TDES0[2]: Excessive Deferral</li> <li>• TDES0[1]: Underflow Error</li> </ul>
14	<p><b>JT:</b> Jabber Timeout When set, this bit indicates the GMAC transmitter has experienced a jabber time-out. This bit is only set when the GMAC configuration register's JD bit is not set.</p>
13	<p><b>FF:</b> Frame Flushed When set, this bit indicates that the DMA/MTL flushed the frame due to a SW flush command given by the CPU.</p>
12	<p><b>PCE:</b> Payload Checksum Error This bit, when set, indicates that the Checksum Offload engine had a failure and did not insert any checksum into the encapsulated TCP, UDP, or ICMP payload. This failure can be either due to insufficient bytes, as indicated by the IP Header's Payload Length field, or the MTL starting to forward the frame to the MAC transmitter in Store-and-Forward mode without the checksum having been calculated yet. This second error condition only occurs when the Transmit FIFO depth is less than the length of the Ethernet frame being transmitted: to avoid deadlock, the MTL starts forwarding the frame when the FIFO is full, even in Store-and-Forward mode. When the Full Checksum Offload engine is not enabled during configuration, this bit is reserved.</p>
11	<p><b>LC:</b> Loss of Carrier When set, this bit indicates that Loss of Carrier occurred during frame transmission (that is, the gmii_crs_i signal was inactive for one or more transmit clock periods during frame transmission). This is valid only for the frames transmitted without collision and when the GMAC operates in Half-Duplex Mode.</p>
10	<p><b>NC:</b> No Carrier When set, this bit indicates that the carrier sense signal from the PHY was not asserted during transmission.</p>
9	<p><b>LC:</b> Late Collision When set, this bit indicates that frame transmission was aborted due to a collision occurring after the collision window (64 byte times including Preamble in MII Mode and 512 byte times including Preamble and Carrier Extension in GMII Mode). Not valid if Underflow Error is set.</p>
8	<p><b>EC:</b> Excessive Collision When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If the DR (Disable Retry) bit in the GMAC Configuration Register is set, this bit is set after the first collision and the transmission of the frame is aborted.</p>
7	<p><b>VF:</b> VLAN Frame When set, this bit indicates that the transmitted frame was a VLAN-type frame.</p>

**Table 7-6** Transmit Descriptor 0 (Continued)

Bit	Description
6:3	CC: Collision Count This 4-bit counter value indicates the number of collisions occurring before the frame was transmitted. The count is not valid when the Excessive Collisions bit (TDES0[8]) is set.
2	ED: Excessive Deferral When set, this bit indicates that the transmission has ended because of excessive deferral of over 24,288 bit times (155,680 bits times in 1000-Mbps mode, or in Jumbo Frame enabled mode) if the Deferral Check (DC) bit is set high in the GMAC Control Register.
1	UF: Underflow Error When set, this bit indicates that the GMAC aborted the frame because data arrived late from the Host memory. Underflow Error indicates that the DMA encountered an empty Transmit Buffer while transmitting the frame. The transmission process enters the suspended state and sets both Transmit Underflow (Register 5[5]) and Transmit Interrupt (Register 5[0]).
0	DB: Deferred Bit When set, this bit indicates that the GMAC defers before transmission because of the presence of carrier. This bit is valid only in Half-Duplex mode.

### 7.1.2.2 Transmit Descriptor 1 (TDES1)

TDES1 contains the buffer sizes and other bits which control the descriptor chain/ring and the frame being transferred.



See “Buffer Size Calculations” on page 67 for further detail on calculating buffer sizes.

**Table 7-7** Transmit Descriptor 1

Bit	Description
31	IC: Interrupt on Completion When set, this bit sets Transmit Interrupt (Register 5[0]) after the present frame has been transmitted.
30	LS: Last Segment When set, this bit indicates that the buffer contains the last segment of the frame.
29	FS: First Segment When set, this bit indicates that the buffer contains the first segment of a frame.

**Table 7-7    Transmit Descriptor 1 (Continued)**

Bit	Description
28:27	CIC: Checksum Insertion Control These bits control the insertion of checksums in Ethernet frames that encapsulate TCP, UDP, or ICMP over IPv4 or IPv6 as described below. <ul style="list-style-type: none"><li>• 2'b00: Do nothing. Checksum Engine is bypassed</li><li>• 2'b01: Insert IPv4 header checksum. Use this value to insert IPv4 header checksum when the frame encapsulates an IPv4 datagram.</li><li>• 2'b10: Insert TCP/UDP/ICMP checksum. The checksum is calculated over the TCP, UDP, or ICMP segment only and the TCP, UDP, or ICMP pseudo-header checksum is assumed to be present in the corresponding input frame's Checksum field. An IPv4 header checksum is also inserted if the encapsulated datagram conforms to IPv4.</li><li>• 2'b11: Insert a TCP/UDP/ICMP checksum that is fully calculated in this engine. In other words, the TCP, UDP, or ICMP pseudo-header is included in the checksum calculation, and the input frame's corresponding Checksum field has an all-zero value. An IPv4 Header checksum is also inserted if the encapsulated datagram conforms to IPv4.</li></ul> The Checksum engine detects whether the TCP, UDP, or ICMP segment is encapsulated in IPv4 or IPv6 and processes its data accordingly.
26	DC: Disable CRC When set, the GMAC does not append the Cyclic Redundancy Check (CRC) to the end of the transmitted frame. This is valid only when the first segment (TDES1[29]).
25	TER: Transmit End of Ring When set, this bit indicates that the descriptor list reached its final descriptor. The returns to the base address of the list, creating a descriptor ring.
24	TCH: Second Address Chained When set, this bit indicates that the second address in the descriptor is the Next Descriptor address rather than the second buffer address. When TDES1[24] is set, TBS2 (TDES1[21–11]) are “don't care” values. TDES1[25] takes precedence over TDES1[24].
23	DP: Disable Padding When set, the GMAC does not automatically add padding to a frame shorter than 64 bytes. When this bit is reset, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes and the CRC field is added despite the state of the DC (TDES1[26]) bit. This is valid only when the first segment (TDES1[29]) is set.
22	TTSE: Transmit Time Stamp Enable When set, this bit enables IEEE1588 hardware time stamping for the transmit frame referenced by the descriptor. This field is valid only when the First Segment control bit (TDES1[29]) is set.
21:11	TBS2: Transmit Buffer 2 Size These bits indicate the Second Data Buffer in bytes. This field is not valid if TDES1[24] is set.
10:0	TBS1: Transmit Buffer 1 Size These bits indicate the First Data Buffer byte size. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or next descriptor depending on the value of TCH (Bit 24).

### 7.1.2.3 Transmit Descriptor 2 (TDES2)

TDES2 contains the address pointer to the first buffer of the descriptor.

**Table 7-8** **Transmit Descriptor 2**

Bit	Description
31:0	<p>Buffer 1 Address Pointer</p> <p>These bits indicate the physical address of Buffer 1. There is no limitation on the buffer address alignment. See “<a href="#">Host Data Buffer Alignment</a>” on page <a href="#">66</a> for further detail on buffer address alignment.</p>

### 7.1.2.4 Transmit Descriptor 3 (TDES3)

TDES3 contains the address pointer either to the second buffer of the descriptor or the next descriptor.

**Table 7-9** **Transmit Descriptor 3**

Bit	Description
31:0	<p>Buffer 2 Address Pointer (Next Descriptor Address)</p> <p>Indicates the physical address of Buffer 2 when a descriptor ring structure is used. If the Second Address Chained (TDES1[24]) bit is set, this address contains the pointer to the physical memory where the Next Descriptor is present. The buffer address pointer must be aligned to the bus width only when TDES1[24] is set. (LSBs are ignored internally.)</p>

### 7.1.3 Descriptor Format With IEEE 1588 Time Stamping Enabled

The default descriptor format (as described in “[Receive Descriptor](#)” on page 341 and “[Transmit Descriptor](#)” on page 347), and field descriptions remain unchanged when created by software (Own bit is set in DES0). However, if the software has enabled IEEE 1588 functionality, the DES2 and DES3 descriptor fields (see [Figure 7-13](#)) take on a different meaning when the DMA closes the descriptor (own bit in DES0 is cleared).

The DMA updates the DES2 and DES3 with the time stamp value before clearing the Own bit in DES0.

When the core is operating in 32-bit mode ([Figures 7-1](#) and [7-2](#)), DES2 is updated with the lower 32 time stamp bits (the Sub-Second field, called TSL in subsequent sections) and DES3 is updated with the upper 32 time stamp bits (the Seconds field, called TSH in subsequent sections).

**Figure 7-13 Receive Descriptor Fields When DMA Clears the Own Bit**

	31	1
DES0		
DES1		
DES2	Time Stamp Low[31:0]	
DES3	Time Stamp High[31:0]	

The above description holds true for all descriptor formats depicted in [Figures 7-3](#), [7-5](#), [7-7](#), and [7-9](#). However, when the core operates in 64- or 128-bit and with a reverse-endian descriptor ([Figures 7-4](#), [7-6](#), [7-8](#), and [7-10](#)), DES2 is updated with TSH and DES3 is updated with TSL. This ensures that the 64-bit time stamp can be processed as-is without any swapping between the two 32-bit words.

The following sections describe the details specific to receive and transmit descriptors in this mode.

#### 7.1.3.1 Receive Descriptor

##### 7.1.3.1.1 Receive Time Stamp

The tables below describe the fields that have different meaning for RDES2 and RDES3 when the receive descriptor is closed and time stamping is enabled. The fields in RDES2 and RDES3 are swapped when the core operates in 64- or 128-bit and with the descriptor in Reverse-Endian mode.



**Note** When software disables the time stamping feature (the TSENA bit in Register 448 is low), the DMA does *not* update the descriptor’s RDES2/RDES3 fields before closing the RDES0.

**Table 7-10 Receive Descriptor Fields (RDES2)**

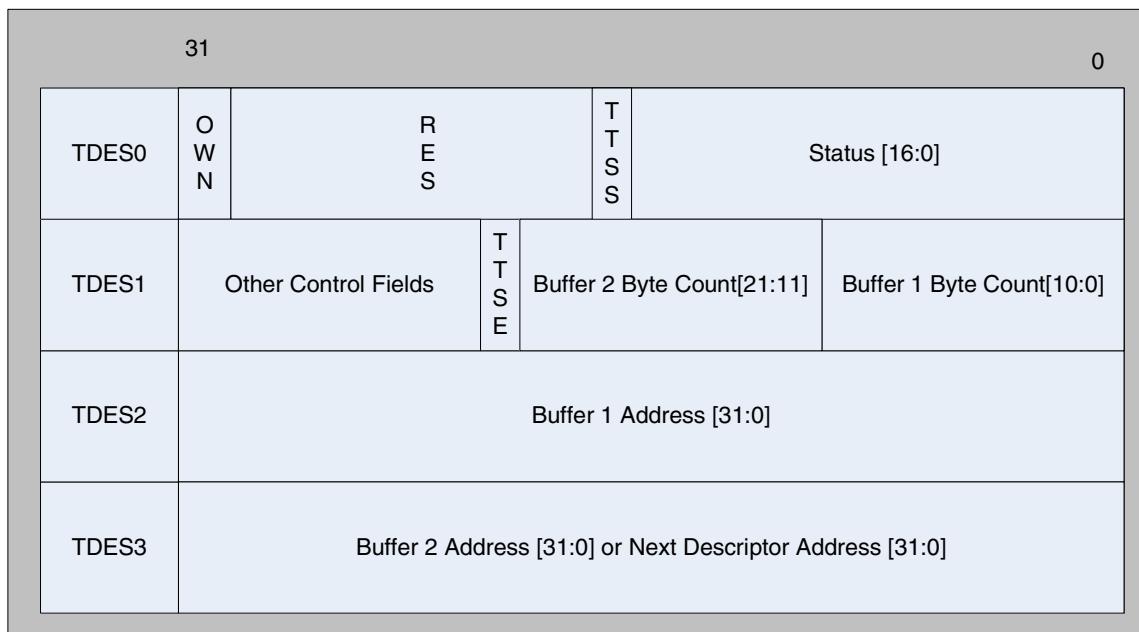
Bit	Description
31:0	RTSL: Receive Frame Time Stamp Low The DMA updates this field with the least significant 32 bits of the time stamp captured for the corresponding receive frame. The DMA updates this field only for the last descriptor of the receive frame indicated by Last Descriptor status bit (RDES0[8]). When this field and the RTSH field in RDES3 show an all-ones value, the time stamp must be treated as corrupt.

**Table 7-11 Receive Descriptor Fields (RDES3)**

Bit	Description
31:0	RTSH: Receive Frame Time Stamp High The DMA updates this field with the most significant 32 bits of the time stamp captured for the corresponding receive frame. The DMA updates this field only for the last descriptor of the receive frame indicated by Last Descriptor status bit (RDES0[8]). When this field and RDES2's RTSL field show all-ones values, the time stamp must be treated as corrupt.

### 7.1.3.2 Transmit Descriptor

In addition to the changes described in “Descriptor Format With IEEE 1588 Time Stamping Enabled” on page 352, the Transmit descriptor has additional control and status bits (TTSE and TTSS, respectively) for time stamping, as shown in Figure 7-14. Software sets the TTSE bit (when the Own bit is set), instructing the core to generate a time stamp for the corresponding Ethernet frame being transmitted. The DMA sets the TTSS bit if the time stamp has been updated in the TDES2 and TDES3 fields when the descriptor is closed (Own bit is cleared).

**Figure 7-14 Transmit Descriptor Fields– Normal Format**

### 7.1.3.2.1 Transmit Time Stamp Control and Status Fields

The position of these fields is different for normal transmit descriptor and enhanced format transmit descriptor. The value of this field in both the cases shall be preserved by the DMA at the time of closing the descriptor.

Updates to [Tables 7-5](#) and [7-6](#) for the default (normal) descriptor format are described below.

**Table 7-12 Transmit Time Stamp Status – Normal Descriptor Format Case (TDES0)**

Bit	Description
17	<b>TTSS: Transmit Time Stamp Status</b> This field is a status bit indicating that a time stamp was captured for the corresponding transmit frame. When this bit is set, both TDES2 and TDES3 have a time stamp value that was captured for the transmit frame. This field is valid only when the Last Segment control bit (TDES1[30] in the descriptor) is set.

**Table 7-13 Transmit Time Stamp Control – Normal Descriptor Format Case (TDES1)**

Bit	Description
22	<b>TTSE: Transmit Time Stamp Enable</b> When set, this field enables IEEE1588 hardware time stamping for the transmit frame described by the descriptor. This field is valid only when the First Segment control bit (TDES1[29] in the descriptor) is set.

### 7.1.3.2.2 Transmit Time Stamp Field

The transmit descriptor format and field descriptions remain unchanged when they are created by software (when the Own bit is set). However, when the DMA closes the last descriptor (marked, in the alternative descriptor format, by the LS bit in TDES1 or TDES0) and IEEE 1588 functionality is enabled (the Own bit is cleared), the TDES2 and TDES3 descriptor fields are updated with the time stamp, if taken, for that frame.

The fields in TDES2 and TDES3 are swapped when the core operates in 64- or 128-bit and with the descriptor in Reverse-Endian mode

[Tables 7-14](#) and [7-15](#) describe the fields that have different meaning when the descriptor is closed.

**Table 7-14 Transmit Descriptor Fields (TDES2)**

Bit	Description
31:0	<b>TTSL: Transmit Frame Time Stamp Low</b> This field is updated by DMA with the least significant 32 bits of the time stamp captured for the corresponding transmit frame. This field has the time stamp only if the Last Segment control bit (LS) in the descriptor is set.

**Table 7-15 Transmit Descriptor Fields (TDES3)**

Bit	Description
31:0	TTSH: Transmit Frame Time Stamp High This field is updated by DMA with the most significant 32 bits of the time stamp captured for the corresponding transmit frame. This field has the time stamp only if the Last Segment control bit (LS) in the descriptor is set.

## 7.2 Alternate or Enhanced Descriptors

The alternate (or enhanced) descriptor structure can have 8 DWORDS (32-bytes) instead of the 4 DWORDS as in the case of normal descriptor format. The features of the alternate descriptor structure are

- ❖ The normal descriptor structure allows data buffers of up to 2,048 bytes. The alternative descriptor structure has been implemented to support buffers of up to 8 KB (useful for Jumbo frames).
- ❖ There is a re-assignment of control and status bits in TDES0, TDES1, RDES0 (Advanced time stamp or IPC full offload configuration), RDES1.
- ❖ The transmit descriptor stores the time stamp in TDES6 and TDES7 when advanced time stamp feature is selected.
- ❖ This receive descriptor structure is also used for storing the extended status (RDES4) and time stamp (RDES6 and RDES7) when advanced time stamp feature or IPC full offload is selected.
- ❖ When alternate descriptor mode is selected, and Time-stamping feature is enabled, the software needs to allocate 32-bytes (8 DWORDS) of memory for every descriptor. When Time-stamping or Receive IPC FullOffload engine are not enabled, the extended descriptors are not required and the SW can use alternate descriptors with the default size of 16 bytes. The core also needs to be configured for this change using the DMA Bus Mode Register[7].
- ❖ When alternate descriptor is chosen without Time Stamp or Full IPC Offload feature, the descriptor size is always 4 DWORDs (DES0-DES3).

The description or bit-mapping alternate descriptor structure (in Little Endian mode) is given below.



**Note** The effect of Big Endian mode (byte-swap) explained in “[Normal Descriptor Formats](#)” on page [337](#) apply to this descriptor structure as well.



**Note** When alternate descriptor with only Full IPC Offload (Type 2) is selected, it is not backward compatible to the previous release 3.4x with respect to status bits[7,5,0] in RDES0. In this mode, you should enable the extended descriptor mode (8 DWORDS) to get the IPC checksum engine status in RDES4.

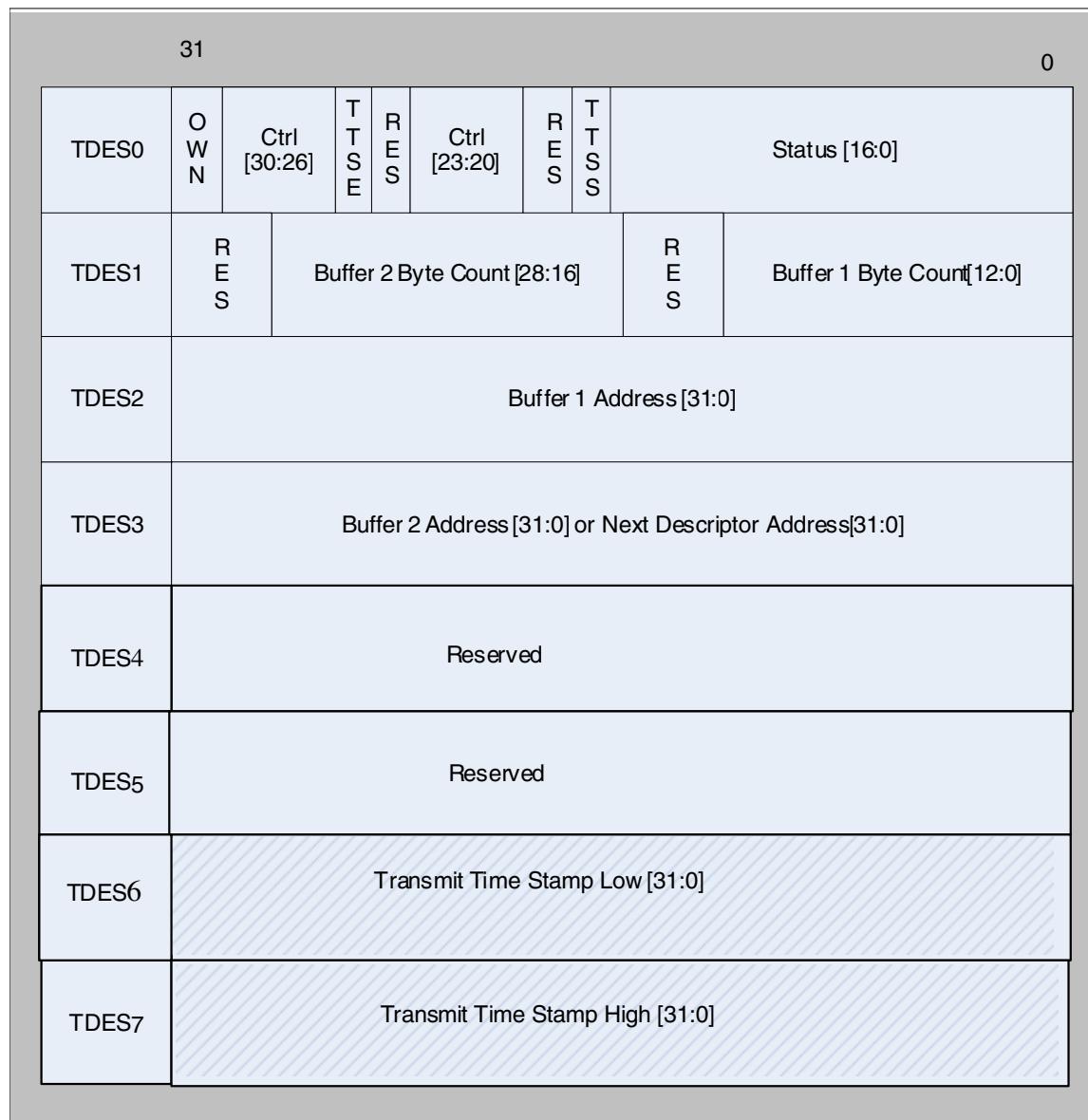
### 7.2.1 Transmit Descriptor

The transmit descriptor structure is shown in [Figure 7-15](#). The application software must program the control bits TDES0[31:20] during descriptor initialization. When the DMA updates the descriptor, it writes back all the control bits except the OWN bit (which it clears) and updates the status bits[19:0]. The contents of the transmitter descriptor word 0 (TDES0) through word 3 (TDES3) are given in [Tables 7-16](#) through [7-19](#), respectively.

With the advance time stamp support, the snapshot of the time stamp to be taken can be enabled for a given frame by setting the “TTSE: Transmit Time Stamp Enable” (bit-25 of TDES0). When the descriptor is closed (i.e. when the OWN bit is cleared), the time-stamp is written into TDES6 and TDES7. This is indicated by the status bit “TTSS: Transmit Time Stamp Status” (bit-17 of TDES0). This is shown in [Figure 7-15](#). The contents of TDES6 and TDES7 are mentioned in [Table 7-20](#) and [Table 7-21](#).



**Note** When either of Advanced Time Stamp or IPC Offload (Type 2) features is enabled, the SW should set the DMA Bus Mode register[7], so that the DMA operates with extended descriptor size. When this control bit is reset, the TDES4-TDES7 descriptor space are not valid.

**Figure 7-15 Transmitter Descriptor Fields - Alternate (Enhanced) Format**

The above description for TDES6/7 holds true for all descriptor formats depicted in [Figures 7-3, 7-5, 7-7, and 7-9](#). However, when the core operates in 64- or 128-bit and with a reverse-endian descriptor ([Figures 7-4, 7-6, 7-8, and 7-10](#)), TDES6 is updated with TTSH and TDES7 is updated with TTSL. This ensures that the 64- bit time stamp can be processed as-is without any swapping between the two 32-bit words.

**Table 7-16 Transmit Descriptor Word 0 (TDES0)**

Bit	Description
31	<p><b>OWN:</b> Own Bit</p> <p>When set, this bit indicates that the descriptor is owned by the DMA. When this bit is reset, it indicates that the descriptor is owned by the Host. The DMA clears this bit either when it completes the frame transmission or when the buffers allocated in the descriptor are read completely. The ownership bit of the frame's first descriptor must be set after all subsequent descriptors belonging to the same frame have been set. This avoids a possible race condition between fetching a descriptor and the driver setting an ownership bit.</p>
30	<p><b>IC:</b> Interrupt on Completion</p> <p>When set, this bit sets the Transmit Interrupt (Register 5[0]) after the present frame has been transmitted.</p>
29	<p><b>LS:</b> Last Segment</p> <p>When set, this bit indicates that the buffer contains the last segment of the frame.</p>
28	<p><b>FS:</b> First Segment</p> <p>When set, this bit indicates that the buffer contains the first segment of a frame.</p>
27	<p><b>DC:</b> Disable CRC</p> <p>When this bit is set, the GMAC does not append a cyclic redundancy check (CRC) to the end of the transmitted frame. This is valid only when the first segment (TDES0[28]) is set.</p>
26	<p><b>DP:</b> Disable Pad</p> <p>When set, the GMAC does not automatically add padding to a frame shorter than 64 bytes. When this bit is reset, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes, and the CRC field is added despite the state of the DC (TDES0[27]) bit. This is valid only when the first segment (TDES0[28]) is set.</p>
25	<p><b>TTSE:</b> Transmit Time Stamp Enable</p> <p>When set, this bit enables IEEE1588 hardware time stamping for the transmit frame referenced by the descriptor. This field is valid only when the First Segment control bit (TDES0[28]) is set.</p>
24	Reserved
23:22	<p><b>CIC:</b> Checksum Insertion Control</p> <p>These bits control the checksum calculation and insertion. Bit encodings are as shown below.</p> <ul style="list-style-type: none"> <li>• 2'b00: Checksum Insertion Disabled.</li> <li>• 2'b01: Only IP header checksum calculation and insertion are enabled.</li> <li>• 2'b10: IP header checksum and payload checksum calculation and insertion are enabled, but pseudo-header checksum is not calculated in hardware.</li> <li>• 2'b11: IP Header checksum and payload checksum calculation and insertion are enabled, and pseudo-header checksum is calculated in hardware.</li> </ul> <p>This field is reserved when the IPC_FULL_OFFLOAD configuration parameter is not selected.</p>
21	<p><b>TER:</b> Transmit End of Ring</p> <p>When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring.</p>

**Table 7-16 Transmit Descriptor Word 0 (TDES0) (Continued)**

Bit	Description
20	TCH: Second Address Chained When set, this bit indicates that the second address in the descriptor is the Next Descriptor address rather than the second buffer address. When TDES0[20] is set, TBS2 (TDES1[28:16]) is a “don’t care” value. TDES0[21] takes precedence over TDES0[20].
19:18	Reserved
17	TTSS: Transmit Time Stamp Status This field is used as a status bit to indicate that a time stamp was captured for the described transmit frame. When this bit is set, TDES2 and TDES3 have a time stamp value captured for the transmit frame. This field is only valid when the descriptor’s Last Segment control bit (TDES0[29]) is set.
16	IHE: IP Header Error When set, this bit indicates that the GMAC transmitter detected an error in the IP datagram header. The transmitter checks the header length in the IPv4 packet against the number of header bytes received from the application and indicates an error status if there is a mismatch. For IPv6 frames, a header error is reported if the main header length is not 40 bytes. Furthermore, the Ethernet Length/Type field value for an IPv4 or IPv6 frame must match the IP header version received with the packet. For IPv4 frames, an error status is also indicated if the Header Length field has a value less than 0x5.
15	ES: Error Summary Indicates the logical OR of the following bits: <ul style="list-style-type: none"> <li>• TDES0[14]: Jabber Timeout</li> <li>• TDES0[13]: Frame Flush</li> <li>• TDES0[11]: Loss of Carrier</li> <li>• TDES0[10]: No Carrier</li> <li>• TDES0[9]: Late Collision</li> <li>• TDES0[8]: Excessive Collision</li> <li>• TDES0[2]: Excessive Deferral</li> <li>• TDES0[1]: Underflow Error</li> <li>• TDES0[16]: IP Header Error</li> <li>• TDES0[12]: IP Payload Error</li> </ul>
14	JT: Jabber Timeout When set, this bit indicates the GMAC transmitter has experienced a jabber time-out. This bit is only set when the GMAC configuration register’s JD bit is not set.
13	FF: Frame Flushed When set, this bit indicates that the DMA/MTL flushed the frame due to a software Flush command given by the CPU.
12	IPE: IP Payload Error When set, this bit indicates that GMAC transmitter detected an error in the TCP, UDP, or ICMP IP datagram payload. The transmitter checks the payload length received in the IPv4 or IPv6 header against the actual number of TCP, UDP, or ICMP packet bytes received from the application and issues an error status in case of a mismatch.

**Table 7-16 Transmit Descriptor Word 0 (TDES0) (Continued)**

Bit	Description
11	LC: Loss of Carrier When set, this bit indicates that a loss of carrier occurred during frame transmission (that is, the gmii_crs_i signal was inactive for one or more transmit clock periods during frame transmission). This is valid only for the frames transmitted without collision when the GMAC operates in Half-Duplex mode.
10	NC: No Carrier When set, this bit indicates that the Carrier Sense signal from the PHY was not asserted during transmission.
9	LC: Late Collision When set, this bit indicates that frame transmission was aborted due to a collision occurring after the collision window (64 byte-times, including preamble, in MII mode and 512 byte-times, including preamble and carrier extension, in GMII mode). This bit is not valid if the Underflow Error bit is set.
8	EC: Excessive Collision When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If the DR (Disable Retry) bit in the GMAC Configuration register is set, this bit is set after the first collision, and the transmission of the frame is aborted.
7	VF: VLAN Frame When set, this bit indicates that the transmitted frame was a VLAN-type frame.
6:3	CC: Collision Count This 4-bit counter value indicates the number of collisions occurring before the frame was transmitted. The count is not valid when the Excessive Collisions bit (TDES0[8]) is set.
2	ED: Excessive Deferral When set, this bit indicates that the transmission has ended because of excessive deferral of over 24,288 bit times (155,680 bits times in 1,000-Mbps mode or if Jumbo Frame is enabled) if the Deferral Check (DC) bit in the GMAC Control register is set high.
1	UF: Underflow Error When set, this bit indicates that the GMAC aborted the frame because data arrived late from the Host memory. Underflow Error indicates that the DMA encountered an empty transmit buffer while transmitting the frame. The transmission process enters the Suspended state and sets both Transmit Underflow (Register 5[5]) and Transmit Interrupt (Register 5[0]).
0	DB: Deferred Bit When set, this bit indicates that the GMAC defers before transmission because of the presence of carrier. This bit is valid only in Half-Duplex mode.

**Table 7-17 Transmit Descriptor Word 1 (TDES1)**

Bit	Description
31:29	Reserved
28:16	TBS2: Transmit Buffer 2 Size These bits indicate the second data buffer size in bytes. This field is not valid if TDES0[20] is set. See “ <a href="#">Buffer Size Calculations</a> ” on page <a href="#">67</a> for further detail on calculating buffer sizes.
15:13	Reserved
12:0	TBS1: Transmit Buffer 1 Size These bits indicate the first data buffer byte size, in bytes. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or the next descriptor, depending on the value of TCH (TDES0[20]).

**Table 7-18 Transmit Descriptor 2 (TDES2)**

Bit	Description
31:0	Buffer 1 Address Pointer These bits indicate the physical address of Buffer 1. There is no limitation on the buffer address alignment. See “ <a href="#">Host Data Buffer Alignment</a> ” on page <a href="#">66</a> for further detail on buffer address alignment.

**Table 7-19 Transmit Descriptor 3 (TDES3)**

Bit	Description
31:0	Buffer 2 Address Pointer (Next Descriptor Address) Indicates the physical address of Buffer 2 when a descriptor ring structure is used. If the Second Address Chained (TDES1[24]) bit is set, this address contains the pointer to the physical memory where the Next Descriptor is present. The buffer address pointer must be aligned to the bus width only when TDES1[24] is set. (LSBs are ignored internally.)

**Table 7-20 Transmit Descriptor 6 (TDES6)**

Bit	Description
31:0	TTSL: Transmit Frame Time Stamp Low This field is updated by DMA with the least significant 32 bits of the time stamp captured for the corresponding transmit frame. This field has the time stamp only if the Last Segment bit (LS) in the descriptor is set and Time stamp status (TTSS) bit is set.

**Table 7-21 Transmit Descriptor 7 (TDES7)**

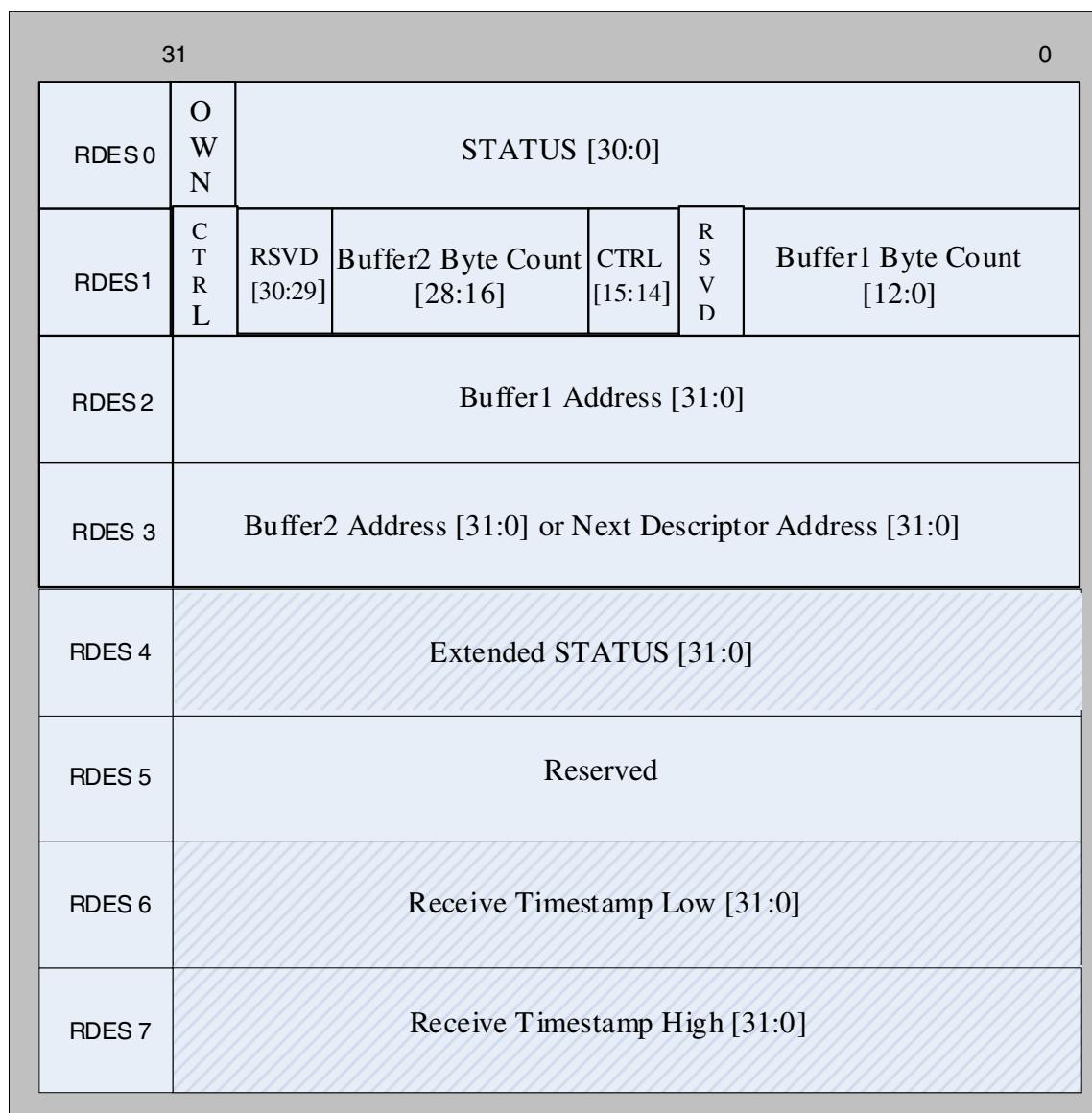
Bit	Description
31:0	TTSH: Transmit Frame Time Stamp High This field is updated by DMA with the most significant 32 bits of the time stamp captured for the corresponding receive frame. This field has the time stamp only if the Last Segment bit (LS) in the descriptor is set and Time stamp status (TTSS) bit is set.

## 7.2.2 Receive Descriptor

The structure of the received descriptor is shown in [Figure 7-16](#). This can have 32 bytes of descriptor data (8 DWORDs) when Advanced Time Stamping or IPC Full Offload feature is selected.



**Note** When either of these features is enabled, the SW should set the DMA Bus Mode register[7] so that the DMA operates with extended descriptor size. When this control bit is reset, RDES0[7] and RDES0[0] will be always cleared and the RDES4-RDES7 descriptor space are not valid

**Figure 7-16 Receive Descriptor Fields - Alternate (Enhanced) Format**

The contents of RDES0 are identified in [Table 7-22](#). The contents of RDES1 through RDES3 are identified in [Tables 7-23](#) through [7-25](#), respectively.



**Note** Some of the bit functions of RDES0 are not backward compatible to Release 3.41a and previous versions. These bits are Bit[7], Bit[0] and Bit[5]. The function of Bit[5] is backward compatible to Release 3.30a and previous versions.

**Table 7-22 Receive Descriptor Fields (RDES0)**

Bit	Description
31	OWN: Own Bit When set, this bit indicates that the descriptor is owned by the DMA of the GMAC Subsystem. When this bit is reset, this bit indicates that the descriptor is owned by the Host. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.
30	AFM: Destination Address Filter Fail When set, this bit indicates a frame that failed in the DA Filter in the GMAC Core.
29:16	FL: Frame Length These bits indicate the byte length of the received frame that was transferred to host memory (including CRC). This field is valid when Last Descriptor (RDES0[8]) is set and either the Descriptor Error (RDES0[14]) or Overflow Error bits are reset. The frame length also includes the two bytes appended to the Ethernet frame when IP checksum calculation (Type 1) is enabled and the received frame is not a MAC control frame. This field is valid when Last Descriptor (RDES0[8]) is set. When the Last Descriptor and Error Summary bits are not set, this field indicates the accumulated number of bytes that have been transferred for the current frame.
15	ES: Error Summary Indicates the logical OR of the following bits: <ul style="list-style-type: none"> <li>• RDES0[1]: CRC Error</li> <li>• RDES0[3]: Receive Error</li> <li>• RDES0[4]: Watchdog Timeout</li> <li>• RDES0[6]: Late Collision</li> <li>• RDES0[7]: Giant Frame</li> <li>• RDES4[4:3]: IP Header/Payload Error</li> <li>• RDES0[11]: Overflow Error</li> <li>• RDES0[14]: Descriptor Error</li> </ul> This field is valid only when the Last Descriptor (RDES0[8]) is set.
14	DE: Descriptor Error When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers, and that the DMA does not own the Next Descriptor. The frame is truncated. This field is valid only when the Last Descriptor (RDES0[8]) is set.
13	SAF: Source Address Filter Fail When set, this bit indicates that the SA field of frame failed the SA Filter in the GMAC Core.

