# LogiCORE IP RXAUI v3.0

## *Product Guide for Vivado Design Suite*

**PG083 March 20, 2013**

XILINX®

# Table of Contents


## IP Facts

## Chapter 1: Overview

Feature Summary . . . . . 5
Applications . . . . . 6
Functional Description . . . . . 6
Licensing and Ordering Information . . . . . 7

## Chapter 2: Product Specification

Standards . . . . . 8
Performance . . . . . 8
Resource Utilization . . . . . 9
Port Descriptions . . . . . 9
Register Space . . . . . 16

## Chapter 3: Designing with the Core

General Design Guidelines . . . . . 48
Clocking . . . . . 49
Resets . . . . . 52
Design Considerations . . . . . 52
Protocol Description . . . . . 52

## Chapter 4: Customizing and Generating the Core

Vivado Integrated Design Environment (IDE) . . . . . 61
Output Generation . . . . . 62

## Chapter 5: Constraining the Core

Required Constraints . . . . . 63
Clock Frequencies . . . . . 63
Clock Management . . . . . 64
Transceiver Placement . . . . . 64

## Chapter 6: Detailed Example Design

Example Design . . . . . 65
Demonstration Test Bench . . . . . 66
Implementation . . . . . 67
Simulation . . . . . 67

## Appendix A: Verification, Compliance, and Interoperability

Simulation . . . . . 68
Hardware Testing . . . . . 68

## Appendix B: Migrating

## Appendix C: Debugging

Finding Help on Xilinx.com . . . . . 70
Debug Tools . . . . . 72
Simulation Debug . . . . . 73
Hardware Debug . . . . . 75

## Appendix D: Additional Resources

Xilinx Resources . . . . . 84
References . . . . . 84
Revision History . . . . . 85
Notice of Disclaimer . . . . . 86

# 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 FPGAs that comply with Dune Networks specifications.

The 7 series 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 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 encrypted HDL
- *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 |
| Supported User Interfaces | XGMII, MDIO |
| Resources | See Table 2-1 and Table 2-3. |
| **Provided with Core** | |
| Design Files | Encrypted RTL |
| Example Design | Verilog/VHDL |
| Test Bench | Verilog/VHDL |
| Constraints File | XDC |
| Simulation Model | VHDL/Verilog |
| Supported S/W Driver | N/A |
| **Tested Design Flows(2)** | |
| Design Entry | Vivado™ Design Suite |
| Simulation | Mentor Graphics Questa® SIM<br>Vivado Simulator |
| Synthesis | Vivado Synthesis |
| **Support** | |
| Provided by Xilinx @ www.xilinx.com/support | |

**Notes:**

1. For a complete listing of supported devices, see the Vivado IP catalog.
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 four 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 and Kintex™-7 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 the following mode for 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.

For the 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.

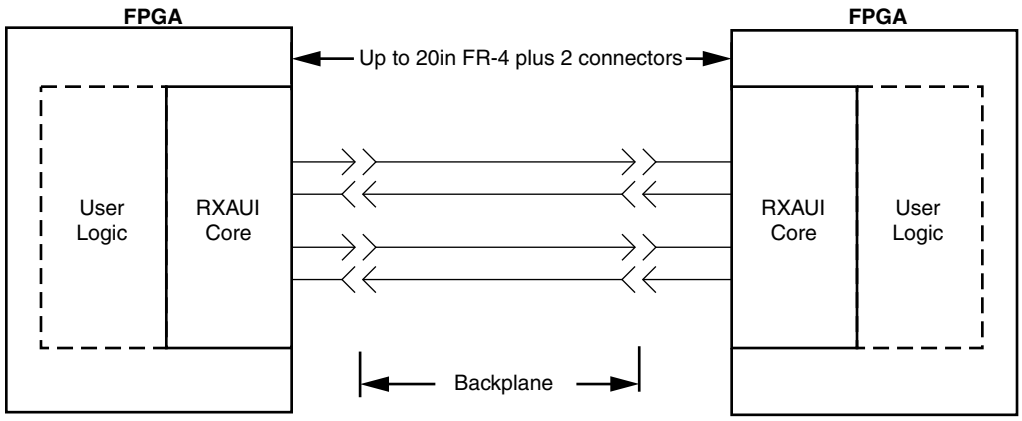


*Figure 1-1:* **Typical Backplane Application for RXAUI**

# 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** – Deskews 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.

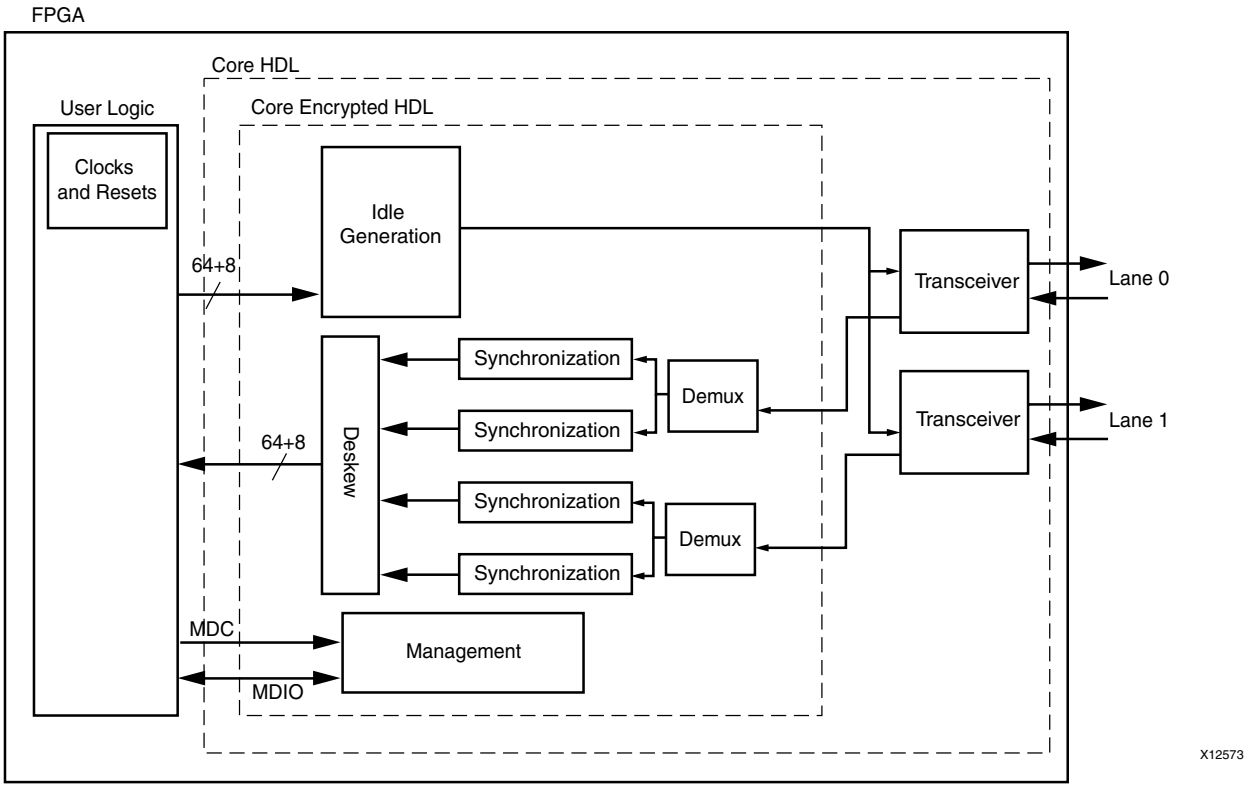


*Figure 1-2:* **Implementation of Dune Networks RXAUI Core**

# Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite 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] 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 internal transceiver interface), the latency through the core for the internal XGMII interface configuration in the transmit direction is 4 clock periods of the core input `usrclk` for Dune Networks mode.

#### Receive Path Latency

Measured from the transceiver output pins `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 6 to 8 clock cycles of `usrclk`. The latency depends on comma alignment position and data positioning within the transceiver 4-byte interface.

# Resource Utilization

Table 2-1 provides approximate utilization for the various core options on Virtex®-7 and Kintex™-7 FPGAs using the Vivado™ Design Suite.

*Table 2-1:* **Device Utilization – Virtex-7 (GTX), Kintex-7, and Zynq™-7000 Devices**

| MDIO Management | RXAUI Mode | LUTs | FFs |
|---|---|---|---|
| FALSE | Dune | 820 | 869 |
| TRUE | Dune | 952 | 969 |

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

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

| MDIO Management | RXAUI Mode | LUTs | FFs |
|---|---|---|---|
| FALSE | Dune | 737 | 826 |
| TRUE | Dune | 850 | 922 |

Table 2-3 provides approximate utilization for the various core options on Artix™-7 FPGAs using the Vivado Design Suite.

*Table 2-3:* **Device Utilization – Artix-7 FPGAs**

| MDIO Management | RXAUI Mode | LUTs | FFs |
|---|---|---|---|
| FALSE | Dune | 724 | 826 |
| TRUE | Dune | 875 | 922 |

# Port Descriptions

## Client-Side Interface

The signals of the client-side interface are shown in Table 2-4. See Protocol Description for details on connecting to the client-side interface.

*Table 2-4:* **Client-Side Interface Ports**

| Signal Name | Direction | Description |
|---|---|---|
| XGMII_TXD[63:0] | In | Transmit data, 8 bytes wide |
| XGMII_TXC[7:0] | In | Transmit control bits, 1-bit per transmit data byte |
| XGMII_RXD[63:0] | Out | Received data, 8 bytes wide |
| XGMII_RXC[7:0] | Out | Receive control bits, 1-bit per received data byte |

## Transceiver Interface

The following signals are directly connected to the transceiver instance.

*Table 2-5:* **Transceiver Interface Ports**

| Signal Name | Direction | Description |
|---|---|---|
| RXAUI_TX_L0_P, RXAUI_TX_L0_N, RXAUI_TX_L1_P, RXAUI_TX_L1 | Out | Differential complements of one another forming a differential transmit output pair. One pair for each of the 2 lanes. |
| RXAUI_RX_L0_P, RXAUI_RX_L0_N, RXAUI_RX_L1_P, RXAUI_RX_L1_N | In | Differential complements of one another forming a differential receiver input pair. One pair for each of the 2 lanes. |
| SIGNAL_DETECT[1:0] | In | 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 2-wire bus should be tied to 0x3. |

## 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 2-6. More information on using this interface can be found in MDIO Interface.

*Table 2-6:* **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 2-59. See Configuration and Status Vectors for details on these signals, including a breakdown of the configuration and status vectors.

*Table 2-7:* **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. |

## Clocking and Reset Signals and Module

Included in the example design output product 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 2-8 and Table 2-11 show the ports on the core that are associated with system clocks and resets.

*Table 2-8:* **Clock and Resets Option Deselected**

| Signal Name | Direction | Description |
|---|---|---|
| CLK156 | In | 156.25 MHz system clock for core. Must be derived from TXOUTCLK. |
| RESET156 | In | Reset port synchronous to CLK156. RESET156 is also the recommended method of resetting the transceiver. RESET156 should be asserted when CLK156 is not valid. If MMCM is used, MMCM_LOCK should be used, if not TXLOCK. |
| DCLK | In | Stable clock that is used in initialization logic for the transceivers and also connected to the DCLK port on the DRP interface of the transceiver. |
| RESET | In | Asynchronous reset connected to the transceiver Channel PLL (if used). This should be asserted if REFCLK is not valid. |
| TXLOCK | Out | Transceiver PLL Lock Status |
| MMCM_LOCK | In | MMCM lock indication. Only applicable if a MMCM is used to generate CLK156/CLK312. Otherwise, it can be tied to the TXLOCK output port. |
| TXOUTCLK | Out | Reference clock out from the transceiver. |

*Table 2-9:* **GTX and GTH Clocking Ports**

| Signal Name | Direction | Description |
|---|---|---|
| REFCLK | In | Transceiver reference clock. Must be driven from the appropriate differential clock buffer. |
| COMMON_PLL_CLK | In | Connect to the common PLL output clock QPLLOUTCLK. |
| COMMON_PLL_LOCK | In | Connect to the common PLL lock output QPLLLOCK. |
| COMMON_PLL_REFCLK | In | Connect to the common PLL output reference clock QPLLOUTREFCLK. |

*Table 2-10:* **GTP Clocking Ports**

| Signal Name | Direction | Description |
|---|---|---|
| REFCLK | In | Transceiver reference clock. Must be driven from the appropriate differential clock buffer. |
| COMMON_PLL0_CLK | In | Connect to the common PLL port PLL0OUTCLK. |
| COMMON_PLL0_LOCK | In | Connect to the common PLL lock port PLL0OLOCK. |
| COMMON_PLL0_REFCLK | In | Connect to the common PLL REFCLK port PLL0REFCLK. |
| CLK312 | In | 312.5 MHz clock is connected to the GTP TXUSRCLK pins. Must be derived from TXOUTCLK. |
| COMMON_PLL1_CLK | In | Connect to the common PLL port PLL1OUTCLK. |
| COMMON_PLL1_LOCK | In | Connect to the common PLL lock port PLL1LOCK. |
| COMMON_PLL1_REFCLK | In | Connect to the common PLL REFCLK port PLL1REFCLK. |

*Table 2-11:* **Clock and Resets Option Selected**

| Signal Name | Direction | Description |
|---|---|---|
| CLK156 | Out | 156.25 MHz output clock that can be used for FPGA logic. |
| CLK156_LOCKED | Out | CLK156 is stable and ready for use. |
| DCLK | Out | Stable clock that is used in initialization logic for the transceivers and also connected to the DCLK port on the DRP interface of the transceiver. |
| RESET | Out | Asynchronous reset connected to the transceiver Channel PLL (if used). |

*Table 2-12:* **GTX and GTH Clocking Ports**

| Signal Name | Direction | Description |
|---|---|---|
| REFCLK_P | Out | Differential transceiver reference clock “p.” |
| REFCLK_N | Out | Differential transceiver reference clock “n.” |
| COMMON_PLL_CLK | Out | Output from the common PLL port QPLLOUTCLK. |
| COMMON_PLL_LOCK | Out | Output from the common PLL port QPLLLOCK. |
| COMMON_PLL_REFCLK | Out | Output from the common PLL port reference clock QPLLREFCLK. |

*Table 2-13:* **GTP Clocking Ports**

| Signal Name | Direction | Description |
|---|---|---|
| COMMON_PLL0_CLK | Out | Output from the common PLL port COMMON_PLL0_CLK. |
| COMMON_PLL0_LOCK | Out | Output from the common PLL port COMMON_PLL0_LOCK. |
| COMMON_PLL0_REFCLK | Out | Output from the common PLL port COMMON_PLL0_REFCLK. |
| COMMON_PLL1_CLK | Out | Output from the common PLL port COMMON_PLL1_CLK. |
| COMMON_PLL1_LOCK | Out | Output from the common PLL port COMMON_PLL1_LOCK. |
| COMMON_PLL1_REFCLK | Out | Output from the common PLL port COMMON_PLL1_REFCLK. |

For a full example of how these ports should be connected to the common module, use the "Support Layer" in the IP example design output products.

## DRP Interface

The DRP interfaces on the transceivers are brought out to the core top-level. These can be used if required, or tied off. For a full description of the DRP interface see the *7 Series FPGAs Configuration User Guide* [Ref 5].

*Table 2-14:* **DRP Ports**

| Signal Name | Direction | Description |
|---|---|---|
| DRP0_ADDR | In | DRP address for transceiver 0 |
| DRP0_EN | In | DRP enable for transceiver 0 |
| DRP0_I | In | DRP data in for transceiver 0 |
| DRP0_O | Out | DRP data out for transceiver 0 |
| DRP0_RDY | Out | DRP ready for transceiver 0 |
| DRP0_WE | – | DRP write enable for transceiver 0 |
| DRP1_ADDR | In | DRP address for transceiver 1 |
| DRP1_EN | In | DRP enable for transceiver 1 |
| DRP1_I | In | DRP data in for transceiver 1 |
| DRP1_O | Out | DRP data out for transceiver 1 |
| DRP1_RDY | Out | DRP ready for transceiver 1 |
| DRP1_WE | – | DRP write enable for transceiver 1 |

## GT Control Interface

This bus is connected to GT transceiver ports. This allows control of various transceiver functions. If these ports are not required they can be tied to default values. The example design output products show these values being tied off.

