LogiCORE IP Quad Serial Gigabit Media Independent v1.3

Product Guide

PG029 July 25, 2012

XILINX®

Table of Contents

SECTION I: SUMMARY

IP Facts

Chapter 1: Overview

System Overview 7
Feature Summary. 10
System Requirements 11
Applications 11
Licensing and Ordering Information 13

Chapter 2: Product Specification

Standards Compliance 14
Performance. 14
Resource Utilization. 15
Port Descriptions 17
Register Space 30

Chapter 3: Designing with the Core

Design Guidelines. 56
Clocking. 59
Resets 59

Chapter 4: Using the Client Side GMII/MII Datapath

Using the Core Netlist Client-Side GMII/MII 61
Additional Client-Side QSGMII Adaptation Logic Provided in the Example Design. 65

Chapter 5: Using the Transceiver

Transceiver Logic 73
Clock Sharing Across Multiple Cores with Transceivers 80

SECTION II: VIVADO DESIGN SUITE

Chapter 6: Customizing and Generating the Core

Graphical User Interface 87
Output Generation. 91

Chapter 7: Constraining the Core

Device, Package, and Speed Grade Selections. 92
I/O Location Constraints 92
Placement Constraints. 92
Transceiver Placement 93
Constraints When Using External GMII/MII. 97

Chapter 8: Detailed Example Design

Example Design 101
Demonstration Test Bench 106

SECTION III: ISE DESIGN SUITE

Chapter 9: Customizing and Generating the Core

GUI 112
Parameter Values in the XCO File. 114
Output Generation. 115

Chapter 10: Constraining the Core

Device, Package, and Speed Grade Selections. 116
I/O Location Constraints 116
Placement Constraints. 116
Transceiver Placement 117
Constraints When Using External GMII/MII. 121

Chapter 11: Detailed Example Design

Directory Structure 126
Directory and File Contents. 127
Example Design 131
Demonstration Test Bench 136
Implementation 141
Functional Simulation 142

SECTION IV: APPENDICES

Appendix A: Verification, Compliance, and Interoperability

Simulation 144
Hardware Testing 144
Compliance Testing 144

Appendix B: Migrating

Appendix C: Implementing External GMII/MII

External GMII Transmitter Logic (Virtex-7, Kintex-7, and Artix-7 Devices) 146
External MII Transmitter Logic (Virtex-7, Kintex-7, and Artix-7 Devices). 148
External GMII/MII Receiver Logic 149

Appendix D: Debugging

General Checks 150
Problems with the MDIO 150
Problems with Data Reception or Transmission 150
Problems with Auto-Negotiation 151
Problems in Obtaining a Link (Auto-Negotiation Disabled) 152
Problems with a High Bit Error Rate 153

Appendix E: Additional Resources

Xilinx Resources 155
Solution Centers 155
References 155
Specifications 156
Technical Support 156
Revision History 156
Notice of Disclaimer 157

SECTION I: SUMMARY

IP Facts

Overview

Product Specification

Designing with the Core

Using the Client Side GMII/MII Datapath

Using the Transceiver

Introduction

The Quad Serial Gigabit Media Independent Interface (QSGMII) core provides a flexible solution for combining four Serial Gigabit Media Independent Interfaces (SGMII) into a single 5 Gigabits per second (Gb/s) Interface, to significantly reduce the number of Input Outputs (I/Os). This core supports Cisco QSGMII specification Version 1.2 (EDCS-540123)

Features

- Integrated transceiver interface using a Virtex®-7 and Kintex™-7 FPGA GTX transceiver
- Integrated transceiver interface using Virtex-7 FPGA GTH transceiver
- Integrated transceiver interface using Artix™-7 FPGA GTP transceiver
- Implements SGMII Adaptation to support 10/100/1000 operation for each port
- Transmitters of all ports transmit only /I1/ Idle ordered set
- Lane alignment based on K28.1 character detection
- Implements QSGMII K28.5 swapper on Port 0 transmit path
- Implements QSGMII K28.1 swapper on Port 0 receive path
- Implements receive link synchronization state machine
- Programmable Decoder running disparity checking for each port
- Per port programmable Auto-Negotiation Link timer

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family (1)	Virtex-7, Kintex-7, and Artix-7
Supported User Interfaces	GMII/MII
Resources	See Table 2-1 and Table 2-3.
Provided with Core	
Design Files	ISE®: Native Generic Circuit (NGC) Netlist Vivado™: Encrypted RTL
Example Design	VHDL and Verilog
Test Bench	Demonstration Test Bench in VHDL and Verilog
Constraints File	ISE®: User Constraint Files (UCF) Vivado™: Xilinx Design Constraints (XDC)
Simulation Model	Verilog and VHDL
Supported S/W Drivers	NA
Tested Design Flows (2)	
Design Entry	ISE Design Suite 14.2 Vivado Design Suite 2012.2 (3)
Simulation (4)	Mentor Graphics Modelsim Cadence Incisive Enterprise Simulator (IES) Synopsys VCS and VCS MX
Synthesis	Xilinx Synthesis Technology (XST) Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

1. For a complete listing of supported devices, see the release notes for this core.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
3. Supports only 7 series devices
4. Virtex-7, Kintex-7, and Artix-7 device designs incorporating a device-specific transceiver require a Verilog LRM-IEEE 1364-2005 encryption-compliant simulator. For VHDL simulation, a mixed HDL license is required.

Chapter 1

Overview

QSGMII is designed to reduce significantly the number of signals that are needed between multiport 10/100/1000 PHYs and Ethernet MAC. QSGMII needs two data signals, each operating at 5 Gb/s, to connect four instances of PHYs and Ethernet MAC.

System Overview

QSGMII core provides the functionality to implement the sublayers as specified by the Cisco QSGMII specification.

The core interfaces to a device-specific transceiver. The transceiver provides some of the PCS functionality, such as 8B/10B encoding/decoding, Physical Medium Attachment (PMA) Serializer/Deserializer (SERDES), and clock recovery. Figure 1-1 illustrates the remaining PCS sublayer functionality and also shows the major functional blocks of the core.

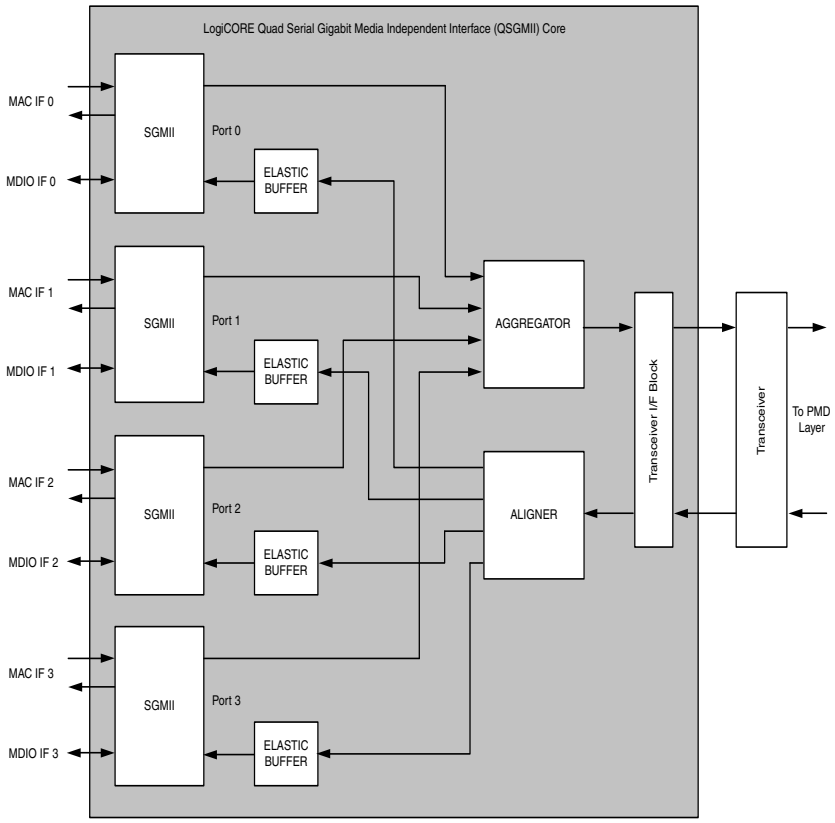

Figure 1-1: **QSGMII System Overview**

SGMII

Figure 1-2 illustrates the sub-blocks of the SGMII module.

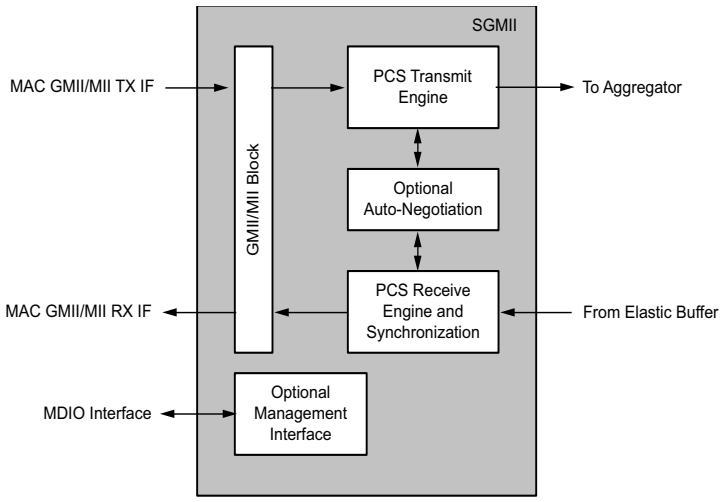

Figure 1-2: **Functional Diagram of SGMII Block**

GMII/MII Block

A client-side GMII is provided with the core, which can be used as an internal interface for connection to an embedded Media Access Controller (MAC) or other custom logic in MAC mode. In PHY mode the GMII/MII can be routed to device Input Output Blocks (IOBs) to provide an external (off-device) GMII/MII.

Virtex®-7 devices support GMII at 3.3 V or lower only in certain parts and packages. See the Virtex-7 device documentation. Kintex™-7 and Artix™-7 devices support GMII at 3.3 V or lower.

PCS Transmit Engine

The Physical Coding Sublayer (PCS) transmit engine converts the GMII data octets into a sequence of ordered sets by implementing the state diagrams of *IEEE 802.3-2008* (Figures 36-5 and 36-6). The transmit engine transmits only /I1/ characters instead of /I2/, as described in the QSGMII specification.

PCS Receive Engine and Synchronization

The synchronization process implements the state diagram of *IEEE 802.3-2008* (Figure 36-9). The PCS receive engine converts the sequence of ordered sets to GMII data octets by implementing the state diagrams of *IEEE 802.3-2008* (figures 36-7a and 36-7b). This module can be programmed to optionally consider disparity. Disparity checking is disabled by default.

Optional Auto-Negotiation Block

Clause 37 in the *IEEE 802.3-2008* specification describes the Auto-Negotiation function that allows a device to advertise the modes of operation that it supports to a device at the remote end of a link segment (link partner), and to detect corresponding operational modes that the link partner might be advertising.

Auto-Negotiation is controlled and monitored through the PCS Management Registers.

Optional PCS Management Registers

Configuration and status of the core, including access to and from the optional Auto-Negotiation function, uses the Management Registers defined in clause 37 of the *IEEE 802.3-2008* specification. These registers are accessed through the serial Management Data Input/Output Interface (MDIO), defined in clause 22 of the *IEEE 802.3-2008* specification, as if it were an externally connected PHY.

An additional configuration vector and status signal interface is provided to configure Base Control Register (Register 0) and Auto-Negotiation Ability Advertisement Register (Register 4).

Aggregator

The Aggregator implements a portion of a modified transmit path diagram (Figure 1 of the QSGMII v1.2 specification). This module receives data and control from each instance of the SGMII module which is aggregated to 32-bit data and 4-bit control and transferred to Transceiver Interface block. The Aggregator also incorporates the K28.5 swapping function on port 0 that assists in port matching at the peer receiver end.

Aligner

The Aligner receives 32 bits of data from the transceiver interface. Port 0 data can be received on any lane, so a search for the K28.1 character is done on all the lanes to start lane alignment. After a match for K28.1 is found in the octet boundary in the 32-bit data, that octet boundary becomes the start of arbitration and the octet assigned to port 0. The next octet is assigned to port 1 and so on. This module also swaps any K28.1 character received on port 0 with the K28.5 character.

Transceiver Interface Block

The Transceiver Interface Block enables the core to connect to a Virtex-7. Kintex-7 or Artix-7 Field Programmable Gate Array (FPGA) serial transceiver.

Elastic Buffer

An Elastic Buffer is instantiated on each port to perform clock correction. The clock correction involves additions and removal of /I1/ characters if disparity is ignored or /I2/ if the disparity is considered. This buffer is 128 locations deep.

Feature Summary

- The core has two modes of operation.
  - Media Access Controller (MAC) mode to connect to a customized MAC or Xilinx Tri-Mode Ethernet MAC LogiCORE™ IP operating in Internal Mode. See QSGMII MAC in the Applications section.
  - Physical-side interface (PHY) mode to connect to an external PHY through Gigabit Media Independent Interface/Media Independent Interface (GMII/MII). See QSGMII PHY in the Application section.
- Each port configured and monitored through independent a serial Management Data Input/Output (MDIO) Interface, which can optionally be omitted from the core. An additional configuration vector interface is provided that can be used to program registers 0 and 4 over and above the MDIO interface.

- Supports Auto-Negotiation according to *IEEE 802.3-2008* Clause 37 on each port for information exchange with a link partner, which can optionally be omitted from the core.
- Integrated transceiver interface using a Virtex-7 and Kintex-7 FPGA GTX transceiver. Integrated transceiver interface using Virtex-7 FPGA GTH transceiver.
- Integrated transceiver interface using Artix-7 FPGA GTP transceiver.
- Implements SGMII Adaptation to support 10/100/1000 operation for each port. Each port can be programmed to operate at a speed independent of other ports.
- Transmitters of all ports transmit only /I1/ Idle ordered set.
- Lane alignment based on K28.1 character detection.
- Implements QSGMII K28.5 swapper on Port 0 transmit path.
- Implements QSGMII K28.1 swapper on Port 0 receive path.
- Implements receive link synchronization state machine.
- Programmable Decoder running disparity checking for each port.
- Per port programmable Auto-Negotiation Link timer.

System Requirements

For a list of System Requirements, see the Xilinx Design Tools: Release Notes Guide.

Applications

Typical applications for the QSGMII core include the following:

- QSGMII MAC
- QSGMII PHY

QSGMII MAC

Figure 1-3 illustrates a typical application for the QSGMII core when operating in MAC mode using a device-specific transceiver to provide the serial interface.

- The device-specific transceiver is connected to an external off-the-shelf QSGMII PHY (This can be a device that supports conversion of QSGMII to 10BASE-T, 100BASE-T, or 1000BASE-T.)
- The GMII interfaces of the QGMII core are connected to multiple instances of an embedded Ethernet MAC, for example, the Xilinx Tri-Mode Ethernet MAC core.

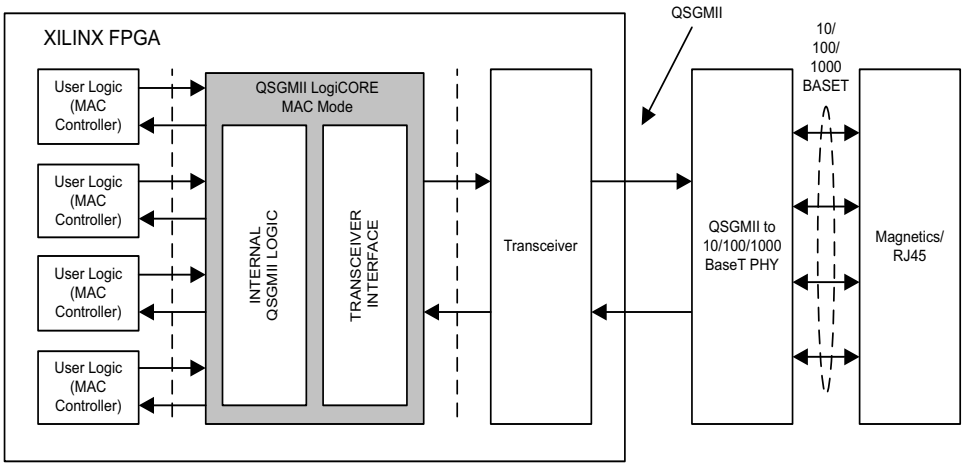

Figure 1-3: **Typical Application of QSGMII in MAC Mode**

QSGMII PHY

Figure 1-4 illustrates a typical application for the QSGMII core when operating in PHY mode, using a device-specific transceiver to provide the serial interface.

- The device-specific transceiver is connected to an external off-the-shelf Ethernet MAC device that also supports QSGMII. (This can be multiple instances of tri-mode MAC providing 10/100/1000 Mb/s operation, for example, the Xilinx Tri-Mode Ethernet MAC core connected to QSGMII core in MAC mode.)
- The GMII/MII interface of QSGMII core is connected to a tri-mode PHY providing 10BASE-T, 100BASE-T, and 1000BASE-T operation.

.

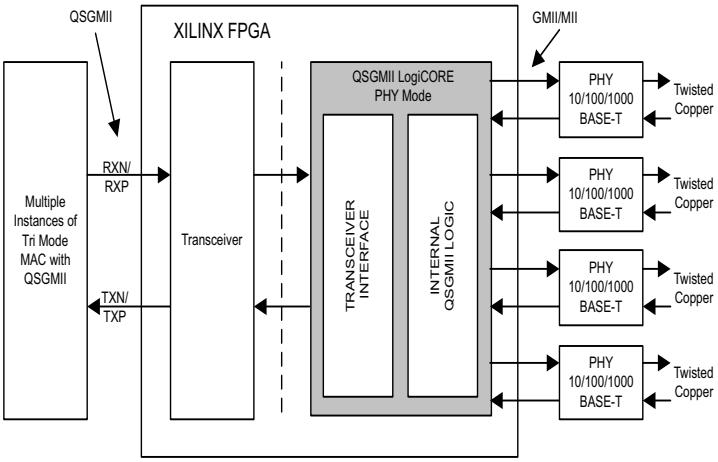

Figure 1-4: **Typical Application of QSGMII in PHY Mode**

Licensing and Ordering Information

This Xilinx LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado™ Design Suite and ISE® Design Suite 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 Compliance

- *Ethernet Standard 802.3-2008* Clauses 22, 35, 36 and 38
- Cisco *Serial GMII Specification* Revision 1.7 (SGMII)
- Cisco *Quad SGMII Specification* Revision 1.2 (QSGMII)

Performance

Latency

These measurements are for the core only; they do not include the latency through the Virtex®-7.Kintex™-7 or Artix™-7 FPGA transceiver, or the Transmitter Elastic Buffer added in the example design.