**Table 7-22 Receive Descriptor Fields (RDES0)**

Bit	Description
12	<b>LE:</b> Length Error When set, this bit indicates that the actual length of the frame received and that the Length/ Type field does not match. This bit is valid only when the Frame Type (RDES0[5]) bit is reset.
11	<b>OE:</b> Overflow Error When set, this bit indicates that the received frame was damaged due to buffer overflow in MTL.
10	<b>VLAN:</b> VLAN Tag When set, this bit indicates that the frame pointed to by this descriptor is a VLAN frame tagged by the GMAC Core.
9	<b>FS:</b> First Descriptor When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next Descriptor contains the beginning of the frame.
8	<b>LS:</b> Last Descriptor When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame
7	<b>Time Stamp Available/IP Checksum Error (Type1) /Giant Frame</b> When Advanced Time Stamp feature is present, this bit, when set, indicates that a snapshot of the timestamp is written in descriptor words 6 (RDES6) and 7 (RDES7). This is valid only when the Last Descriptor bit (RDES0[8]) is set. When IP Checksum Engine (Type 1) is selected, this bit, when set, indicates that the 16-bit IPv4 Header checksum calculated by the core did not match the received checksum bytes. Otherwise, this bit, when set, indicates the Giant Frame Status. Giant frames are larger-than-1,518-byte (or 1,522-byte for VLAN) normal frames and larger-than-9,018-byte (9,022-byte for VLAN) frame when Jumbo Frame processing is enabled.
6	<b>LC:</b> Late Collision When set, this bit indicates that a late collision has occurred while receiving the frame in Half-Duplex mode.
5	<b>FT:</b> Frame Type When set, this bit indicates that the Receive Frame is an Ethernet-type frame (the LT field is greater than or equal to 16'h0600). When this bit is reset, it indicates that the received frame is an IEEE802.3 frame. This bit is not valid for Runt frames less than 14 bytes.
4	<b>RWT:</b> Receive Watchdog Timeout When set, this bit indicates that the Receive Watchdog Timer has expired while receiving the current frame and the current frame is truncated after the Watchdog Timeout.
3	<b>RE:</b> Receive Error When set, this bit indicates that the gmii_rxer_i signal is asserted while gmii_rxdv_i is asserted during frame reception. This error also includes carrier extension error in GMII and Half-duplex mode. Error can be of less/no extension, or error ( $rxd \neq 0f$ ) during extension.
2	<b>DE:</b> Dribble Bit Error When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles). This bit is valid only in MII Mode.

**Table 7-22 Receive Descriptor Fields (RDES0)**

Bit	Description
1	<p><b>CE: CRC Error</b></p> <p>When set, this bit indicates that a Cyclic Redundancy Check (CRC) Error occurred on the received frame. This field is valid only when the Last Descriptor (RDES0[8]) is set.</p>
0	<p><b>Extended Status Available/Rx MAC Address</b></p> <p>When either Advanced Time Stamp or IP Checksum Offload (Type 2) is present, this bit, when set, indicates that the extended status is available in descriptor word 4 (RDES4). This is valid only when the Last Descriptor bit (RDES0[8]) is set.</p> <p>When Advance Time Stamp Feature or IPC Full Offload is not selected, this bit indicates Rx MAC Address status. When set, this bit indicates that the Rx MAC Address registers value (1 to 31) matched the frame's DA field. When reset, this bit indicates that the Rx MAC Address Register 0 value matched the DA field.</p>

**Table 7-23 Receive Descriptor Fields 1 (RDES1)**

Bit	Description
31	<p><b>DIC: Disable Interrupt on Completion</b></p> <p>When set, this bit prevents setting the Status Register's RI bit (CSR5[6]) for the received frame ending in the buffer indicated by this descriptor. This, in turn, disables the assertion of the interrupt to Host due to RI for that frame.</p>
30:29	<b>Reserved</b>
28:16	<p><b>RBS2: Receive Buffer 2 Size</b></p> <p>These bits indicate the second data buffer size, in bytes. The buffer size must be a multiple of 4, 8, or 16, depending on the bus widths (32, 64, or 128, respectively), even if the value of RDES3 (buffer2 address pointer) is not aligned to bus width. If the buffer size is not an appropriate multiple of 4, 8, or 16, the resulting behavior is undefined. This field is not valid if RDES1[14] is set. See “<a href="#">Buffer Size Calculations</a>” on page <a href="#">67</a> for further details on calculating buffer sizes.</p>
15	<p><b>RER: Receive End of Ring</b></p> <p>When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring.</p>
14	<p><b>RCH: Second Address Chained</b></p> <p>When set, this bit indicates that the second address in the descriptor is the Next Descriptor address rather than the second buffer address. When this bit is set, RBS2 (RDES1[28:16]) is a “don’t care” value. RDES1[15] takes precedence over RDES1[14].</p>
13	<b>Reserved</b>
12:0	<p><b>RBS1: Receive Buffer 1 Size</b></p> <p>Indicates the first data buffer size in bytes. The buffer size must be a multiple of 4, 8, or 16, depending upon the bus widths (32, 64, or 128), even if the value of RDES2 (buffer1 address pointer) is not aligned. When the buffer size is not a multiple of 4, 8, or 16, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or next descriptor depending on the value of RCH (Bit 14). See “<a href="#">Buffer Size Calculations</a>” on page <a href="#">67</a> for further details on calculating buffer sizes.</p>

**Table 7-24 Receive Descriptor Fields 2 (RDES2)**

Bit	Description
31:0	Buffer 1 Address Pointer These bits indicate the physical address of Buffer 1. There are no limitations on the buffer address alignment except for the following condition: The DMA uses the configured value for its address generation when the RDES2 value is used to store the start of frame. Note that the DMA performs a write operation with the RDES2[3/2/1:0] bits as 0 during the transfer of the start of frame but the frame data is shifted as per the actual Buffer address pointer. The DMA ignores RDES2[3/2/1:0] (corresponding to bus width of 128/64/32) if the address pointer is to a buffer where the middle or last part of the frame is stored. See " <a href="#">Host Data Buffer Alignment</a> " on page <a href="#">66</a> for further details on buffer address alignment.

**Table 7-25 Receive Descriptor Fields 3 (RDES3)**

Bit	Description
31:0	Buffer 2 Address Pointer (Next Descriptor Address) These bits indicate the physical address of Buffer 2 when a descriptor ring structure is used. If the Second Address Chained (RDES1[24]) bit is set, this address contains the pointer to the physical memory where the Next Descriptor is present. If RDES1[24] is set, the buffer (Next Descriptor) address pointer must be bus width-aligned (RDES3[3, 2, or 1:0] = 0, corresponding to a bus width of 128, 64, or 32. LSBs are ignored internally.) However, when RDES1[24] is reset, there are no limitations on the RDES3 value, except for the following condition: The DMA uses the configured value for its buffer address generation when the RDES3 value is used to store the start of frame. The DMA ignores RDES3[3, 2, or 1:0] (corresponding to a bus width of 128, 64, or 32) if the address pointer is to a buffer where the middle or last part of the frame is stored.

The extended status written is as shown in [Table 7-26](#). The extended status is written only when there is status related to IPC or Time Stamp available. The availability of extended status is indicated by bit-0 of RDES0. This status is available only when Advance Time Stamp or IPC Full Offload feature is selected.

**Table 7-26 Receive Descriptor Fields 4 (RDES4)**

Bit	Description
31:14	Reserved
13	PTP Version When set, indicates that the received PTP message is having the IEEE 1588 version 2 format. When reset, it has the version 1 format. This is valid only if the message type is non-zero. This bit is available only if Advance Time Stamp feature is selected else it is reserved
12	PTP Frame Type When set, this bit that the PTP message is sent directly over Ethernet. When this bit is not set and the message type is non-zero, it indicates that the PTP message is sent over UDP-IPv4 or UDP-IPv6. The information on IPv4 or IPv6 can be obtained from bits 6 and 7. This bit is available only if Advanced Time Stamp feature is selected

**Table 7-26 Receive Descriptor Fields 4 (RDES4)**

Bit	Description
11:8	<p><b>Message Type</b>  These bits are encoded to give the type of the message received.</p> <ul style="list-style-type: none"> <li>• 0000: No PTP message received</li> <li>• 0001: SYNC (all clock types)</li> <li>• 0010: Follow_Up (all clock types)</li> <li>• 0011: Delay_Req (all clock types)</li> <li>• 0100: Delay_Resp (all clock types)</li> <li>• 0101: Pdelay_Req (in peer-to-peer transparent clock) or Announce (in ordinary or boundary clock)</li> <li>• 0110: Pdelay_Resp (in peer-to-peer transparent clock) or Management (in ordinary or boundary clock)</li> <li>• 0111: Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock) or Signaling (for ordinary or boundary clock)</li> <li>• 1xxx - Reserved</li> </ul> <p>These bits are valid only if Advance Time Stamp feature is selected</p>
7	<p><b>IPv6 Packet Received</b>  When set, this bit indicates that the received packet is an IPv6 packet.</p>
6	<p><b>IPv4 Packet Received</b>  When set, this bit indicates that the received packet is an IPv4 packet.</p>
5	<p><b>IP Checksum Bypassed</b>  When set, this bit indicates that the checksum offload engine is bypassed.</p>
4	<p><b>IP Payload Error</b>  When set, this bit indicates that the 16-bit IP payload checksum (that is, the TCP, UDP, or ICMP checksum) that the core calculated does not match the corresponding checksum field in the received segment. It is also set when the TCP, UDP, or ICMP segment length does not match the payload length value in the IP Header field.</p>
3	<p><b>IP Header Error</b>  When set, this bit indicates either that the 16-bit IPv4 header checksum calculated by the core does not match the received checksum bytes, or that the IP datagram version is not consistent with the Ethernet Type value.</p>
2:0	<p><b>IP Payload Type</b>  These bits indicate the type of payload encapsulated in the IP datagram processed by the Receive Checksum Offload Engine (COE). The COE also sets these bits to 2'b00 if it does not process the IP datagram's payload due to an IP header error or fragmented IP.</p> <ul style="list-style-type: none"> <li>• 3'b000: Unknown or did not process IP payload</li> <li>• 3'b001: UDP</li> <li>• 3'b010: TCP</li> <li>• 3'b011: ICMP</li> <li>• 3'b1xx: Reserved</li> </ul>

RDES6 and RDES7 contain the snapshot of the time-stamp. The availability of the snapshot of the time-stamp in RDES6 and RDES7 is indicated by bit-7 in the RDES0 descriptor. The contents of RDES6 and RDES7 are identified in [Table 7-27](#) and [Table 7-28](#), respectively.

**Table 7-27 Receive Descriptor Fields 6 (RDES6)**

Bit	Description
31:0	<b>RTSL: Receive Frame Time Stamp Low</b> This field is updated by DMA with the least significant 32 bits of the time stamp captured for the corresponding receive frame. This field is updated by DMA only for the last descriptor of the receive frame which is indicated by Last Descriptor status bit (RDES0[8]).

**Table 7-28 Receive Descriptor Fields 7 (RDES7)**

Bit	Description
31:0	<b>RTSH: Receive Frame Time Stamp High</b> This field is updated by DMA with the most significant 32 bits of the time stamp captured for the corresponding receive frame. This field is updated by DMA only for the last descriptor of the receive frame which is indicated by Last Descriptor status bit (RDES0[8]).

The above description holds true for all descriptor formats depicted in [Figures 7-3, 7-5, 7-7, and 7-9](#). However, when the core operates in 64- or 128-bit and with a reverse-endian descriptor ([Figures 7-4, 7-6, 7-8, and 7-10](#)), RDES6 is updated with RTSH and RDES7 is updated with RTSL. This ensures that the 64-bit time stamp can be processed as-is without any swapping between the two 32-bit words.



## 8

# Implementation Guidelines

---

## 8.1 Clocks With GMII/MII

The clocking scheme for the GMAC-AHB is shown in [Figure 8-1](#). The GMAC-AHB uses three clock inputs for normal operation with the GMII/MII interface:

- ❖ One clock (`clk_tx_i`) for transmission functions
- ❖ One clock (`clk_rx_i`) for reception functions
- ❖ One application clock (`hclk_i`)

All clocks except the Application clock are generated from an external PHY. The Application clock is used for the control interface of the GMAC-AHB. Transmit clock `clk_tx_i` is used for the transmission function of the GMAC-AHB. Similarly, receive clock `clk_rx_i` is used for the receive function of the GMAC-AHB. The transfer of data from/to the application clock domain to the Transmit and Receive clock domains is performed in the MTL module.

Clock `clk_tx_i` switches between the MII transmit clock input from an external PHY (MII mode) and the 125-MHz clock input from an oscillator (GMII mode), based on the `mac_portselect_o` signal from the GMAC-AHB. The external PHY outputs a 25-MHz or 2.5-MHz transmit clock on `MII_CLK` when it operates at 100 Mbps or 10 Mbps, respectively.

The frequency of clock `clk_rx_i` is same as receive clock `GMII/MII_RX_CLK` generated by the external PHY. The PHY outputs a 125-MHz, 25-MHz, or 2.5-MHz receive clock on `GMII/MII_RX_CLK` when it operates at 1000 Mbps, 100 Mbps, or 10 Mbps, respectively.

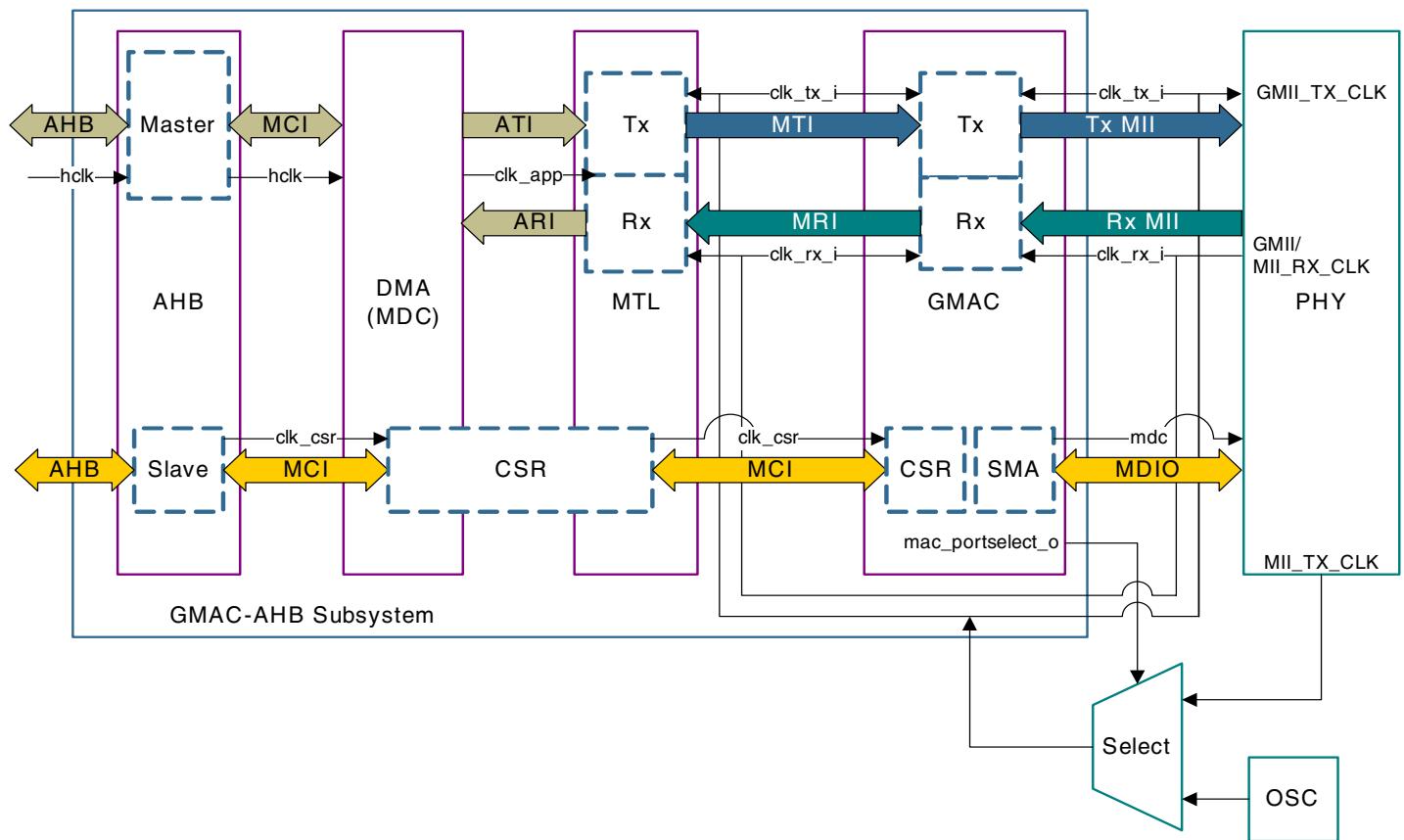


### Caution

In all PHY interfaces explained in [Sections 8.1 to 8.7](#) (except MII, SGMII, SMII, and RMII), your chip containing the MAC core must drive the Transmit clock towards the PHY. All the transmit data and control signals output from the core must be timed with respect to this output clock, as specified in the respective PHY interface specifications (See the IEEE 802.3, SGMII, RMII, and RGMII/RTBI specifications).

For MII and SGMII, the transmit data and control signals output from the core must be timed with respect to either the transmit clock input from the PHY chip (MII) or the SerDes (SGMII).

For RMII/SMII, you can either connect the Oscillator clock directly to both the MAC and PHY or drive an RMII/SMII clock towards the PHY, according to the recommendations in the RMII/SMII specification.

**Figure 8-1 GMAC-AHB Clocking Scheme**

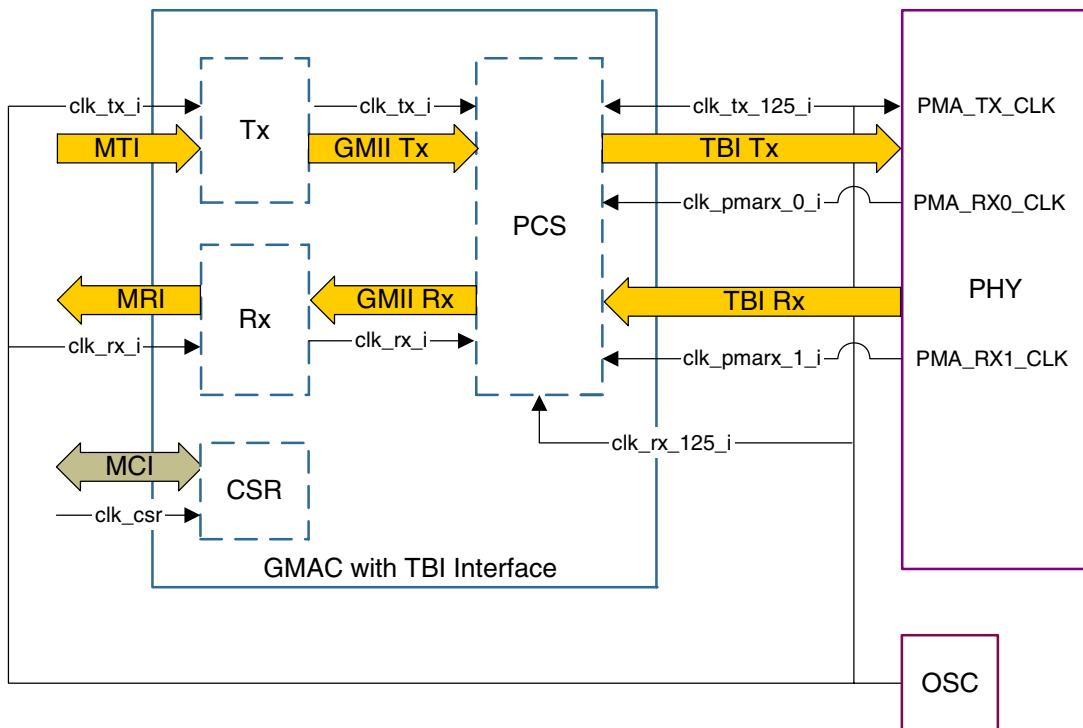
## 8.2 Clocks With TBI

The clocking scheme used for the GMAC-CORE in TBI mode is shown in [Figure 8-2](#). The clocking strategy is the same for other GMAC-UNIV configurations such as GMAC-AHB, GMAC-MTL, etc. The GMAC-CORE uses four clock inputs for normal operation:

- ❖ One clock (`clk_tx_i`) for transmission functions
- ❖ Two clocks (`clk_pmarx0_i` and `clk_pmarx1_i`) for reception functions
- ❖ One application clock (`clk_csr_i`)

Note that you should connect the same `clk_tx_i` to the `clk_tx_125_i`, `clk_rx_i`, and `clk_rx_125_i` ports of the GMAC core in TBI mode. The `clk_tx_125_180_i` port should be connected to the invert of `clk_tx_i`.

The GMAC-CORE outputs the transmit code group to the external PHY synchronous to the `clk_tx_i` (125-MHz clock). The GMAC-CORE internally transfers the receive code group from the `clk_pmarx_x_i` clock domain to the `clk_rx_i` (same as `clk_tx_i`) clock domain.

**Figure 8-2 GMAC-CORE Clocking Scheme (TBI Mode)**

### 8.3 Clocks With SGMII

The clocking scheme used for the GMAC-CORE in SGMII mode is shown in Figure 8-3. The clocking strategy is the same for other GMAC-UNIV configurations such as GMAC-AHB, GMAC-MTL, etc. The GMAC-CORE uses five clock inputs for normal operation:

- ❖ One clock (clk\_tx\_125\_i) for transmission functions with the RAL, PCS, and SerDes (Note that the clk\_tx\_180\_i port should be connected to the invert of clk\_tx\_125\_i.)
- ❖ One clock (clk\_rx\_125\_i) for reception functions with the RAL, PCS, and SerDes
- ❖ One clock (clk\_tx\_i) for DWC Ether MAC 10/100/1000 Universal transmission functions
- ❖ One clock (clk\_rx\_i) for DWC Ether MAC 10/100/1000 Universal reception functions
- ❖ One application clock (clk\_csr\_i)

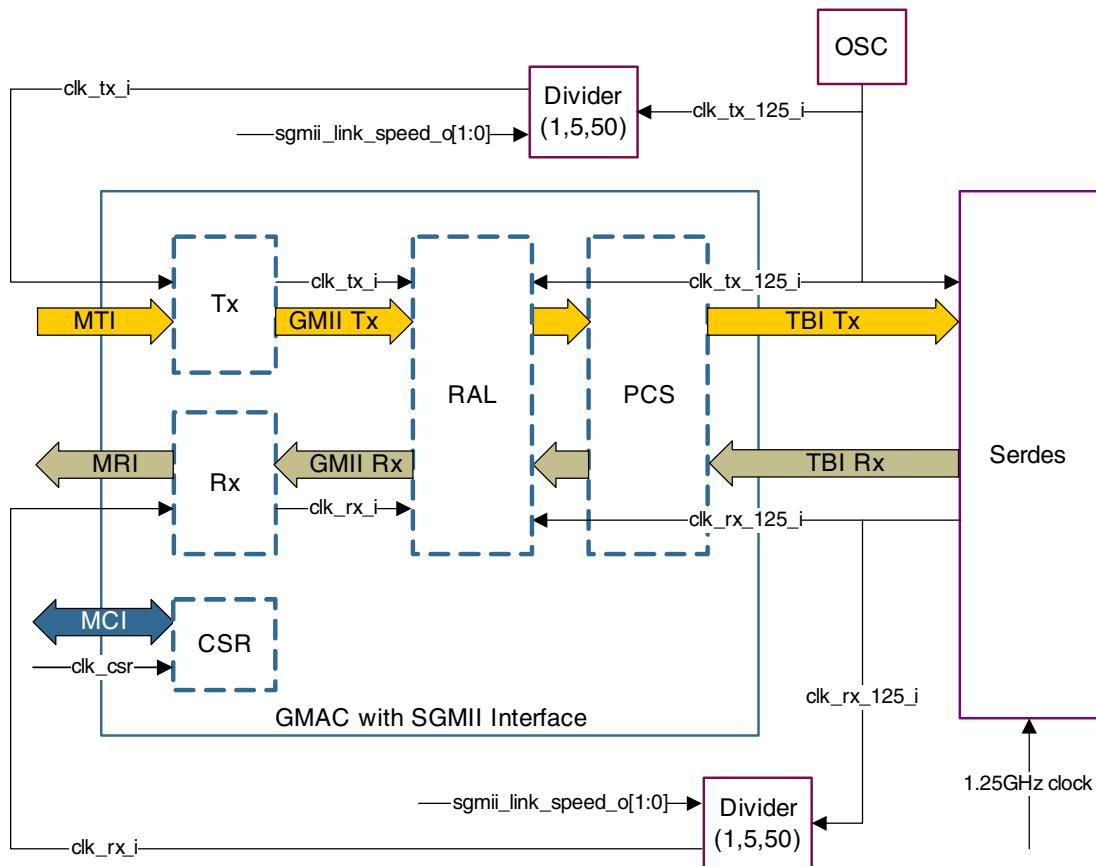
The GMAC-CORE outputs the transmit code group to the RAL synchronous to clk\_tx\_i. Similarly, the GMAC-CORE inputs the receive code group from the RAL, synchronous to clk\_rx\_i. The PCS layer runs on a 125-MHz clock while the GMII interface may run on a 2.5-MHz, 25-MHz, or 125-MHz clock. Therefore the clk\_tx\_i clock is derived from clock clk\_tx\_125\_i, with a frequency either equal to the frequency of clk\_tx\_125\_i, one fifth the frequency, or one fiftieth the frequency of clk\_tx\_125\_i, as determined by signal sgmii\_link\_speed\_o. (Refer to “Serial Gigabit Media Independent Interface” on page 171 for a functional description of the SGMII interface.) Similarly, clk\_rx\_i is derived from clk\_rx\_125\_i by a clock divider circuit that changes the frequency with the link speed. You can also choose the output signal mac\_speed\_o (instead of sgmii\_link\_speed\_o) as the control signal for the Transmit clock divider according to your system requirements.

Because clk\_tx\_i and clk\_rx\_i are derived synchronously (divided clock) from the corresponding clk\_tx\_125\_i and clk\_rx\_125\_i, it is easier to meet the static timing of signals crossing the clock domains if

the derived clock's rising edge is synchronous to the master clocks' falling edge. Hence, the timing budget for such signals is half the clock width of the master clock.

If the timing delays are not met, the clock phases between the master and derived clock can be shifted, with the signals in the `clk_tx_i` domain sampled in the `clk_tx_125_i` domain in the transmit path and the signals in `clk_rx_125_i` are sampled in the `clk_rx_i` domain. Thus, `clk_tx_i` must lead `clk_tx_125_i`, while `clk_rx_i` must lag `clk_rx_125_i`.

**Figure 8-3 GMAC Clocking Scheme (SGMII Mode)**



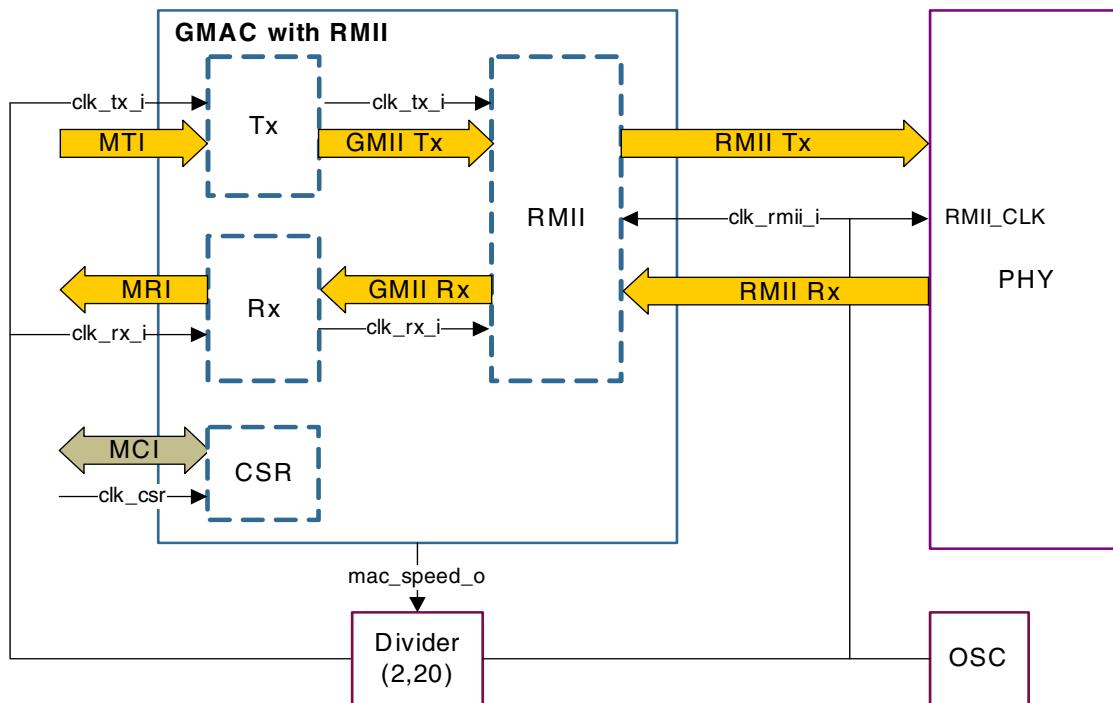
## 8.4 Clocks With RMII

The clocking scheme used for the GMAC-CORE in RMII mode is shown in [Figure 8-4](#). The clocking strategy is the same for other GMAC-UNIV configurations, such as GMAC-AHB, GMAC-MTL, and so forth. The GMAC-CORE uses three clock inputs for normal operation:

- ❖ One clock (`clk_rx_i`) for transmission and reception functions
- ❖ One clock (`clk_rmii_i`) for RMII
- ❖ One application clock (`clk_csr_i`)

You should connect the same `clk_rx_i` to the `clk_tx_i` ports of the GMAC core in RMII mode. The `clk_rx_i` should be synchronously derived from the `clk_rmii_i`, by dividing it by 20 for 10-Mbps operation or by 2 for 100-Mbps operation. You can use either the `mac_speed_i` or `mac_speed_o[0]` signals as the control signal to the clock divider.

**Figure 8-4 Clocking Scheme (RMII Mode)**



Because `clk_tx_i` (which is the same as `clk_rx_i`) is synchronously derived (divided) from the corresponding `clk_rmii_i`, it is easier to meet the static timing of signals crossing the clock domains if the derived clock's rising edge is synchronous to the RMII clock's falling edge. You therefore get half the RMII clock's clock width as the timing budget for such signals.

If the timing delays are not met, the clock phases between the master and derived clock can be shifted (during clock tree synthesis). Because the signals in the `clk_tx_i` domain are sampled in the `clk_rmii_i` domain in the transmit path, while the signals in `clk_rmii_i` are sampled in the `clk_rx_i` domain, `clk_tx_i` must lead `clk_rmii_i` and `clk_rx_i` must lag `clk_rmii_i`.

## 8.5 Clocks With SMII

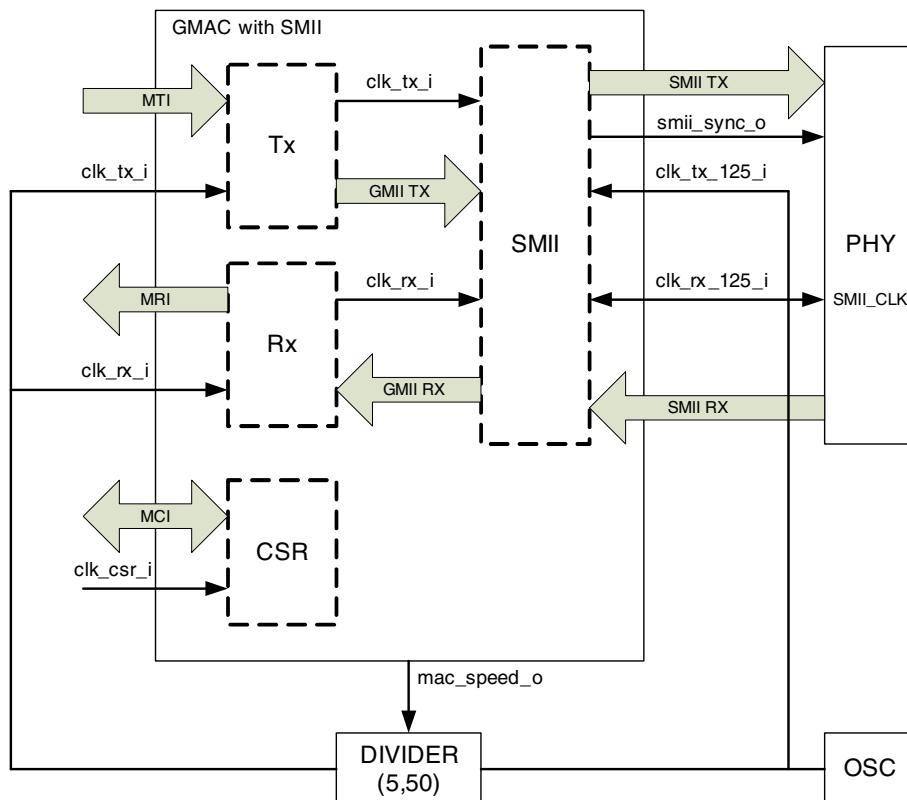
### 8.5.1 Non Source Synchronous Mode

The clocking scheme used for the GMAC-CORE in SMII Non Source Synchronous mode is shown in [Figure 8-5](#). The clocking strategy is the same for other GMAC-UNIV configurations, such as GMAC-AHB, GMAC-MTL, and so forth. The GMAC-CORE uses three clock inputs for normal operation:

- ❖ One clock (`clk_tx_i`) for transmission and reception functions
- ❖ One clock (`clk_tx_125_i`) for SMII
- ❖ One application clock (`clk_csr_i`)

You should connect the same `clk_tx_i` to the `clk_rx_i` ports of the GMAC core in SMII mode. The `clk_tx_i` should be synchronously derived from the `clk_tx_125_i`, by dividing it by 50 for 10 Mbps operation or by 5 for 100 Mbps operation. You can use either the `mac_speed_i` or `mac_speed_o[0]` signals as the control signal to the clock divider. The global sync signal `smii_sync_o` will be provided by the SMII block.

**Figure 8-5 Non Source Synchronous Mode Clocking Scheme**



Because `clk_tx_i` (which is the same as `clk_rx_i`) is synchronously derived (divided) from the corresponding `clk_tx_125_i`, it is easier to meet the static timing of signals crossing the clock domains if the derived clock's rising edge is synchronous to the SMII clock's falling edge. You therefore get half the SMII clock's clock width as the timing budget for such signals.

If the timing delays are not met, the clock phases between the master and derived clock can be shifted (during clock tree synthesis). Because the signals in the `clk_tx_i` domain are sampled in the `clk_tx_125_i` domain in the transmit path, while the signals in `clk_rx_125_i` are sampled in the `clk_rx_i` domain, `clk_tx_i` must lead `clk_tx_125_i` and `clk_rx_i` must lag `clk_rx_125_i`.

### 8.5.2 Source Synchronous Mode

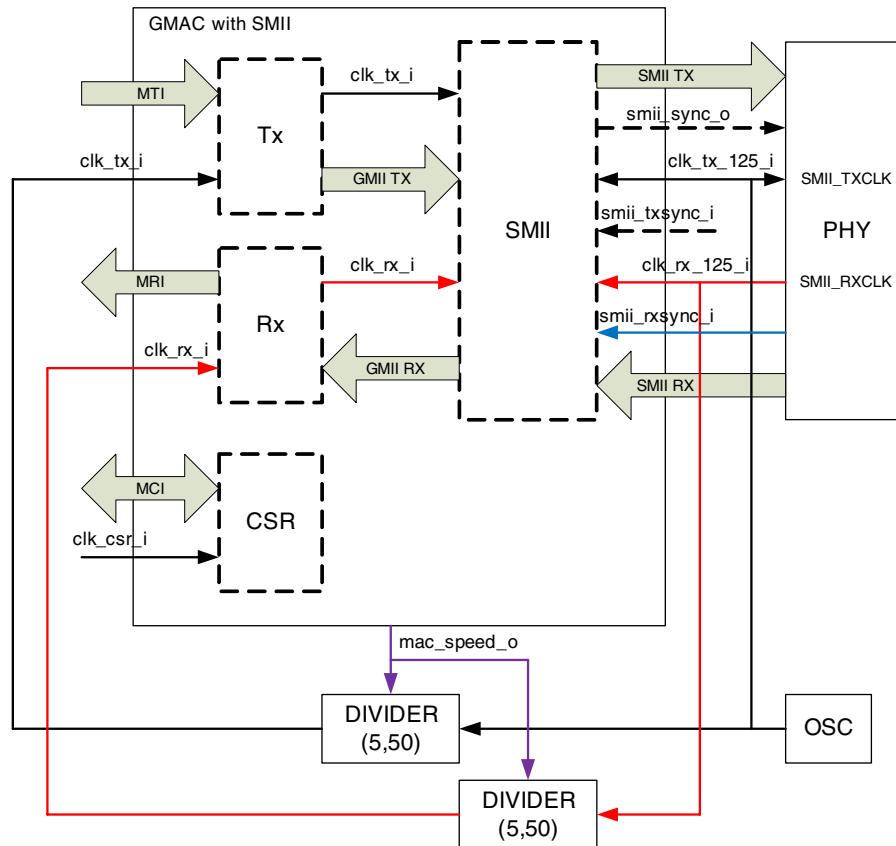
The clocking scheme used for the GMAC-CORE in SMII Source Synchronous mode is shown in [Figure 8-6](#). The clocking strategy is the same for other GMAC-UNIV configurations, such as GMAC-AHB, GMAC-MTL, and so forth. The GMAC-CORE uses five clock inputs for normal operation:

- ❖ One clock (`clk_tx_i`) for transmission function
- ❖ One clock (`clk_rx_i`) for reception function
- ❖ One clock (`clk_rx_125_i`) for SMII receive path
- ❖ One clock (`clk_tx_125_i`) for SMII transmit path
- ❖ One application clock (`clk_csr_i`)

The `clk_rx_i` should be synchronously derived from the `clk_rx_125_i`, by dividing it by 50 for 10 Mbps operation or by 5 for 100 Mbps operation. Similarly, the `clk_tx_i` should be synchronously derived from the `clk_tx_125_i`, by dividing it by 50 for 10 Mbps operation or by 5 for 100 Mbps operation. You can use either the `mac_speed_i` or `mac_speed_o[0]` signals as the control signal to the clock dividers. The TXSYNC signal will be provided by the SMII block if `SSSMII_TXSYNC_IN` configuration parameter is not selected.