*Table 2-15:* **GT Control Ports**

| Signal Name | Direction | Description |
|---|---|---|
| GT_CONTROL | In | Transceiver Specific Control Ports. Defaults for GT_CONTROL can be found in the example design.<br>**GTP Transceiver**:<br>Bit[30] = GT1 RXLPMHFHOLD<br>Bit[29] = GT1 RXLPMLFHOLD<br>Bit[1] = GT0 RXLPMHFHOLD<br>Bit[0] = GT0 RXLPMLFHOLD<br>**GTX/GTH Transceiver**:<br>Bit[30] = GT1 RXLPMEN<br>Bit[29] = GT1 RXDFELPMRESET<br>Bit[1] = GT0 RXLPMEN<br>Bit[0] = GT0 RXDFELPMRESET<br>The following signals are common to all transceivers.<br>**These signals apply to transceiver 0**:<br>Bits[28:26] = LOOPBACK<br>Bit[25] = TXPOLARITY<br>Bits[24:20] = TXPRECURSOR<br>Bits[19:15] = TXPOSTCURSOR<br>Bits[14:11] = TXDIFFCTRL<br>Bits[10:8] = TXPRBSSEL<br>Bit[7] = TXPRBSFORCEERR<br>Bit[6] = RXPOLARITY<br>Bit[5] = RXPRBSCNTRESET<br>Bits[4:2] = RXPRBSSEL<br>**These signals apply to transceiver 1**:<br>Bits[57:55] = LOOPBACK<br>Bit[54] = TXPOLARITY<br>Bits[53:49] = TXPRECURSOR<br>Bits[48:44] = TXPOSTCURSOR<br>Bits[43:40] = TXDIFFCTRL<br>Bits[39:37] = TXPRBSSEL<br>Bit[36] = TXPRBSFORCEERR<br>Bit[35] = RXPOLARITY<br>Bit[34] = RXPRBSCNTRESET<br>Bits[33:31] = RXPRBSSEL |

1. The Actual GT Loopback can be set by the external gt_control port input OR the XAUI Loopback bit (set by either the MDIO register or configuration vector). You should not drive both the XAUI Loopback and gt_control ports simultaneously.

## Debug Interface

A debug port is provided so it can connect to external logic for debug (for example, processor monitoring). This port contains transceiver and core debug information.

*Table 2-16:* **Debug Ports**

| Signal Name | Direction | Description |
|---|---|---|
| DEBUG | Out | Debug Port<br>**Dune RXAUI Port**:<br>Bit[8] = mgt_enchansync<br>Bit[7] = mgt_rx_reset<br>Bit[6] = mgt_tx_reset<br>Bit[5] = Align Status<br>Bits[4:1] = Sync Status<br>Bit[0] = TX Phase Align Complete<br>**These signals apply to transceiver 0**:<br>Bits[38:35] = rxchariscomma<br>Bits[34:31] = rxcharisk<br>Bits[30:27] = rxdisperr<br>Bits[26:23] = rxnotintable<br>Bits[22:20] = rxbufstatus<br>Bit[19] = mgt_rxresetdone<br>Bit[18] = rxchanbondseq<br>Bit[17] = rxchanisaligned<br>Bit[16] = rxchanrealign<br>Bits[15:14] = rxclkcorcnt<br>Bit[13] = rxbyteisaligned<br>Bit[12] = rxbyterealign<br>Bit[11] = rxcommadet<br>Bit[10] = rxprnserr<br>Bit[9] = mgt_txresetdone<br>**These signals apply to transceiver 1**:<br>Bits[68:65] = rxchariscomma<br>Bits[64:61] = rxcharisk<br>Bits[60:57] = rxdisperr<br>Bits[56:53] = rxnotintable<br>Bits[52:50] = rxbufstatus<br>Bit[49] = rxresetdone<br>Bit[48] = rxchanbondseq<br>Bit[47] = rxchanisaligned<br>Bit[46] = rxchanrealign<br>Bits[45:44] = rxclkcorcnt<br>Bit[43] = rxbyteisaligned<br>Bit[42] = rxbyterealign<br>Bit[41] = rxcommadet<br>Bit[40] = rxprbserr<br>Bit[39] = txresetdone |

# 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 2-17.

*Table 2-17:* **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 |

*Table 2-17:* **10GBASE-X PCS/PMA MDIO Registers *(Cont'd)***

| Register Address | Register Name |
|---|---|
| 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.

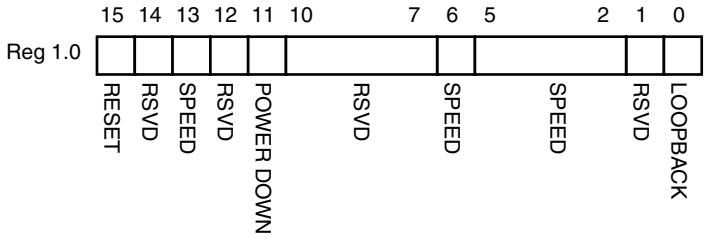


*Figure 2-1:* **PMA/PMD Control 1 Register**

Table 2-18 shows the PMA Control 1 register bit definitions.

*Table 2-18:* **PMA/PMD Control 1 Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.0.15 | Reset | 0 | R/W<br>Self-clearing | 0 = Normal operation<br>1 = Block reset<br>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. |
| 1.0.14 | Reserved | 0 | R | The block always returns 0 for this bit and ignores writes. |
| 1.0.13 | Speed Selection | 1 | R | The block always returns 1 for this bit and ignores writes. |
| 1.0.12 | Reserved | 0 | R | The block always returns 0 for this bit and ignores writes. |

*Table 2-18:* **PMA/PMD Control 1 Register Bit Definitions *(Cont'd)***

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.0.11 | Power down | 0 | R/W | 0 = Normal operation<br>1 = Power down mode<br>When set to 1, the serial transceivers are placed in a low power state. Set to 0 to return to normal operation |
| 1.0.10:7 | Reserved | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 1.0.6 | Speed Selection | 1 | R | The block always returns 1 for this bit and ignores writes. |
| 1.0.5:2 | Speed Selection | All 0s | R | The block always returns 0s for these bits and ignores writes. |
| 1.0.1 | Reserved | All 0s | R | The block always returns 0 for this bit and ignores writes |
| 1.0.0 | Loopback | 0 | R/W | 0 = Disable loopback mode<br>1 = Enable loopback mode<br>The RXAUI block loops the signal in the serial transceivers back into the receiver. Near-end PMA loopback is always used. |

### MDIO Register 1.1: PMA/PMD Status 1

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

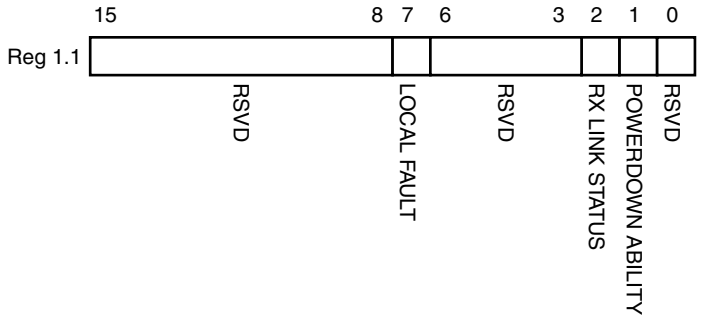


*Figure 2-2:* **PMA/PMD Status 1 Register**

Table 2-19 shows the PMA/PMD Status 1 register bit definitions.

*Table 2-19:* **PMA/PMD Status 1 Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.1.15:8 | Reserved | 0 | R | The block always returns 0 for this bit. |
| 1.1.7 | Local Fault | 0 | R | The block always returns 0 for this bit. |
| 1.1.6:3 | Reserved | 0 | R | The block always returns 0 for this bit. |
| 1.1.2 | Receive Link Status | 1 | R | The block always returns 1 for this bit. |

*Table 2-19:* **PMA/PMD Status 1 Register Bit Definitions *(Cont'd)***

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.1.1 | Power Down Ability | 1 | R | The block always returns 1 for this bit. |
| 1.1.0 | Reserved | 0 | R | The block always returns 0 for this bit. |

### 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.

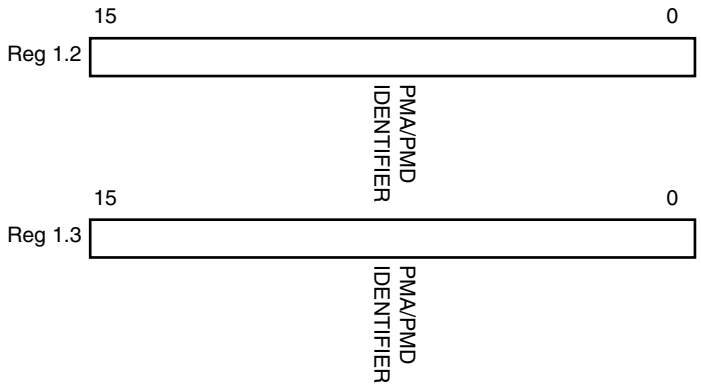


*Figure 2-3:* **PMA/PMD Device Identifier Registers**

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

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

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.2.15:0 | PMA/PMD Identifier | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 1.3.15:0 | PMA/PMD Identifier | All 0s | R | The block always returns 0 for these bits and ignores writes. |

### MDIO Register 1.4: PMA/PMD Speed Ability

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

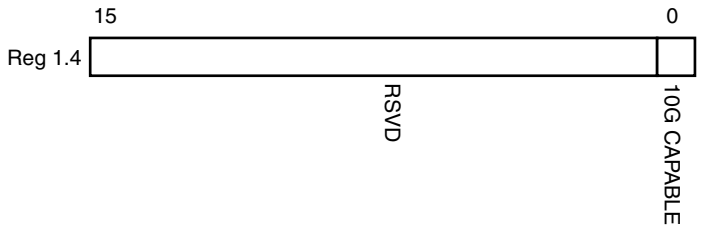


*Figure 2-4:* **PMA/PMD Speed Ability Register**

Table 2-21 shows the PMA/PMD Speed Ability register bit definitions.

*Table 2-21:* **PMA/PMD Speed Ability Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.4.15:1 | Reserved | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 1.4.0 | 10G Capable | 1 | R | The block always returns 1 for this bit and ignores writes. |

### 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.

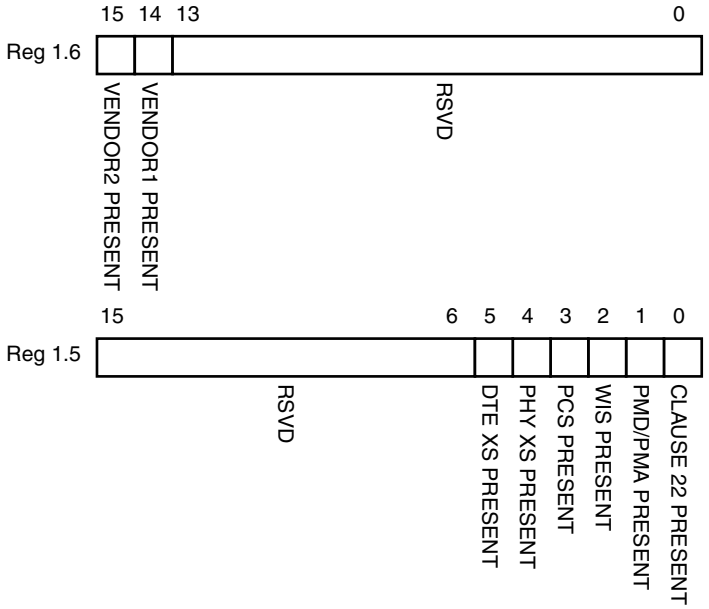


*Figure 2-5:* **PMA/PMD Devices in Package Registers**

Table 2-22 shows the PMA/PMD Device in Package registers bit definitions.

*Table 2-22:* **PMA/PMD Devices in Package Registers Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.6.15 | Vendor- specific Device 2 present | 0 | R | The block always returns 0 for this bit. |
| 1.6.14 | Vendor-specific Device 1 present | 0 | R | The block always returns 0 for this bit. |
| 1.6.13:0 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 1.5.15:6 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 1.5.5 | DTE XS present | 0 | R | The block always returns 0 for this bit. |
| 1.5.4 | PHY XS present | 0 | R | The block always returns 0 for this bit. |
| 1.5.3 | PCS present | 1 | R | The block always returns 1 for this bit. |
| 1.5.2 | WIS present | 0 | R | The block always returns 0 for this bit. |

*Table 2-22:* **PMA/PMD Devices in Package Registers Bit Definitions *(Cont'd)***

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.5.1 | PMA/PMD present | 1 | R | The block always returns 1 for this bit. |
| 1.5.0 | Clause 22 Device present | 0 | R | The block always returns 0 for this bit. |

### MDIO Register 1.7: 10G PMA/PMD Control 2

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

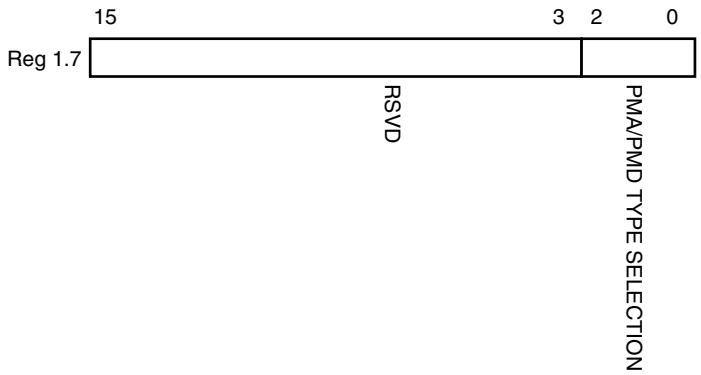


*Figure 2-6:* **10G PMA/PMD Control 2 Register**

Table 2-23 shows the PMA/PMD Control 2 register bit definitions.

*Table 2-23:* **10G PMA/PMD Control 2 Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.7.15:3 | Reserved | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 1.7.2:0 | PMA/PMD Type Selection | 100 | R | The block always returns 100 for these bits and ignores writes. This corresponds to the 10GBASE-X PMA/PMD. |

### MDIO Register 1.8: 10G PMA/PMD Status 2

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

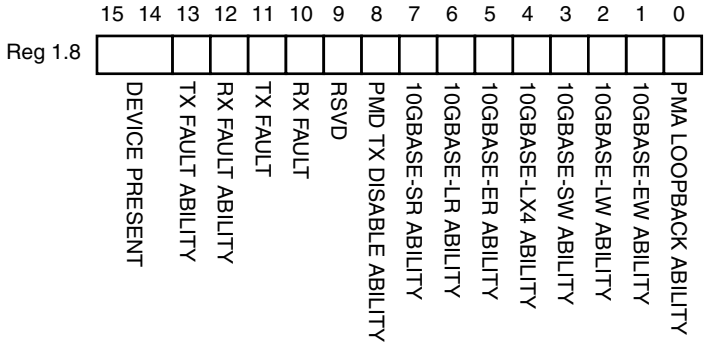


*Figure 2-7:* **10G PMA/PMD Status 2 Register**

Table 2-24 shows the PMA/PMD Status 2 register bit definitions.

*Table 2-24:* **10G PMA/PMD Status 2 Register Bit Definitions**

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

### MDIO Register 1.10: 10G PMD Signal Receive OK

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

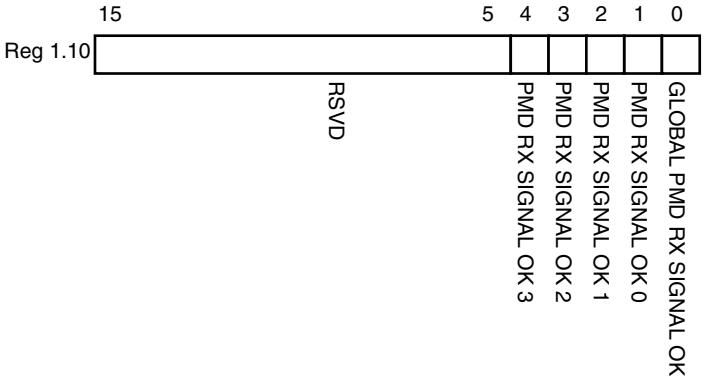


*Figure 2-8:* **10G PMD Signal Receive OK Register**

Table 2-25 shows the 10G PMD Signal Receive OK register bit definitions.

