LogiCORE IP RXAUI v2.4

Product Guide

PG083 October 16, 2012

XILINX®

Table of Contents

Appendix A: Verification, Compliance, and Interoperability

Appendix B: Migrating

Appendix C: Debugging

Appendix D: Additional Resources

SECTION I: SUMMARY

Introduction

The LogiCORE™ IP RXAUI core is a high-performance, low pin count 10 Gb/s interface intended to allow physical separation between the data-link layer and physical layer devices in a 10 Gb Ethernet system.

The RXAUI core implements a single-speed full-duplex 10 Gb/s Ethernet Reduced Pin eXtended Attachment Unit Interface (RXAUI) solution for Xilinx® 7 series and Virtex®-6 FPGAs that comply with the Dune Networks and Marvell RXAUI specifications.

7 series and Virtex-6 FPGAs in combination with the RXAUI core, enable the design of RXAUI-based interconnects whether they are chip-to-chip, over backplanes, or connected to 10 Gb optical modules.

Features

- Designed to Dune Networks and Marvell RXAUI specifications
- Uses two transceivers at 6.25 Gb/s line rate to achieve 10 Gb/s data rate
- Implements DTE XGXS, PHY XGXS, and 10GBASE-X PCS in a single netlist
- *IEEE 802.3-2008* clause 45 MDIO interface (optional)
- Available under the Xilinx End User License Agreement

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Zynq™-7000, Virtex-7, Kintex-7, Artix™-7, Virtex-6
Supported User Interfaces	XGMII, MDIO
Resources	See Table 2-1 and Table 2-2.
Provided with Core	
Design Files	ISE: NGC Netlist Vivado: Encrypted RTL
Example Design	Verilog/VHDL
Test Bench	Verilog/VHDL
Constraints File	ISE: UCF Vivado: XDC
Simulation Model	VHDL/Verilog
Supported S/W Driver	N/A
Tested Design Flows(2)	
Design Entry	ISE® Design Suite v14.3 Vivado™ Design Suite v2012.3
Simulation	Mentor Graphics ModelSim Cadence Incisive Enterprise Simulator (IES) Synopsys VCS and VCS MX
Synthesis	Xilinx Synthesis Technology (XST) Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete listing of supported devices, see the release notes for this core.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The RXAUI standard was developed as a means to improve the 10-Gigabit Ethernet port density. The number of XAUI interfaces that could be implemented was limited by the number of available transceivers, with capacity and performance still to be utilized. RXAUI halves the number of transceivers required compared with a XAUI implementation.

RXAUI is a two-lane, 6.25 Gb/s-per-lane serial interface. It is intended to work with an existing XAUI implementation and multiplexes/demultiplexes the two physical RXAUI lanes into 4 logical XAUI lanes. Each RXAUI lane is a differential pair carrying current mode logic (CML) signaling, and the data on each lane is 8B/10B encoded before transmission.

In this document:

- Virtex®-7, Kintex™-7 and Virtex-6 FPGAs GTX transceivers are abbreviated to GTX transceivers.
- Virtex-7 FPGA GTH transceiver is abbreviated to GTH transceiver.
- Artix™-7 FPGA GTP transceiver is abbreviated to GTP transceiver.

Feature Summary

The RXAUI core can be configured in one of two modes.

- **Dune Networks**: This RXAUI implementation maintains 8B/10B disparity on the RXAUI physical lane. The Dune Networks RXAUI standard is fully specified in DN-DS-RXAUI-Spec v.1.0.
- **Marvell**: This RXAUI implementation maintains 8B/10B disparity on the XAUI logical lane. The Marvell RXAUI standard is fully specified in MV-S105398-00.

Management Interface:

The RXAUI Core can be customized with either a two-wire low-speed serial MDIO Interface, or a configuration and status vector interface.

Applications

The applications of RXAUI have extended beyond 10-Gigabit Ethernet to the backplane and other general high-speed interconnect applications. A typical backplane application is shown in Figure 1-1.

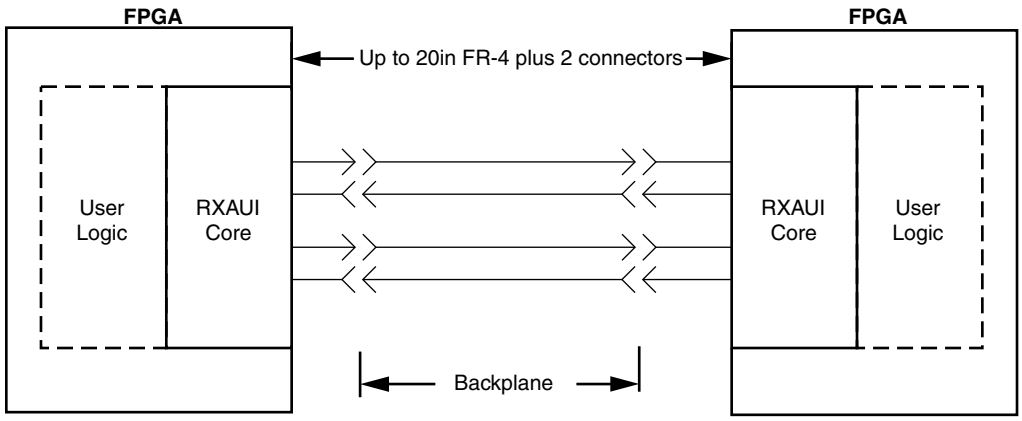

Figure 1-1: **Typical Backplane Application for RXAUI**

Functional Description

Figure 1-2 shows a block diagram of the Dune Networks RXAUI core implementation. The major functional blocks of the core include the following:

- **Transmit Idle Generation Logic:** Creates the code groups to allow synchronization and alignment at the receiver.
- **Demux Logic:** Separates the two physical RXAUI lanes into four logical XAUI lanes.
- **Synchronization State Machine (one per lane):** Identifies byte boundaries in incoming serial data.
- **Deskew State Machine:** De-skews the four received lanes into alignment.
- **Optional MDIO Interface:** A two-wire low-speed serial interface used to manage the core.
- **Embedded Transceivers:** Provide high-speed transceivers as well as 8B/10B encode and decode, and elastic buffering in the receive datapath.

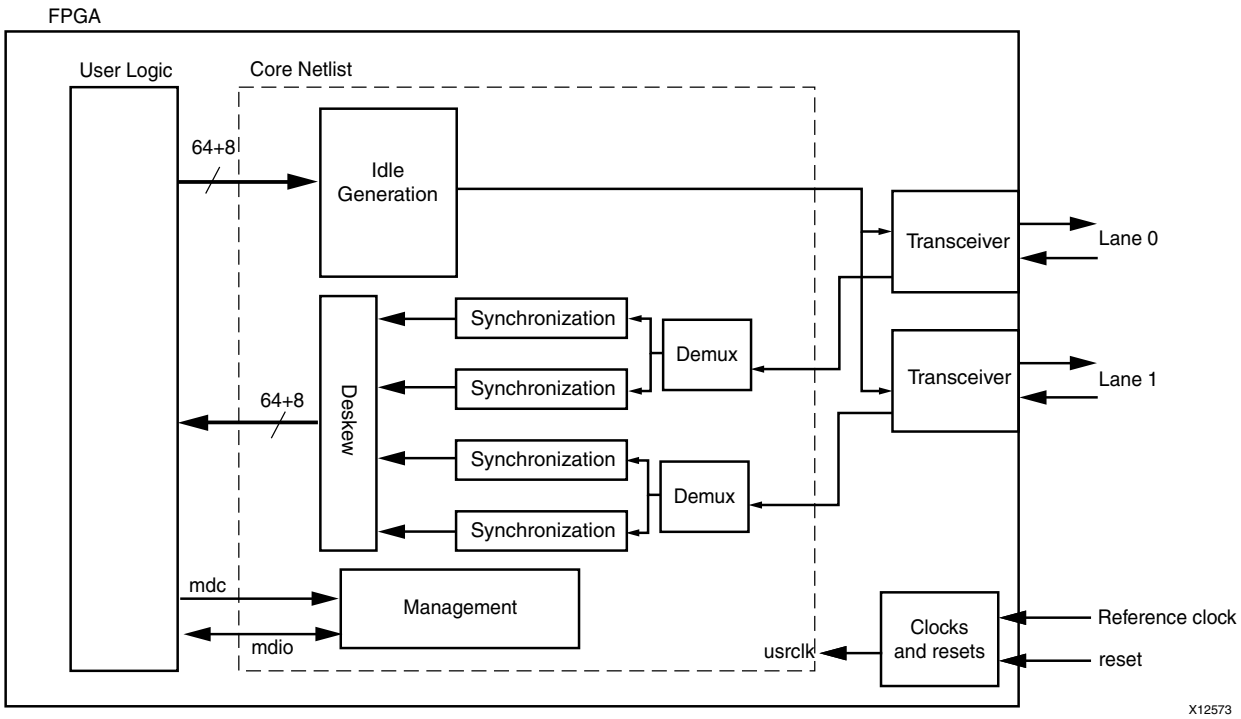

Figure 1-2: **Implementation of Dune Networks RXAUI Core**

Figure 1-3 shows a block diagram of the Marvell RXAUI core implementation. The major functional blocks of the core include the ones previously described in the Dune Networks implementation, and the following:

8B/10B encoder / decoder: Performs 8B/10B data conversion

Deskew: De-skews the four received lanes into alignment

Clock Correction: Elastic buffer and clock correction manipulation to handle difference in clock rates.

The core is implemented with the transceiver instantiations in the source code rather than in the netlist. This provides more flexibility in a particular application to use additional device-specific transceiver features and resolve placement issues.

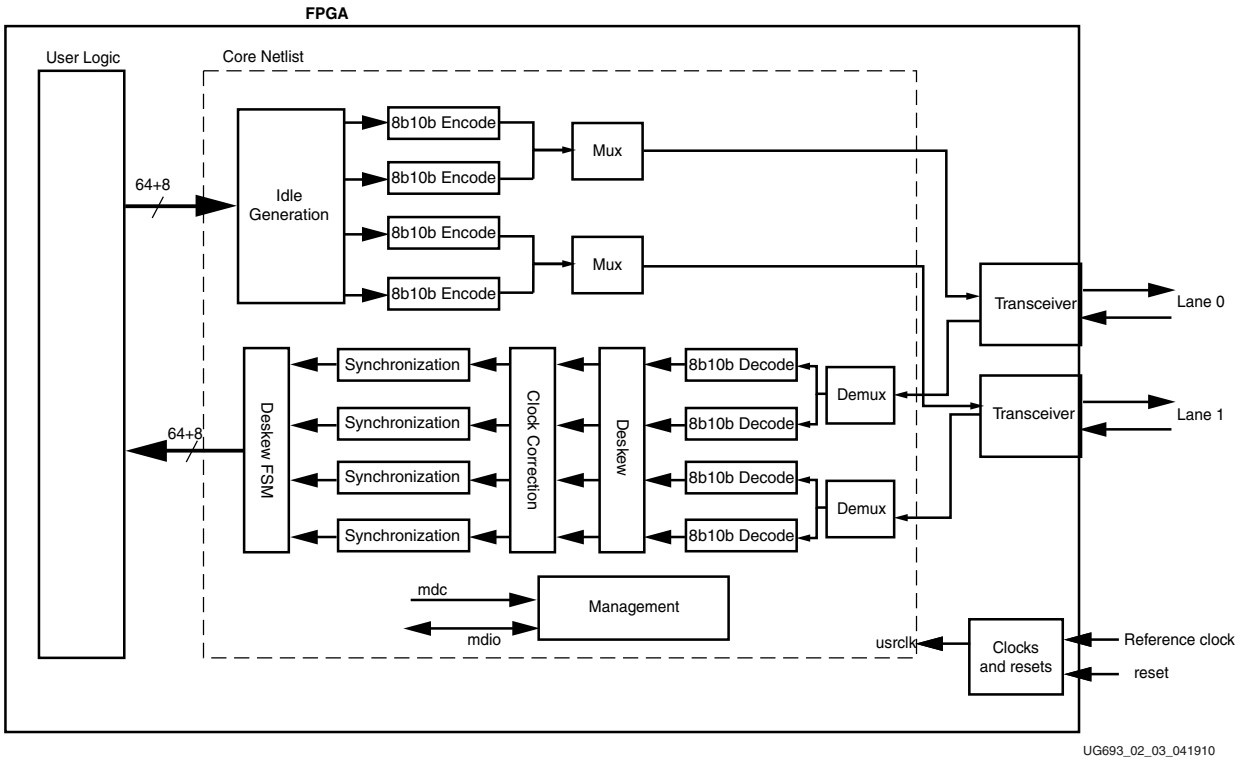

Figure 1-3: **Implementation of Marvell RXAUI Core**

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite and ISE Design Suite tools under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

The RXAUI IP core is designed to the Dune Networks [Ref 2] and Marvell RXAUI [Ref 3] specifications.

Performance

Latency

These measurements are for the core only - they do not include the latency through the transceiver. The latency through the transceiver can be obtained from the relevant user guide.

Transmit Path Latency

As measured from the input port `xgmii_txd[63:0]` of the transmitter side XGMII (until that data appears on `mgt_txdata[63:0]` on the transceiver interface), the latency through the core for the internal XGMII interface configuration in the transmit direction is 3 clock periods of the core input `usrclk` for Dune Networks mode and 4 clock periods for Marvell mode.

Receive Path Latency

Measured from the input into the core on `mgt_rxdata[63:0]` until the data appears on `xgmii_rxdata[63:0]` of the receiver side XGMII interface, the latency through the core in the receive direction for Dune Networks mode is equal to 5-7 clock cycles of `usrclk`. The latency depends on comma alignment position and data positioning within the transceiver 4-byte interface.

For Marvell Mode, the exact latency depends on comma alignment position, lane skew, and elastic buffer occupancy. Typically this is 9-14 RXCLK cycles + 13-14 `usrclk` cycles. There is an additional 1 clock cycle of `usrclk` RX pipelining in the `rxaui_block.v[hd]` file if this is being used.

Resource Utilization

Table 2-1 provides approximate utilization for the various core options on Virtex-6 LXT FPGAs using the ISE® Design Suite.

Table 2-1: **Device Utilization – Virtex-6 LXT FPGAs**

MDIO Management	RXAUI Mode	LUTs	FFs	BUFRs
FALSE	Dune	793	844	0
TRUE	Dune	913	939	0
FALSE	Marvell	1423	1429	1
TRUE	Marvell	1614	1524	1

Table 2-2 provides approximate utilization for the various core options on Virtex-7 and Kintex-7 FPGAs using the ISE® Design Suite.

Table 2-2: **Device Utilization – Zynq™-7000, Virtex-7 (GTX) and Kintex-7 FPGAs**

MDIO Management	RXAUI Mode	LUTs	FFs
FALSE	Dune	820	869
TRUE	Dune	952	969
FALSE	Marvell	1506	1466
TRUE	Marvell	1712	1561

Table 2-3 provides approximate utilization for the various core options on Virtex-7 (GTH) FPGAs using the ISE® Design Suite.

Table 2-3: **Device Utilization – Virtex-7 (GTH) FPGAs**

MDIO Management	RXAUI Mode	LUTs	FFs
FALSE	Dune	737	826
TRUE	Dune	850	922
FALSE	Marvell	1429	1423
TRUE	Marvell	1607	1518

Table 2-4 provides approximate utilization for the various core options on Artix™-7 FPGAs using the ISE® Design Suite.

Table 2-4: **Device Utilization – Artix-7 FPGAs**

MDIO Management	RXAUI Mode	LUTs	FFs
FALSE	Dune	724	826
TRUE	Dune	875	922
FALSE	Marvell	1395	1423
TRUE	Marvell	1606	1518

Port Descriptions

Client-Side Interface

The signals of the client-side interface (between the User Logic block and the Core Netlist block in Figure 1-2) are shown in Table 2-5. See Protocol Description for details on connecting to the client-side interface.

Table 2-5: **Client-Side Interface Ports**

Signal Name	Direction	Description
XGMII_TXD[63:0]	IN	Transmit data, eight bytes wide
XGMII_TXC[7:0]	IN	Transmit control bits, one bit per transmit data byte
XGMII_RXD[63:0]	OUT	Received data, eight bytes wide
XGMII_RXC[7:0]	OUT	Receive control bits, one bit per received data byte

Transceiver Interface

The interface to the device-specific transceivers is a simple pin-to-pin interface on those pins that need to be connected. Table 2-6 and Table 2-7 show the transceiver ports on the core depending on the RXAUI mode chosen when customizing the core. These are connected between the core and the transceivers in the 'block' HDL as part of the example output products (Vivado) or example design (CORE Generator). Table 2-6 shows the RXAUI Dune Mode ports and Table 2-7 shows the RXAUI Marvell Mode Ports. See Protocol Description for details on connecting the device-specific transceivers to the RXAUI core.

Table 2-6: **Dune Networks Transceiver Interface Ports**

Signal Name	Direction	Description
MGT_TXDATA[63:0]	OUT	Transceiver transmit data
MGT_TXCHARISK[7:0]	OUT	Transceiver transmit control flag
MGT_RXDATA[63:0]	IN	Transceiver receive data
MGT_RXCHARISK[7:0]	IN	Transceiver receive control signals
MGT_CODEVALID[7:0]	IN	Transceiver receive control signals
MGT_CODECOMMA[7:0]	IN	Transceiver receive control signals
MGT_ENABLE_ALIGN[1:0]	OUT	Transceiver control signals
MGT_ENCHANSYNC	OUT	Transceiver control signal
MGT_RXLOCK[1:0]	IN	Device-specific transceiver control signals
MGT_LOOPBACK	OUT	Transceiver control signal
MGT_POWERDOWN	OUT	Transceiver control signal
SIGNAL_DETECT[1:0]	IN	Status signal from attached optical module

Table 2-7: **Marvell Transceiver Interface Ports**

Signal Name	Direction	Description
MGT_TXDATA[79:0]	OUT	Transceiver transmit data
MGT_RXDATA[39:0]	IN	Transceiver receive data
MGT_ENABLE_ALIGN[1:0]	OUT	Transceiver control signals
MGT_LOOPBACK	OUT	Transceiver control signal
MGT_POWERDOWN	OUT	Transceiver control signal
SIGNAL_DETECT[1:0]	IN	Status signal from attached optical module (if present)

The `SIGNAL_DETECT` signals are intended to be driven by an attached optical module; they signify that each of the two optical receivers is receiving illumination and is therefore not just putting out noise. If an optical module is not in use, this two-wire bus should be tied to 11.

No timing diagrams are presented here for the device-specific transceiver signals. You should treat this interface as a black box. If customization of this interface is required, see the *Virtex-6 FPGA GTX Transceivers User Guide* [Ref 9], the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 10], or the *7 Series FPGAs GTP Transceivers User Guide* [Ref 11] for detailed descriptions of the transceiver ports.

This section describes the interfaces available for dynamically setting the configuration and obtaining the status of the RXAUI core. There are two interfaces for configuration; depending on the core customization, only one is available in a particular core instance.

In addition, there are output ports on the core signalling alignment and synchronization status. These ports are described in Alignment and Synchronization Status Ports.

MDIO Ports

The RXAUI core, when generated with an MDIO interface, implements an MDIO Interface Register block. The core responds to MDIO transactions as either a 10GBASE-X PCS, a DTE XS, or a PHY XS depending on the setting of the `type_sel` port (see Table 3-3). The MDIO Interface Ports are described in Table 2-8. More information on using this interface can be found in MDIO Interface.

Table 2-8: **MDIO Management Interface Ports**

Signal Name	Direction	Description
MDC	IN	Management clock
MDIO_IN	IN	MDIO input
MDIO_OUT	OUT	MDIO output
MDIO_TRI	OUT	MDIO 3-state. 1 disconnects the output driver from the MDIO bus.
TYPE_SEL[1:0]	IN	Type select
PRTAD[4:0]	IN	MDIO port address

Configuration and Status Signals

The Configuration and Status Signals are shown in Table 2-53. See Configuration and Status Vectors for details on these signals, including a breakdown of the configuration and status vectors.

Table 2-9: **Configuration and Status Ports**

Signal Name	Direction	Description
CONFIGURATION_ VECTOR[6:0]	IN	Configuration information for the core.
STATUS_VECTOR[7:0]	OUT	Status information from the core.
ALIGN_STATUS	OUT	1 when the RXAUI receiver is aligned across all four logical XAUI lanes, 0 otherwise.
SYNC_STATUS[3:0]	OUT	Each pin is 1 when the respective XAUI logical lane receiver is synchronized to byte boundaries, 0 otherwise.

Clocking and Reset Signals and Module

Included in the example design top-level sources are circuits for clock and reset management. These can include Mixed-Mode Clock Managers (MMCMs), reset synchronizers, or other useful utility circuits that can be useful in your particular application. Table 2-10 shows the ports on the netlist that are associated with system clocks and resets.

Table 2-10: **Clock and Reset Ports**

Signal Name	Direction	Description
USRCLK	IN	156.25 MHz system clock for core.
RESET	IN	Reset port synchronous to USRCLK.
SOFT_RESET	OUT	Reset signal controlled by MDIO register bit. This reset signal also resets the transceivers.
MGT_TX_RESET	IN	The reset signal used to drive the transceiver TXRESET signal.
MGT_RX_RESET	IN	The reset signal used to drive the transceiver RXRESET signal.
RXCLK	IN	Transceiver recovered clock (Marvell Mode Only).

Register Space

This section describes the interfaces available for dynamically setting the configuration and obtaining the status of the RXAUI core. There are two interfaces for configuration; depending on the core customization, only one is available in a particular core instance. The interfaces are:

- MDIO Interface Registers
- Configuration and Status Vectors

In addition, there are output ports on the core signalling alignment and synchronization status. These ports are described in Alignment and Synchronization Status Ports.

MDIO Interface Registers

For a description of the MDIO Interface, see MDIO Interface.

10GBASE-X PCS/PMA Register Map

When the core is configured as a 10GBASE-X PCS/PMA, it occupies MDIO Device Addresses 1 and 3 in the MDIO register address map, as shown in Table 2-11.

Table 2-11: **10GBASE-X PCS/PMA MDIO Registers**

Register Address	Register Name
1.0	PMA/PMD Control 1
1.1	PMA/PMD Status 1
1.2,1.3	PMA/PMD Device Identifier
1.4	PMA/PMD Speed Ability
1.5, 1.6	PMA/PMD Devices in Package
1.7	10G PMA/PMD Control 2
1.8	10G PMA/PMD Status 2
1.9	Reserved
1.10	10G PMD Receive Signal OK
1.11 TO 1.13	Reserved
1.14, 1.15	PMA/PMD Package Identifier
1.16 to 1.65 535	Reserved
3.0	PCS Control 1
3.1	PCS Status 1
3.2, 3.3	PCS Device Identifier
3.4	PCS Speed Ability
3.5, 3.6	PCS Devices in Package
3.7	10G PCS Control 2
3.8	10G PCS Status 2
3.9 to 3.13	Reserved
3.14, 3.15	Package Identifier
3.16 to 3.23	Reserved
3.24	10GBASE-X PCS Status
3.25	10GBASE-X Test Control
3.26 to 3.65 535	Reserved

MDIO Register 1.0: PMA/PMD Control 1

Figure 2-1 shows the MDIO Register 1.0: PMA/PMD Control 1.

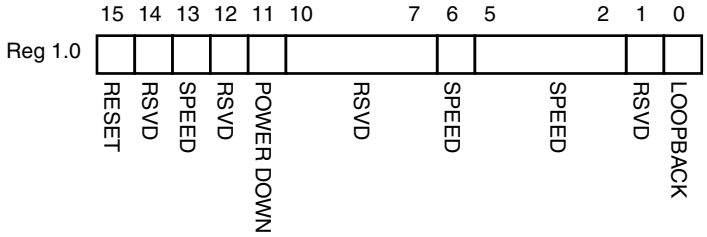

Figure 2-1: **PMA/PMD Control 1 Register**

Table 2-12 shows the PMA Control 1 register bit definitions.

Table 2-12: **PMA/PMD Control 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.0.15	Reset	1 = Block reset 0 = Normal operation The RXAUI block is reset when this bit is set to 1. It returns to 0 when the reset is complete. The 'soft_reset' pin is connected to this bit. This can be connected to the reset of any other MMDs.	R/W Self-clearing	0
1.0.14	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0
1.0.13	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
1.0.12	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0
1.0.11	Power down	1 = Power down mode 0 = Normal operation When set to 1, the serial transceivers are placed in a low power state. Set to 0 to return to normal operation	R/W	0
1.0.10:7	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
1.0.6	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
1.0.5:2	Speed Selection	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
1.0.1	Reserved	The block always returns 0 for this bit and ignores writes	R/O	All 0s
1.0.0	Loopback	1 = Enable loopback mode 0 = Disable loopback mode The RXAUI block loops the signal in the serial transceivers back into the receiver.	R/W	0

MDIO Register 1.1: PMA/PMD Status 1

Figure 2-2 shows the MDIO Register 1.1: PMA/PMD Status 1.

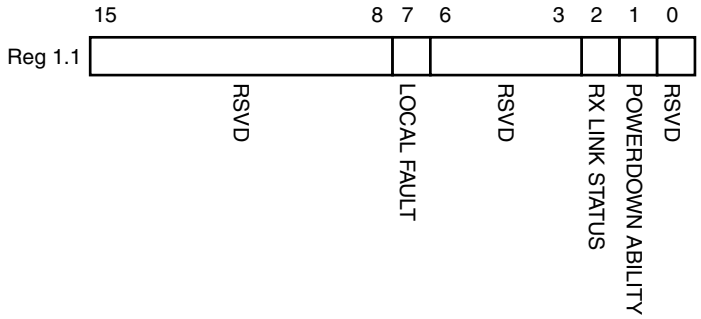

Figure 2-2: **PMA/PMD Status 1 Register**

Table 2-13 shows the PMA/PMD Status 1 register bit definitions.