However, if `SSSMII_TXSYNC_IN` configuration parameter is selected, the SMII transmit block will transmit synchronized to the `smii_txsync_i` input. The `smii_rxsync_i` needs to be driven by the transmitting source.

**Figure 8-6 Source Synchronous Mode Clocking Scheme**



Because `clk_tx_i` (and `clk_rx_i`) is synchronously derived (divided) from the corresponding `clk_tx_125_i` (or `clk_rx_125_i`), it is easier to meet the static timing of signals crossing the clock domains if the derived clock's rising edge is synchronous to the SMII clock's falling edge. You therefore get half the SMII clock's clock width as the timing budget for such signals.

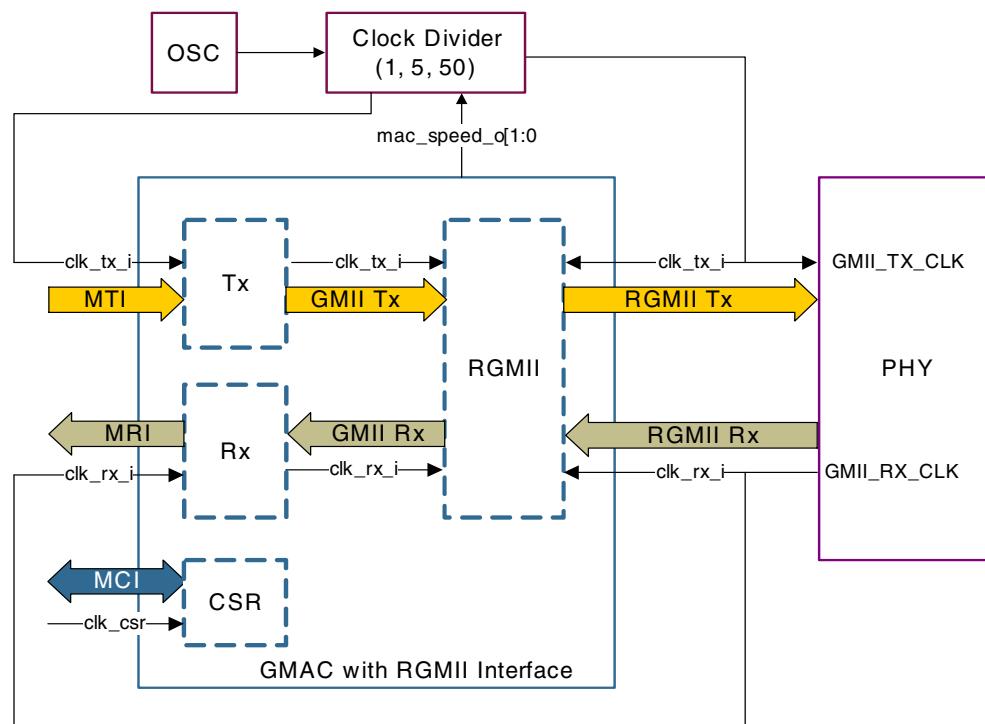
If the timing delays are not met, the clock phases between the master and derived clock can be shifted (during clock tree synthesis). Because the signals in the `clk_tx_i` domain are sampled in the `clk_tx_125_i` domain in the transmit path, while the signals in `clk_rx_125_i` are sampled in the `clk_rx_i` domain, `clk_tx_i` must lead `clk_tx_125_i` and `clk_rx_i` must lag `clk_rx_125_i`.

## 8.6 Clocks With RGMII

The clocking scheme used for the GMAC-CORE in RGMII mode is shown in [Figure 8-7](#). The clocking strategy is the same for other GMAC-UNIV configurations such as GMAC-AHB, GMAC-MTL, etc. The GMAC-CORE uses three clock inputs for normal operation:

- ❖ One clock (`clk_tx_i`) for transmission function. The `clk_tx_180_i` port should be connected to the invert of `clk_tx_i` in RGMII mode.
- ❖ One clock (`clk_rx_i`) for reception function. The `clk_rx_180_i` port should be connected to the invert of `clk_rx_i`.
- ❖ One application clock (`clk_csr_i`)

**Figure 8-7 Clocking Scheme (RGMII Mode)**



The RGMII specifications state that the transmit clock (`clk_tx_i`) frequency must be 2.5, 25, or 125 MHz, respectively, for 10-, 100-, or 1000-Mbps operation. Hence the 125-MHz oscillator must be divided dynamically based on the operating speed. The signal output from the GMAC-UNIV (`mac_speed_o[1:0]`) controls the clock divider as follows:

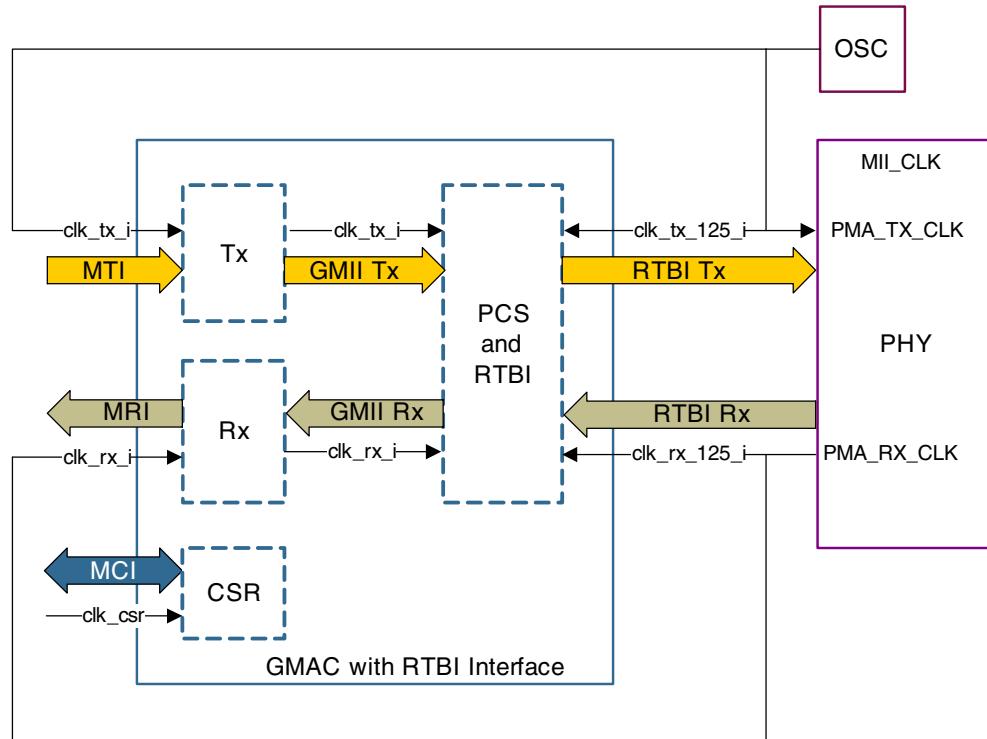
<code>mac_speed_o[1,0]</code>	Clock Divider
2'b0x	Divide by 1
2'b11	Divide by 5
2'b10	Divide by 50

## 8.7 Clocks With RTBI

The clocking scheme used for the GMAC-CORE in RTBI mode is shown in [Figure 8-8](#). The clocking strategy is the same for other GMAC-UNIV configurations such as GMAC-AHB, GMAC-MTL, and so on. The GMAC-CORE uses three clock inputs for normal operation:

- ❖ One clock (`clk_tx_i`) for transmission function. The `clk_tx_125_180_i` port should be connected to the inverse of `clk_tx_i` in RTBI mode, while the `clk_tx_125_i` port should be the same as `clk_tx_i`.
- ❖ One clock (`clk_rx_i`) for reception function. The `clk_rx_180_i` port should be connected to the inverse of `clk_rx_i`, while the `clk_rx_125_i` port should be the same as `clk_rx_i`.
- ❖ One application clock (`clk_csr_i`)

**Figure 8-8 Clocking Scheme (RTBI Mode)**



## 8.8 Host (System Interface) Clock

The GMAC-UNIV has different clock names for different configuration as listed below for the transfer of data to and from the GMAC to the host. Even though the names are different, the required properties and frequency ranges for all the clocks are same. In addition to this, you have an option to have a separate clock (clk\_csr\_i) for accessing the CSR through the slave port.

GMAC-DMA, GMAC-MTL	: clk_app_i
GMAC-AHB	: hclk_i
GMAC-AXI	: aclk_i

The minimum and maximum clock frequency are specified between the range 20MHz-300 MHz for the host clock. This is not due to any functional limitation but due to the fact that we verify it & synthesize the design for this range. You can go for a higher frequency as your technology permits.

Another factor to keep in mind is that the host clock (or clk\_csr\_i when present) is used to generate the MDC clock for the SMA interface. Due to the requirement of IEEE 802.3 that the MDC clock should have a maximum frequency of 2.5 MHz, your clock frequency may get limited by the clock-divider options provided. But if your system (or PHY chip) permits a higher frequency for MDC clock than specified by 802.3, then you can ignore this constraint for the maximum frequency possible.

When the RMON (MMC) counters are present and the core is operating in gigabit speed, then the CSR clock should have a minimum frequency of 25 MHz for proper collection of statistics of all consecutive small-sized (or runt) frames.

Note: You can dynamically change the frequency of host clock when the GMAC is active (without any software intervention) to save power consumption of your chip. This is tested in our regression run in our simulation environment between the frequency ranges specified (25Mhz -300 MHz).

## 8.9 Resets

When the GMAC-UNIV is configured for DMA (with or without AHB/ AXI interface), internal reset generation logic is also automatically instantiated. This reset logic takes in the hreset\_n for GMAC-AHB (pwr\_on\_rst\_n for GMAC-DMA or areset\_n in GMAC-AXI) input signal, and generates resets for all other clock domains internally. It also manages reset assertion when the host triggers the SWR bit of DMA Register 0. Internal reset generation (for other clock domains) can be bypassed by setting the test\_mode signal input high. In this mode, the input reset (hreset\_n or pwr\_on\_rst\_n) is directly used in all clock domains. This mode is normally used during scan tests only.

For GMAC-CORE and GMAC-MTL configurations, no reset-generation logic is present in the core. you must therefore input separate resets corresponding to each clock input. These resets must always be deasserted synchronously with their corresponding clock edges. For proper core initialization, all reset inputs must be asserted together for at least 3 cycles of the minimum-frequency clock. Otherwise, the clock domain transfer logic is not properly initialized, with possible corruption of the first frame after such a reset.

## 8.10 SMA to PHY Interface

As defined in IEEE 802.3, Clause 22, the MAC station management interface to the external PHY consists of two signals:

- ❖ MDIO: Bidirectional data
- ❖ MDC: Clock input to the PHYs from the MAC

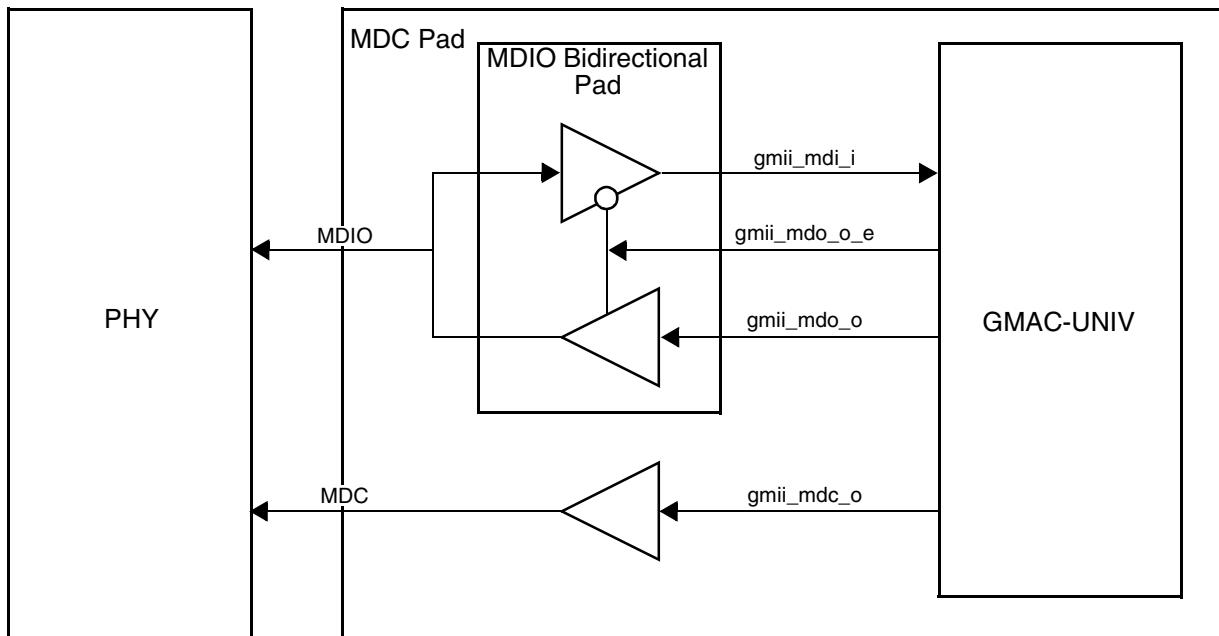
These signals are detailed in IEEE 802.3, Clause 22.

The GMAC-UNIV core supports this interface with the following four signals (to be connected to the ASIC's MDIO and MDC pads):

- ❖ gmac\_mdi\_i: input MDIO data signal (from the PHY to the MAC)
- ❖ gmac\_mdo\_o: output MDIO data signal (from the MAC to the PHY)
- ❖ gmac\_mdo\_e: a control signal to the bidirectional MDIO pad, which signal controls the pad's input/output directionality. When asserted, this signal indicates that the MDIO is in Output mode.
- ❖ gmac\_mdc\_o: output MDC clock signal (from the MAC to PHY).

A connectivity example is shown in [Figure 8-9](#)

**Figure 8-9 SMA to PHY Connectivity**



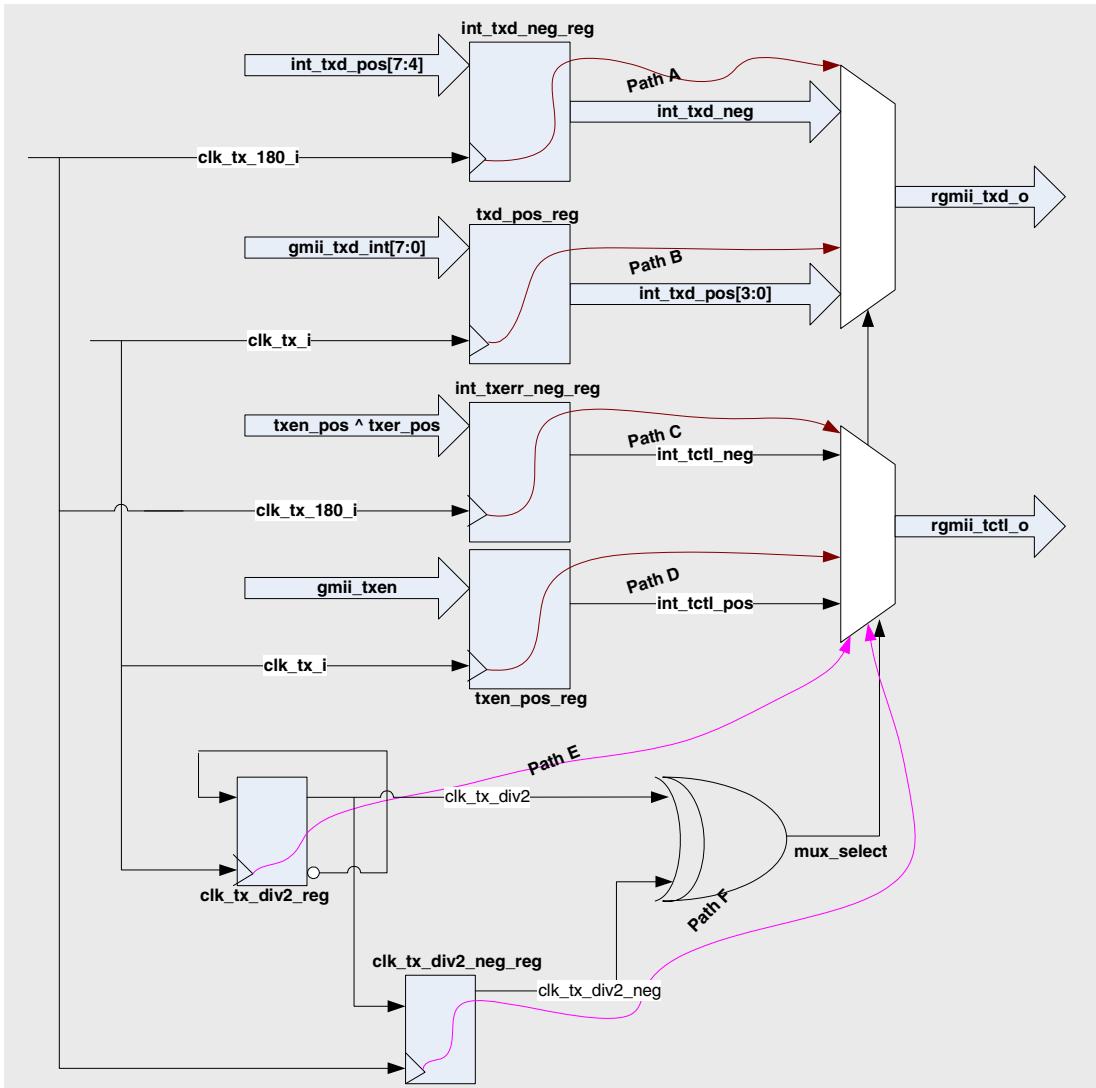
## 8.11 Timing Guidelines for Dual Data Rate RGMII/RTBI Outputs



Though this section refers only to the RGMII PHY, it also applies when the core is RTBI-PHY configured.

When the RGMII PHY interface is configured, the data and control signals are valid for both clock edges. The transmit output data rgmii\_txd\_o and control rgmii\_tctl\_o can transition at both clock edges of clk\_tx\_i. This logic is implemented in GMAC using two clocks (clk\_tx\_i and clk\_tx\_180\_i, which is 180° out of phase with clk\_tx\_i) as shown in [Figure 8-10](#).

**Figure 8-10** Dual Data Rate Output Paths



This logic can be found in the DWC\_gmac\_rgmii\_gmrt.v file in the <config>/src/rgmii/ directory. Paths A-D in [Figure 8-10](#) are considered the data path delays (clock-to-output delay for the registers, plus the net delay from the register output to the multiplexer's input). Similarly, paths E and F are considered the control path delay for the MUX Select signal. To ensure that no glitches are generated in the output data, the data signals at the input of the MUX (paths A-D) must be stable before the MUX select signal (paths E-F) is

asserted. The control path E-F timing delays must be greater than the data path delays. The following paragraphs give a guideline on how to ensure such delays during implementation.

The data path timing report in [Example 8-1](#) shows the total delay for the data path to be 0.57 ns. This timing report can be obtained by giving the following command in DC shell after completion of synthesis.

```
report_timing -from DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/
txd_pos_reg*/CK
```



- Check your technology library for the exact clock port name.
- You can obtain a similar timing report for int\_txd\_neg\_reg.

### Example 8-1 Data Path Timing Report

Startpoint: DWC\_gmac\_inst/DWC\_gmac\_rgmii\_inst/DWC\_gmac\_rgmii\_gmrt\_inst/txd\_pos\_reg[3]  
(rising edge-triggered flip-flop clocked by clk\_tx\_i)

Endpoint: phy\_txd\_o[3] (output port clocked by clk\_tx\_180\_i)

Path Group: clk\_tx\_i\_outputs

Path Type: max

Des/Clust/Port	Wire Load Model	Library
DWC_gmac_top	tsmc090_wl10	slow
Point	Incr	Path
clock clk_tx_i (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/txd_pos_reg[3]/CK (DFFRQX2)	0.00	0.00 r
DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/txd_pos_reg[3]/Q (DFFRQX2)	0.36	0.36 f
DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/U34/Y (MX2X1)	0.21	0.57 f
DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/rgmii_txd_o[3] (DWC_gmac_rgmii)	0.00	0.57 f
DWC_gmac_inst/DWC_gmac_rgmii_inst/rgmii_txd_o[3] (DWC_gmac_rgmii)	0.00	0.57 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/rgmii_txd_i[3] (DWC_gmac_phy_txmux)	0.00	0.57 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/U9/Y (MX2X1)	0.20	0.76 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/phy_txd_o[3] (DWC_gmac_phy_txmux)	0.00	0.76 f
DWC_gmac_inst/phy_txd_o[3] (DWC_gmac)	0.00	0.76 f
phy_txd_o[3] (out)	0.00	0.76 f
data arrival time	0.76	
clock clk_tx_180_i (rise edge)	4.00	4.00
clock network delay (ideal)	0.00	4.00
output external delay	-1.60	2.40
data required time	2.40	
-----		
data required time	2.40	
data arrival time	-0.76	

slack (MET)	1.64
-------------	------

The Multiplexer Select signal shows a 1.08-ns path delay. The timing report below can be obtained by giving the following command in DC shell.

```
report_timing -from DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/
clk_tx_div2_reg*/CK
```

 **Note** You can obtain a similar report for clk\_tx\_div2\_neg\_reg.

### Example 8-2 Multiplexer Select Signal Timing Report

Startpoint: DWC\_gmac\_inst/DWC\_gmac\_rgmii\_inst/DWC\_gmac\_rgmii\_gmrt\_inst/clk\_tx\_div2\_reg  
(rising edge-triggered flip-flop clocked by clk\_tx\_i)  
Endpoint: phy\_txd\_o[0] (output port clocked by clk\_tx\_180\_i)  
Path Group: clk\_tx\_i\_outputs  
Path Type: max

Des/Clust/Port	Wire Load Model	Library
<hr/>		
DWC_gmac_top	tsmc090_wl10	slow
<hr/>		
Point	Incr	Path
clock clk_tx_i (rise edge)	0.00	0.00
clock network delay (ideal)	0.00	0.00
DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/clk_tx_div2_reg/CK (DFFRQX2)	0.00	0.00 r
DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/clk_tx_div2_reg/Q (DFFRQX2)	0.36	0.36 f
DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/U26/Y (CLKXOR2X2)	0.47	0.83 f
DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/U36/Y (MX2X1)	0.26	1.08 f
DWC_gmac_inst/DWC_gmac_rgmii_inst/DWC_gmac_rgmii_gmrt_inst/rgmii_txd_o[0] (DWC_gmac_rgmii)	0.00	1.08 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/rgmii_txd_i[0] (DWC_gmac_phy_txmux)	0.00	1.08 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/U12/Y (MX2X1)	0.20	1.28 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/phy_txd_o[0] (DWC_gmac_phy_txmux)	0.00	1.28 f
DWC_gmac_inst/phy_txd_o[0] (DWC_gmac)	0.00	1.28 f
phy_txd_o[0] (out)	0.00	1.28 f
data arrival time	1.28	
clock clk_tx_180_i (rise edge)	4.00	4.00
clock network delay (ideal)	0.00	4.00
output external delay	-1.60	2.40
data required time	2.40	

data required time	2.40
data arrival time	-1.28
slack (MET)	1.12

For proper circuit operation, the multiplexer select path delay (see paths E and F in [Figure 8-10](#)) must be greater than the data path delay (Paths A-D). In the Multiplexer Select Signal Timing Report, this constraint is met because the net delay is obtained using a statistical wire load model. In reality, the net delay for the data path can be greater than the delay for the Multiplexer Select signal because of routing delays. Based on the preliminary routing delays, you must set a proper minimum delay (value greater than the data arrival time of the data paths, that is, paths A-D) to the Multiplexer Select signal path using the sample constraint mentioned below.

```
set_min_delay <delay> -from {DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmiigmrt_
inst/ clk_tx_div2_reg/CK DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmiigmrt_
inst/ clk_tx_div2_neg_reg/CK} -to [find port phy_txd_o]
```



- Rerun the synthesis with Synopsys Design Compiler, with compile options.
- Because the valid endpoint for a constraint can only be the input of a sequential cell or a port, you must ensure that the minimum delay value set takes care of the delay from output of the MUX to the port.

A synthesis pragma has been added to ensure that a multiplexer is inferred for the rgmii\_txd\_o and rgmii\_tctl\_o outputs. This code is as follows:

```
always @(*)
begin
    case (mux_select) // synopsys infer_mux
        1'b1: begin
            rgmii_txd_o = int_txd_pos;
            rgmii_tctl_o = int_tctl_pos;
        end
        1'b0: begin
            rgmii_txd_o = int_txd_neg;
            rgmii_tctl_o = int_tctl_neg;
        end
    endcase
end
```

It is necessary to set minimum delays. For example, if the net delay for the data path after routing is 0.63 ns, the total path delay is 1.2 ns and the minimum delay must be set to 1.5 (taking care of 0.2-ns delay for the output MUX), as shown below.

```
set_min_delay 1.5 -from {DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmiigmrt_
inst/ clk_tx_div2_reg/CK DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmiigmrt_
inst/ clk_tx_div2_neg_reg/CK} -to [find port phy_txd_o]
```

The set\_min\_delay constraint ensures that buffers or gates are added to delay the path. Make sure that this delay is not greater than (2.4 – clock net delay) ns.

The following timing report shows the additional delay in the Multiplexer Select path achieved using a combination of INVX2 and OAI2BB2XL gates.

### Example 8-3 Multiplexer Select Signal Timing Report With set\_min\_delay of 1.5 Applied

```

Startpoint: DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmi_gmrt_inst/clk_tx_div2_reg
            (rising edge-triggered flip-flop clocked by clk_tx_i)
Endpoint: phy_txd_o[0]
            (output port clocked by clk_tx_180_i)
Path Group: clk_tx_i_outputs
Path Type: max

Des/Clust/Port      Wire Load Model      Library
-----
DWC_gmac_top        tsmc090_wl10       slow

Point                      Incr      Path
-----
clock clk_tx_i (rise edge)          0.00      0.00
clock network delay (ideal)        0.00      0.00
DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmi_gmrt_inst/clk_tx_div2_reg/CK (DFFRQX2)    0.00      0.00 r
DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmi_gmrt_inst/clk_tx_div2_reg/Q (DFFRQX2)      0.33      0.33 f
DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmi_gmrt_inst/U21/Y (INVX2)                  0.14      0.47 r
DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmi_gmrt_inst/U3/Y (CLKXOR2X4)                0.41      0.87 f
DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmi_gmrt_inst/U4/Y (OAI2BB2XL)                 0.35      1.22 f
DWC_gmac_inst/DWC_gmac_rgmiinst/DWC_gmac_rgmi_gmrt_inst/rgmii_txd_o[0]                   0.00      1.22 f
(DWC_gmac_rgmi_gmrt)                                         0.00      1.22 f
DWC_gmac_inst/DWC_gmac_rgmiinst/rgmii_txd_o[0] (DWC_gmac_rgmi)                         0.00      1.22 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/rgmii_txd_i[0] (DWC_gmac_phy_txmux)               0.00      1.22 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/U5/Y (AO22X1)                                 0.35      1.57 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/phy_txd_o[0] (DWC_gmac_phy_txmux)                 0.00      1.57 f
DWC_gmac_inst/phy_txd_o[0] (DWC_gmac)                                         0.00      1.57 f
phy_txd_o[0] (out)                                         0.00      1.57 f
data arrival time                                         1.57

clock clk_tx_180_i (rise edge)          4.00      4.00
clock network delay (ideal)        0.00      4.00
output external delay             -1.60      2.40
data required time                  2.40

data required time                  2.40
data arrival time                  -1.57

slack (MET)                           0.83

```

Another example timing report for the Multiplexer Select path, where the set\_min\_delay value is 1.9, is shown below. An additional delay, using an INVX2 and OAI22X1 gate to meet the minimum delay constraint, is added.

#### Example 8-4 Multiplexer Select Path Timing Report With set\_min\_delay Value of 1.9 Applied

```

Startpoint: DWC_gmac_inst/DWC_gmac_rgmi_i_inst/DWC_gmac_rgmi_i_gmrt_inst/clk_tx_div2_reg
            (rising edge-triggered flip-flop clocked by clk_tx_i)
Endpoint: phy_txd_o[0]
           (output port clocked by clk_tx_180_i)
Path Group: clk_tx_i_outputs
Path Type: max

Des/Clust/Port      Wire Load Model      Library
-----
DWC_gmac_top        tsmc090_wl10       slow

Point                Incr      Path
-----
clock clk_tx_i (rise edge)          0.00      0.00
clock network delay (ideal)        0.00      0.00
DWC_gmac_inst/DWC_gmac_rgmi_i_inst/DWC_gmac_rgmi_i_gmrt_inst/clk_tx_div2_reg/CK (DFFRQX2)    0.00      0.00 r
DWC_gmac_inst/DWC_gmac_rgmi_i_inst/DWC_gmac_rgmi_i_gmrt_inst/clk_tx_div2_reg/Q (DFFRQX2)        0.33      0.33 f
DWC_gmac_inst/DWC_gmac_rgmi_i_inst/DWC_gmac_rgmi_i_gmrt_inst/U23/Y (INVX2)                  0.14      0.46 r
DWC_gmac_inst/DWC_gmac_rgmi_i_inst/DWC_gmac_rgmi_i_gmrt_inst/U12/Y (CLKXOR2X2)              0.47      0.93 f
DWC_gmac_inst/DWC_gmac_rgmi_i_inst/DWC_gmac_rgmi_i_gmrt_inst/U6/Y (INVX2)                   0.38      1.32 r
DWC_gmac_inst/DWC_gmac_rgmi_i_inst/DWC_gmac_rgmi_i_gmrt_inst/U14/Y (OAI22X1)                 0.25      1.57 f
DWC_gmac_inst/DWC_gmac_rgmi_i_inst/DWC_gmac_rgmi_i_gmrt_inst/rgmii_txd_o[0]
(DWC_gmac_rgmi_i_gmrt)               0.00      1.57 f
DWC_gmac_inst/DWC_gmac_rgmi_i_inst/rgmii_txd_o[0] (DWC_gmac_rgmi)                         0.00      1.57 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/rgmii_txd_i[0] (DWC_gmac_phy_txmux)                  0.00      1.57 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/U5/Y (AO22X1)                                     0.34      1.91 f
DWC_gmac_inst/DWC_gmac_phy_txmux_inst/phy_txd_o[0] (DWC_gmac_phy_txmux)                  0.00      1.91 f
DWC_gmac_inst/phy_txd_o[0] (DWC_gmac)                                         0.00      1.91 f
phy_txd_o[0] (out)                           0.00      1.91 f
data arrival time                          1.91

clock clk_tx_180_i (rise edge)          4.00      4.00
clock network delay (ideal)        0.00      4.00
output external delay                -1.60     2.40
data required time                  2.40
-----
```

data required time	2.40
data arrival time	-1.91
slack (MET)	0.49

## 8.12 Synthesis Guidelines

The coreKit supports the complete synthesis flow on your configured RTL core using Synopsys Design Compiler. All the default and required synthesis constraints are automatically generated and applied as per your configured core. After the synthesis is completed, this is generated as DWC\_gmac\_top.sdc in the <workspace>/export/ directory. For users who do not have a license for Design Compiler, an example synthesis script and constraints file is given in <workspace>/resources/synth/ directory

The list below gives the major types of constraints applied to the core during synthesis.

1. By default, all input and output signals have 60% of corresponding clock period as SetInput and SetOutput delays. Some of the input/output signals which have higher delays, are constrained to 40% clock-period. You can change the constraints as per your requirement.
2. Multi-cycle paths are defined for the RMON (MMC) counter updates just for saving area. Functionally, they will operate properly even if the multi-cycle path constraints are removed.
3. False paths are only set for static input signals such as phy\_select\_i, etc.
4. Signals crossing clock domains are NOT set as False-Paths. Instead, they are constrained to have a maximum delay of 1 clock-period of the destination clock. This constraint is recommended to ensure proper functioning of the CDC synchronizers (especially the data path synchronizer) as intended.
5. Clock period of the various PHY-side clock inputs are defined as per Ethernet PHY interface requirements. For the host/system clock input, the default clock period is set as 6 ns.



# A

## Workspace Directory and Files

The following directory and file information is provided for reference. In most cases, you will interface with the DWC Ether MAC 10/100/1000 Universal through coreConsultant and will not work directly with the directories and files. However, to understand the simulation environment, we first need to understand the structure of the directories and files first created when you configure the core tool (coreConsultant).

### A.1 Workspace File Directory Structure

#### A.1.1 Directory Structures of Files (For Multiple Configurations)

After installation, you create a workspace where you generate, simulate, and synthesize your own configuration of the DWC Ether MAC 10/100/1000 Universal. [Figure A-1](#) shows the directory that will be visible once coreConsultant is invoked and the configuration is created. Multiple configurations can be created, as shown.

**Figure A-1** Directory Structure After Configuration

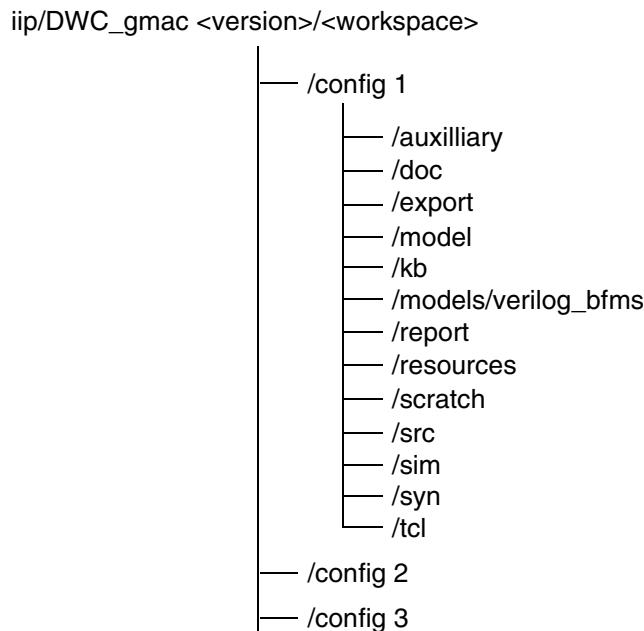
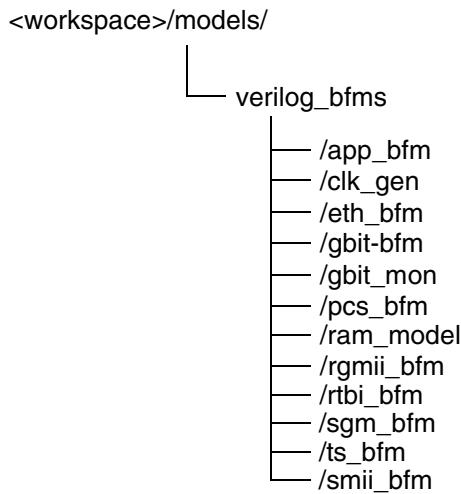


Figure A-2 shows the directories created in models. The models contain various bus functional models required for the test environment.

### Figure A-2 Directory Structure for Models



The source directories (src/) created are based on the selected configuration. For example, if we chose the GMAC-CORE option (refer to “[Parameters](#)” on page 309 for detailed descriptions of all configuration options and parameters), directories mdc/, ahb/, and mtl/ do not appear. Directories ahb/ and mdc/ are not available for the GMAC\_MTL option. Directory ahb/ is not available for the GMAC-DMA option. Each of the PHY interface directories (sgmii/, rgmii/, rmii/, smii/, pcs/, and rtbi/) is available only when enabled during configuration.

Figure A-3 shows all of the source directories.

### Figure A-3 Directory Structure for Source Directories

```
<workspace>/src/
    └── DWC_gmac_top_cc_constants.v
    └── DWC_gmac_top.lst
    └── /mac
    └── /mdc
    └── /ahb
    └── /axi
    └── /common
    └── /mtl
    └── /pcs
    └── /sgmii
    └── /rgmii
    └── /rtbi
    └── /rmii
    └── /smii
    └── /top
```

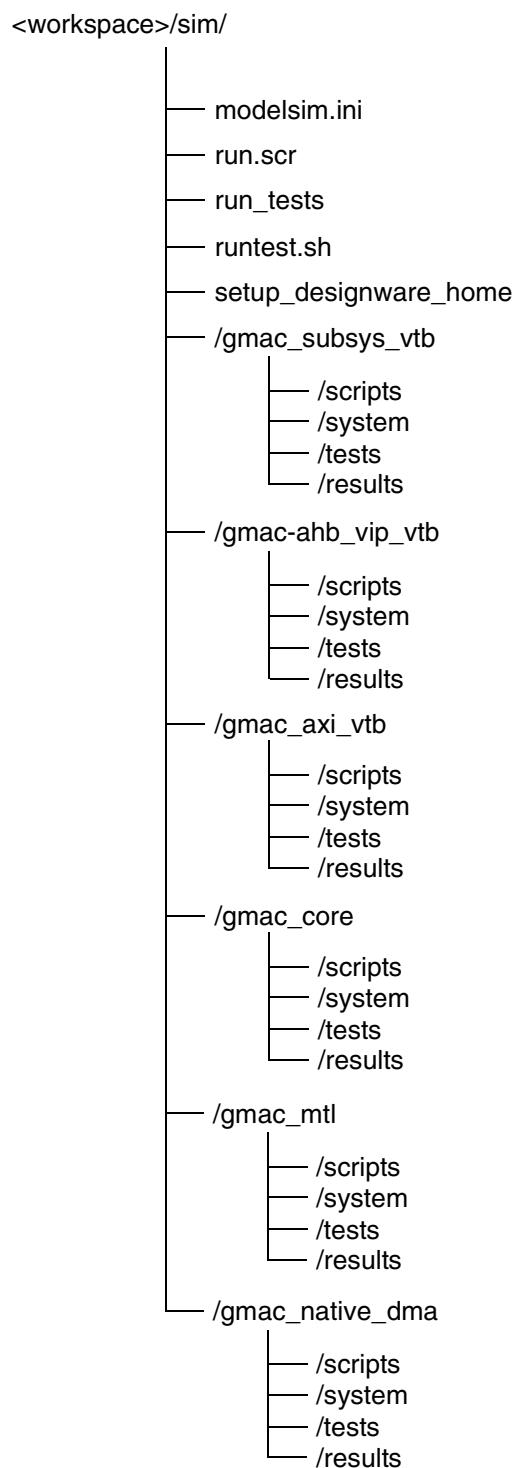
The DWC\_gmac\_top.lst file contains a list of files in compilation order. The files in this list are added based on the configuration selected. The top is the top-level directory that contains the top-level file DWC\_gmac\_top.v. The top-level design name is DWC\_gmac\_top

DWC\_gmac\_top\_cc\_constants.v is a CoreConsultant-generated file that contains parameter values and definitions for all of the selected parameters. (Refer to ["Parameters" on page 309](#) for detailed descriptions of all configuration options and parameters.)