Transmit Path Latency

As measured from a data octet input into `gmii_txd[7:0]` of the transmitter side GMII of SGMII on port 0 (until that data appears on `txdata[7:0]` on the serial transceiver interface), the latency through the core in the transmit direction is five clock periods of `userclk2`.

Receive Path Latency

Receive Path Latency is variable due to the presence of an elastic buffer on each lane for clock compensation; therefore, the latency is measured from the output of the elastic buffer until the octet appears on the receiver side GMII. As measured from a data octet output from the elastic buffer until that data appears on `gmii_rxd[7:0]` of the receiver side GMII of port 0, the latency through the core in the receive direction is six clock periods of `userclk2`.

Throughput

QSGMII Interface operates at a full line rate of 5 Gb/s.

Resource Utilization

Approximate resource utilization of the core for different devices is listed in the following tables. Utilization figures are obtained by implementing the block-level wrapper for the core. This wrapper is part of the example design and connects the core to the selected physical interface.

BUFG usage does not consider multiple instantiations of the core, where clock resources can often be shared. BUFG usage does not include the reference clock required for IDELAYCTRL. This clock source can be shared across the entire device and is not core specific.

Virtex-7 Devices

Table 2-1 provides approximate utilization figures for various core options when a single instance of the core is instantiated in a Virtex-7 device.

Table 2-1: **Resource Utilization for Virtex-7 Devices**

Parameter Values			Device Resources				
Mode	**MDIO Interface**	**Auto-Negotiation**	**Slices**	**FFs**	**LUTs**	**LUTRAM**	**BUFGs**
MAC Mode	Yes	Yes	1419	2509	2340	464	2
	Yes	No	1055	1869	1708	460	2
	No	Yes	1179	2217	2017	432	2
	No	No	807	1629	1419	428	2
PHY GMII Mode	Yes	Yes	1421	2573	2361	464	2
	Yes	No	1042	1873	1716	460	2
	No	Yes	1178	2249	1986	432	2
	No	No	852	1629	1389	428	2
PHY MII Mode	Yes	Yes	1423	2577	2392	464	2
	Yes	No	1047	1869	1715	460	2
	No	Yes	1195	2249	1973	432	2
	No	No	806	1629	1432	428	2

Note: Resource Utilization numbers in PHY MII mode are more than PHY GMII mode due to extra logic required for 4-bit MII to 8-bit internal conversion and nibble alignment to proper octet boundaries.

Kintex-7 Devices

Table 2-3 provides approximate utilization figures for various core options when a single instance of the core is instantiated in a Kintex-7 device.

Table 2-2: **Resource Utilization for Kintex-7 Devices**

Parameter Values			Device Resources				
Mode	**MDIO Interface**	**Auto-Negotiation**	**Slices**	**FFs**	**LUTs**	**LUTRAM**	**BUFGs**
MAC Mode	Yes	Yes	1410	2509	2399	464	2
	Yes	No	996	1869	1754	460	2
	No	Yes	1174	2217	2036	432	2
	No	No	845	1625	1391	428	2
PHY GMII Mode	Yes	Yes	1401	2577	2442	464	2
	Yes	No	1025	1873	1743	460	2
	No	Yes	1178	2249	2017	432	2
	No	No	842	1629	1442	428	2
PHY MII Mode	Yes	Yes	1418	2573	2402	464	2
	Yes	No	1075	1873	1677	460	2
	No	Yes	1199	2249	1991	432	2
	No	No	833	1629	1436	428	2

Note: Resource Utilization numbers in PHY MII mode are more than PHY GMII mode due to extra logic required for 4-bit MII to 8-bit internal conversion and nibble alignment to proper octet boundaries.

Artix-7 Devices

Table 2-3 provides approximate utilization figures for various core options when a single instance of the core is instantiated in a Artix™-7 device.

Table 2-3: **Resource Utilization for Artix-7 Devices**

Parameter Values			Device Resources				
Mode	**MDIO Interface**	**Auto-Negotiation**	**Slices**	**FFs**	**LUTs**	**LUTRAM**	**BUFGs**
MAC Mode	Yes	Yes	1565	2946	2752	467	0
	Yes	No	1167	2310	2138	463	0
	No	Yes	1307	2658	2342	435	0
	No	No	1157	2074	1656	431	0
PHY GMII Mode	Yes	Yes	1531	2979	2797	467	0
	Yes	No	1314	2275	1941	463	0
	No	Yes	1389	2655	2295	435	0
	No	No	973	2039	1787	431	0
PHY MII Mode	Yes	Yes	1596	3127	2895	467	0
	Yes	No	1370	2423	2051	463	0
	No	Yes	1493	2803	2367	435	0
	No	No	1135	2187	1829	431	0

Note: Resource Utilization numbers in PHY MII mode are more than PHY GMII mode due to extra logic required for 4-bit MII to 8-bit internal conversion and nibble alignment to proper octet boundaries.

Port Descriptions

All ports of the core are internal connections in FPGA logic. An HDL example design (delivered with the core) connects the core, where appropriate, to a device-specific transceiver, and/or add IBUFs, OBUFs, and IOB flip-flops to the external signals of the GMII/MII. IOBs are added to the remaining unconnected ports to take the example design through the Xilinx implementation software.

All clock management logic is placed in this example design, allowing you more flexibility in implementation (such as designs using multiple cores). This example design is provided in both VHDL and Verilog. For more information on the example design provided, see Chapter 11, Detailed Example Design.

Figure 2-1 shows the pinout for the QSGMII core using a device-specific transceiver *with* the optional MDIO Management and optional Auto-Negotiation. For more information see Chapter 9, Customizing and Generating the Core.

The port name for multiple instances of an interface is generalized as "CHx". "CHx" takes the value "CH0", "CH1", "CH2", and "CH3".

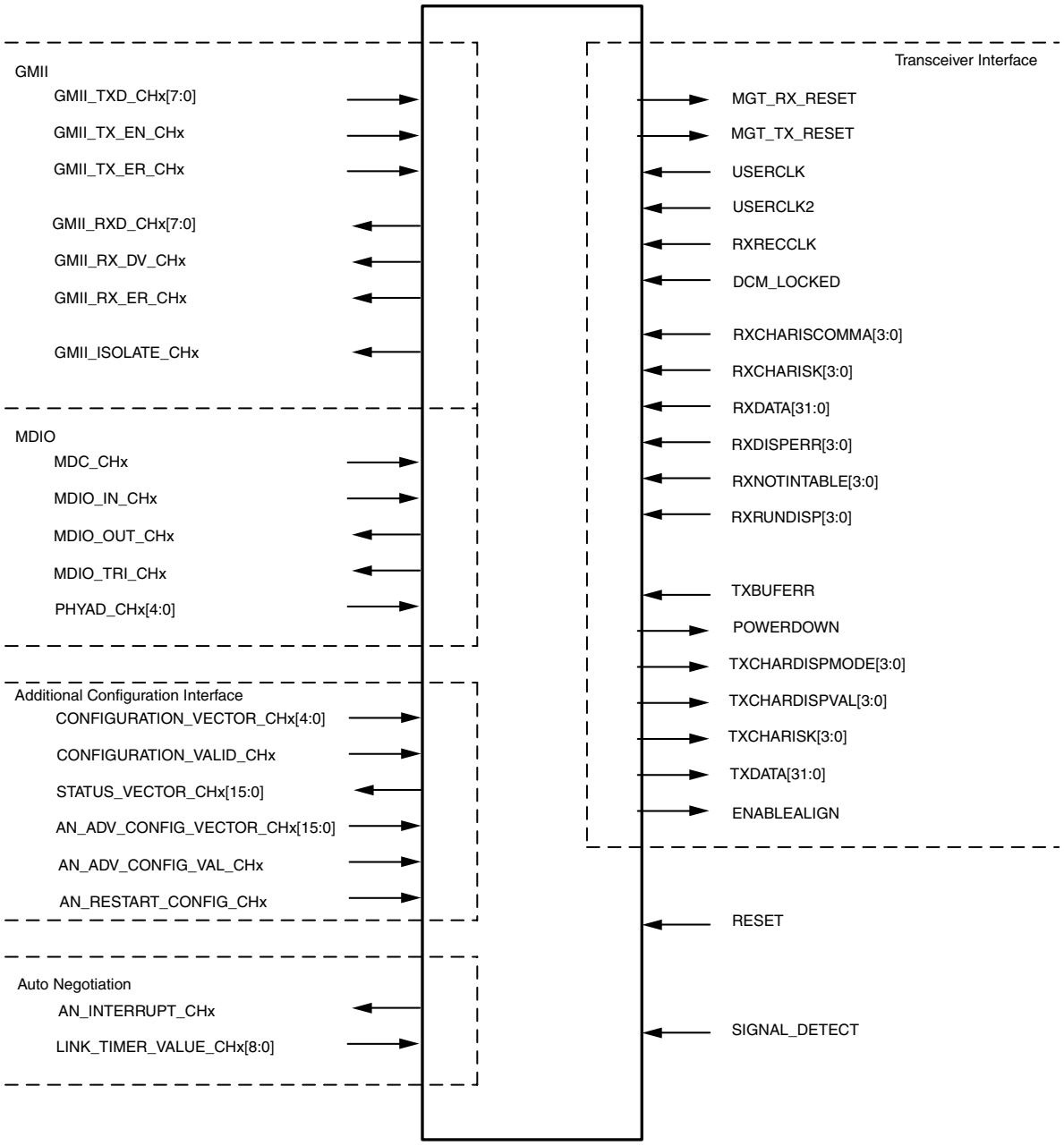

Figure 2-1: **Component Pinout of QSGMII *with* Optional MDIO and Auto-Negotiation**

Figure 2-2 shows the pinout for the QSGMII core using a device-specific transceiver *with only* the optional MDIO Management. For more information see Chapter 9, Customizing and Generating the Core.

The port name for multiple instances of an interface is generalized as "CHx". "CHx" takes the value "CH0", "CH1", "CH2", and "CH3".

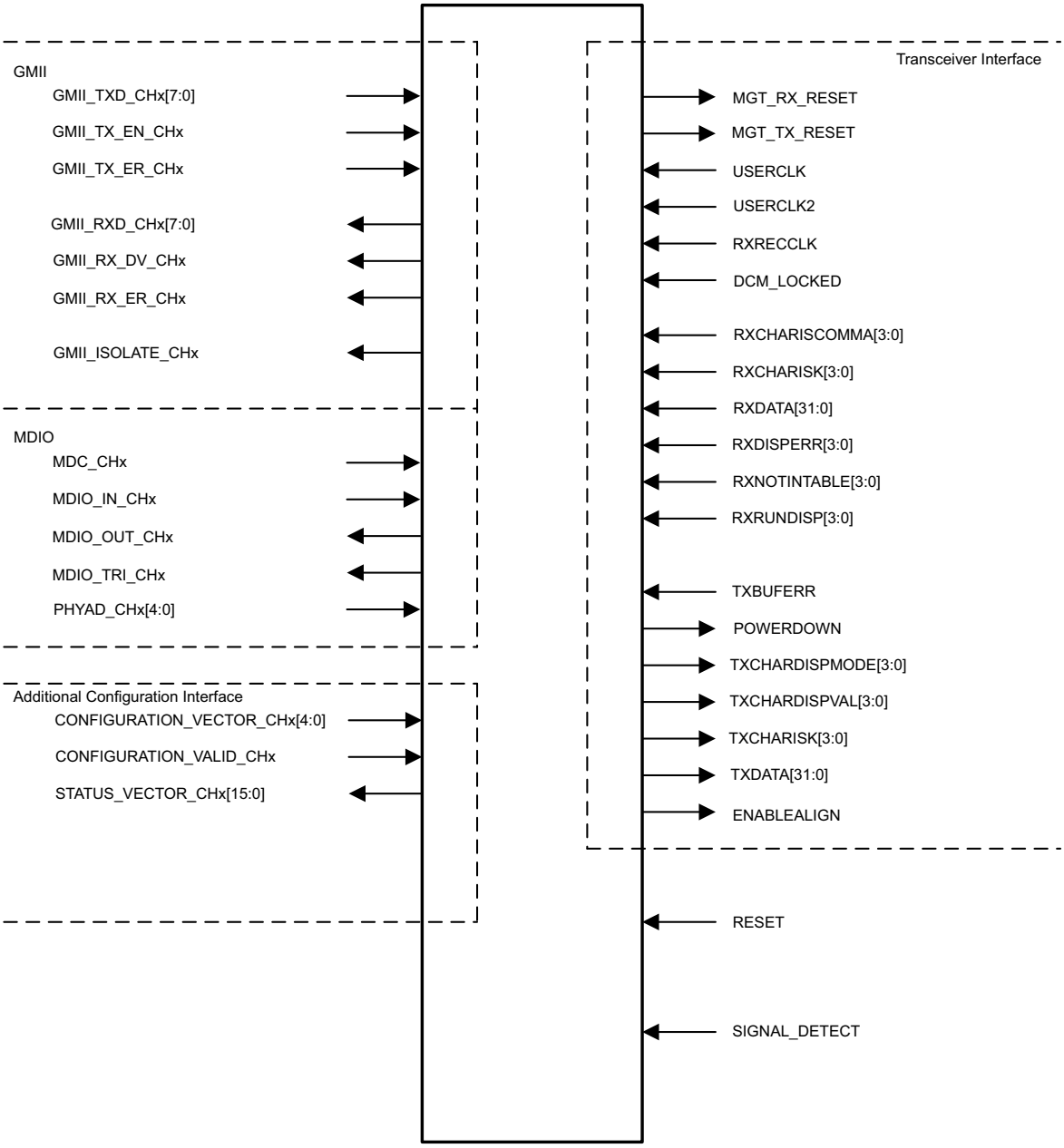

Figure 2-2: **Component Pinout of QSGMII *with* only Optional MDIO Management**

Figure 2-3 shows the pinout for the QSGMII core using a device-specific transceiver *with only* the optional Auto-Negotiation. For more information see Chapter 9, Customizing and Generating the Core.

The port name for multiple instances of an interface is generalized as "CHx". "CHx" takes the value "CH0", "CH1", "CH2", and "CH3".

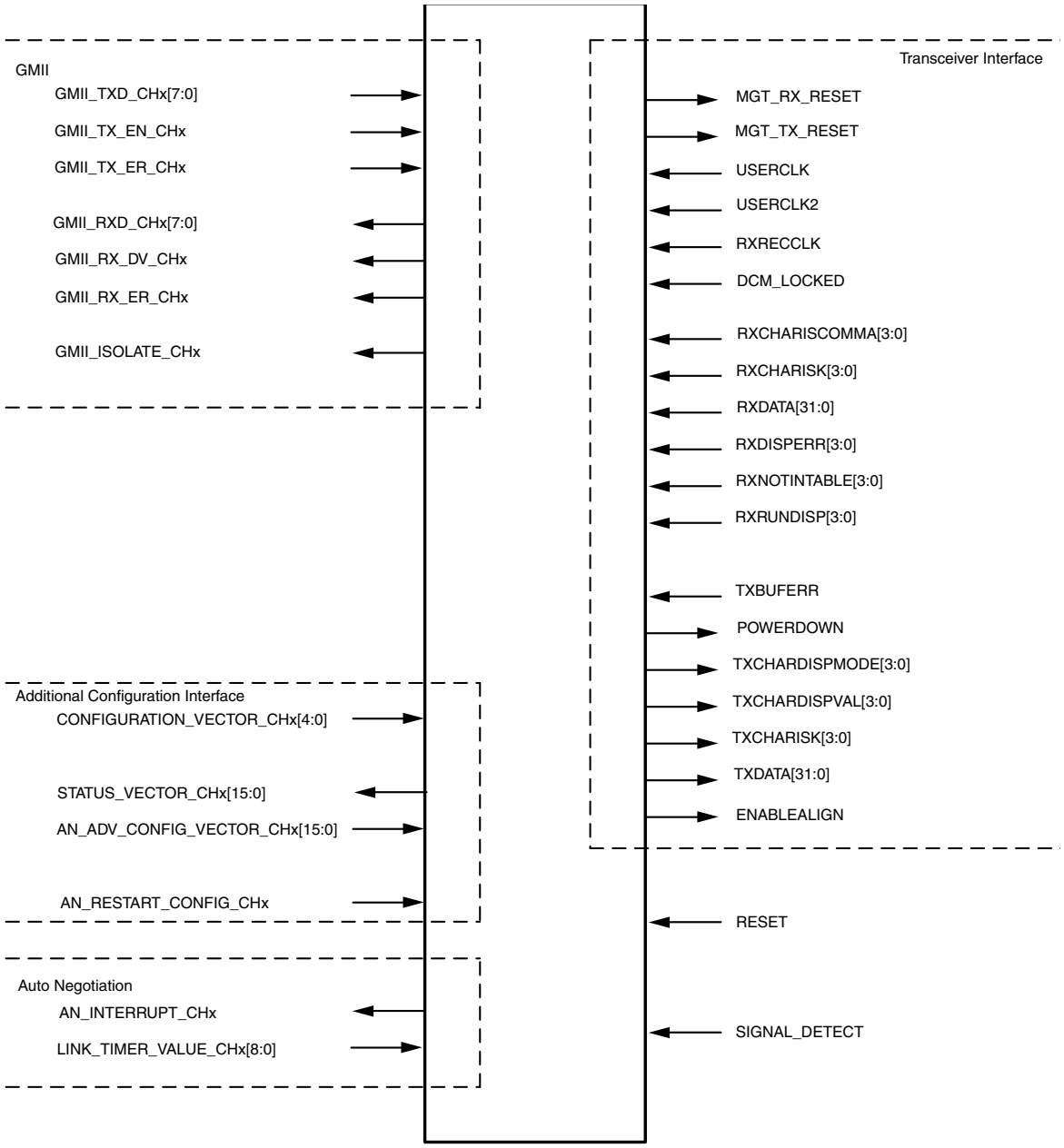

Figure 2-3: **Component Pinout of QSGMII *with* only Optional Auto-Negotiation**

Figure 2-4 shows the pinout for the QSGMII core using a device-specific transceiver *without* optional MDIO or Auto-Negotiation. For more information see Chapter 9, Customizing and Generating the Core.