Table 2-13: **PMA/PMD Status 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.1.15:8	Reserved	The block always returns 0 for this bit.	R/O	0
1.1.7	Local Fault	The block always returns 0 for this bit.	R/O	0
1.1.6:3	Reserved	The block always returns 0 for this bit.	R/O	0
1.1.2	Receive Link Status	The block always returns 1 for this bit.	R/O	1
1.1.1	Power Down Ability	The block always returns 1 for this bit.	R/O	1
1.1.0	Reserved	The block always returns 0 for this bit.	R/O	0

MDIO Registers 1.2 and 1.3: PMA/PMD Device Identifier

Figure 2-3 shows the MDIO Registers 1.2 and 1.3: PMA/PMD Device Identifier.

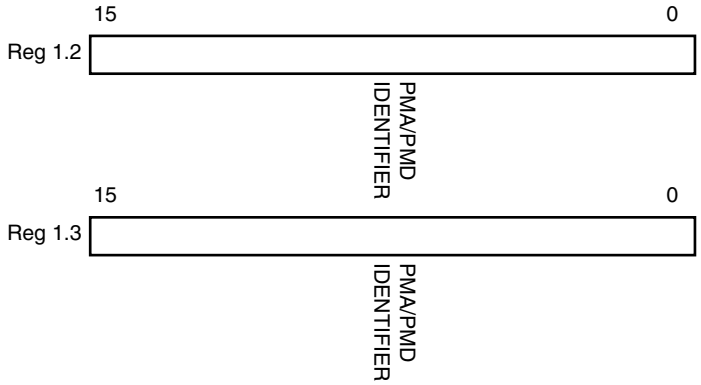

Figure 2-3: **PMA/PMD Device Identifier Registers**

Table 2-14 shows the PMA/PMD Device Identifier registers bit definitions.

Table 2-14: **PMA/PMD Device Identifier Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.2.15:0	PMA/PMD Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
1.3.15:0	PMA/PMD Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s

MDIO Register 1.4: PMA/PMD Speed Ability

Figure 2-4 shows the MDIO Register 1.4: PMA/PMD Speed Ability.

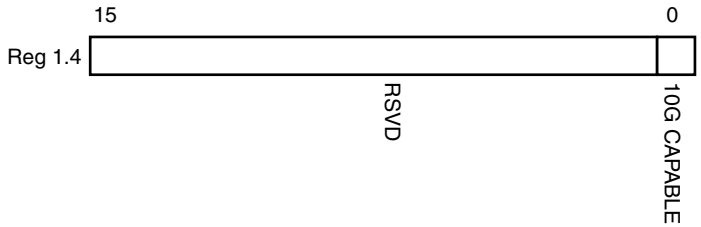

Figure 2-4: **PMA/PMD Speed Ability Register**

Table 2-15 shows the PMA/PMD Speed Ability register bit definitions.

Table 2-15: **PMA/PMD Speed Ability Register Bit Definitions**

Bit(s)	Name	Description	Attribute	Default Value
1.4.15:1	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
1.4.0	10G Capable	The block always returns 1 for this bit and ignores writes.	R/O	1

MDIO Registers 1.5 and 1.6: PMA/PMD Devices in Package

Figure 2-5 shows the MDIO Registers 1.5 and 1.6: PMA/PMD Devices in Package.

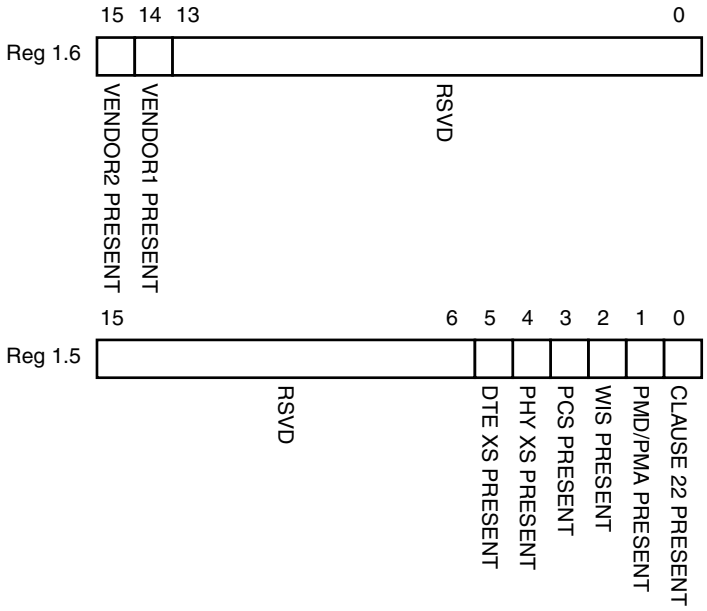

Figure 2-5: **PMA/PMD Devices in Package Registers**

Table 2-16 shows the PMA/PMD Device in Package registers bit definitions.

Table 2-16: **PMA/PMD Devices in Package Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.6.15	Vendor- specific Device 2 present	The block always returns 0 for this bit.	R/O	0
1.6.14	Vendor-specific Device 1 present	The block always returns 0 for this bit.	R/O	0
1.6.13:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s
1.5.15:6	Reserved	The block always returns 0 for these bits.	R/O	All 0s
1.5.5	DTE XS present	The block always returns 0 for this bit.	R/O	0
1.5.4	PHY XS present	The block always returns 0 for this bit.	R/O	0
1.5.3	PCS present	The block always returns 1 for this bit.	R/O	1
1.5.2	WIS present	The block always returns 0 for this bit.	R/O	0
1.5.1	PMA/PMD present	The block always returns 1 for this bit.	R/O	1
1.5.0	Clause 22 Device present	The block always returns 0 for this bit.	R/O	0

MDIO Register 1.7: 10G PMA/PMD Control 2

Figure 2-6 shows the MDIO Register 1.7: 10G PMA/PMD Control 2.

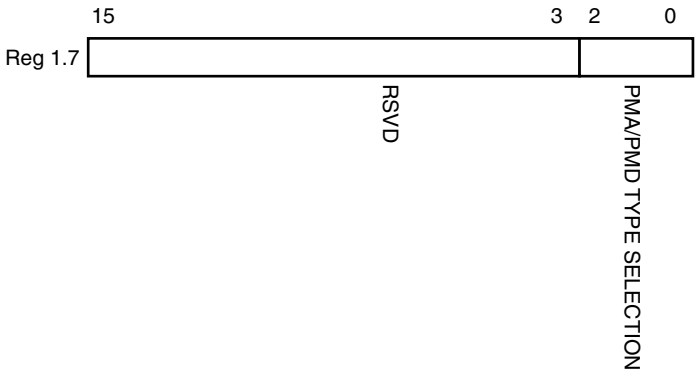

Figure 2-6: **10G PMA/PMD Control 2 Register**

Table 2-17 shows the PMA/PMD Control 2 register bit definitions.

Table 2-17: **10G PMA/PMD Control 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.7.15:3	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
1.7.2:0	PMA/PMD Type Selection	The block always returns 100 for these bits and ignores writes. This corresponds to the 10GBASE-X PMA/PMD.	R/O	100

MDIO Register 1.8: 10G PMA/PMD Status 2

Figure 2-7 shows the MDIO Register 1.8: 10G PMA/PMD Status 2.

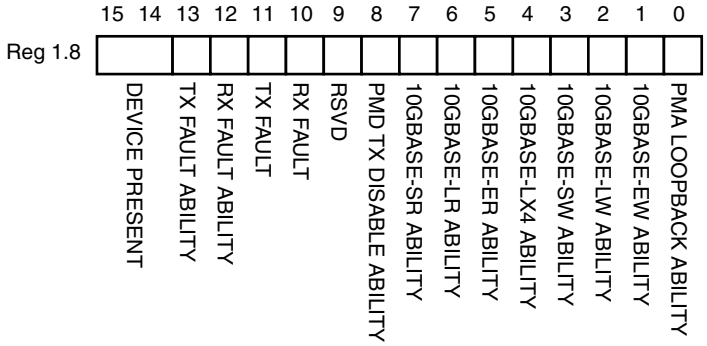

Figure 2-7: **10G PMA/PMD Status 2 Register**

Table 2-18 shows the PMA/PMD Status 2 register bit definitions.

Table 2-18: **10G PMA/PMD Status 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.8.15:14	Device present	The block always returns 10 for these bits.	R/O	10
1.8.13	Transmit Local Fault Ability	The block always returns 0 for this bit.	R/O	0
1.8.12	Receive Local Fault Ability	The block always returns 0 for this bit.	R/O	0
1.8.11	Transmit Fault	The block always returns 0 for this bit.	R/O	0
1.8.10	Receive Fault	The block always returns 0 for this bit.	R/O	0
1.8.9	Reserved	The block always returns 0 for this bit.	R/O	0
1.8.8	PMD Transmit Disable Ability	The block always returns 0 for this bit.	R/O	0
1.8.7	10GBASE-SR Ability	The block always returns 0 for this bit.	R/O	0
1.8.6	10GBASE-LR Ability	The block always returns 0 for this bit.	R/O	0
1.8.5	10GBASE-ER Ability	The block always returns 0 for this bit.	R/O	0
1.8.4	10GBASE-LX4 Ability	The block always returns 1 for this bit.	R/O	1
1.8.3	10GBASE-SW Ability	The block always returns 0 for this bit.	R/O	0
1.8.2	10GBASE-LW Ability	The block always returns 0 for this bit.	R/O	0
1.8.1	10GBASE-EW Ability	The block always returns 0 for this bit.	R/O	0
1.8.0	PMA Loopback Ability	The block always returns 1 for this bit.	R/O	1

MDIO Register 1.10: 10G PMD Signal Receive OK

Figure 2-8 shows the MDIO 1.10 Register: 10G PMD Signal Receive OK.

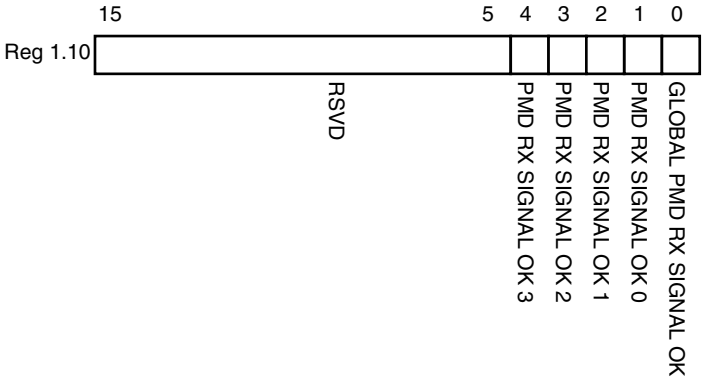

Figure 2-8: **10G PMD Signal Receive OK Register**

Table 2-19 shows the 10G PMD Signal Receive OK register bit definitions.

Table 2-19: **10G PMD Signal Receive OK Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.10.15:5	Reserved	The block always returns 0s for these bits.	R/O	All 0s
1.10.4	PMD Receive Signal OK 3	1 = Signal OK on receive Lane 3 0 = Signal not OK on receive Lane 3 This is the value of the SIGNAL_DETECT[3] port.	R/O	-
1.10.3	PMD Receive Signal OK 2	1 = Signal OK on receive Lane 2 0 = Signal not OK on receive Lane 2 This is the value of the SIGNAL_DETECT[2] port.	R/O	-
1.10.2	PMD Receive Signal OK 1	1 = Signal OK on receive Lane 1 0 = Signal not OK on receive Lane 1 This is the value of the SIGNAL_DETECT[1] port.	R/O	-
1.10.1	PMD Receive Signal OK 0	1 = Signal OK on receive Lane 0 0 = Signal not OK on receive Lane 0 This is the value of the SIGNAL_DETECT[0] port.	R/O	-
1.10.0	Global PMD Receive Signal OK	1 = Signal OK on all receive lanes 0 = Signal not OK on all receive lanes	R/O	-

MDIO Registers 1.14 and 1.15: PMA/PMD Package Identifier

Figure 2-9 shows the MDIO Registers 1.14 and 1.15: PMA/PMD Package Identifier register.

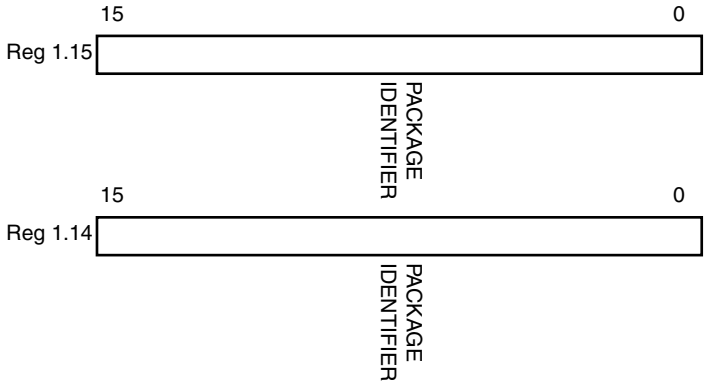

Figure 2-9: **PMA/PMD Package Identifier Registers**

Table 2-20 shows the PMA/PMD Package Identifier registers bit definitions.

Table 2-20: **PMA/PMD Package Identifier Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
1.15.15:0	PMA/PMD Package Identifier	The block always returns 0 for these bits.	R/O	All 0s
1.14.15:0	PMA/PMD Package Identifier	The block always returns 0 for these bits.	R/O	All 0s

MDIO Register 3.0: PCS Control 1

Figure 2-10 shows the MDIO Register 3.0: PCS Control 1.

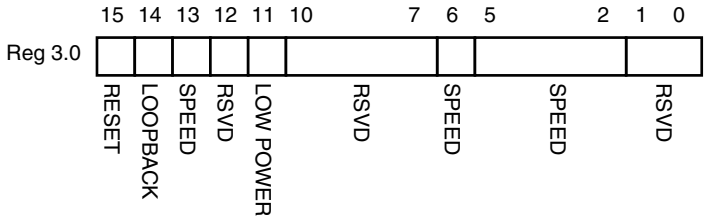

Figure 2-10: **PCS Control 1 Register**

Table 2-21 shows the PCS Control 1 register bit definitions.

Table 2-21: **PCS Control 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.0.15	Reset	1 = Block reset 0 = Normal operation The RXAUI block is reset when this bit is set to 1. It returns to 0 when the reset is complete.	R/W Self-clearing	0
3.0.14	10GBASE-R Loopback	The block always returns 0 for this bit and ignores writes.	R/O	0
3.0.13	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
3.0.12	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0
3.0.11	Power down	1 = Power down mode 0 = Normal operation When set to 1, the serial transceivers are placed in a low power state. Set to 0 to return to normal operation.	R/W	0
3.0.10:7	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
3.0.6	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
3.0.5:2	Speed Selection	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
3.0.1:0	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	All 0s

MDIO Register 3.1: PCS Status 1

Figure 2-11 shows the MDIO Register 3.1: PCS Status 1.

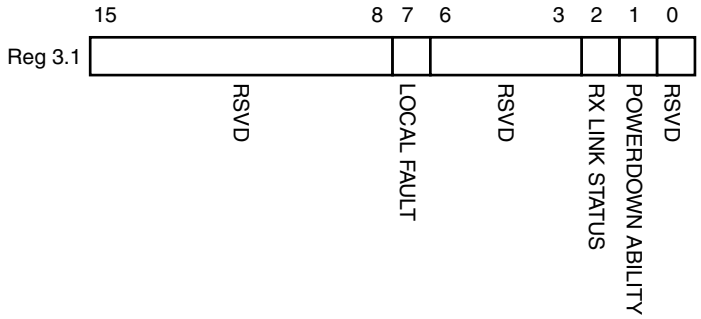

Figure 2-11: **PCS Status 1 Register**

Table 2-22 show the PCS 1 register bit definitions.

Table 2-22: **PCS Status 1 Register Bit Definition**

Bit(s)	Name	Description	Attributes	Default Value
3.1.15:8	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
3.1.7	Local Fault	1 = Local fault detected 0 = No local fault detected This bit is set to 1 whenever either of the bits 3.8.11, 3.8.10 are set to 1.	R/O	-
3.1.6:3	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
3.1.2	PCS Receive Link Status	1 = The PCS receive link is up 0 = The PCS receive link is down This is a latching Low version of bit 3.24.12.	R/O Self-setting	-
3.1.1	Power Down Ability	The block always returns 1 for this bit.	R/O	1
3.1.0	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0

MDIO Registers 3.2 and 3.3: PCS Device Identifier

Figure 2-12 shows the MDIO Registers 3.2 and 3.3: PCS Device Identifier.

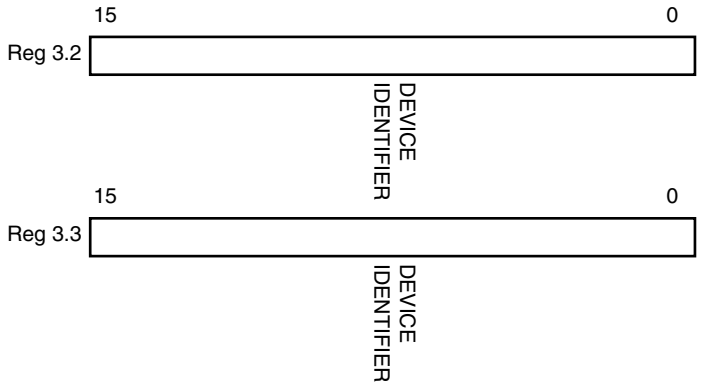

Figure 2-12: **PCS Device Identifier Registers**

Table 2-23 shows the PCS Device Identifier registers bit definitions.

Table 2-23: **PCS Device Identifier Registers Bit Definition**

Bit(s)	Name	Description	Attributes	Default Value
3.2.15:0	PCS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
3.3.15:0	PCS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s

MDIO Register 3.4: PCS Speed Ability

Figure 2-13 shows the MDIO Register 3.4: PCS Speed Ability.

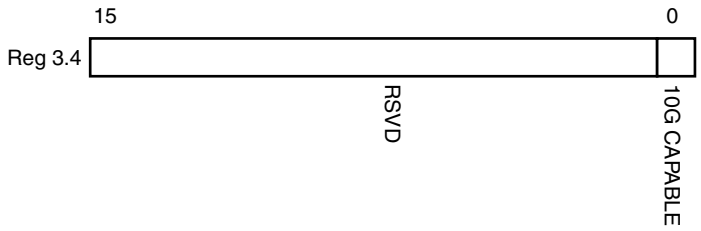

Figure 2-13: **PCS Speed Ability Register**

Table 2-24 shows the PCS Speed Ability register bit definitions.

Table 2-24: **PCS Speed Ability Register Bit Definition**

Bit(s)	Name	Description	Attribute	Default Value
3.4.15:1	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
3.4.0	10G Capable	The block always returns 1 for this bit and ignores writes.	R/O	1

MDIO Registers 3.5 and 3.6: PCS Devices in Package

Figure 2-14 shows the MDIO Registers 3.5 and 3.6: PCS Devices in Package.

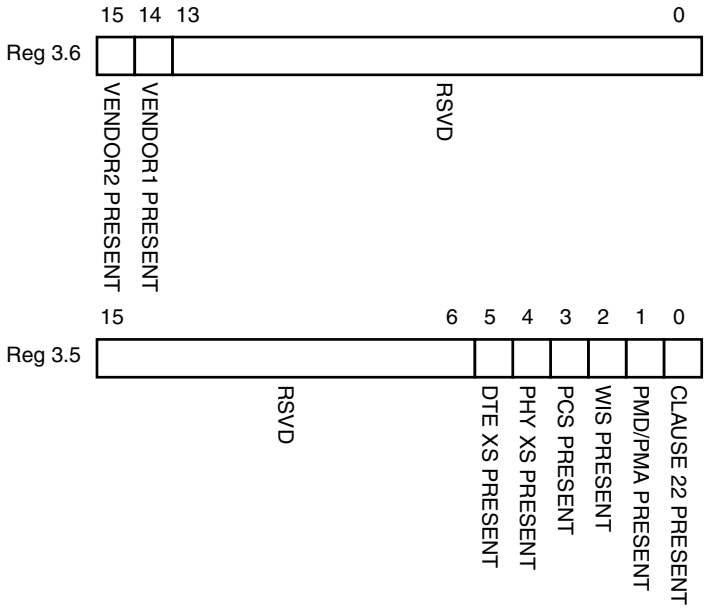

Figure 2-14: **PCS Devices in Package Registers**

Table 2-25 shows the PCS Devices in Package registers bit definitions.

Table 2-25: **PCS Devices in Package Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.6.15	Vendor-specific Device 2 present	The block always returns 0 for this bit.	R/O	0
3.6.14	Vendor- specific Device 1 present	The block always returns 0 for this bit.	R/O	0
3.6.13:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.5.15:6	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.5.5	PHY XS present	The block always returns 0 for this bit.	R/O	0
3.5.4	PHY XS present	The block always returns 0 for this bit.	R/O	0
3.5.3	PCS present	The block always returns 1 for this bit.	R/O	1
3.5.2	WIS present	The block always returns 0 for this bit.	R/O	0
3.5.1	PMA/PMD present	The block always returns 1 for this bit.	R/O	1
3.5.0	Clause 22 device present	The block always returns 0 for this bit.	R/O	0

MDIO Register 3.7: 10G PCS Control 2

Figure 2-15 shows the MDIO Register 3.7: 10G PCS Control 2.

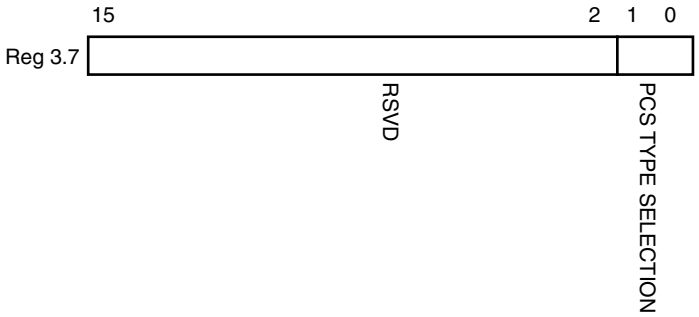

Figure 2-15: **10G PCS Control 2 Register**

Table 2-26 shows the 10 G PCS Control 2 register bit definitions.

Table 2-26: **10G PCS Control 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.7.15:2	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
3.7.1:0	PCS Type Selection	The block always returns 01 for these bits and ignores writes.	R/O	01

MDIO Register 3.8: 10G PCS Status 2

Figure 2-16 shows the MDIO Register 3.8: 10G PCS Status 2.

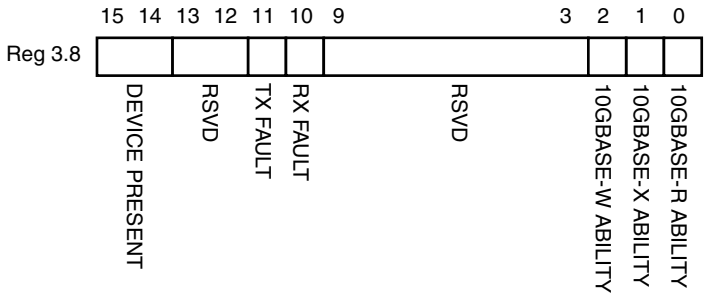

Figure 2-16: **10G PCS Status 2 Register**

Table 2-27 shows the 10G PCS Status 2 register bit definitions.

Table 2-27: **10G PCS Status 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.8.15:14	Device present	The block always returns 10.	R/O	10
3.8.13:12	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.8.11	Transmit local fault	1 = Fault condition on transmit path 0 = No fault condition on transmit path	R/O Latching High	-
3.8.10	Receive local fault	1 = Fault condition on receive path 0 = No fault condition on receive path	R/O Latching High	-
3.8.9:3	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.8.2	10GBASE-W Capable	The block always returns 0 for this bit.	R/O	0
3.8.1	10GBASE-X Capable	The block always returns 1 for this bit.	R/O	1
3.8.0	10GBASE-R Capable	The block always returns 0 for this bit.	R/O	0

MDIO Registers 3.14 and 3.15: PCS Package Identifier

Figure 2-17 shows the MDIO Registers 3.14 and 3.15: PCS Package Identifier.

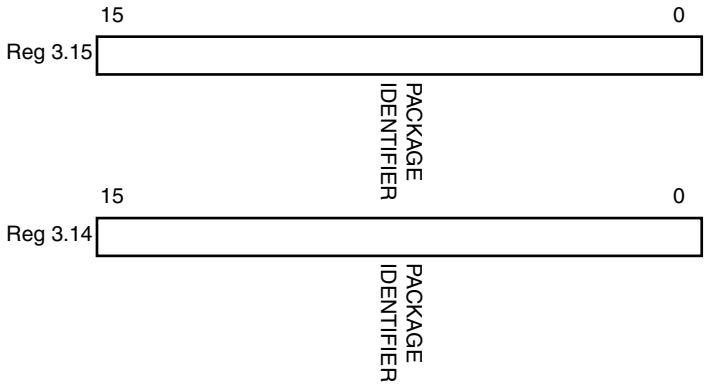

Figure 2-17: **Package Identifier Registers**

Table 2-28 shows the PCS Package Identifier registers bit definitions.

Table 2-28: **PCS Package Identifier Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.14.15:0	Package Identifier	The block always returns 0 for these bits.	R/O	All 0s
3.15.15:0	Package Identifier	The block always returns 0 for these bits.	R/O	All 0s

MDIO Register 3.24: 10GBASE-X Status

Figure 2-18 shows the MDIO Register 3.24: 10GBase-X Status.

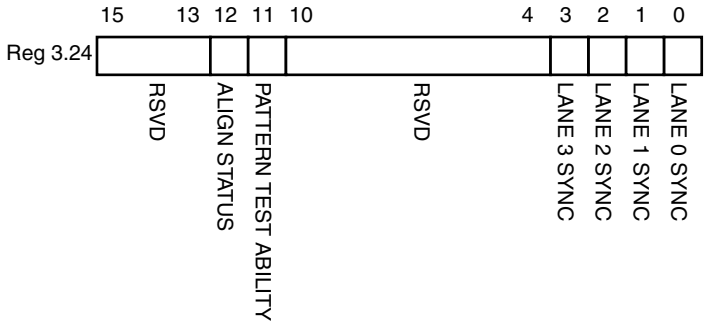

Figure 2-18: **10GBASE-X Status Register**

Table 2-29 shows the 10GBase-X Status register bit definitions.