The pcs/, sgmii/, rgmii/, smii/, and rmii/ directories contain files for the PHY interface. These directories will be available based on the corresponding PHY interface selected.

### A.1.2 Directory Structures for the Simulation Environment

All simulation (sim/) directory files are created during the Simulate activity of coreConsultant. The sim/ directories are also created conditionally, based on the options selected. For example, the gmac\_subsys\_vtb/ and gmac-ahb\_vip\_vtb/ directories are only created if the AHB subsystem is chosen. The run.scr is the top-level run script. All of the simulation in command line mode should be run from the sim/ directory. The sim/ directory structure is shown in Figure A-4.

**Figure A-4 Directory Structure for Simulation**

runtest is a Perl file generated by CoreConsultant to run various tests selected. This file also performs various tasks such as taking setting the environment variables for third-party simulators and providing pass/fail status for tests that are run.

The runtest.sh file includes environment variable definitions (VERA\_HOME, VCS\_HOME, and so forth) and the runtest command with different options based on the configuration.

The setup\_designware\_home file contains the paths of the DESIGNWARE\_HOME and VERA\_HOME environment variables.

The modelsim.ini file is the initialization file required for Modelsim.

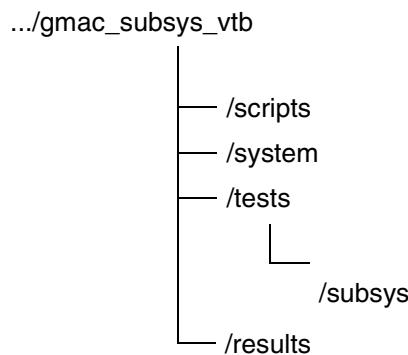
One directory (gmac-axi\_vtb/, gmac\_subsys\_vtb/ & gmac-ahb\_vip\_vtb/, gmac\_native\_dma/, gmac\_core/, or gmac\_mtl/) is created, as determined by the selected configuration (GMAC-AXI, GMAC-AHB, GMAC-DMA, GMAC-CORE, or GMAC-MTL).

The gmac\_subsys\_vtb/ directory structure is shown in [Figure A-5](#). The gmac-ahb\_vip\_vtb/ and gmac\_native\_dma/ directory structures and file contents are similar to that of gmac\_subsys\_vtb/.

The scripts directory contains the following files (only top-level, which can be modified are explained):

- ❖ run\_mti is a run script for ModelSim called by runall\_subsys or runall\_userlist.
- ❖ run\_vcs is a run script for VCS called by runall\_subsys or runall\_userlist.
- ❖ run\_ncv is a run script for NC Verilog called by runall\_subsys or runall\_userlist
- ❖ runall\_subsys is a generated script that has all the tests defined based on the selected configuration.
- ❖ runall\_userlist is a script containing a subset of the runall\_subsys for a quicker check.
- ❖ The system/ directory contains the testbench files.
- ❖ The tests/ directory contains a subsys/ directory that contains test case files.
- ❖ The results/ directory has a testlog/ directory where the results of the individual test cases are stored. The compiled result summary is stored as runtest.log in the sim/ directory.

**Figure A-5 gmac\_subsys\_vtb Directory Structure**



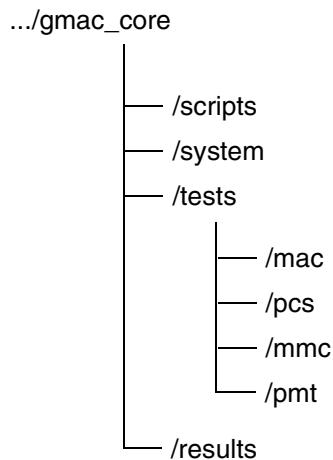
The gmac\_core/ directory structure is shown in [Figure A-6](#). The system and the results directories are similar to what is explained above. The tests directory contains mac/, pcs/, mmc/, and pmt/ subdirectories, which are test case directories for MAC tests, PCS tests, RMON counter tests, and Power Management block tests, respectively.

The scripts directory contains the following important script files:

- ❖ run\_mti, which is a run script for ModelSim called by runall\_xxx or runall\_userlist.
- ❖ run\_vcs, which is a run script for VCS called by runall\_xxx or runall\_userlist.

- ❖ run\_ncv, which is a run script for NC Verilog called by runall\_core or runall\_userlist.
- ❖ runall\_core, which is a script used to run all GMAC-CORE test cases.
- ❖ runall\_userlist, which is a script that contains a subset of the runall\_core tests for quick checks. The appropriate tests are run based on the selected interface.

**Figure A-6 gmac\_core Directory Structure**

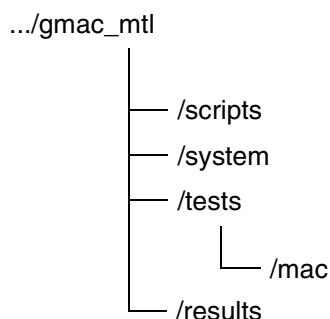


The gmac\_mtl/ directory structure is shown in [Figure A-7](#). The system and the results directories are similar to what is explained for gmac\_subsys\_vtb. The tests contains the mac directory that contains all of the test cases.

The scripts directory contains the following files.

- ❖ run\_mti is a run script for ModelSim called by runall\_gmac\_mtl or runall\_userlist.
- ❖ run\_vcs is a run script for VCS called by runall\_gmac\_mtl or runall\_userlist.
- ❖ run\_ncv is a run script for NC Verilog called by runall\_gmac\_mtl or runall\_userlist.
- ❖ runall\_gmac\_mtl is a script that runs all of the MTL tests.
- ❖ runall\_userlist contains a subset of the runall\_gmac\_mtl test for quick simulation.

**Figure A-7 gmac\_mtl Directory Structure**



Note that all of the tests can be run through the coreConsultant GUI, and that the results are displayed on the GUI (a sample output is shown below). The same output is logged as a text file (runttest.log).

```
=====
Results of the Script Run on Date: 06:09:05 Time: 20:21:34
=====
Note: Running all tests using VCS
Start of GMAC MTL Half Duplex GMII Tests
DATE: 06/09/05 TIME : 20:21:47 PASSED : TEST_gth_lcra_APPCLK_TYP_____
DATE: 06/09/05 TIME : 20:21:57 PASSED : TEST_gth_fifo_full_APPCLK_TYP_____
DATE: 06/09/05 TIME : 20:22:08 PASSED : TEST_grh_ra_APPCLK_TYP_____
DATE: 06/09/05 TIME : 20:22:18 PASSED : TEST_grh_runt_drop_APPCLK_TYP_____
Start of GMAC MTL Full Duplex GMII Tests
DATE: 06/09/05 TIME : 20:22:58 PASSED : TEST_gtf_def_APPCLK_TYP_____
Start of GMAC MTL Half Duplex MII Tests
DATE: 06/09/05 TIME : 20:23:10 PASSED : TEST_mth_lcra_APPCLK_TYP_____
DATE: 06/09/05 TIME : 20:23:29 PASSED : TEST_mth_lcra_APPCLK_TYP_BASE10_____
DATE: 06/09/05 TIME : 20:23:39 PASSED : TEST_mrh_ra_APPCLK_TYP_____
DATE: 06/09/05 TIME : 20:23:51 PASSED : TEST_mrh_ra_APPCLK_TYP_BASE10_____
DATE: 06/09/05 TIME : 20:24:01 PASSED : TEST_mrh_runt_drop_APPCLK_TYP_____
DATE: 06/09/05 TIME : 20:24:12 PASSED : TEST_mrh_runt_drop_APPCLK_TYP_BASE10_____
Start of GMAC MTL Full Duplex MII Tests
DATE: 06/09/05 TIME : 20:25:31 PASSED : TEST_mtf_def_APPCLK_TYP_____
DATE: 06/09/05 TIME : 20:29:54 PASSED : TEST_mtf_def_APPCLK_TYP_BASE10_____
```

## A.2 Workspace Directory and File Descriptions

Table A-1 describes the contents of the various directories and files.

**Table A-1**    **Workspace Directory and File Descriptions**

Directory	Contents
/doc/README_POST_INSTALL/	Contains this core's readme file
/ip/DWC_gmac/<version>	Installed DWC_gmac files
<workspace>	<ul style="list-style-type: none"> <li>• Top-level configured core directory</li> <li>• Created in coreConsultant during the Specify Configuration phase</li> <li>• Named by the user (for e.g. config1)</li> <li>• Contains user specified configuration information and supporting files</li> </ul>
/<workspace>/config#/	Contains the files for specified configurations. Created in coreConsultant during the Specify configuration phase (for example, config1, config2, and so forth).
/config#/doc/	<p>Contains the following documentation for the GMAC-UNIV</p> <ul style="list-style-type: none"> <li>• Databook</li> <li>• Release Notes</li> </ul> <p>Also contains links to the following documents:</p> <ul style="list-style-type: none"> <li>• Quick Start</li> <li>• Installation Guide</li> </ul>
/config#/auxillary/	coreConsultant directory common to all DesignWare coreKits. Contains build and tool information related to coreKit.

**Table A-1    Workspace Directory and File Descriptions (Continued)**

Directory	Contents
/config#/export/	coreConsultant generated files <ul style="list-style-type: none"> <li>• src – Configured source code</li> <li>• DWC_gmac_top.lst – List of source files in proper analysis order</li> <li>• DWC_gmac_top_inst.v – Verilog testbench template</li> <li>• DWC_gmac_top.db – DB format for the top design</li> <li>• DWC_gmac_top.v – Netlist of top design</li> <li>• DWC_gmac_top.sdc – SDC file for the net list</li> <li>• DWC_gmac_top.xml – IP-XACT component representation of the IP (XML format)</li> </ul>
/config#/model/	Reserved for future use
/config#/kb/	.kb files used by CoreConsultant to store coreKit data
/config#/models/verilog_bfms	Contains the bus functional models and monitors for the following <ul style="list-style-type: none"> <li>• Application (app_bfm)</li> <li>• GBIT (gbit_bfm)</li> <li>• GBIT Monitor (gbit_mon)</li> <li>• PCS (pcs_bfm)</li> <li>• RAM (ram_model)</li> <li>• RGMII (rgmii_bfm)</li> <li>• RTBI (rtbi_bfm)</li> <li>• SGMII (sgm_bfm)</li> <li>• Clock generator (clk_gen)</li> <li>• Time Stamp BFM (ts_bfm)</li> <li>• SMII (smii_bfm)</li> </ul>
/config#/report/	Simulation and synthesis report HTML files
/config#/resources/	Contains the following sub-directories <ul style="list-style-type: none"> <li>• atpg: ATPG and Synopsys TetraMAX scripts</li> <li>• dcfpga: README instructions for DC-FPGA synthesis</li> <li>• drivers: Example SW driver source code</li> <li>• gatesim: README and example scripts for Gate-level Net-list simulations</li> <li>• leda_reports: LEDA RTL check reports for 1 (example) configuration</li> <li>• pvci: Example source code and application note for implementing a PVCI/VCI interface for the GMAC-DMA core.</li> <li>• synth: Example Synopsys DC synthesis scripts for the default configuration.</li> </ul>
/config#/scratch/	Temporary files generated and used by CoreConsultant
/config#/src/	coreConsultant configured source RTL files
/config#/sim/	Scripts and files generated for simulations and parsing the results
/config#/syn/	Synthesis strategy scripts and Design compiler output files. coreConsultant controls all operations with these directories and files.
/config#/tcl/	Synthesis input files for coreTools

**B**

# GMAC-AHB Verification Environment Using AMBA and Ethernet VIPs

---

This appendix describes the example Verilog Testbench (VTB) used to test the GMAC-AHB using AMBA Verification IP (VIP) and Ethernet Verification IP. The VTB provides a starting point for understanding how to use AMBA Verification IP (VIP), Ethernet VIP and the configured DWC Ethernet core (configured for an AHB system interface) together in the verification environment. This appendix also provides guidelines for modifying the test environment to use in SoC-level verification.

## B.1 VTB Overview

The example VTB demonstrates the following:

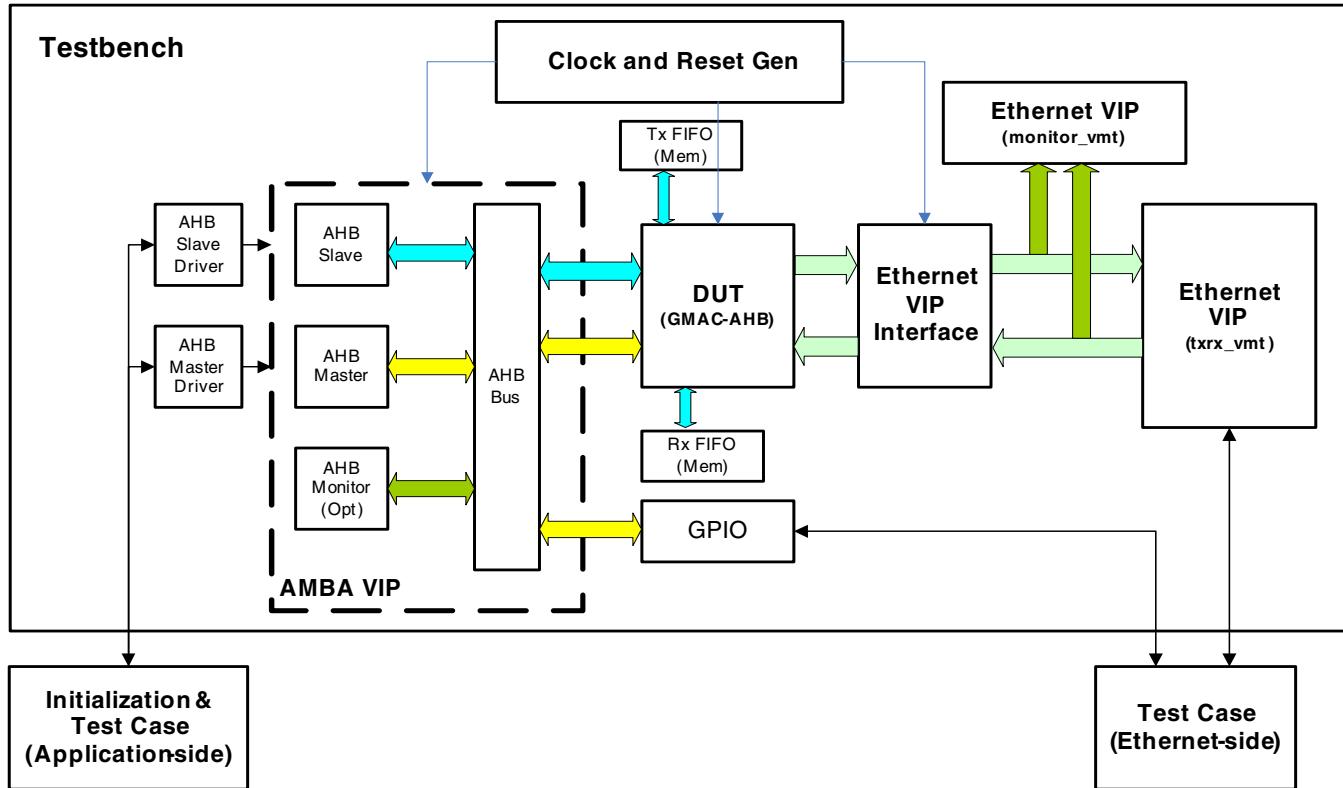
- ❖ How to connect the DesignWare AMBA VIP, Ethernet VIP, and DWC Ethernet (RTL) synthesizable components in a VTB
- ❖ How to use the AMBA VIP and Ethernet VIP with the DWC Ethernet (RTL) synthesizable component to perform basic operating functions

The system uses AMBA VIP on the application side and Ethernet VIP on the Ethernet side. The VIPs have built-in tasks to perform various operations that verify the DUT. For more details about Verification IP, refer to the following documents:

- ❖ *DesignWare AHB Verification IP Databook*, Synopsys Inc., (included in DWC Ethernet download image)  
<https://www.synopsys.com/dw/doc.php/vip/amba/latest/doc/ahbvipdb.pdf>
- ❖ *DesignWare Ethernet Verification IP User Manual*, Synopsys Inc., (included in the DWC Ethernet download image)  
[https://www.synopsys.com/dw/doc.php/vip/ethernet/latest/doc/ethernet\\_vmt\\_user.pdf](https://www.synopsys.com/dw/doc.php/vip/ethernet/latest/doc/ethernet_vmt_user.pdf)
- ❖ *VMT User's Manual*, Synopsys Inc., (included in the DWC Ethernet Database)  
<https://www.synopsys.com/dw/doc.php/vip/vmt/latest/doc/vmtum.pdf>

The VTB's basic structure is shown in [Figure B-1](#). The VTB supports use of the MII, RMII, GMII, RGMII, TBI, or SGMII to connect the DUT and Ethernet VIP. VTB is made compatible to SoC-level verification by constraining application to communicate with Ethernet VIP only through general-purpose I/O (GPIO).

**Figure B-1 GMAC-AHB\_VIP Verification Environment**



The testbench consists of the following blocks:

- ❖ DUT: The device under test (DWC Ethernet core). The VTB instantiates the core configured to AHB configuration.
- ❖ AHB VIP: The DW AMBA Bus VIP (ahb\_bus\_vmt), which is an AHB bus fabric (arbiter and decoder) that can connect up to 15 additional masters and slaves.
- ❖ AHB Master VIP: The DW AMBA VIP (ahb\_master\_vmt), an AHB master that can perform reads and writes to the slaves currently in the system
- ❖ AHB Slave VIP: The DW AMBA VIP (ahb\_slave\_vmt), an AHB slave that interfaces to a Host RAM where data resides. The test case commits writes to memory by means of back-door commands, not through actual AHB transactions.
- ❖ Master and Slave Drivers: A collection of bus-independent tasks called from the Test - Configuration and Control block
- ❖ GPIO: All communication between the application-end and Ethernet-end test cases is routed only through the GPIO block, making the VTB reusable for SoC-level verification. For further details on the GPIO block, see "[Verification Flow](#)" on page [401](#).

- ❖ Ethernet VIP Interface: This block connects DUT ports to corresponding Ethernet VIP ports. Depending on the specified interface (MII, RMII, GMII, RGMII, TBI, or SGMII), the Ethernet VIP permits only specific ports to be connected, while leaving other VIP ports left unconnected as required.
- ❖ Ethernet VIP Monitor: This block passively monitors Ethernet transactions, generates reports about transactions, and logs either all or selected behavior.
- ❖ Ethernet VIP Transceiver model: A transceiver model with a command-based interface that transmits and receives correct and erroneous traffic on the supported interfaces (MII, RMII, GMII, RGMII, TBI, or SGMII).
- ❖ Clock and Reset Gen: This block generates the clock and reset signals the VTB requires for each interface.
- ❖ Test case (application-side): The application-side test case initializes the VTB and controls the application side of the DUT. This also consists of a set of tasks that can be used to control the AMBA VIP. For SoC-level verification, this set of test cases can be rewritten in a high-level language, such as C.
- ❖ Test case (Ethernet-side): This set of test cases controls the Ethernet VIP using predefined tasks. This block communicates with the other VTB blocks only through GPIO.

## B.2 Verification Flow

The basic verification flow is explained in this section. The transmission- and reception-side verification flows are split, the first half pointing to the application side, the second to the Ethernet VIP side.

All communications and handshakes occur between the application and line sides through the GPIO port on the AMBA bus in the testbench. The GPIO port bits are detailed in “[GPIO](#)” on page [411](#)

### B.2.1 Transmission Verification Flow

#### B.2.1.1 Application-Side Test Case Flow

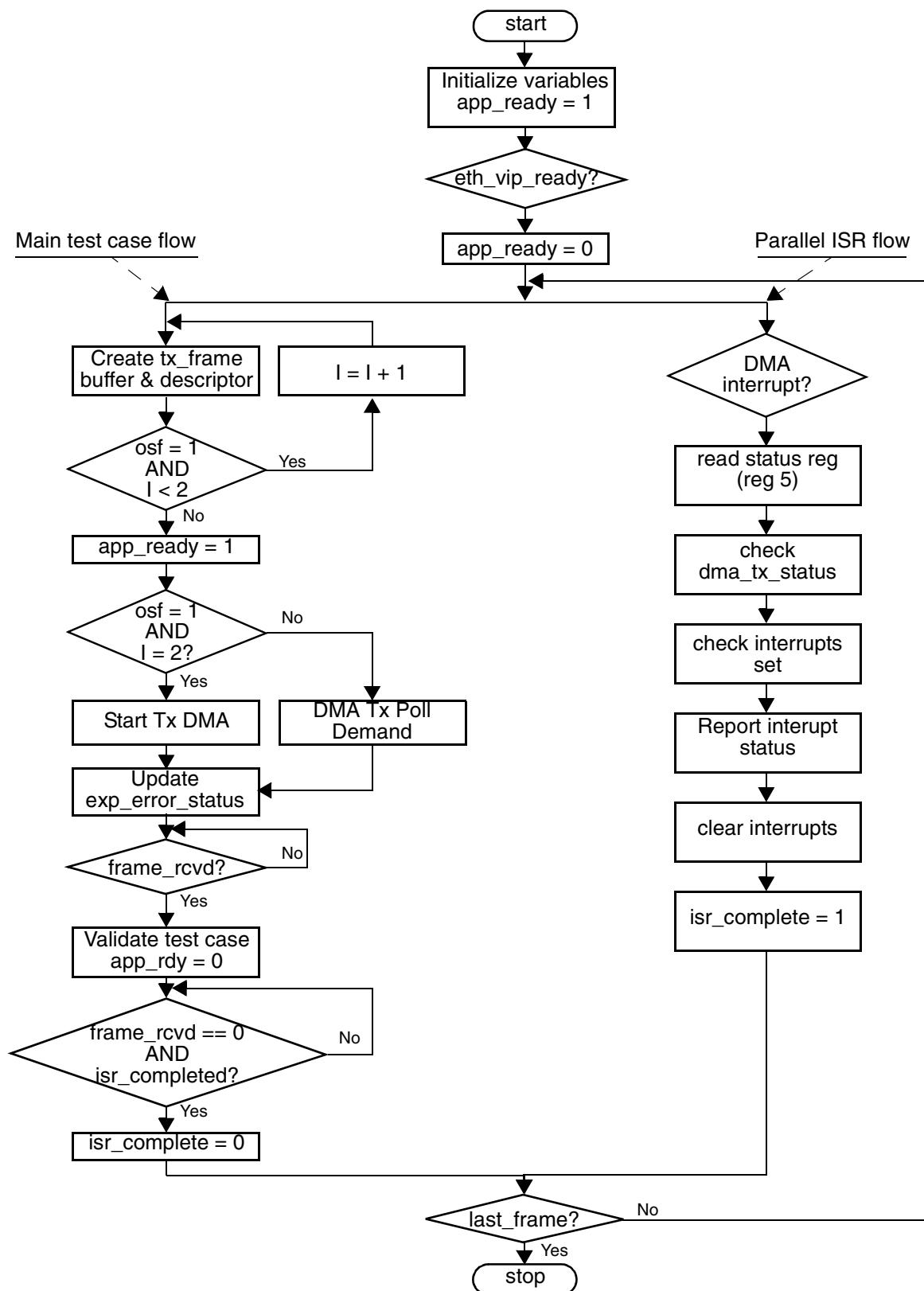
The basic data flow at the host end for verification of the GMAC subsystem (transmission) is as depicted in [Figure B-2](#). The flow is as follows:

1. The application initiates the data transfer by programming the MAC, then initializing all the application-side variables used by the test case.
2. The application-side test case and the Ethernet-side test case perform a handshake to synchronize. The application-side test case sets the app\_ready bit in the GPIO, then reads the GPIO to check the eth\_vip\_ready bit's setting. The Ethernet-side test case sets the eth\_vip\_ready bit after it completes its initialization. The application then clears the app\_ready bit.
3. The application-side test case creates a transmit descriptor and sets the app\_ready bit in the GPIO to indicate that a frame is pending for transmission.
4. Program the DUT (set Start TxDMA or Tx Poll Demand) to start the frame transmission and initialize the expected transmission status in the internal exp\_error\_status variable.
5. The host waits for the Ethernet VIP to set the frame\_rcvd bit in the GPIO.
6. When the Ethernet VIP sets the frame\_rcvd bit, the application-side test case reads rcvd\_error\_status from the GPIO and compares it to exp\_error\_status.
7. The test case passes if exp\_error\_status and rcvd\_error\_status match, and if the Payload Check Status bit (exp\_data\_rcvd) in the GPIO is set.

8. The host clears the app\_ready bit and waits for the Ethernet VIP to clear the frame\_rcvd bit.
9. The host waits for the setting of the isr\_complete bit, then clears it before transmitting the next frame. This step synchronizes the main loop of the test case with the Interrupt Service Routine loop of the test case (shown in [Figure B-2](#) on page [403](#)) for every frame transmitted.
10. If more frames are to be transmitted, the process returns to [Step 2](#), continuing through [Step 9](#).
11. If the OSF bit is set, the host creates two descriptors before transmitting the first frame.

In [Figure B-2](#) on page [403](#), a second thread or process (on the right side) is executed in parallel with the above steps. This is the emulation of the Interrupt Service Routine (ISR) in verilog. You must port the code given in the second thread to the ISR. The isr\_complete variable is used to handshake and synchronize the application-side testcase routines inside and outside of the ISR in this flow. This loop is triggered whenever the DUT asserts an interrupt. The ISR reads the status registers, checking whether the proper and expected interrupt bits are set, then clears the interrupt and waits for the next interrupt.

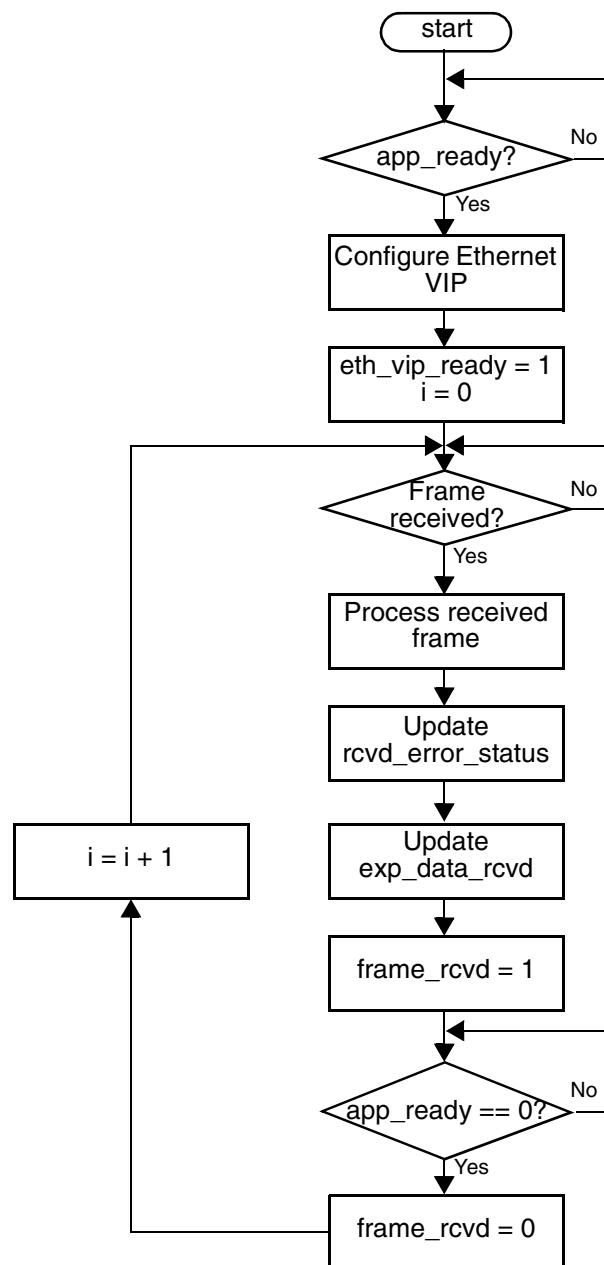
Similarly, the frame\_rcvd variable is used as handshake to synchronize the routines in the application-side test case and the Ethernet-side test case (as explained in [“Ethernet-Side Test Case Flow”](#) on page [404](#)).

**Figure B-2 Application-Side Test Case Flow (Tx)**

### B.2.1.2 Ethernet-Side Test Case Flow

The basic test case flow at the Ethernet VIP (transmission) is as shown in [Figure B-3](#). The flow is as follows:

1. The Ethenet\_VIP test case checks for the app\_ready bit to be set
2. When the app\_ready bit is set, the test case configures the Ethernet VIP and sets the eth\_vip\_ready bit.
3. The Ethernet VIP waits for the frame. When it receives the frame, the test case processes it, checking for any errors the Ethernet VIP may have reported.
4. The corresponding error status is set in rcvd\_error\_status and exp\_data\_rcvd bit is set if there is no mismatch in the expected and received payload.
5. The frame\_rcvd bit is set, until the host clears the app\_ready bit in the GPIO.
6. The Ethernet VIP returns to [Step 3](#)

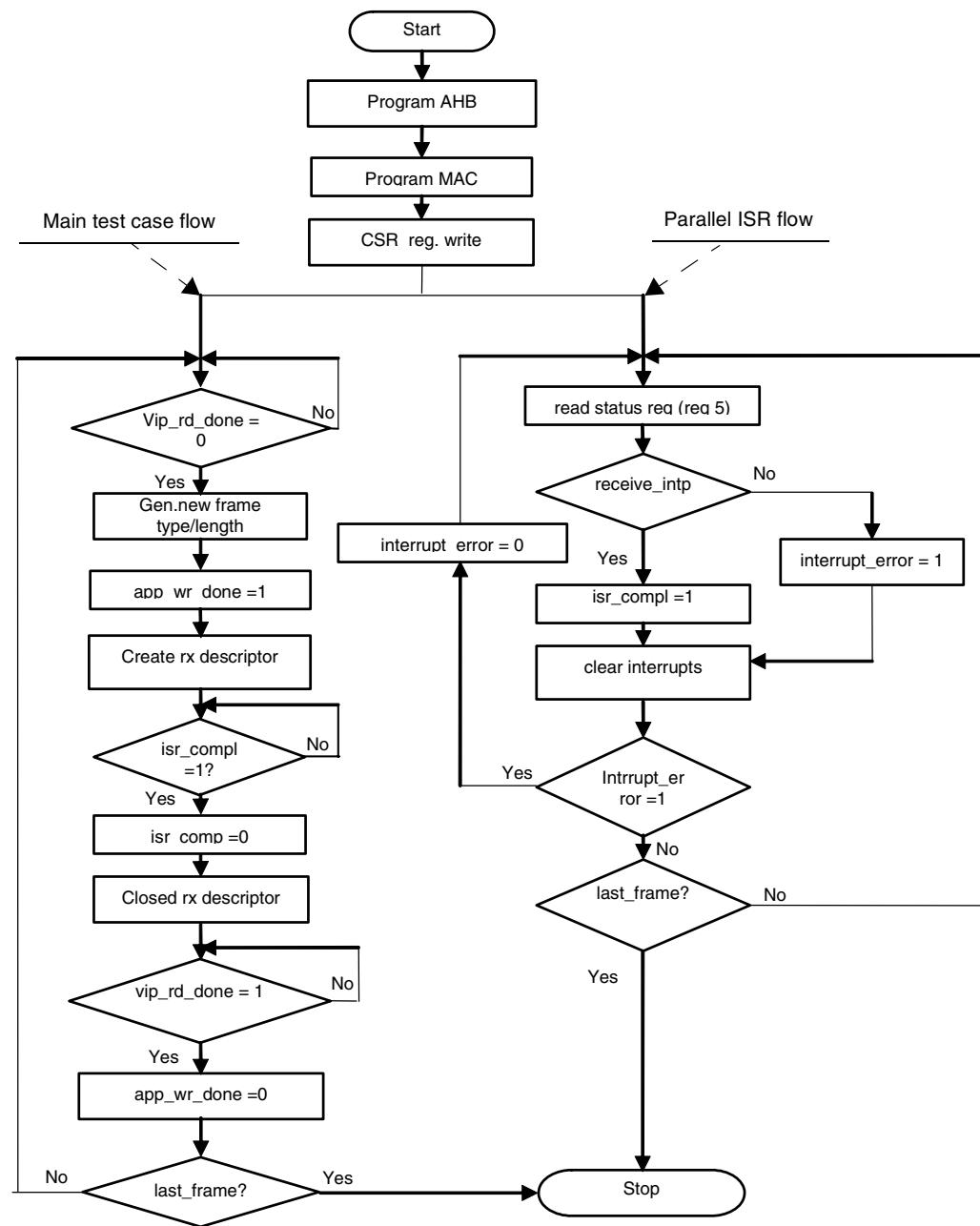
**Figure B-3** Ethernet-Side Test Case Flow (Tx)

## B.2.2 Receive Verification Flow

### B.2.2.1 Host-Side Test Case Flow (Rx)

The basic data flow at the host end (receiving) for the verification of GMAC subsystem is depicted in [Figure B-4](#). The flow is as follows:

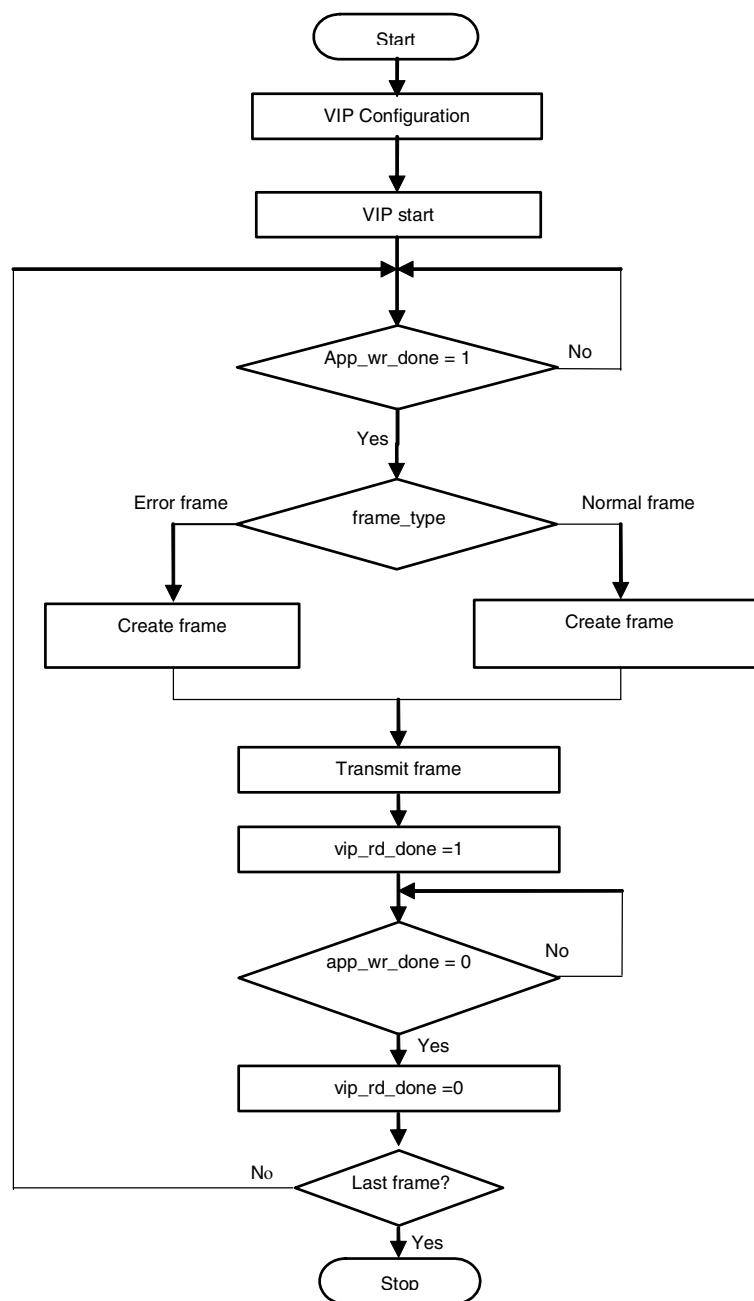
1. The application configures the AHB VIP and MAC and initializes all CSRs.
2. The application waits until the `vip_rd_done` bit in the `GPIO_IN` register is cleared.
3. The application determines the frame's type and length and writes the frame type and length to the appropriate `GPIO_OUT` fields.
4. After writing all information, the application sets the `app_wr_done` bit.
5. The application creates an Rx descriptor of the same frame length it wrote to `GPIO_OUT`.
6. The application waits for a Receive interrupt by checking whether the `isr_complete` variable is set. When the interrupt occurs, the application clears the `isr_complete` variable and moves to [Step 7](#). This step synchronizes the main loop with the Interrupt Service Routine (ISR) loop for each frame. The DUT's interrupt assertion triggers the ISR loop, which checks whether the expected interrupt is asserted. If the expected interrupt is not asserted, the ISR sets the `interrupt_error` variable, clears all interrupts, then waits for the next one.
7. After closing the descriptor task, the application reads the `GPIO_IN` register until it finds the `vip_rd_done` bit set, then clears the `app_wr_done` bit in `GPIO_OUT`.
8. The application repeats [steps 2](#) through [7](#) for the next frame, until the application-side test case sets the `last_frame` bit in the GPIO.

**Figure B-4 Application-Side Test Case Flow (Rx)**

### B.2.2.2 Ethernet-Side Test Case Flow (Rx)

The basic test case flow at the Ethernet VIP (receiving) side is depicted in [Figure B-5](#). The flow is as follows:

1. The VIP is configured, then started.
2. The VIP waits for an application write task on GPIO\_OUT.
3. When app\_wr\_done is set, the VIP reads the current frame information from GPIO\_OUT, creates a frame based on that information, and transmits it to DUT.
4. After transmitting a new frame, the VIP sets vip\_rd\_done on GPIO\_IN, waits for app\_wr\_done to be cleared, then clears the vip\_rd\_done signal.
5. Steps [2](#) through [4](#) are repeated for the next frame, until the application-side test case sets the Last Frame bit.