*Table 2-25:* **10G PMD Signal Receive OK Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.10.15:5 | Reserved | All 0s | R | The block always returns 0s for these bits. |
| 1.10.4 | PMD Receive Signal OK 3 | – | R | 0 = Signal not OK on receive Lane 3<br>1 = Signal OK on receive Lane 3<br>This is the value of the SIGNAL_DETECT[1] port. |
| 1.10.3 | PMD Receive Signal OK 2 | – | R | 0 = Signal not OK on receive Lane 2<br>1 = Signal OK on receive Lane 2<br>This is the value of the SIGNAL_DETECT[1] port. |
| 1.10.2 | PMD Receive Signal OK 1 | – | R | 0 = Signal not OK on receive Lane 1<br>1 = Signal OK on receive Lane 1<br>This is the value of the SIGNAL_DETECT[0] port. |
| 1.10.1 | PMD Receive Signal OK 0 | – | R | 0 = Signal not OK on receive Lane 0<br>1 = Signal OK on receive Lane 0<br>This is the value of the SIGNAL_DETECT[0] port. |
| 1.10.0 | Global PMD Receive Signal OK | – | R | 0 = Signal not OK on all receive lanes<br>1 = Signal OK on all receive lanes |

### 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.

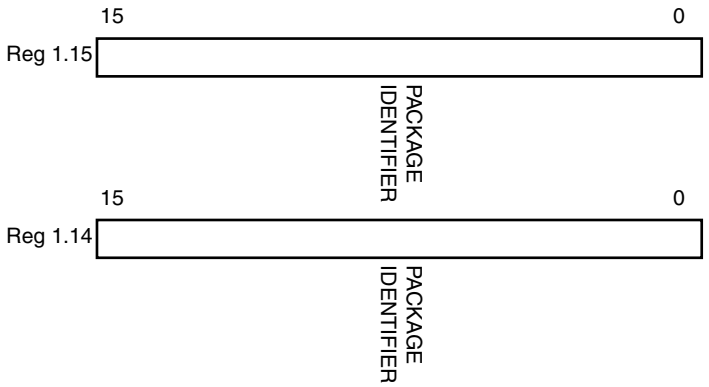


*Figure 2-9:* **PMA/PMD Package Identifier Registers**

Table 2-26 shows the PMA/PMD Package Identifier registers bit definitions.

*Table 2-26:* **PMA/PMD Package Identifier Registers Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 1.15.15:0 | PMA/PMD Package Identifier | All 0s | R | The block always returns 0 for these bits. |
| 1.14.15:0 | PMA/PMD Package Identifier | All 0s | R | The block always returns 0 for these bits. |

### MDIO Register 3.0: PCS Control 1

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

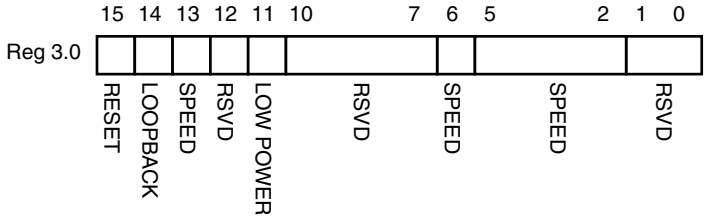


*Figure 2-10:* **PCS Control 1 Register**

Table 2-27 shows the PCS Control 1 register bit definitions.

*Table 2-27:* **PCS Control 1 Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 3.0.15 | Reset | 0 | R/W<br>Self-clearing | 0 = Normal operation<br>1 = Block reset<br>The RXAUI block is reset when this bit is set to 1. It returns to 0 when the reset is complete. |
| 3.0.14 | 10GBASE-R Loopback | 0 | R | The block always returns 0 for this bit and ignores writes. |

*Table 2-27:* **PCS Control 1 Register Bit Definitions *(Cont'd)***

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 3.0.13 | Speed Selection | 1 | R | The block always returns 1 for this bit and ignores writes. |
| 3.0.12 | Reserved | 0 | R | The block always returns 0 for this bit and ignores writes. |
| 3.0.11 | Power down | 0 | R/W | 0 = Normal operation<br>1 = Power down mode<br>When set to 1, the serial transceivers are placed in a low power state. Set to 0 to return to normal operation. |
| 3.0.10:7 | Reserved | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 3.0.6 | Speed Selection | 1 | R | The block always returns 1 for this bit and ignores writes. |
| 3.0.5:2 | Speed Selection | All 0s | R | The block always returns 0s for these bits and ignores writes. |
| 3.0.1:0 | Reserved | All 0s | R | The block always returns 0 for this bit and ignores writes. |

### MDIO Register 3.1: PCS Status 1

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

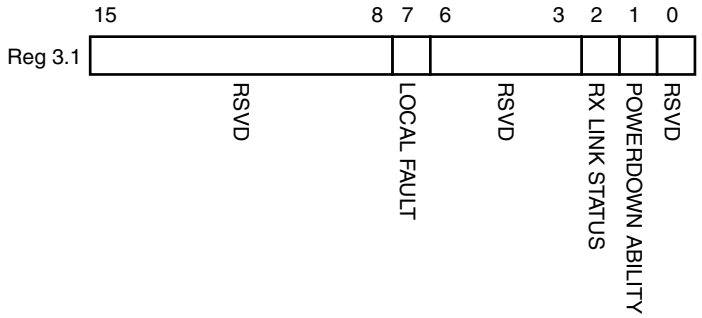


*Figure 2-11:* **PCS Status 1 Register**

Table 2-28 show the PCS 1 register bit definitions.

*Table 2-28:* **PCS Status 1 Register Bit Definition**

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

### 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.

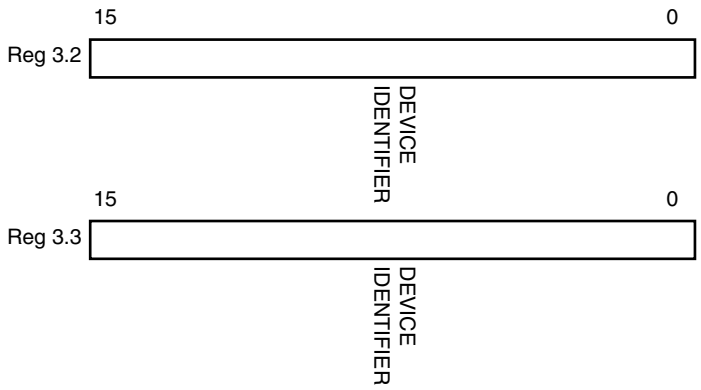


*Figure 2-12:* **PCS Device Identifier Registers**

Table 2-29 shows the PCS Device Identifier registers bit definitions.

*Table 2-29:* **PCS Device Identifier Registers Bit Definition**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 3.2.15:0 | PCS Identifier | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 3.3.15:0 | PCS Identifier | All 0s | R | The block always returns 0 for these bits and ignores writes. |

### MDIO Register 3.4: PCS Speed Ability

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

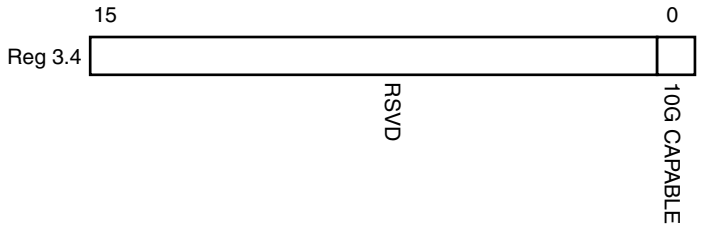


*Figure 2-13:* **PCS Speed Ability Register**

Table 2-30 shows the PCS Speed Ability register bit definitions.

*Table 2-30:* **PCS Speed Ability Register Bit Definition**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 3.4.15:1 | Reserved | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 3.4.0 | 10G Capable | 1 | R | The block always returns 1 for this bit and ignores writes. |

### 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.

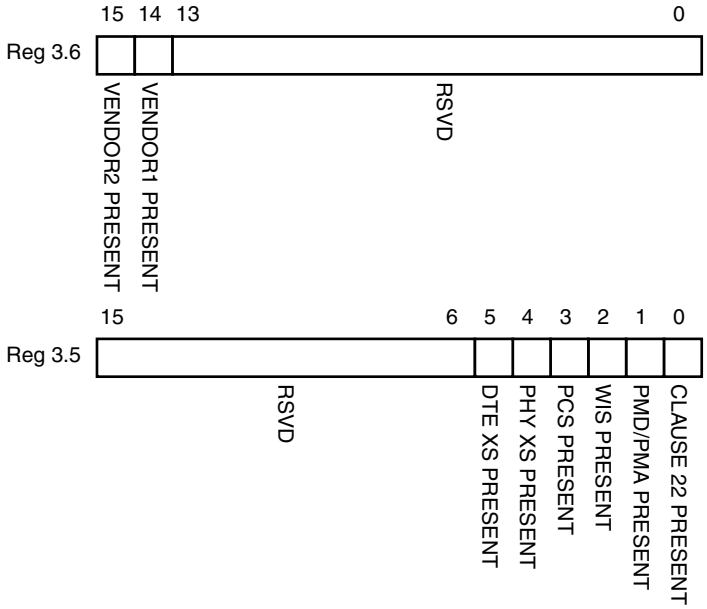


*Figure 2-14:* **PCS Devices in Package Registers**

Table 2-31 shows the PCS Devices in Package registers bit definitions.

*Table 2-31:* **PCS Devices in Package Registers Bit Definitions**

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

### MDIO Register 3.7: 10G PCS Control 2

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

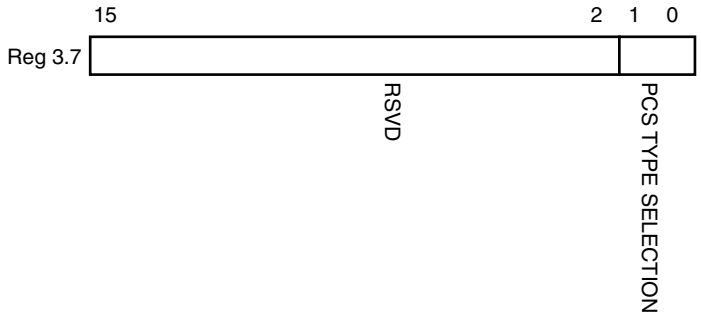


*Figure 2-15:* **10G PCS Control 2 Register**

Table 2-32 shows the 10 G PCS Control 2 register bit definitions.

*Table 2-32:* **10G PCS Control 2 Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 3.7.15:2 | Reserved | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 3.7.1:0 | PCS Type Selection | 01 | R | The block always returns 01 for these bits and ignores writes. |

### MDIO Register 3.8: 10G PCS Status 2

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

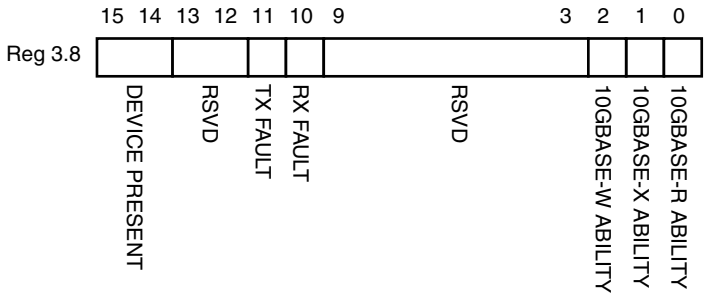


*Figure 2-16:* **10G PCS Status 2 Register**

Table 2-33 shows the 10G PCS Status 2 register bit definitions.

*Table 2-33:* **10G PCS Status 2 Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 3.8.15:14 | Device present | 10 | R | The block always returns 10. |
| 3.8.13:12 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 3.8.11 | Transmit local fault | – | R<br>Latching High | 0 = No fault condition on transmit path<br>1 = Fault condition on transmit path |
| 3.8.10 | Receive local fault | – | R<br>Latching High | 0 = No fault condition on receive path<br>1 = Fault condition on receive path |
| 3.8.9:3 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 3.8.2 | 10GBASE-W Capable | 0 | R | The block always returns 0 for this bit. |
| 3.8.1 | 10GBASE-X Capable | 1 | R | The block always returns 1 for this bit. |
| 3.8.0 | 10GBASE-R Capable | 0 | R | The block always returns 0 for this bit. |

### 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.

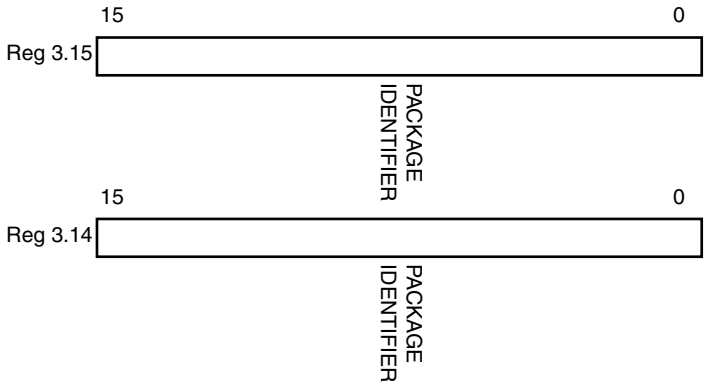


*Figure 2-17:* **Package Identifier Registers**

Table 2-34 shows the PCS Package Identifier registers bit definitions.

*Table 2-34:* **PCS Package Identifier Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 3.14.15:0 | Package Identifier | All 0s | R | The block always returns 0 for these bits. |
| 3.15.15:0 | Package Identifier | All 0s | R | The block always returns 0 for these bits. |

### MDIO Register 3.24: 10GBASE-X Status

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

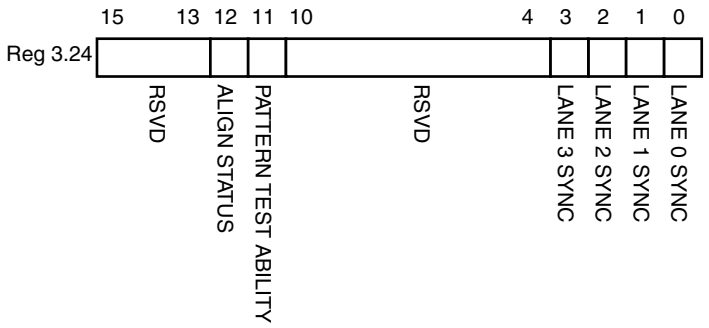


*Figure 2-18:* **10GBASE-X Status Register**

Table 2-35 shows the 10GBase-X Status register bit definitions.

*Table 2-35:* **10GBASE-X Status Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 3.24.15:13 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 3.24.12 | 10GBASE-X Lane Alignment Status | – | R | 0 = 10GBASE-X receive lanes not aligned<br>1 = 10GBASE-X receive lanes aligned |
| 3.24.11 | Pattern Testing Ability | 1 | R | The block always returns 1 for this bit. |
| 3.24.10:4 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 3.24.3 | Lane 3 Sync | – | R | 0 = Lane 3 is not synchronized<br>1 = Lane 3 is synchronized |
| 3.24.2 | Lane 2 Sync | – | R | 0 =Lane 2 is not synchronized<br>1 =Lane 2 is synchronized |
| 3.24.1 | Lane 1 Sync | – | R | 0 = Lane 1 is not synchronized<br>1 = Lane 1 is synchronize |
| 3.24.0 | Lane 0 Sync | – | R | 0 = Lane 0 is not synchronized<br>1 = Lane 0 is synchronized |

### MDIO Register 3.25: 10GBASE-X Test Control

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

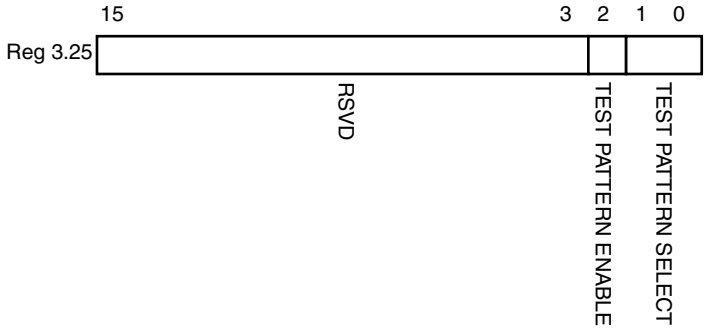


*Figure 2-19:* **Test Control Register**

Table 2-36 shows the 10GBase-X Test Control register bit definitions.

*Table 2-36:* **10GBASE-X Test Control Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 3.25.15:3 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 3.25.2 | Transmit Test Pattern Enable | 0 | R/W | 0 = Transmit test pattern disabled<br>1 = Transmit test pattern enable |
| 3.25.1:0 | Test Pattern Select | 00 | R/W | 00 = High frequency test pattern<br>01 = Low frequency test pattern<br>10 = Mixed frequency test pattern<br>11 = Reserved |

## 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 2-37).

*Table 2-37:* **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 |

*Table 2-37:* **DTE XS MDIO Registers *(Cont'd)***

| Register Address | Register Name |
|---|---|
| 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.

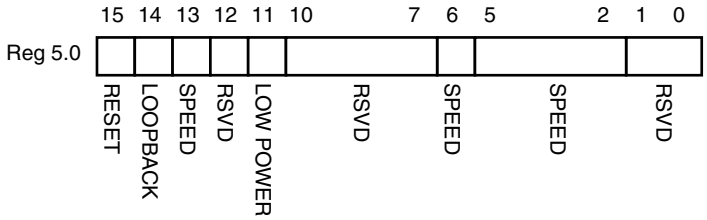


*Figure 2-20:* **DTE XS Control 1 Register**

Table 2-38 shows the DTE XS Control 1 register bit definitions.