Table 2-29: **10GBASE-X Status Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.24.15:13	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.24.12	10GBASE-X Lane Alignment Status	1 = 10GBASE-X receive lanes aligned 0 = 10GBASE-X receive lanes not aligned.	RO	-
3.24.11	Pattern Testing Ability	The block always returns 1 for this bit.	R/O	1
3.24.10:4	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.24.3	Lane 3 Sync	1 = Lane 3 is synchronized 0 = Lane 3 is not synchronized.	R/O	-
3.24.2	Lane 2 Sync	1 =Lane 2 is synchronized 0 =Lane 2 is not synchronized.	R/O	-
3.24.1	Lane 1 Sync	1 = Lane 1 is synchronized 0 = Lane 1 is not synchronized.	R/O	-
3.24.0	Lane 0 Sync	1 = Lane 0 is synchronized 0 = Lane 0 is not synchronized.	R/O	-

MDIO Register 3.25: 10GBASE-X Test Control

Figure 2-19 shows the MDIO Register 3.25: 10GBase-X Test Control.

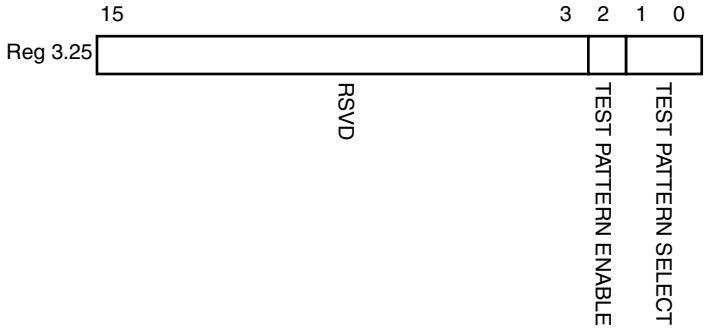

Figure 2-19: **Test Control Register**

Table 2-30 shows the 10GBase-X Test Control register bit definitions.

Table 2-30: **10GBASE-X Test Control Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
3.25.15:3	Reserved	The block always returns 0 for these bits.	R/O	All 0s
3.25.2	Transmit Test Pattern Enable	1 = Transmit test pattern enable 0 = Transmit test pattern disabled	R/W	0
3.25.1:0	Test Pattern Select	11 = Reserved 10 = Mixed frequency test pattern 01 = Low frequency test pattern 00 = High frequency test pattern	R/W	00

DTE XS MDIO Register Map

When the core is configured as a DTE XGXS, it occupies MDIO Device Address 5 in the MDIO register address map (Table 2-31).

Table 2-31: **DTE XS MDIO Registers**

Register Address	Register Name
5.0	DTE XS Control 1
5.1	DTE XS Status 1
5.2, 5.3	DTE XS Device Identifier
5.4	DTE XS Speed Ability
5.5, 5.6	DTE XS Devices in Package
5.7	Reserved
5.8	DTE XS Status 2
5.9 to 5.13	Reserved
5.14, 5.15	DTE XS Package Identifier
5.16 to 5.23	Reserved
5.24	10G DTE XGXS Lane Status
5.25	10G DTE XGXS Test Control

MDIO Register 5.0:DTE XS Control 1

Figure 2-20 shows the MDIO Register 5.0: DTE XS Control 1.

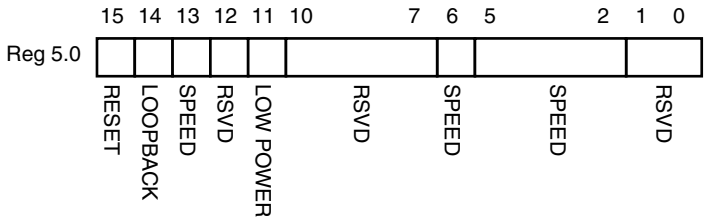

Figure 2-20: **DTE XS Control 1 Register**

Table 2-32 shows the DTE XS Control 1 register bit definitions.

Table 2-32: **DTE XS Control 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.0.15	Reset	1 = Block reset 0 = Normal operation The RXAUI block is reset when this bit is set to 1. It returns to 0 when the reset is complete.	R/W Self-clearing	0
5.0.14	Loopback	1 = Enable loopback mode 0 = Disable loopback mode The RXAUI block loops the signal in the serial transceivers back into the receiver.	R/W	0
5.0.13	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
5.0.12	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0
5.0.11	Power down	1 = Power down mode 0 = Normal operation When set to 1, the serial transceivers are placed in a low power state. Set to 0 to return to normal operation	R/W	0
5.0.10:7	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
5.0.6	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
5.0.5:2	Speed Selection	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
5.0.1:0	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s

MDIO Register 5.1: DTE XS Status 1

Figure 2-21 shows the MDIO Register 5.1: DTE XS Status 1.

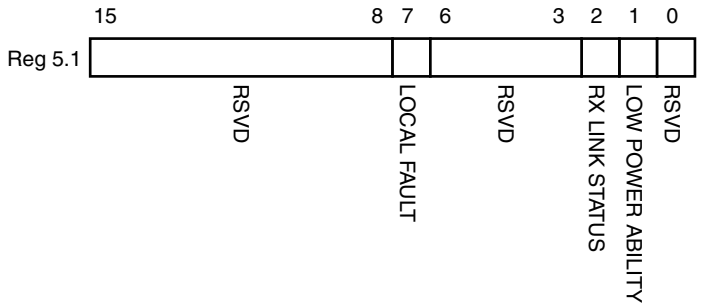

Figure 2-21: **DTE XS Status 1 Register**

Table 2-33 shows the DET XS Status 1 register bit definitions.

Table 2-33: **DTE XS Status 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.1.15:8	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
5.1.7	Local Fault	1 = Local fault detected 0 = No Local Fault detected This bit is set to 1 whenever either of the bits 5.8.11, 5.8.10 are set to 1.	R/O	-
5.1.6:3	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
5.1.2	DTE XS Receive Link Status	1 = The DTE XS receive link is up. 0 = The DTE XS receive link is down. This is a latching Low version of bit 5.24.12.	R/O Self-setting	-
5.1.1	Power Down Ability	The block always returns 1 for this bit.	R/O	1
5.1.0	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0

MDIO Registers 5.2 and 5.3: DTE XS Device Identifier

Figure 2-22 shows the MDIO Registers 5.2 and 5.3: DTE XS Device Identifier.

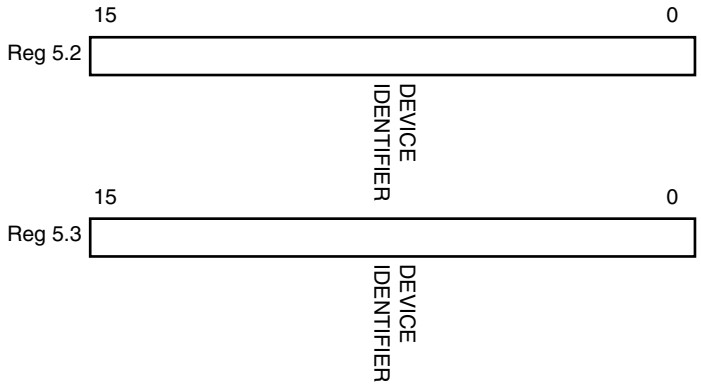

Figure 2-22: **DTE XS Device Identifier Registers**

Table 2-34 shows the DTE XS Device Identifier registers bit definitions.

Table 2-34: **DTE XS Device Identifier Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.2.15:0	DTE XS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
5.3.15:0	DTE XS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s

MDIO Register 5.4: DTE XS Speed Ability

Figure 2-23 shows the MDIO Register 5.4: DTE Speed Ability.

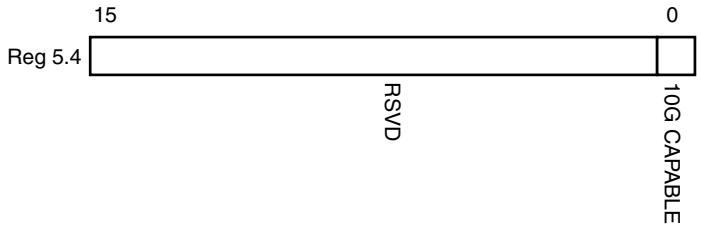

Figure 2-23: **DTE XS Speed Ability Register**

Table 2-35 shows the DTE XS Speed Ability register bit definitions.

Table 2-35: **DTE XS Speed Ability Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.4.15:1	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
5.4.0	10G Capable	The block always returns 1 for this bit and ignores writes.	R/O	1

MDIO Registers 5.5 and 5.6: DTE XS Devices in Package

Figure 2-24 shows the MDIO Registers 5.5 and 5.6: DTE XS Devices in Package.

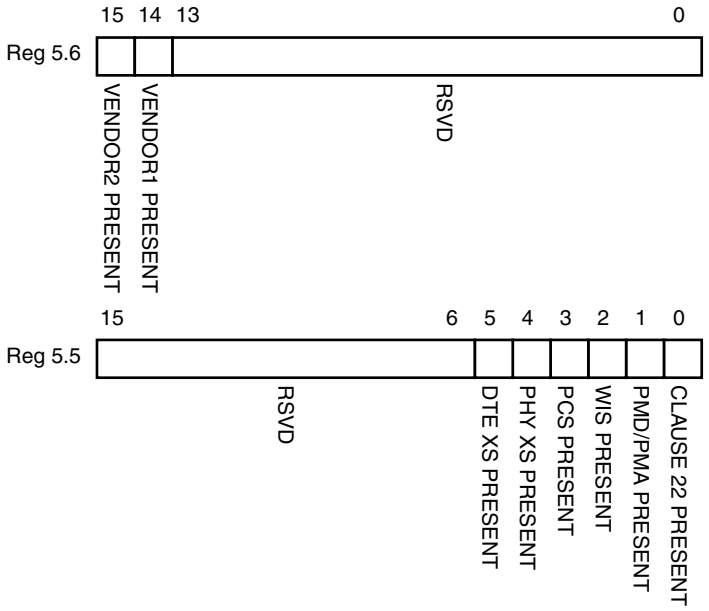

Figure 2-24: **DTE XS Devices in Package Register**

Table 2-36 shows the DTE XS Devices in Package registers bit definitions.

Table 2-36: **DTE XS Devices in Package Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.6.15	Vendor-specific Device 2 present	The block always returns 0 for this bit.	R/O	0
5.6.14	Vendor-specific Device 1 present	The block always returns 0 for this bit.	R/O	0
5.6.13:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.6.15:6	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.5.5	DTE XS present	The block always returns 1 for this bit.	R/O	1
5.5.4	PHY XS present	The block always returns 0 for this bit.	R/O	0
5.5.3	PCS present	The block always returns 0 for this bit.	R/O	0
5.5.2	WIS present	The block always returns 0 for this bit.	R/O	0
5.5.1	PMA/PMD present	The block always returns 0 for this bit.	R/O	0
5.5.0	Clause 22 Device present	The block always returns 0 for this bit.	R/O	0

MDIO Register 5.8: DTE XS Status 2

Figure 2-25 shows the MDIO Register 5.8: DTE XS Status 2.

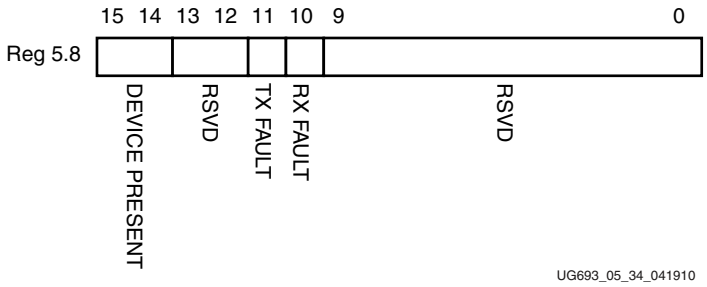

Figure 2-25: **DTE XS Status 2 Register**

Table 2-37 show the DTE XS Status 2 register bits definitions.

Table 2-37: **DTE XS Status 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.8.15:14	Device present	The block always returns 10.	R/O	10
5.8.13:12	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.8.11	Transmit Local Fault	1 = Fault condition on transmit path 0 = No fault condition on transmit path	R/O Latching High	-
5.8.10	Receive Local Fault	1 = Fault condition on receive path 0 = No fault condition on receive path	R/O Latching High	-
5.8.9:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s

MDIO Registers 5.14 and 5.15: DTE XS Package Identifier

Figure 2-26 shows the MDIO Registers 5.14 and 5.15: DTE XS Package Identifier.

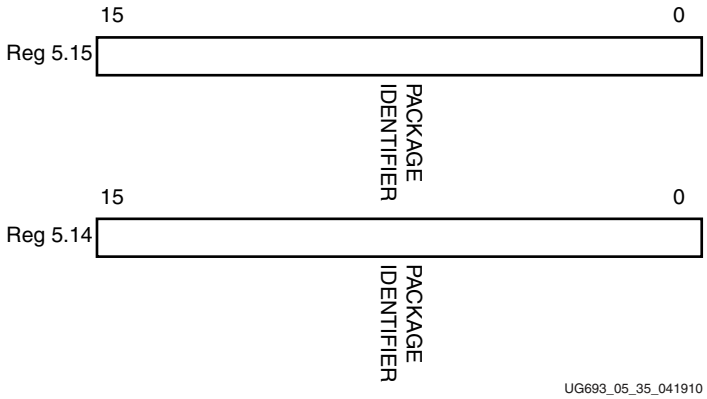

Figure 2-26: **DTE XS Package Identifier Registers**

Table 2-38 shows the DTE XS Package Identifier registers bit definitions.

Table 2-38: **DTE XS Package Identifier Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.14.15:0	DTE XS Package Identifier	The block always returns 0 for these bits.	R/O	All 0s
5.15.15:0	DTE XS Package Identifier	The block always returns 0 for these bits.	R/O	All 0s

Test Patterns

The RXAUI core is capable of sending test patterns for system debug. These patterns are defined in Annex 48A of *IEEE Std. 802.3-2008* and transmission of these patterns is controlled by the MDIO Test Control Registers.

There are three types of pattern available:

- High frequency test pattern of 1010101010.... at each device-specific transceiver output
- Low frequency test pattern of 11111000001111100000111110000.... at each device-specific transceiver output
- mixed frequency test pattern of 11111010110000010100111110101100000101 0... at each device-specific transceiver output.

MDIO Register 5.24: DTE XS Lane Status

Figure 2-27 shows the MDIO Register 5.24: DTE XS Lane Status.

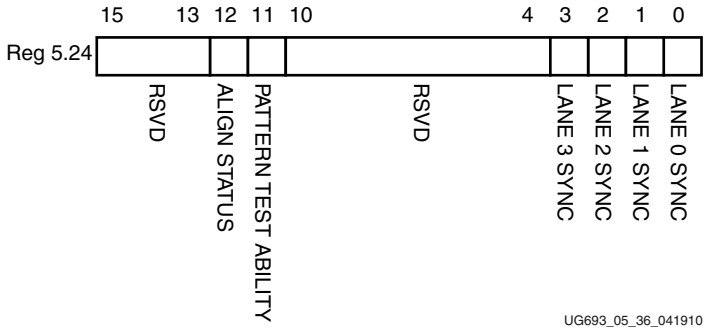

Figure 2-27: **DTE XS Lane Status Register**

Table 2-39 shows the DTE XS Lane Status register bit definitions.

Table 2-39: **DTE XS Lane Status Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.24.15:13	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.24.12	DTE XGXS Lane Alignment Status	1 = DTE XGXS receive lanes aligned 0 = DTE XGXS receive lanes not aligned	R/O	-
5.24.11	Pattern testing ability	The block always returns 1 for this bit.	R/O	1
5.24.10:4	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.24.3	Lane 3 Sync	1 = Lane 3 is synchronized 0 = Lane 3 is not synchronized.	R/O	-
5.24.2	Lane 2 Sync	1 = Lane 2 is synchronized 0 = Lane 2 is not synchronized.	R/O	-
5.24.1	Lane 1 Sync	1 = Lane 1 is synchronized 0 = Lane 1 is not synchronized.	R/O	-
5.24.0	Lane 0 Sync	1 = Lane 0 is synchronized 0 = Lane 0 is not synchronized.	R/O	-

MDIO Register 5.25: 10G DTE XGXS Test Control

Figure 2-28 shows the MDIO Register 5.25: 10G DTE XGXS Test Control.

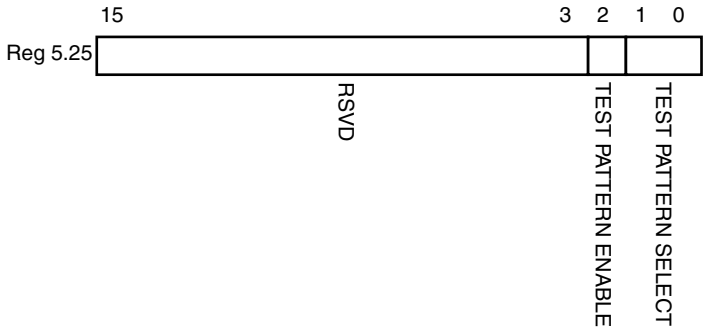

Figure 2-28: **10G DTE XGXS Test Control Register**

Table 2-40 shows the 10G DTE XGXS Test Control register bit definitions.

Table 2-40: **10G DTE XGXS Test Control Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
5.25.15:3	Reserved	The block always returns 0 for these bits.	R/O	All 0s
5.25.2	Transmit Test Pattern Enable	1 = Transmit test pattern enable 0 = Transmit test pattern disabled	R/W	0
5.25.1:0	Test Pattern Select	11 = Reserved 10 = Mixed frequency test pattern 01 = Low frequency test pattern 00 = High frequency test pattern	R/W	00

PHY XS MDIO Register Map

When the core is configured as a PHY XGXS, it occupies MDIO Device Address 4 in the MDIO register address map (Table 2-41).

Table 2-41: **PHY XS MDIO Registers**

Register Address	Register Name
4.0	PHY XS Control 1
4.1	PHY XS Status 1
4.2, 4.3	Device Identifier
4.4	PHY XS Speed Ability
4.5, 4.6	Devices in Package
4.7	Reserved
4.8	PHY XS Status 2
4.9 to 4.13	Reserved
4.14, 4.15	Package Identifier
4.16 to 4.23	Reserved
4.24	10G PHY XGXS Lane Status
4.25	10G PHY XGXS Test Control

MDIO Register 4.0: PHY XS Control 1

Figure 2-29 shows the MDIO Register 4.0: PHY XS Control 1.

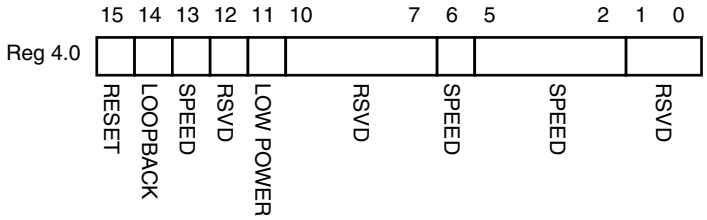

Figure 2-29: **PHY XS Control 1 Register**

Table 2-42 shows the PHY XS Control 1 register bit definitions.

Table 2-42: **PHY XS Control 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.0.15	Reset	1 = Block reset 0 = Normal operation The RXAUI block is reset when this bit is set to 1. It returns to 0 when the reset is complete.	R/W Self-clearing	0
4.0.14	Loopback	1 = Enable loopback mode 0 = Disable loopback mode The RXAUI block loops the signal in the serial transceivers back into the receiver.	R/W	0
4.0.13	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
4.0.12	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0
4.0.11	Power down	1 = Power down mode 0 = Normal operation When set to 1, the serial transceivers are placed in a low power state. Set to 0 to return to normal operation	R/W	0
4.0.10:7	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
4.0.6	Speed Selection	The block always returns 1 for this bit and ignores writes.	R/O	1
4.0.5:2	Speed Selection	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
4.0.1:0	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s

MDIO Register 4.1: PHY XS Status 1

Figure 2-30 shows the MDIO Register 4.1: PHY XS Status 1.

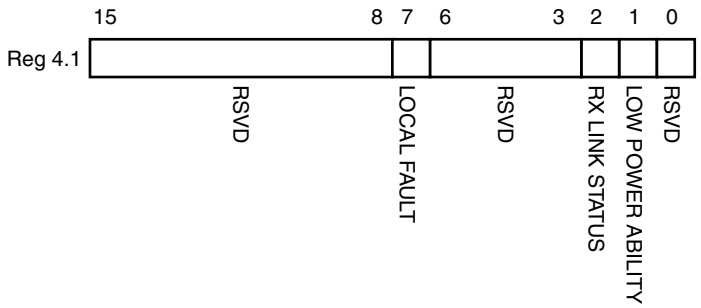

Figure 2-30: **PHY XS Status 1 Register**

Table 2-43 shows the PHY XS Status 1 register bit definitions.

Table 2-43: **PHY XS Status 1 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.1.15:8	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
4.1.7	Local Fault	1 = Local fault detected 0 = No Local Fault detected This bit is set to 1 whenever either of the bits 4.8.11, 4.8.10 are set to 1.	R/O	-
4.1.6:3	Reserved	The block always returns 0s for these bits and ignores writes.	R/O	All 0s
4.1.2	PHY XS Receive Link Status	1 = The PHY XS receive link is up. 0 =The PHY XS receive link is down. This is a latching Low version of bit 4.24.12.	R/O Self-setting	-
4.1.1	Power Down Ability	The block always returns 1 for this bit.	R/O	1
4.1.0	Reserved	The block always returns 0 for this bit and ignores writes.	R/O	0

MDIO Registers 4.2 and 4.3: PHY XS Device Identifier

Figure 2-31 shows the MDIO Registers 4.2 and 4.3: PHY XS Device Identifier.

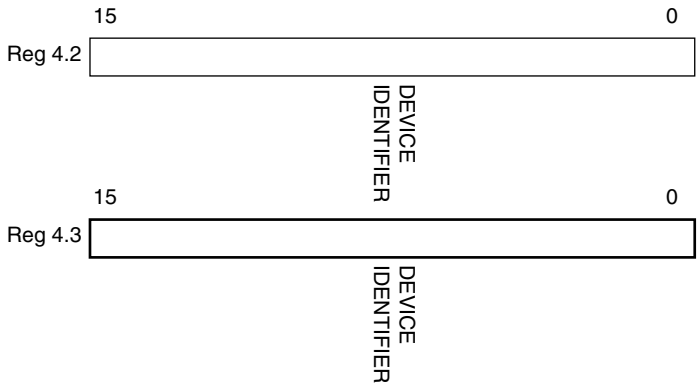

Figure 2-31: **PHY XS Device Identifier Registers**

Table 2-44 shows the PHY XS Devices Identifier registers bit definitions.

Table 2-44: **PHY XS Device Identifier Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.2.15:0	PHY XS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
4.3.15:0	PHY XS Identifier	The block always returns 0 for these bits and ignores writes.	R/O	All 0s

MDIO Register 4.4: PHY XS Speed Ability

Figure 2-32 shows the MDIO Register 4.4: PHY XS Speed Ability.

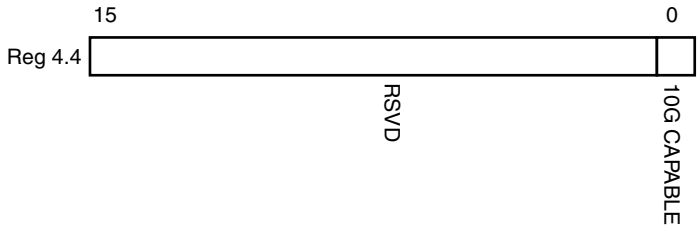

Figure 2-32: **PHY XS Speed Ability Register**

Table 2-45 shows the PHY XS Speed Ability register bit definitions.

Table 2-45: **PHY XS Speed Ability Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.4.15:1	Reserved	The block always returns 0 for these bits and ignores writes.	R/O	All 0s
4.4.0	10G Capable	The block always returns 1 for this bit and ignores writes.	R/O	1

MDIO Registers 4.5 and 4.6: PHY XS Devices in Package

Figure 2-33 shows the MDIO Registers 4.5 and 4.6: PHY XS Devices in Package.

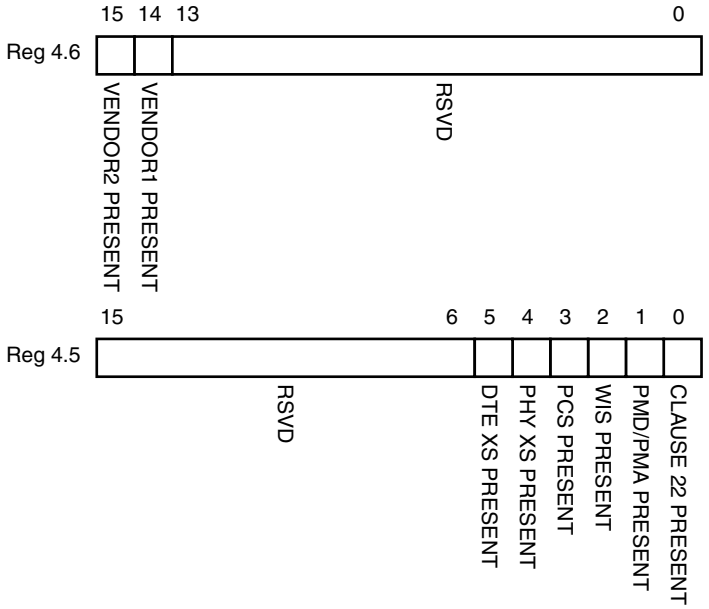

Figure 2-33: **PHY XS Devices in Package Registers**

Table 2-46 shows the PHY XS Devices in Package registers bit definitions.