**Figure B-5 Ethernet-Side Test Case Flow (Rx)**

## B.3 Directory Structure

**Figure B-6 gmac-ahb\_vip\_vtb Directory Structure**

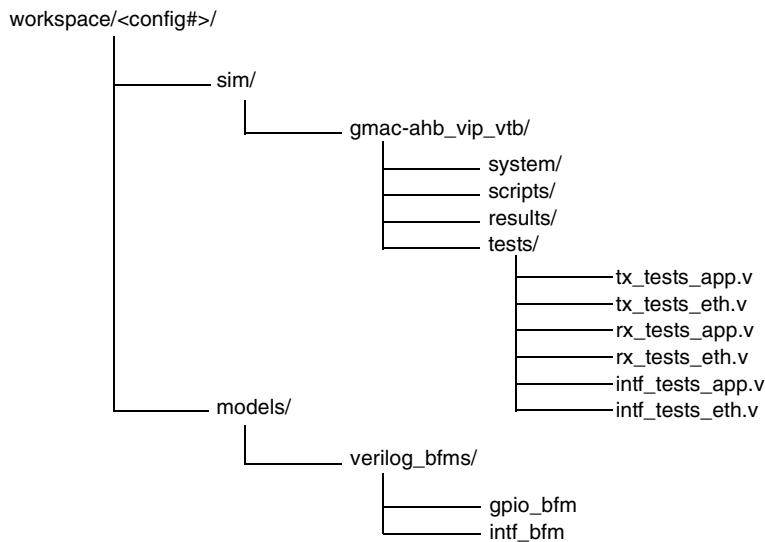


Figure B-6 shows the organization of testbench files, BFM s and test vectors for the gmac-ahb\_vip. The coreKit installation path is the root directory. The workspace/<config#/> directory contains the configured RTL and the testbench. Other directories are explained in Table B-1.

**Table B-1 gmac-ahb\_vip\_vtb Directory Structure**

File Name	Description
sim	Simulation directory containing the testbench, test vectors, and scripts. All simulations are run from this directory
system	Directory containing the testbench, the clock generation module, and the Ethernet VIP tasks, host tasks, host commands, and so forth.
scripts	Directory containing scripts for running test vectors.
results	This directory's test_log subdirectory is a repository for simulation logs.
tests	This directory contains all the transmit and receive test vectors. Test case files with an “_app” suffix contain the application-side code. Test case files with an “_eth” suffix control the Ethernet VIP.
gpio_bfm	This directory contains a GPIO model that the application and Ethernet sides use to communicate.
intf_bfm	This consists of an interface file used for proper port mapping between the DUT and the Ethernet VIP, depending on the selected interface.

## B.4 GPIO

The GPIO is used for communication between the host and the Ethernet VIP. The GPIO has two types of register sets for input ports (GPIO\_IN) and output ports (GPIO\_OUT). The register set is further classified by whether the test case controls frame transmission or reception.

The GPIO base address in the testbench is 0x0000\_C000. The GPIO BFM implements a set of sixteen 32-bit registers to control the I/O ports, with each register bit corresponding to a port bit. The Bit 5 address indicates whether the register is used by the transmit test case (Bit 5 = 0) or the receive test case (Bit 5 = 1). Similarly, Bit 4 of the address decodes whether the register controls the output ports (Bit 4 = 0) or the input ports (Bit 4 = 1).

The address mapping for GPIO is as follows:

**Table B-2** **GPIO Address Map**

Register	Address	Description
1	0000_C000	Transmission-side GPIO_OUT control register
2–4	0000_C004–0000_C00C	Reserved for transmission-side GPIO_OUT
5	0000_C010	Transmission-side GPIO_IN control register
6	0000_C014	Transmission-side GPIO_IN error_status [63:32]
7	0000_C018	Transmission-side GPIO_IN error_status [31:0]
8	0000_C01C	Reserved for Transmission-side GPIO_IN
9	0000_C020	Receive-side GPIO_OUT control register
10–12	0000_C024–0000_C02C	Reserved for Receive-side GPIO_OUT
13	0000_C030	Receive-side GPIO_IN control register
14–16	0000_C034	Reserved for Receive-side GPIO_IN

### B.4.1 GPIO\_OUT (Tx)

The GPIO\_OUT structure for transmission is shown in [Figure B-7](#).

**Figure B-7** **GPIO\_OUT Structure (Tx)**



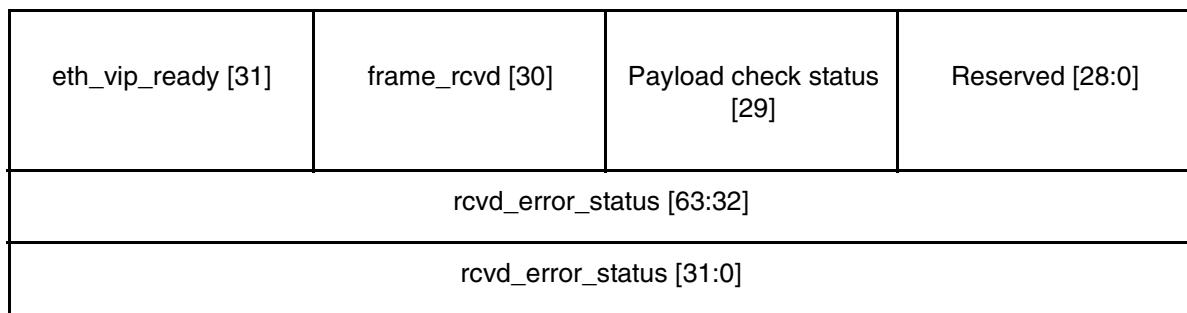
### B.4.2 GPIO\_IN(Tx)

The GPIO\_IN structure for transmission is shown in [Figure B-8](#), GPIO\_IN uses three 32-bit registers: one for the control variables, the other two for the error status received from the Ethernet VIP.

GPIO\_IN fields are as follows:

- ❖ eth\_vip\_ready: This bit is set when the Ethernet VIP is ready to receive the frame.
- ❖ frame\_rcvied: This bit is set when the Ethernet VIP has received the frame.
- ❖ Payload check status: The VIP sets this bit when it receives data that matches the expected data frame.
- ❖ rcvd\_error\_status[63:0]: The 64-bit error status generated by the Ethernet VIP for the received frame is stored in two 32-bit registers, as shown [Figure B-8](#).

**Figure B-8** **GPIO\_IN Structure (Tx)**



### B.4.3 GPIO\_OUT (Rx)

The GPIO\_OUT structure is shown in [Figure B-9](#).

The fields in GPIO\_OUT (Rx) are:

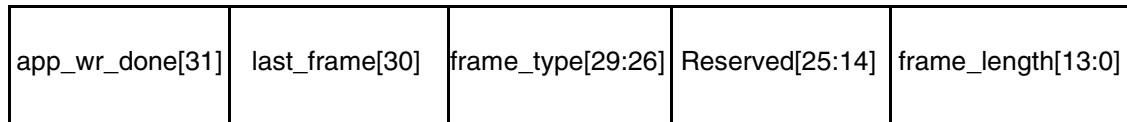
- ❖ app\_wr\_done: After writing all information, the host side sets this bit, which VIP requires to create a new frame for transmission.
- ❖ last\_frame: The host sets this bit when the last frame is being processed.
- ❖ frame\_type: These bits [29:26] indicate the type of frame that the VIP must create for processing. The types of different error the VIP can insert in the frame are based on these bits. Details of these bits are presented in [Table B-3](#)
- ❖ frame\_length [13:0]: These bits indicate the length of the frame.

**Table B-3** **Frame Type Description**

GPIO[29:26]	Frame Type
0000	Normal frame
0001	CRC error frame
0010	Destination Address Filter error frame

**Table B-3 Frame Type Description (Continued)**

GPIO[29:26]	Frame Type
0011	Checksum error

**Figure B-9 gpio\_out Structure (Rx)**

#### B.4.4 GPIO\_IN (Rx)

As shown in [Figure B-10](#), the GPIO\_IN structure uses 32-bit registers of which it currently uses only the following port:

- ❖ vip\_rd\_done: The Ethernet VIP sets this bit when it has read out GPIO\_OUT.

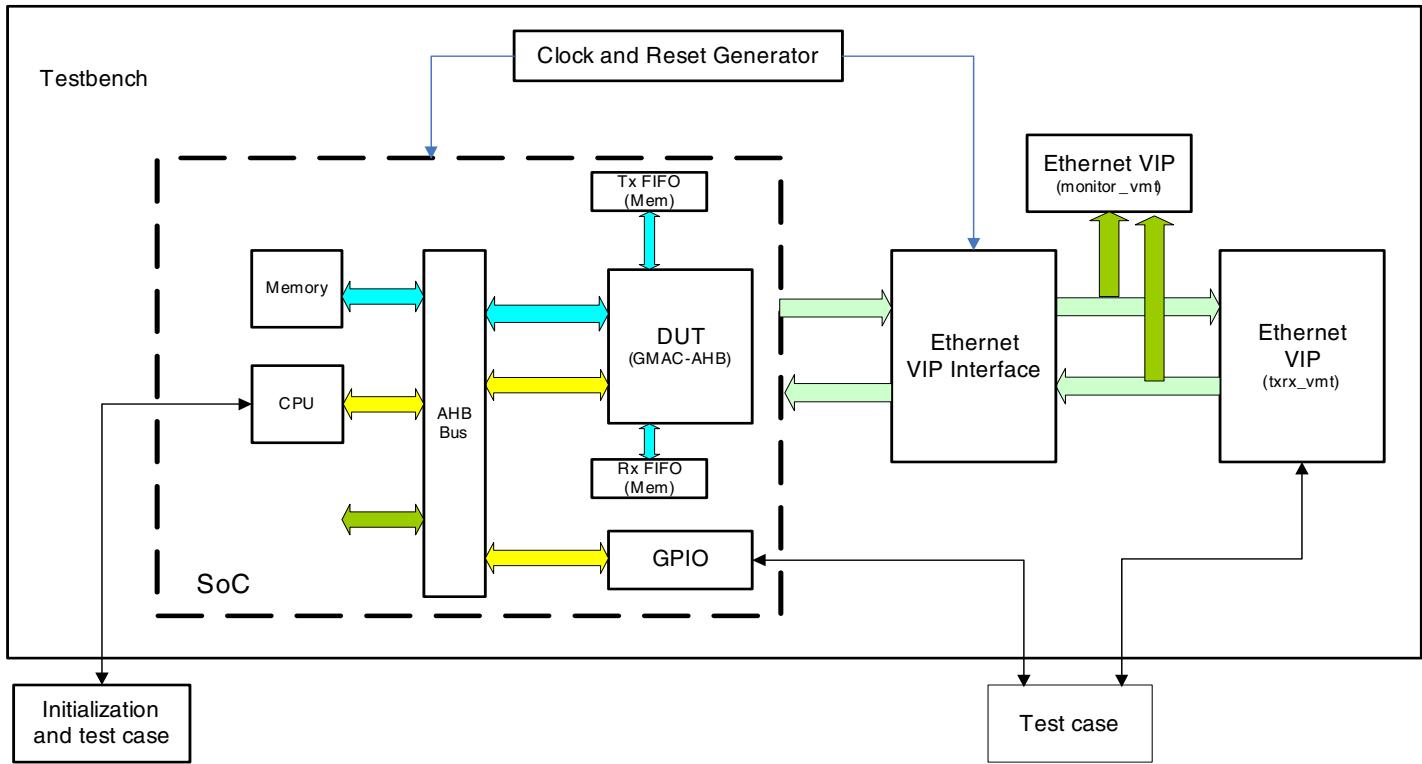
**Figure B-10 gpio\_in Structure (Rx)**

### B.5 SoC-Level Verification Guidelines

This section provides guidelines for SoC-level verification. The test environment is depicted in [Figure B-11](#). In this environment, the SoC, which contains an ARM core (CPU) is the DUT. The application-side vectors are now created in software and loaded into the memory for the CPU to execute.

The verilog testcases given in this testbench can be easily ported to create test cases that can be used to verify SoC integration. The verification strategy must adhere to the following guidelines.

- ❖ In SoC-level verification, you must ensure that all communication between the application-side test case and the Ethernet-side test case passes through the GPIO only.
- ❖ You must rewrite the application-side test case in a high-level language, such as C, and compile it to create machine code. Load this code into the system memory (instruction code memory) of the SoC, then start the simulation. The CPU reads the instructions from memory and executes the test as indicated in the flow charts provided in “[Verification Flow](#)” on page 401.
- ❖ The Ethernet VIP is controlled only with the existing Ethernet-side test case in verilog.
- ❖ You can use the reserved fields in the GPIO to pass control data between the application and the Ethernet VIP.

**Figure B-11** Setup for SoC-Level Verification

## B.6 Running Simulations

To run the individual test case simulations from the command line, enter the following commands from the configured coreKit's <workspace>/sim/ directory:

- ❖ For VCS, enter:

```
./gmac-ahb_vip_vtb/scripts/run_vcs <TEST_CASE_NAME> <OTHER_OPTIONS>
```



To run the receive-side test case, you must pass the RX\_TEST\_EN variable from the command line. The details of the <TEST\_CASE\_NAME> and <OTHER\_OPTIONS> can be obtained from the ./gmac-ahb\_vip\_vtb/scripts/runall\_subsys file.

You can use the logged output of the run (the verilog.log file) to check for errors.

- ❖ For MTI, enter:

```
./gmac-ahb_vip_vtb/scripts/run_mti <TEST_CASE_NAME> <OTHER_OPTIONS>
```

- ❖ For NCV, enter:

```
./gmac-ahb_vip_vtb/scripts/run_ncv <TEST_CASE_NAME> <OTHER_OPTIONS>
```

## B.7 Simulation PASS/FAIL

The simulation run status is logged to the <workspace>/sim/runtest.log file. You can access the details of the run from the results/testlog/<TEST\_NAME>.log file at the end of the simulation.

**C**

# **GMAC-AHB Verification Environment**

This appendix describes the example Verilog Testbench (VTB) used to test the GMAC-AHB and is packaged in the DWC Ethernet coreKit, and explains how to use it. The VTB provides a starting point for understanding how to use DesignWare AMBA Verification IP (VIP), Verilog BFM models, and the configured DWC Ethernet core (configured for AHB system interface) together in the verification environment.

## **C.1 VTB Overview**

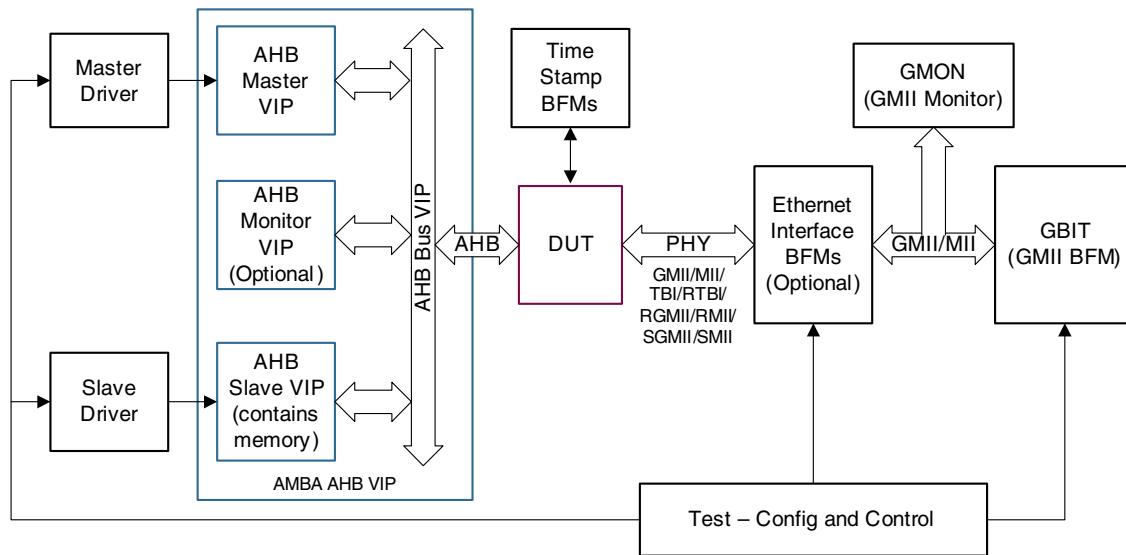
The example VTB demonstrates the following:

- ❖ How to connect the DesignWare AMBA VIP, Verilog BFMs, and DWC Ethernet (RTL) synthesizable component in a VTB
- ❖ How to initialize and program (using Verilog VIP commands) the DWC Ethernet to perform basic operating functions.

On the Ethernet side, the DWC Ethernet DUT provides a GMII/MII/RMII/RGMII/SGMII/TBI/RTBI/SMII interface. On the application side, it interfaces to an AHB. The VIP is used in the VTB as regular Verilog instances. These VIP have their own built-in tasks that can be used just as any regular Verilog task in the VTB.

For more details about VIP, refer to the following documents.

- ❖ DesignWare AHB Verification IP Databook, Synopsys Inc., (included in DWC Ethernet database).  
<https://www.synopsys.com/dw/doc.php/vip/amba/latest/doc/ahbvipdb.pdf>
- ❖ VMT User's Manual, Synopsys Inc., (included in the DWC Ethernet Database)  
<https://www.synopsys.com/dw/doc.php/vip/vmt/latest/doc/vmtum.pdf>

**Figure C-1 GMAC-AHB Verification Environment**

The testbench consists of the following blocks:

- ❖ DUT: The device under test – The DWC Ethernet core. The VTB instantiates the configured core.
- ❖ AHB Bus VIP: The DW AMBA VIP (ahb\_bus\_vmt) is used, which is an AHB bus fabric (arbiter and decoder) that can connect up to 15 additional masters and slaves.
- ❖ AHB Master VIP: The DW AMBA VIP (ahb\_master\_vmt) is used, which is an AHB Bus Master that can perform reads and writes to the slaves presently in the system.
- ❖ Master/Slave Driver: Is a collection of bus independent tasks called from the Test - Config and control block.
- ❖ AHB Slave VIP: The DW AMBA VIP (ahb\_slave\_vmt) is used, which is an AHB Bus Slave that interfaces to a Host RAM where data resides. The writes to memory occur by means of back-door commands and not through actual AHB bus transactions.
- ❖ Ethernet Interface BFM: Translates the interface specific protocol (SGMII/TBI etc.) to the GMII/MII and feeds the GBIT BFM, and vice-versa.
- ❖ GMII Monitor: Monitors the protocol on the GMII/MII bus.
- ❖ GBIT BFM: Transmits/receives the Ethernet frames as per the control of TEST - Config and Control block.
- ❖ Time Stamp BFM: Contains the time stamping BFM files. Captures time stamp values at GMII for verification, and generates time stamps if the External Time Stamp option is selected.
- ❖ TEST - Config and Control: This block is the main test block that configures the VIP and performs the actual tests. It also includes clock/reset blocks. The main testbench control is coded in a simple verilog format. The tests in the VTB support all three speed modes (1000, 100 and 10 Mbps), in Full- and Half-Duplex modes.

## C.2 Initialization

The following sequential steps describe how the DWC Ethernet is initialized and how the VIP is set up in the VTB.

Restart, Start, and Initialization:

1. Perform a hardware power-on reset to the DWC Ethernet (DUT).
2. Configure the VIP (ahb\_master, ahb\_slave, ahb\_bus and ahb\_monitor) and start them. This is done by calling the ahb\_vip\_init task.
3. Configure the DUT. The default configuration is programmed by the task program\_mac\_default.
4. Run the specific scenario of the test case. (For example, see the tf\_basic test case in sim/gmac\_subsys\_vtb/tests/subsys/tx\_tests.v file.)

## C.3 Using VTB

This section briefly describes the files used in the VTB.

Directory Structure and Files:

All files related to this VTB environment are under the configured coreKit:<workspace>/sim/gmac\_subsys\_vtb/system and coreKit:<workspace>/sim/gmac\_subsys\_vtb/scripts. The test case files are under the configured directory coreKit:<workspace>/sim/gmac\_subsys\_vtb/tests/subsys.

[Table C-1](#) describes the VTB source files.

**Table C-1 GMAC-AHB Verification Environment Directory Structure and Files**

File Name	Description
sim/gmac_subsys_vtb/system/test_bench.v	Main testbench Verilog module that has all instantiations, testbench control, tests, and tasks
sim/gmac_subsys_vtb/system/ahb_vip_init.v	AHB initialization task
sim/gmac_subsys_vtb/system/clk_div.v	Clock divider module
sim/gmac_subsys_vtb/system/clk_pll.v	Clock generator module
sim/gmac_subsys_vtb/system/eth_commands.v	This file includes all test files and the mac init tasks
sim/gmac_subsys_vtb/system/eth_host_tasks.v	All tasks required to control the application and Ethernet side are defined here
sim/gmac_subsys_vtb/system/master_driver.v	This file has the memory read and write tasks to the DUT
sim/gmac_subsys_vtb/scripts/mon_cmds.v	Controls certain parameters in the GBIT BFM
sim/gmac_subsys_vtb/scripts/timescale.v	Defines the time scale for the simulation
sim/gmac_subsys_vtb/system/mac_init_tasks.v	This file configures the MAC core in the DUT and the GBIT BFM
sim/gmac_subsys_vtb/scripts/dump_incl.v	In post processing mode, the user can put commands in this file
sim/gmac_subsys_vtb/tests/subsys/frame1.dat	Data in this file is used by Rx Test case (rgf_ipc) to check the frames with Header IP checksum

**Table C-1 GMAC-AHB Verification Environment Directory Structure and Files (Continued)**

<b>File Name</b>	<b>Description</b>
sim/gmac_subsys_vtb/tests/subsys/frame2.dat	Data in this file is used by Rx Test case (rgf_ipc) to check the frames with Header IP checksum
sim/gmac_subsys_vtb/tests/subsys/frame3.dat	Data in this file is used by Rx Test case (rgf_ipc) to check the frames with Header IP checksum
sim/gmac_subsys_vtb/tests/subsys/frame4.dat	Data in this file is used by Rx Test case (rgf_ipc) to check the frames with Header IP checksum.
sim/gmac_subsys_vtb/tests/subsys/frame5.dat	Data in this file is used by Rx Test case (rgf_ipc) to check the frames with Header IP checksum
sim/gmac_subsys_vtb/tests/subsys/frame6.dat	Data in this file is used by Rx Test case (rgf_ipc) to check the frames with Header IP checksum
sim/gmac_subsys_vtb/tests/subsys/frame7.dat	Data in this file is used by Rx Test case (rgf_ipc) to check the frames with Header IP checksum
sim/gmac_subsys_vtb/tests/subsys/frame8.dat	Data in this file is used by Rx Test case (rgf_ipc) to check the frames with Header IP checksum
sim/gmac_subsys_vtb/tests/subsys/frame9.dat	Data in this file is used by Rx Test case (rgf_ipc) to check the frames with Header IP checksum
sim/gmac_subsys_vtb/tests/subsys/frame10.dat	Data in this file is used by Rx Test case (rgf_ipc) to check the frames with Header IP checksum
sim/gmac_subsys_vtb/tests/subsys/intf_tests.v	Has the test for different PHY interfaces
sim/gmac_subsys_vtb/tests/subsys/rx_tests.v	Has all the Rx tests
sim/gmac_subsys_vtb/tests/subsys/tx_tests.v	Has all the Tx tests
sim/gmac_subsys_vtb/tests/subsys/txrx_tests.v	Has a combined Tx and Rx test case
sim/gmac_subsys_vtb/tests/subsys/user_tests.v	You can code your own test cases in this file
sim/gmac_subsys_vtb/tests/subsys/README.rx _tests	README for the RX tests
sim/gmac_subsys_vtb/tests/subsys/README.tx _tests	README for the TX tests

All BFM s are under the configured coreKit:<workspace>/models/verilog\_bfms/ directory. Note that the Ethernet Interface BFM instantiates all PHY interface BFM s. [Table C-2](#) defines the BFM directories.

**Table C-2 BFM Directories**

Directory Name	Description
models/app_bfm	The application BFM for the GMAC-CORE and GMAC_MTL tests is present in this directory
models/eth_bfm	Contains the Ethernet Interface BFM files
models/gbit_bfm	Contains the GBIT BFM files
models/gbit_mon	Contains the GBIT Monitor files
models/pcs_bfm	Contains all PCS BFM-related files
models/ram_model	Contains the DPRAM model, which is instantiated in the testbench
models/rgmii_bfm	Contains the RGMII BFM file
models/rtbi_bfm	Contains the RTBI BFM file
models/sgm_bfm	Contains the SGMII-related files
models/ts_bfm	Contains the time stamping BFM files
models/smii_bfm	Contains the SMII BFM file

[Table C-2](#) describes the simulation output files in the <workspace>/sim/ directory.

**Table C-3 <workspace>/sim Directory Simulation Output Files**

File	Description
results/testlog/*.log	The complete log file of the run corresponding to each test case. This is common for all simulators
runtest.log	The PASS/FAIL status of each test case run. This is common for all simulators.
simv,simv.daidir,csrc	VCS related files
veriuser*, work	Intermediate files generated by MTI simulator
veriuser*,hdl.var,cds.lib,INCA_LIB_S	Intermediate files generated by NC Verilog simulator

## C.4 Tool and Setup Requirements

The prerequisites for running VTB are:

- ❖ All necessary tools installed as per the requirements.  
All tools required to run the DWC Ethernet coreKit simulations are required to run the VTB. Please refer to the DWC Ethernet coreKit Release Notes for tools/license requirements.
- ❖ Environment variables set properly.  
Set the following environment variables:
  - ◆ \$SYNOPSYS – this must be set to a valid DC installation root. The minimum/latest version of the DC version that is supported for VTB is the same as that of the coreKit.
  - ◆ \$DESIGNWARE\_HOME – This must point to the required DesignWare home installation directory for the coreKit.
- ❖ AHB VIP is installed in the DesignWare home. (This step will be done as a part of coreKit simulation).
- ❖ coreKit simulations are running properly.

## C.5 Running Simulations

Currently, VTB supports simulations with VCS, MTI, and NC-Verilog simulators through the GUI by default.

To run the individual test-case simulations from command line prompt, just enter the command as follows from the configured coreKit:<workspace>/sim/ directory:

- ❖ For VCS, enter the command:

```
./gmac_subsys_vtb/scripts/run_vcs <TEST_CASE_NAME> <OTHER_OPTIONS>
```

The details of the <TEST\_CASE\_NAME> and <OTHER\_OPTIONS> can be obtained from the ./gmac\_subsys\_vtb/scripts/runall\_subsys file. The output of the run will be logged in a file called verilog.log file, and can be used to check errors.

- ❖ For MTI, enter the command:

```
./gmac_subsys_vtb/scripts/run_mti <TEST_CASE_NAME> <OTHER_OPTIONS>
```

The details of the <TEST\_CASE\_NAME> and <OTHER\_OPTIONS> can be obtained from the ./gmac\_subsys\_vtb/scripts/runall\_subsys file. The output of the run will be logged in a file called verilog.log file, and can be used to check errors.

- ❖ For NC Verilog, enter the command:

```
./gmac_subsys_vtb/scripts/run_ncv <TEST_CASE_NAME> <OTHER_OPTIONS>
```

The details of the <TEST\_CASE\_NAME> and <OTHER\_OPTIONS> can be obtained from the ./gmac\_subsys\_vtb/scripts/runall\_subsys file. The output of the run will be logged in a file called verilog.log file, and can be used to check errors.

## C.6 Simulation PASS/FAIL

The simulation run status is logged into the coreKit:<workspace>/sim/runtest.log file. The details of the run can be accessed from the results/testlog/<TEST\_NAME>.log file at the end of simulation.

**D**

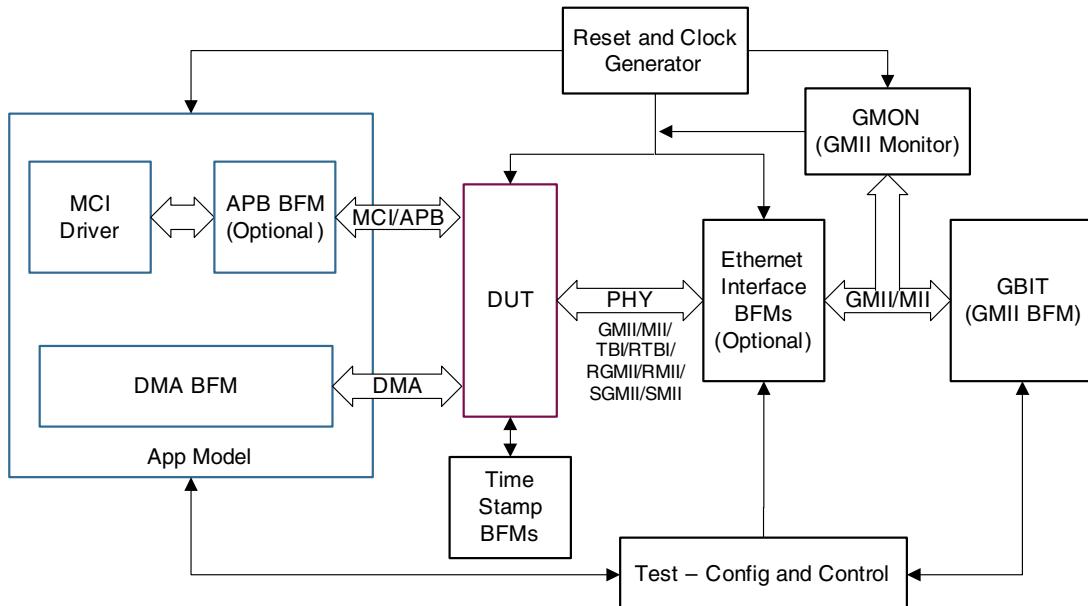
# GMAC-DMA Verification Environment

This appendix describes the example verification environment that is packaged in the DWC Ethernet coreKit for the GMAC-DMA configuration, and explains how to use it.

## D.1 Testbench Overview

[Figure D-1](#) shows the Verification Environment used for GMAC Subsystem with Native DMA Interface.

**Figure D-1 GMAC-DMA Verification Environment**



The testbench contains the following blocks.

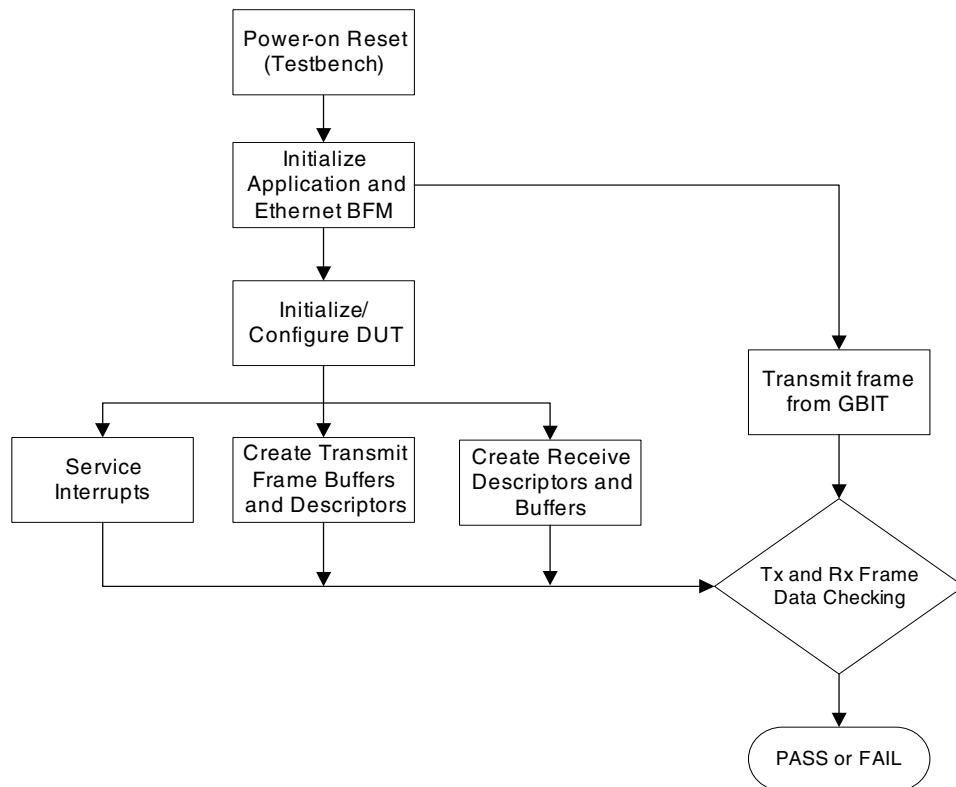
- ❖ Clock Reset Generator: This verilog block generates the clocks and resets for the DUT including the clocks for the PHY interface (GMII, MII, RMII, RGMII, SGMII, SMII, and RTBI) and for all the testbench BFM models.
- ❖ Application BFM: This model contains the

- ◆ DMA BFM which interacts with the DMA of the DUT. This module access the Host memory resided in the testbench for each and every DMA transaction. It has the capability for Generating the Variable PUSH/POP lengths and also error conditions for the MDC interface
- ◆ MCI Driver: This module implements the MCI interface protocol to access the CSR Register set of the DUT.
- ◆ Optional APB BFM: This module implements the APB Protocol required for accessing the internal register set of DUT when optional APB port is selected.
- ❖ Ethernet BFM: Translates the interface specific protocol (SGMII/TBI etc.) to the GMII/MII and feeds the GBIT BFM, and vice-versa.
- ❖ GMII Monitor: Monitors the protocol on the GMII/MII bus.
- ❖ GBIT BFM: Transmits/receives the Ethernet frames with (G)MII protocol as per the control of TEST - Config and Control block.
- ❖ Time Stamp BFM: Contains the time stamping BFM files. Captures time stamp values at GMII for verification. Also generates time stamps if the External Time Stamp option is selected.
- ❖ TEST - Config and Control: This block is the main test block that configures the BFM, DUT and performs the actual tests. The main testbench control is coded in a simple verilog format. The tests in this environment support all 3 speed modes (1000, 100 and 10 Mbps), in Full- and Half-duplex modes.

## D.2 Test Flow

Figure D-2 shows the typical test flow of the GMAC subsystem for Receive and Transmit operations. In both the cases the POR sequence and Initial DUT configuration remains the same.

**Figure D-2 GMAC-DMA Verification Environment Test Flow**



### D.2.1 Frame Receive Flow

The host task creates the receive descriptor on the host memory and then configures the DUT to start the DMA transaction. It then initiates the Ethernet model to send the frame with a predefined data pattern. The DUT starts transferring the data to the host once it started receiving the frame. Upon completion of the reception host compares the received data with the predefined data pattern.

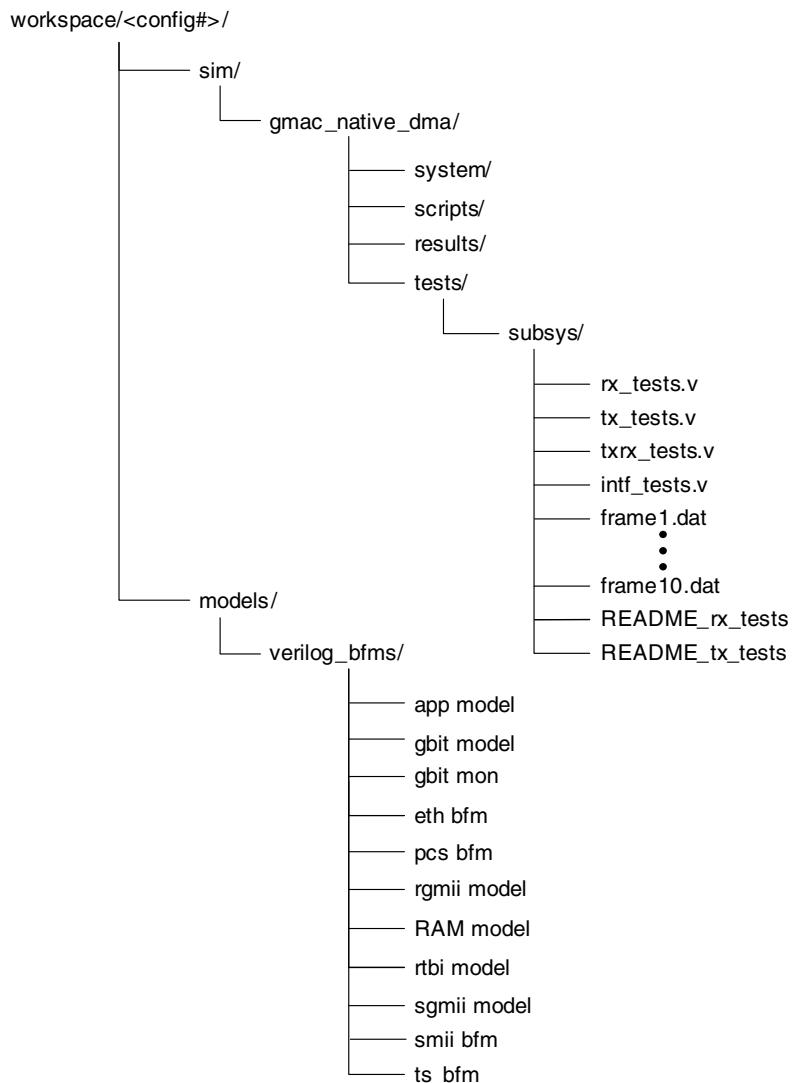
### D.2.2 Frame Transmit Flow

The host fills the host buffer with the predefined data pattern and creates the transmit descriptor on the host memory. It then initiates the DUT to transmit the frame. The DUT starts transferring the data from the host memory through DMA protocol. It waits till the complete frame reaches the Ethernet BFM. Upon reception it does the data integrity check.

## D.3 Directory Structure

[Figure D-3](#) illustrates how the testbench files, BFM models and the test vectors for the GMAC-DMA are organized.

**Figure D-3 GMAC-DMA Verification Environment Directory Structure**



coreKit Installation path is the root directory. The workspace/ directory contains the configured RTL and the testbench. The other directories are described in [Table D-1](#).

**Table D-1 GMAC-DMA Verification Environment Directory Files**

File Name	Description
sim	Simulation directory containing the testbench, test vectors, and scripts. All simulations are run from this directory.
system	Directory containing the testbench and the clock generation module, and also the DMA BFM and MCI driver modules, host tasks, host commands etc.

**Table D-1 GMAC-DMA Verification Environment Directory Files (Continued)**

File Name	Description
tests	Directory containing transmit and receive test vectors
results	This directory has a sub-directory called test log which is a repository for simulation logs
scripts	Directory containing the scripts for running the test vectors
models	Directory consisting of all the verilog BFM's used in the test environment. <ul style="list-style-type: none"> <li>• app_bfm: directory consisting of application side models</li> <li>• gbit_bfm: directory consisting of Ethernet models</li> <li>• gbit_mon: directory hosting the GBIT monitor model</li> <li>• eth_bfm: directory hosting Ethernet interface BFM</li> <li>• pcs_bfm: directory consisting of PCS models</li> <li>• rgmii_bfm: directory consisting of RGMII interface model</li> <li>• sgm_bfm: directory consisting of SGMII interface model</li> <li>• rtbi_bfm: directory consisting of RTBI interface model</li> <li>• ram_model: directory hosting the RAM model for Tx FIFO and Rx FIFO</li> <li>• ts_bfm: directory consisting of the time stamping BFM files</li> <li>• smii_bfm: directory consisting of SMII interface model</li> </ul>

## D.4 Running Simulations

Currently, the verification environment supports simulations with VCS, MTI, and NC-Verilog simulators through the GUI by default.