*Table 2-38:* **DTE XS Control 1 Register Bit Definitions**

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

*Table 2-38:* **DTE XS Control 1 Register Bit Definitions *(Cont'd)***

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 5.0.5:2 | Speed Selection | All 0s | R | The block always returns 0s for these bits and ignores writes. |
| 5.0.1:0 | Reserved | All 0s | R | The block always returns 0s for these bits and ignores writes. |

### MDIO Register 5.1: DTE XS Status 1

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

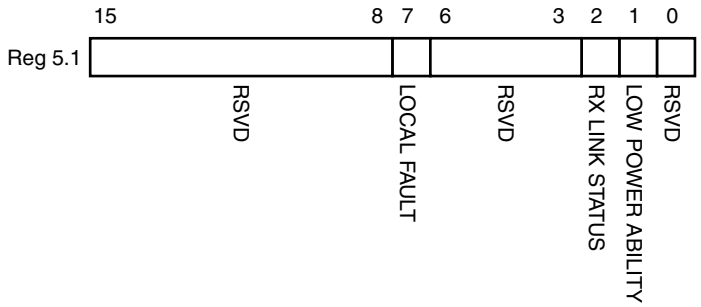


*Figure 2-21:* **DTE XS Status 1 Register**

Table 2-39 shows the DET XS Status 1 register bit definitions.

*Table 2-39:* **DTE XS Status 1 Register Bit Definitions**

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

### 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.

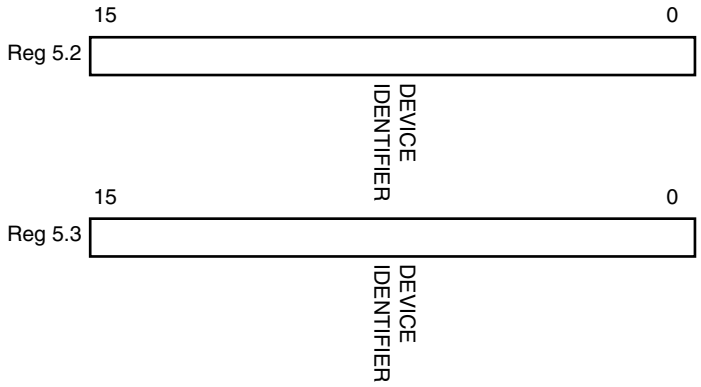


*Figure 2-22:* **DTE XS Device Identifier Registers**

Table 2-40 shows the DTE XS Device Identifier registers bit definitions.

*Table 2-40:* **DTE XS Device Identifier Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 5.2.15:0 | DTE XS Identifier | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 5.3.15:0 | DTE XS Identifier | All 0s | R | The block always returns 0 for these bits and ignores writes. |

### MDIO Register 5.4: DTE XS Speed Ability

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

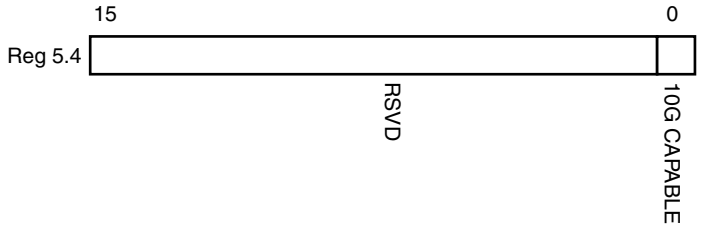


*Figure 2-23:* **DTE XS Speed Ability Register**

Table 2-41 shows the DTE XS Speed Ability register bit definitions.

*Table 2-41:* **DTE XS Speed Ability Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 5.4.15:1 | Reserved | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 5.4.0 | 10G Capable | 1 | R | The block always returns 1 for this bit and ignores writes. |

### 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.

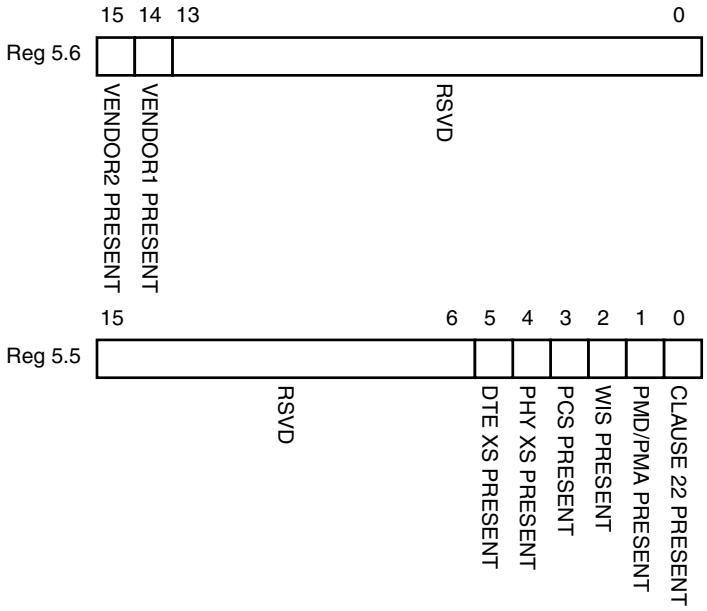


*Figure 2-24:* **DTE XS Devices in Package Register**

Table 2-42 shows the DTE XS Devices in Package registers bit definitions.

*Table 2-42:* **DTE XS Devices in Package Registers Bit Definitions**

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

### MDIO Register 5.8: DTE XS Status 2

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

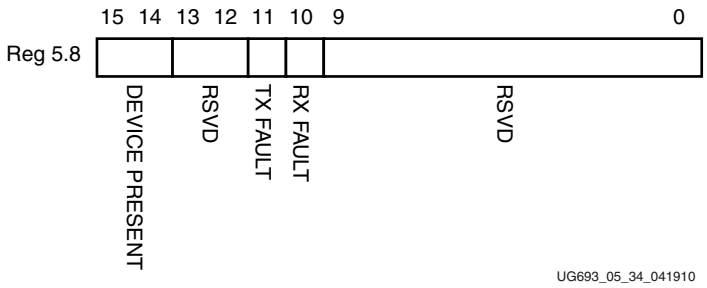


*Figure 2-25:* **DTE XS Status 2 Register**

Table 2-43 show the DTE XS Status 2 register bits definitions.

*Table 2-43:* **DTE XS Status 2 Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 5.8.15:14 | Device present | 10 | R | The block always returns 10. |
| 5.8.13:12 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 5.8.11 | Transmit Local Fault | – | R<br>Latching High | 0 = No fault condition on transmit path<br>1 = Fault condition on transmit path |
| 5.8.10 | Receive Local Fault | – | R<br>Latching High | 0 = No fault condition on receive path<br>1 = Fault condition on receive path |
| 5.8.9:0 | Reserved | All 0s | R | The block always returns 0 for these bits. |

### 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.

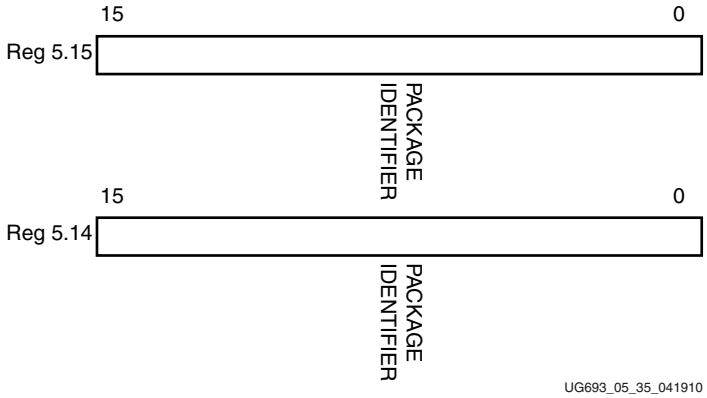


*Figure 2-26:* **DTE XS Package Identifier Registers**

Table 2-44 shows the DTE XS Package Identifier registers bit definitions.

*Table 2-44:* **DTE XS Package Identifier Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 5.14.15:0 | DTE XS Package Identifier | All 0s | R | The block always returns 0 for these bits. |
| 5.15.15:0 | DTE XS Package Identifier | All 0s | R | The block always returns 0 for these bits. |

## 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 0.... at each device-specific transceiver output
- mixed frequency test pattern of 11111010110000010100111110101100000 1010... 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.

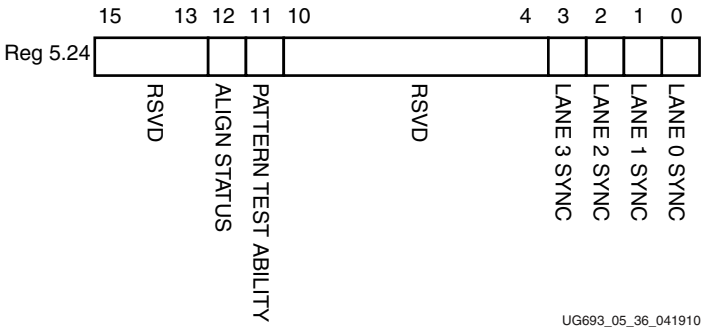


*Figure 2-27:* **DTE XS Lane Status Register**

Table 2-45 shows the DTE XS Lane Status register bit definitions.

*Table 2-45:* **DTE XS Lane Status Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 5.24.15:13 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 5.24.12 | DTE XGXS Lane Alignment Status | – | R | 0 = DTE XGXS receive lanes not aligned<br>1 = DTE XGXS receive lanes aligned |

*Table 2-45:* **DTE XS Lane Status Register Bit Definitions *(Cont'd)***

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 5.24.11 | Pattern testing ability | 1 | R | The block always returns 1 for this bit. |
| 5.24.10:4 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 5.24.3 | Lane 3 Sync | – | R | 0 = Lane 3 is not synchronized<br>1 = Lane 3 is synchronized |
| 5.24.2 | Lane 2 Sync | – | R | 0 = Lane 2 is not synchronized<br>1 = Lane 2 is synchronized |
| 5.24.1 | Lane 1 Sync | – | R | 0 = Lane 1 is not synchronized<br>1 = Lane 1 is synchronized |
| 5.24.0 | Lane 0 Sync | – | R | 0 = Lane 0 is not synchronized<br>1 = Lane 0 is synchronized |

### MDIO Register 5.25: 10G DTE XGXS Test Control

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

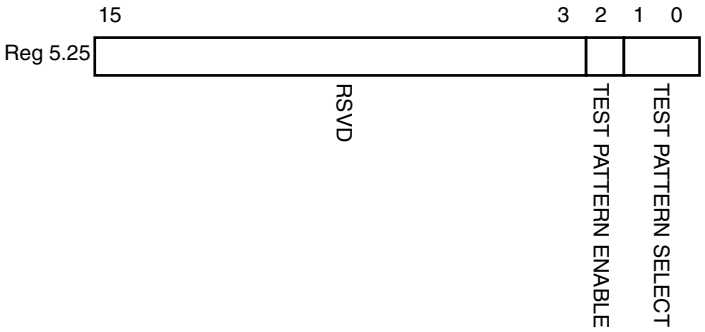


*Figure 2-28:* **10G DTE XGXS Test Control Register**

Table 2-46 shows the 10G DTE XGXS Test Control register bit definitions.

*Table 2-46:* **10G DTE XGXS Test Control Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 5.25.15:3 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 5.25.2 | Transmit Test Pattern Enable | 0 | R/W | 0 = Transmit test pattern disabled<br>1 = Transmit test pattern enable |
| 5.25.1:0 | Test Pattern Select | 00 | R/W | 00 = High frequency test pattern<br>01 = Low frequency test pattern<br>10 = Mixed frequency test pattern<br>11 = Reserved |

## 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 2-47).

*Table 2-47:* **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.

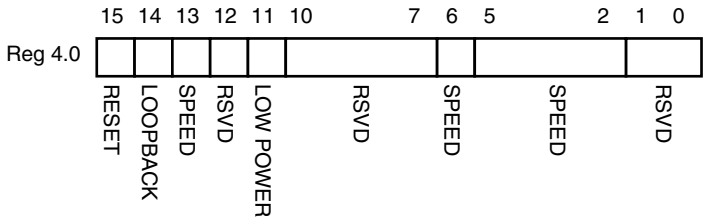


*Figure 2-29:* **PHY XS Control 1 Register**

Table 2-48 shows the PHY XS Control 1 register bit definitions.

*Table 2-48:* **PHY XS Control 1 Register Bit Definitions**

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

### MDIO Register 4.1: PHY XS Status 1

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

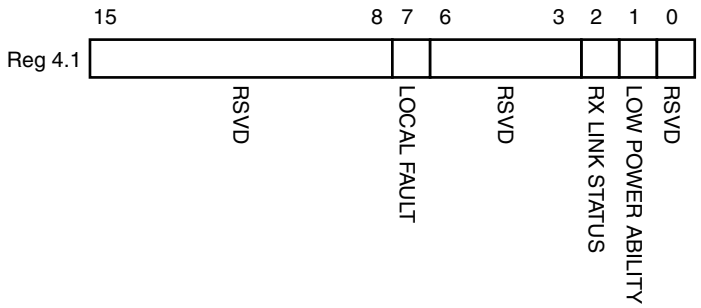


*Figure 2-30:* **PHY XS Status 1 Register**

Table 2-49 shows the PHY XS Status 1 register bit definitions.

*Table 2-49:* **PHY XS Status 1 Register Bit Definitions**

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

### 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.

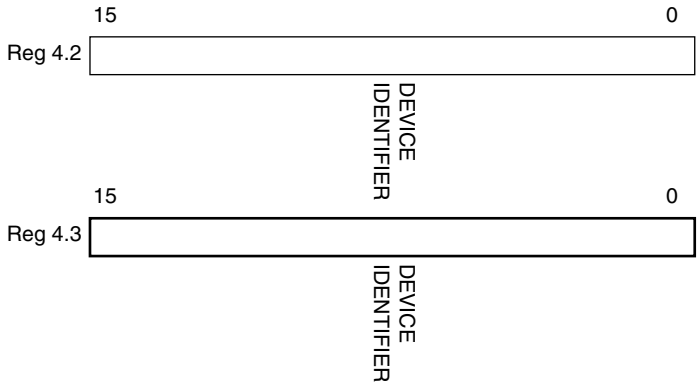


*Figure 2-31:* **PHY XS Device Identifier Registers**

Table 2-50 shows the PHY XS Devices Identifier registers bit definitions.

*Table 2-50:* **PHY XS Device Identifier Registers Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 4.2.15:0 | PHY XS Identifier | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 4.3.15:0 | PHY XS Identifier | All 0s | R | The block always returns 0 for these bits and ignores writes. |

### MDIO Register 4.4: PHY XS Speed Ability

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

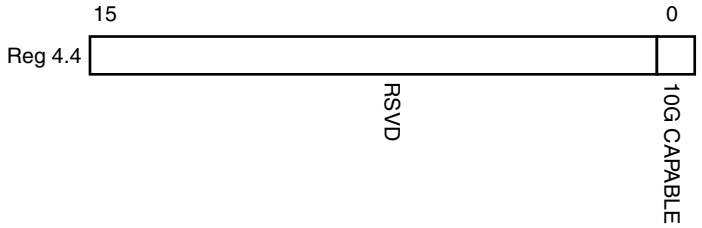


*Figure 2-32:* **PHY XS Speed Ability Register**

Table 2-51 shows the PHY XS Speed Ability register bit definitions.

*Table 2-51:* **PHY XS Speed Ability Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 4.4.15:1 | Reserved | All 0s | R | The block always returns 0 for these bits and ignores writes. |
| 4.4.0 | 10G Capable | 1 | R | The block always returns 1 for this bit and ignores writes. |

### 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.

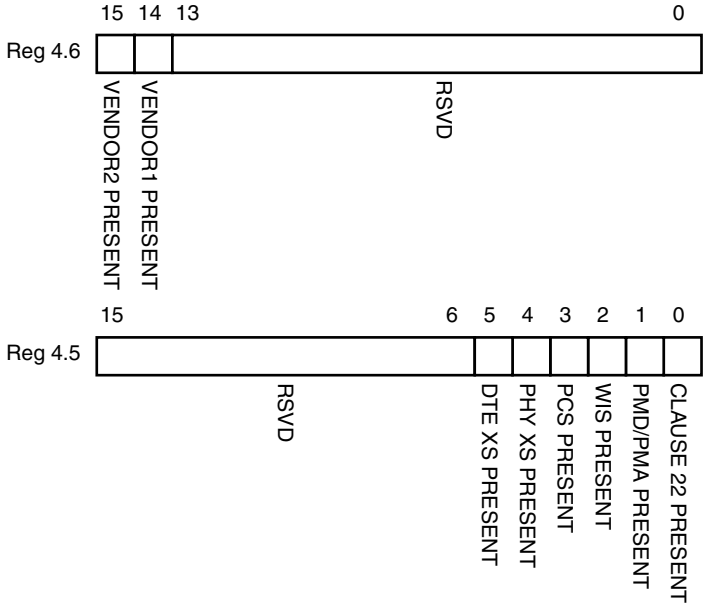


*Figure 2-33:* **PHY XS Devices in Package Registers**

Table 2-52 shows the PHY XS Devices in Package registers bit definitions.