The port name for multiple instances of an interface is generalized as “CHx”. “CHx” takes the value “CH0”, “CH1”, “CH2”, and “CH3”.

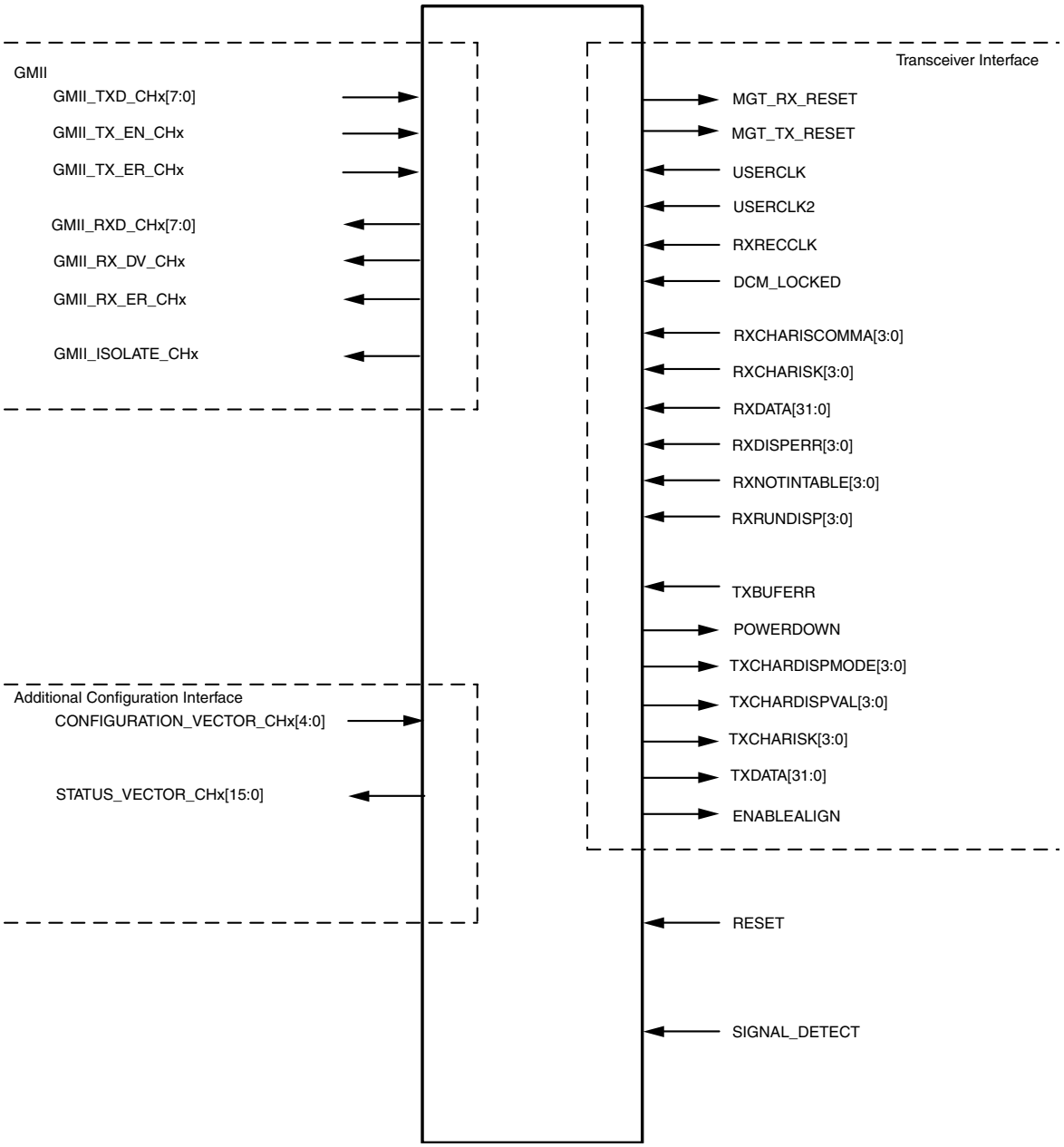

Figure 2-4: **Component Pinout for QSGMII *without* Optional MDIO or Optional Auto-Negotiation**

Client Side Interface

This interface contains four groups of interfaces, with each group containing a set of the GMII interface, the optional management interface if supported and configuration vectors. The interfaces end in “chx”, taking the values ch0 to ch3, indicating the port connection to the respective GMII interface.

GMII Pinout

Table 2-4 describes the GMII-side interface signals of the core that are common to all parameterizations of the core. These are typically attached to an Ethernet MAC, either off-chip or internally integrated. The HDL example design delivered with the core connects these signals to IOBs to provide a place-and-route example.

Table 2-4: **GMII Interface Signals Pinout**

Signal	Direction	Description
gmii_txd_chx[7:0] (1)	Input	GMII Transmit data from MAC
gmii_tx_en_chx(1)	Input	GMII Transmit control signal from MAC
gmii_tx_er_chx(1)	Input	GMII Transmit control signal from MAC
gmii_rxd_chx[7:0] (2)	Output	GMII Received data to MAC
gmii_rx_dv_chx(2)	Output	GMII Received control signal to MAC
gmii_rx_er_chx(2)	Output	GMII Received control signal to MAC
gmii_isolate_chx(2)	Output	IOB 3-state control for GMII Isolation. Only of use when implementing an external GMII as illustrated by the example design HDL.

1. When the Transmitter Elastic Buffer is present, these signals are synchronous to gmii_tx_clk. When the Transmitter Elastic Buffer is omitted, see (2).
2. These signals are synchronous to the internal 125 MHz reference clock of the core. This is userclk2.

Common Signals

Table 2-5 describes the remaining signals common to all parameterizations of the core.

Table 2-5: **Other Common Signals**

Signal	Direction	Description
reset	Input	Asynchronous reset for the entire core. Active-High. Clock domain is not applicable.
signal_detect	Input	Signal direct from the Physical Medium Dependent (PMD) sublayer indicating the presence of light detected at the optical receiver. If set to 1, indicates that the optical receiver has detected light. If set to 0, this indicates the absence of light. If unused, this signal should be set to 1 to enable correct operation the core. Clock domain is not applicable.

MDIO Management Interface Pinout (Optional)

The optional MDIO Management Interface is provided for each instance of SGMII. The "chx" suffix denotes a generic nomenclature for describing the interface. Each of the interfaces are identified with "chx" taking values from "ch0" to "ch3".

Table 2-6 describes the optional MDIO interface signals of the core that are used to access the PCS Management Registers. Each of these interfaces is typically connected to the MDIO port of a MAC device, either off-chip or to an internally integrated MAC core. For more information, see Management Registers.

Table 2-6: **Optional MDIO Interface Pinout**

Signal	Direction	Clock Domain	Description
mdc_chx	Input	NA	Management clock (<= 2.5 MHz).
mdio_in_chx(1)	Input	mdc_chx	Input data signal for communication with the instance number "x" of the MDIO controller (for example, an Ethernet MAC). Tie High if unused.
mdio_out_chx(1)	Output	mdc_chx	Output data signal for communication with the instance number "x" of the MDIO controller (for example, an Ethernet MAC).
mdio_tri_chx(1)	Output	mdc_chx	3-state control for MDIO signals. The value 0 signals that the value on mdio_out should be asserted onto the MDIO interface.
phyad_chx[4:0]	Input	NA	Physical Addresses of the PCS Management register set of each "x" instance of SGMII. It is expected that this signal will be tied off to a logical value.

1. These signals can be connected to a 3-state buffer to create a bidirectional mdio signal suitable for connection to an external MDIO controller (for example, an Ethernet MAC).

Auto-Negotiation Interface Pinout (Optional)

Table 2-7 describes the signals present when the optional Auto-Negotiation functionality is present.

Table 2-7: **Optional Auto-Negotiation Interface signal Pinout**

Signal	Direction	Description
link_timer_value_chx[8:0][(1)]	Input	Used to configure the duration of the Auto-Negotiation Link Timer period. The duration of this timer is set to the binary number input into this port multiplied by 4096 clock periods of the 125 MHz reference clock (8 ns). It is expected that this signal will be tied off to a logical value.
an_interrupt_chx[(1)]	Output	When an optional management interface is present, active-High interrupt to signal the completion of an Auto-Negotiation cycle. This interrupt can be enabled/disabled and cleared by writing to the appropriate PCS Management Register. When an optional management interface is not present, this signal just indicates the completion of the Auto-Negotiation cycle. Is reset automatically if Auto-Negotiation restarts. This bit cannot be cleared.

1. These signals are synchronous to the internal 125 MHz reference clock of the core. This is userclk2 when the core is used with the device-specific transceiver

Additional Configuration Interface

This interface can be used over and above the optional management interface to write into the Control Register (Register 0) and the Auto-Negotiation Advertisement Register (Register 4).

Table 2-8: **Additional Configuration Interface Signal Pinout**

Signal	Direction	Description
configuration_vector_chx[5:0][(1)]		• Bit[0]:Unidirectional Enable When set to 1, Enable Transmit irrespective of the state of RX. When set to 0, Normal operation
		• Bit[1]: Reserved
	Input	• Bit[2]: Power Down When set to 1, the device-specific transceiver is placed in a low-power state. A reset must be applied to clear. MDIO must be present to apply reset. This bit is valid only on configuration_vector_ch0 and is reserved in other instances of configuration_vector.
		• Bit[3] Isolate When set to 1, the GMII should be electrically isolated. When set to 0, normal operation is enabled
		• Bit[4] Auto-Negotiation Enable This signal is valid only if the Auto-Negotiation (AN) module is enabled through the CORE Generator™/Vivado™ IP Catalog tools. When set to 1, the signal enables the AN feature. When set to 0, AN is disabled.
configuration_vector_valid_chx[(1)]	Input	This signal is valid only when the MDIO interface is present. The rising edge of this signal is the enable signal to overwrite the Register 0 contents that were written from the MDIO interface. For triggering a fresh update of Register 0 through configuration_vector_chx, this signal should be deasserted and then reasserted.

Table 2-8: **Additional Configuration Interface Signal Pinout *(Cont'd)***

Signal	Direction	Description
an_adv_config_vector_chx[15:0][(1)]	Input	QSGMII operating in MAC Mode, the AN_ADV register is hard wired internally to "0x0001" and this bus has no effect. For QSGMII operating in PHY mode, the AN_ADV register is programmed by this bus as specified for the following bits. Bit[0]: Always 1 Bits [9:1]: Reserved Bits [11:10]: Speed 1 1 Reserved 1 0 1000 Mb/s 0 1 100 Mb/s 0 0 10 Mb/s Bits [12]:Duplex Mode 1 Full Duplex 0 Half Duplex Bit[13]: Reserved Bit [14]: Acknowledge Bit [15]: PHY Link Status 1 Link Up 0 Link Down
an_adv_config_valid_chx [(1)]	Input	This signal is valid only when the MDIO interface is present. The rising edge of this signal is the enable signal to overwrite the Register 4 contents that were written from the MDIO interface. For triggering a fresh update of Register 4 through an_adv_config_vector_chx, this signal should be deasserted and then reasserted.
an_restart_config_chx[(1)]	Input	This signal is valid only when AN is present. The rising edge of this signal is the enable signal to overwrite Bit 9 of Register 0. For triggering a fresh AN Start, this signal should be deasserted and then reasserted.

Table 2-8: **Additional Configuration Interface Signal Pinout *(Cont'd)***

Signal	Direction	Description
status_vector_chx[15:0][(1)]	Output	• Bit[0]: Link Status This signal indicates the status of the link. When high, the link is valid; synchronization of the link has been obtained and Auto-Negotiation (if present and enabled) has successfully completed. When low, a valid link has not been established. Either link synchronization has failed or Auto-Negotiation (if present and enabled) has failed to complete. When auto-negotiation is enabled, this signal is identical to Status Register Bit 1.2: Link Status. When auto-negotiation is disabled, this signal is identical to status_vector_chx Bit[1].
		• Bit[1]: Link Synchronization This signal indicates the state of the synchronization state machine (*IEEE 802.3* figure 36-9) which is based on the reception of valid 8B/10B code groups. This signal is similar to Bit[0] (Link Status), but is not qualified with Auto-Negotiation. When high, link synchronization has been obtained and in the synchronization state machine, sync_status=OK. When low, synchronization has failed.
		• Bit[2]: RUDI(/C/) The core is receiving /C/ ordered sets (Auto-Negotiation Configuration sequences).
		• Bit[3]: RUDI(/I/) The core is receiving /I/ ordered sets (Idles).
		• Bit[4]: RUDI(INVALID) The core has received invalid data while receiving/C/ or /I/ ordered set.
		• Bit[5]: RXDISPERR The core has received a running disparity error during the 8B/10B decoding function.

Table 2-8: **Additional Configuration Interface Signal Pinout *(Cont'd)***

Signal	Direction	Description
status_vector_chx[15:0][(1)] (continued)	Output	• Bit[6]: RXNOTINTABLE The core has received a code group that is not recognized from the 8B/10B coding tables.
		• Bit[7]: PHY Link Status This bit represents the link status of the external PHY device attached to the other end of the QSGMII link (high indicates that the PHY has obtained a link with its link partner; low indicates that is has not linked with its link partner.)
		• Bit[9:8]: Remote Fault Encoding This signal indicates the remote fault encoding (*IEEE 802.3* table 37-3). This signal is validated by bit 13 of the status_vector_chx and is only valid when Auto-Negotiation is enabled. This signal has no significance when the core is in PHY mode and indicates "00".
		• Bit [11:10]: SPEED This signal indicates that the speed is negotiated and is only valid when Auto-Negotiation is enabled. The signal encoding follows: Bit[11] Bit[10] 1 1 Reserved 1 0 1000 Mb/s 0 1 100 Mb/s 0 0 10 Mb/s
		• Bit[12]: Duplex Mode This bit indicates the Duplex mode negotiated with the link partner. 1 = Full Duplex 0 = Half Duplex
		• Bit[13] Remote Fault When this bit is logic 1, it indicates that a remote fault is detected and the type of remote fault is indicated by status_vector_chx bits[9:8]. ***Note:*** This bit is only deasserted when an MDIO read is made to status register (register 1). This signal has no significance in QSGMII PHY mode.

Table 2-8: **Additional Configuration Interface Signal Pinout *(Cont'd)***

Signal	Direction	Description
status_vector_chx[15:0][1] (continued)	Output	• Bits[15;14]: Pause These bits reflect the bits [8:7] of Register 5 (Link Partner Base AN Register). Bit[15] Bit[14] 0 0 No Pause 0 1 Symmetric Pause 1 0 Asymmetric Pause towards Link partner 1 1 Both Symmetric Pause and Asymmetric Pause towards link partner
1. Signals are synchronous to the core internal 125 MHz reference clock userclk2 when used with a device-specific transceiver.		

Physical Side Interface

Table 2-9 describes the interface to the device-specific transceiver. The core is connected to the chosen transceiver in the appropriate HDL example design delivered with the core.

Table 2-9: **Transceiver Interface Pinout**

Signal	Direction	Description
mgt_rx_reset[1]	Output	Reset signal issued by the core to the device-specific transceiver receiver path. Connects to the RXRESET signal of the device-specific transceiver.
mgt_tx_reset[2]	Output	Reset signal issued by the core to the device-specific transceiver transmitter path. Connects to the TXRESET signal of the device-specific transceiver.
userclk	Input	Also connected to TXUSRCLK of the device-specific transceiver. Clock domain is not applicable.
userclk2	Input	Also connected to TXUSRCLK2 of the device-specific transceiver. Clock domain is not applicable.
rxrecclk	Input	Also connected to RXUSRCLK2 of the device-specific transceiver. Clock domain is not applicable.
dcm_locked	Input	A Digital Clock Manager (DCM) can be used to derive userclk and userclk2. This is implemented in the HDL design example delivered with the core. The core uses this input to hold the device-specific transceiver in reset until the DCM obtains lock. Clock domain is not applicable.
rxchariscomma[3:0][1]	Input	Connects to device-specific transceiver signal of the same name.
rxcharisk[3:0][1]	Input	Connects to device-specific transceiver signal of the same name.
rxdata[31:0][1]	Input	Connects to device-specific transceiver signal of the same name.
rxdisperr[3:0][1]	Input	Connects to device-specific transceiver signal of the same name.
rxnotintable[3:0][1]	Input	Connects to device-specific transceiver signal of the same name.
rxrundisp[3:0][1]	Input	Connects to device-specific transceiver signal of the same name.
txbuferr[2]	Input	Connects to device-specific transceiver signal of the sam e name.
powerdown[2]	Output	Connects to device-specific transceiver signal of the same name.

Table 2-9: **Transceiver Interface Pinout *(Cont'd)***

Signal	Direction	Description
txchardispmode[3:0][(2)]	Output	Connects to device-specific transceiver signal of the same name.
txchardispval[3:0][(2)]	Output	Connects to device-specific transceiver signal of the same name.
txcharisk[3:0][(2)]	Output	Connects to device-specific transceiver signal of the same name.
txdata[31:0][(2)]	Output	Connects to device-specific transceiver signal of the same name.
enablealign[(2)]	Output	Connects to device-specific transceiver signal of the same name.

1. When the core is used with a device-specific transceiver, rxrecclk is used as the 125 MHz reference clock for driving these signals.
2. When the core is used with a device-specific transceiver, userclk2 is used as the 125 MHz reference clock for driving these signals.

Register Space

MDIO Management System

This section gives the description of one instance MDIO_CH0 of the four instances of the MDIO Management System. The other instances follow the same actions.

When the optional MDIO Management Interface is selected, the configuration and status of the SGMII module instance is achieved by the Management Registers accessed through the serial Management Data Input/Output Interface (MDIO).