To run the individual test-case simulations from command line prompt, just enter the command as follows from the configured coreKit:

<workspace>/sim directory:

To run the simulation in VCS use run\_vcs script with the following arguments

```
./gmac_native_dma/scripts/run_vcs <test_name> <appclock_freq> <Tx/Rxfifo_size> <base>
<duplex_mode>
```

where

<test\_name> = Name of the testcase (You can get the list of testcase names from the  
`./gmac_native_dma/scripts/runall_subsys` script file.)

<appclock\_freq> = APPCLK\_MIN, APPCLK\_MAX or APPCLK\_TYP

<Tx/Rxfifo\_size> = Size of the Tx FIFO. Permissible values are:  
TXFIFO\_SIZE\_512, TXFIFO\_SIZE\_1K, TXFIFO\_SIZE\_2K,  
TXFIFO\_SIZE\_4K, TXFIFO\_SIZE\_8k.

RXFIFO\_SIZE\_512, RXFIFO\_SIZE\_1K, RXFIFO\_SIZE\_2K,  
RXFIFO\_SIZE\_4K, RXFIFO\_SIZE\_8k, RXFIFO\_SIZE\_128,  
RXFIFO\_SIZE\_256

<base> = Operational speed mode. Values allowed are:  
BASE10  
BASE100  
BASE1000

<duplex\_mode> =                   The duplex mode of operation. Allowed values are  
  FULL  
   HALF

Similarly, simulations can be run on MTI and NC Verilog as follows:

- ❖ For MTI, enter the command:

```
./gmac_native_dma/scripts/run_mti <test_name> <appclock_freq> <Tx/Rxfifo_size> <base>  
<duplex_mode>
```

- ❖ For NC Verilog, enter the command:

```
./gmac_native_dma/scripts/run_ncv <test_name> <appclock_freq> <Tx/Rxfifo_size> <base>  
<duplex_mode>
```

## D.5 Simulation PASS/FAIL

The simulation run status is logged into the coreKit:

<workspace>/sim/runttest.log file (if simulation is run from GUI).

The details of the run can be accessed from the results/testlog/<TEST\_NAME>.log file at the end of the simulation.

E

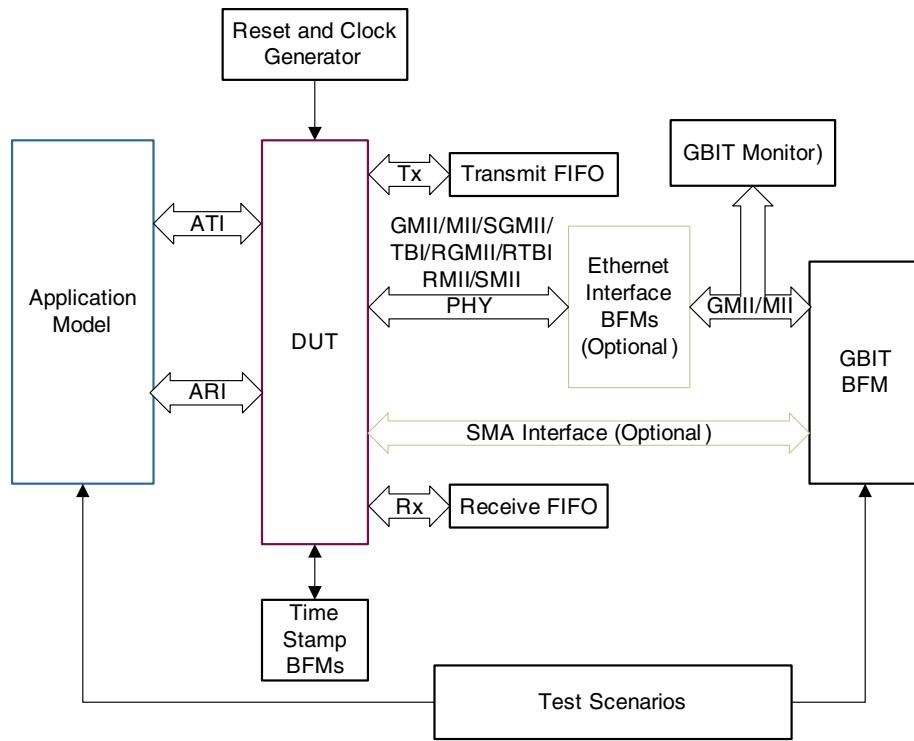
# GMAC-MTL Verification Environment

This appendix describes the example verification environment that is packaged in the DW Ethernet corekit for the GMAC-MTL configuration, and explains how to use it.

## E.1 Testbench Overview

Figure E-1 shows the verification environment used for GMAC-MTL Subsystem with Native FIFO Interface.

## **Figure E-1 GMAC-MTL Verification Environment**



The testbench contains the following blocks.

- ❖ **Clock and Reset:** Verilog model generating the clocks and resets for the DUT. The clocks for the PHY interface (GMII, MII, RMII, RGMII, SGMII, and RTBI) of the DUT and the BFM<sub>s</sub> are also generated by this block.

- ❖ **Application BFM:** The application BFM provides set of tasks for transmitting frames on the transmit interface (ATI) of the DUT. This Model also provides tasks for reading and checking the Transmit Frame status from DUT and tasks for configuring the DUT.

Frames received from the DUT on the receive interface (ARI) of the DUT are retrieved by the application model and checked for data integrity with the data transmitted from the GBIT model. As the application interface of GMAC-MTL is similar to the GMAC-CORE most of the application model tasks are reused in this subsystem verification.

- ❖ **Ethernet Interface BFM:** The PHY Interface BFM, which connects with the DUT through one or many of these PHY interfaces (GMII/MII/SGMII/RGMII/TBI/RTBI/RMII/SMII). Irrespective of the interface on the DUT, this model transforms these into GMII/MII signals which are understood by the GBIT model.
- ❖ **GBT BFM:** A gigabit model has a gigabit transmitter model and a gigabit receiver model. The transmitter model enables you to send all kinds of packets. The transmitter model transfers the frames through the Ethernet bus, per the transmission protocol. The receiver model performs data integrity checks on frames received from the DUT. The gigabit model also provides a PHY model that can be connected to the DUT Station Management interface.
- ❖ **GBT Monitor:** The Monitor Model is connected to the GMII/MII of the Ethernet Interface BFM. This model monitors the DUT's GMII/MII for Ethernet bus protocol conformance and reports misbehavior.
- ❖ **Memory Model:** A simple verilog model that implements the MTL transmit and receive FIFOs. This memory model supports both asynchronous and synchronous read options.
- ❖ **Time Stamp BFM:** Contains the time-stamping BFM files. Captures time stamp values at the GMII for verification. Also generates time stamps if the External Time Stamp option is selected.

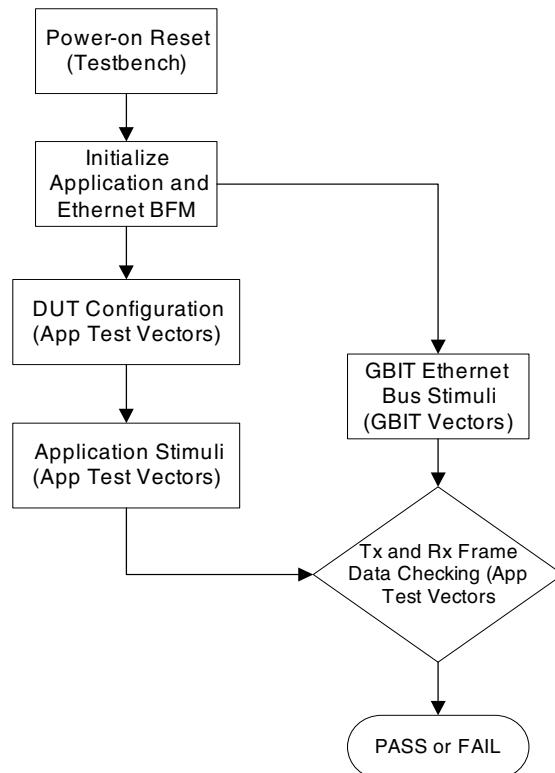
Different test scenarios involve frame transmission from the application and GBIT models. The test cases direct the DUT's configuration through the MAC Control interface.

## E.2 Test Flow

The synchronization of the application and GBIT Ethernet bus stimuli is illustrated through the flow diagram shown in [Figure E-2](#). The figure identifies the different steps involved in the test flow and the testbench components involved in these steps.

The application test vectors and GBIT vectors are organized in different verilog files and run in tandem. The synchronization between the application and GBIT stimuli is accomplished through shared variables.

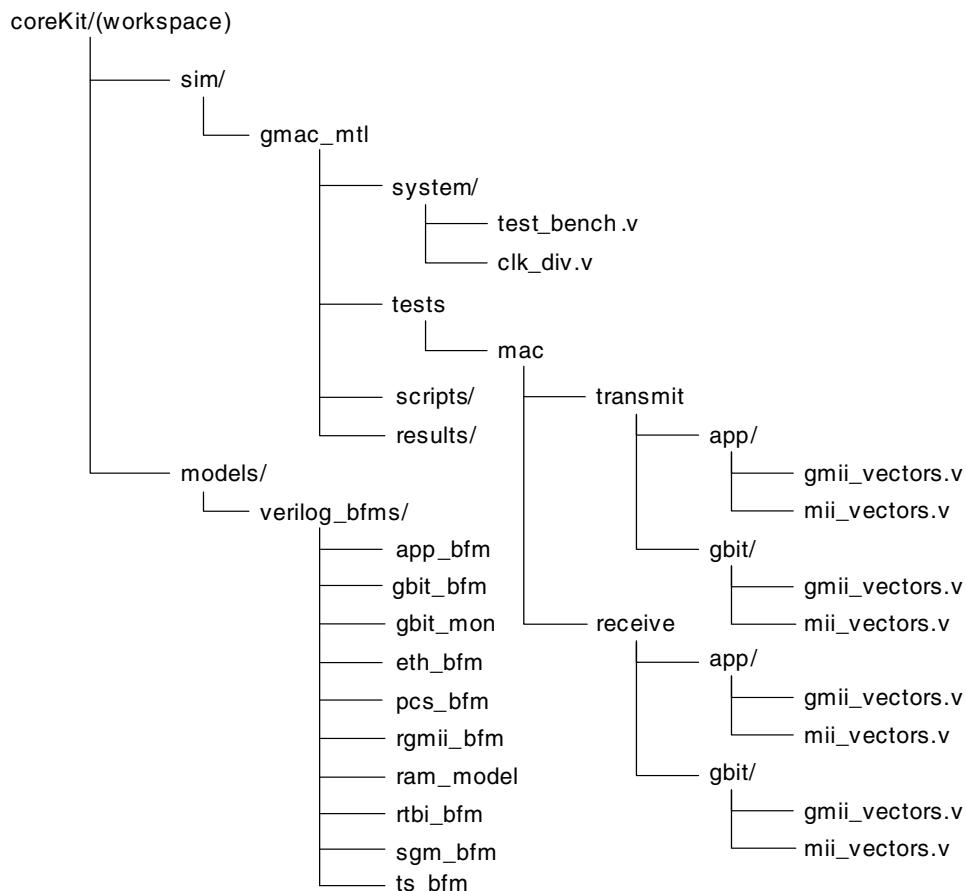
**Figure E-2 GMAC-MTL Verification Environment Test Flow**



## E.3 Directory Structure

Figure E-3 illustrates how the testbench files and the test vectors for the GMAC-MTL verification are organized.

**Figure E-3 GMAC-MTL Verification Environment Directory Structure**



Corekit Installation path is the root directory. The workspace/ directory contains the configured RTL and the testbench. The other directories are described in [Table E-1](#).

**Table E-1 GMAC-DMA Verification Environment Directory Files**

File Name	Description
sim	Simulation directory containing the testbench and the test vectors and the scripts All the simulations are run from this directory.
system	Directory containing the testbench and the clock generation module
tests	Directory containing transmit and receive test vectors
transmit	Directory containing the application and GBIT test vectors for all the transmit test cases
receive	Directory containing the application and GBIT test vectors for all of the receive test cases
app	Directory having the application test vectors for GMII and MII modes

**Table E-1 GMAC-DMA Verification Environment Directory Files (Continued)**

File Name	Description
gbit	Directory having the gbit side test vectors for GMII and MII modes
results	This directory has a sub-directory called testlog, which is a repository for simulation logs.
scripts	Directory containing the scripts for running the test vectors
models	Directory consisting of all the verilog BFM's used in the test environment: <ul style="list-style-type: none"> <li>• app_bfm: directory consisting of application-side models</li> <li>• gbit_bfm: directory consisting of Ethernet models</li> <li>• gbit_mon: directory hosting the GBIT monitor model</li> <li>• eth_bfm: directory hosting Ethernet Interface BFM</li> <li>• pcs_bfm: directory consisting of PCS models</li> <li>• rgmii_bfm: directory consisting of RGMII interface model</li> <li>• sgm_bfm: directory consisting of SGMII interface model</li> <li>• rtbi_bfm: directory consisting of RTBI interface model</li> <li>• ram_model: directory hosting the RAM model for Tx FIFO and Rx FIFO</li> <li>• ts_bfm: directory consisting of the time stamping BFM files</li> <li>• smii_bfm: directory consisting of the SMII interface model</li> </ul>

## E.4 Running Simulations

Currently, the verification environment supports simulations with VCS, MTI, and NC-Verilog simulators through the GUI by default.

To run the individual test-case simulations from command line prompt, just enter the command as follows from the configured coreKit:

<workspace>/sim directory:

To run the simulation in VCS use run\_vcs script with the following arguments:

```
./gmac_mtl/scripts/run_vcs <test_name> <appclock_freq> <apbclk_freq>
|<csrclk_freq> <base> where
<test_name> = Name of the testcase (You can get the list of testcase names from the
               ./gmac_mtl/scripts/runall_gmac_mtl script file.)
<appclock_freq> = can be set to any one of the following values:
                   APPCLK_MIN, APPCLK_MAX or APPCLK_TYP
<apbclk_freq> = can be set to any one of the following values:
                   APBCLK_MIN, APBCLK_MAX or APBCLK_TYP
<csrclk_freq> = can be set to any one of the following values:
                   CSRCLK_MIN, CSRCLK_MAX, CSR_CLK_TYP
```

apbclk\_freq and csrclk\_freq are mutually exclusive options. Different values of apbclk\_freq are used with configuration having APB\_SLAVE and a different (from the application clock) CSR clock. csrclk\_freq is used with configurations having a different CSR clock and the normal MCI interface.

<base> = option to configure 10/100-Mbps mode (BASE10 BASE100 respectively) only for MII mode testcases.

Similarly, simulations can be run on MTI and NC-Verilog simulators as follows:

- ❖ For MTI, enter the command:  
`./gmac_mtl/scripts/run_mti <test_name> <appclock_freq>`
- ❖ For NC-Verilog, enter the command:  
`./gmac_mtl/scripts/run_ncv <test_name> <appclock_freq>`

## E.5 Simulation PASS/FAIL

The simulation run status is logged into the coreKit:

`<workspace>/sim/runtest.log` file (if run from GUI).

The details of the run can be accessed from the results/testlog/<TEST\_NAME>.log file at the end of simulation.

# F

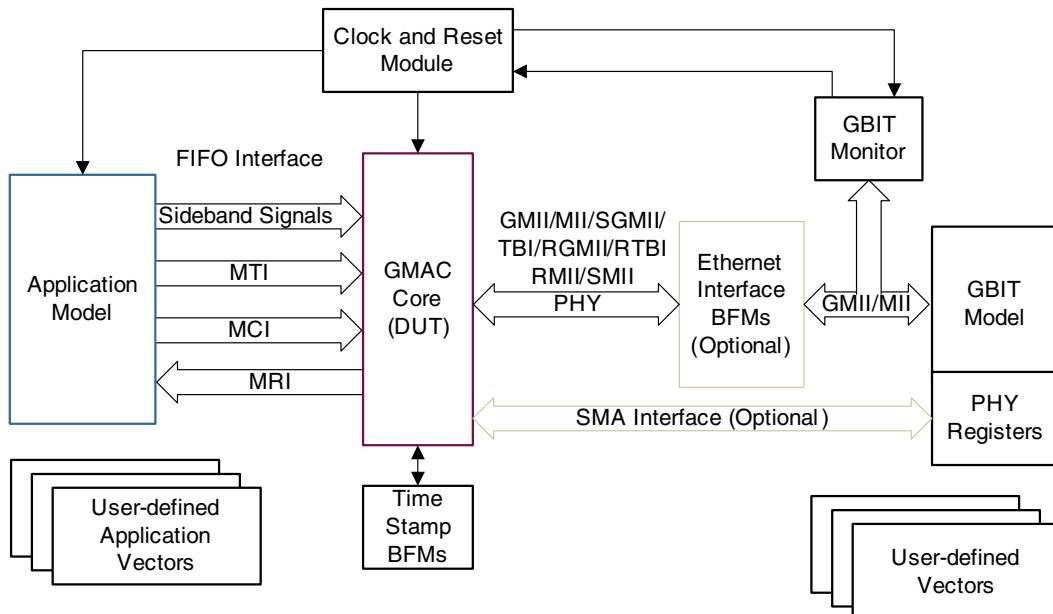
## GMAC-Core Verification Environment

This appendix describes the example verification environment that is packaged in the DWC Ethernet coreKit for the GMAC-CORE configuration, and explains how to use it.

### F.1 Testbench Overview

[Figure F-1](#) shows the Verification Environment used for GMAC-CORE with native FIFO Interface

**Figure F-1 GMAC-Core Verification Environment With Native FIFO Interface**



The testbench contains the following blocks:

- ❖ **Clock and Reset:** Verilog Model generating the clocks and resets for the DUT. The clocks for the PHY interface (GMII, MII, RMII, RGMII, SGMII, SMII, and RTBI) of the DUT and the BFMs are also generated by this block.
- ❖ **Application BFM:** The application BFM provides set of tasks for transmitting frames on the transmit interface (MTI) of the DUT. This model also provides tasks for reading and checking the transmit frame status from DUT and tasks for configuring the DUT.

Frames received from the DUT on the receive interface (MRI) of the DUT are retrieved by the application model and checked for data integrity with the data transmitted from the GBIT model.

- ❖ **Ethernet Interface BFM:** The PHY Interface BFM which connects with the DUT through one or many of these PHY interfaces (GMII, MII, SGMII, RGMII, TBI, RMII, SMII, and RTBI). Irrespective of the interface on the DUT, this model transforms these into GMII/MII signals that the GBIT model can process.
- ❖ **GBT Model:** A gigabit model has a gigabit transmitter model and a gigabit receiver model. The transmitter model enables you to send all kinds of packets. The transmitter model transfers the frames through the Ethernet bus, per the transmission protocol. The receiver model performs data integrity checks on the frames received from the DUT. The gigabit model also provides a PHY model that can be connected to the DUT Station Management interface.
- ❖ **GBT Monitor:** The monitor model is connected to the Ethernet interface BFM's GMII/MII. This model monitors the DUT's GMII/MII for Ethernet Bus protocol conformance and reports misbehavior.
- ❖ **Time Stamp BFM:** Contains time stamping BFM files. Captures time stamp values at the GMII for verification. Generates time stamps if the External Time Stamp option is selected.

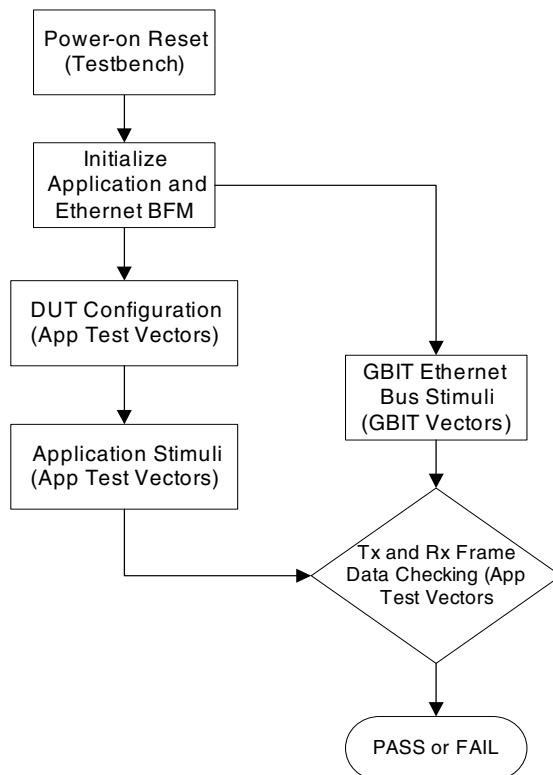
Different test scenarios involve frames transmission from the application and GBIT models. The test cases direct configuration of the DUT through the MAC Control interface.

## F.2 Test Flow

The synchronization of the application and GBIT Ethernet bus stimuli is illustrated through the flow diagram shown in [Figure F-2](#). The figure identifies the different steps involved in the test flow and the testbench components involved in these steps.

The application test vectors and GBIT vectors are organized in different verilog files and run in tandem. The synchronization between the application and GBIT stimuli is accomplished through shared variables.

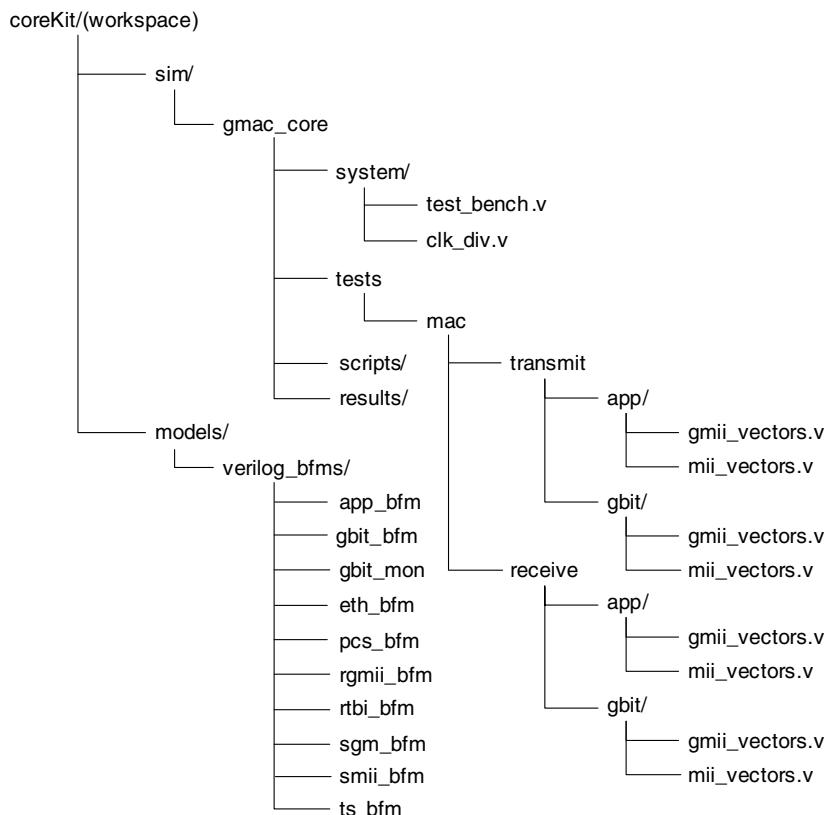
**Figure F-2 GMAC-Core Verification Environment Test Flow**



## F.3 Directory Structure

Figure F-3 illustrates how the testbench files and the test vectors for the GMAC-CORE verification are organized.

**Figure F-3 GMAC-Core Verification Environment Directory Structure**



coreKit Installation path is the root directory. Workspace is the directory containing the configured RTL and the testbench. The other directories are described in [Table F-1](#).

**Table F-1 GMAC-Core Verification Environment Directory Files**

File Name	Description
sim	Simulation directory containing the testbench and the test vectors and the scripts All the simulations are run from this directory. This is the root directory of the simulation environment.
system	Directory containing the testbench and the clock generation module
tests	Directory containing transmit and receive test vectors: <ul style="list-style-type: none"> <li><b>transmit:</b> Directory containing the application and GBIT test vectors for all of the transmit test cases. This directory in turn contains two sub directories: <ul style="list-style-type: none"> <li>- <b>app:</b> Directory having the application test vectors for GMII and MII modes</li> <li>- <b>gbit:</b> Directory having the GBIT side test vectors for GMII and MII modes</li> </ul> </li> <li><b>receive:</b> Directory containing the application and GBIT test vectors for all the receive test cases. This directory in turn contains two sub directories: <ul style="list-style-type: none"> <li>- <b>app:</b> Directory having the application test vectors for GMII and MII modes</li> <li>- <b>gbit:</b> Directory having the GBIT side test vectors for GMII and MII modes</li> </ul> </li> </ul>

**Table F-1 GMAC-Core Verification Environment Directory Files**

File Name	Description
results	This directory has a sub-directory called testlog which is a repository for simulation logs
scripts	Directory containing the scripts for running the test vectors
models	Directory consisting of all the verilog BFM's used in the test environment: <ul style="list-style-type: none"> <li>• app_bfm: directory consisting of application-side models</li> <li>• gbit_bfm: directory consisting of Ethernet models</li> <li>• gbit_mon: directory hosting the GBIT monitor model</li> <li>• eth_bfm: directory hosting Ethernet interface BFM</li> <li>• pcs_bfm: directory consisting of PCS models</li> <li>• rgmii_bfm: directory consisting of RGMII interface model</li> <li>• sgm_bfm: directory consisting of SGMII interface model</li> <li>• rtbi_bfm: directory consisting of RTBI interface model</li> <li>• ts_bfm: directory consisting of the time stamping BFM files</li> <li>• smii_bfm: directory consisting of SMII interface model</li> </ul>

## F.4 Running Simulations

The simulation environment is generated after the simulate activity is completed in the coreKit. You can run the simulations through the GUI or generate scripts only and run them using the command prompt.

Sample scripts are available in the scripts directory to run simulations using VCS, MTI, and NC-Verilog.

To run an individual test in VCS use run\_vcs script from the sim/ directory with the following arguments

```
./gmac_core/scripts/run_vcs <test_name> <base> <interface> where
<test_name>=          Name of the testcase
<base> =             BASE10, BASE100 (used in MII mode, to differentiate between 10-Mbps and
                      100-Mbps tests.
<interface> =         If you have opted for any interface other than GMII, the following must be
                      passed as an argument:
                      RGMII_SEL: RGMII interface
                      SGMII_SEL: SGMII interface
                      TBI_SEL: TBI interface
                      RTBI_SEL: RTBI interface
                      RMII_SEL: RMII interface
                      SMII_SEL: SMII interface
```

Similarly, simulations can be run on MTI and NC-Verilog as follows:

- ❖ For MTI, enter the command:

```
./gmac_core/scripts/run_mti test_name base interface
```

- ❖ For NC-Verilog, enter the command:

```
./gmac_core/scripts/run_ncv test_name base interface
```

## F.5 Simulation PASS/FAIL

The simulation run status is logged into the coreKit:

<workspace>/sim/runtest.log file (if run from the GUI)

A sample log file of the runall\_userlist is shown below. The details of the individual test case run can be accessed from the results/testlog/<TEST\_NAME>.log file at the end of simulation.

```
+-----+
| Summary of all Results |
+-----+
```

```
+-----+
| Verification Activity Log |
+-----+
```

This logfile is created by the runtest.sh script found in  
 /remote/dept-SG-Reg/ethernet\_regress/revankar/simulation/config4/sim  
 The simulator and testbench preparation steps should follow immediately.

```
+-----+
| Testbench Preparation |
+-----+
```

(this section of runtest.log supplied by runtest script)

```
runtest: To recreate the run from this point for debug, do the following
runtest: % cd /remote/dept-SG-Reg/ethernet_regress/revankar/simulation/config4/sim
runtest: % runtest\
          --test test_core\
          --SimChoice VCS\
          --DesignView RTL\
          --NetlistDir <coreKit>/src
```

```
runtest: coreKit in file:/simulation
runtest: runtest in file:/simulation/config
runtest: test is in file:/simulation/config/sim
runtest: Creating simulation command file:
runtest:   file:/simulation/config/sim/test.sim_command
runtest: Creating simulation start script test.startsim containing:
runtest: % gmac_core/scripts/runall_m1000 VCS > /dev/null
runtest: Running test.startsim at Thu Sep  8 16:38:44 IST 2005
```

```
+-----+
| Individual Testcase Logs |
+-----+
```

(The logs are stored in gmac\_core/results/testlog directory)

```
+-----+
| Compile Execution   |
+-----+
```

(this section of runtest.log supplied by test.startsim script)

=====

Results of the Script Run on Date: 09:08:05 Time: 16:38:44

=====

Note: Running GMII tests using VCS

Start of Transmit Tests

Start of Transmit Half Duplex GMII Tests

DATE: 09/08/05 TIME : 16:39:28 PASSED : TEST\_gth\_cop\_\_\_\_\_  
DATE: 09/08/05 TIME : 16:40:12 PASSED : TEST\_gth\_cod\_\_\_\_\_  
DATE: 09/08/05 TIME : 16:40:38 PASSED : TEST\_gth\_ncrs\_\_\_\_\_  
DATE: 09/08/05 TIME : 16:53:50 PASSED : TEST\_gth\_rcnt\_\_\_\_\_

Start of Transmit Full Duplex GMII Tests

DATE: 09/08/05 TIME : 17:25:34 PASSED : TEST\_gtf\_def\_\_\_\_\_  
DATE: 09/08/05 TIME : 17:28:00 PASSED : TEST\_gtf\_dc\_\_\_\_\_  
DATE: 09/08/05 TIME : 17:30:57 PASSED : TEST\_gtf\_dp\_\_\_\_\_  
DATE: 09/08/05 TIME : 17:33:29 PASSED : TEST\_gtf\_dpdc\_\_\_\_\_

Start of Receive Tests

Start of Receive Half Duplex GMII Tests

DATE: 09/08/05 TIME : 17:45:20 PASSED : TEST\_grh\_ra\_\_\_\_\_  
DATE: 09/08/05 TIME : 17:45:27 PASSED : TEST\_grh\_rper\_\_\_\_\_  
DATE: 09/08/05 TIME : 17:45:34 PASSED : TEST\_grh\_de\_\_\_\_\_  
DATE: 09/08/05 TIME : 17:45:41 PASSED : TEST\_grh\_dro\_\_\_\_\_  
DATE: 09/08/05 TIME : 17:47:23 PASSED : TEST\_grh\_mc\_\_\_\_\_

Start of Receive Full Duplex GMII Tests

DATE: 09/08/05 TIME : 18:59:23 PASSED : TEST\_grf\_pass\_\_\_\_\_  
DATE: 09/08/05 TIME : 18:59:40 PASSED : TEST\_grf\_lc\_\_\_\_\_  
DATE: 09/08/05 TIME : 18:59:48 PASSED : TEST\_grf\_ty\_\_\_\_\_  
DATE: 09/08/05 TIME : 19:00:06 PASSED : TEST\_grf\_pctrl\_\_\_\_\_



**G**

# **GMAC-AXI Verification Environment**

This appendix describes the example Verilog Testbench used to perform functional verification of the GMAC-AXI subsystem. This VTB provides a starting point for understanding how to use DesignWare AMBA Verification IP (VIP), Ethernet VIP and the configured DWC Ethernet core (configured for AXI system interface) together in the verification environment.

## **G.1 VTB Overview**

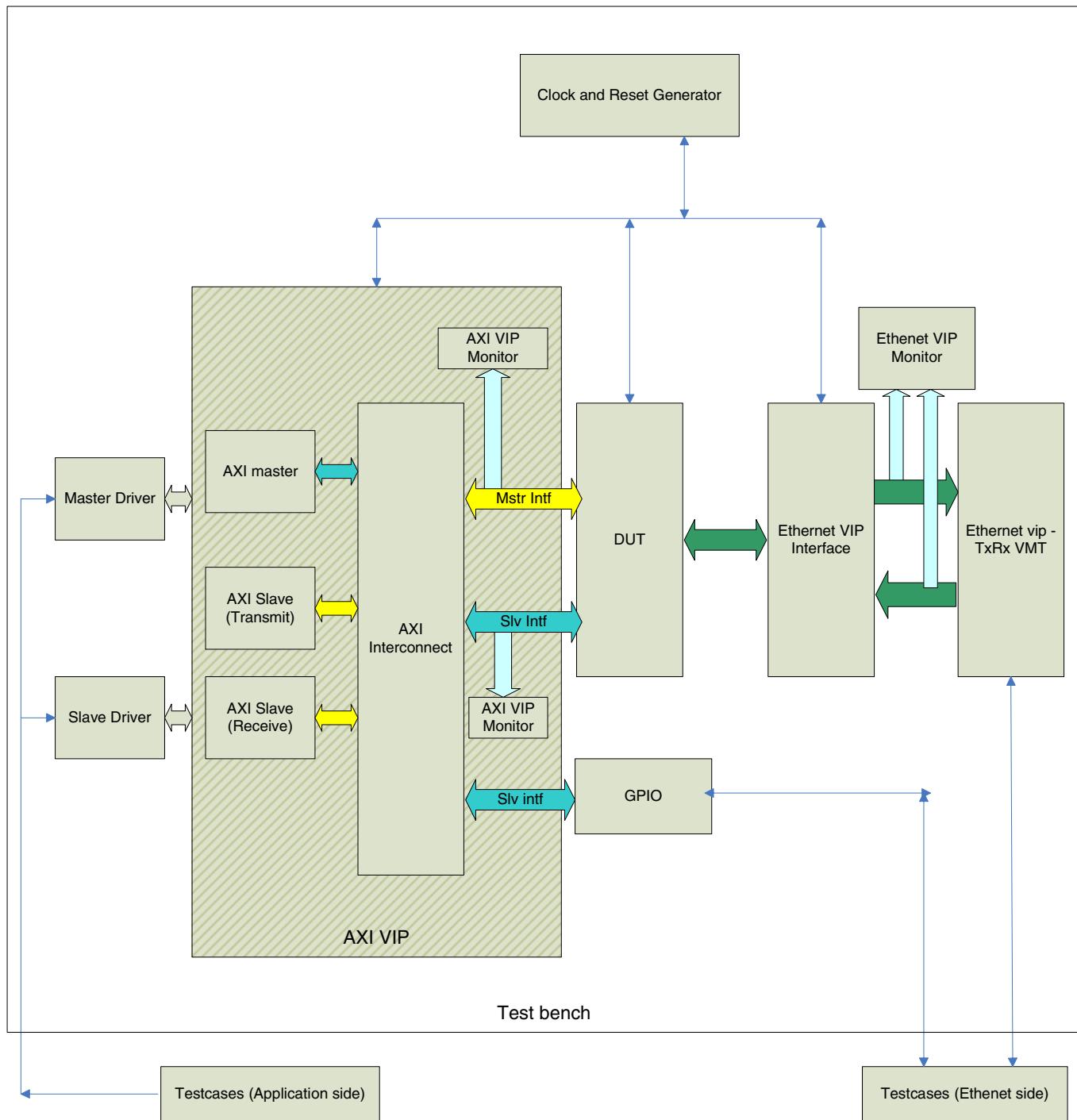
The example VTB demonstrates the following:

- ❖ How to connect the AMBA VIP, Ethernet VIP and DWC Ethernet (RTL) synthesizable components in a VTB.
- ❖ How to initialize and program (using Verilog VIP commands) the DWC Ethernet to perform basic operating functions.

The system uses AMBA VIP on the application side and Ethernet VIP on the Ethernet side (see [Figure G-1](#)). The VIPs have their own built-in tasks that are used to perform various operations to verify the DUT. This VTB is made compatible to SoC level verification by constraining the application-side test to communicate with Ethernet VIP only through GPIO.

For more details about verification IP, refer to the following Documents:

- ❖ Designware AXI verification IP Databook Databook, Synopsys Inc., (included in DWC Ethernet database).  
[http://www.synopsys.com/dw/doc.php/vip/axi/latest/doc/axi\\_um.pdf](http://www.synopsys.com/dw/doc.php/vip/axi/latest/doc/axi_um.pdf)
- ❖ *DesignWare Ethernet Verification IP User Manual*, Synopsys Inc., (included in the DWC Ethernet database).  
[https://www.synopsys.com/dw/doc.php/vip/ethernet/latest/doc/ethernet\\_vmt\\_user.pdf](https://www.synopsys.com/dw/doc.php/vip/ethernet/latest/doc/ethernet_vmt_user.pdf)
- ❖ *VMT User.s Manual*, Synopsys Inc., (included in the DWC Ethernet Database).  
<https://www.synopsys.com/dw/doc.php/vip/vmt/latest/doc/vmtum.pdf>

**Figure G-1 GMAC AXI Verification Environment**

The VTB consists of the following blocks:

- ❖ DUT: The device under test – DW\_GMAC\_AXI\_SUBSYS core, The VTB instantiates the core configured to AXI configuration.
- ❖ AXI Interconnect: The DW AMBA VIP “axi\_interconnect\_vmt”, an Interconnect to connect the AXI masters and slaves.
- ❖ AXI Master: The DW AMBA VIP “axi\_master\_vmt”, an AXI Master Model that can perform reads and writes to the slaves present in the system. This will be used to write and read the DUT CSR for initialization, interrupt service, etc. This will also be used to write from and read into the GPIO which is used to control the Ethernet VIP operation.
- ❖ AXI Slave: The DW AMBA VIP “axi\_slave\_vmt”, an AXI Slave model that interfaces to Memory. The test case transfers data to/from this memory by means of back-door commands, not through actual AXI transactions. But, the DUT accesses are through proper interface transactions.
- ❖ AXI Monitor VIP: The DW AMBA VIP “axi\_monitor\_vmt”, an AXI monitor to perform AXI protocol checking.
- ❖ Master Driver: This master driver is a collection of bus independent tasks called from the test control block. These tasks in turn will call the AXI Master VIP commands.
- ❖ Slave Driver: It consists of bus independent tasks to perform read and writes to the memory in axi\_slave\_vmt through “backdoor” commands.
- ❖ Ethernet TxRX VIP: The DW ETHERNET VIP “ethernet\_txrx\_vmt”, A transceiver model with a command-based interface that transmits and receives correct and erroneous traffic on the supported interfaces (MII, RMII, GMII, RGMII, TBI, or SGMII).
- ❖ Ethernet monitor VIP: The DW ETHERNET VIP “Ethernet\_monitor\_vmt”, This block passively monitors Ethernet transactions, generates reports about transactions and logs. Either all or selected behavior is used to track the various transactions with Ethernet VIP.
- ❖ GPIO: All communication between the application-end and Ethernet-end test cases is routed only through the GPIO block, making the VTB reusable for SoC-level verification.
- ❖ Interface BFM: This block connects DUT ports to corresponding Ethernet VIP ports. Depending on the specified interface (MII, RMII, GMII, RGMII, TBI, or SGMII), the Ethernet VIP permits only specific ports to be connected, while leaving other VIP ports left unconnected as required.
- ❖ Test case (application-side): The application-side or host-side test case initializes the VTB and controls the application side of the DUT. This also consists of a set of tasks that can be used to control the AMBA VIP. For SoC-level verification, this set of test cases can be rewritten in a high-level language, such as C.
- ❖ Test case (Ethernet-side): This set of test cases controls the Ethernet VIP using predefined tasks. This block communicates with the other VTB blocks only through GPIO.
- ❖ Clock and Reset Generation: This block is used to generate the Clock and Reset required for the entire system.