*Table 2-52:* **PHY XS Devices in Package Registers Bit Definitions**

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

### MDIO Register 4.8: PHY XS Status 2

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

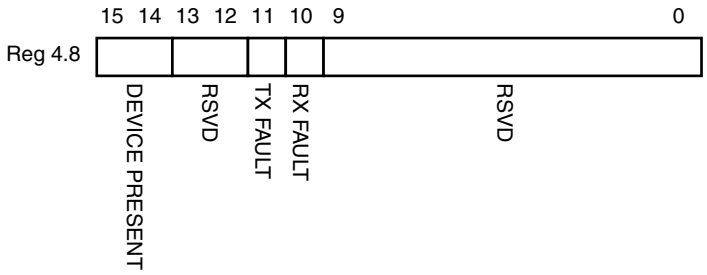


*Figure 2-34:* **PHY XS Status 2 Register**

Table 2-53 shows the PHY XS Status 2 register bit definitions.

*Table 2-53:* **PHY XS Status 2 Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 4.8.15:14 | Device present | 10 | R | The block always returns 10. |
| 4.8.13:12 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 4.8.11 | Transmit local fault | – | R<br>Latching High | 0 = No fault condition on transmit path<br>1 = Fault condition on transmit path |
| 4.8.10 | Receive local fault | – | R<br>Latching High | 0 = No fault condition on receive path<br>1 = Fault condition on receive path |
| 4.8.9:0 | Reserved | All 0s | R | The block always returns 0 for these bits. |

### 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.

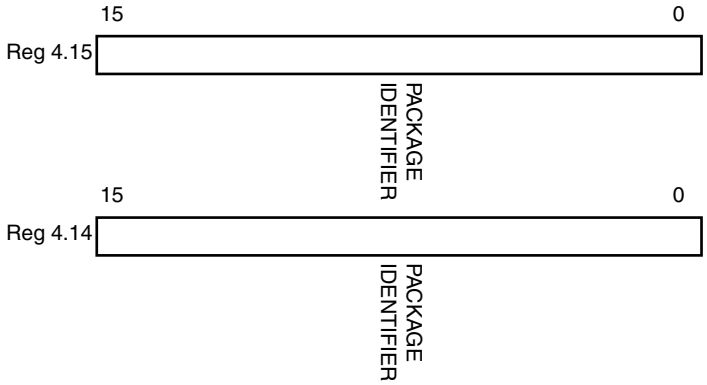


*Figure 2-35:* **PHY XS Package Identifier Registers**

Table 2-54 shows the Package Identifier registers bit definitions.

*Table 2-54:* **Package Identifier Registers Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 4.15.15:0 | PHY XS Package Identifier | All 0s | R | The block always returns 0 for these bits. |
| 4.14.15:0 | PHY XS Package Identifier | All 0s | R | The block always returns 0 for these bits. |

### MDIO Register 4.24: 10G PHY XGXS Lane Status

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

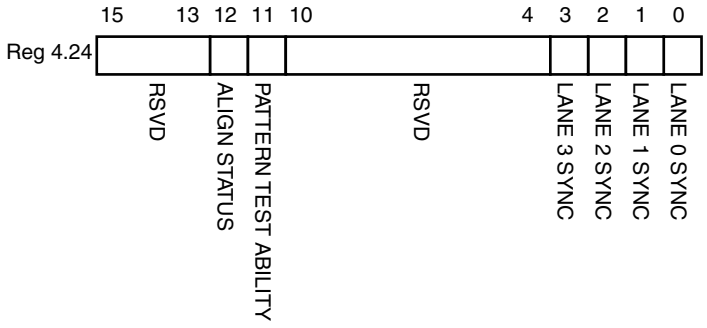


*Figure 2-36:* **10G PHY XGXS Lane Status Register**

Table 2-55 shows the 10G PHY XGXS Lane register bit definitions.

*Table 2-55:* **10G PHY XGXS Lane Status Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 4.24.15:13 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 4.24.12 | PHY XGXS Lane Alignment Status | – | R | 0 = PHY XGXS receive lanes not aligned<br>1 = PHY XGXS receive lanes aligned |
| 4.24.11 | Pattern Testing Ability | 1 | R | The block always returns 1 for this bit. |
| 4.24.10:4 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 4.24.3 | Lane 3 Sync | – | R | 0 = Lane 3 is not synchronized<br>1 = Lane 3 is synchronized |
| 4.24.2 | Lane 2 Sync | – | R | 0 = Lane 2 is not synchronized<br>1 = Lane 2 is synchronized |
| 4.24.1 | Lane 1 Sync | – | R | 0 = Lane 1 is not synchronized<br>1 = Lane 1 is synchronized |
| 4.24.0 | Lane 0 Sync | – | R | 0 = Lane 0 is not synchronized<br>1 = Lane 0 is synchronized |

### MDIO Register 4.25: 10G PHY XGXS Test Control

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

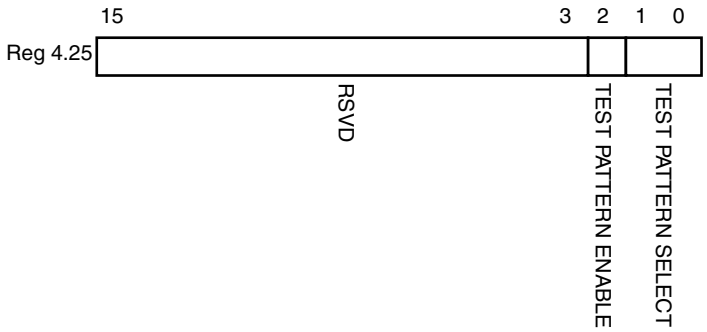


*Figure 2-37:* **10G PHY XGXS Test Control Register**

Table 2-56 shows the 10G PHY XGXS Test Control register bit definitions.

*Table 2-56:* **10G PHY XGXS Test Control Register Bit Definitions**

| Bits | Name | Reset Value | Access Type | Description |
|---|---|---|---|---|
| 4.25.15:3 | Reserved | All 0s | R | The block always returns 0 for these bits. |
| 4.25.2 | Transmit Test Pattern Enable | 0 | R/W | 0 = Transmit test pattern disabled<br>1 = Transmit test pattern enable |
| 4.25.1:0 | Test Pattern Select | 00 | R/W | 00 = High frequency test pattern<br>01 = Low frequency test pattern<br>10 = Mixed frequency test pattern<br>11 = Reserved |

## 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 2-57 shows the Configuration Vector bit definitions.

*Table 2-57:* **Configuration Vector Bit Definitions**

| Bits | Name | Description |
|---|---|---|
| 6:5 | Test Select[1:0] | Selects the test pattern. See bits 5.25.1:0 in Table 2-46, page 38. |
| 4 | Test Enable | Enables transmit test pattern generation.<br>See bit 5.25.2 in Table 2-46, page 38. |
| 3 | Reset RX Link Status | Sets the RX Link Status bit (status_vector[7]). See Table 2-58. This bit should be driven by a register on the same clock domain as the RXAUI core. |
| 2 | Reset Local Fault | Clears both TX Local Fault and RX Local Fault bits (status_vector[0] and status_vector[1]). See Table 2-58. This bit should be driven by a register on the same clock domain as the RXAUI core. |
| 1 | Power Down | Sets the device-specific transceivers into power down mode.<br>See bit 5.0.11 in Table 2-38, page 32. |
| 0 | Loopback | Sets serial loopback in the device-specific transceivers.<br>See bit 5.0.14 in Table 2-38, page 32. |

Table 2-58 shows the Status Vector bit definitions.

*Table 2-58:* **Status Vector Bit Definitions**

| Bits | Name | Description |
|---|---|---|
| 7 | RX Link Status | 1 if the Receiver link is up, otherwise 0. See bit 5.1.2 in Table 2-39, page 33. Latches Low.<br>Cleared by rising edge on configuration_vector[3]. |
| 6 | Alignment | 1 if the RXAUI receiver is aligned over all four logical XAUI lanes, otherwise 0. See bit 5.24.12 in Table 2-44, page 37.<br>This is also used to generate the align_status signal described in Table 2-59. |
| 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 2-44, page 37.<br>These four bits are also used to generate the sync_status[3:0] signal described in Table 2-59. |
| 1 | RX Local Fault | 1 if there is a fault in the receive path, otherwise 0. See bit 5.8.10 in Table 2-43, page 36. Latches High.<br>Cleared by rising edge on configuration_vector[2]. |
| 0 | TX Local Fault | 1 if there is a fault in the transmit path, otherwise 0. See bit 5.8.11 in Table 2-43, page 36. Latches High.<br>Cleared by rising edge on configuration_vector[2]. |

Bits[1:0] 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.

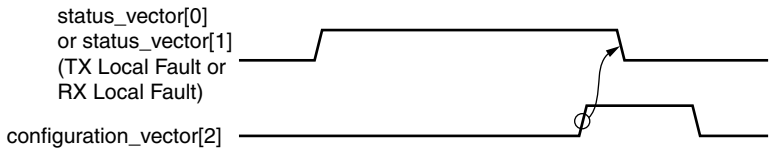


*Figure 2-38:* **Clearing the Local Fault Status Bits**

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.

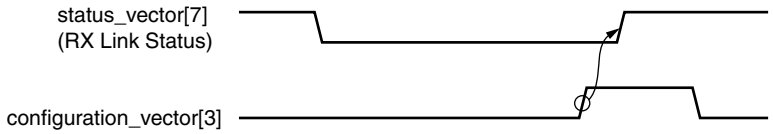


*Figure 2-39:* **Setting the RX Link Status Bit**

## Alignment and Synchronization Status Ports

In addition to the configuration and status interfaces described in the previous section, there is some information on the debug output port signalling the alignment and synchronization status of the receiver (Table 2-59).

*Table 2-59:* **Alignment Status and Synchronization Status Ports**

| Port Name | Description |
|---|---|
| debug[5] | 1 when the RXAUI receiver is aligned across all four XAUI logical lanes, 0 otherwise. |
| debug[4:1] | 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.

**IMPORTANT:** *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 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 XDC 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 for further information.

### Use Supported Design Flows

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

### 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 tool when the core is generated. See Chapter 4, Customizing and Generating the Core.

## Clocking

The clocking schemes in this section are illustrative only and might require customization for a specific application. See Table 2-8 for information on the clock ports.

These clocking schemes (with the exception of DCLK) can be included as part of the core if the **Include Common Clocking and Resets** GUI option is checked.

If this option is deselected, they appear in the “support” layer in the example design for reference.

### 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_GTE2 (7 series FPGAs) is used to feed the reference clocks for all transceivers. In addition, timing requirements are met if both transceivers are placed next to each other.

For details about transceiver clock distribution, see the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 3] and the *7 Series FPGAs GTP Transceivers User Guide* [Ref 4].

## Dune Networks RXAUI (7 Series FPGAs)

The clocking scheme for 7 series GTX transceivers is 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 encrypted HDL 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 3] and the *7 Series FPGAs GTP Transceivers User Guide* [Ref 4] for details about this clock.

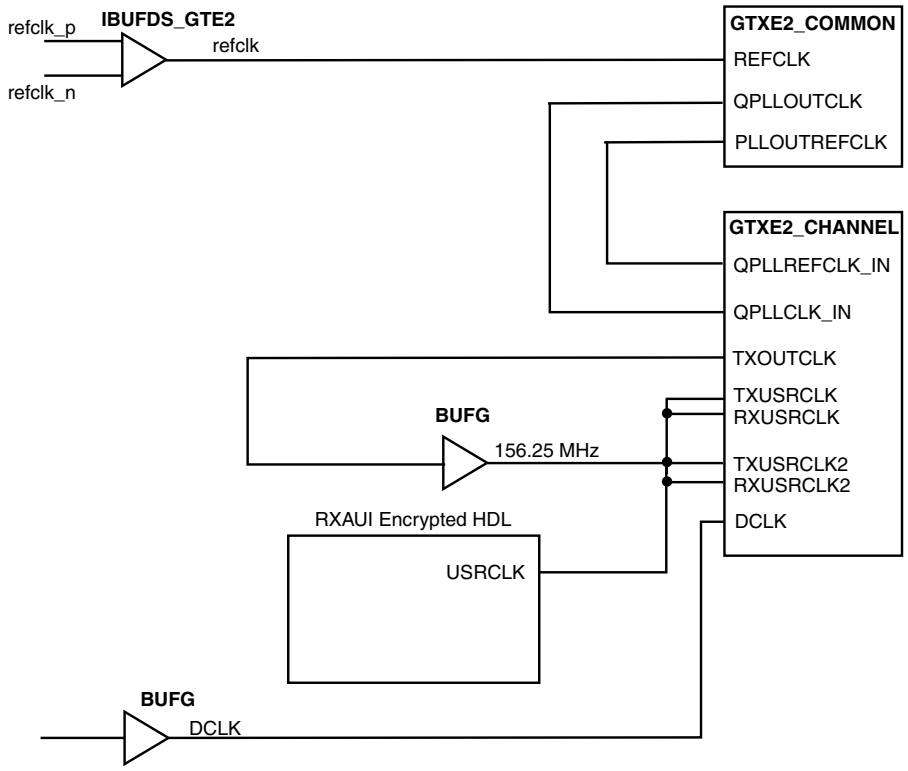


*Figure 3-1:* **Clock Scheme for Dune Networks RXAUI – 7 Series GTX Transceivers**

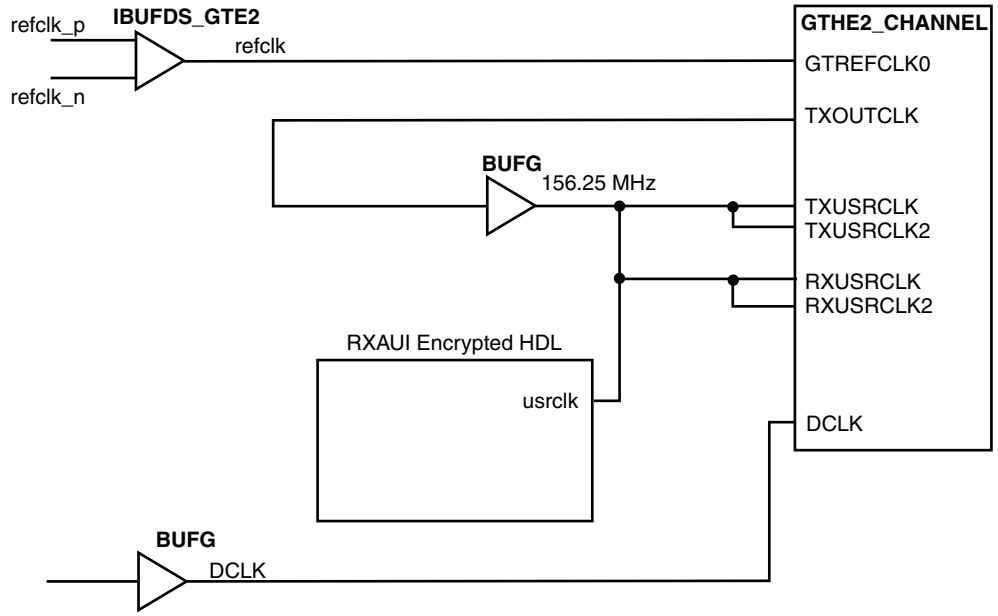


*Figure 3-2:* **Clock Scheme for Dune Networks RXAUI – 7 Series GTH Transceivers**