MDIO Bus System

The MDIO interface for 1 Gb/s operation (and slower speeds) is defined in *IEEE 802.3-2008*, clause 22. Figure 2-5 illustrates an example MDIO bus system. This two-wire interface consists of a clock (MDC) and a shared serial data line (MDIO). The maximum permitted frequency of MDC is set at 2.5 MHz. An Ethernet MAC is shown as the MDIO bus master (the Station Management (STA) entity). Two PHY devices are shown connected to the same bus, both of which are MDIO slaves (MDIO Managed Device (MMD) entities).

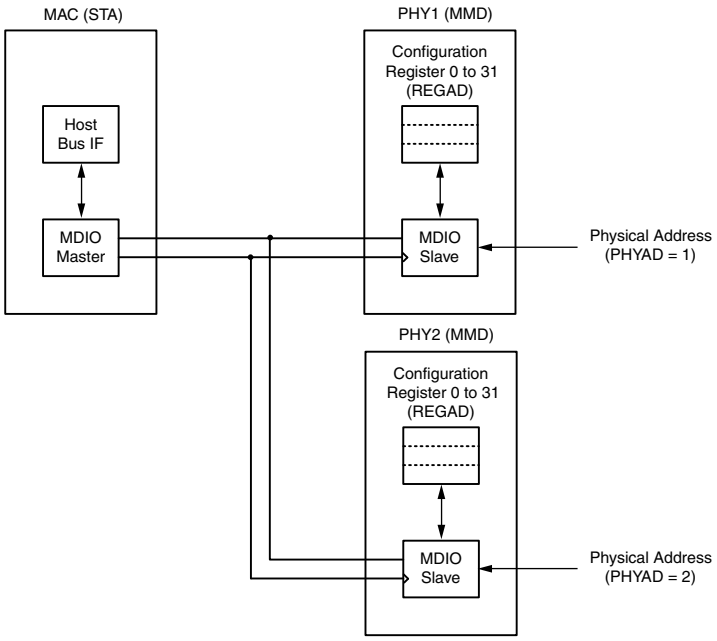

Figure 2-5: **Typical MDIO Managed System**

The MDIO bus system is a standardized interface for accessing the configuration and status registers of Ethernet PHY devices. In the example illustrated, the Management Host Bus I/F of the Ethernet MAC is able to access the configuration and status registers of two PHY devices through the MDIO bus.

MDIO Transactions

All transactions, read or write, are initiated by the MDIO master. All MDIO slave devices, when addressed, must respond. MDIO transactions take the form of an MDIO frame, containing fields for transaction type, address and data. This MDIO frame is transferred across the MDIO wire synchronously to MDC. The abbreviations that are used in this section are explained in Table 2-10.

Table 2-10: **Abbreviations and Terms**

Abbreviation	Term
PRE	Preamble
ST	Start of Frame
OP	Operation Code
PHYAD	Physical Address
REGAD	Register Address
TA	Turnaround

Write Transaction

Figure 2-6 shows a write transaction across the MDIO, defined as OP="01." The addressed PHY device (with physical address PHYAD) takes the 16-bit word in the Data field and writes it to the register at REGAD.

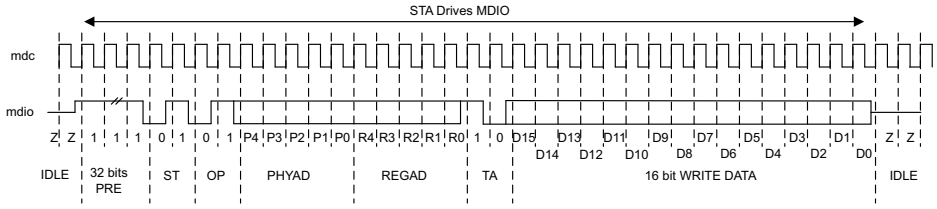

Figure 2-6: **MDIO Write Transaction**

Read Transaction

Figure 2-7 shows a read transaction, defined as OP="10." The addressed PHY device (with physical address PHYAD) takes control of the MDIO wire during the turn-around cycle and then returns the 16-bit word from the register at REGAD.

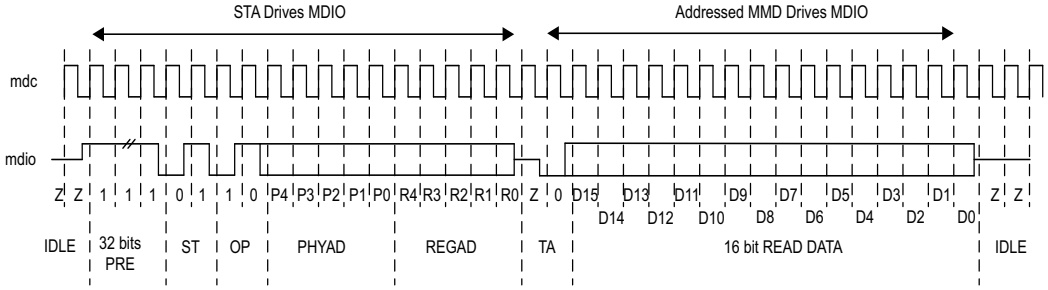

Figure 2-7: **MDIO Read Transaction**

MDIO Addressing

MDIO Addresses consists of two stages: Physical Address (PHYAD) and Register Address (REGAD).

Physical Address (PHYAD)

As shown in Figure 2-5, two PHY devices are attached to the MDIO bus. Each of these has a different physical address. To address the intended PHY, its physical address should be known by the MDIO master (in this case an Ethernet MAC) and placed into the PHYAD field of the MDIO frame (see MDIO Transactions).

The PHYAD field for an MDIO frame is a 5-bit binary value capable of addressing 32 unique addresses. However, every MDIO slave must respond to physical address 0. This requirement dictates that the physical address for any particular PHY must not be set to 0 to avoid MDIO contention. Physical Addresses 1 through to 31 can be used to connect up to 31 PHY devices onto a single MDIO bus.

Register Address (REGAD)

Having targeted a particular PHY using PHYAD, the individual configuration or status register within that particular PHY must now be addressed. This is achieved by placing the individual register address into the REGAD field of the MDIO frame (see MDIO Transactions).

The REGAD field for an MDIO frame is a 5-bit binary value capable of addressing 32 unique addresses. The first 16 of these (registers 0 to 15) are defined by the *IEEE 802.3-2008*. The remaining 16 (registers 16 to 31) are reserved for PHY vendors own register definitions.

For details of the register map of PHY layer devices and a more extensive description of the operation of the MDIO Interface, see *IEEE 802.3-2008*.

Connecting the MDIO to an Internally Integrated STA

The MDIO ports of the QSGMII core can be connected to the MDIO ports of an internally integrated Station Management (STA) entity, such as the MDIO port of multi-instances of the Tri-Mode Ethernet MAC core.

Connecting the MDIO to an External STA

Figure 2-8 shows the MDIO ports of the QSGMII core connected to the MDIO of an external STA entity. In this situation, `mdio_in_chx`, `mdio_out_chx`, and `mdio_tri_chx` must be connected to a 3-state buffer to create a bidirectional wire, `mdio_chx`.

This 3-state buffer can either be external to the FPGA or internally integrated by using an IOB IOBUF component with an appropriate SelectIO™ interface standard suitable for the external PHY.

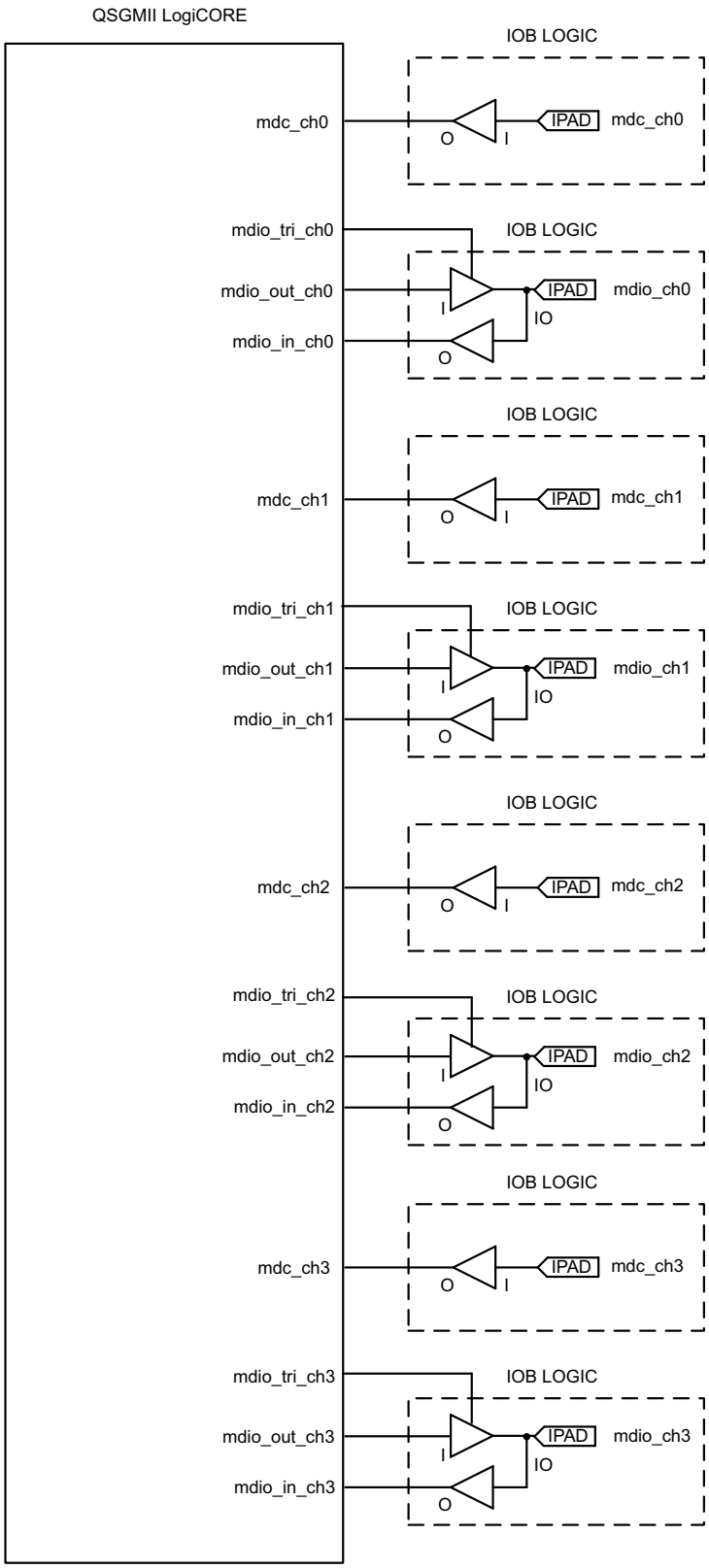

Figure 2-8: **Creating an External MDIO Interface**

Management Registers

The contents of the Management Registers can be accessed using the REGAD field of the MDIO frame. Contents vary depending on the Xilinx CORE Generator™ or Vivado™ tool options, and are defined in the following sections in this guide.

- QSGMII Using Optional Auto-Negotiation
- QSGMII Without Optional Auto-Negotiation

QSGMII Using Optional Auto-Negotiation

The registers provided are duplicated for each instance of the SGMII module in this core. The registers are adaptations of those defined in clauses 22 and 37 of the *IEEE 802.3-2008* specification. In a QSGMII implementation, two different types of links exist. They are the QSGMII link between the MAC and PHY (QSGMII link) and the link across the Ethernet Medium itself (Medium).

Information regarding the state of both of these links is contained within the following registers. Where applicable, the abbreviations *QSGMII link* and *Medium* are used in the register descriptions. Registers at undefined addresses are read-only and return 0s.

Table 2-11: **Management Registers for QSGMII with Auto-Negotiation**

Register Address	Register Name
0	SGMII Control Register
1	SGMII Status Register
2, 3	PHY Identifier
4	SGMII Auto-Negotiation Advertisement Register
5	SGMII Auto-Negotiation Link Partner Ability Base Register
6	SGMII Auto-Negotiation Expansion Register
7	SGMII Auto-Negotiation Next Page Transmit Register
8	SGMII Auto-Negotiation Next Page Receive Register
15	SGMII Extended Status Register
16	SGMII Vendor Specific: Auto-Negotiation Interrupt Control
18	SGMII Generic Control

Register 0: SGMII Control Register

Management Registers Channel/Module 0

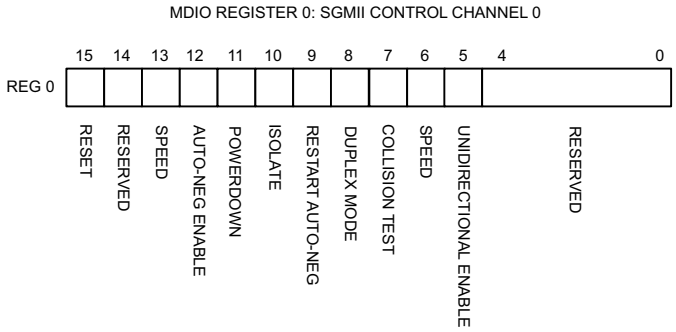

Figure 2-9: **MDIO Register 0: SGMII Control Register Channel/Module 0**

Table 2-12: **SGMII Control Register Channel/Module 0 (REGISTER 0)**

Bit(s)	Name	Description	Attributes	Default Value
0.15	Reset	1 = SGMII module 0 Reset 0 = Normal Operation	read/write self clearing	0
0.14	Reserved	Returns what is written	read/write	0
0.13	Speed Selection (LSB)	Always returns a '0' for this bit. Together with bit 0.6, speed selection of 1000 Mb/s is identified.	returns 0	0
0.12	Auto-Negotiation Enable	1 = Enable SGMII Auto-Negotiation Process 0 = Disable SGMII Auto-Negotiation Process	read/write	1
0.11	Power Down	1 = Power down 0 = Normal operation When set to 1, the device-specific transceiver is placed in a low-power state. This bit requires a reset (see bit 0.15) to clear.	read/ write	0
0.10	Isolate	1 = Electrically Isolate SGMII logic from GMII 0 = Normal operation	read/write	1
0.9	Restart Auto- Negotiation	1 = Restart Auto-Negotiation Process across SGMII link 0 = Normal Operation	read/write self clearing	0

Table 2-12: **SGMII Control Register Channel/Module 0 (REGISTER 0) *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
0.8	Duplex Mode	Always returns a 1 for this bit to signal Full-Duplex Mode	returns 1	1
0.7	Collision Test	Always returns a 0 for this bit to disable Collision (COL) test	returns 0	0
0.6	Speed Selection (MSB)	Always returns a 1 for this bit. Together with bit 0.13, speed selection of 1000 Mb/s is identified.	returns 1	1
0.5	Unidirectional al Enable	Enable transmit regardless of whether a valid link has been established provided AN is disabled.	read/ write	0
0.4:0.0	Reserved	Always return 0s, writes ignored.	returns 0s	00000

Management Registers Channels/Modules 1-3

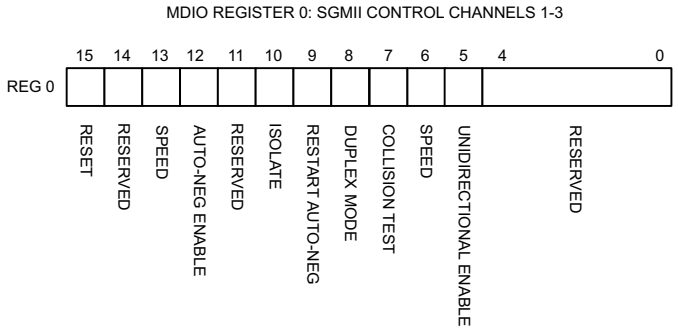

Figure 2-10: **MDIO Register 0: SGMII Control Channels/Modules 1-3**

Table 2-13: **SGMII Control Register Channels/Modules 1-3 (REGISTER 0)**

Bit(s)	Name	Description	Attributes	Default Value
0.15	Reset	1 = SGMII module 1-3 Reset 0 = Normal Operation	read/write self clearing	0
0.14	Reserved	Returns what is written	read/write	0
0.13	Speed Selection (LSB)	Always returns a 0 for this bit. Together with bit 0.6, speed selection of 1000 Mb/s is identified.	returns 0	0
0.12	Auto-Negotiation Enable	1 = Enable SGMII Auto-Negotiation Process 0 = Disable SGMII Auto-Negotiation Process	read/write	1
0.11	Reserved	Returns what is written	read/ write	0
0.10	Isolate	1 = Electrically Isolate SGMII logic from GMII 0 = Normal operation	read/write	1
0.9	Restart Auto-Negotiation	1 = Restart Auto-Negotiation Process across SGMII link 0 = Normal Operation	read/write self clearing	0
0.8	Duplex Mode	Always returns a 1 for this bit to signal Full-Duplex Mode	returns 1	1
0.7	Collision Test	Always returns a 0 for this bit to disable COL test	returns 0	0
0.6	Speed Selection (MSB)	Always returns a 1 for this bit. Together with bit 0.13, speed selection of 1000 Mb/s is identified.	returns 1	1
0.5	Unidirectional Enable	Enable transmit regardless of whether a valid link has been established provided AN is disabled.	read/ write	0
0.4:0.0	Reserved	Always return 0s, writes ignored	returns 0s	00000

Register 1: SGMII Status Register

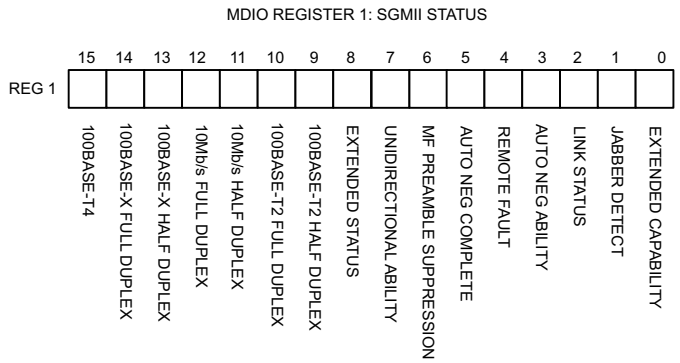

Figure 2-11: **MDIO Register 1: SGMII Status Register**

Table 2-14: **SGMII Status Register (REGISTER 1)**

Bit(s)	Name	Description	Attributes	Default Value
1.15	100BASE-T4	Always returns a 0 for this bit because 100BASE-T4 is not supported.	returns 0	0
1.14	100BASE-X Full Duplex	Always returns a 0 for this bit because 100BASE-X Full Duplex is not supported.	returns 0	0
1.13	100BASE-X Half Duplex	Always returns a 0 for this bit because 100BASE-X Half Duplex is not supported.	returns 0	0
1.12	10 Mb/s Full Duplex	Always returns a 0 for this bit because 10 Mb/s Full Duplex is not supported.	returns 0	0
1.11	10 Mb/s Half Duplex	Always returns a 0 for this bit because 10 Mb/s Half Duplex is not supported.	returns 0	0
1.10	100BASE-T2 Full Duplex	Always returns a 0 for this bit because 100BASE-T2 Full Duplex is not supported.	returns 0	0
1.9	100BASE-T2 Half Duplex	Always returns a 0 for this bit because 100BASE-T2 Half Duplex is not supported.	returns 0	0
1.8	Extended Status	Always returns a 1 for this bit to indicate the presence of the Extended Register (Register 15).	returns 1	1
1.7	Unidirectional Ability	Always returns 1, writes ignored.	returns 1	1
1.6	MF Preamble Suppression	Always returns a 1 for this bit to indicate that Management Frame Preamble Suppression is supported.	returns 1	1

Table 2-14: **SGMII Status Register (REGISTER 1) *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
1.5	Auto- Negotiation Complete	1 = Auto-Negotiation process completed across SGMII link. 0 = Auto-Negotiation process not completed across SGMII link.	read only	0
1.4	Remote Fault	1 = A fault on the Medium has been detected. 0 = No fault of the Medium has been detected.	read only self clearing on read	0
1.3	Auto- Negotiation Ability	Always returns a 1 for this bit to indicate that the SGMII core is capable of Auto-Negotiation.	returns 1	1
1.2	SGMII Link Status	1 = SGMII Link is up 0 = SGMII Link is down Latches 0 if SGMII Link Status goes down. Clears to current SGMII Link Status on read. See the following Link Status section for further details.	read only self clearing on read	0
1.1	Jabber Detect	Always returns a 0 for this bit because Jabber Detect is not supported.	returns 0	0
1.0	Extended Capability	Always returns a 0 for this bit because no extended register set is supported.	returns 0	0

Link Status

When high, the link is valid and has remained valid after this register was last read; synchronization of the link has been obtained and Auto-Negotiation (if enabled) has completed.