Table 2-46: **PHY XS Devices in Package Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.6.15	Vendor-specific Device 2 present	The block always returns 0 for this bit.	R/O	0
4.6.14	Vendor-specific Device 1 present	The block always returns 0 for this bit.	R/O	0
4.6.13:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.5.15:6	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.5.5	DTE XS present	The block always returns 0 for this bit.	R/O	0
4.5.4	PHY XS present	The block always returns 1 for this bit.	R/O	1
4.5.3	PCS present	The block always returns 0 for this bit.	R/O	0
4.5.2	WIS present	The block always returns 0 for this bit.	R/O	0
4.5.1	PMA/PMD present	The block always returns 0 for this bit.	R/O	0
4.5.0	Clause 22 device present	The block always returns 0 for this bit.	R/O	0

MDIO Register 4.8: PHY XS Status 2

Figure 2-34 shows the MDIO Register 4.8: PHY XS Status 2.

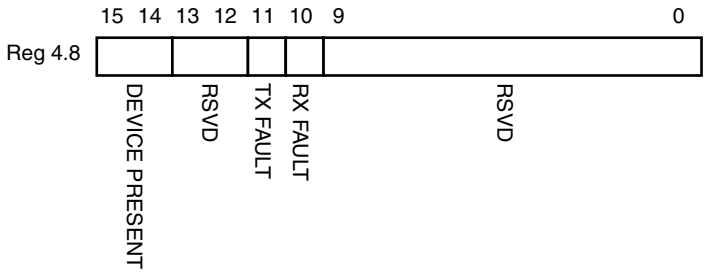

Figure 2-34: **PHY XS Status 2 Register**

Table 2-47 shows the PHY XS Status 2 register bit definitions.

Table 2-47: **PHY XS Status 2 Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.8.15:14	Device present	The block always returns 10.	R/O	10
4.8.13:12	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.8.11	Transmit local fault	1 = Fault condition on transmit path 0 = No fault condition on transmit path	R/O Latching High	-
4.8.10	Receive local fault	1 = Fault condition on receive path 0 = No fault condition on receive path	R/O Latching High	-
4.8.9:0	Reserved	The block always returns 0 for these bits.	R/O	All 0s

MDIO Registers 4.14 and 4.15: PHY XS Package Identifier

Figure 2-35 shows the MDIO 4.14 and 4.15 Registers: PHY XS Package Identifier.

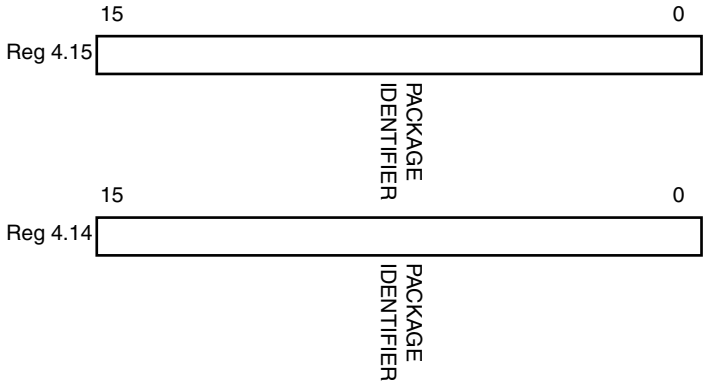

Figure 2-35: **PHY XS Package Identifier Registers**

Table 2-48 shows the Package Identifier registers bit definitions.

Table 2-48: **Package Identifier Registers Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.15.15:0	PHY XS Package Identifier	The block always returns 0 for these bits.	R/O	All 0s
4.14.15:0	PHY XS Package Identifier	The block always returns 0 for these bits.	R/O	All 0s

MDIO Register 4.24: 10G PHY XGXS Lane Status

Figure 2-36 shows the MDIO Register 4.24: 10G XGXS Lane Status.

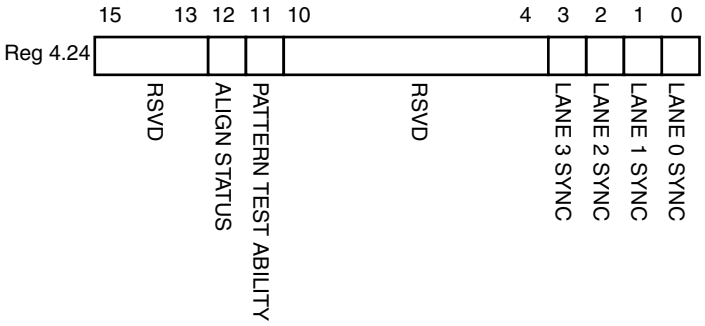

Figure 2-36: **10G PHY XGXS Lane Status Register**

Table 2-49 shows the 10G PHY XGXS Lane register bit definitions.

Table 2-49: **10G PHY XGXS Lane Status Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.24.15:13	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.24.12	PHY XGXS Lane Alignment Status	1 = PHY XGXS receive lanes aligned 0 = PHY XGXS receive lanes not aligned.	RO	-
4.24.11	Pattern Testing Ability	The block always returns 1 for this bit.	R/O	1
4.24.10:4	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.24.3	Lane 3 Sync	1 = Lane 3 is synchronized 0 = Lane 3 is not synchronized.	R/O	-
4.24.2	Lane 2 Sync	1 = Lane 2 is synchronized 0 = Lane 2 is not synchronized.	R/O	-
4.24.1	Lane 1 Sync	1 = Lane 1 is synchronized 0 = Lane 1 is not synchronized.	R/O	-
4.24.0	Lane 0 Sync	1 = Lane 0 is synchronized 0 = Lane 0 is not synchronized.	R/O	-

MDIO Register 4.25: 10G PHY XGXS Test Control

Figure 2-37 shows the MDIO Register 4.25: 10G XGXS Test Control.

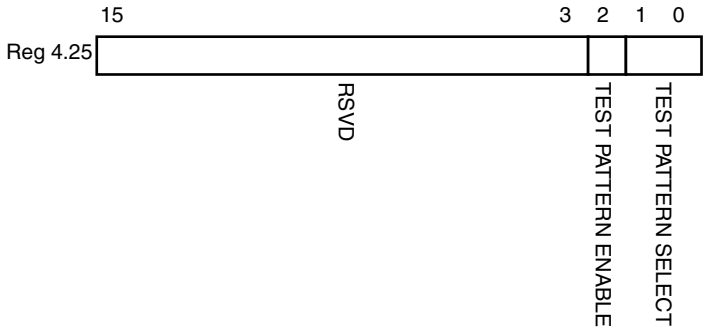

Figure 2-37: **10G PHY XGXS Test Control Register**

Table 2-50 shows the 10G PHY XGXS Test Control register bit definitions.

Table 2-50: **10G PHY XGXS Test Control Register Bit Definitions**

Bit(s)	Name	Description	Attributes	Default Value
4.25.15:3	Reserved	The block always returns 0 for these bits.	R/O	All 0s
4.25.2	Transmit Test Pattern Enable	1 = Transmit test pattern enable 0 = Transmit test pattern disabled	R/W	0
4.25.1:0	Test Pattern Select	11 = Reserved 10 = Mixed frequency test pattern 01 = Low frequency test pattern 00 = High frequency test pattern	R/W	00

Configuration and Status Vectors

If the RXAUI core is generated without an MDIO interface, the key configuration and status information is carried on simple bit vectors, which are:

- configuration_vector[6:0]
- status_vector[7:0]

Table 2-51 shows the Configuration Vector bit definitions.

Table 2-51: **Configuration Vector Bit Definitions**

Bit(s)	Name	Description
0	Loopback	Sets serial loopback in the device-specific transceivers. See bit 5.0.14 in Table 2-32, page 36.
1	Power Down	Sets the device-specific transceivers into power down mode. See bit 5.0.11 in Table 2-32, page 36.
2	Reset Local Fault	Clears both TX Local Fault and RX Local Fault bits (status_vector[0] and status_vector[1]). See Table 2-52. This bit should be driven by a register on the same clock domain as the RXAUI core.
3	Reset Rx Link Status	Sets the RX Link Status bit (status_vector[7]). See Table 2-52. This bit should be driven by a register on the same clock domain as the RXAUI core.
4	Test Enable	Enables transmit test pattern generation. See bit 5.25.2 in Table 2-40, page 43.
6:5	Test Select(1:0)	Selects the test pattern. See bits 5.25.1:0 in Table 2-40, page 43.

Table 2-52 shows the Status Vector bit definitions.

Table 2-52: **Status Vector Bit Definitions**

Bit(s)	Name	Description
0	Tx Local Fault	1 if there is a fault in the transmit path, otherwise 0. See bit 5.8.11 in Table 2-37, page 40. Latches High. Cleared by rising edge on configuration_vector[2].
1	Rx Local Fault	1 if there is a fault in the receive path, otherwise 0. See bit 5.8.10 in Table 2-37, page 40. Latches High. Cleared by rising edge on configuration_vector[2].
5:2	Synchronization	Each bit is 1 if the corresponding RXAUI lane is synchronized on receive, otherwise 0. See bits 5.24.3:0 in Table 2-38, page 41. These four bits are also used to generate the sync_status[3:0] signal described in Table 2-53.
6	Alignment	1 if the RXAUI receiver is aligned over all four logical XAUI lanes, otherwise 0. See bit 5.24.12 in Table 2-38, page 41. This is also used to generate the align_status signal described in Table 2-53.
7	Rx Link Status	1 if the Receiver link is up, otherwise 0. See bit 5.1.2 in Table 2-33, page 37. Latches Low. Cleared by rising edge on configuration_vector[3].

Bits 0 and 1 of the status_vector port, the 'Local Fault' bits, are latching-high and cleared low by bit 2 of the configuration_vector port. Figure 2-38 shows how the status bits are cleared.

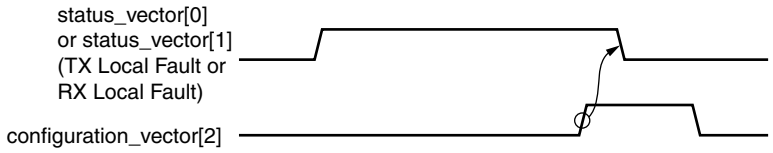

Figure 2-38: **Clearing the Local Fault Status Bits**

Bit 7 of the status_vector port, the 'RX Link Status' bit, is latching-low and set high by bit 3 of the configuration vector. Figure 2-39 shows how the status bit is set.

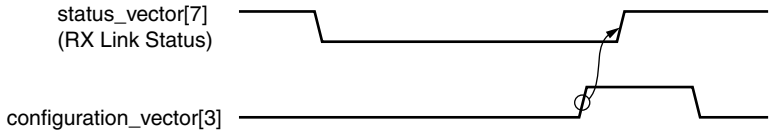

Figure 2-39: **Setting the RX Link Status Bit**

Alignment and Synchronization Status Ports

In addition to the configuration and status interfaces described in the previous section, there are always available two output ports signalling the alignment and synchronization status of the receiver. (Table 2-53.)

Table 2-53: **Alignment Status and Synchronization Status Ports**

Port Name	Description
align_status	1 when the RXAUI receiver is aligned across all four XAUI logical lanes, 0 otherwise.
sync_status[3:0]	Each pin is 1 when the respective XAUI logical lane receiver is synchronized to byte boundaries, 0 otherwise.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

General Design Guidelines

This section describes the steps required to turn a RXAUI core into a fully-functioning design with user-application logic. It is important to realize that not all implementations require all of the design steps listed in this chapter. Follow the logic design guidelines in this manual carefully.

Use the Example Design as a Starting Point

Each instance of the RXAUI core is delivered with an example design that can be implemented in an FPGA and simulated. This design can be used as a starting point for your own design or can be used to sanity-check your application in the event of difficulty.

See Chapter 6, Detailed Example Design (Vivado™ Design Suite) and Chapter 9, Detailed Example Design (ISE® Design Suite) for information about using and customizing the example designs for the RXAUI core.

Know the Degree of Difficulty

RXAUI designs are challenging to implement in any technology, and the degree of difficulty is further influenced by:

- Maximum system clock frequency
- Targeted device architecture
- Nature of your application

All RXAUI implementations need careful attention to system performance requirements. Pipelining, logic mapping, placement constraints, and logic duplication are all methods that help boost system performance.

Keep It Registered

To simplify timing and increase system performance in an FPGA design, keep all inputs and outputs registered between your application and the core. This means that all inputs and outputs from your application should come from, or connect to a flip-flop. While registering signals might not be possible for all paths, it simplifies timing analysis and makes it easier for the Xilinx tools to place and route the design.

Recognize Timing Critical Signals

The UCF provided with the example design for the core identifies the critical signals and the timing constraints that should be applied. See Chapter 5, Constraining the Core (Vivado Design Suite or Chapter 9, Detailed Example Design (ISE Design Suite) for further information.

Use Supported Design Flows

Vivado Design Tools

The core HDL is added to the open Vivado project. Later the core is synthesized along with the rest of the project as part of project synthesis.

ISE Design Tools

The core is synthesized in the CORE Generator™ tool and is delivered to you as an NGC netlist. The example implementation scripts provided currently use Xilinx Synthesis Technology (XST) as the synthesis tool for the HDL example design that is delivered with the core. Other synthesis tools can be used for your application logic; the core is always unknown to the synthesis tool and appears as a black box.

Post synthesis, only Xilinx ISE Design Suite v14.3 tools are supported.

Make Only Allowed Modifications

The RXAUI core is not user-modifiable. Do not make modifications as they can have adverse effects on system timing and protocol compliance. Supported user configurations of the RXAUI core can only be made by selecting the options from within the Vivado Design Suite or CORE Generator tool when the core is generated. See Chapter 4, Customizing and Generating the Core (Vivado Design Suite) or Chapter 7, Customizing and Generating the Core (ISE Design Suite).

Clocking

The clocking schemes in this section are illustrative only and might require customization for a specific application. See Table 2-10 for information on the clock ports.

Reference Clock

The transceivers typically use a reference clock of 156.25 MHz to operate at a line rate of 6.25 Gb/s.

Transceiver Placement

Common to all schemes shown is that a single IBUFDS_GTXE1 (Virtex-6 FPGA) or IBUFDS_GTE2 (7 series FPGAs) is used to feed the reference clocks for all transceivers. In addition, timing requirements are more easily met if both transceivers are placed next to each other.

For details about transceiver clock distribution, see the section on Clocking in the *Virtex-6 FPGA GTX Transceivers User Guide* [Ref 9], the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 10] and the *7 Series FPGAs GTP Transceivers User Guide* [Ref 11].

Dune Networks RXAUI (7 Series FPGAs)

The clocking scheme for 7 Series GTX transceivers is as shown in Figure 3-1.

The transceiver primitives require a 156.25 MHz clock. The 156.25 MHz clock is used as the clock for the netlist part of the RXAUI core and is typically also used for your logic.

A dedicated clock is used by the transceivers. The example design uses a 50 MHz clock. Choosing a different frequency might allow sharing of clock resources. See the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 10] and the *7 Series FPGAs GTP Transceivers User Guide* [Ref 11] for details about this clock.

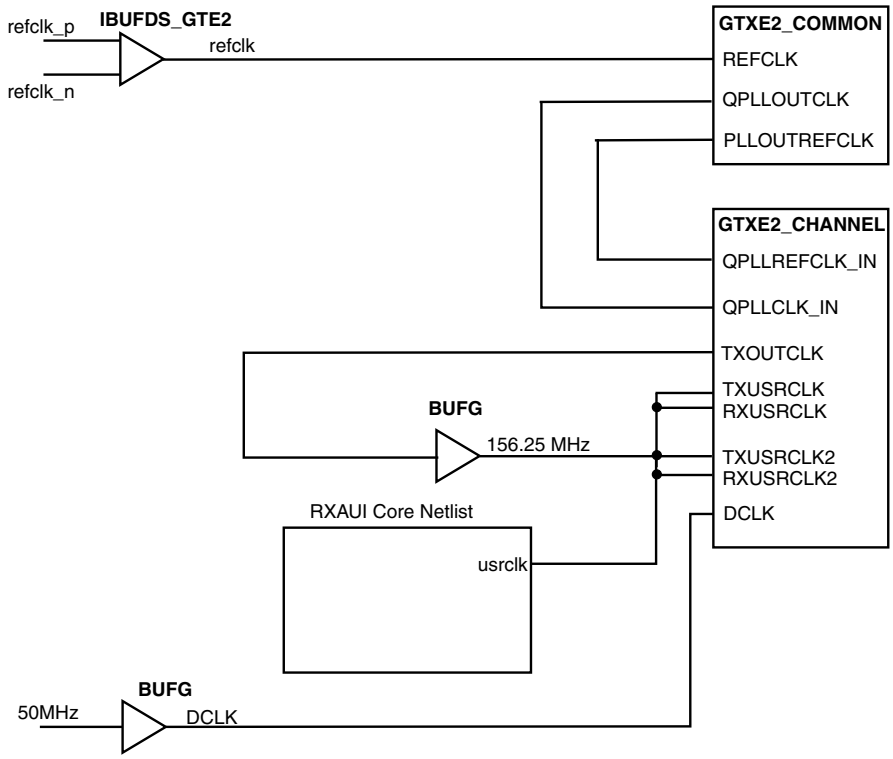

Figure 3-1: **Clock Scheme for Dune Networks RXAUI: 7 Series GTX Transceivers**

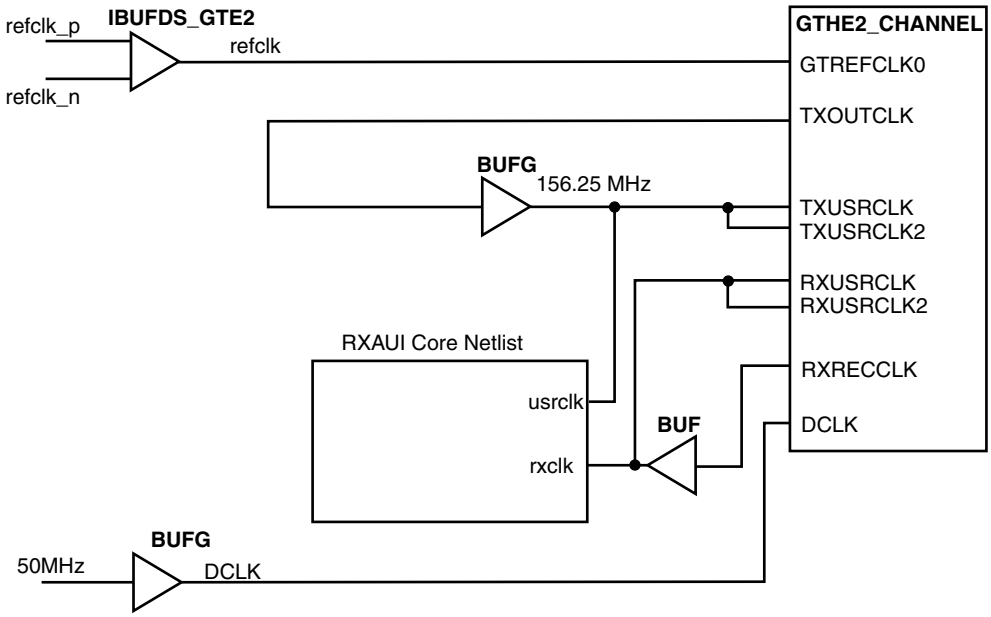

Figure 3-2: **Clock Scheme for Dune Networks RXAUI: 7 Series GTH Transceivers**

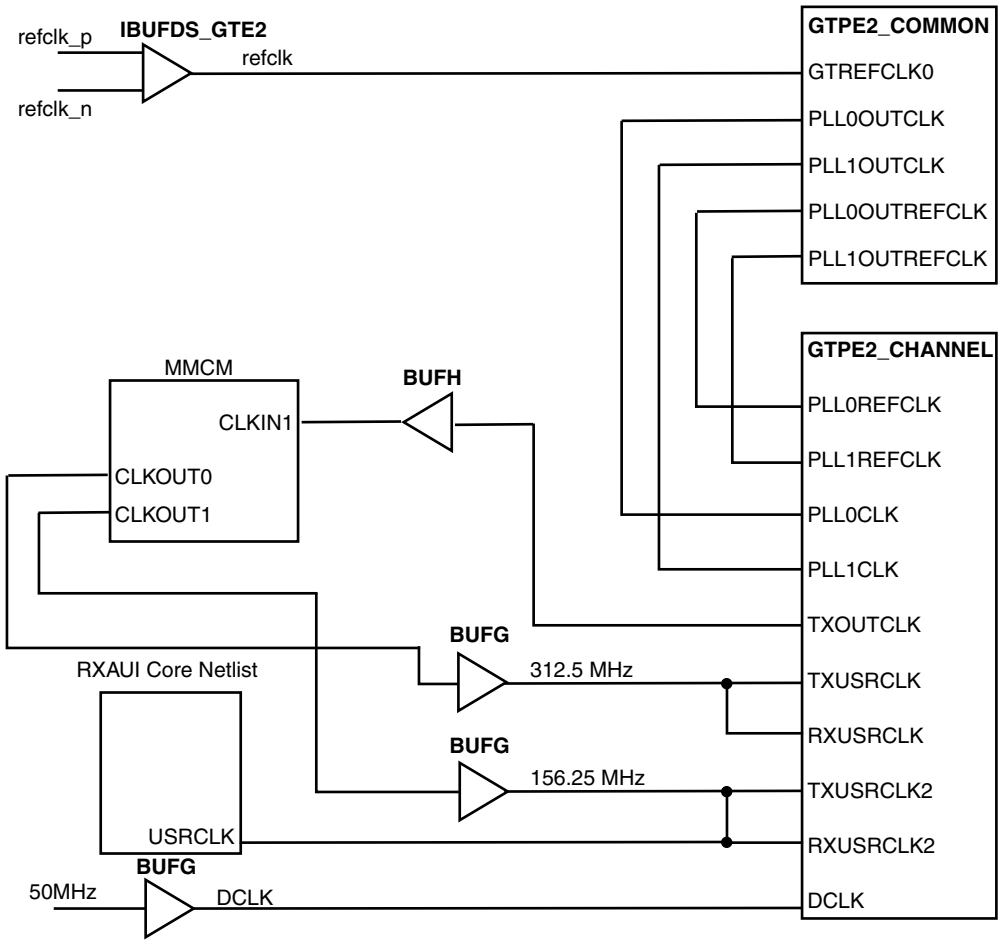

Figure 3-3: **Clock Scheme for Dune Networks RXAUI: 7 Series GTP Transceivers**

Dune Networks RXAUI (Virtex-6 LXT/SXT/HXT FPGAs)

The clocking scheme is shown in Figure 3-4.

The GTX transceiver primitives require a 156.25 MHz clock and a 312.5 MHz clock. The 156.25 MHz clock is used as the clock for the netlist part of the RXAUI core and is typically also used for your logic.

A dedicated clock is used by the GTX transceivers. The example design uses a 50 MHz clock. Choosing a different frequency allows sharing of clock resources. See the *Virtex-6 FPGA GTX Transceivers User Guide* [Ref 9] for details about this clock.

Note: For the Virtex-6 FPGA GTX transceiver, the MMCM must be located in the same clock region as the GTX transceiver. See Virtex-6 FPGA MMCM Restrictions.

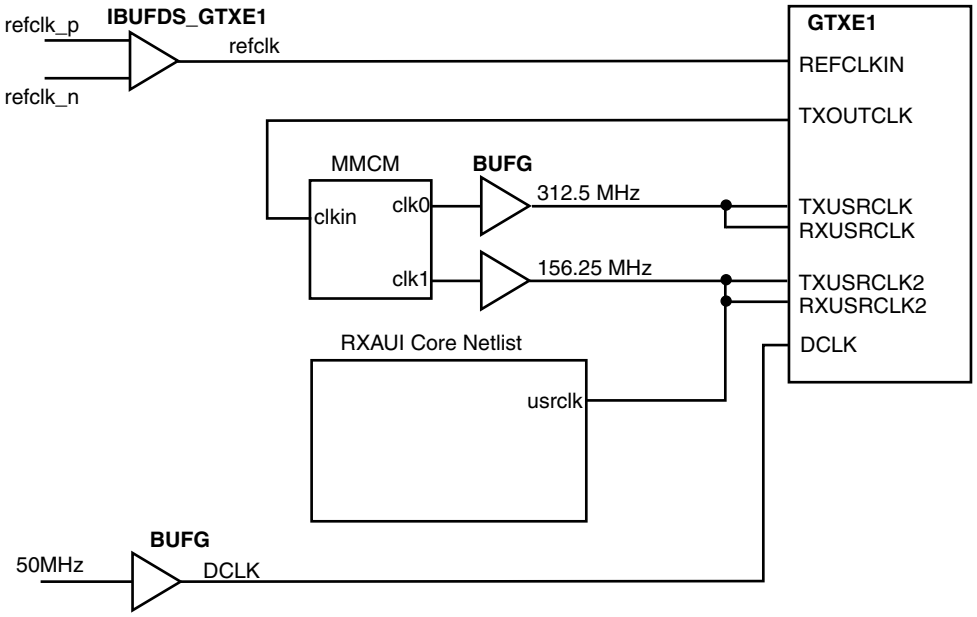

Figure 3-4: **Clock Scheme for Dune Networks RXAUI: Virtex-6 LXT/SXT/XHT FPGAs**

Marvell RXAUI (7 Series FPGAs)

The transceivers require a 156.25 MHz clock this is used to clock the transmit part of the transceiver (Figure 3-5). The 156.25 MHz clock is used as the clock for the netlist part of the RXAUI core and is typically also used for your logic.

The RXAUI core also uses the recovered clock from the transceivers to clock the receive path. This is connected to the core using a BUFH.

A dedicated clock is used by the transceivers. The example design uses a 50 MHz clock. Choosing a different frequency might allow sharing of clock resources. See the *7 Series FPGAs GTX/GTP Transceivers User Guide* [Ref 10] and the *7 Series FPGAs GTP Transceivers User Guide* [Ref 11] for details about this clock.