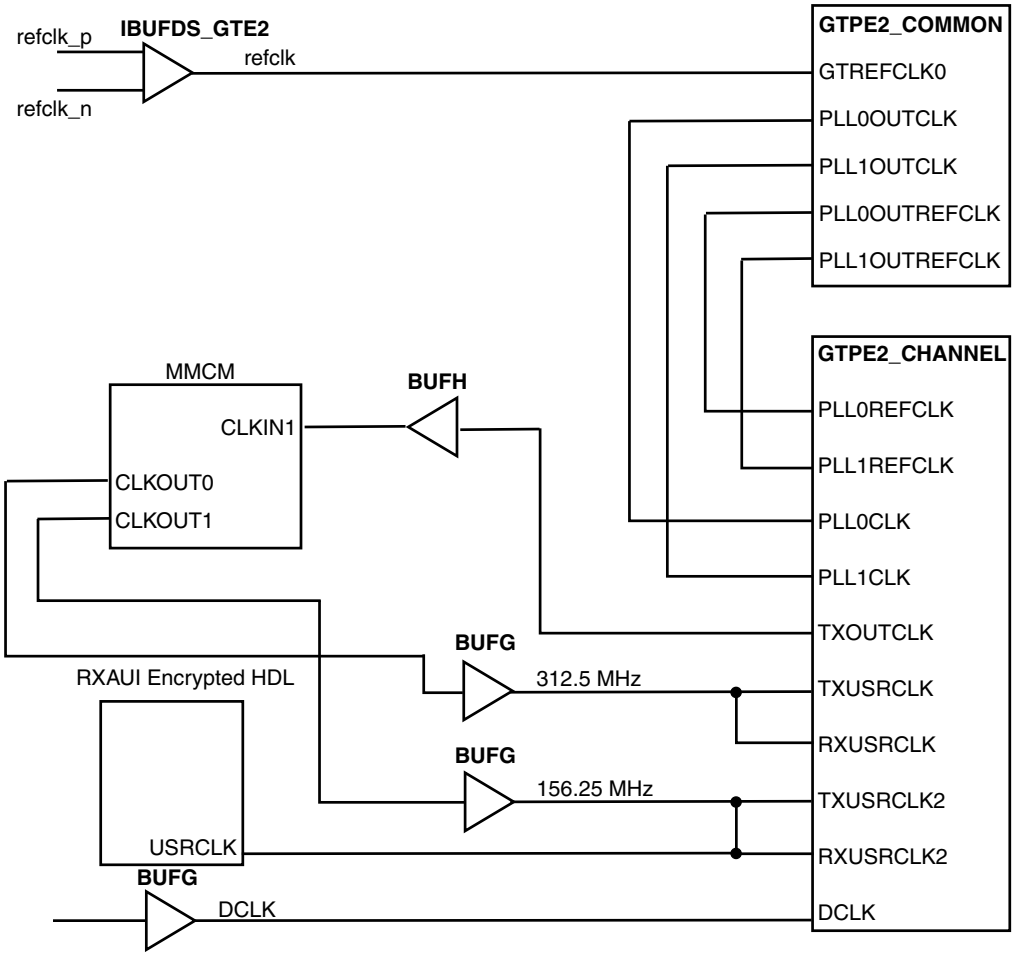


*Figure 3-3:* **Clock Scheme for Dune Networks RXAUI – 7 Series GTP Transceivers**

# Resets

See Table 2-8 for information on the reset ports. All register resets within the RXAUI core 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 component multiple times. Some clocking resources such as the transceiver "common" module can be shared between multiple core instances. Due to the transceiver phase alignment procedure, `clk156` must be generated from the core `txoutclk` port and is not shareable with another core instance. If it is desired to share the Common PLL clocks with another instance, then one core can be generated with the clocking option selected and one core without. The allows the common PLL signals to be connected between two cores. See the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 3] for details on sharing the reference clock and any limitations.

## Receiver Termination – Virtex-7 and Kintex-7 FPGAs

See the *7 Series FPGAs GTX/GTH Transceivers User Guide* [Ref 3] and the *7 Series FPGAs GTP Transceivers User Guide* [Ref 4].

# 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 3-4. The beginning of the data frame is shown by the presence of the Start character (the /S/ codegroup in lane 4 of Figure 3-4) 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 3-4). The Terminate character can occur in any lane; the remaining lanes are padded by XGMII idle characters.

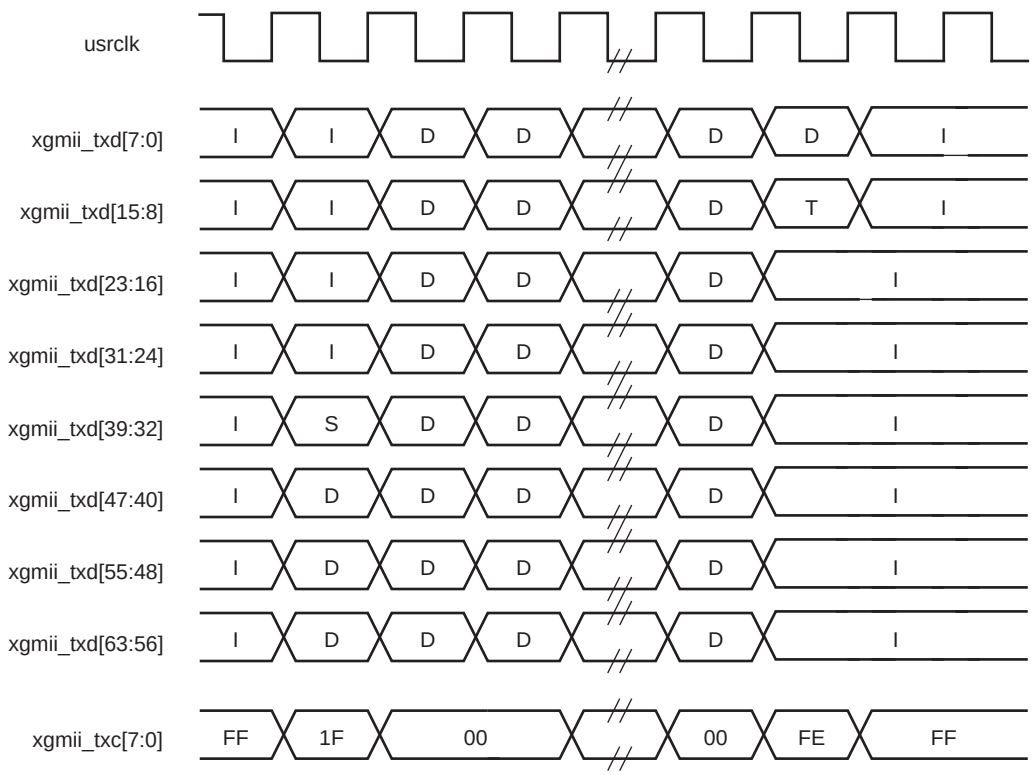


*Figure 3-4:* **Normal Frame Transmission Across the Internal 64-bit Client-Side I/F**

Figure 3-5 depicts a similar frame to that in Figure 3-4, 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.

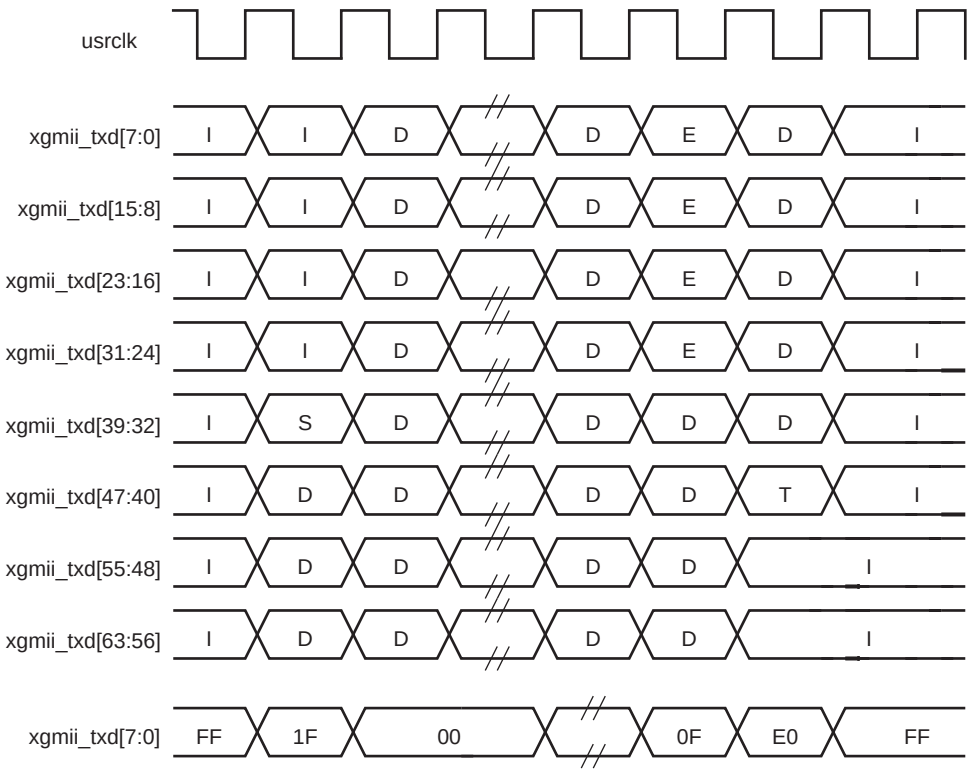


*Figure 3-5:* **Frame Transmission with Error Across Internal 64-bit Client-Side I/F**

# Interfacing to the Receive Client Interface

## Internal 64-bit Client-Side Interface

The timing of a normal inbound frame transfer is shown in Figure 3-6. 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.

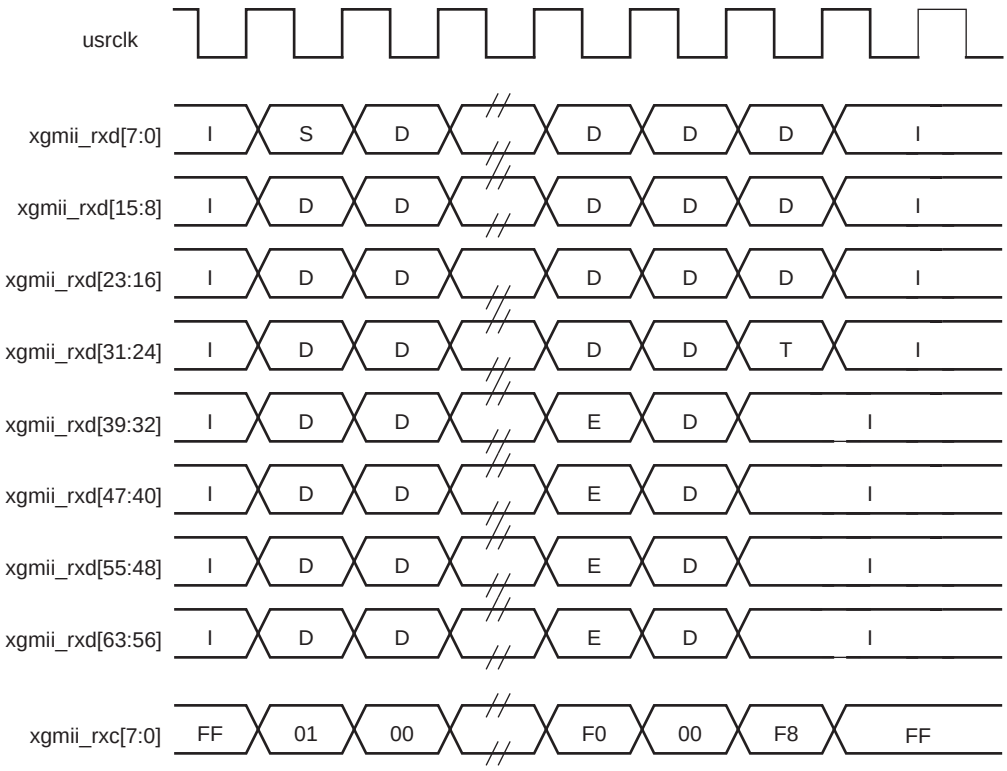


*Figure 3-6:* **Frame Reception Across the Internal 64-bit Client Interface**

Figure 3-7 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.

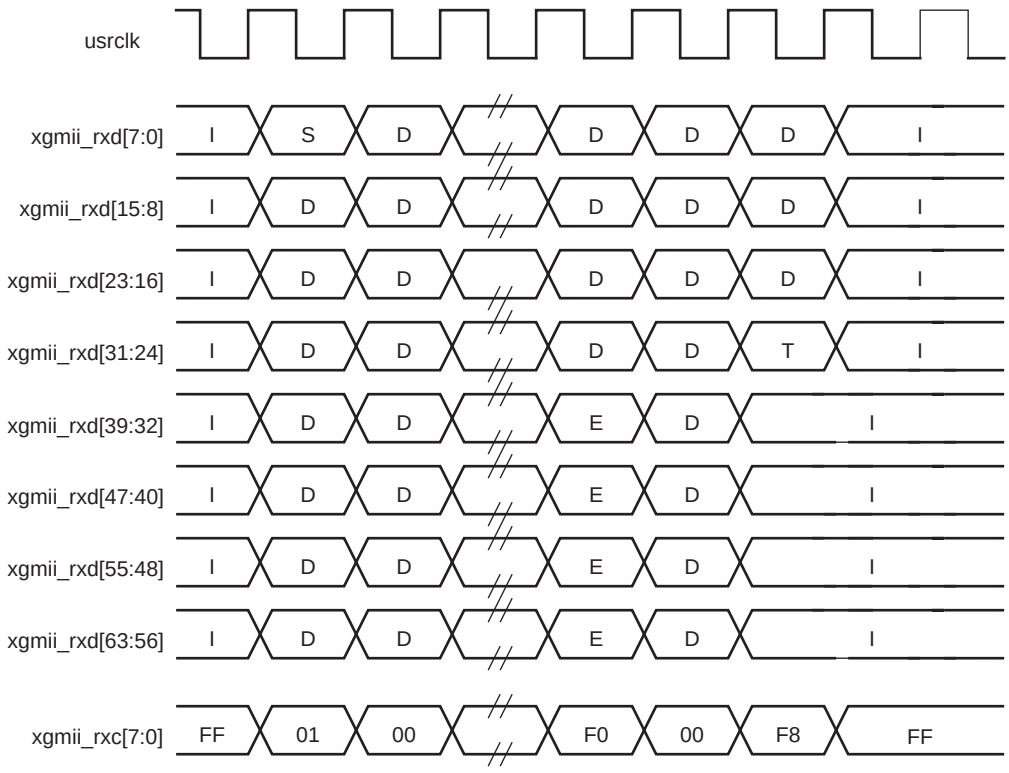


*Figure 3-7:* **Frame Reception with Error Across the Internal 64-bit Client Interface**

## 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 3-8 illustrates a typical system. All transactions are initiated by the STA entity. The RXAUI core implements an MMD.

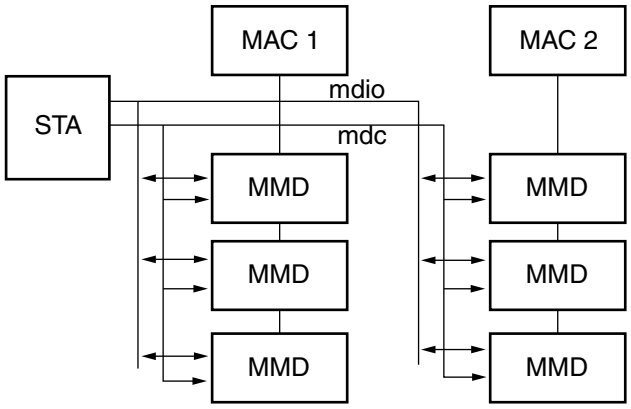


*Figure 3-8:* **A Typical MDIO-Managed System**

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 3-9 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).

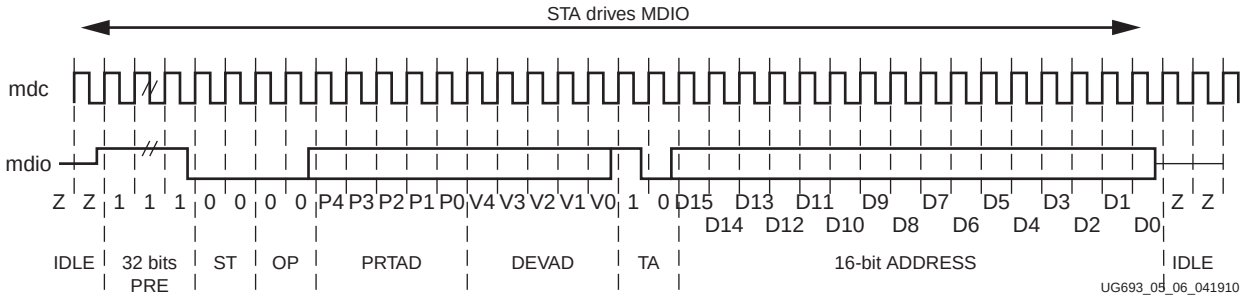


*Figure 3-9:* **MDIO Set Address Transaction**

### Write Transaction

Figure 3-10 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.