## G.2 Verification Flow

The basic verification flow is explained in this section. The transmission- and reception-side verification flows are split, the first half pointing to the application side, the second to the Ethernet VIP side. All communications and handshakes occur between the application and line sides through the GPIO port only.

### G.2.1 Transmission Verification Flow

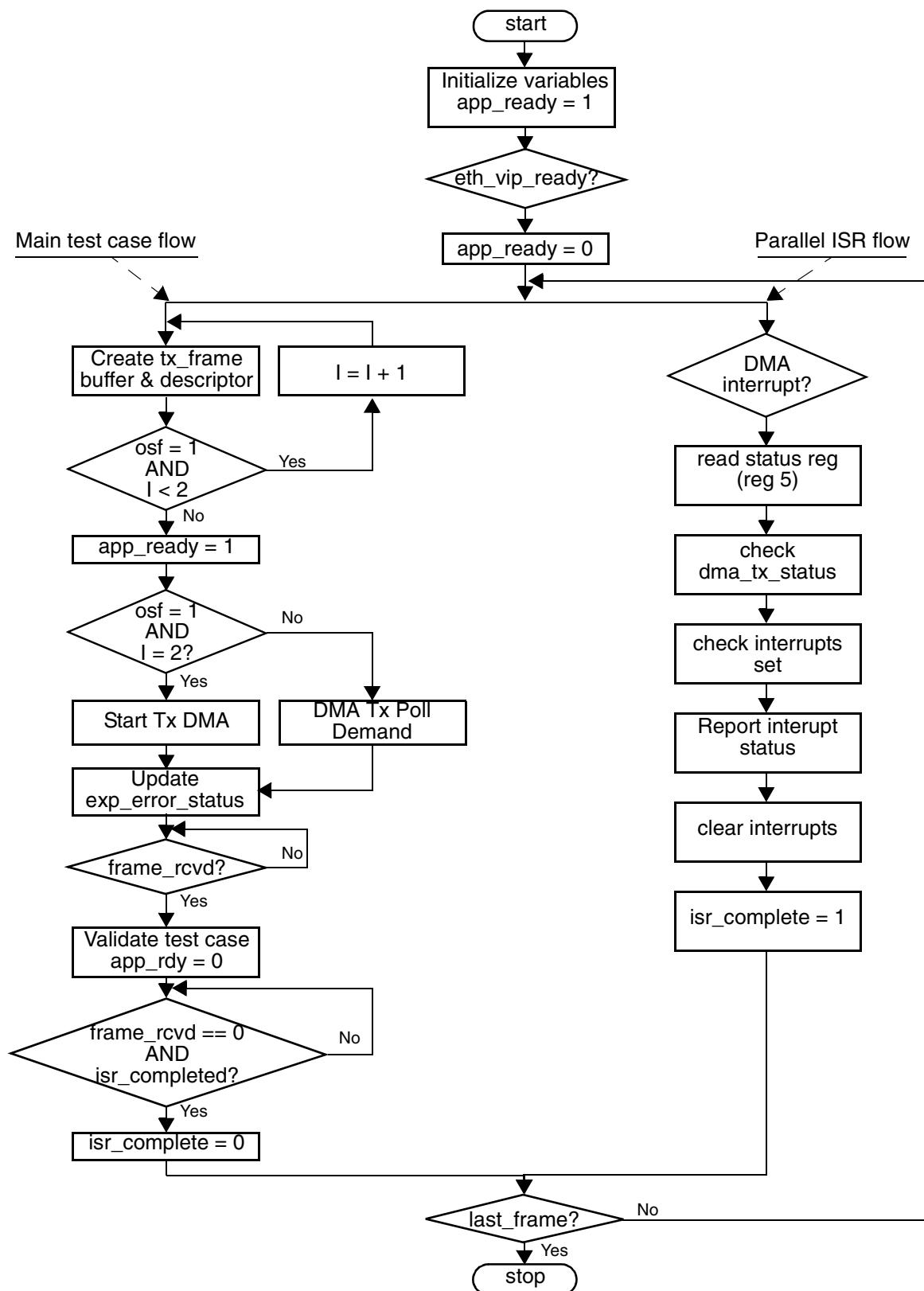
#### G.2.1.1 Application-Side Test Case Flow

The basic data flow at the host end for verification of the GMAC subsystem (transmission) is as depicted in [Figure G-2](#). The flow is as follows:

1. The application initiates the data transfer by programming the MAC, then initializing all the application-side variables used by the test case.
2. The application-side test case and the Ethernet-side test case perform a handshake to synchronize. The application-side test case sets the app\_ready bit in the GPIO, then reads the GPIO to check the eth\_vip\_ready bit's setting. The Ethernet-side test case sets the eth\_vip\_ready bit after it completes its initialization. The application then clears the app\_ready bit.
3. The application-side test case creates a transmit descriptor and sets the app\_ready bit in the GPIO to indicate that a frame is pending for transmission.
4. Program the DUT (set Start TxDMA or Tx Poll Demand) to start the frame transmission and initialize the expected transmission status in the internal exp\_error\_status variable.
5. The host waits for the Ethernet VIP to set the frame\_rcvd bit in the GPIO.
6. When the Ethernet VIP sets the frame\_rcvd bit, the application-side test case reads rcvd\_error\_status from the GPIO and compares it to exp\_error\_status.
7. The test case passes if exp\_error\_status and rcvd\_error\_status match, and if the Payload Check Status bit (exp\_data\_rcvd) in the GPIO is set.
8. The host clears the app\_ready bit and waits for the Ethernet VIP to clear the frame\_rcvd bit.
9. The host waits for the setting of the isr\_complete bit, then clears it before transmitting the next frame. This step synchronizes the main loop of the test case with the Interrupt Service Routine loop of the test case (shown in [Figure G-2](#) on page 445) for every frame transmitted.
10. If more frames are to be transmitted, the process returns to [Step 2](#), continuing through [Step 9](#).
11. If the OSF bit is set, the host creates two descriptors before transmitting the first frame.

In [Figure G-2](#) on page 445, a second thread or process (on the right side) is executed in parallel with the above steps. This is the emulation of the Interrupt Service Routine (ISR) in verilog. You must port the code given in the second thread to the ISR. The isr\_complete variable is used to handshake and synchronize the application-side testcase routines inside and outside of the ISR in this flow. This loop is triggered whenever the DUT asserts an interrupt. The ISR reads the status registers, checking whether the proper and expected interrupt bits are set, then clears the interrupt and waits for the next interrupt.

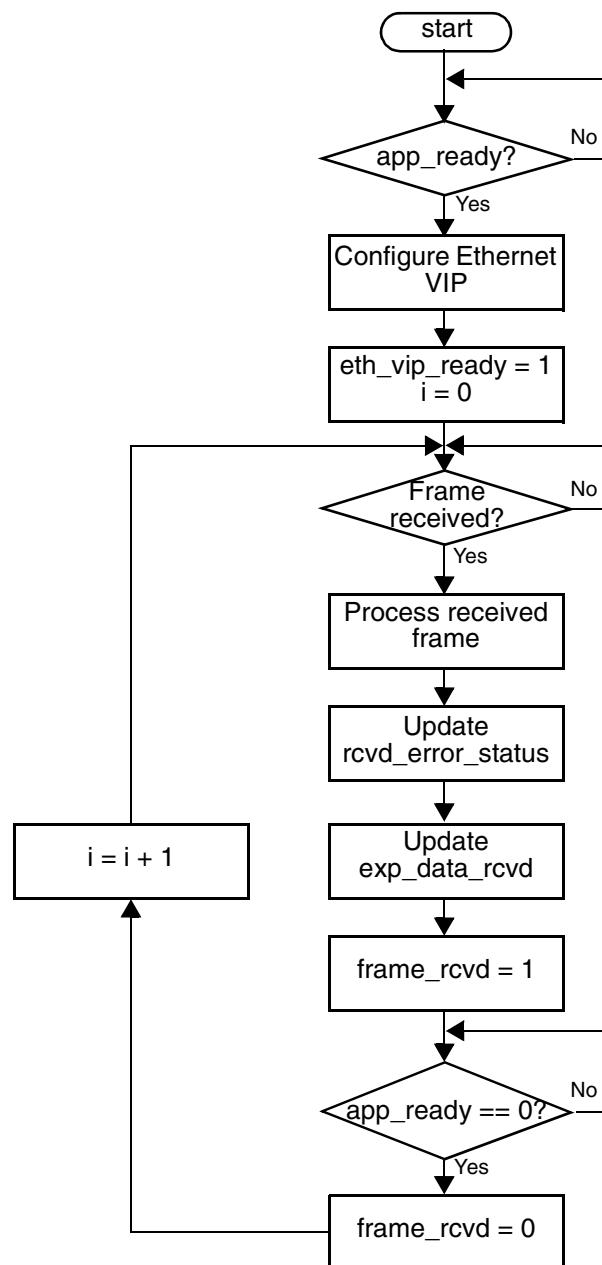
Similarly, the frame\_rcvd variable is used as handshake to synchronize the routines in the application-side test case and the Ethernet-side test case (as explained in "[Ethernet-Side Test Case Flow](#)" on page 446).

**Figure G-2 Application-Side Test Case Flow (Tx)**

### G.2.1.2 Ethernet-Side Test Case Flow

The basic test case flow at the Ethernet VIP (transmission) is as shown in [Figure G-3](#). The flow is as follows:

1. The Ethenet\_VIP test case checks for the app\_ready bit to be set
2. When the app\_ready bit is set, the test case configures the Ethernet VIP and sets the eth\_vip\_ready bit.
3. The Ethernet VIP waits for the frame. When it receives the frame, the test case processes it, checking for any errors the Ethernet VIP may have reported.
4. The corresponding error status is set in rcvd\_error\_status and exp\_data\_rcvd bit is set if there is no mismatch in the expected and received payload.
5. The frame\_rcvd bit is set, until the host clears the app\_ready bit in the GPIO.
6. The Ethernet VIP returns to [Step 3](#)

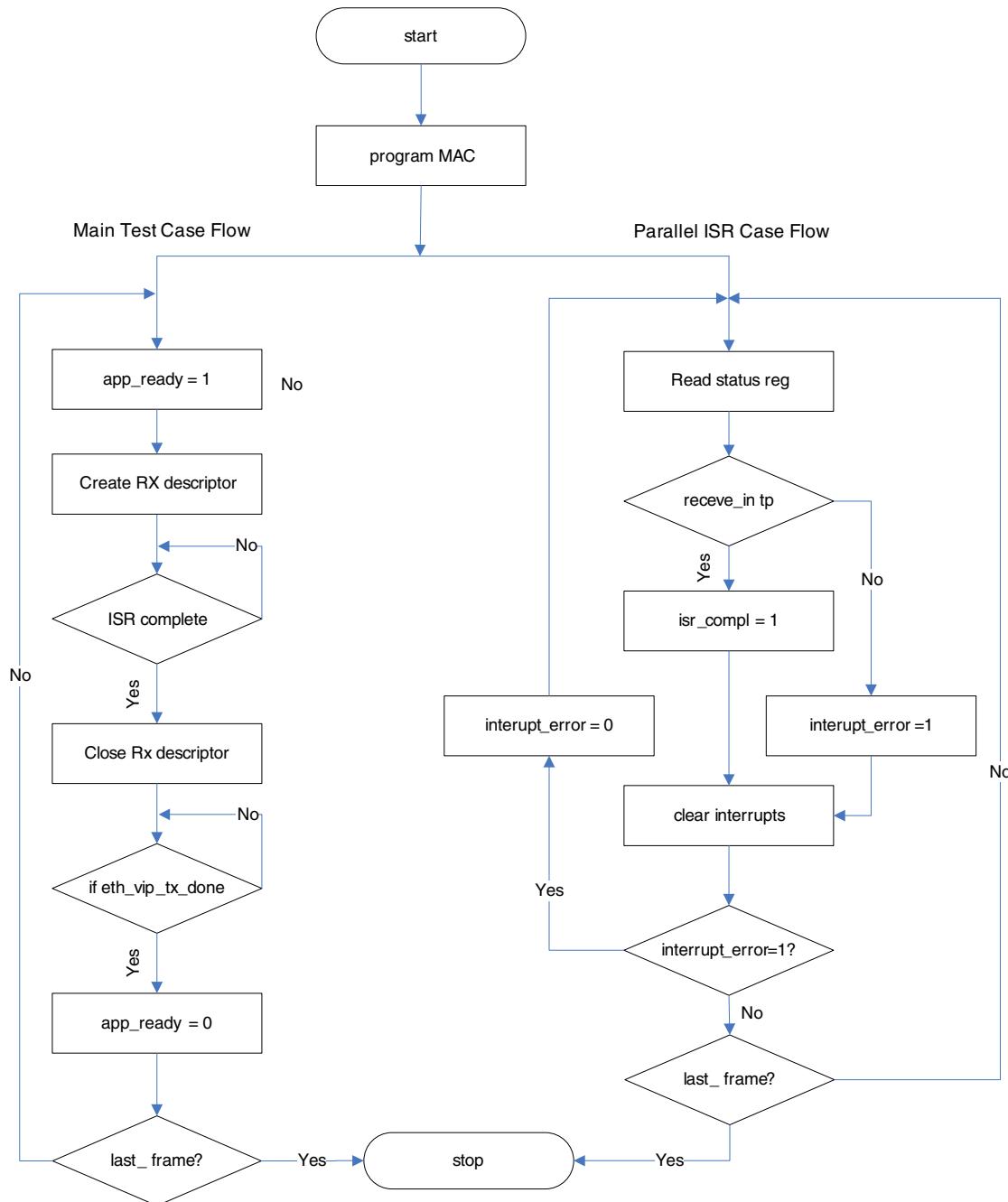
**Figure G-3 Ethernet-Side Test Case Flow (Tx)**

## G.2.2 Receive Verification Flow

### G.2.2.1 Application-Side Test Case Flow (Rx)

The basic test case flow at the host end (Rx) for the verification of GMAC subsystem is shown in [Figure G-4](#). A brief description of the flow follows.

**Figure G-4 Host Side Test Case Flow (Rx)**



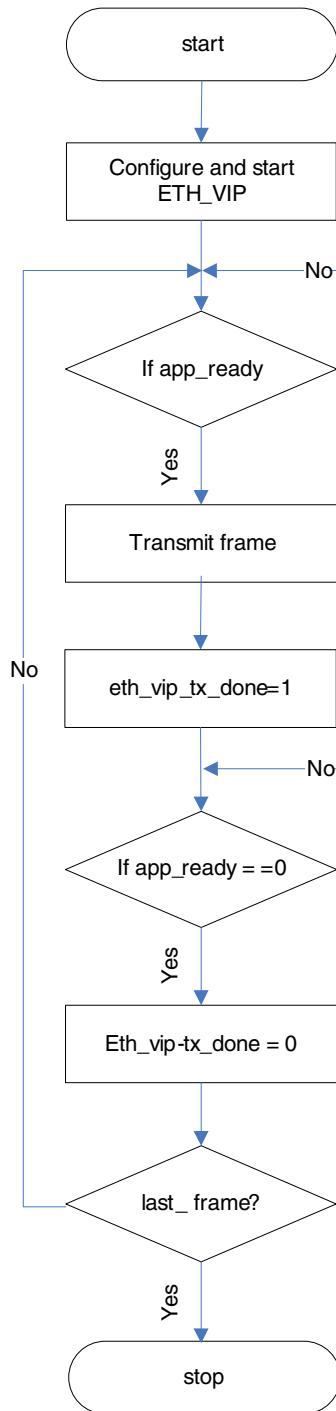
1. The host initializes the DUT by programming the MAC and initializing all the variables used by the test case.
2. The host creates a receive descriptor for the expected frame and then programs the RxDMA for proper operation, including enabling the appropriate interrupts.
3. The application-side test case and the Ethernet-side test case perform a handshake to synchronize its activities. The host sets the app\_ready bit in the GPIO to indicate to the Ethernet-side that it is ready to receive a frame.
4. It then waits for the frame by checking whether the isr\_complete flag is set. This flag is set in the parallel ISR after it completes the processing of the interrupt asserted due to a received frame.
5. The host then checks the Received status and frame data for errors.
6. The host resets the app\_ready bit in GPIO after it finds the vip\_tx\_done bit as high in the GPIO\_IN register. This is the handshake process to synchronizing both the sides of the testcase.
7. Steps 2 to 6 are repeated until all the expected frames are received.

In [Figure G-4](#), a second thread or process (on the right side) is executed in parallel with the above steps. This is the emulation of the Interrupt Service Routine (ISR) in verilog. You must port the code given in the second thread to the ISR. The isr\_complete variable is used to handshake and synchronize the application-side testcase routines inside and outside of the ISR in this flow. This loop is triggered whenever the DUT asserts an interrupt. The ISR reads the status registers, checking whether the proper and expected interrupt bits are set, then clears the interrupt and waits for the next interrupt.

### G.2.2.2 Ethernet-Side Test Case Flow (Rx)

The basic test case flow at the Ethernet VIP (Rx) is as shown in [Figure G-5](#). The flow is as follows.

1. The Ethernet VIP is configured to correct mode of operation and then started.
2. The test case waits for the assertion of app\_ready bit in GPIO.
3. Once the app\_ready bit is set, the Ethernet VIP transmits a frame of size equal to looping index K.
4. The testcase then synchronizes with the host-side testcase by first setting the vip\_tx\_frame bit and then waiting for the deassertion of app\_ready\_bit.
5. It then clears the vip\_tx\_frame bit input to the GPIO.
6. Repeat steps 2 to 6 until all the expected frames are transmitted.

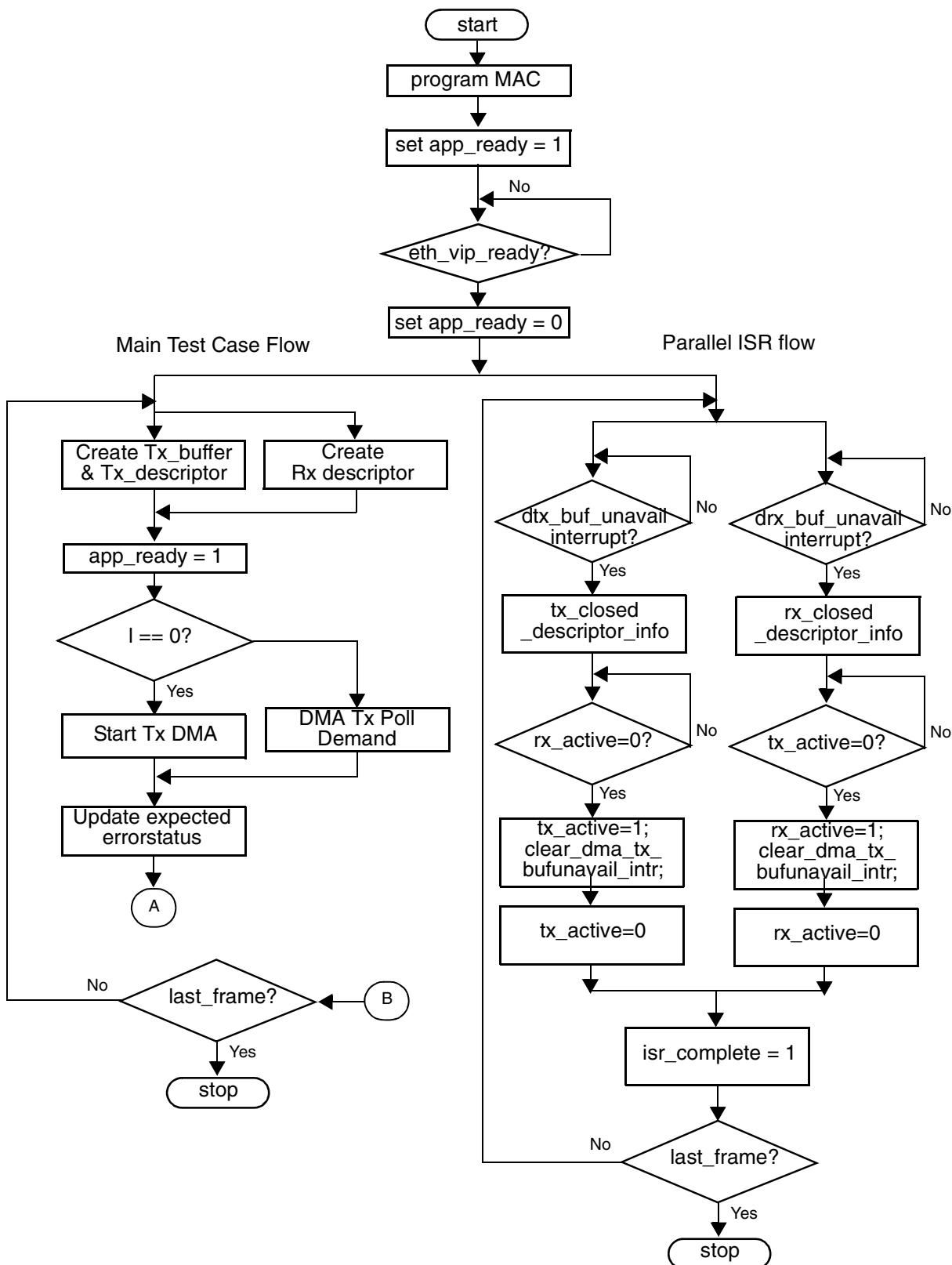
**Figure G-5 Ethernet Side Test Case Flow (Rx)**

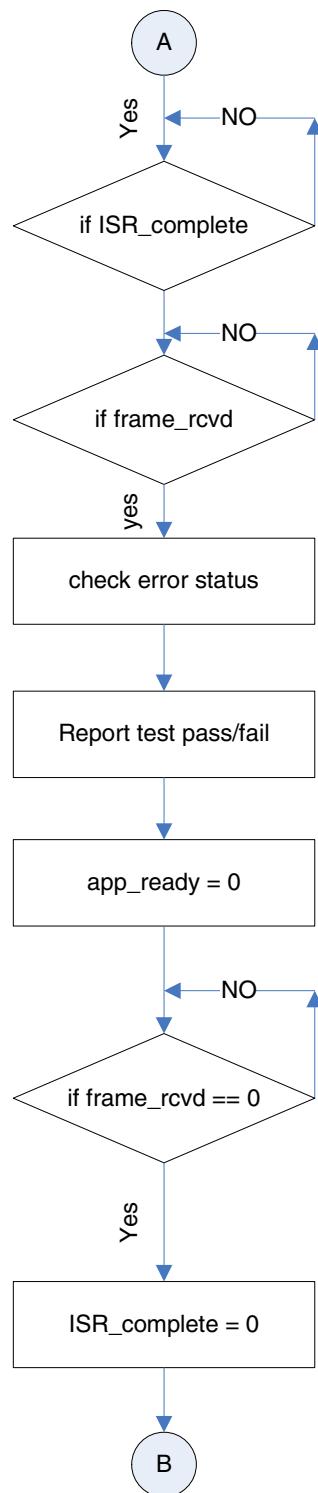
## G.2.3 Transmit-Receive Verification Flow

### G.2.3.1 Application-Side Test Case Flow (TxRx)

The basic data flow at the host end for the verification of GMAC subsystem when both transmission and reception are occurring simultaneously is as show in [Figure G-6](#) and [Figure G-7](#). A brief description of the flow follows.

1. The Application initiates the data transfer, by programming the MAC and initializing all the variables used by the test case.
2. Then the app\_ready bit in the GPIO is set, till the eth\_vip\_ready bit in the GPIO\_IN is set, then the app\_ready bit is cleared.
3. Then transmit descriptor and receive descriptor are created simultaneously and the app\_ready bit in the GPIO is set.
4. Then the frame is transmitted and the expected error status is stored in internal variable exp\_error\_status, at the same time the frame transmitted by the Ethernet VIP is received and processed.
5. Then host waits for isr\_complete bit.
6. The host then waits for the frame\_rcvd bit in the GPIO to be set by the eth\_vip.
7. When the Ethernet VIP sets the frame\_rcvd bit, the rcvd\_error\_status is read and compared with the exp\_error\_status.
8. The test-case is said to pass if frame has been received without any errors and for transmit frame exp\_error\_status and the rcvd\_error\_status matches and the exp\_data\_rcvd bit in the GPIO is set.
9. The host then clears the app\_ready bit and waits for the Ethernet VIP to clear the frame\_rcvd bit.
10. Then host clears the isr\_complete bit before transmitting the next frame.
11. If there are more frames to be transmitted, steps 2 to 9 are repeated.

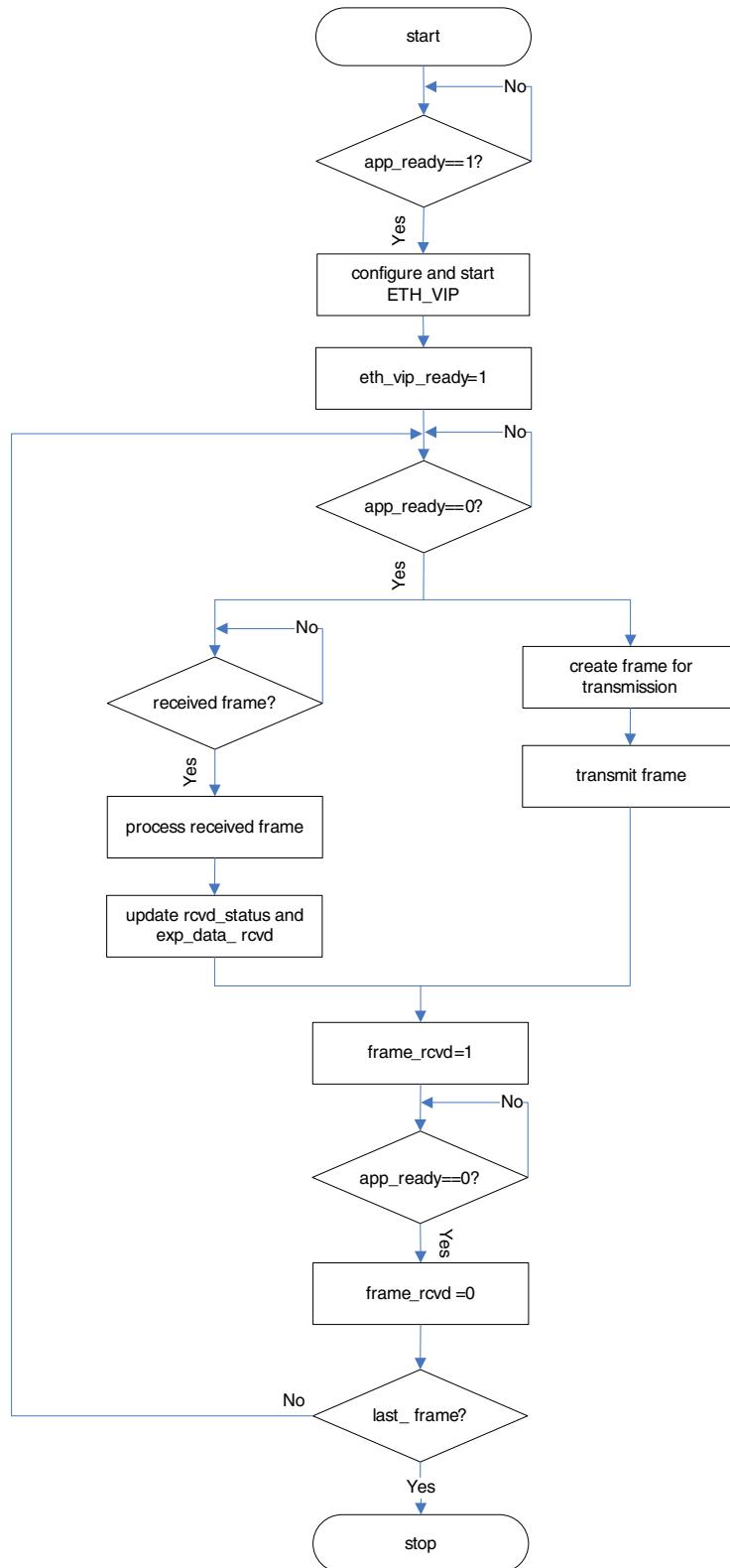
**Figure G-6 Application Side Test Case Flow (txrx) Part 1**

**Figure G-7 Application Side Test Case Flow (txrx) Part 2**

### G.2.3.2 Ethernet-side testcase flow (TxRx)

The basic test case flow at the Ethernet VIP (Tx and Rx) is as shown in [Figure G-8](#). An overview of the flow follows.

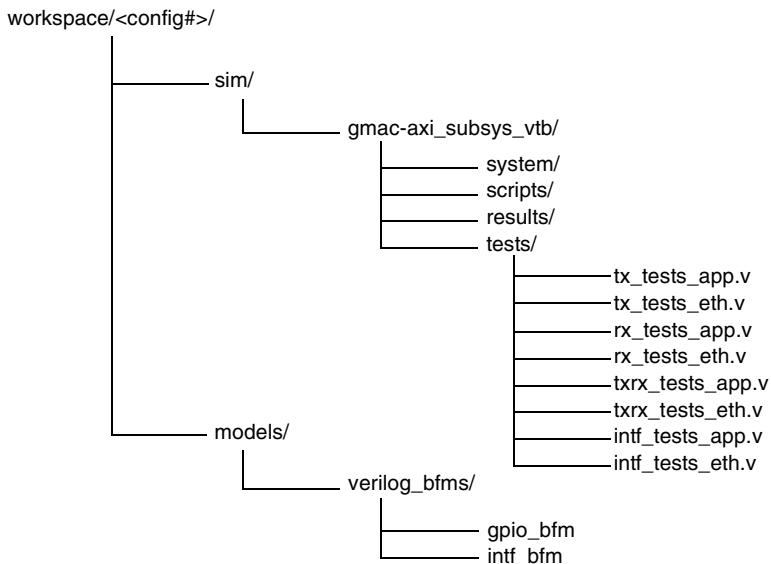
1. Ethenet\_VIP test case keep checking for the app\_ready bit to be set.
2. When app\_ready bit is set it configures the Ethernet VIP, and sets the eth\_vip\_ready bit.
3. Then Ethernet vip transmits a frame and also waits for reception of the frame transmitted by the DUT.
4. Ethernet VIP then processes the received frame.
5. The corresponding error status is set in rcvd\_error\_status and rcvd\_exp\_data bit is set if there is no mismatch in the expected and received payload.
6. Then the frame\_rcvd bit is set.
7. Then wait for the host to reset the app\_ready bit.
8. Then the frame\_rcvd bit is reset.
9. If there are more frames to be transmitted and received, steps 3 to 9 are repeated.

**Figure G-8 Ethernet VIP Side Test Case Flow (txrx)**

## G.3 Directory Structure

[Figure G-9](#) shows the organization of testbench files, BFM s, and test vectors for the gmac\_axi\_subsys\_vtb. The coreKit installation path is the root directory. The workspace/<config#> directory contains the configured RTL and the testbench. Other directories are explained [Table G-1](#).

**Figure G-9 gmac\_axi\_subsys\_vtb Directory Structure**



**Table G-1 gmac-ahb\_subsys\_vtb Directory Structure**

File Name	Description
sim	Simulation directory containing the testbench, test vectors, and scripts. All simulations are run from this directory
system	Directory containing the testbench, the clock generation module, and the Ethernet VIP tasks, host tasks, host commands, etc..
scripts	Directory containing scripts for running test vectors.
results	This directory's test_log subdirectory is a repository for simulation logs.
tests	This directory contains all the transmit and receive test vectors. Test case files with an “_app” suffix contain the application-side code. Test case files with an “_eth” suffix control the Ethernet VIP.
gpio_bfm	This directory contains a GPIO model that the application and Ethernet sides use to communicate.
intf_bfm	This consists of an interface file used for proper port mapping of the ports, depending on the selected interface.

## G.4 GPIO

The GPIO is used for communication between the host and the Ethernet VIP. The GPIO has two types of register sets for input ports (GPIO\_IN) and output ports (GPIO\_OUT). The register set is further classified by whether the test case controls frame transmission or reception.

The GPIO base address in the testbench is 0x0000\_C000. The GPIO BFM implements a set of sixteen 32-bit registers to control the I/O ports, with each register bit corresponding to a port bit. The Bit 5 address indicates whether the register is used by the transmit test case (Bit 5 = 0) or the receive test case (Bit 5 = 1). Similarly, Bit 4 of the address decodes whether the register controls the output ports (Bit 4 = 0) or the input ports (Bit 4 = 1).

The address mapping for GPIO is as follows:

**Table G-2** GPIO Address Map

Register	Address	Description
1	0000_C000	Transmission-side GPIO_OUT control register
2–4	0000_C004–0000_C00C	Reserved for transmission-side GPIO_OUT
5	0000_C010	Transmission-side GPIO_IN control register
6	0000_C014	Transmission-side GPIO_IN error_status [63:32]
7	0000_C018	Transmission-side GPIO_IN error_status [31:0]
8	0000_C01C	Reserved for Transmission-side GPIO_IN
9	0000_C020	Receive-side GPIO_OUT control register
10–12	0000_C024–0000_C02C	Reserved for Receive-side GPIO_OUT
13	0000_C030	Receive-side GPIO_IN control register
14–16	0000_C034	Reserved for Receive-side GPIO_IN

### G.4.1 GPIO\_OUT (Tx)

The GPIO\_OUT structure for transmission is shown in [Figure G-10](#).

**Figure G-10** GPIO\_OUT Structure (Tx)

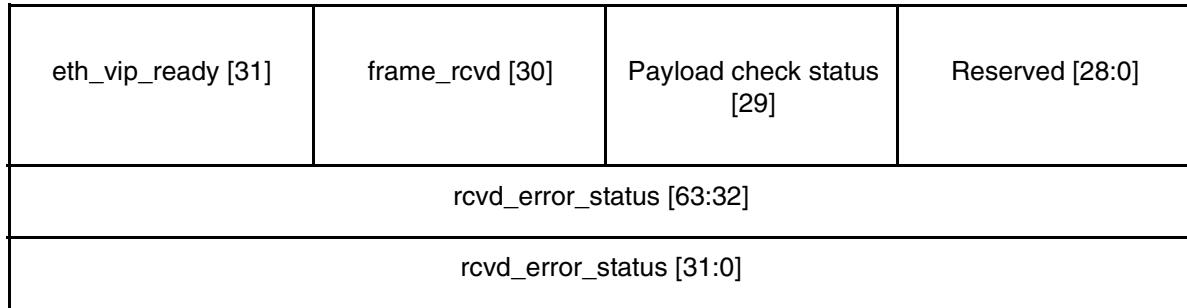


### G.4.2 GPIO\_IN (Tx)

The GPIO\_IN structure for transmission is shown in [Figure G-11](#), GPIO\_IN uses three 32-bit registers: one for the control variables, the other two for the error status received from the Ethernet VIP. GPIO\_IN fields are as follows:

- ❖ eth\_vip\_ready: This bit is set when the Ethernet VIP is ready to receive the frame.
- ❖ frame\_recieved: This bit is set when the Ethernet VIP has received the frame.

- ❖ Payload check status: The VIP sets this bit when it receives data that matches the expected data frame.
- ❖ rcvd\_error\_status[63:0]: The 64-bit error status generated by the Ethernet VIP for the received frame is stored in two 32-bit registers, as shown in [Figure G-11](#).

**Figure G-11** **GPIO\_IN Structure (Tx)**

#### **G.4.3      GPIO\_OUT (Rx)**

The GPIO\_OUT structure for transmission is shown in [Figure G-12](#).

**Figure G-12** **GPIO\_OUT Structure (Rx)**

#### **G.4.4      GPIO\_IN (Rx)**

As shown in [Figure G-13](#), the GPIO\_IN structure uses 32-bit registers of which it currently uses only the following port:

vip\_rd\_done: The Ethernet VIP sets this bit when it has read out GPIO\_OUT.

**Figure G-13** **GPIO\_IN Structure (Rx)**

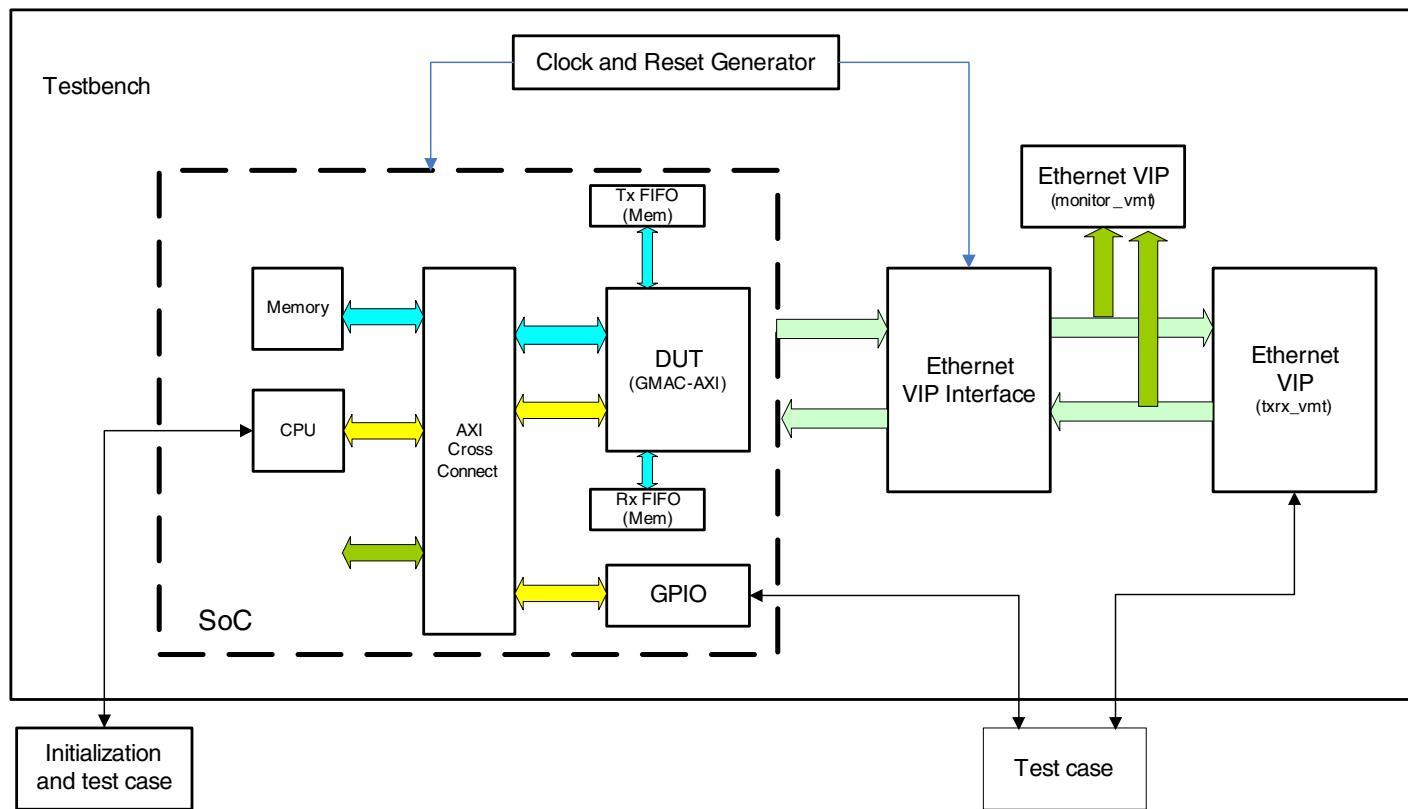
## G.5 SoC-Level Verification Guidelines

This section provides guidelines for SoC-level verification. The test environment is depicted in [Figure G-14](#). In this environment, the SoC, which contains an ARM core (CPU) is the DUT. The application-side vectors are now created in software and loaded into the memory for the CPU to execute.