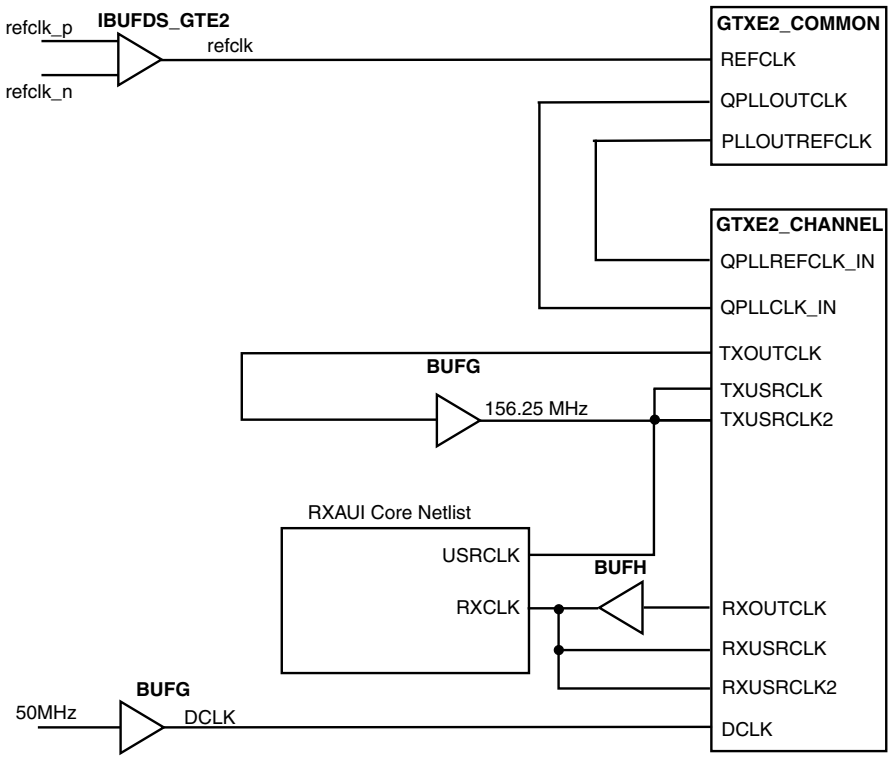

Figure 3-5: **Clock Scheme for Marvell RXAUI: 7 Series GTX Transceivers**

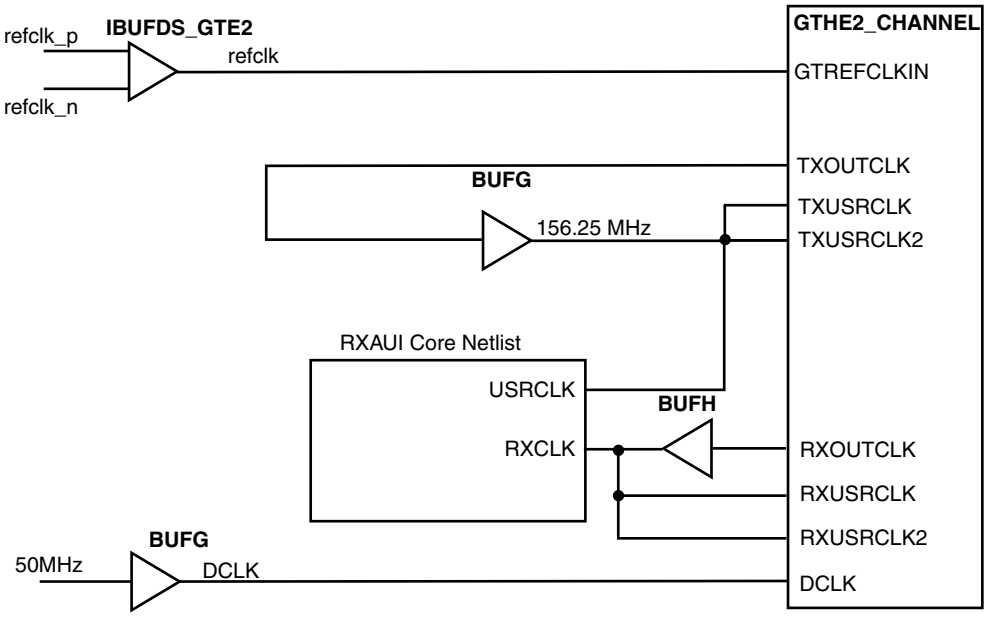

Figure 3-6: **Clock Scheme for Marvell RXAUI: 7 Series GTH Transceivers**

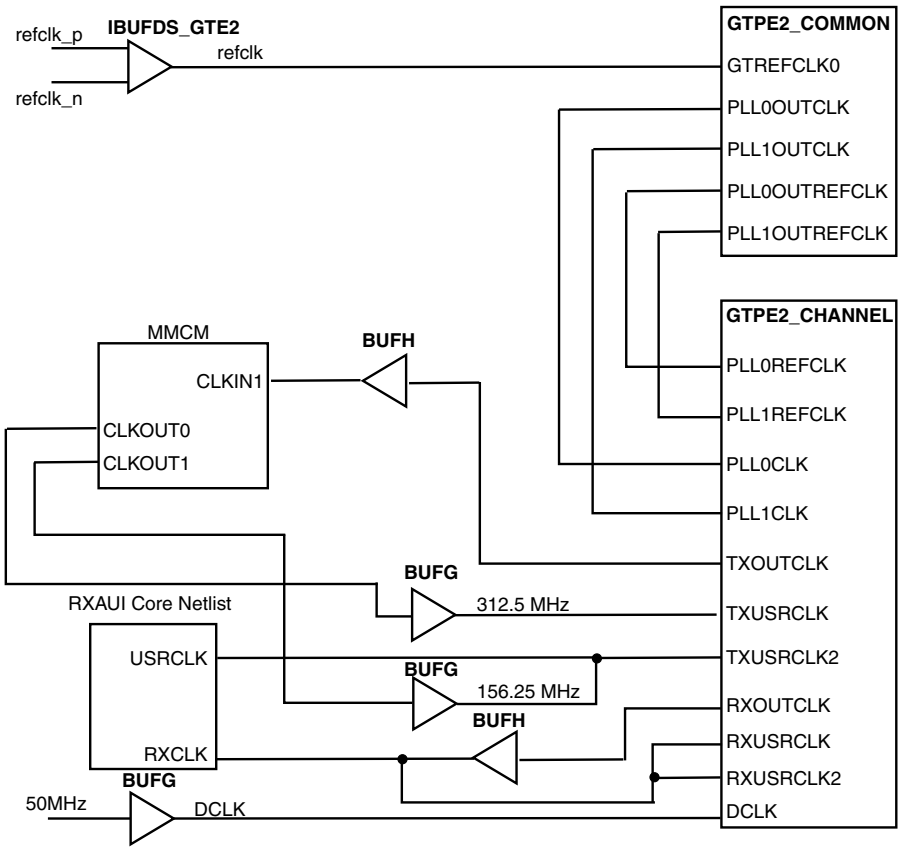

Figure 3-7: **Clock Scheme for Marvell RXAUI: 7 Series GTP Transceivers**

Marvell Internal Client-Side Interface (Virtex-6 LXT/SXT/HXT FPGAs)

The GTX transceivers require a 156.25 MHz clock and a 312.5 MHz clock. In Marvell mode this is used to clock the transmit part of the transceiver (Figure 3-8). The 156.25 MHz clock is used as the clock for the netlist part of the RXAUI core and is typically also used for your logic.

The RXAUI core also uses the recovered clock from the GTX transceivers to clock the receive path. This is connected to the core through a BUFR.

A dedicated clock is used by the GTX transceivers. The example design uses a 50 MHz clock. Choosing a different frequency might allow sharing of clock resources. See the *Virtex-6 FPGA GTX Transceivers User Guide* [Ref 9] and the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 10] for details about this clock.

Note: For the Virtex-6 FPGA GTX transceiver, the MMCM must be located in the same clock region as the GTX transceiver. See Virtex-6 FPGA MMCM Restrictions.

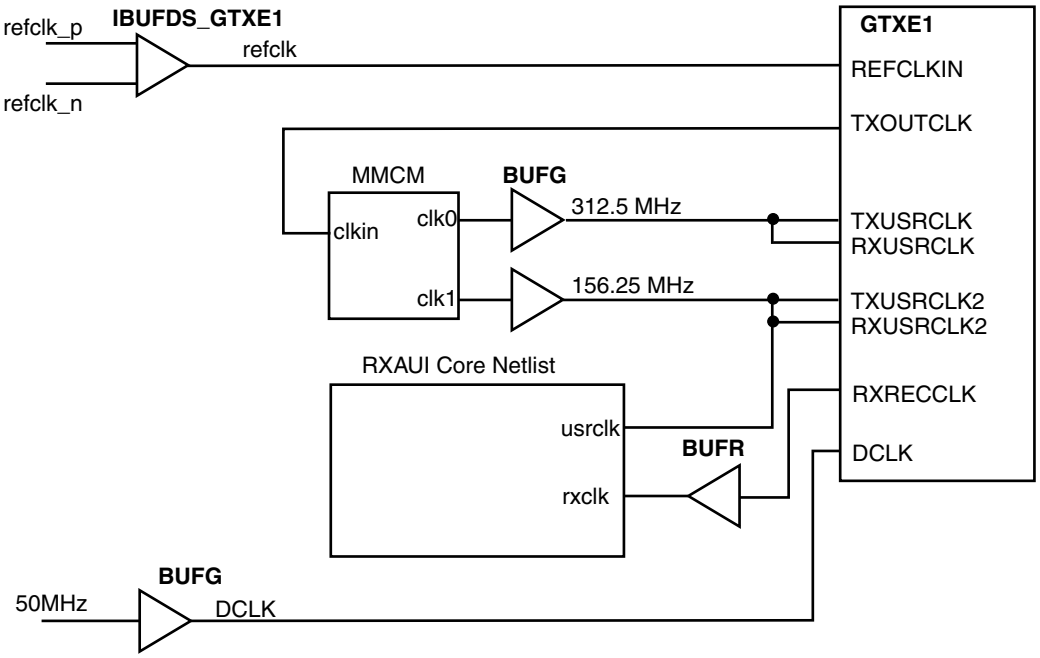

Figure 3-8: **Clock Scheme for Marvell RXAUI: Virtex-6 LXT/SXT/HXT FPGAs**

Virtex-6 FPGA MMCM Restrictions

GTX TXOUTCLK must drive the MMCM directly with no BUFG in the path. By default, in the wrapper, TXOUTCLK already drives the MMCM directly with no BUFG in the path. This means that the MMCM is restricted to the same region as the GTX transceiver and you must not add a BUFG to move the MMCM out of the region. Each GTX Quad spans an entire clocking region and there are two available MMCMs per clocking region; the MMCM used must be constrained to the same clock region otherwise a BUFG is automatically inserted. These locations can be determined using the *Virtex-6 FPGA Packaging and Pinout Specifications* [Ref 12].

Resets

See Table 2-10 for information on the reset ports. All register resets within the RXAUI core netlist are synchronous to the `usrclk` port.

Design Considerations

This section describes considerations that can apply in particular design cases.

Multiple Core Instances

To instantiate multiple cores, instantiate the RXAUI block level component multiple times. The clocking resources can be shared between multiple instances. See the *Virtex-6 FPGA GTX Transceivers User Guide* [Ref 9] and the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 10] for details on sharing the reference clock and any limitations.

In Virtex-7 and Kintex-7 devices, the reference clock can be shared from a neighboring quad. Logic clocks cannot be shared between core instances with the supplied design. The USRCLKs on each core and quad of transceivers must be sourced from the TXOUTCLK port of that quad. Some modification of the example design is required to enable the GTXE2_COMMON to be shared between two instances of the RXAUI core.

Receiver Termination: Virtex-7 and Kintex-7 FPGAs

See the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 10] and the *7 Series FPGAs GTP Transceivers User Guide* [Ref 11].

Receiver Termination: Virtex-6 FPGAs

The receiver termination must be set correctly. The default setting is 2/3 VTTRX. See the *Virtex-6 FPGA GTX Transceivers User Guide* [Ref 9].

Protocol Description

Data Interface: Internal Interfaces

Internal 64-bit SDR Client-side Interface

The 64-bit single-data rate (SDR) client-side interface is based upon the 32-bit XGMII-like interface. The bus is demultiplexed from 32- bits wide to 64-bits wide on a single rising clock edge. This demultiplexing is done by extending the bus upwards so that there are now eight lanes of data numbered 0-7; the lanes are organized such that data appearing on lanes 4–7 is transmitted or received *later* in time than that in lanes 0-3.

The mapping of lanes to data bits is shown in Table 3-1. The lane number is also the index of the control bit for that particular lane; for example, `XGMII_TXC[2]` and `XGMII_TXD[23:16]` are the control and data bits respectively for lane 2.

Table 3-1: **XGMII_TXD, XGMII_RXD Lanes for Internal 64-bit Client-Side Interface**

Lane	XGMII_TXD, XGMII_RXD Bits
0	7:0
1	15:8
2	23:16
3	31:24
4	39:32
5	47:40
6	55:48
7	63:56

Definitions of Control Characters

Reference is regularly made to certain XGMII control characters signifying Start, Terminate, Error and others. These control characters all have in common that the control line for that lane is 1 for the character and a certain data byte value. The relevant characters are defined in the *IEEE Std. 802.3-2008* and are reproduced in Table 3-2 for reference.

Table 3-2: **Partial list of XGMII Characters**

Data (Hex)	Control	Name, Abbreviation
00 to FF	0	Data (D)
07	1	Idle (I)
FB	1	Start (S)
FD	1	Terminate (T)
FE	1	Error (E)

Interfacing to the Transmit Client Interface

Internal 64-bit Client-Side Interface

The timing of a data frame transmission using the internal 64-bit client-side interface is shown in Figure 3-9. The beginning of the data frame is shown by the presence of the Start character (the /S/ codegroup in lane 4 of Figure 3-9) followed by data characters in lanes 5, 6, and 7. Alternatively the start of the data frame can be marked by the occurrence of a Start character in lane 0, with the data characters in lanes 1 to 7.

When the frame is complete, it is completed by a Terminate character (the T in lane 1 of Figure 3-9). The Terminate character can occur in any lane; the remaining lanes are padded by XGMII idle characters.

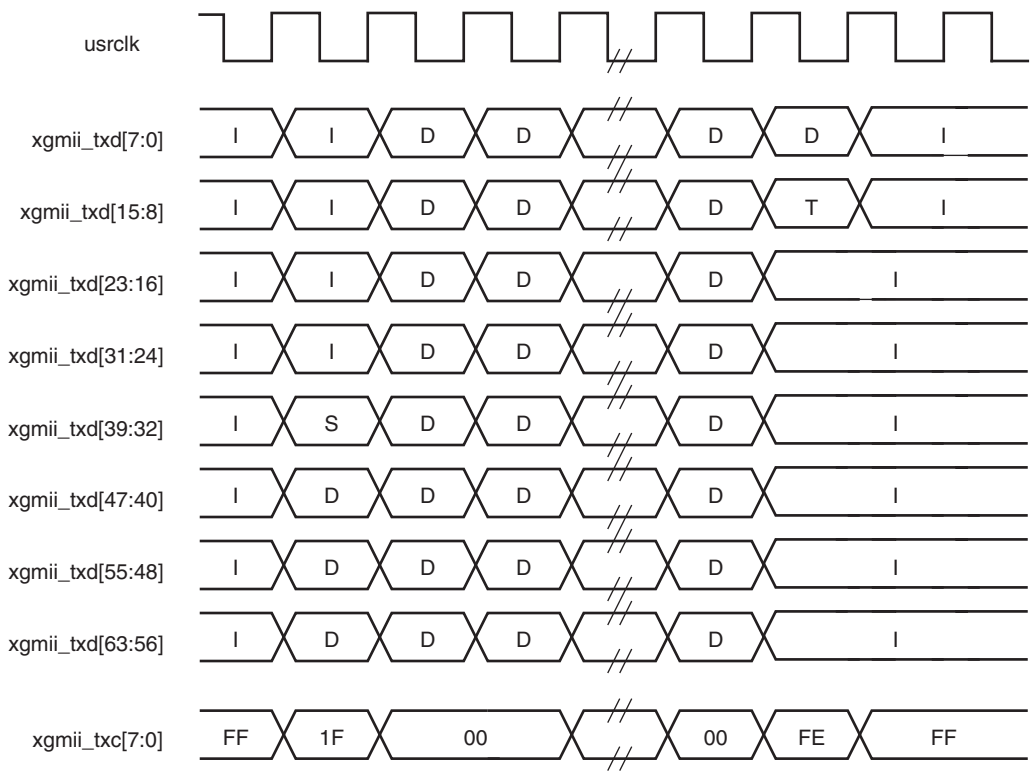

Figure 3-9: **Normal Frame Transmission Across the Internal 64-bit Client-Side I/F**

Figure 3-10 depicts a similar frame to that in Figure 3-9, with the exception that this frame is propagating an error. The error code is denoted by the letter E, with the relevant control bits set.

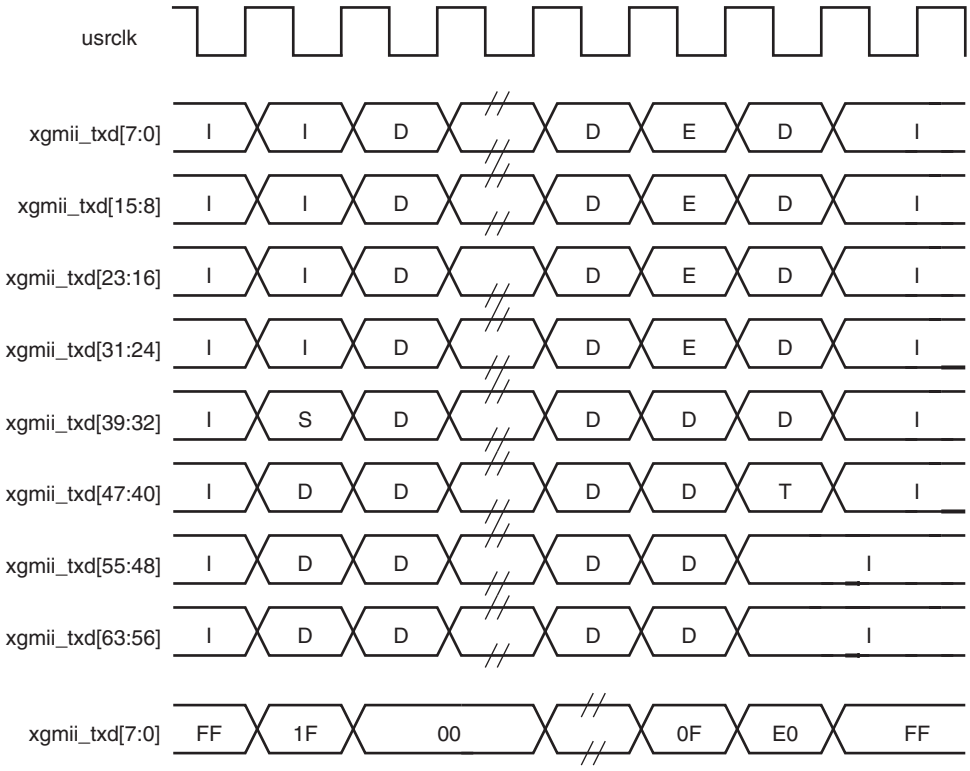

Figure 3-10: **Frame Transmission with Error Across Internal 64-bit Client-Side I/F**

Interfacing to the Receive Client Interface

Internal 64-Bit Client-Side Interface

The timing of a normal inbound frame transfer is shown in Figure 3-11. As in the transmit case, the frame is delimited by a Start character (S) and by a Terminate character (T). The Start character in this implementation can occur in either lane 0 or in lane 4. The Terminate character, T, can occur in any lane.

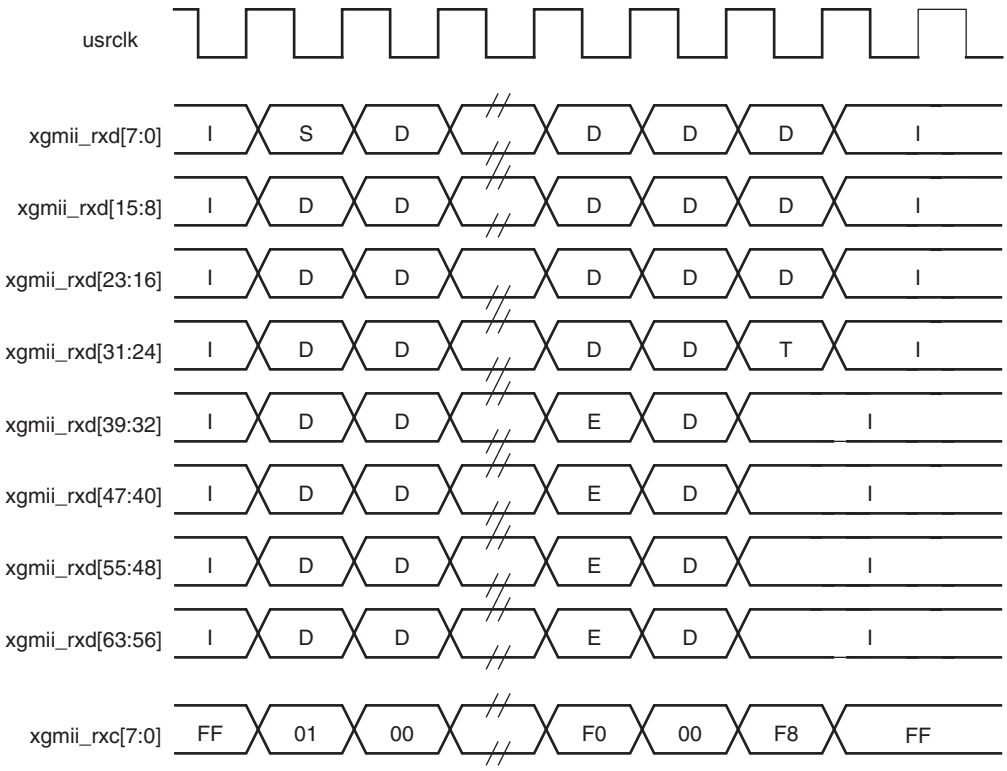

Figure 3-11: **Frame Reception Across the Internal 64-bit Client Interface**

Figure 3-12 shows an inbound frame of data propagating an error. In this instance, the error is propagated in lanes 4 to 7, shown by the letter E.

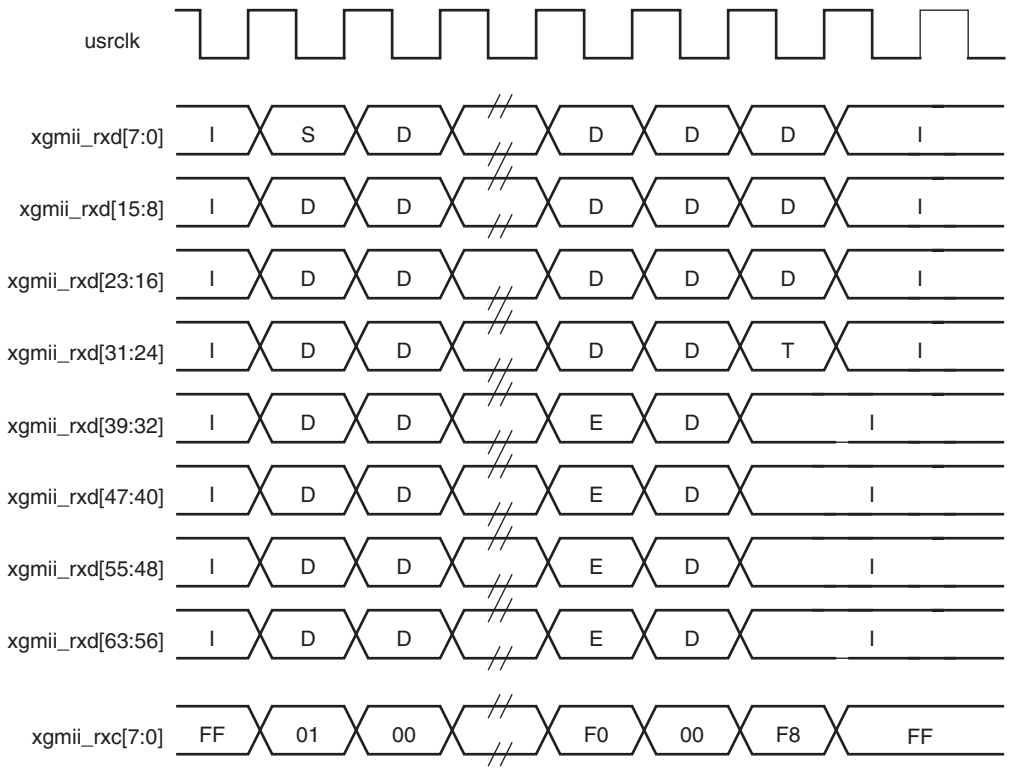

Figure 3-12: **Frame Reception with Error Across the Internal 64-bit Client Interface**

MDIO Interface

The Management Data Input/Output (MDIO) interface is a simple, low-speed 2-wire interface for management of the RXAUI core consisting of a clock signal and a bidirectional data signal. It is defined in clause 45 of I*EEE Standard 802.3-2008*.

An MDIO bus in a system consists of a single Station Management (STA) master management entity and several MDIO Managed Device (MMD) slave entities. Figure 3-13 illustrates a typical system. All transactions are initiated by the STA entity. The RXAUI core implements an MMD.

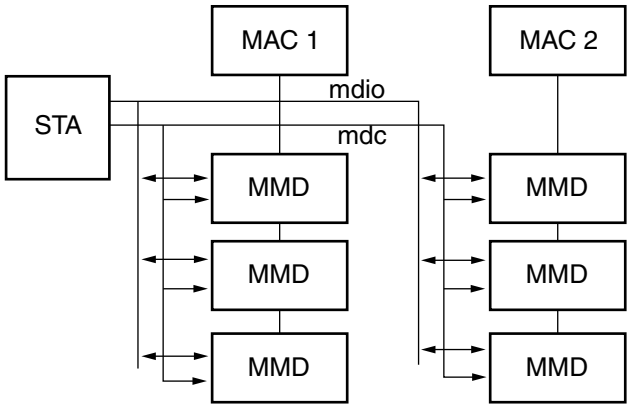

Figure 3-13: **A Typical MDIO-Managed System**

If implemented, the MDIO interface is implemented as four unidirectional signals. These can be used to drive a 3-state buffer either in the FPGA SelectIO™ interface buffer or in a separate device.