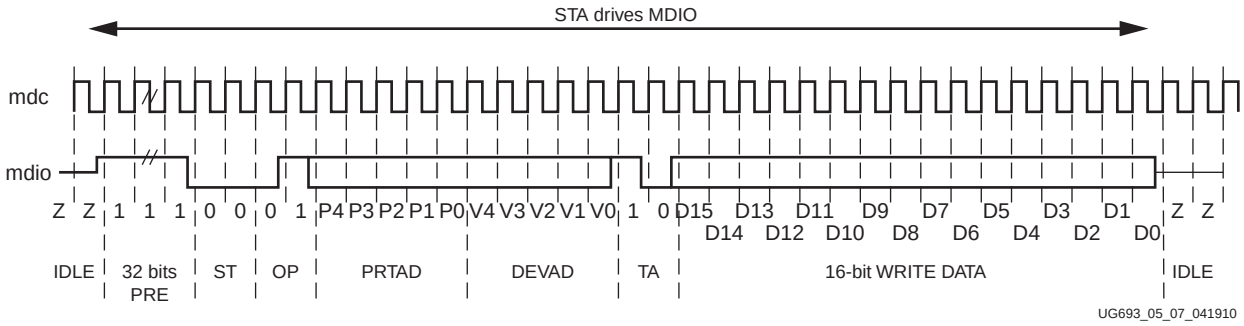


*Figure 3-10:* **MDIO Write Transaction**

### Read Transaction

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

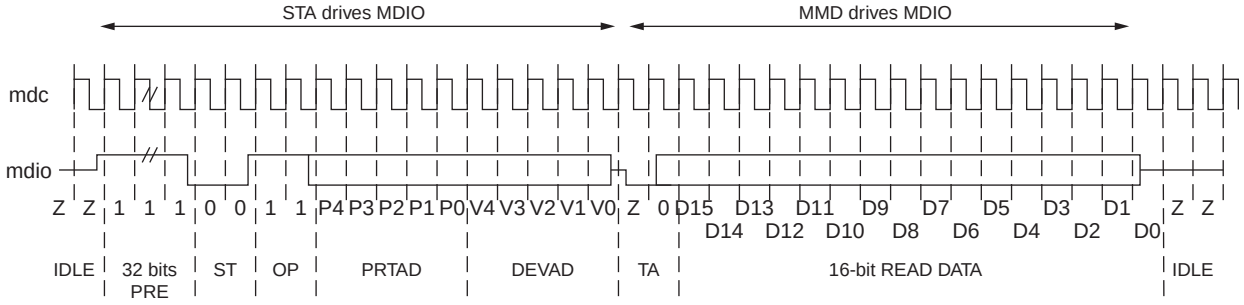


*Figure 3-11:* **MDIO Read Transaction**

### Post-Read-increment-address Transaction

Figure 3-12 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.

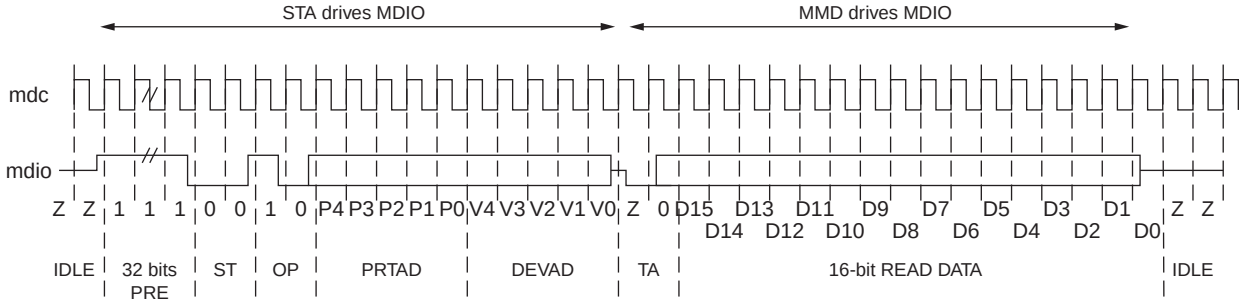


*Figure 3-12:* **MDIO Read-and-increment Transaction**

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

# 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.

## Vivado Integrated Design Environment (IDE)

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

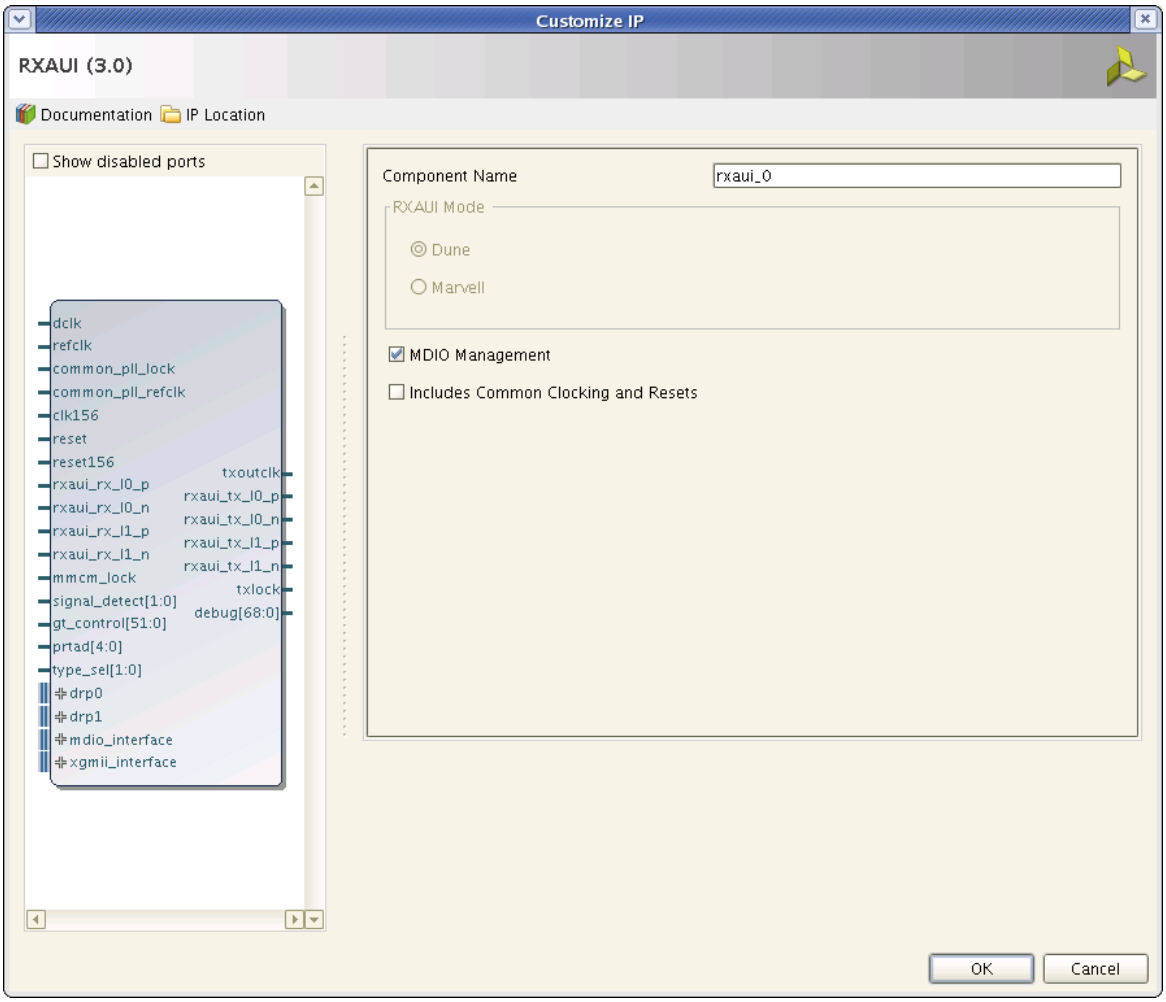


*Figure 4-1:* **RXAUI Main Screen**

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).

### 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.

### Common Clocking and Resets

Select this option to include the Common Transceiver PLL, clock buffers, and any clocking reset logic within the core.

Deselect this option if you require to use external clocking logic, wish to source the Common PLL clocks from another core, or use it externally. When this option is deselected, the clocking logic and resets are placed in the example design files.

## 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.

# 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. An XDC file is applied to the core instance. This XDC is available through the **IP Sources** view under **Synthesis**. No modification is required to this file.

There are some additional constraints required that are design specific such as transceiver location constraints. These are detailed in this chapter and an example of such constraints can be found in the example design. 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 additional 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.

## Clock Frequencies

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.

# 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 core XDC applies a `create_clock` constraint on the `TXOUTCLK` output port.

# Transceiver Placement

## 7 Series GTH Transceivers

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

## 7 Series GTX Transceivers

```
set_property LOC GTXE2_CHANNEL_X0Y0 [get_cells rxaui_support_i/rxaui_block_i/
gt0_<ComponentName>_gt_wrapper_i/gtxe2_i]
set_property LOC GTXE2_CHANNEL_X0Y1 [get_cells rxaui_support_i/rxaui_block_i/
gt0_<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_support_i/rxaui_block_i/
gt0_<ComponentName>_gt_wrapper_i/gtpe2_i]
set_property LOC GTPE2_CHANNEL_X1Y0 [get_cells rxaui_support_i/rxaui_block_i/
gt1_<ComponentName>_gt_wrapper_i/gtpe2_i]
```

# 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 clock and reset enabled configuration of the example design.

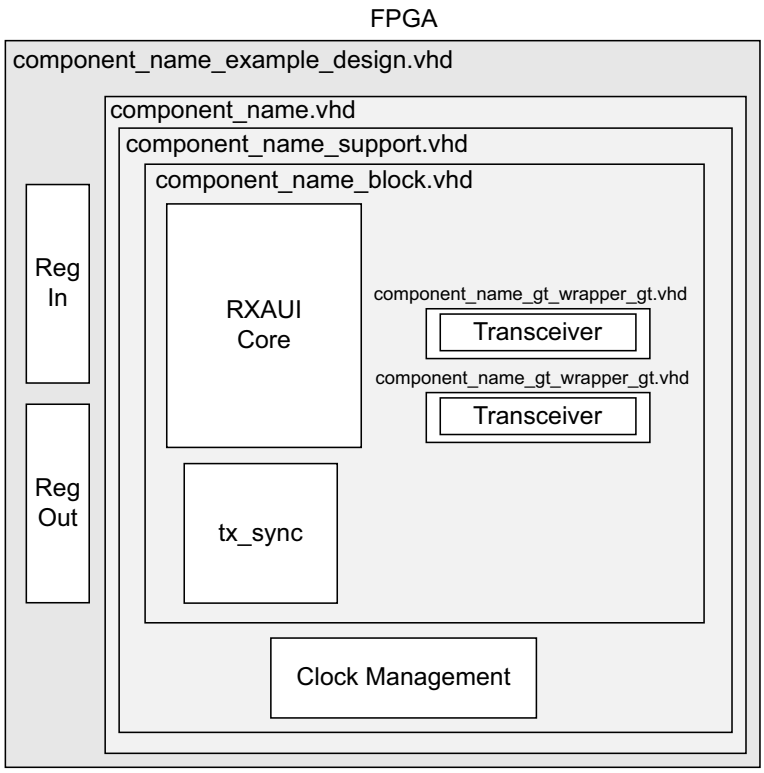


*Figure 6-1:* **RXAUI Example Design and Test Bench – Clock and Reset Enabled**

Figure 6-2 illustrates the clock and reset disabled configuration of the example design.

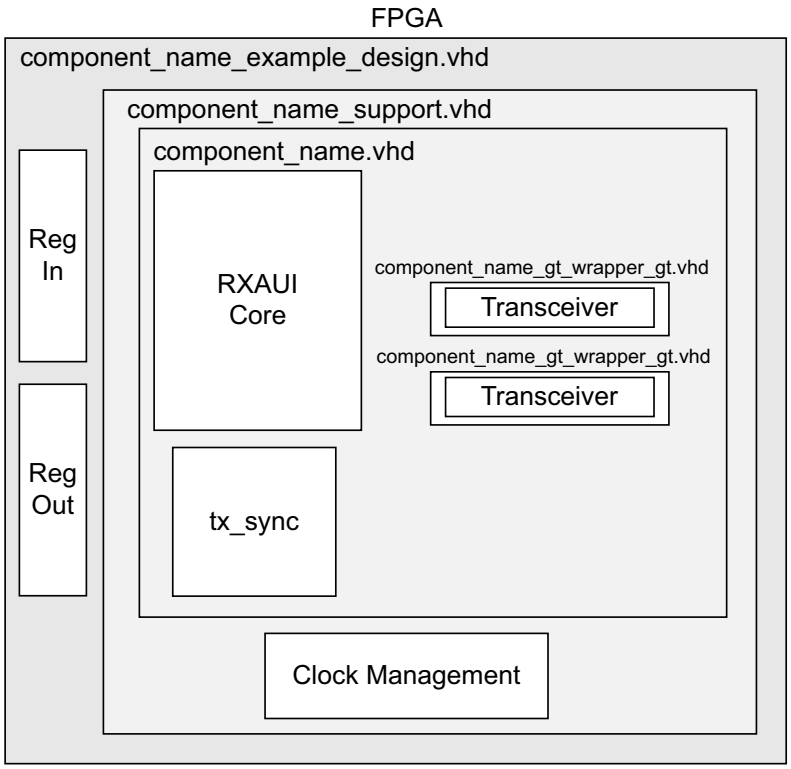


*Figure 6-2:* **RXAUI Example Design and Test Bench – Clock and Reset Disabled**

The RXAUI example design consists of the following:

- Clock buffers for DCLK
- Re-timing registers on the parallel data interface, both on inputs and outputs
- An instance of the support level module which contains the core, clocking modules, reset modules, and transceiver common module instance

**IMPORTANT:** *This module is only present in the example design if the* ***Common Clocking and Resets*** *option is deselected, otherwise it is part of the core logic.*

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

# Demonstration Test Bench

In Figure 6-3, 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.

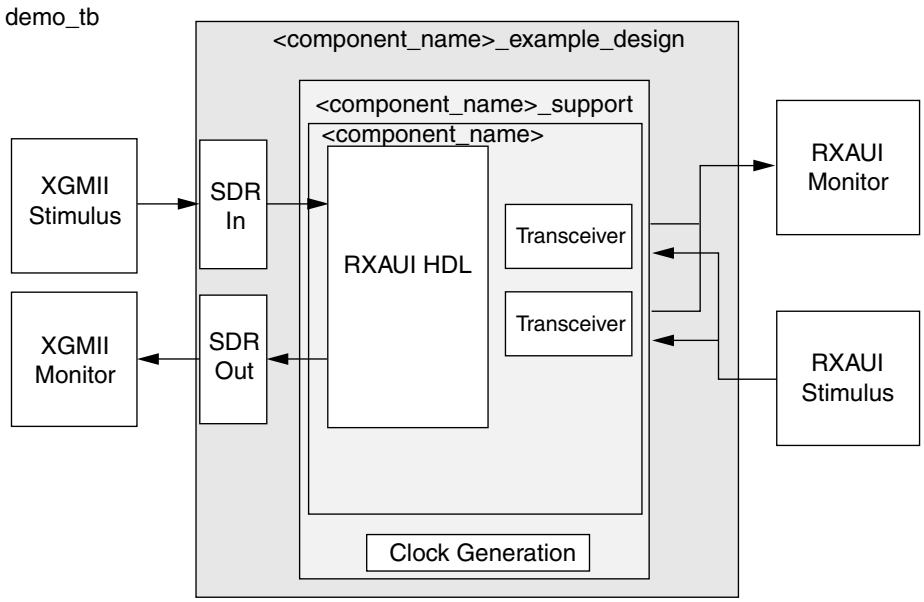


*Figure 6-3:* **Demonstration Test Bench for RXAUI**

# 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 9].

# 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].

*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* (UG911) [Ref 11].

# Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools. In addition, this appendix provides a step-by-step debugging process and a flow diagram to guide you through debugging the RXAUI core.

The following topics are included in this appendix:


- Finding Help on Xilinx.com
- Debug Tools
- Simulation Debug
- Hardware Debug


## Finding Help on Xilinx.com

To help in the design and debug process when using the RXAUI, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support WebCase.

### Documentation

This product guide is the main document associated with the RXAUI. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

### 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 listed below.

- Xilinx Ethernet IP Solution Center

## Known Issues

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are listed below, and can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

**Master Answer Record for the RXAUI**

AR 54249

## Contacting 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.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

# Debug Tools

There are many tools available to address RXAUI design issues. It is important to know which tools are useful for debugging various situations.