When low, either:

- A valid link has not been established; link synchronization has failed or Auto-Negotiation (if enabled) has failed to complete.

  OR

- Link synchronization was lost at some point after this register was previously read. However, the current link status might be good. *Therefore read this register a second time to get confirmation of the current link status*.

Regardless of whether Auto-Negotiation is enabled or disabled, there can be some delay in the deassertion of Link Status following the loss of synchronization of a previously successful link. This is due to the Auto-Negotiation state machine that requires that synchronization is lost for an entire link timer duration before changing state. For more information, see the 802.3 specification (the *an_sync_status* variable).

Registers 2 and 3 (PHY IDENTIFIER)

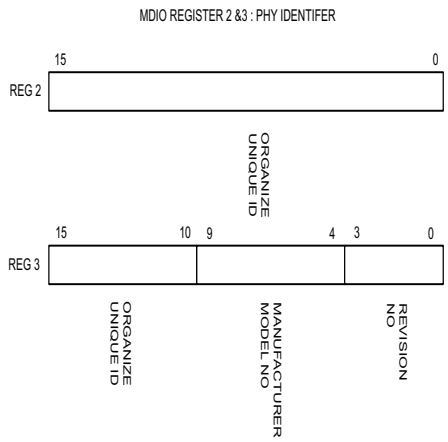

Figure 2-12: **MDIO Registers 2 and 3: (PHY IDENTIFIER)**

Table 2-15: **PHY Identifier (Registers 2 and 3)**

Bit(s)	Name	Description	Attributes	Default Value
2.15:0	Organizationally Unique Identifier	Always return 0s	returns 0s	0000000000000000
3.15:10	Organizationally Unique Identifier	Always return 0s	returns 0s	000000
3.9:4	Manufacturer model number	Always return 0s	returns 0s	000000
3.3:0	Revision Number	Always return 0s	returns 0s	0000

Register 4: SGMII Auto-Negotiation Advertisement

MAC Mode Of Operation

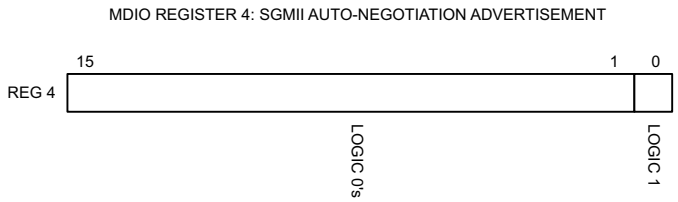

Figure 2-13: **MDIO Register 4: SGMII Auto-Negotiation Advertisement**

Table 2-16: **SGMII Auto-Negotiation Advertisement (Register 4)**

Bit(s)	Name	Description	Attributes	Default Value
4.15:0	All bits	SGMII defined value sent from the MAC to the PHY.	read only	0100000000000001

PHY Mode Of Operation

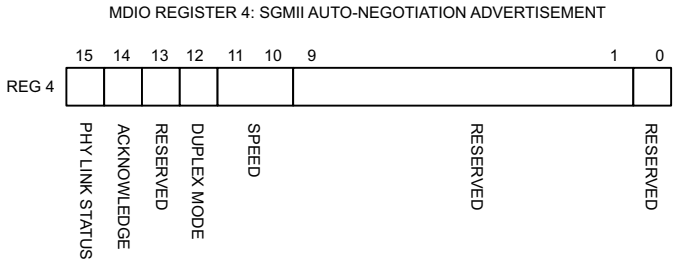

Figure 2-14: **MDIO Register 4: SGMII Auto-Negotiation Advertisement**

Table 2-17: **SGMII Auto-Negotiation Advertisement in PHY Mode (Register 4)**

Bit(s)	Name	Description	Attributes	Default Value
4.15	PHY Link Status	This refers to the link status of the PHY with its link partner across the Medium. 1 = Link Up 0 = Link Down	read/write	0
4.14	Acknowledge	Used by Auto-Negotiation function to indicate reception of a link partner's base or next page.	read/write	0
4.13	Reserved	Always returns 0, writes ignored	returns 0	0
4.12	Duplex Mode	1= Full Duplex 0 = Half Duplex	read/write	0
4.11:10	Speed	11 = Reserved 10 = 1 Gb/s 01 = 100 Mb/s 00 = 10 Mb/s	read/write	00
4.9:1	Reserved	Always return 0s	returns 0s	000000000
4.0	Reserved	Always returns 1	returns 1	1

Register 5: SGMII Auto-Negotiation Link Partner Ability

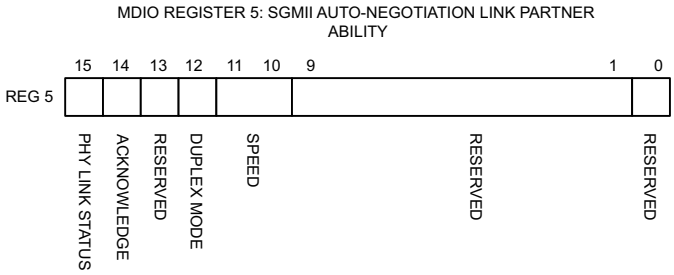

Figure 2-15: **MDIO Register 5: SGMII Auto-Negotiation Link Partner Ability**

The Auto-Negotiation Ability Base Register (Register 5) contains information related to the status of the link between the PHY and its physical link partner across the Medium.

Table 2-18: **SGMII Auto-Negotiation Link Partner Ability Base (Register 5)**

Bit(s)	Name	Description	Attributes	Default Value
5.15	PHY Link Status	This refers to the link status of the PHY with its link partner across the Medium. 1 = Link Up 0 = Link Down	read only	1
5.14	Acknowledge	Used by Auto-Negotiation function to indicate reception of a link partner's base or next page	read only	0
5.13	Reserved	Always returns 0, writes ignored	returns 0	0
5.12	Duplex Mode	1= Full Duplex 0 = Half Duplex	read only	0
5.11:10	Speed	11 = Reserved 10 = 1 Gb/s 01 = 100 Mb/s 00 = 10 Mb/s	read only	00
5.9:1	Reserved	Always return 0s	returns 0s	000000000
5.0	Reserved	Always returns 1	returns 1	1

Register 6: SGMII Auto-Negotiation Expansion

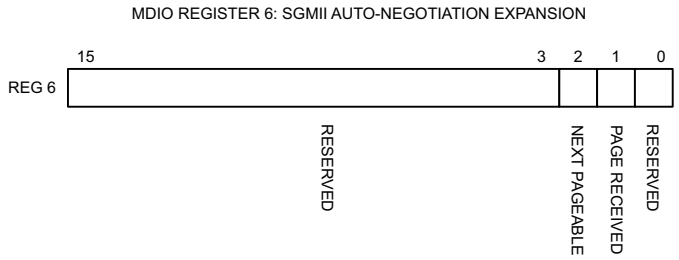

Figure 2-16: **MDIO Register 6: SGMII Auto-Negotiation Expansion**

Table 2-19: **SGMII Auto-Negotiation Expansion (Register 6)**

Bit(s)	Name	Description	Attributes	Default Value
6.15:3	Reserved	Always return 0s	returns 0s	0000000000000
6.2	Next Page Able	This bit is ignored as the core currently does not support next page. This feature can be enabled on request.	returns 1	1
6.1	Page Received	1 = A new page has been received 0 = A new page has not been received	read only self clearing on read	0
6.0	Reserved	Always return 0s	returns 0s	0000000

Register 7: SGMII Auto-Negotiation Next Page Transmit

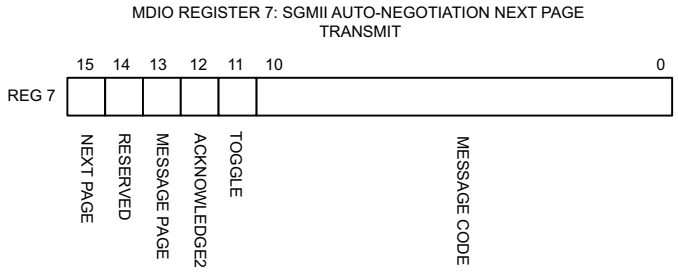

Figure 2-17: **MDIO Register 7: SGMII Auto-Negotiation Next Page Transmit**

Table 2-20: **SGMII Auto-Negotiation Next Page Transmit (Register 7)**

Bit(s)	Name	Description	Attributes	Default Value[1]
7.15	Next Page	1 = Additional Next Page(s) will follow 0 = Last page	read/ write	0
7.14	Reserved	Always returns 0	returns 0	0
7.13	Message Page	1 = Message Page 0 = Unformatted Page	read/ write	1
7.12	Acknowledge 2	1 = Comply with message 0 = Cannot comply with message	read/ write	0
7.11	Toggle	Value toggles between subsequent Next Pages	read only	0
7.10:0	Message / Unformatted Code Field	Message Code Field or Unformatted Page Encoding as dictated by 7.13	read/ write	00000000001 (Null Message Code)

1. This register returns the default values because the core does not support next page. The feature can be enabled, if requested.

Register 8: SGMII Next Page Receive

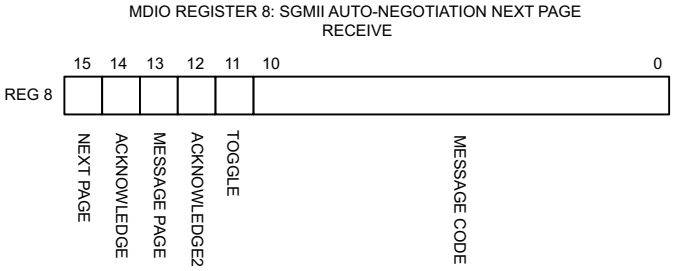

Figure 2-18: **MDIO Register 8: SGMII Auto-Negotiation Next Page Receive**

Table 2-21: **SGMII Auto-Negotiation Next Page Receive (Register 8)**

Bit(s)	Name	Description	Attributes	Default Value
8.15	Next Page	1 = Additional Next Page(s) will follow 0 = Last page	read only	0
8.14	Acknowledge	Used by Auto-Negotiation function to indicate reception of a link partner's base or next page	read only	0
8.13	Message Page	1 = Message Page 0 = Unformatted Page	read only	0
8.12	Acknowledge 2	1 = Comply with message 0 = Cannot comply with message	read only	0
8.11	Toggle	Value toggles between subsequent Next Pages	read only	0
8.10:0	Message / Unformatted Code Field	Message Code Field or Unformatted Page Encoding as dictated by 8.13	read only	00000000000

Register 15: SGMII Extended Status

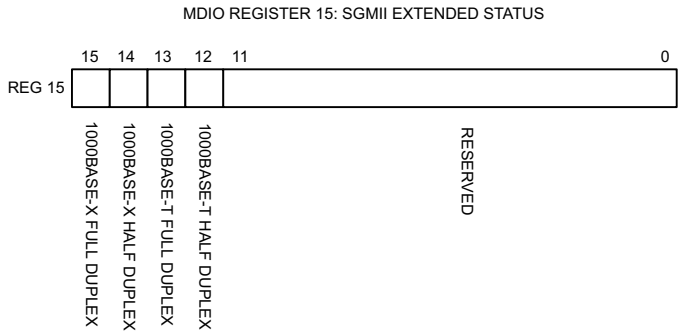

Figure 2-19: **MDIO Register 15: SGMII Extended Status**

Table 2-22: **SGMII Extended Status (Register 15)**

Bit(s)	Name	Description	Attributes	Default Value
15.15	1000BASE-X Full Duplex	Always returns a 1 for this bit because 1000BASE-X Full Duplex is supported	returns 1	1
15.14	1000BASE-X Half Duplex	Always returns a 0 for this bit because 1000BASE-X Half Duplex is not supported	returns 0	0
15.13	1000BASE-T Full Duplex	Always returns a 0 for this bit because 1000BASE-T Full Duplex is not supported	returns 0	0
15.12	1000BASE-T Half Duplex	Always returns a 0 for this bit because 1000BASE-T Half Duplex is not supported	returns 0	0
15:11:0	Reserved	Always return 0s	returns 0s	000000000000

Register 16: SGMII Auto-Negotiation Interrupt Control

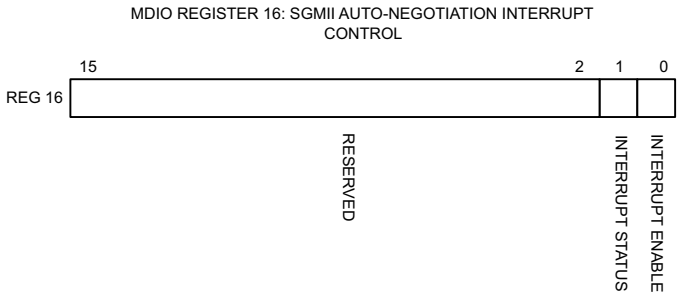

Figure 2-20: **MDIO Register 16: SGMII Auto-Negotiation Interrupt Control**

Table 2-23: **SGMII Auto-Negotiation Interrupt Control (Register 16)**

Bit(s)	Name	Description	Attributes	Default Value
16.15:2	Reserved	Always return 0s	returns 0s	00000000000000
16.1	Interrupt Status	1 = Interrupt is asserted 0 = Interrupt is not asserted If the interrupt is enabled, this bit is asserted on completion of an Auto-Negotiation cycle across the SGMII link; it is only cleared by writing 0 to this bit. If the Interrupt is disabled, the bit is set to 0. ***Note:*** The an_interrupt port of the core is wired to this bit.	read/write	0
16.0	Interrupt Enable	1 = Interrupt enabled 0 = Interrupt disabled	read/write	1

Register 18: SGMII Generic Control

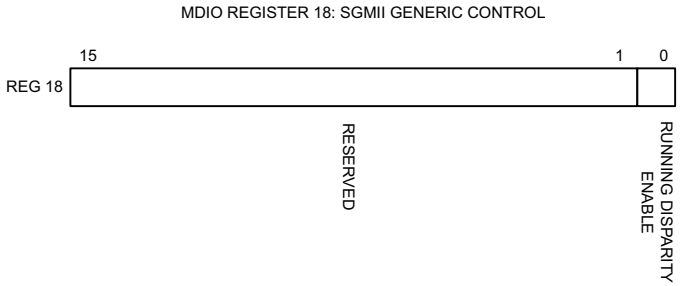

Figure 2-21: **MDIO Register 18: SGMII Generic Control**

Table 2-24: **SGMII Generic Control (Register 18)**

Bit(s)	Name	Description	Attributes	Default Value
18.15:1	Reserved	Always return 0s	returns 0s	000000000000000
18.0	Running Disparity Enable	1 =Running Disparity Checking enabled 0 = Running Disparity Checking disabled	read/ write	0

QSGMII Without Optional Auto-Negotiation

The registers provided are duplicated for each instance of the SGMII module in this core. The registers provided for SGMII operation in this core are adaptations of those defined in clauses 22 and 37 of the *IEEE 802.3-2008* specification. In a QSGMII implementation, two different types of links exist. They are the QSGMII link between the MAC and PHY (QSGMII link) and the link across the Ethernet Medium itself (Medium). Information about the state of the QSGMII link is available in the registers that follow.

The state of the link across the Ethernet Medium itself is not directly available when QSGMII Auto-Negotiation is not present. For this reason, the status of the link and the results of the PHYs Auto-Negotiation (for example, Speed and Duplex mode) must be obtained directly from the management interface of the connected PHY module. Registers at undefined addresses are read-only and return 0s.