The `type_sel` port is registered into the core at FPGA configuration and core hard reset; changes after that time are ignored by the core. Table 3-3 shows the mapping of the `type_sel` setting to the implemented register map.

Table 3-3: **Mapping of type_sel Port Settings to MDIO Register Type**

type_sel setting	MDIO Register	Description
00 or 01	10GBASE-X PCS/PMA	When driving a 10GBASE-X PHY
10	DTE XGXS	When connected to a 10GMAC through XGMII
11	PHY XGXS	When connected to a PHY through XGMII

The `prtad[4:0]` port sets the port address of the core instance. Multiple instances of the same core can be supported on the same MDIO bus by setting the `prtad[4:0]` to a unique value for each instance; the RXAUI core ignores transactions with the PRTAD field set to a value other than that on its `prtad[4:0]` port.

MDIO Transactions

The MDIO interface should be driven from a STA master according to the protocol defined in *IEEE Std. 802.3-2008*. An outline of each transaction type is described in the following sections. In these sections, these abbreviations apply:

- PRE: preamble
- ST: start
- OP: operation code
- PRTAD: port address
- DEVAD: device address
- TA: turnaround

Set Address Transaction

Figure 3-14 shows an Address transaction defined by OP=00. Set Address is used to set the internal 16-bit address register of the RXAUI core for subsequent data transactions (called the 'current address' in the following sections).

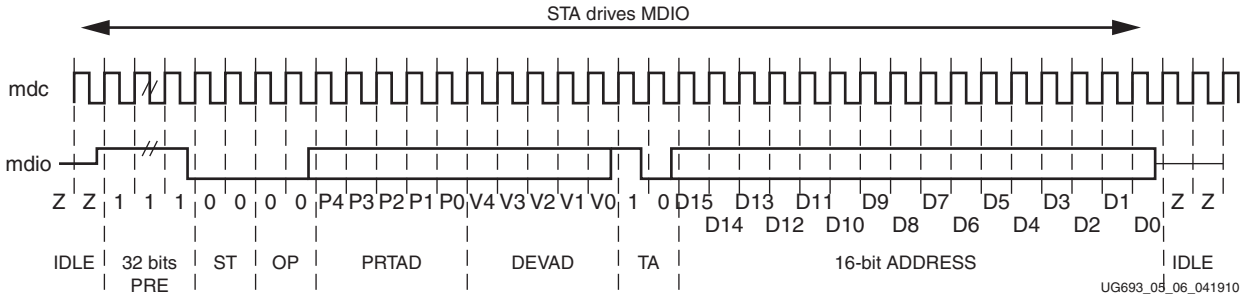

Figure 3-14: **MDIO Set Address Transaction**

Write Transaction

Figure 3-15 shows a Write transaction defined by OP=01. The RXAUI core takes the 16-bit word in the data field and writes it to the register at the current address.

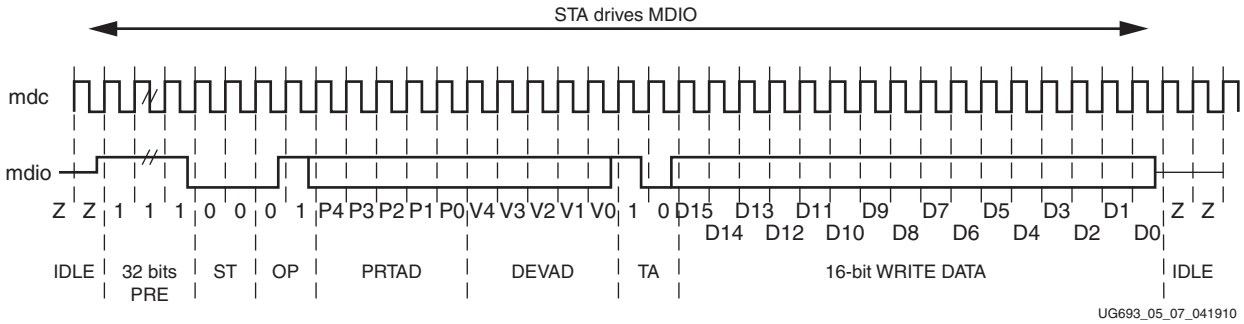

Figure 3-15: **MDIO Write Transaction**

Read Transaction

Figure 3-16 shows a Read transaction defined by OP=11. The RXAUI core returns the 16-bit word from the register at the current address.

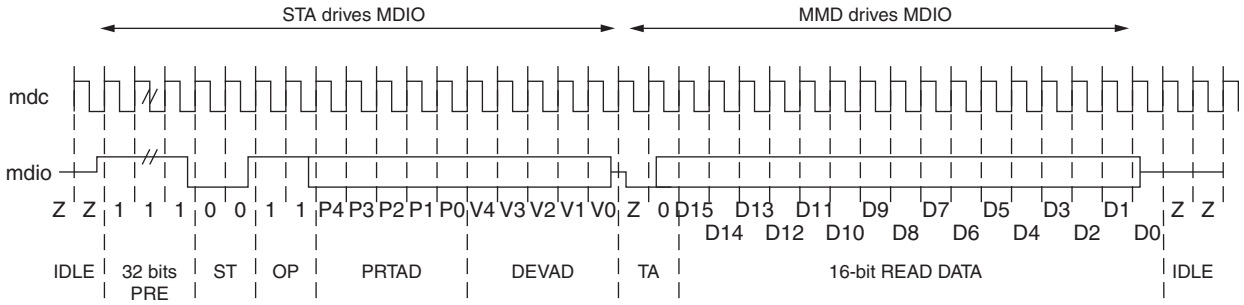

Figure 3-16: **MDIO Read Transaction**

Post-Read-increment-address Transaction

Figure 3-17 shows a Post-read-increment-address transaction, defined by OP=10. The RXAUI core returns the 16-bit word from the register at the current address then increments the current address. This allows sequential reading or writing by a STA master of a block of register addresses.

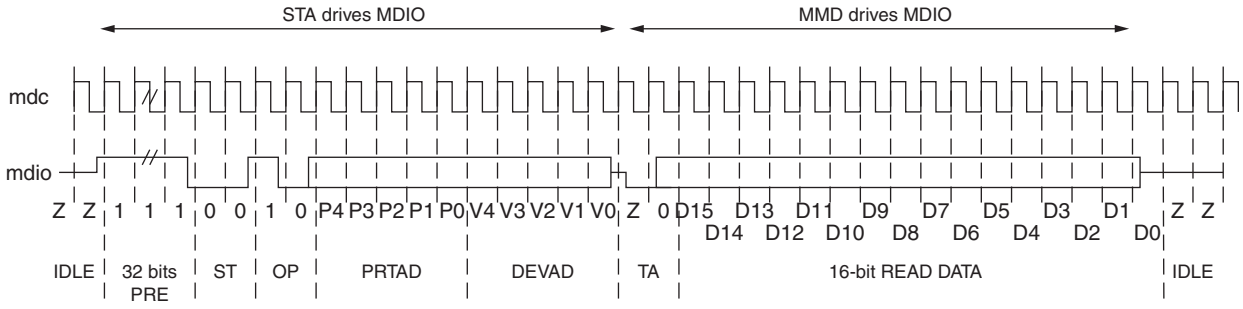

Figure 3-17: **MDIO Read-and-increment Transaction**

For detail on the MDIO registers, see MDIO Interface Registers.

SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core in the Vivado™ Design Suite environment.

GUI

Figure 4-1 displays the main screen for customizing the RXAUI core.

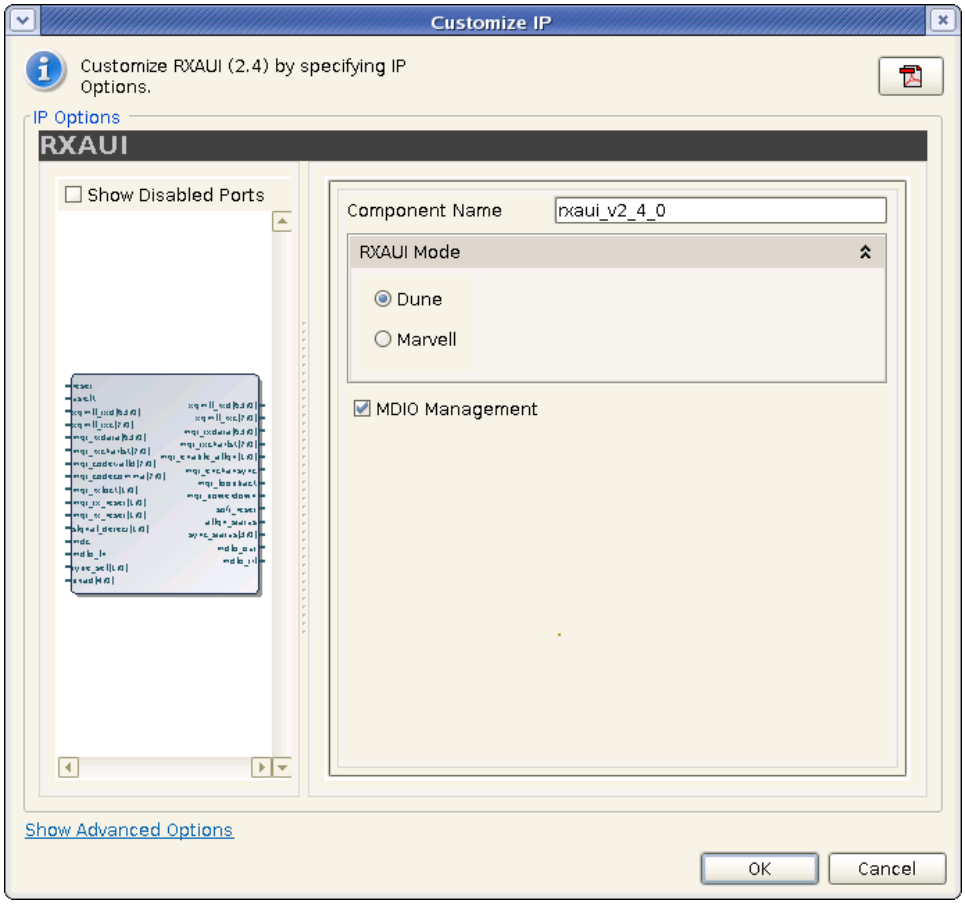

Figure 4-1: **RXAUI Main Screen**

For general help with starting and using the Vivado GUI, see the documentation supplied with the Vivado™ Design Suite.

Component Name

The component name is used as the base name of the output files generated for the core. Names must begin with a letter and must be composed from the following characters: a through z, 0 through 9 and "_" (underscore).

RXAUI Mode

This option selects between the three different RXAUI implementations that are supported. In 'Dune' mode, deinterleaving is performed on the |A||A| double character that is received.

In 'Marvell' mode multiplexing of the lanes is done to preserve 8B/10B disparity on the XAUI logical lanes. This means that the deinterleaving must be performed prior to any other processing. 8B/10B, deskew, and clock correction are all performed within the core.

MDIO Management

Select this option to implement the MDIO interface for managing the core. Deselect the option to remove the MDIO interface and expose a simple bit vector to manage the core.

The default is to implement the MDIO interface.

Output Generation

The core has various selectable output products. These can be generated by right-clicking

on the customized piece of IP in the Sources window.

- Examples - Source HDL and constraints for the example project
- Simulation - Simulation source files
- Synthesis - Synthesis source files
- Examples Simulation - Test bench for the example design
- Instantiation Template - Example instantiation template for the core level module.
- Miscellaneous - Simulation scripts and support files required for running netlist based functional simulation. The files delivered as part of this filegroup are not used or understood by Vivado tools. These files are delivered into the project source directory.

Constraining the Core

This chapter is applicable to the Vivado™ Design Suite environment.

This chapter describes how to constrain a design containing the RXAUI core. This is illustrated by the XDC delivered with the core at generation time. See Chapter 6, Detailed Example Design, for a complete description of the Vivado™ Design Suite output files.

> ***Caution!*** Not all constraints are relevant to specific implementations of the core; consult the XDC created with the core instance to see exactly what constraints are relevant.

Required Constraints

This section defines the constraint requirements for the core. Constraints are provided with an XDC file. An XDC is provided with the HDL example design to give a starting point for constraints for the user design. The following constraints are required.

If the MDIO interface is enabled, it should be constrained to 2.5 MHz.

```
set_max_delay 400.000 -from [get_cells -hierarchical -filter {NAME =~rxaui_block/
rxaui_core/U0/rxaui_inst/xaui_i/*management_1/mdio_interface_1/*_reg*}] -to
[get_cells -hierarchical -filter {NAME =~rxaui_block/rxaui_core/U0/rxaui_inst/
xaui_i/*management_1/*_reg*}]

set_max_delay 400.000 -from [get_cells -hierarchical -filter {NAME =~rxaui_block/
rxaui_core/U0/rxaui_inst/xaui_i/*management_1/mdio_interface_1/*_reg*}] -to
[get_cells -hierarchical -filter {NAME =~rxaui_block/rxaui_core/U0/rxaui_inst/
xaui_i/*management_1/mdio_interface_1/*_reg*}]
```

Clock Frequencies

A constraint is required to specify a 156.25 MHz clock. This is typically sourced from the TXOUTCLK_OUT port on the transceiver.

```
create_clock -name TXOUTCLK_OUT -period 6.400 [get_pins  rxaui_block/gt_wrapper_i/
gt0_<ComponentName>_gt_wrapper_i/gtxe2_i/TXOUTCLK]
```

Recovered Clock (Marvell Mode)

```
create_clock -name RXOUTCLK_OUT -period 3.200 [get_pins  rxaui_block/gt_wrapper_i/
gt0_<ComponentName>_gt_wrapper_i/gtxe2_i/RXOUTCLK]
```

General Clocking

If used, DCLK should also be specified:

```
create_clock -name dclk -period 20.000 [get_ports dclk]
```

This constraint defines the frequency of DCLK that is supplied to the transceivers. The example design uses a nominal 50 MHz clock.

Specify the clocks as asynchronous:

```
set_clock_groups -group [get_clocks dclk] -group [get_clocks RXOUTCLK_OUT]
-asynchronous
set_clock_groups -group [get_clocks TXOUTCLK_OUT] -group [get_clocks RXOUTCLK_OUT]
-asynchronous
set_clock_groups -group [get_clocks RXOUTCLK_OUT] -group [get_clocks TXOUTCLK_OUT]
-asynchronous
```

Clock Management

The Dune Networks RXAUI core has one clock domain:

- The `refclk` domain derived from the TXOUTCLK output of the GTX/GTH transceiver

The Marvell core has two clock domains:

- The refclk domain derived from the TXOUTCLK output of the GTX/GTH transceiver
- The rxclk domain derived from a single recovered clock output of a transceiver

Transceiver Placement

7 Series GTH Transceivers

```
set_property LOC GTHE2_CHANNEL_X0Y0 [get_cells rxaui_block/gt_wrapper_i/
gt0_<ComponentName>_gt_wrapper_i/gthe2_i]
set_property LOC GTHE2_CHANNEL_X0Y1 [get_cells rxaui_block/gt_wrapper_i/
gt1_<ComponentName>_gt_wrapper_i/gthe2_i]
```

7 Series GTX Transceivers

```
set_property LOC GTXE2_CHANNEL_X0Y0 [get_cells rxaui_block/gt_wrapper_i/
gt0_<ComponentName>_gt_wrapper_i/gtxe2_i]
set_property LOC GTXE2_CHANNEL_X0Y1 [get_cells rxaui_block/gt_wrapper_i/
gt1_<ComponentName>_gt_wrapper_i/gtxe2_i]
```

These constraints lock down the placement of the transceivers.

7 Series GTP Transceivers

```
set_property LOC GTPE2_CHANNEL_X0Y0 [get_cells rxaui_block/gt_wrapper_i/
gt0_<ComponentName>_gt_wrapper_i/gtpe2_i]
set_property LOC GTPE2_CHANNEL_X1Y0 [get_cells rxaui_block/gt_wrapper_i/
gt1_<ComponentName>_gt_wrapper_i/gtpe2_i]
```

MDIO

If the MDIO interface is enabled, it should be constrained to 2.5 MHz.

```
set_max_delay 400.000 -from [get_cells -hierarchical -filter {NAME =~
rxaui_block/rxaui_core/U0/rxaui_inst/xaui_i/*management_1/mdio_interface_1/*_reg*}]
-to [get_cells -hierarchical -filter {NAME =~
rxaui_block/rxaui_core/U0/rxaui_inst/xaui_i/*management_1/*_reg*}]
set_max_delay 400.000 -from [get_cells -hierarchical -filter {NAME =~
rxaui_block/rxaui_core/U0/rxaui_inst/xaui_i/*management_1/mdio_interface_1/*_reg*}]
-to [get_cells -hierarchical -filter {NAME =~
rxaui_block/rxaui_core/U0/rxaui_inst/xaui_i/*management_1/mdio_interface_1/*_reg*}]
```

Detailed Example Design

This chapter provides detailed information about the example design, including a description of the files and the directory structure generated by the Xilinx® Vivado™ Design Suite, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

Example Design

Figure 6-1 illustrates the default configuration of the example design.

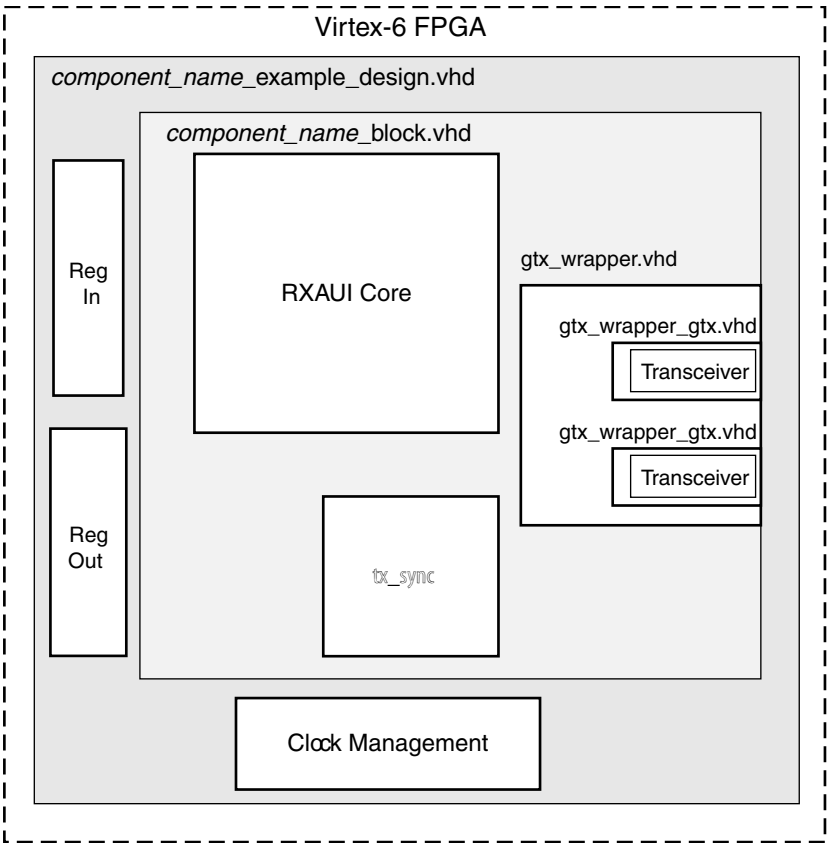

Figure 6-1: **RXAUI Example Design and Test Bench: Default Configuration**

The RXAUI example design consists of the following:

- Clock management logic and Clock Buffer Instances
- Re-timing registers on the parallel data interface, both on inputs and outputs
- An instance of the 'block' level module which contains the core, transceiver wrappers and associated logic.

The RXAUI Design Example has been tested with Xilinx® Vivado™ Design Suite, Mentor Graphics ModelSim, Cadence IES, and Synopsys (the versions of these tools are available in the Xilinx Design Tools: Release Notes Guide.

Demonstration Test Bench

In Figure 6-2, the demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself. This test bench consists of transactor procedures or tasks that connect to the major ports of the example design, and a control program that pushes frames of varying length and content through the design and checks the values as they exit the core. The test bench is supplied as part of the Example Simulation output product group.

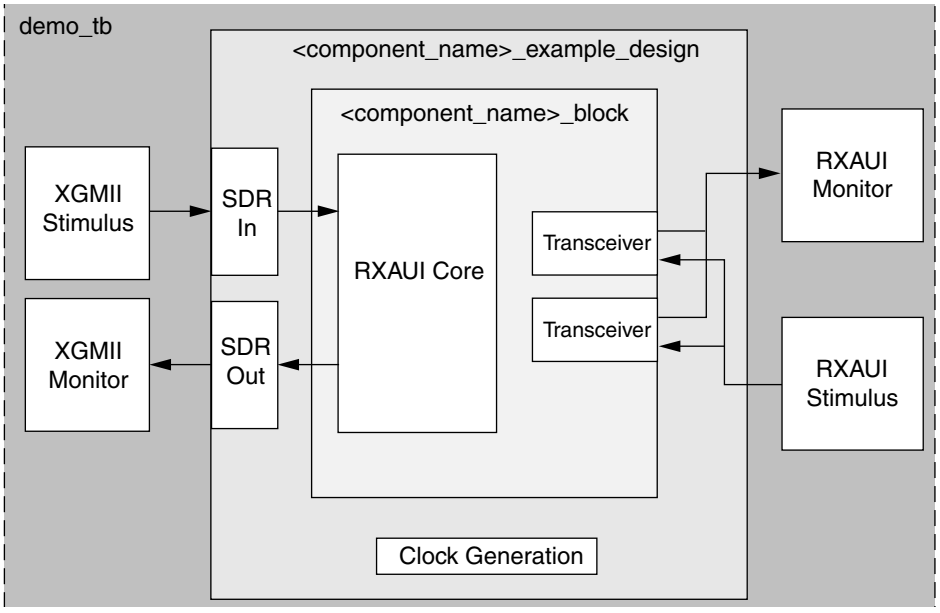

Figure 6-2: **Demonstration Test Bench for RXAUI**

Implementation

To implement the example design, select **Run Implementation** in the Vivado Project Manager window. For further details on setting up the implementation, see the *Vivado Design Suite User Guide, Implementation* [Ref 6].

Simulation

To simulate the example design, select Run Simulation in the Vivado Project Manager window. For further details on setting up the implementation, see the *Vivado Design Suite User Guide, Designing with IP* [Ref 7].

SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core in the ISE® Design Suite environment.

GUI

Figure 7-1 displays the main screen for customizing the RXAUI core.

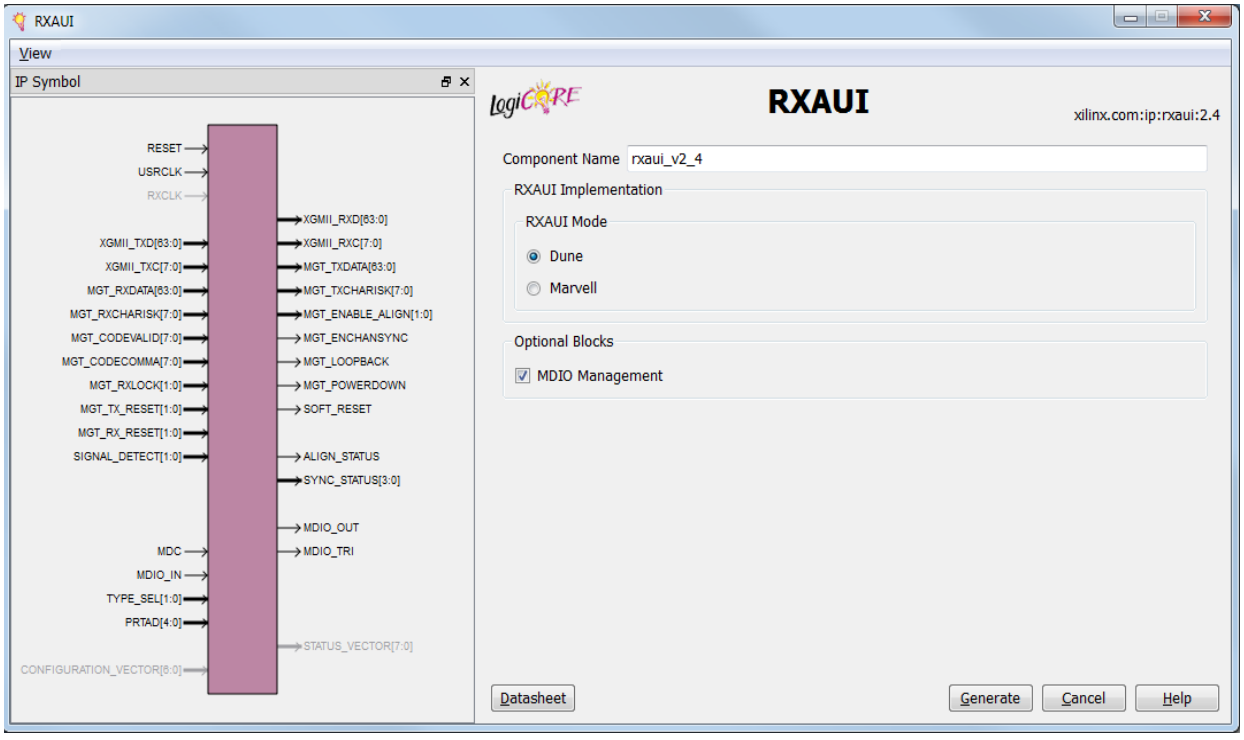

Figure 7-1: **RXAUI Main Screen**

For general help with starting and using the CORE Generator™ GUI, see the documentation supplied with the ISE® Design Suite.

Component Name

The component name is used as the base name of the output files generated for the core. Names must begin with a letter and must be composed from the following characters: a through z, 0 through 9 and "_" (underscore).

RXAUI Mode

This option selects between the three different RXAUI implementations that are supported. In 'Dune' mode, deinterleaving is performed on the |A||A| double character that is received.

In 'Marvell' mode multiplexing of the lanes is done to preserve 8B/10B disparity on the XAUI logical lanes. This means that the deinterleaving must be performed prior to any other processing. 8B/10B, deskew, and clock correction are all performed within the core.