## Example Design

The RXAUI is delivered with an example design that can be synthesized, complete with functional test benches. Information about the example design can be found in *Chapter 6, Detailed Example Design*.

## Link Analyzers

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

- SMARTBITS
- IXIA

## Vivado Lab Tools

Vivado inserts logic analyzer and virtual I/O cores directly into your design. Vivado Lab Tools allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature represents the functionality in the Vivado IDE that is used for logic debugging and validation of a design running in Xilinx FPGA devices in hardware.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

The RXAUI core has useful signals that have the `MARK_DEBUG` attribute applied to them. This shows up as debug nets within Vivado.

For more information on how to use Vivado to debug your design, see *Vivado Design Suite User Guide, Programming and Debugging* (UG908) [Ref 10].

# Simulation Debug

The simulation debug flow for Questa® SIM is illustrated in Figure C-1. A similar approach can be used with other simulators.

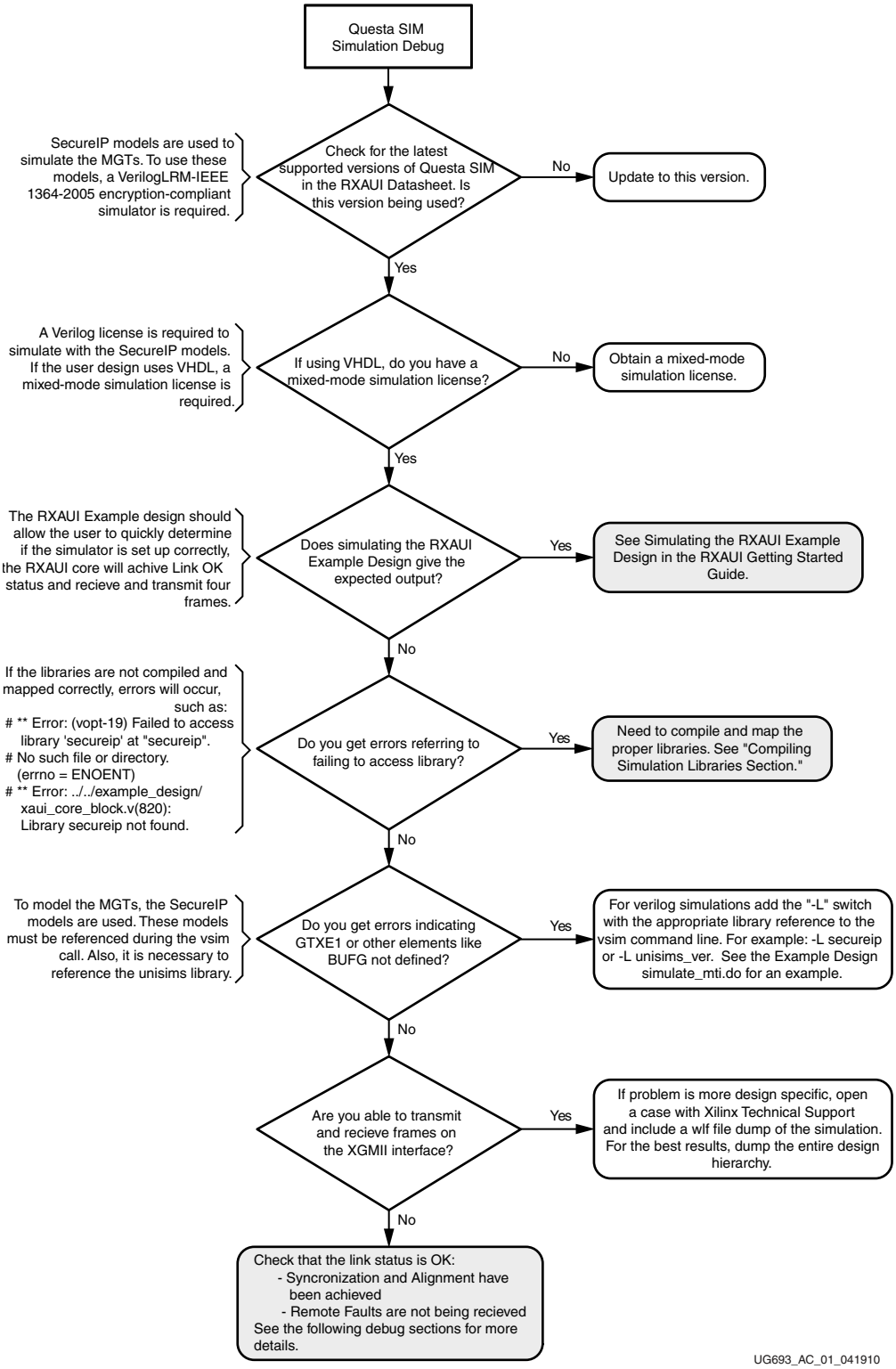


*Figure C-1:* **Questa SIM Debug Flow Diagram**

## Compiling Simulation Libraries

To run simulation with third-party simulators, it is necessary to compile the Xilinx Simulation Libraries. For full details on how to perform this, see Appendix B of the *Vivado Design Suite User Guide, Logic Simulation* (UG900) [Ref 8].

## 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 Vivado Analyzer tool is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado Analyzer tool for debugging the specific problems. Many of these common issues 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 Vivado Lab Tools

- The RXAUI core has the MARK_DEBUG attribute applied to the signal for debugging and checking the status of the RXAUI core and transceivers for easy access to adding these to the Vivado logic analyzer.
- A debug port is also provided so it can connect to external logic for debug (for example, processor monitoring). This port contains transceiver and core debug information.
- XGMII signals and signals between RXAUI core and the transceiver can be added to monitor data transmitted and received. See Table 2-4, page 9 and Table 2-8, page 11 for a list of signal names.
- Status signals added to check status of link: STATUS_VECTOR[7:0], ALIGN_STATUS, SYNC_STATUS, and Debug port Bits[5:0].
- 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.

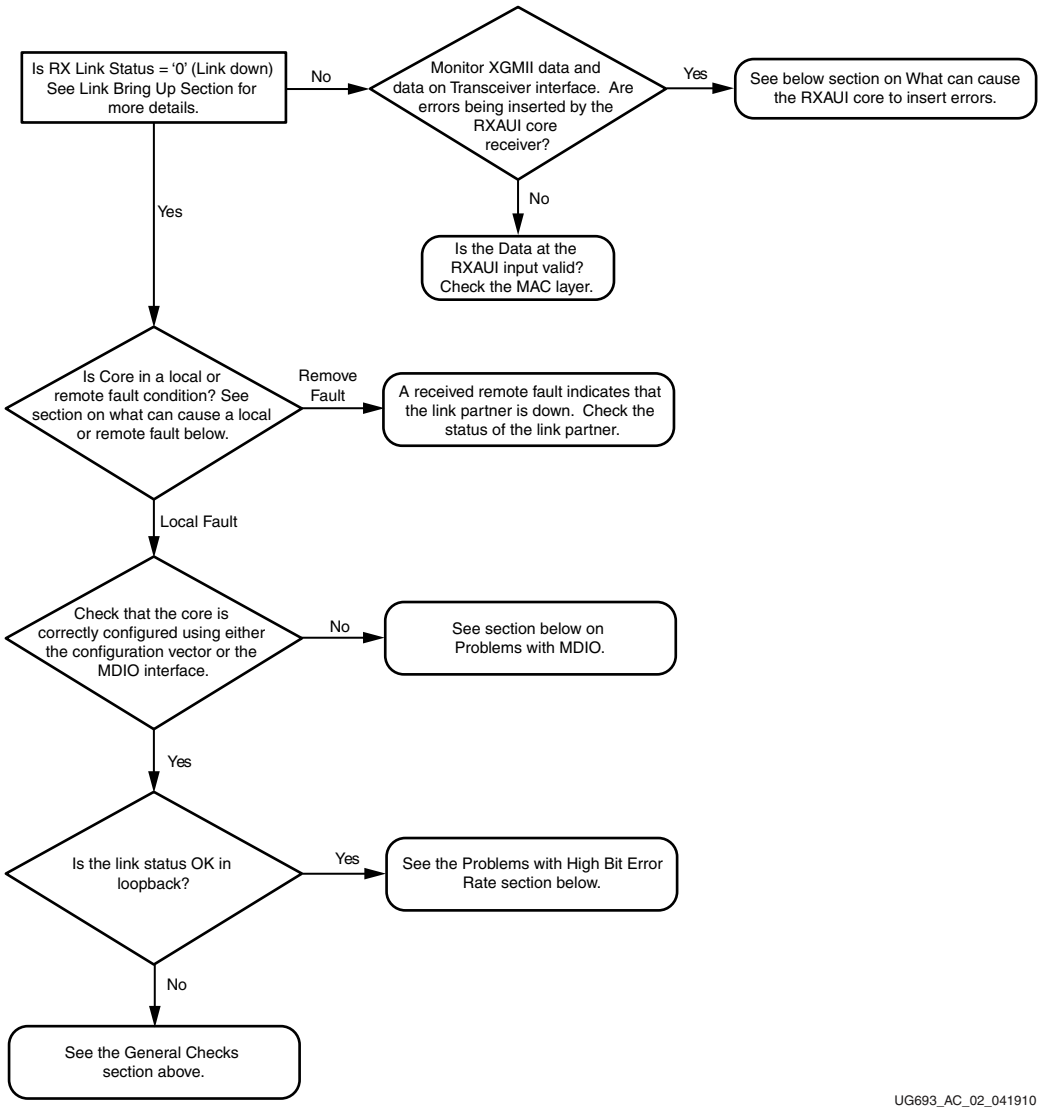


*Figure C-2:* **Flow Diagram for Debugging Problems with Data Reception or Transmission**

## 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`
- 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

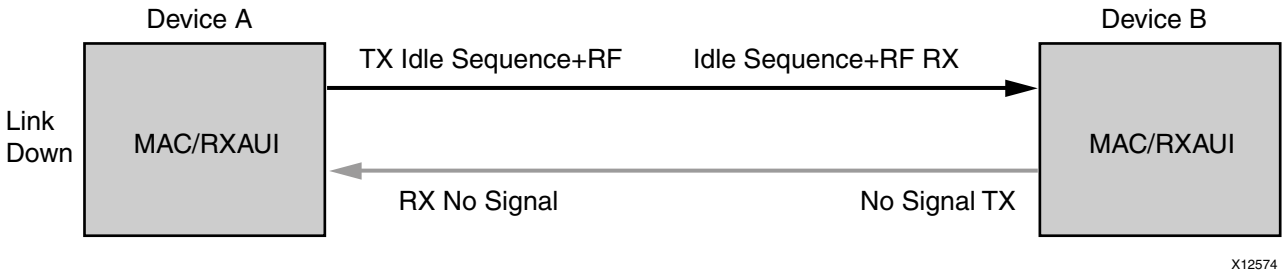


*Figure C-3:* **Device A Powered Up, but Device B Powered Down**

## 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.

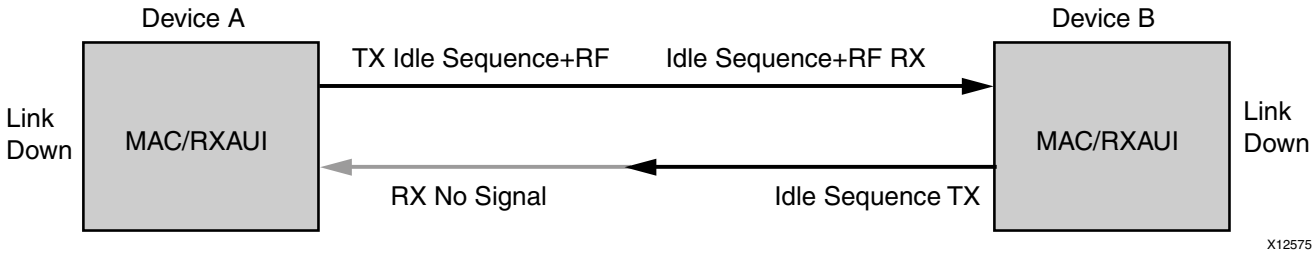


*Figure C-4:* **Device B Powers Up and Resets**

## 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.

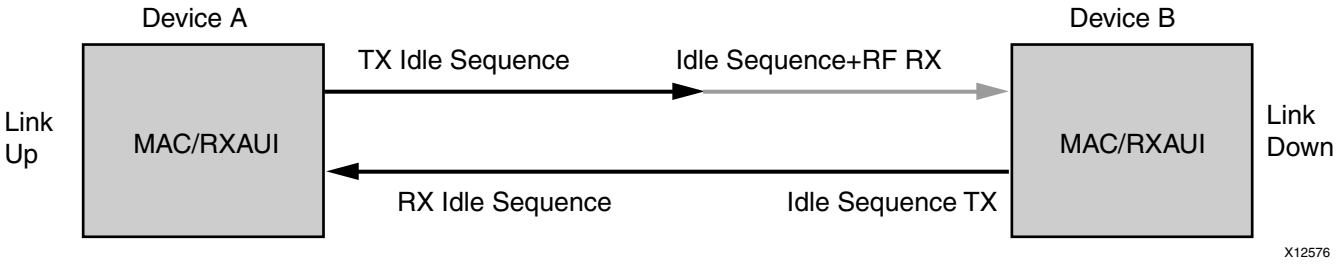


*Figure C-5:* **Device A Receives Idle Sequence**

### 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.

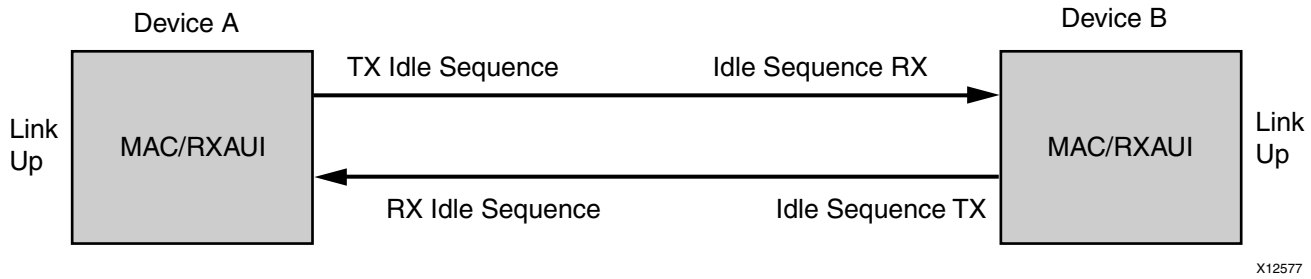


*Figure C-6:* **Normal Operation**

## What Can Cause Synchronization and Alignment to Fail?

Synchronization (SYNC_STATUS) 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 four 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 "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 Terminate control character and all Idles up to and immediately preceding the following frame 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 Bit Error Rate (BER) 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 Vivado Analyzer 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, page 57 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 Vivado Analyzer 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 Vivado analyzer 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. *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)
4. *7 Series FPGAs GTP Transceivers User Guide* (UG482)
5. 7 *Series FPGAs Configuration User Guide* (UG470)
6. Vivado™ Design Suite user documentation
7. *Vivado Design Suite User Guide, Designing with IP* (UG896)
8. *Vivado Design Suite User Guide, Logic Simulation* (UG900)
9. *Vivado Design Suite User Guide, Implementation* (UG904)
10. *Vivado Design Suite User Guide, Programming and Debugging* (UG908)
11. *Vivado Design Suite Migration Methodology Guide* (UG911)

# Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 10/16/2012 | 1.0 | Initial Xilinx release. This Product Guide is derived from DS740 and UG693. Vivado Design Suite and Artix-7 support added. |
| 03/20/2013 | 1.1 | • Updated to v3.0 for Vivado Design Suite only and removed ISE.<br>• Updated Fig. 1-2 Implementation of Dune Networks RXAUI Core.<br>• Updated Table 2-5 Transceiver Interface Ports.<br>• Updated Table 2-7 Configuration and Status Ports.<br>• Added Tables 2-11 Clock and Resets Option Selected to 2-13 GTP Clocking Ports.<br>• Updated SIGNAL_DETECT descriptions in Table 2-25 10G PMD Signal Receive OK Register Bit Definitions.<br>• Added new Clocking and Reset Signals and Module to Debug Interface sections.<br>• Updated Alignment Status and Synchronization Status Ports table.<br>• Updated Fig. 3-1 7 Series GTX Transceivers to Fig. 3-3 7 Series GTP Transceivers.<br>• Updated Fig. 4-1 RXAUI Main Screen.<br>• Updated Constraining the Core section.<br>• Updated Fig. 6-1 RXAUI Example Design and Test Bench and added Fig. 6-2.<br>• Updated to new Debug section.<br>• Updated to Questa SIM. |

# Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.

© Copyright 2012–2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.