Table 2-25: **MDIO Registers for SGMII without Optional Auto-Negotiation**

Register Address	Register Name
0	SGMII Control Register
1	SGMII Status Register
2, 3	PHY Identifier
15	SGMII Extended Status Register
18	SGMII Generic Control

Register 0: SGMII Control Register

Management Registers Channel/Module 0

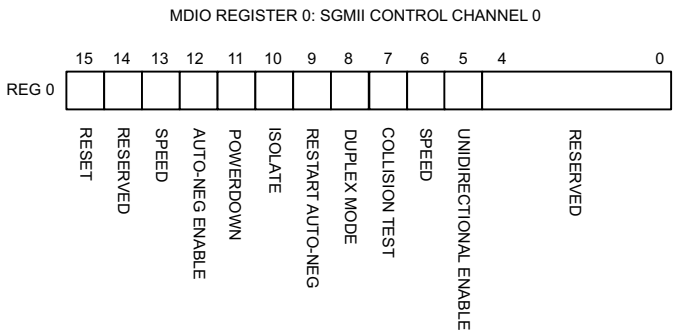

Figure 2-22: **MDIO Register 0: SGMII Control Register Channel/Module 0**

Table 2-26: **SGMII Control Register Channel/Module 0 (REGISTER 0)**

Bit(s)	Name	Description	Attributes	Default Value
0.15	Reset	1 = SGMII module 0 Reset 0 = Normal Operation	read/write self clearing	0
0.14	Reserved	Returns what is written	read/write	0
0.13	Speed Selection (LSB)	Always returns a 0 for this bit. Together with bit 0.6, speed selection of 1000 Mb/s is identified.	returns 0	0
0.12	Auto-Negotiation Enable	1 = Enable SGMII Auto-Negotiation Process 0 = Disable SGMII Auto-Negotiation Process	read/write	1
0.11	Power Down	1 = Power down 0 = Normal operation When set to 1, the device-specific transceiver is placed in a low-power state. This bit requires a reset (see bit 0.15) to clear.	read/ write	0
0.10	Isolate	1 = Electrically Isolate SGMII logic from GMII 0 = Normal operation	read/write	1
0.9	Restart Auto-Negotiation	1 = Restart Auto-Negotiation Process across SGMII link 0 = Normal Operation	read/write self clearing	0
0.8	Duplex Mode	Always returns a 1 for this bit to signal Full-Duplex Mode.	returns 1	1

Table 2-26: **SGMII Control Register Channel/Module 0 (REGISTER 0) *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
0.7	Collision Test	Always returns a 0 for this bit to disable COL test.	returns 0	0
0.6	Speed Selection (MSB)	Always returns a 1 for this bit. Together with bit 0.13, speed selection of 1000 Mb/s is identified.	returns 1	1
0.5	Unidirectional al Enable	Enable transmit regardless of whether a valid link has been established provided AN is disabled.	read/write	0
0.4:0.0	Reserved	Always return 0s, writes ignored.	returns 0s	00000

Management Registers Channels/Modules 1-3

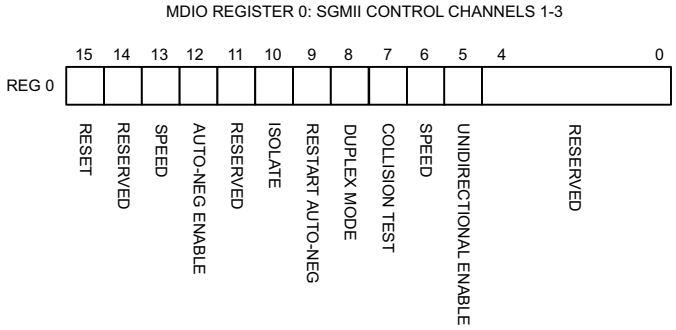

Figure 2-23: **MDIO Register 0: SGMII Control Channels/Modules 1-3**

Table 2-27: **SGMII Control Register Channels/Modules 1-3 (REGISTER 0)**

Bit(s)	Name	Description	Attributes	Default Value
0.15	Reset	1 = SGMII module 1-3 Reset 0 = Normal Operation	read/write self clearing	0
0.14	Reserved	Returns what is written	read/write	0
0.13	Speed Selection (LSB)	Always returns a 0 for this bit. Together with bit 0.6, speed selection of 1000 Mb/s is identified.	returns 0	0
0.12	Auto-Negotiation Enable	1 = Enable SGMII Auto-Negotiation Process 0 = Disable SGMII Auto-Negotiation Process	read/write	1
0.11	Reserved	Returns what is written	read/ write	0

Table 2-27: **SGMII Control Register Channels/Modules 1-3 (REGISTER 0) *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
0.10	Isolate	1 = Electrically Isolate SGMII logic from GMII 0 = Normal operation	read/write	1
0.9	Restart Auto-Negotiation	1 = Restart Auto-Negotiation Process across SGMII link 0 = Normal Operation	read/write self clearing	0
0.8	Duplex Mode	Always returns a 1 for this bit to signal Full-Duplex Mode	returns 1	1
0.7	Collision Test	Always returns a 0 for this bit to disable COL test	returns 0	0
0.6	Speed Selection (MSB)	Always returns a 1 for this bit. Together with bit 0.13, speed selection of 1000 Mb/s is identified.	returns 1	1
0.5	Unidirectional Enable	Enable transmit regardless of whether a valid link has been established provided AN is disabled	read/write	0
0.4:0.0	Reserved	Always return 0s, writes ignored	returns 0s	00000

Register 1: SGMII Status Register

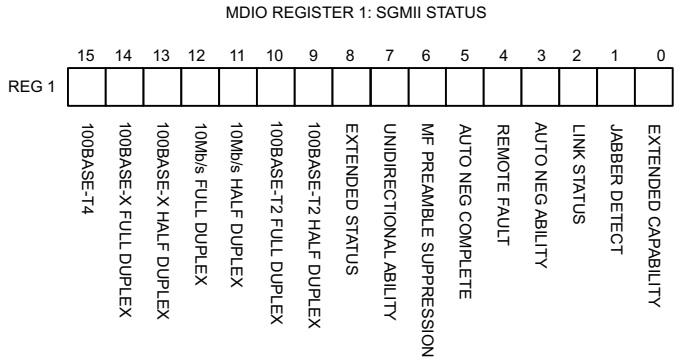

Figure 2-24: **MDIO Register 1: SGMII Status Register**

Table 2-28: **SGMII Status Register (REGISTER 1)**

Bit(s)	Name	Description	Attributes	Default Value
1.15	100BASE-T4	Always returns a 0 for this bit because 100BASE-T4 is not supported	returns 0	0
1.14	100BASE-X Full Duplex	Always returns a 0 for this bit because 100BASE-X Full Duplex is not supported	returns 0	0

Table 2-28: **SGMII Status Register (REGISTER 1) *(Cont'd)***

Bit(s)	Name	Description	Attributes	Default Value
1.13	100BASE-X Half Duplex	Always returns a 0 for this bit because 100BASE-X Half Duplex is not supported	returns 0	0
1.12	10 Mb/s Full Duplex	Always returns a 0 for this bit because 10 Mb/s Full Duplex is not supported	returns 0	0
1.11	10 Mb/s Half Duplex	Always returns a 0 for this bit because 10 Mb/s Half Duplex is not supported	returns 0	0
1.10	100BASE-T2 Full Duplex	Always returns a 0 for this bit because 100BASE-T2 Full Duplex is not supported	returns 0	0
1.9	100BASE-T2 Half Duplex	Always returns a 0 for this bit because 100BASE-T2 Half Duplex is not supported	returns 0	0
1.8	Extended Status	Always returns a 1 for this bit to indicate the presence of the Extended Register (Register 15)	returns 1	1
1.7	Unidirectional Ability	Always returns 1, writes ignored	returns 1	1
1.6	MF Preamble Suppression	Always returns a 1 for this bit to indicate that Management Frame Preamble Suppression is supported	returns 1	1
1.5	Auto-Negotiation Complete	1 = Auto-Negotiation process completed across SGMII link 0 = Auto-Negotiation process not completed across SGMII link	read only	0
1.4	Remote Fault	1 = A fault on the Medium has been detected 0 = No fault of the Medium has been detected	read only self clearing on read	0
1.3	Auto-Negotiation Ability	Always returns a 1 for this bit to indicate that the SGMII core is capable of Auto-Negotiation	returns 1	1
1.2	SGMII Link Status	1 = SGMII Link is up 0 = SGMII Link is down Latches 0 if the SGMII Link Status goes down. Clears to current SGMII Link Status on read. See the following Link Status section for further details.	read only self clearing on read	0
1.1	Jabber Detect	Always returns a 0 for this bit because Jabber Detect is not supported	returns 0	0
1.0	Extended Capability	Always returns a 0 for this bit because no extended register set is supported	returns 0	0

Link Status

When high, the link is valid and has remained valid after this register was last read; synchronization of the link has been obtained and Auto-Negotiation (if enabled) has completed.

When low, either:

- A valid link has not been established; link synchronization has failed or Auto-Negotiation (if enabled) has failed to complete.

  OR

- Link synchronization was lost at some point because this register was previously read. However, the current link status might be good. *Therefore read this register a second time to get confirmation of the current link status*.

Regardless of whether Auto-Negotiation is enabled or disabled, there can be some delay in the deassertion of Link Status following the loss of synchronization of a previously successful link. This is due to the Auto-Negotiation state machine which requires that synchronization is lost for an entire link timer duration before changing state. For more information, see the 802.3 specification (the *an_sync_status* variable).

Registers 2 and 3 (PHY IDENTIFIER)

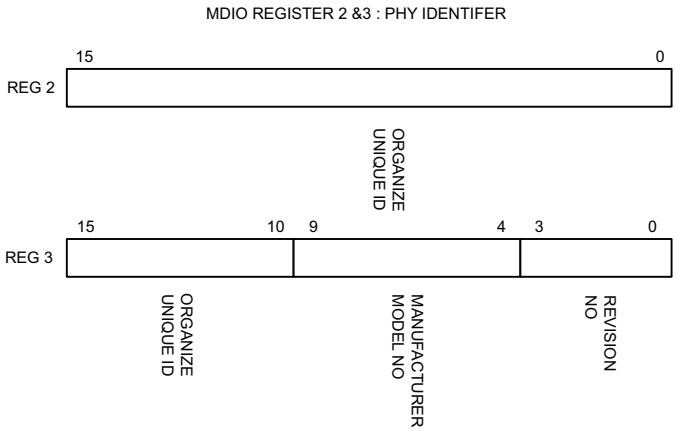

Figure 2-25: **MDIO Register 2 and 3: PHY Identifier**

Table 2-29: **PHY Identifier (Registers 2 and 3)**

Bit(s)	Name	Description	Attributes	Default Value
2.15:0	Organizationally Unique Identifier	Always return 0s	returns 0s	0000000000000000
3.15:10	Organizationally Unique Identifier	Always return 0s	returns 0s	000000
3.9:4	Manufacturer model number	Always return 0s	returns 0s	000000
3.3:0	Revision Number	Always return 0s	returns 0s	0000

Register 15: SGMII Extended Status

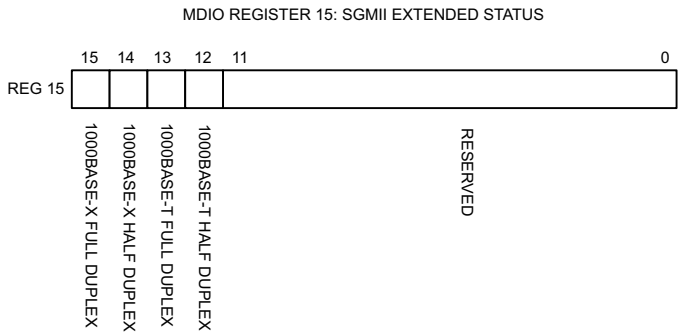

Figure 2-26: **MDIO Register 15: SGMII Extended Status**

Table 2-30: **SGMII Extended Status (Register 15)**

Bit(s)	Name	Description	Attributes	Default Value
15.15	1000BASE-X Full Duplex	Always returns a 1 for this bit because 1000BASE-X Full Duplex is supported	returns 1	1
15.14	1000BASE-X Half Duplex	Always returns a 0 for this bit because 1000BASE-X Half Duplex is not supported	returns 0	0
15.13	1000BASE-T Full Duplex	Always returns a 0 for this bit because 1000BASE-T Full Duplex is not supported	returns 0	0
15.12	1000BASE-T Half Duplex	Always returns a 0 for this bit because 1000BASE-T Half Duplex is not supported	returns 0	0
15:11:0	Reserved	Always return 0s	returns 0s	000000000000

Register 18: SGMII Generic Control (Register 18)

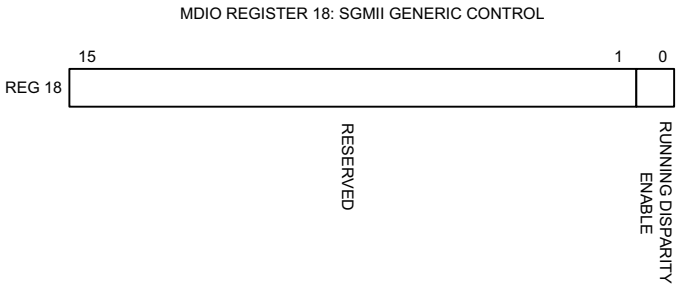

Figure 2-27: **MDIO Register 18: SGMII Generic Control**

Table 2-31: **SGMII Generic Control (Register 18)**

Bit(s)	Name	Description	Attributes	Default Value
18.15:1	Reserved	Always return 0s	returns 0s	000000000000000
18.0	Running Disparity Enable	1 =Running Disparity Checking enabled 0 = Running Disparity Checking disabled	read/ write	0

Designing with the Core

This chapter provides an introduction about creating your own designs using the QSGMII core.

Design Guidelines

Understand the Features and Interfaces Provided by the Core Netlist

Chapter 1, Overview introduces the features and Chapter 2, Core Interfaces and Management Register Space introduces the interfaces and registers that are present in the logic of the QSGMII netlist. This chapter assumes a working knowledge of the *IEEE 802.3-2008* Ethernet specification, in particular the Gigabit Ethernet 1000BASE-X sections: clauses 34 through to 37 and SGMII and QSGMII Cisco Specifications.

Customize and Generate the Core

Generate the core with your desired options using the CORE Generator™ tool, as described in Chapter 9, Customizing and Generating the Core or the Vivado™ IP Catalog as described in Chapter 6, Customizing and Generating the Core.

Examine the Example Design Provided with the Core

An HDL example design built around the core is provided through the CORE Generator and Vivado tools that allows for a demonstration of core functionality using either a simulation package or in hardware if placed on a suitable board.

Example designs are provided depending upon the core customization options that have been selected. See Example Design in Chapter 11.

Before implementing the core in your application, examine the example design provided with the core to identify the steps that can be performed:

1. Edit the HDL top level of the example design file to change the clocking scheme, add or remove IOBs as required, and replace the GMII IOB logic with user-specific application logic (for example, an Ethernet MAC).
2. Synthesize the entire design.
3. Implement the entire design.

   After implementation is complete you can also create a bitstream that can be downloaded to a Xilinx device.
4. Download the bitstream to a target device.

Implement the QSGMII Core in Your Application

Before implementing your application, examine the example design delivered with the core for information about the following:

- Instantiating the core from HDL
- Connecting the physical-side interface of the core
- Deriving the clock management logic

It is expected that the block-level module from the example design will be instantiated directly into customer designs rather than the core netlist itself. The block level contains the core and a completed physical interface.

Write an HDL Application

After reviewing the example design delivered with the core, write an HDL application that uses single or multiple instances of the block level module for the QSGMII core.

Synthesize your Design and Create a Bitstream

Synthesize your entire design using the desired synthesis tool. Care must be taken to constrain the design correctly; the constraints provided with the core should be used as the basis for your own. See the constraint chapters in either the Vivado Design Suite or ISE Design Suite sections as appropriate.

Simulate and Download your Design

After creating a bitstream that can be downloaded to a Xilinx device, simulate the entire design and download it to the desired device.

Know the Degree of Difficulty

An QSGMII core is challenging to implement in any technology and as such, all QSGMII core applications require 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 to increase system performance in an FPGA design, keep all inputs and outputs registered between the user application and the core. All inputs and outputs from the user 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 constraints provided with the example design for the core identifies the critical signals and the timing constraints that should be applied. See Chapter 7, Constraining the Core (Vivado) and Chapter 10, Constraining the Core (CORE Generator) for more information.

Use Supported Design Flows

The core is pre-synthesized and is delivered as an NGC netlist. The example implementation scripts provided currently uses ISE® v14.2 tools as the synthesis tool for the HDL example design delivered with the core. Other synthesis tools can be used for the user application logic. The core is always unknown to the synthesis tool and should appear as a black box. Post synthesis, only ISE v14.2 tools are supported.

Make Only Allowed Modifications

The QSGMII core should not be modified. Modifications can have adverse effects on system timing and protocol compliance. Supported user configurations of the QSGMII core can only be made by selecting the options from within the CORE Generator or Vivado tool when the core is generated.

- CORE Generator tool -- Chapter 9, Customizing and Generating the Core
- Vivado IP Catalog -- Chapter 6, Customizing and Generating the Core

Clocking

For clocking constraints see Chapter 10, Constraining the Core (CORE Generator tool) or Chapter 7, Constraining the Core (Vivado tool).

For clocking information on client interface, see Clock Generation Module in Chapter 4.

For clocking information on transceiver interface, see Chapter 5, Using the Transceiver.

Resets

Due to the number of clock domains in this IP core, the reset structure is not simple and involves a number of separate reset regions, with the number of regions being dependent upon the particular parameterization of the core. Figure 3-1 shows the most common reset structure for the core.