MDIO Management

Select this option to implement the MDIO interface for managing the core. Deselect the option to remove the MDIO interface and expose a simple bit vector to manage the core.

The default is to implement the MDIO interface.

Parameter Values in the XCO File

XCO files contain parameterization information for an instance of a core; an XCO file is created when a core is generated and can be used to recreate a core. The text in an XCO file is case-insensitive.

Table 7-1 shows the XCO file parameters and values, and summarizes the GUI defaults. This is an example extract from an XCO file:

```
SELECT RXAUI family Xilinx,_Inc. 2.4
CSET component_name = the_core
CSET rxaui_mode = Dune
CSET mdio_management = true
GENERATE
```

Table 7-1: **XCO File Values and Defaults**

Parameter	XCO File Values	Defaults
component_name	ASCII text starting with a letter and based upon the following character set: a...z, 0...9 and _	rxaui_v2_4
mdio_management	TRUE, FALSE	TRUE
rxaui_mode	Dune and Marvell	Dune

Output Generation

The output files generated from CORE Generator are placed in the project directory. The list of output files includes:

- The netlist files for the core
- XCO files
- Release notes and documentation
- An HDL example design
- Scripts to synthesize, implement and simulate the example design.

See the Chapter 6, Detailed Example Design for a complete description of the CORE Generator output files and for details of the HDL example design.

Pre-implementation Simulation

A unit delay gate-level model of the RXAUI core netlist is provided as a CORE Generator™ output file. This can be used for simulation of the block in the design phase of a project.

Using the Simulation Model

For information about setting up your simulator to use the gate-level model, consult the *Xilinx ISE Synthesis and Simulation Design Guide* [Ref 4], included in your Xilinx software installation.

The unit delay gate-level model of the RXAUI core can be found in the CORE Generator project directory. Details of the CORE Generator outputs can be found in the Chapter 10, Detailed Example Design.

VHDL

```
component_name.vhd
```

Verilog

```
component_name.v
```

Constraining the Core

This chapter is applicable to the ISE® Design Suite environment.

This chapter describes how to constrain a design containing the RXAUI core. This is illustrated by the UCF delivered with the core at generation time. See Chapter 9, Detailed Example Design, for a complete description of the CORE Generator output files.

> ***Caution!*** Not all constraints are relevant to specific implementations of the core; consult the UCF created with the core instance to see exactly what constraints are relevant.

Device, Package, and Speed Grade Selections

This line selects the part to be used in the implementation run. Change this line so that it matches the part intended for the final application.

```
# Select the part to be used in the implementation run
CONFIG PART = xc6vlx240t-ff1156-2;
```

The RXAUI core can be implemented in all Virtex®-6 LXT/SXT/HXT devices with a speed grade of -2 or higher and all Kintex™-7 or Virtex-7 devices.

Clock Frequencies

The main clock frequencies for the design are specified as follows:

Virtex-7, Kintex-7 and Virtex-6 FPGAs

```
##########################################################################
# Clock frequencies and clock management #
##########################################################################
NET "*txoutclk*" TNM_NET="clk156_top";
TIMESPEC "TS_clk156_top" = PERIOD "clk156_top" 156.25MHz;
```

Recovered Clock (Marvell Mode)

```
NET "rxaui_block/rxclk" TNM_NET="rxclk312"; TIMESPEC "TS_rxclk312" = PERIOD
"rxclk312" 312.5 MHz;
```

General Clocking

```
NET "dclk" TNM_NET=DCLK_CLK;
TIMESPEC TS_DCLK_CLK = PERIOD DCLK_CLK 50 MHz;
```

This constraint defines the frequency of DCLK that is supplied to the Virtex-7, Kintex-7 and Virtex-6 FPGAs transceivers. The example design uses a nominal 50 MHz clock.

Clock Management

The Dune Networks RXAUI core has one clock domain:

- The `refclk` domain derived from the TXOUTCLK output of the transceiver

The Marvell core has two clock domains:

- The refclk domain derived from the TXOUTCLK output of the GTX/GTH transceiver
- The rxclk domain derived from the RXRECCLK output of the Virtex-6 FPGA GTX transceiver or the RXOUTCLK output of the 7 series transceivers

Transceiver Placement

Virtex-7 GTH Transceivers

```
INST rxaui_block/gt_wrapper_i/gt0_<ComponentName>_gt_wrapper_i/
gthe2_i LOC=GTHE2_CHANNEL_X0Y0
INST rxaui_block/gt_wrapper_i/gt1_<ComponentName>_gt_wrapper_i/
gthe2_i LOC=GTHE2_CHANNEL_X0Y1
```

7 Series GTX Transceivers

```
INST rxaui_block/gt_wrapper_i/gt0_<ComponentName>_gt_wrapper_i/
gtxe2_i LOC=GTXE2_CHANNEL_X0Y0
INST rxaui_block/gt_wrapper_i/gt1_<ComponentName>_gt_wrapper_i/
gtxe2_i LOC=GTXE2_CHANNEL_X0Y1
```

These constraints lock down the placement of the transceivers.

7 Series GTP Transceivers

```
INST rxaui_block/gt_wrapper_i/gt0_<ComponentName>_gt_wrapper_i/
gtpe2_i LOC=GTPE2_CHANNEL_X0Y0
INST rxaui_block/gt_wrapper_i/gt1_<ComponentName>_gt_wrapper_i/
gtpe2_i LOC=GTPE2_CHANNEL_X1Y0
```

Virtex-6 LXT/SXT/HXT FPGAs

```
INST rxaui_block/gtx_wrapper_i/gtx0_<ComponentName>_gtx_wrapper_i/
gtxe1_i LOC=GTXE1_X0Y0
INST rxaui_block/gtx_wrapper_i/gtx1_<ComponentName>_gtx_wrapper_i/
gtxe1_i LOC=GTXE1_X0Y1
```

MDIO

```
##################################################################
# MDIO-related constraints #
##################################################################

NET "*rxaui_core/BU2/U0/xaui_i/*management_1/mdc_rising*" TNM_NET =
"rxaui_mdc_grp";
INST "*rxaui_core/BU2/U0/xaui_i/type_sel_reg_1" TNM = FFS "rxaui_mdc_grp";
TIMESPEC "TS_RXAUI_mdc" = FROM "rxaui_mdc_grp" to "rxaui_mdc_grp" 400 ns;
```

These constraints set the correct attributes for the registers at the edge of the MDIO block. The TIMESPEC constrains the MDIO interface to 2.5 MHz. If you wish to overclock the MDIO interface, you must alter this constraint.

Detailed Example Design

This chapter provides detailed information about the example design, including a description of the files and the directory structure generated by the Xilinx® CORE Generator™ tool, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

Directory and File Contents

<project directory>
Top-level project directory; name is user-defined.

<project directory>/<component name>
Core release notes file

<component name>/example_design
Verilog and VHDL design files

<component name>/implement
Implementation script files

implement/results
Results directory, created after implementation scripts are run, and contains implement script results

<component name>/simulation
Simulation scripts

simulation/functional
Functional simulation files

<project directory>

The project directory contains all the CORE Generator project files.

Table 9-1: **Project Directory**

Name	Description
<project_dir>	
<component_name>.ngc	A binary Xilinx implementation netlist. Describes how the core is to be implemented. Used as an input to the Xilinx implementation tools.
<component_name>.v[hd]	VHDL or Verilog structural simulation model. File used to support functional simulation of a core.
<component_name>.xco	As an output file, the XCO file is a log file which records the settings used to generate a particular core. An XCO file is generated by CORE Generator for each core that it creates in the current project directory. An XCO file can also be used as an input to CORE Generator.
<component_name>_flist.txt	List of files delivered with the core
<component_name>.{veo|vho}	A VHDL or Verilog template for the core. This can be copied into your design.

Back to Top

<project directory>/<component name>

The <component name> directory contains the release notes file provided with the core, which can include last-minute changes and updates.

Table 9-2: **Component Name Directory**

Name	Description
<project_dir>/<component_name>	
rxaui_readme.txt	Core release notes file

Back to Top

<component name>/example_design

The example design directory contains the example design files provided with the core.

Table 9-3: **Example Design Directory**

Name	Description
<project_dir>/<component_name>/example_design	
<component_name>_block.v[hd]	Block entity containing the RXAUI core and transceiver wrappers
<component_name>_example_design.v[hd]	Top-level entity for the example design containing the block level design and clocking circuitry
component_name>_example_design.ucf	User constraints file for the core and example design
<component_name>_mod.v	Wrapper file for the RXAUI core
<component_name>_chanbond_monitor.v[hd]	Transceiver Channel Bonding Monitor
<component_name>_gt[x]_wrapper.v[hd] <component_name>_gt[x]_wrapper_gt[x].v[hd]	Wrappers for the transceivers

Back to Top

<component name>/implement

This directory contains the support files necessary for implementation of the example design with the Xilinx tools. Execution of an implement script creates a results directory and an `xst` project directory.

Table 9-4: **Implement Directory**

Name	Description
<project_dir>/<component_name>/implement	
implement.bat	Windows batch file that process the example design through the Xilinx tool flow
implement.sh	Linux shell script that processes the example design through the Xilinx tool flow
xst.scr	XST script file for the example design
xst.prj	XST project file for the example design

Back to Top

implement/results

This directory is created by the implement scripts and is used to run the example design files and the `<component_name>.ngc` file through the Xilinx implementation tools. On completion of an implement script, this directory contains the following files for timing simulation. Output files from the Xilinx implementation tools can also be found in this directory.

Table 9-5: **Results Directory**

Name	Description
<project_dir>/<component_name>/implement/results	
routed.v[hd]	The back-annotated SimPrim-based VHDL or Verilog design. Used for timing simulation.
routed.sdf	Timing information for simulation

Back to Top

<component name>/simulation

The simulation directory and the subdirectories below it contain the files necessary to test a VHDL or Verilog implementation of the example design.

Table 9-6: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
demo_tb.v[hd]	The VHDL or Verilog demonstration test bench for the RXAUI core

Back to Top

simulation/functional

The functional directory contains functional simulation scripts provided with the core.

Table 9-7: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
`simulate_mti.do`	ModelSim macro file that compiles the example design sources, the structural simulation model and the demonstration test bench then runs the functional simulation to completion.
`simulate_ncsim.sh`	Linux shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using the Cadence IES simulator.
`simulate_vcs.sh (verilog only)`	Linux shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using VCS.

Table 9-7: **Functional Directory *(Cont'd)***

Name	Description
`ucli_commands.key (verilog only)`	VCS command file. This file is called by the simulate_vcs.sh script.
`vcs_session.tcl (verilog only)`	VCS DVE tcl script that opens wave windows and adds interesting signals to it. This macro is used by the simulate_vcs.sh script.
`wave_mti.do`	ModelSim macro file that opens a wave window and adds interesting signals to it. This macro is called by the `simulate_mti.do` macro file.
`wave_ncsim.sv`	The Cadence IES simulator macro file that opens a wave windows and adds interesting signals to it. This macro is called by the simulate_ncsim.sh script.

Back to Top

Example Design

Figure 9-1 illustrates the default configuration of the example design.

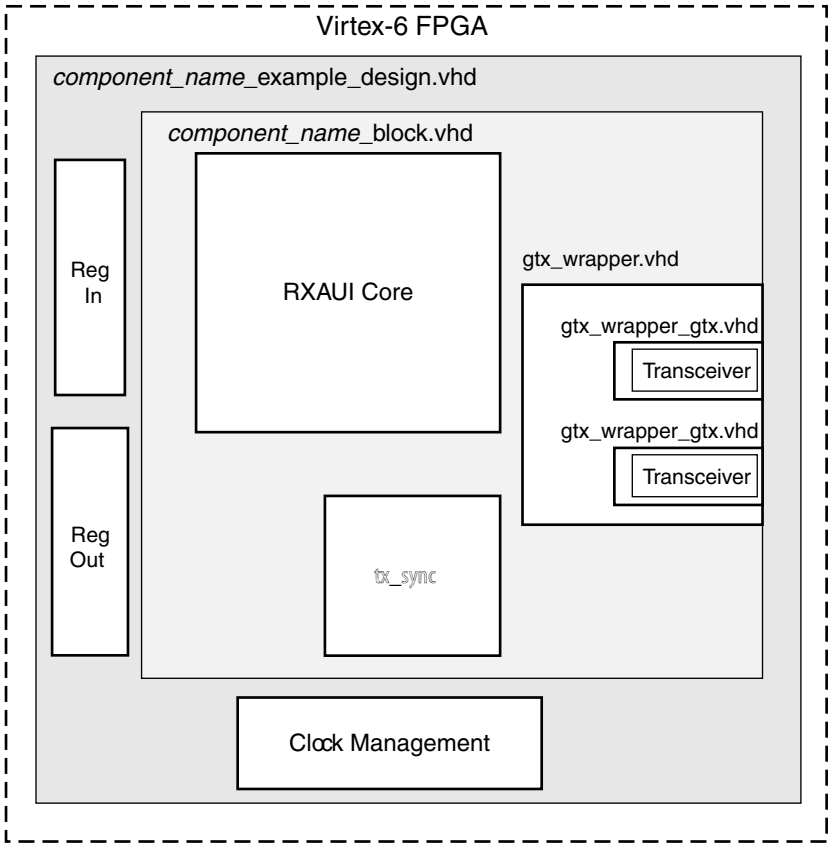

Figure 9-1: **RXAUI Example Design: Default Configuration**

The RXAUI example design consists of the following:

- A RXAUI core netlist
- Transceiver wrappers
- A transceiver transmit initialization block
- Clock management logic and clock buffer instances
- Re-timing registers on the parallel data interface, both on inputs and outputs

The RXAUI Design Example has been tested with Xilinx® ISE® Design Suite, Mentor Graphics ModelSim, Cadence IES, and Synopsys (the versions of these tools are available in the Xilinx Design Tools: Release Notes Guide).

Demonstration Test Bench

In Figure 9-2, the demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself. This test bench consists of transactor procedures or tasks that connect to the major ports of the example design, and a control program that pushes frames of varying length and content through the design and checks the values as they exit the core.

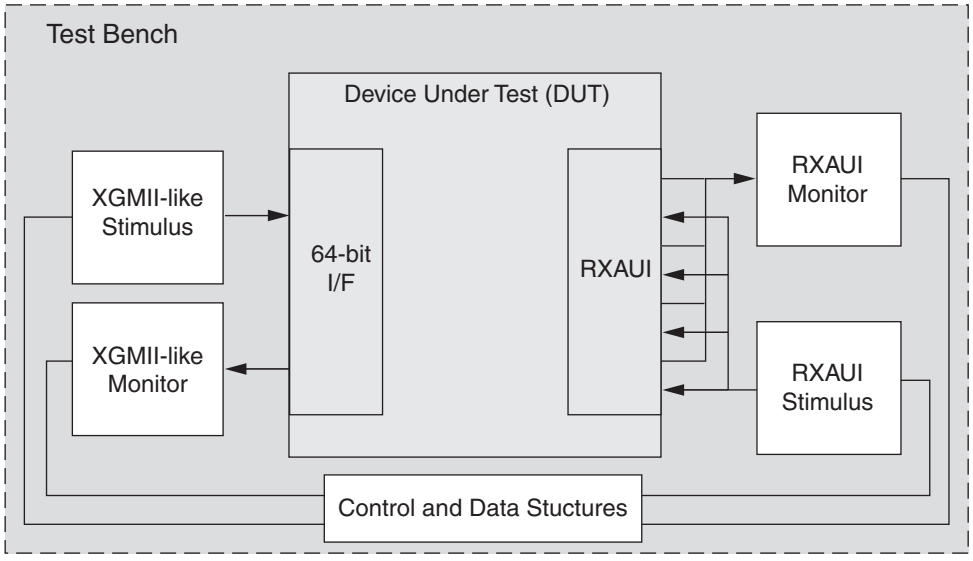

Figure 9-2: **Demonstration Test Bench for RXAUI**

Generating the Core

To generate a RXAUI core with default values using the CORE Generator™ tool perform the following steps:

1. Start CORE Generator.

   For help starting and using CORE Generator, see the documentation supplied with the ISE Design Suite.

2. Choose File > New Project.
3. Type a directory name.
4. Perform these steps to set project options:
   a. From the Part tab, select a silicon family, part, speed grade, and package that supports the RXAUI core, for example, Virtex®-6 FPGAs.

IMPORTANT: *If an unsupported silicon family is selected, the RXAUI core does not appear in the taxonomy tree.*

   b. From the Generation tab, select VHDL or Verilog; for Vendor, select Other.
   c. On the Advanced tab, accept the default values.
5. After creating the project, locate the core in the taxonomy tree at the left side of the CORE Generator window. The RXAUI core appears under these categories:
   - Communications & Networking/Ethernet
   - Communications & Networking/Networking
   - Communications & Networking/Telecommunications
6. Double-click the core to open it.
7. In the Component Name field, enter a name for the core instance.
8. Accept the remaining default options and click Finish to generate the core.

   The core and its supporting files, including the example design, are generated in the project directory. For a detailed description of the directory structure and files, see Directory and File Contents.

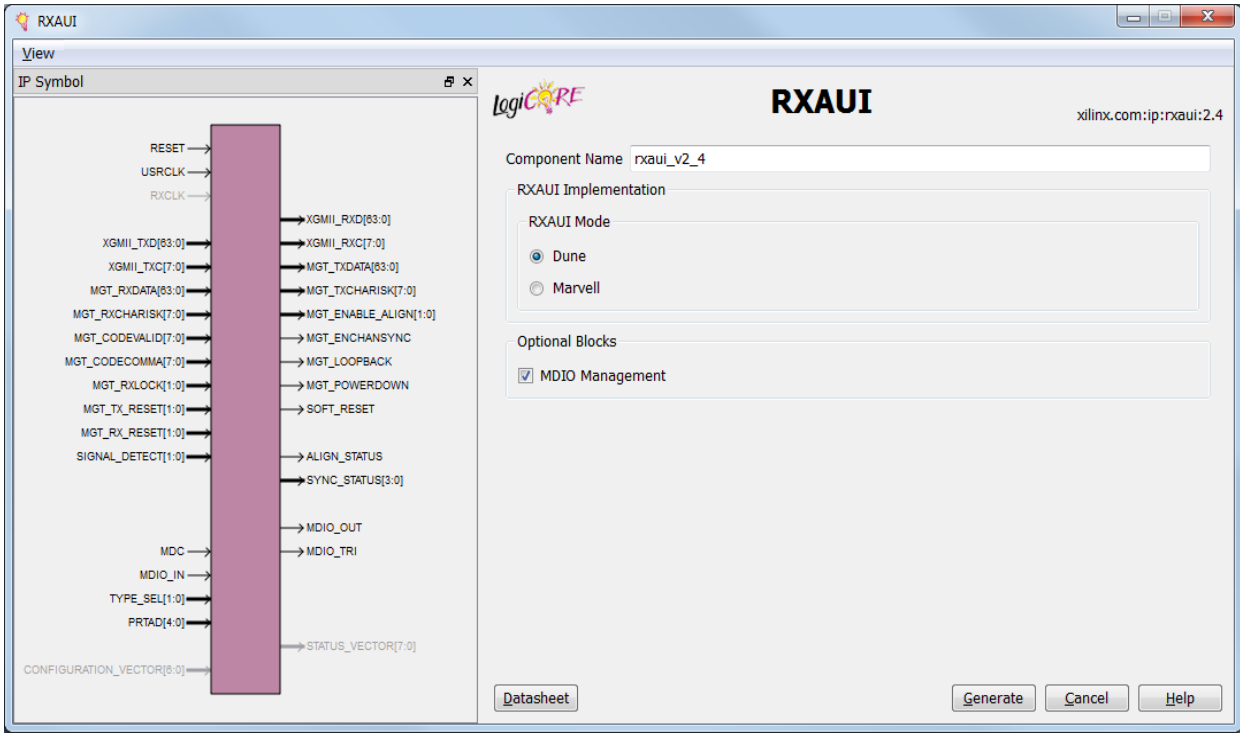

Figure 9-3: **RXAUI Main Screen**

Implementation

After the core is successfully generated, the netlist and example design HDL wrapper can be processed through the Xilinx implementation tools. The generated outputs include several scripts to assist in processing.

Implementation Script

The implementation script is either a shell script or batch file that processes the example design through the Xilinx tool flow. The script is located at:

Linux

```
<project_dir>/<component_name>/implement/implement.sh
```

Windows

```
<project_dir>/<component_name>/implement/implement.bat
```

The implement script performs these actions:

- The example HDL wrapper is synthesized using XST.
- ngdbuild is run to consolidate the core netlist and the wrapper netlist into the NGD file containing the entire design.
- The design is mapped to the target technology.
- The design is place-and-routed on the target device.
- Static timing analysis is performed on the routed design using trce.
- netgen runs on the routed design to generate VHDL and Verilog netlists and timing information in the form of SDF files.

Simulation

The example design provided with the RXAUI core provides a complete environment which allows you to simulate the core and view the outputs. Scripts are provided for pre- and pre-layout simulation. The simulation model is either in VHDL or Verilog depending on the CORE Generator Design Entry project option.

Setting up for Simulation

To run the gate-level simulation you must have the Xilinx Simulation Libraries compiled for your system. See the Compiling Xilinx Simulation Libraries (COMPXLIB) in the *Xilinx ISE Synthesis and Simulation Design Guide* [Ref 13], and the Xilinx ISE® Design Suite Software *Manuals and Help* [Ref 14].

The Xilinx simulation libraries must be mapped into the simulator. If the libraries are not set for your environment, go to Answer Record 15338 on www.xilinx.com/support for assistance compiling Xilinx simulation models and setting up the simulator environment.

All Virtex device designs require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. For a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator, these simulators are supported (the versions of these tools are available in the Xilinx Design Tools: Release Notes Guide):

- Mentor Graphics ModelSim
- Cadence IES
- Synopsys

Simulation Scripts

Simulation macro files are provided for ModelSim and shell scripts are provided for the Cadence IES simulator and Synopsys VCS simulator. The scripts automate the simulation of the test bench and can be found in the following location:

```
<project_dir>/<component_name>/simulation/functional/simulate_mti.do
<project_dir>/<component_name>/simulation/functional/simulate_ncsim.sh
<project_dir>/<component_name>/simulation/functional/simulate_vcs.sh
```

The scripts perform these tasks:

- Compiles the gate level netlist
- Compiles the demonstration test bench
- Starts a simulation of the test bench (with timing information if a Full-system Evaluation license or Full license is in use)
- Opens a Wave window and adds some interesting signals (`wave_mti.do`/ `wave_ncsim.sv`/`vcs_session.tcl`)
- Runs the simulation to completion

SECTION IV: APPENDICES

Appendix A

Verification, Compliance, and Interoperability

The RXAUI core has been verified using both simulation and hardware testing.

Simulation

A highly parameterizable transaction-based simulation test suite has been used to verify the core. Tests included:

- Register access over MDIO
- Loss and re-gain of synchronization
- Loss and re-gain of alignment
- Frame transmission
- Frame reception
- Clock compensation
- Recovery from error conditions

Hardware Testing

The core has been used in several hardware test platforms within Xilinx. In particular, the core has been used in a test platform design with the Xilinx® 10-Gigabit Ethernet MAC core. This design comprises the MAC, RXAUI, a 'ping' loopback FIFO, and a test pattern generator all under embedded MicroBlaze™ processor control. This design has been used for interoperability testing at Dune Networks.

Appendix B

Migrating

For information on migrating to the Vivado™ Design Suite, see the *Vivado Design Suite Migration Methodology Guide* [Ref 8].

Appendix C

Debugging

This chapter provides information on using resources available on the Xilinx® Support website, available debug tools, and a step-by-step process for debugging designs that use the RXAUI core. This information is found in this chapter:

- Finding Help on xilinx.com
- Contacting Xilinx Technical Support
- Debug Tools
- Simulation Specific Debug
- Hardware Debug

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

The Solution Center specific to the RXAUI core is located at:

- Xilinx Ethernet IP Solution Center

Finding Help on xilinx.com

To help in the design and debug process when using the RXAUI core, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as Product documentation, Release Notes, Answer Records, and links to opening a Technical Support case.

Documentation

This document along with documentation related to all products that aid in the design process can be found on the Xilinx Support web page. Documentation is sorted by product family at the main support page or by solution at the Documentation Center.

Release Notes and Known Issues

Known issues for all cores, including the RXAUI core, are described in the *IP Release Notes Guide* [Ref 16].

Answer Records

Answer Records include information on commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a product. Answer Records are created and maintained daily ensuring that users have access to the most up-to-date information on Xilinx products. Answer Records can be found by searching the Answers Database on www.xilinx.com/support.

Contacting Xilinx Technical Support

Xilinx provides premier technical support for customers encountering issues that requires additional assistance.

To contact Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under **Additional Resources**.

When opening a WebCase, include:

- Target FPGA including package and speed grade
- All applicable versions of ISE® Design Suite, synthesis (if not XST), and simulator
- The xco file created during generation of the LogiCORE™ IP wrapper. This file is located in the directory targeted for the CORE Generator™ project.

Additional files might be required based on the specific issue. See the relevant sections in this debug guide for further information on specific files to include with the WebCase.

Debug Tools

There are many tools available to debug RXAUI design issues. It is important to know which tools are useful for debugging various situations that you encounter. This chapter references these tools:

- Example Design
- ChipScope Pro Tool
- Link Analyzers
- Link Analyzers

Example Design

The RXAUI core comes with a synthesizable example design complete with functional and post-place and route simulation test benches. Information on the example design can be found in the Chapter 6, Detailed Example Design.

ChipScope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O software cores directly into your design. The ChipScope Pro tool allows you to set trigger conditions to capture application and Integrated Block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information on the ChipScope Pro tool, visit www.xilinx.com/chipscope.

Link Analyzers

Link Analyzers can be used to generate and analyzer traffic for hardware debug and testing. Common link analyzers include:

- SMARTBITS
- IXIA

Simulation Specific Debug

This section provides simulation debug flow diagrams for some of the most common issues experienced by users. Endpoints that are shaded gray indicate that more information can be found in sections after the figure.