The verilog testcases given in this testbench can be easily ported to create test cases that can be used to verify SoC integration. The verification strategy must adhere to the following guidelines.

- ❖ In SoC-level verification, you must ensure that all communication between the application-side test case and the Ethernet-side test case passes through the GPIO only.
- ❖ You must rewrite the application-side test case in a high-level language, such as C, and compile it to create machine code. Load this code into the system memory (instruction code memory) of the SoC, then start the simulation. The CPU reads the instructions from memory and executes the test as indicated in the flow charts provided in “[Verification Flow](#)” on page [444](#).
- ❖ The Ethernet VIP is controlled only with the existing Ethernet-side test case in verilog.
- ❖ You can use the reserved fields in the GPIO to pass control data between the application and the Ethernet VIP.

**Figure G-14** Setup for SoC-Level Verification



## G.6 Running Simulations

To run the individual test case simulations from the command line, enter the following commands from the configured corekit's <workspace>/sim/ directory:

- ❖ For VCS enter:

```
./gmac_axi_subsys_vtb/scripts/run_vcs <TEST_CASE_NAME> <OTHER_OPTIONS>
```

- ❖ For NCV enter:

```
./gmac_axi_subsys_vtb/scripts/run_ncv <TEST_CASE_NAME> <OTHER_OPTIONS>
```

- ❖ For MTI enter:

```
./gmac_axi_subsys_vtb/scripts/run_mti <TEST_CASE_NAME> <OTHER_OPTIONS>
```

## G.7 Simulation PASS/FAIL

The simulation run status is logged into the corekit:<workspace>/sim/runtest.log file. The details of the run can be accessed from the results/testlog/<TEST\_NAME>.log file at the end of simulation.

**H****Area and Power****H.1 Area****H.1.1 Gate Count Summary**

This section summarizes the area in terms of gate count for the various modules/configurations listed in [Table H-1](#). Gate count was calculated with the following parameters:

DC Version:	2008.09
Technology synthesized for:	45n, 65n, 90n, 130n
Application clock (10/100 MAC only):	100 MHz
Application clock (Gigabit MAC):	167 MHz to 250 MHz
MII/GMII clock:	25/125 MHz
Gate size:	NAND2x1
Data bus width:	32 bits

**Table H-1 Gate Count**

Sr No.	Blocks	Gate Count
1	MAC core (10/100 only)	13–15K
2	GMAC Core (1000 only)	14–17K
3	GMAC Core (10/100/1000)	14–17K
4	Disable Half Duplex	~(2K)
5	Disable Hash Filter	~(0.8K)
6	MMC (RMON with all MAC counters) module MMC (RMON with all counters) module	26–28K 43–46K
7	PMT (Power Mgmt) module	7–8K

**Table H-1 Gate Count (Continued)**

Sr No.	Blocks	Gate Count
8	Opt MAC Address Reg (31)	18–19K
9	Dbl-Sync Address Reg (all 32)	18–19K
10	IP Checksum module (Type 1) Receive Checksum Offload (Type 2)	~ 1K ~3K
11	SMA module	~ 0.9K
12	Dbl-Sync Hash/VLAN/PT registers	~ 1K
13	RGMII	~ 0.35K
14	RMII	~ 0.5K
15	SMII	~0.7K
16	TBI + RTBI + SGMII	5–8K
17	MTL (default FIFO layer)	6.5–8K
18	TxCOE (Checksum Offload Engine)	~8K
19	MDC (DMA) module	11–14K
20	IEEE-1588 TimeStamping - Default with external time reference(GMAC-AHB)	~4.5K
21	IEEE-1588 TimeStamping - Advanced with external time reference(GMAC-AHB)	~6K
22	IEEE-1588 TimeStamping - with internal time reference (add to above)	~5K
23	IEEE-1588 TimeStamping - Axillary Snapshot feature	~3K
25	EEE-1588 TimeStamping - Advanced - Higher16-bit for seconds time	~200
26	Separate CSR clock in DMA	~4.5K
28	AHB (interface) module	3–4K
29	AXI Master interface	7-8K
30	AXI Slave interface	~3K
31	MAC (10/100) Subsystem with AHB (Min config)	~ 35K
32	Gigabit MAC Subsystem with AHB (Min config)	~ 37K
33	Gigabit MAC Subsystem with AXI (Max config = Enable all)	~175K

As noted, [Table H-1](#) gives the gate count numbers in 32-bit data bus configuration. The increase in approximate area due to a 64-bit or 128-bit data bus configuration (in affected blocks only, with respect to the numbers given in [Table H-1](#)) is given in [Table H-2](#) below.

**Table H-2 Area Increases in 64- and 128-Bit Data Bus Configurations**

Data Bus	Module	Gate Count
64-bit	AHB (interface)	1K
	AXI Master interface	3K
	AXI Slave interface	1K
	MDC (DMA) module	2K
	MTL (default FIFO layer)	2K
	MTL (with Tx checksum offload)	6K
128-bit	GMAC core	0.8K
	AHB (interface)	3K
	AXI Master interface	9K
	AXI Slave interface	3K
	MDC (DMA) module	7K
	MTL (default FIFO layer)	6K
	MTL (with Tx checksum offload)	16K
	GMAC core	2.5K

With all features enabled, the Gigabit MAC Subsystem with AXI has approximately 190K gates in 64-bit data bus configuration and 210K gates in 128-bit data bus configuration.

### H.1.2 Area Differences Due to Scan Ready and Clock Gating

The area differences due to scan ready and clock gating for the two configurations described below are shown in [Table H-4](#).

**Table H-3** Sample Configuration for Estimating Area

Main Parameter	Configuration 1	Configuration 2
Core Features	<ul style="list-style-type: none"> <li>• 10/100 Mbps only</li> <li>• Async Reset</li> <li>• System interface = AHB</li> <li>• 32-bit, little endian</li> <li>• MTL Rx/Tx FIFO = 2K</li> <li>• PHY IF = MII only</li> <li>• 1 MAC address</li> <li>• Optional modules = SMA</li> </ul>	<ul style="list-style-type: none"> <li>• 10/100/1000 Mbps</li> <li>• Async Reset</li> <li>• System interface = AHB</li> <li>• 32-bit, little endian</li> <li>• MTL Rx/Tx FIFO = 2K</li> <li>• PHY IF = GMII only</li> <li>• 1 MAC address</li> <li>• Optional modules = SMA</li> </ul>
Technology	TSMC 90G	TSMC 90G
Gate Size	NAND2X1 cells (area 2.42)	NAND2X1 cells (area 2.42)
Application Clock Frequency	100 MHz	200 MHz
PHY Clock Frequency	25 MHz	125 MHz

**Table H-4** Area Differences Due to Scan Ready and Clock Gating

Clock Gating	Percentage Gating	Scan Ready	Test Coverage	Gate Count
<b>Configuration 1</b>				
No	NA	No	N/A	39,850
No	NA	Yes	99.35%	46,504
Yes	78.7	No	N/A	34,687
Yes	78.7	Yes	99.51%	40,840
<b>Configuration 2</b>				
No	NA	No	N/A	41,158
No	NA	Yes	99.36%	48,030
Yes	78.9	No	N/A	35,783
Yes	78.9	Yes	99.48%	42,159

Tetramax, version 2008.09, was used for test coverage estimation.

## H.2 Power

Power dissipation estimates with and without clock gating are shown in [Table H-6](#). The power estimates are for simulations with continuous transmission and reception of about 100 frames each, at the Application clock frequency given below.

**Table H-5 Sample Configuration for Estimating Power Dissipation**

Main Parameter	Configuration 1	Configuration 2
Core Features	<ul style="list-style-type: none"> <li>• 10/100 Mbps only</li> <li>• Async Reset</li> <li>• System interface = AHB</li> <li>• 32-bit, little endian</li> <li>• MTL Rx/Tx FIFO = 2K</li> <li>• PHY Interface = MII only</li> <li>• 1 MAC address</li> <li>• Optional modules = SMA</li> </ul>	<ul style="list-style-type: none"> <li>• 10/100/1000 Mbps</li> <li>• Async Reset</li> <li>• System interface = AHB</li> <li>• 32-bit, little endian</li> <li>• MTL Rx/Tx FIFO = 2K</li> <li>• PHY Interface = GMII only</li> <li>• 1 MAC address</li> <li>• Optional modules = SMA</li> </ul>
Technology	TSMC 90G	TSMC 90G
Global Operating Voltage	0.9 volts	0.9 volts
Gate Size	NAND2X1 cells (area 2.42)	NAND2X1 cells (area 2.42)
Application Clock Frequency	100 MHz	200 MHz
PHY Clock Frequency	25 MHz	125 MHz

**Table H-6 Power Dissipation with and without Clock Gating**

Clock Gating	Gate Count	Comb / Seq Counts	Dyn Power (mW)	Static Power (mW)
<b>Configuration 1</b>				
No	39,848	8077 / 3115	1.56	0.344
Yes	34,687	8395 / 3303	0.564	0.313
<b>Configuration 2</b>				
No	41,158	8499 / 3204	2.953	0.354
Yes	35,783	8829 / 3396	1.214	0.320

Prime Power, version 2008.09, was used for power estimation.



# Programming Guide

This appendix provides instructions for initializing the DMA/MAC registers in the proper sequence. Abbreviations are used for the register names and their fields. Refer to “[Registers](#)” on page [247](#) for detailed information on registers.

## I.1 DMA Initialization - Descriptors

The following operations must be performed to initialize the DMA. This initialization sequence is used for GMAC-AHB, GMAC-AXI and GMAC-DMA configurations only. For the GMAC-AXI configuration, AXI-specific control register also must be programmed.

1. Provide a software reset. This will reset all of the GMAC internal registers and logic. (DMA Register0 Bus mode register – bit 0).
2. Wait for the completion of the reset process (poll bit 0 of the DMA Register0 Bus Mode Register, which is only cleared after the reset operation is completed).
3. Program the following fields to initialize the Bus Mode Register by setting values in DMA Register0 (Bus Mode Register)
  - a. Mixed Burst and AAL (only if GMAC-AHB/GMAC-AXI configuration is selected)
  - b. Fixed burst or undefined burst
  - c. Burst length values and burst mode values.
  - d. Descriptor Length (only valid if Ring Mode is used)
  - e. Tx and Rx DMA Arbitration scheme
4. Program the AXI Interface options in the AXI Bus Mode register
  - a. If fixed burst-length is enabled, then select the maximum burst-length possible on the AXI bus (Bits[7:1])
5. A proper descriptor chain for transmit and receive must be created. It should also ensure that the receive descriptors are owned by DMA (bit 31 of descriptor should be set). Refer to “[Descriptors](#)” on page [337](#) for more information. When OSF mode is used, at least two descriptors are required.
6. Software should create three or more different transmit or receive descriptors in the chain before reusing any of the descriptors.
7. Initialize receive and transmit descriptor list address with the base address of transmit and receive descriptor (DMA Register 3 – Receive Descriptor List Address Register and 4 – Transmit Descriptor List Address register, respectively).

8. Program the following fields to initialize the mode of operation by setting values in DMA Register6 (Operation Mode Register)
  - a. Receive and Transmit Store And Forward
  - b. Receive and Transmit Threshold Control (RTC and TTC)
  - c. Hardware Flow Control enable
  - d. Flow Control Activation and De-activation thresholds for MTL Receive and Transmit FIFO (RFA and RFD)
  - e. Error Frame and undersized good frame forwarding enable
  - f. OSF Mode
9. Clear the interrupt requests, by writing to those bits of the status register (interrupt bits only) which are set. For example, by writing 1 into bit 16 – normal interrupt summary will clear this bit (DMA Register5 – Status Register).
10. Enable the interrupts by programming the interrupt enable register (DMA Register7 – Interrupt enable register).
11. Start the Receive and Transmit DMA by setting SR (bit 1) and ST (bit 13) of the control register (DMA Register6 – operation mode register).

## I.2 MAC Initialization

The following MAC Initialization operations can be performed after the DMA initialization sequence. If the MAC Initialization is done before the DMA is set-up, then enable the MAC receiver (last step below) only after the DMA is active. Otherwise, received frames will fill the RxFIFO and overflow. Steps (1) and (2) are to be followed if the TBI/SGMII/RTBI PHY interface is enabled, otherwise follow steps (3) and (4).

1. Program the AN Control register (GMAC Register48-AN Control Register) to enable Auto-negotiation ANE (bit-12). Setting ELE (bit-14) of this register will enable the PHY to loop back the transmit data and RAN (bit-9) can be set to restart Auto negotiation (for TBI/SGMII/RTBI PHY interfaces only).
2. Check the AN Status Register (GMAC Register49-AN Status Register) for completion of the Auto-negotiation process. ANC (bit-5) should be set, and link status (bit-2), when set, indicates that the link is up (For TBI/SGMII/RTBI PHY interfaces only).
3. Program the (GMAC Register4 – GMII Address Register) for controlling the management cycles for external PHY, for example, Physical Layer Address PA (bits 15-11). Also set bit 0 (GMII Busy) for writing into PHY and reading from PHY.
4. Read the 16-bit data of (GMAC Register5 – GMII Data Register) from the PHY for link up, speed of operation, and mode of operation, by specifying the appropriate address value in GMAC Register4 (bits 15-11).
5. Provide the MAC address registers (GMAC Register16 for MAC address 0 high register and GMAC Register17 for MAC address 0 low register). If more than one MAC address is enabled in your configuration (during configuration in coreConsultant), program them appropriately.
6. If Hash filtering is enabled in your configuration, program the Hash filter register (GMAC Register2 – Hash table high register and GMAC Register3 – Hash table low register).
7. Program the following fields to set the appropriate filters for the incoming frames in (GMAC Register1 – MAC frame filter)

- a. Receive All
  - b. Promiscuous mode
  - c. Hash or Perfect Filter
  - d. Unicast, Multicast, broad cast and control frames filter settings etc.
8. Program the following fields for proper flow control in (GMAC Register6 – Flow Control Register).
    - a. Pause time and other pause frame control bits
    - b. Receive and Transmit Flow control bits
    - c. Flow Control Busy/Backpressure Activate
  9. Program the Interrupt Mask register bits, as required, and if applicable, for your configuration.
  10. Program the appropriate fields in (GMAC Register0 – MAC configuration register) for example, Inter-frame gap while transmission, jabber disable, etc. Based on the Auto-negotiation you can set the Duplex mode (bit 11), port select (bit 15), etc.
  11. Set the bits Transmit enable (TE bit-3) and Receive Enable (RE bit-2) in (GMAC Register0 – MAC configuration register).

### I.3 Normal Receive and Transmit Operation

For normal operation, the following steps can be followed.

- ❖ For normal transmit and receive interrupts, read the interrupt status. Then poll the descriptors, reading the status of the descriptor owned by the Host (either transmit or receive).
- ❖ On completion of the above step, set appropriate values for the descriptors, ensuring that transmit and receive descriptors are owned by the DMA to resume the transmission and reception of data.
- ❖ If the descriptors were not owned by the DMA (or no descriptor is available), the DMA will go into SUSPEND state. The transmission or reception can be resumed by freeing the descriptors and issuing a poll demand by writing 0 into the Tx/Rx poll demand register (DMA Register1 – Transmit Poll Demand Register and DMA Register2 – Receive Poll Demand Register).
- ❖ The values of the current host transmitter or receiver descriptor address pointer can be read for the debug process (DMA Register18 – Current Host Transmit Descriptor Register and DMA Register19 – Current Host Receive Descriptor Register).
- ❖ The values of the current host transmit buffer address pointer and receive buffer address pointer can be read for the debug process (DMA Register20 – Current Host Transmit Buffer Address Register and DMA Register21 – Current Host Receive Buffer Address Register).

### I.4 Stop and Start Operation

When the transmission is required to be paused for some time then the following steps can be followed.

1. Disable the Transmit DMA (if applicable), by clearing ST (bit 13) of the control register (DMA Register6 – Operation Mode Register).
2. Wait for any previous frame transmissions to complete. This can be checked by reading the appropriate bits of MAC Debug register.
3. Disable the MAC transmitter and MAC receiver by clearing the bits Transmit enable (TE bit-3) and Receive Enable (RE bit-2) in GMAC Register0 (MAC configuration register).

4. Disable the Receive DMA (if applicable), after making sure the data in the RX FIFO is transferred to the system memory (by reading the MAC Debug register).
5. Make sure both the TX FIFO and RX FIFO are empty.
6. To re-start the operation, start the DMAs first, before enabling the MAC Transmitter and Receiver.

## I.5 Programming Guideline for IEEE 1588 Time Stamping

### I.5.1 Initialization Guideline for System Time Generation

The time-stamping feature can be enabled by setting Bit 0 of the Time Stamp control register. However, it is essential that the Time Stamp counter be initialized *after* this bit is set, starting time stamp operation. Do this by following these steps during GMAC core initialization:

1. Mask the Time Stamp Trigger interrupt by setting Bit 9 of Register 15.
2. Program Time Stamp register Bit 0, enabling time stamping.
3. Program the Sub-Second Increment register based on the PTP clock frequency.
4. If you are using the Fine Correction approach, program the Time Stamp Addend register and set Time Stamp Control register Bit 5 (addend reg update).
5. Poll the Time Stamp Control register until this Bit 5 is cleared.
6. To select the Fine Update method (if required), program Time Stamp Control register Bit 1.
7. Program the Time Stamp High Update and Time Stamp Low Update registers with the appropriate time value.
8. Set Time Stamp Control register Bit 2 (Time Stamp Init).
9. The Time Stamp counter starts operation as soon as it is initialized with the value written in the Time Stamp Update register.
10. Enable the MAC receiver and transmitter for proper time stamping.



If time stamp operation is disabled by clearing Bit 0 of the TS Control register, the above steps must be repeated to restart time stamp operation.

### I.5.2 System Time Correction

To synchronize or update the system time in one process (coarse correction method), perform the following steps:

1. Write the offset (positive or negative) in the Time Stamp Update registers (registers 452 and 453).
2. Set Bit 3 (TSUPDT) of the Time Stamp Control register (Register 448).
3. The value in the Time Stamp Update registers is added to or subtracted from the system time when the TSUPDT bit is cleared.

To synchronize or update the system time to reduce system-time jitter (fine correction method), perform the following steps:

1. With the help of the algorithm explained in “[System Time Register Module](#)” on page [128](#), calculate the rate by which you want to make the system time increments slower or faster.

2. Update the Time Stamp Addend register with the new value, then set Time Stamp Control register bit 5 (TSADDUPDT).
3. Wait for the time you want the new value of the Addend register to be active. You can do this by activating the Time Stamp Trigger interrupt after the system time reaches the target value.
4. Program the required target time in registers 455 and 456. Unmask the Time Stamp interrupt by clearing Bit 9 of Register 15
5. Set Time Stamp Control register bit 4 (TSTRIG)
6. When this trigger causes an interrupt, read Register 14.
7. Reprogram the Time Stamp Addend register with the old value and set Bit 5 again.



**J**

# Synchronizer Methods

---

This appendix documents the synchronizer methods (blocks of synchronizer functionality) used in DesignWare Ethernet Mac Universal IP to cross clock boundaries.

Note that the DesignWare Building Blocks (DWBB) contain several synchronizer components with functionality similar to methods documented in this appendix. For more information about DWBB synchronizer components, refer to this page:

[www.synopsys.com/products/designware/docs/doc/dwf/datasheets/interface\\_cdc\\_overview.pdf](http://www.synopsys.com/products/designware/docs/doc/dwf/datasheets/interface_cdc_overview.pdf)

This appendix documents the following synchronizer methods:

[“Synchronizer 1: Simple Double Register Synchronizer \(DWC\\_gmac\\_bcm21.v\)”](#) on page [474](#)

[“Synchronizer 2: Pulse Synchronizer \(DWC\\_gmac\\_bcm22.v\)”](#) on page [476](#)

[“Synchronizer 3: Pulse Synchronizer with Acknowledge \(DWC\\_gmac\\_bcm23.v\)”](#) on page [477](#)

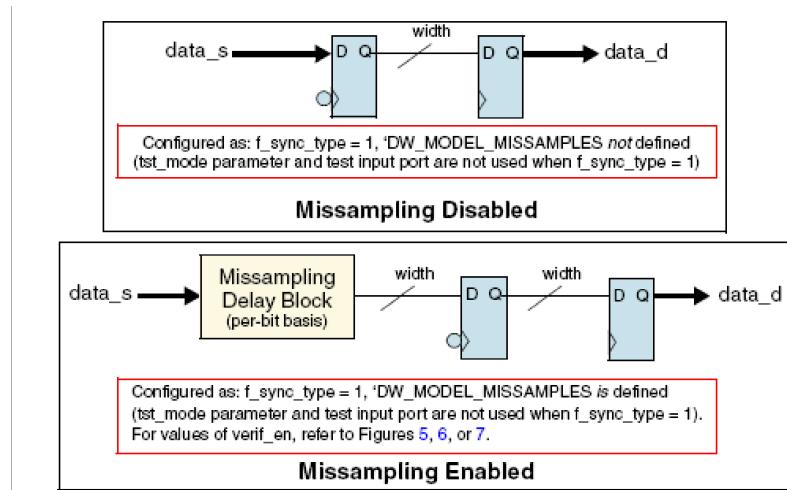
[“Synchronizer 4: Gray Coded Synchronizer \(DWC\\_gmac\\_bcm24.v\)”](#) on page [478](#)

A few more components like Asynchronous register FIFO (DWC\_gmac\_async\_fifo) and bus synchronizers (DWC\_gmac\_bus\_synczr) are used in this core. These components are built with some of the above synchronizers for the CDC signals.

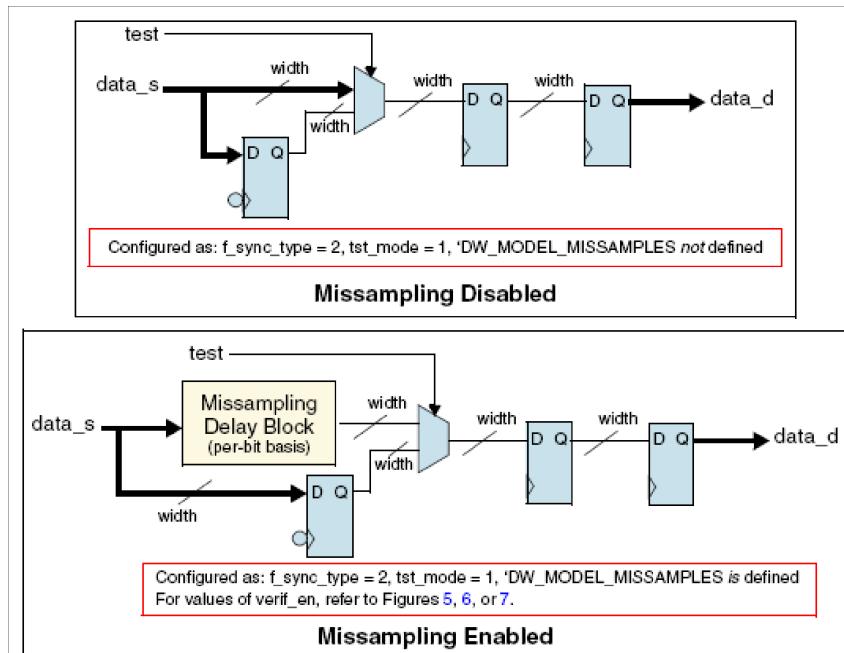
## J.1 Synchronizer 1: Simple Double Register Synchronizer (DWC\_gmac\_bcm21.v)

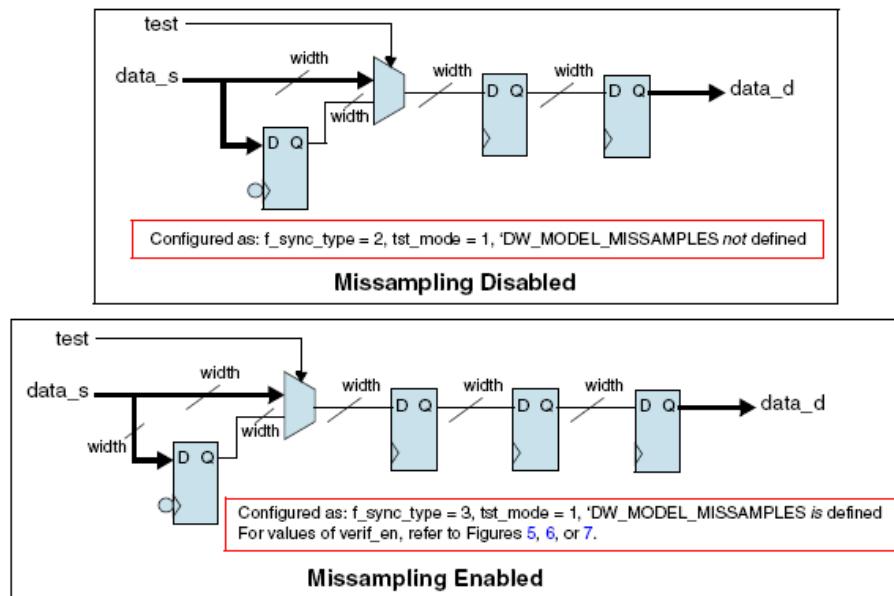
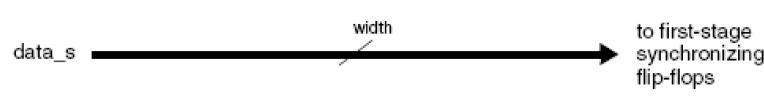
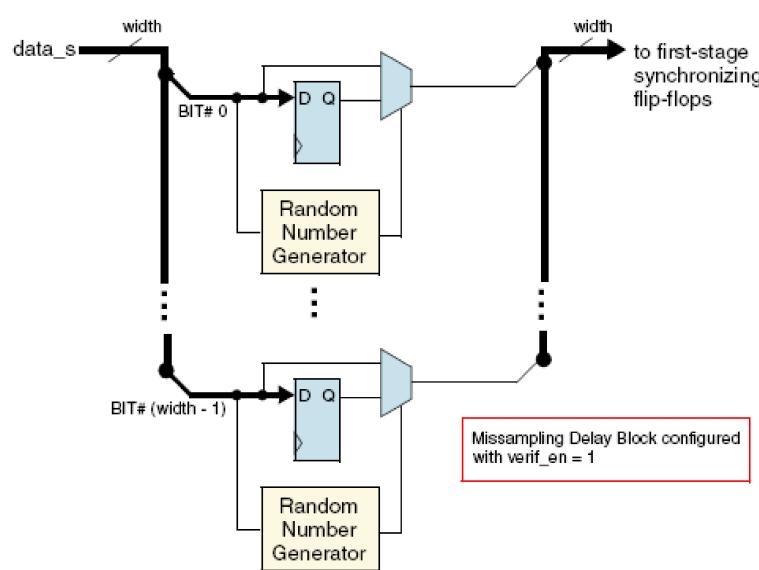
This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries. Your core may have parameters that allow you to configure for the number of synchronizing stages (2 or 3), the style of first-stage capturing flip-flop needed (negative or positive edge-triggered), and insertion of 'hold latches' to facilitate scan testing.

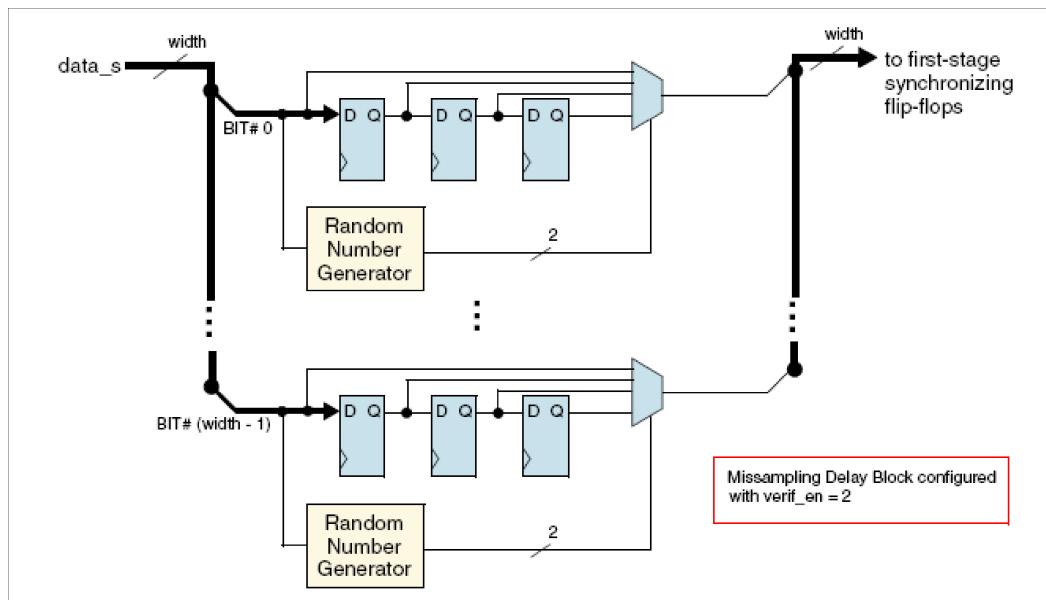
**Figure J-1 Synchronizer 1: Synchronization Type 1**



**Figure J-2 Synchronizer 1: Synchronization Type 2**

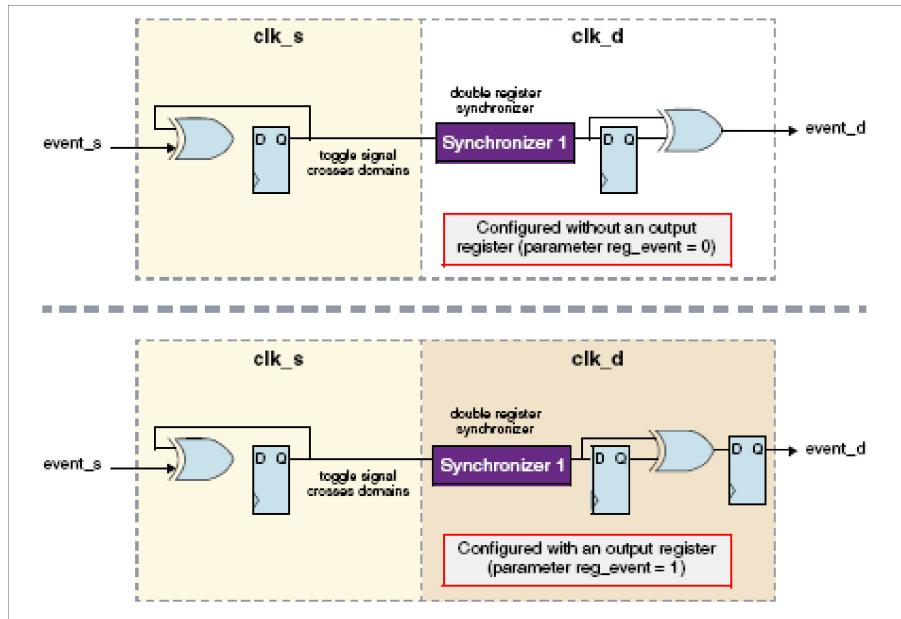


**Figure J-3 Synchronizer 1: Synchronization Type 3****Figure J-4 Synchronizer 1: Missampling Model Type 0****Figure J-5 Synchronizer 1: Missampling Model Type 1**

**Figure J-6 Synchronizer 1: Missampling Model Type 1**

## J.2 Synchronizer 2: Pulse Synchronizer (DWC\_gmac\_bcm22.v)

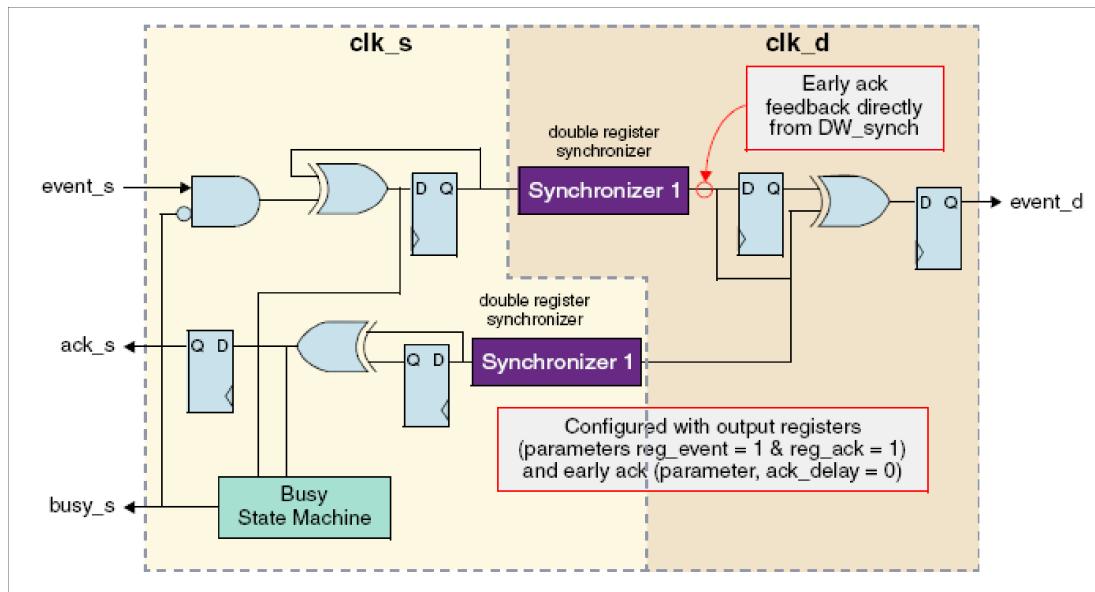
This is a dual clock pulse synchronizer for transmitting single clock cycle pulses between two different clock domains. This method uses clock domain crossing techniques to safely transfer pulses between logic operating on different clocks.

**Figure J-7 Synchronizer 2 Block Diagram**

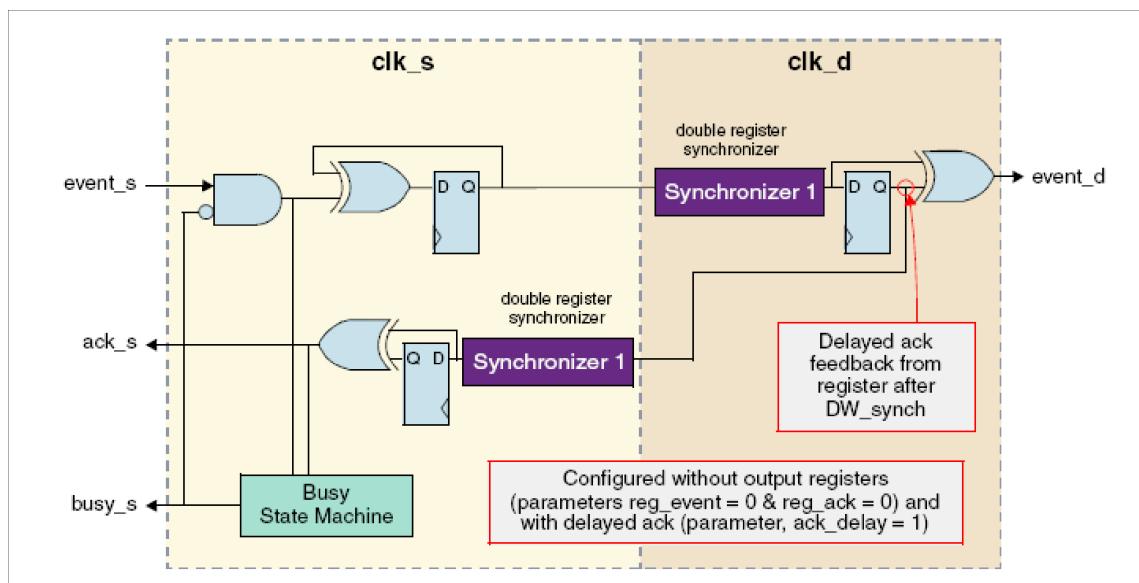
### J.3 Synchronizer 3: Pulse Synchronizer with Acknowledge (DWC\_gmac\_bcm23.v)

This synchronizer passes an event (represented by a single clock cycle pulse) from the source domain to the destination domain and passes an acknowledge. This synchronizer is used within other synchronizers to implement hand-shake mechanisms. It is constructed with feedback and an interlock.

**Figure J-8 Synchronizer 3 (with Output Registers)**



**Figure J-9 Synchronizer 3 (without Output Registers)**



## J.4 Synchronizer 4: Gray Coded Synchronizer (DWC\_gmac\_bcm24.v)

This synchronizer includes a binary counter in the source domain whose value is recoded as a binary-reflected-Gray code. The recoded count value is synchronized to the destination domain and then decoded from binary-reflected-Gray back to binary. The binary count value and an offset version (which is only useful for non-integer power-of-two sequences) are both available in the clk\_s domain as count\_s and offset\_count\_s. This synchronizer is used within other synchronizers to pass incrementing pointers across domains.

**Figure J-10 Synchronizer 4 Block Diagram**