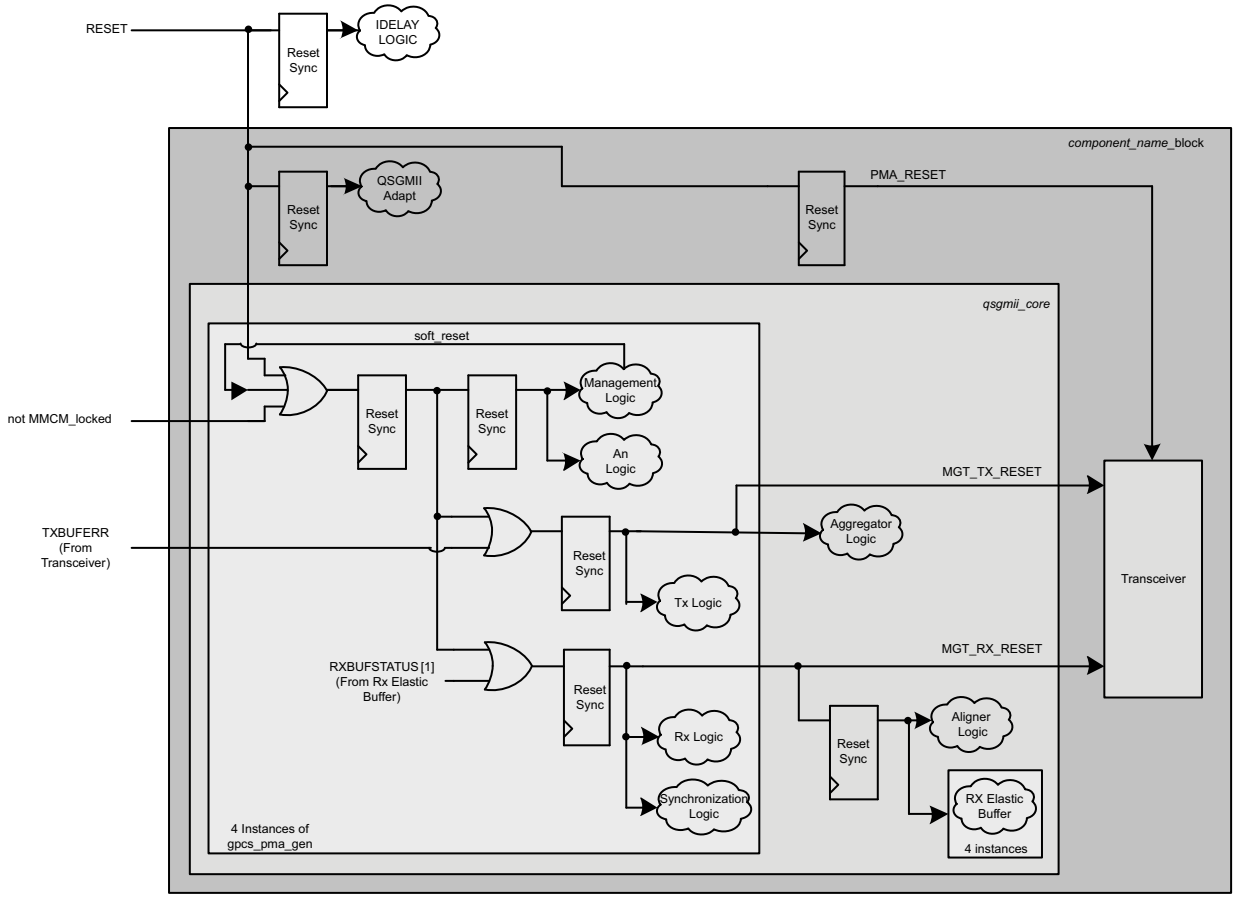

Figure 3-1: **Reset Structure**

Using the Client Side GMII/MII Datapath

This chapter provides general guidelines for using the client-side instances of GMII/MII interfaces of the QSGMII core. In most applications, the client-side GMII is expected to be used as an internal interface connecting to either:

- Proprietary customer logic

  This chapter describes the GMII-styled interface that is present on the netlist of the core.

  The chapter then also focuses on additional adaptation logic (which is provided by the example design delivered with the core). This logic enhances the internal GMII-styled interface to support 10 Mb/s and 100 Mb/s Ethernet speeds in addition to the nominal 1 Gb/s speed of SGMII.

- The Xilinx LogiCORE™ IP Tri-Mode Ethernet MAC

  The QSGMII core can be integrated in a single device with multiple instances of the Tri-Mode Ethernet MAC core to extend the system functionality to include the MAC sublayer. See "Chapter 13, Interfacing to Other Cores" in the *LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII Product Guide* (PG047).

In rare applications, the Client-Side GMII datapath can be used as a true GMII/MII to connect externally off-chip across a Printed Circuit Board (PCB). This external GMII functionality is included in the HDL example design delivered with the core by the ISE® or Vivado™ tools to act as an illustration. The extra logic required to create a true external GMII is detailed in Appendix C, Implementing External GMII/MII.

Using the Core Netlist Client-Side GMII/MII

GMII Transmission

This section includes figures that illustrate GMII transmission. In these figures the clock is not labeled. The source of this clock signal varies, depending on the options selected when the core is generated.

Normal Frame Transmission

Normal outbound frame transfer timing is illustrated in Figure 4-1. This figure shows that an Ethernet frame is proceeded by an 8-byte preamble field (inclusive of the Start of Frame Delimiter (SFD), and completed with a 4-byte Frame Check Sequence (FCS) field. This frame is created by the MAC connected to the other end of the GMII. The PCS logic itself does not recognize the different fields within a frame and treats any value placed on `gmii_txd_chx[7:0]` within the `gmii_tx_en_chx` assertion window as data.

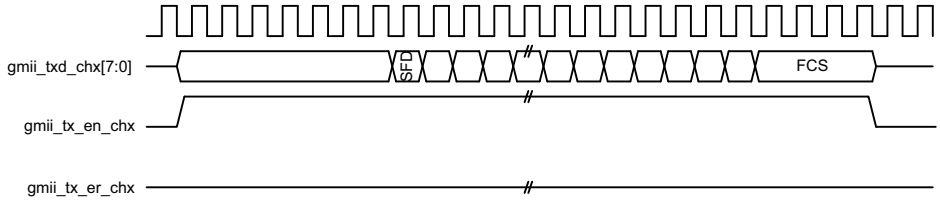

Figure 4-1: **GMII Normal Frame Transmission**

Error Propagation

A corrupted frame transfer is illustrated in Figure 4-2. An error can be injected into the frame by asserting `gmii_tx_er_chx` at any point during the `gmii_tx_en_chx` assertion window. The core ensures that all errors are propagated through both transmit and receive paths so that the error is eventually detected by the link partner.

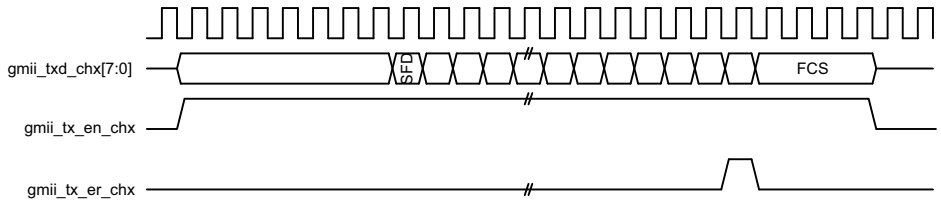

Figure 4-2: **GMII Error Propagation Within a Frame**

GMII Reception

This section includes figures that illustrate GMII reception. In these figures the clock is not labeled. The source of this clock signal varies, depending on the options used when the core is generated.

Normal Frame Reception

The timing of normal inbound frame transfer is illustrated in Figure 4-3. This shows that Ethernet frame reception is proceeded by a preamble field. The *IEEE 802.3-2008* specification (see clause 35) allows for up to all of the seven preamble bytes that proceed the Start of Frame Delimiter (SFD) to be lost in the network. The SFD is always present in well-formed frames.

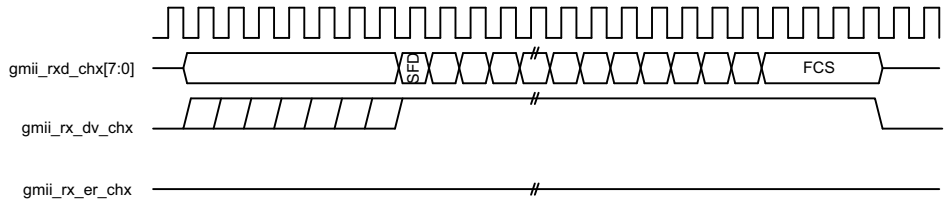

Figure 4-3: **GMII Normal Frame Reception**

Normal Frame Reception with Extension Field

In accordance with the *IEEE 802.3-2008*, clause 36, state machines for the 1000BASE-X PCS, `gmii_rx_er_chx` can be driven high following reception of the end frame in conjunction with `gmii_rxd_chx[7:0]` containing the hexadecimal value of 0x0F to signal carrier extension. This is illustrated in Figure 4-4.

This is not an error condition and can occur even for full-duplex frames.

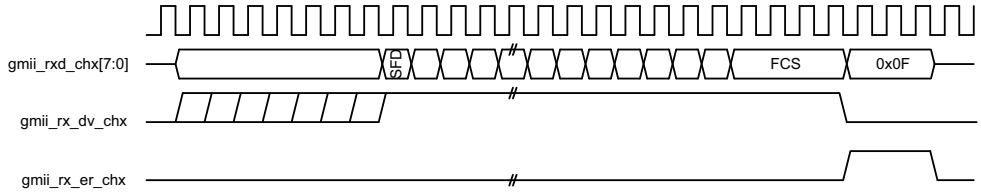

Figure 4-4: **GMII Normal Frame Reception WIth Carrier Extension**

Frame Reception with Errors

The signal `gmii_rx_er_chx` when asserted within the assertion window signals that a frame was received with a detected error (Figure 4-5). In addition, a late error can also be detected during the Carrier Extension interval. This is indicated by `gmii_rxd_chx[7:0]` containing the hexadecimal value 0x1F, also illustrated in Figure 4-5.

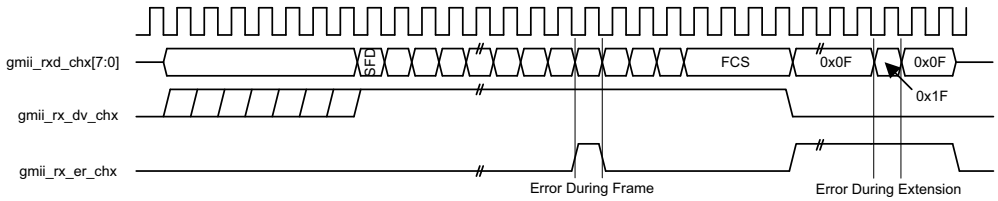

Figure 4-5: **GMII Frame Reception WIth Errors**

MII Transmission

100 Mb/s Frame Transmission

The operation of the core remains unchanged. It is the responsibility of the client logic (for example, an Ethernet MAC) to enter data at the correct rate. When operating at a speed of 100 Mb/s, every byte of the MAC frame (from preamble field to the Frame Check Sequence field, inclusive) should each be repeated for 10 clock periods to achieve the desired bit rate, as illustrated in Figure 4-6. It is also the responsibility of the client logic to ensure that the interframe gap period is legal for the current speed of operation.

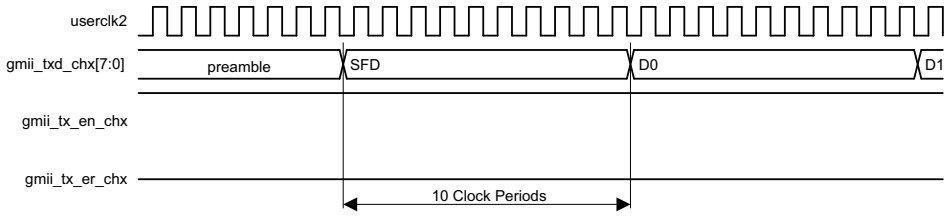

Figure 4-6: **100 Mb/s Frame Transmission**

10 Mb/s Frame Transmission

The operation of the core remains unchanged. It is the responsibility of the client logic (for example, an Ethernet MAC), to enter data at the correct rate. When operating at a speed of 10 Mb/s, every byte of the MAC frame (from preamble to the frame check sequence field, inclusive) should each be repeated for 100 clock periods to achieve the desired bit rate. It is also the responsibility of the client logic to ensure that the interframe gap period is legal for the current speed of operation.

MII Reception

100 Mb/s Frame Reception

The operation of the core remains unchanged. When operating at a speed of 100 Mb/s, every byte of the MAC frame (from preamble to the frame check sequence field, inclusive) is repeated for 10 clock periods to achieve the desired bit rate. See Figure 4-7. It is the responsibility of the client logic, for example an Ethernet MAC, to sample this data correctly.

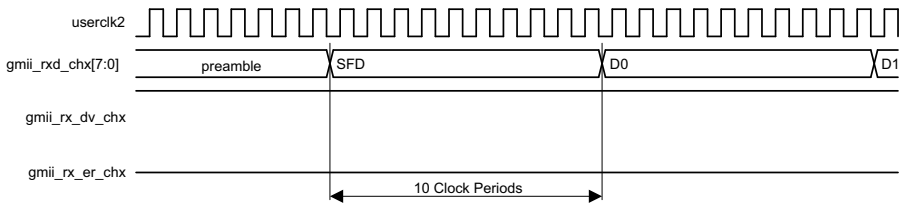

Figure 4-7: **100 Mb/s Frame Reception**

10 Mb/s Frame Reception

The operation of the core remains unchanged. When operating at a speed of 10 Mb/s, every byte of the MAC frame (from preamble to the frame check sequence field, inclusive) is repeated for 100 clock periods to achieve the desired bit rate. It is the responsibility of the client logic (for example, an Ethernet MAC) to sample this data correctly.

Additional Client-Side QSGMII Adaptation Logic Provided in the Example Design

The block level of the core contains the *QSGMII Adaptation Module* (this is illustrated in Figure 4-8). This QSGMII adaptation module is described in the remainder of this section.

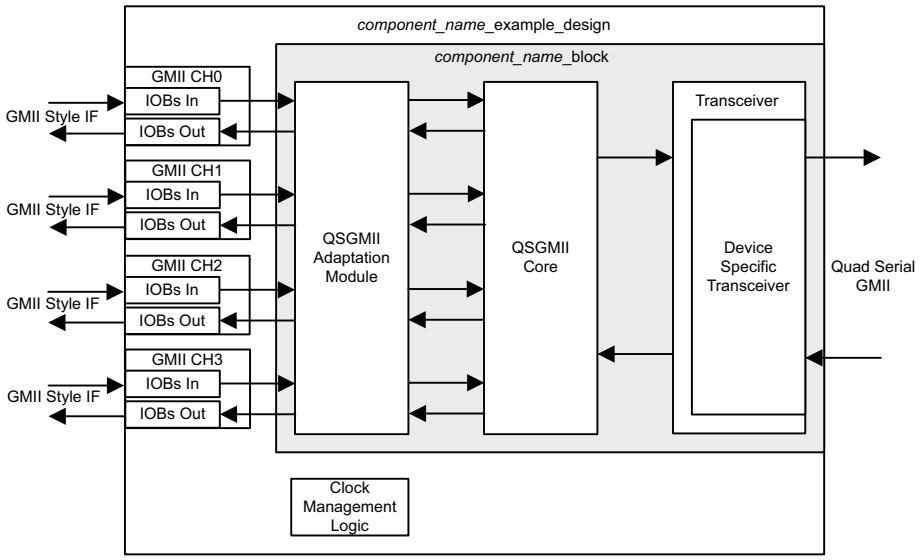

Figure 4-8: **Example Design HDL for QSGMII**

Because the GMII of the core always operates at 125 MHz, the core makes no differentiation between the three QSGMII speeds of operation. It always effectively operates at 1 Gb/s. However, as described previously in Using the Core Netlist Client-Side GMII/MII, at 100 Mb/s, every data byte run through the core is repeated ten times to achieve the required bit rate; at 10 Mb/s, each data byte run through the core is repeated 100 times to achieve the required bit rate. Dealing with this repetition of bytes is the function of the QSGMII adaptation module. In addition in PHY mode and operating in MII, the QSGMII adaptation module is also responsible for conversion of the 4-bit MII interface to an 8-bit GMII core interface.

The provided QSGMII adaptation module (Figure 4-9) creates a GMII interface that drives/samples the GMII data and control signals at the following frequencies:

- 125 MHz when operating at a speed of 1 Gb/s (with no repetition of data bytes).
- 12.5 MHz at a speed of 100 Mb/s (each data byte is repeated and run through the core 10 times).
- 1.25 MHz at a speed of 10 Mb/s (each data byte is repeated and run through the core 100 times).

Therefore, the result of the QSGMII adaptation module is to create a proprietary interface that is based on GMII (true GMII only operates at a clock frequency of 125 MHz). This interface then allows a straightforward internal connection to an Ethernet MAC core when operating in MAC mode or the GMII can be brought out on pads to connect to an external PHY when the core operates in PHY mode. For example, the QSGMII adaptation module can be used to interface to the QSGMII core, operating in MAC, to four instances of the Xilinx Tri-Mode Ethernet MAC core directly. The GMII interface of the QSGMII adaptation module can brought out to the pads and connected to an external PHY module that converts GMII to PMD signals when the QSGMII core is operating in PHY mode.

QSGMII Adaptation Module Top Level

The QSGMII adaptation module contains four instances of the SGMII adaptation module with each instance corresponding to one channel as shown in Figure 4-9.