ModelSim Debug

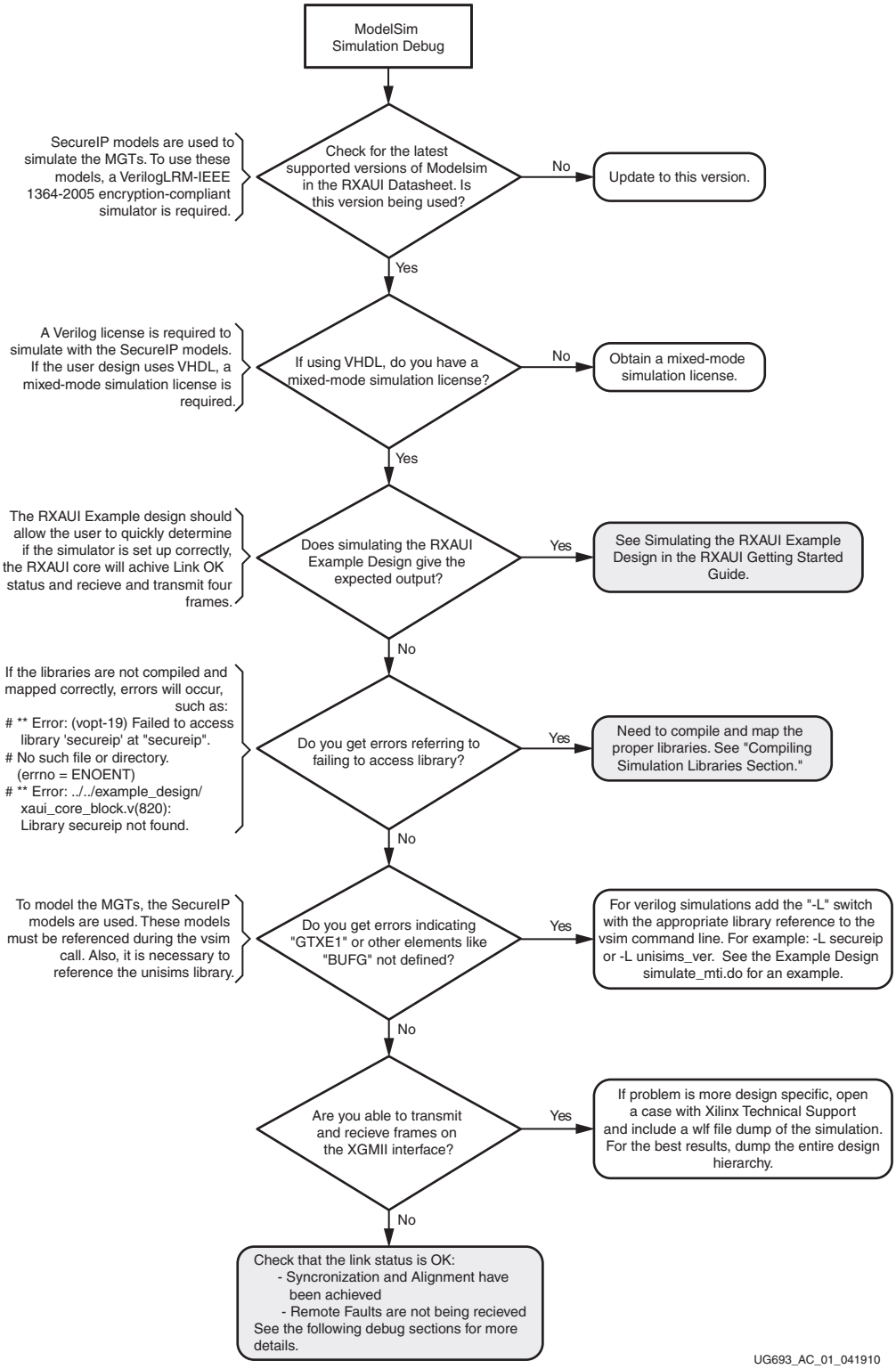

Figure C-1: **ModelSim Debug Flow Diagram**

Compiling Simulation Libraries

Compile the Xilinx simulation libraries, either by using the Xilinx Simulation Library Compilation Wizard, or by using the compxlib command line tool.

Xilinx Simulation Library Compilation Wizard

A GUI wizard provided as part of the Xilinx software can be launched to assist in compiling the simulation libraries by typing "compxlib" in the command prompt.

Compxlib

A compxlib command line can also be used to compile simulation libraries. This tool is delivered as part of the Xilinx software. For details see the section compxlib in the *Command Line Tools User Guide* [Ref 17].

Assuming the Xilinx and ModelSim environments are set up correctly, this is an example of compiling the SecureIP and UNISIM libraries for Verilog into the current directory

```
compxlib -s mti_se -arch virtex6 -l verilog -lib secureip -lib unisims -dir ./
```

There are many other options available for compxlib described in the *Command Line Tools User Guide* [Ref 17].

Compxlib produces a modelsim.ini file containing the library mappings. In ModelSim, to see the current library mappings, type "vmap" at the prompt. The mappings can be updated in the ini file or to map a library at the ModelSim prompt type:

```
vmap [<logical_name>] [<path>]
```

For example:

```
vmap unisims_ver C:\my_unisim_lib
```

Next Step

If the debug suggestions listed previously do not resolve the issue, open a support case to have the appropriate Xilinx expert assist with the issue. To create a technical support case in WebCase, see the Xilinx website at:

www.xilinx.com/support/clearexpress/websupport.htm

Items to include when opening a case:

- Detailed description of the issue and results of the steps listed previously.
- Attach a VCD or WLF dump of the simulation.

To discuss possible solutions, use the Xilinx User Community: forums.xilinx.com/xlnx/

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The ChipScope tool is a valuable resource to use in hardware debug and the signal names mentioned in the following individual sections can be probed using the ChipScope tool for debugging the specific problems. Many of these common issue can also be applied to debugging design simulations.

General Checks

Ensure that all the timing constraints for the core were met during Place and Route.

- Does it work in timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue.
- Ensure that all clock sources are clean. If using DCMs in the design, ensure that all DCMs have obtained lock by monitoring the LOCKED port.

Monitoring the RXAUI Core with ChipScope Tool

- XGMII signals and signals between RXAUI core and the transceiver can be added to monitor data transmitted and received. See Table 2-1, page 13 and Table 2-2, page 14 for a list of signal names.
- Status signals added to check status of link: STATUS_VECTOR[7:0], ALIGN_STATUS, and SYNC_STATUS[3:0].
- To interpret control codes in on the XGMII interface or the interface to the transceiver, see Table C-1 and Table C-2.
- An Idle (0x07) on the XGMII interface is encoded to be a randomized sequence of /K/ (Sync), /R/ (Skip), /A/(Align) codes on the RXAUI interface. For details on this encoding, see the IEEE 802.3-2008 specification (section 48.2.4.2) for more details.

Table C-1: **XGMII Control Codes**

TXC	TXD	Description
0	0x00 through 0xFF	Normal data transmission
1	0x07	Idle
1	0x9C	Sequence
1	0xFB	Start
1	0xFD	Terminate
1	0xFE	Error

Table C-2: **RXAUI Control Codes**

Codegroup	8-bit value	Description
Dxx.y	**0xXX**	Normal data transmission
K28.5	0xBC	/K/ (Sync)
K28.0	0x1C	/R/ (Skip)
K28.3	0x7C	/A/ (Align)
K28.4	0x9C	/Q/ (Sequence)
K27.7	0xFB	/S/ (Start)
K29.7	0xFD	/T/ (Terminate)
K30.7	0xFE	/E/ (Error)

Issues with Data Reception or Transmission

Issues with data reception or transmission can be caused by a wide range of factors. Following is a flow diagram of steps to debug the issue. Each of the steps are discussed in more detail in the following sections.

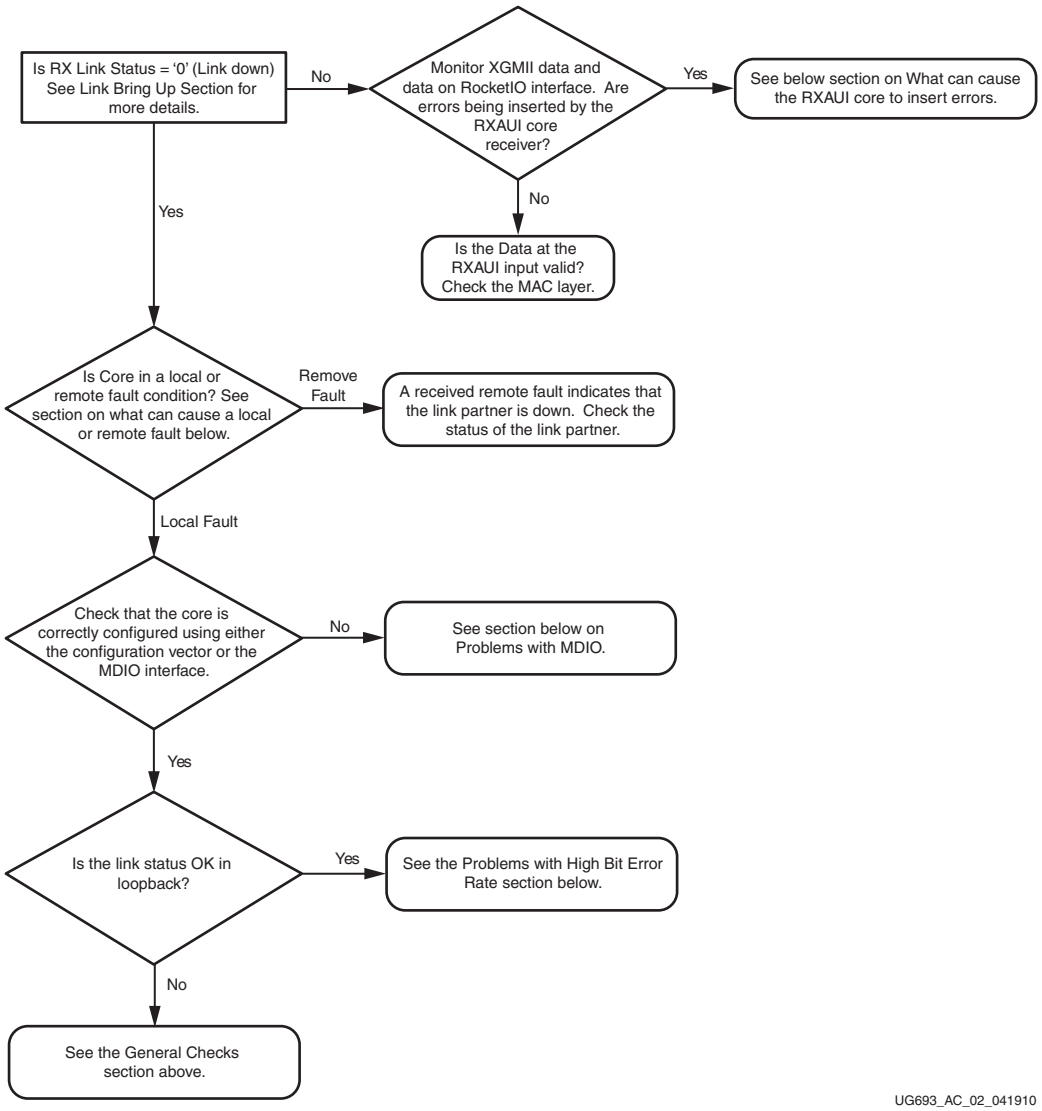

Figure C-2: **Flow Diagram for Debugging Problems with Data Reception or Transmission**

What Can Cause a Local or Remote Fault?

Local Fault and Remote Fault codes both start with the sequence TXD/RXD=0x9C, TXC/RXC=1 in XGMII lane 0. Fault conditions can also be detected by looking at the status vector or MDIO registers. The Local Fault and Link Status are defined as latching error indicators by the IEEE specification. This means that the Local Fault and Link Status bits in the status vector or MDIO registers must be cleared with the Reset Local Fault bits and Link Status bits in the Configuration vector or MDIO registers.

Local Fault

The receiver outputs a local fault when the receiver is not up and operational. This rx local fault is also indicated in the status and MDIO registers. The most likely causes for an rx local fault are:

- The transceiver has not locked or the receiver is being reset.
- At least one of the lanes is not synchronized - SYNC_STATUS[3:0]
- The lanes are not properly aligned - ALIGN_STATUS

Note: The SYNC_STATUS and ALIGN_STATUS signals are not latching.

A tx local fault is indicated in the status and MDIO registers when the transceiver transmitter is in reset or has not yet completed any other initialization or synchronization procedures needed.

Remote Fault

Remote faults are only generated in the MAC reconciliation layer in response to a Local Fault message. When the receiver receives a remote fault, this means that the link partner is in a local fault condition.

When the MAC reconciliation layer receives a remote fault, it silently drops any data being transmitted and instead transmit IDLEs to help the link partner resolve its local fault condition. When the MAC reconciliation layer receives a local fault, it silently drops any data being transmitted and instead transmit a remote fault to inform the link partner that it is in a fault condition. Be aware that the Xilinx 10GEMAC core has an option to disable remote fault transmission.

Link Bring Up

The following link initialization stages describe a possible scenario of the Link coming up between device A and device B.

Stage 1: Device A Powered Up, but Device B Powered Down

- Device A is powered up and reset.
- Device B powered down
- Device A detects a fault because there is no signal received. The Device A RXAUI core indicates an rx local fault.
- The Device A MAC reconciliation layer receives the local fault. This triggers the MAC reconciliation layer to silently drop any data being transmitted and instead transmit a remote fault.
- RX Link Status = '0' (link down) in Device A

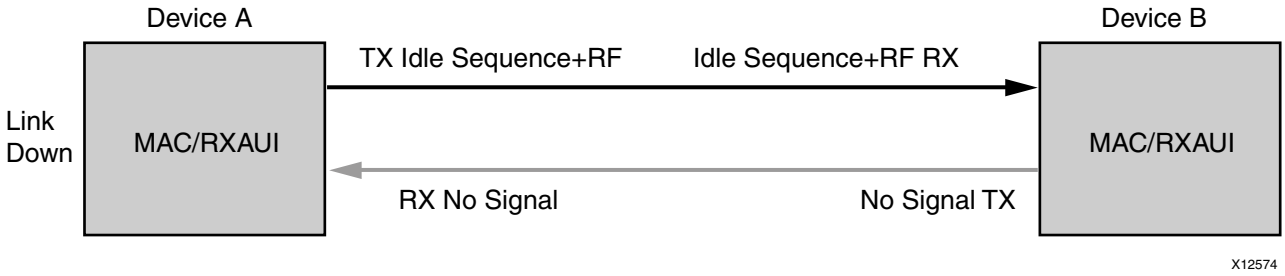

Figure C-3: **Device A Powered Up, but Device B Powered Down**

Stage 2: Device B Powers Up and Resets

- Device B Powers Up and Resets.
- Device B RXAUI completes Synchronization and Alignment.
- Device A has not synchronized and aligned yet. It continues to send remote faults.
- Device B RXAUI passes received remote fault to MAC.
- Device B MAC reconciliation layer receives the remote fault. It silently drops any data being transmitted and instead transmits IDLEs.
- Link Status = '0' (link down) in both A and B.

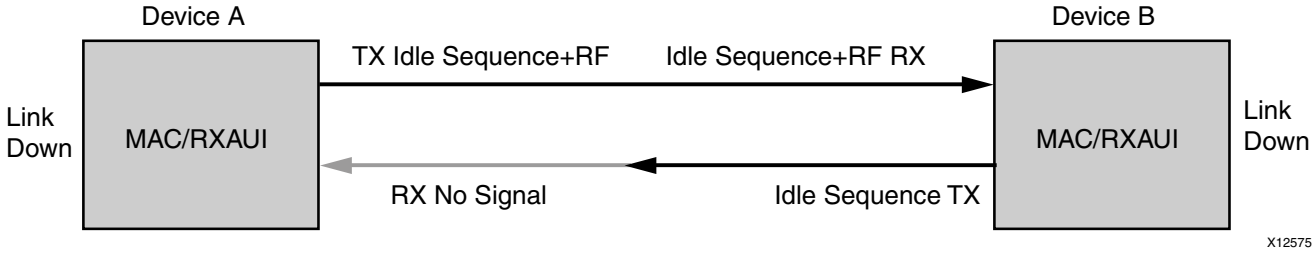

Figure C-4: **Device B Powers Up and Resets**

Stage 3: Device A Receives Idle Sequence

- Device A RXAUI RX detects idles, synchronizes and aligns.
- Device A reconciliation layer stops dropping frames at the output of the MAC transmitter and stops sending remote faults to Device B.
- Device A Link Status='1' (Link Up)
- After Device B stops receiving the remote faults, normal operation starts.

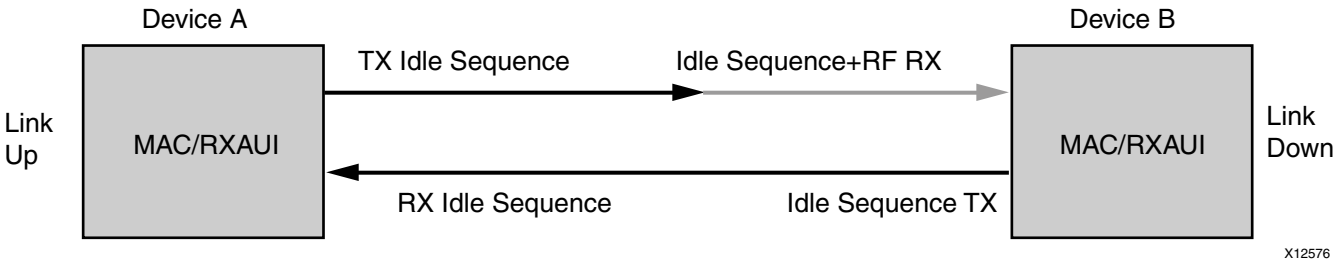

Figure C-5: **Device A Receives Idle Sequence**

Stage 4: Normal Operation

In Stage 4 shown in Figure C-6, Device A and Device B have both powered up and been reset. The link status is '1' (link up) in both A and B and in both the MAC can transmit frames successfully.

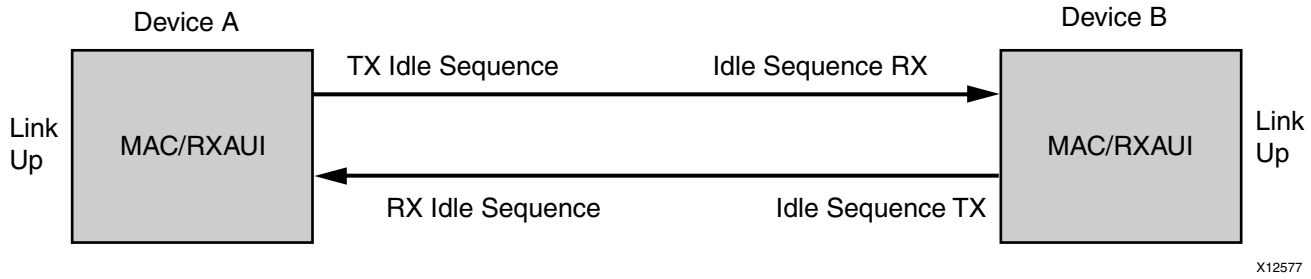

Figure C-6: **Normal Operation**

What Can Cause Synchronization and Alignment to Fail?

Synchronization (sync_status[3:0]) occurs when each respective XAUI logical lane receiver is synchronized to byte boundaries. Alignment (align_status) occurs when the RXAUI receiver is aligned across all four logical XAUI lanes.

Following are suggestions for debugging loss of Synchronization and Alignment:

- Monitor the state of the SIGNAL_DETECT[1:0] input to the core. This should either be:
  - connected to an optical module to detect the presence of light. Logic '1' indicates that the optical module is correctly detecting light; logic '0' indicates a fault. Therefore, ensure that this is driven with the correct polarity.
  - tied to logic '1' (if not connected to an optical module).

Note: When `signal_detect` is set to logic '0,' this forces the receiver synchronization state machine of the core to remain in the loss of sync state.

- Loss of Synchronization can happen when invalid characters are received.

- Loss of Alignment can happen when invalid characters are seen or if an /A/ code is not seen in all 4 XAUI logical lanes at the same time.
- See the section, Problems with a High Bit Error Rate.

Transceiver Specific

- Ensure that the polarities of the TXN/TXP and RXN/RXP lines are not reversed. If they are, these can be fixed by using the TXPOLARITY and RXPOLARITY ports of the transceiver.
- Check that the transceiver is not being held in reset or still be initialized by monitoring the `mgt_tx_reset`, `mgt_rx_reset`, and `mgt_rxlock` input signals to the RXAUI core. The `mgt_rx_reset` signal is also asserted when there is an rx buffer error. An rx buffer error means that the Elastic Buffer in the receiver path of the transceiver is either under or overflowing. This indicates a clock correction issue caused by differences between the transmitting and receiving ends. Check all clock management circuitry and clock frequencies applied to the core and to the transceiver.

What Can Cause the RXAUI Core to Insert Errors?

On the receive path the RXAUI core inserts errors RXD=FE, RXC=1, when disparity errors or invalid data are received or if the received interframe gap (IFG) is too small.

Disparity Errors or Invalid Data

Disparity Errors or Invalid data can be checked for by monitoring the `mgt_codevalid` input to the RXAUI core.

Small IFG

The RXAUI core inserts error codes into the Received XGMII data stream, RXD, when there are three or fewer IDLE characters (0x07) between frames. The error code (0xFE) precedes the frame's "Terminate" delimiter (0xFD).

The IEEE 802.3-2008 specification (Section 46.2.1) requires a minimum interframe gap of five octets on the receive side. This includes the preceding frame's Terminate control character and all Idles up to and immediately preceding the following frame's Start control character. Because three (or fewer) Idles and one Terminate character are less than the required five octets, this would not meet the specification; therefore, the RXAUI core is expected to signal an error in this manner if the received frame does not meet the specification.

Problems with a High Bit Error Rate

Symptoms

If the link comes up but then goes down again or never comes up following a reset, the most likely cause for a Rx Local Fault is a BER (Bit Error Rate) that is too high. A high BER causes incorrect data to be received, which leads to the lanes losing synchronization or alignment.

Debugging

Compare the issue across several devices or PCBs to ensure that the issue is not a one-off case.

- Try using an alternative link partner or test equipment and then compare results.
- Try putting the core into loopback (both by placing the core into internal loopback, and by looping back the optical cable) and compare the behavior. The core should always be capable of gaining synchronization and alignment when looping back with itself from transmitter to receiver so direct comparisons can be made. If the core exhibits correct operation when placed into internal loopback, but not when loopback is performed using an optical cable, this can indicate a faulty optical module or a PCB issue.
- Try swapping the optical module on a misperforming device and repeat the tests.

Transceiver Specific Checks

- Monitor the `MGT_CODEVALID`[7:0] input to the RXAUI core by triggering on it using the ChipScope tool. This input is a combination of the transceiver rx disparity error and rx not in table error outputs.
- These signals should not be asserted over the duration of a few seconds, minutes or even hours. If they are frequently asserted, it can indicate an issue with the transceiver.
- Place the transceiver into parallel or serial near-end loopback.
    - If correct operation is seen in the transceiver serial loopback, but not when loopback is performed using an optical cable, it can indicate a faulty optical module.
    - If the core exhibits correct operation in the transceiver parallel loopback but not in serial loopback, this can indicate a transceiver issue.
- A mild form of bit error rate can be solved by adjusting the transmitter Pre-Emphasis and Differential Swing Control attributes of the transceiver.

Problems with the MDIO

See MDIO Interface for detailed information about performing MDIO transactions.

Things to check for:

- Ensure that the MDIO is driven properly. Check that the mdc clock is running and that the frequency is 2.5 MHz or less.
- Ensure that the RXAUI core is not held in reset.
- Read from a configuration register that does not have all 0s as a default. If all 0s are read back, the read was unsuccessful. Check that the PRTAD field placed into the MDIO frame matches the value placed on the PRTAD[4:0] port of the RXAUI core.
- Verify in simulation and/or a ChipScope capture that the waveform is correct for accessing the host interface for a MDIO read/write.

Next Steps

If the debug suggestions listed previously do not resolve the issue, open a support case to have the appropriate Xilinx expert assist with the issue.

To create a technical support case in Webcase, see the Xilinx website at:

www.xilinx.com/support/clearexpress/websupport.htm

Items to include when opening a case:

- Detailed description of the issue and results of the steps listed previously.
- Attach ChipScope tool VCD captures taken in the steps previously.

To discuss possible solutions, use the Xilinx User Community:

forums.xilinx.com/xlnx/

Appendix D

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

References

These documents provide supplemental material useful with this product guide:

1. IEEE Std. 802.3-2008, Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.
2. Dune Networks DN-DS-RXAUI-Spec v1.0, RXAUI - Reduced Pin XAUI.
3. Marvell MV-S 105386-00 RXAUI Interface and RXAUI Adapter Specifications.
4. RXAUI core release notes.
5. Vivado™ Design Suite user documentation
6. Vivado Design Suite User Guide, Implementation (UG904)
7. Vivado Design Suite User Guide, Designing with IP (UG896)
8. Vivado Design Suite Migration Methodology Guide (UG911)
9. Virtex-6 FPGA GTX Transceivers User Guide (UG366)
10. 7 Series FPGAs GTX/GTH Transceivers User Guide (UG476)
11. 7 Series FPGAs GTP Transceivers User Guide (UG482)
12. Virtex-6 FPGA Packaging and Pinout Specifications (UG365)

13. Xilinx ISE Synthesis and Simulation Design Guide (UG626)
14. Xilinx ISE Manuals and Help
15. XST User Guide
16. IP Release Notes Guide (XTP025)
17. Command Line Tools User Guide (UG628)

Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the IP Release Notes Guide (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/16/12	1.0	Initial Xilinx release. This Product Guide is derived from DS740 and UG693. Vivado Design Suite and Artix-7 support added.

Notice of Disclaimer