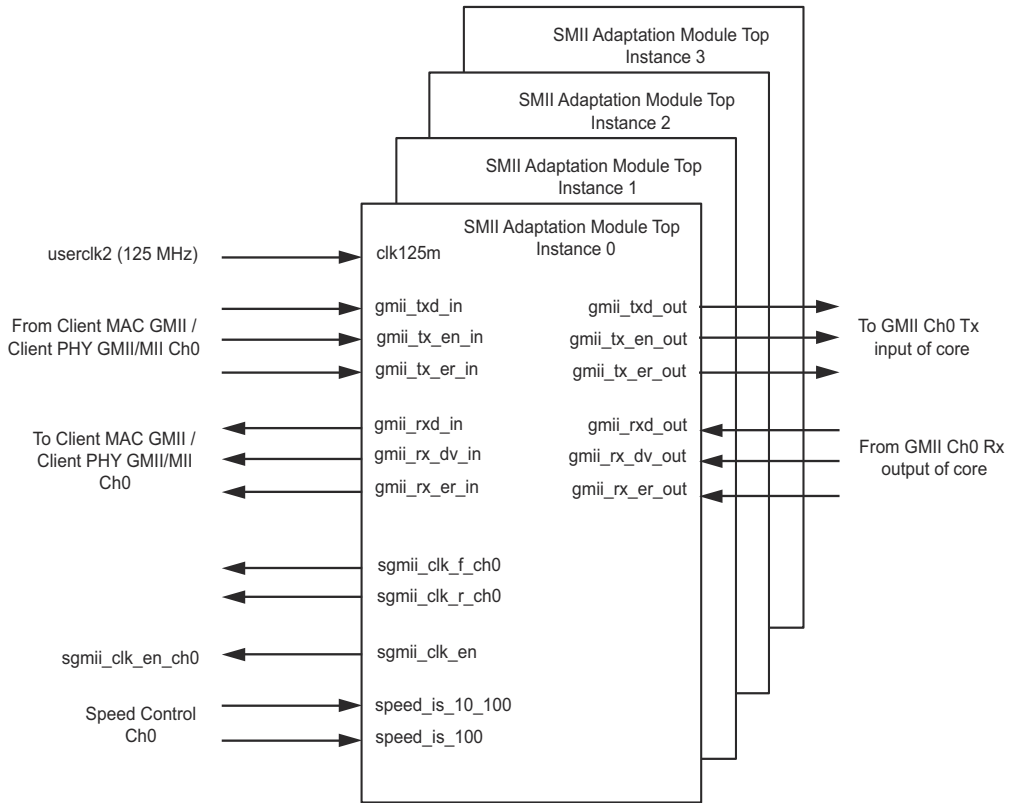

Figure 4-9: **QSGMII Adaptation Module**

SGMII Adaptation Module Top Level

The SGMII adaptation module is described in several hierarchical submodules as illustrated in Figure 4-10. These submodules are each described in separate HDL files and are described in the following sections.

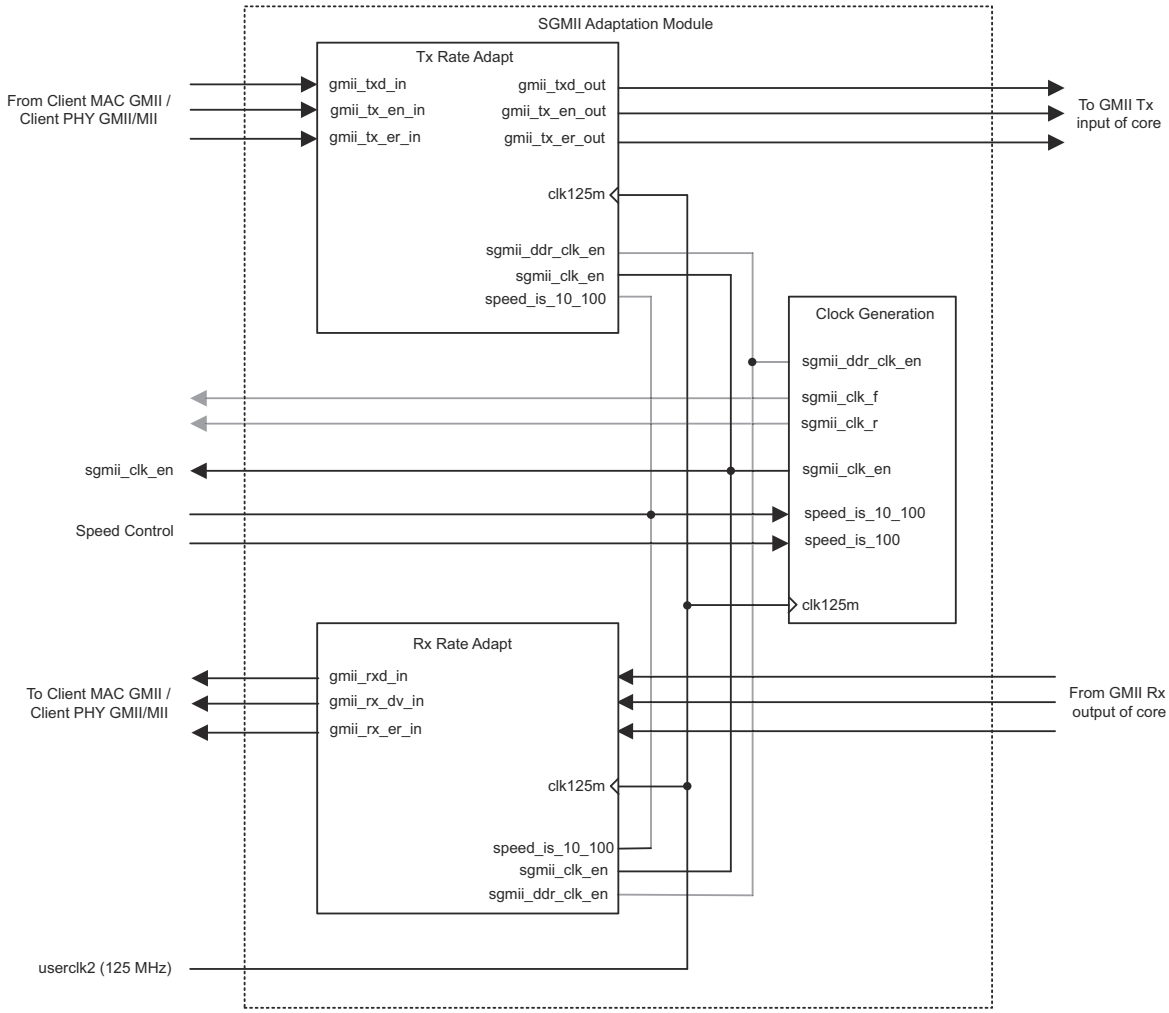

Figure 4-10: **SGMII Adaptation Module**

Transmitter Rate Adaptation Module

QSGMII Operating in MAC Mode

This module accepts transmitter data from the GMII-style interface from the attached client MAC and samples the input data on the 125 MHz reference clock, `clk125m`. This sampled data can then be connected directly to the input GMII instances of the QSGMII netlist. The 1 Gb/s and 100 Mb/s cases are illustrated in Figure 4-11.

At all speeds, the client MAC logic should drive the GMII transmitter data synchronously to the rising edge of the 125 MHz reference clock while using `sgmii_clk_en` (derived from the Clock Generation module) as a clock enable. The frequency of this clock enable signal ensures the correct data rate and correct data sampling between the two devices.

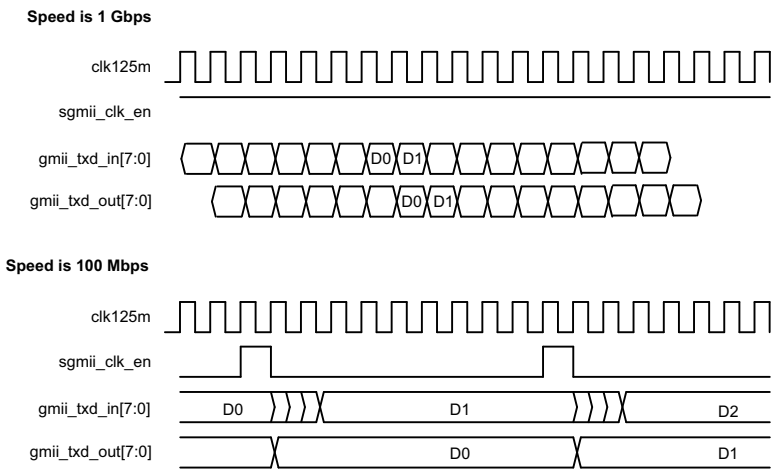

Figure 4-11: **Transmitter Rate Adaptation Module Sampling in MAC Mode**

QSGMII Operating in PHY Mode

QSGMII in PHY mode follows the true GMII/MII interface. When the GMII interface is selected, the data is received on the 125 MHz clock (`clk125m`). When the MII interface is selected, 4 bits of MII are received on the LSB 4 bits of the GMII interface. This interface is converted to 8 bits by sampling with `sgmii_ddr_clk_en` (derived from the Clock Generation module).

This 8-bit interface should drive the GMII transmitter data synchronously to the rising edge of the 125 MHz reference clock while using `sgmii_clk_en` (derived from the Clock Generation module) as a clock enable.

It is possible that the SFD could have been skewed across two separate bytes, so 8-bit Start of Frame Delimiter (SFD) code is detected, and if required, it is realigned across the 8-bit datapath

The 1 Gb/s and 100 Mb/s cases are illustrated in Figure 4-12.

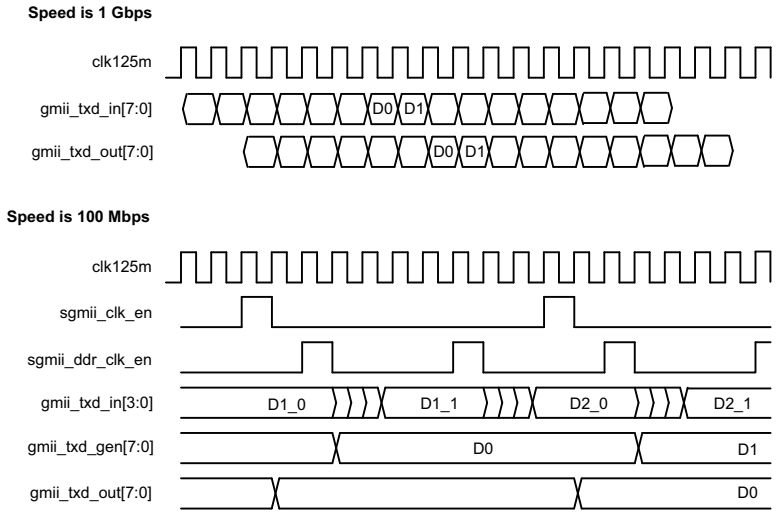

Figure 4-12: **Transmitter Rate Adaptation Module Sampling in PHY Mode**

Receiver Rate Adaptation Module

QSGMII Operating in MAC mode

This module accepts received data from the QSGMII core. This data is sampled and sent out of the GMII receiver interface for the attached client MAC. The 1 Gb/s and 100 Mb/s cases are illustrated in Figure 4-13.

At 1 Gb/s, the data is valid on every clock cycle of the 125 MHz reference clock (`clk125m`). Data received from the core is clocked straight through the Receiver Rate Adaptation module.

At 100 Mb/s, the data is repeated for a 10 clock period duration of `clk125m`; at 10 Mb/s, the data is repeated for a 100 clock period duration of `clk125m`. The Receiver Rate Adaptation Module samples this data using the `sgmii_clk_en` clock enable.

The Receiver Rate Adaptation module also performs a second function that accounts for the latency inferred in Figure 4-13. The 8-bit Start of Frame Delimiter (SFD) code is detected, and if required, it is realigned across the 8-bit datapath of `gmii_rxd_out[7:0]` before being presented to the attached client MAC. It is possible that this SFD could have been skewed across two separate bytes by MACs operating on a 4-bit datapath.

At all speeds, the client MAC logic should sample the GMII receiver data synchronously to the rising edge of the 125 MHz reference clock while using `sgmii_clk_en` (derived from the Clock Generation module) as a clock enable. The frequency of the `sgmii_clk_en` clock enable signal ensures the correct data rate and correct data sampling between the two devices.

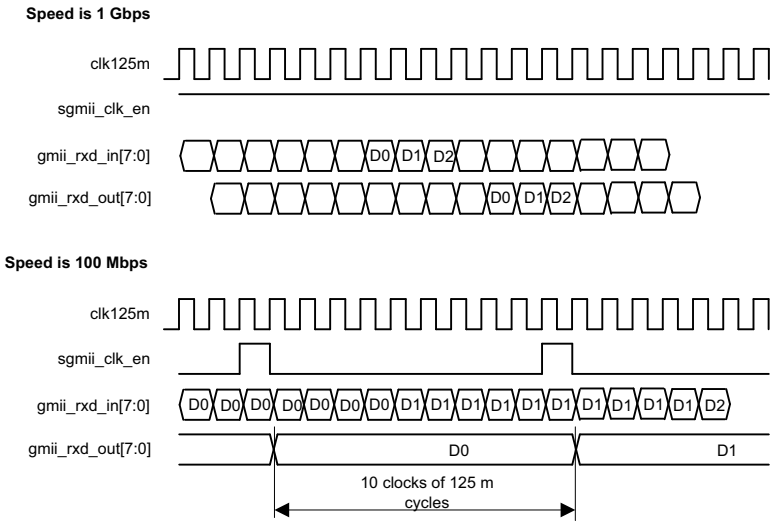

Figure 4-13: **Receiver Rate Adaptation Module Sampling in MAC Mode**

QSGMII Operating in PHY Mode

This module accepts received data from the QSGMII core. This data is sampled and sent out of the GMII receiver interface for the attached external PHY. The 1 Gb/s and 100 Mb/s cases are illustrated in Figure 4-14.

At 1 Gb/s the data is valid on every clock cycle of the 125 MHz reference clock (`clk125m`). Data received from the core is clocked straight through the Receiver Rate Adaptation module.

At 100 Mb/s, the data is repeated for a 10 clock period duration of `clk125m`; at 10 Mb/s, the data is repeated for a 100 clock period duration of `clk125m`. The Receiver Rate Adaptation Module samples this data using the `sgmii_clk_en` clock enable. Then the lower half of the byte is sent on the LSB 4 bits of `gmii_rxd_out[3:0]` followed by the upper nibble. This operation is done on `sgmii_ddr_clk_en`.

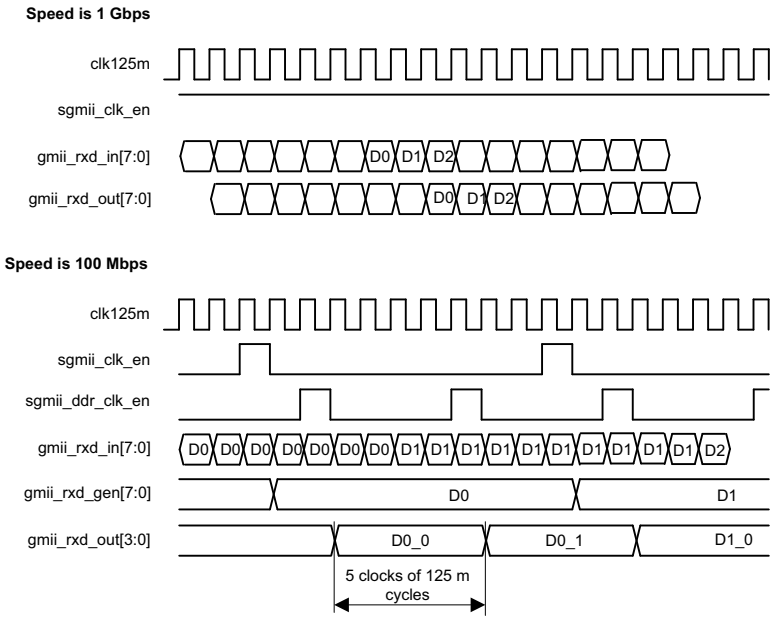

Figure 4-14: **Receiver Rate Adaptation Module Sampling in PHY Mode**

Clock Generation Module

This module creates the `sgmii_clk_en` clock enable signal for use throughout the SGMII adaptation module.

Clock enabled frequencies are

- 125 MHz at an operating speed of 1 Gb/s
- 12.5 MHz at an operating speed of 100 Mb/s
- 1.25 MHz at an operating speed of 10 Mb/s

This module also creates output clock `sgmii_clk_chx` from rise and fall clocks.

Clock generation rise and fall frequencies are

- 125 MHz at an operating speed of 1 Gb/s in PHY mode and for all speeds in MAC mode
- 25 MHz at an operating speed of 100 Mb/s
- 2.5 MHz at an operating speed of 10 Mb/s

Figure 4-15 illustrates the output clock enable signal for the Clock Generation module at 1 Gb/s and 100 Mb/s speeds.

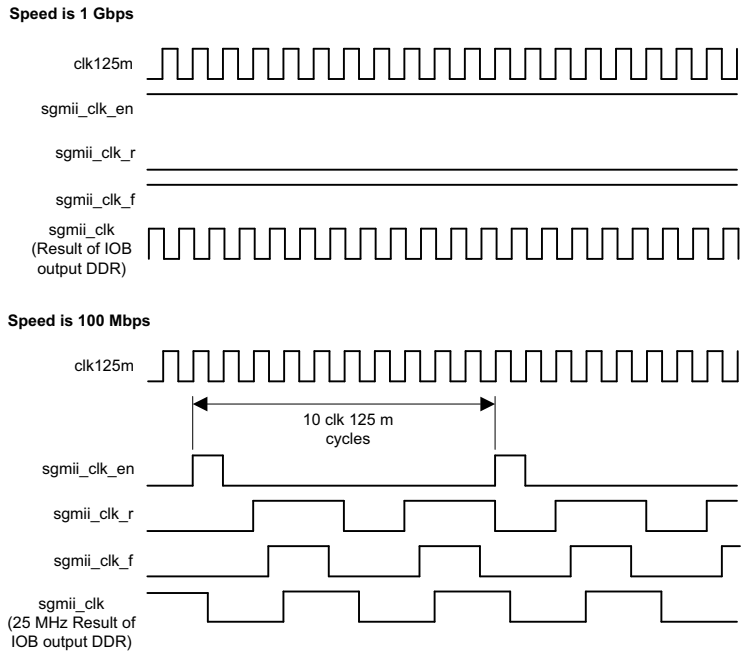

Figure 4-15: **Clock Generator Output Clock and Clock Enables**

Figure 4-15 also illustrates the formation of the `sgmii_clk_r` and `sgmii_clk_f` signals. These are used only in the example design delivered with the core, where they are routed to a device IOB DDR output register. This provides SGMII clock forwarding at the correct frequency.

Using the Transceiver

This chapter provides general guidelines for using transceivers with Virtex®-7, Kintex™-7 or Artix™-7 devices.

This chapter is organized into the following main sections, with each section being organized into FPGA families:

- Transceiver Logic

  Providing a more detailed look at the device-specific transceivers and their connections to the netlist of the core.

- Clock Sharing Across Multiple Cores with Transceivers

  Providing guidance for using several cores and transceiver instantiations; clock sharing should occur whenever possible to save device resources.

Transceiver Logic

The example is split between two discrete hierarchical layers, as illustrated in Figure 5-1. The block level is designed so that it can be instantiated directly into customer designs and provides the following functionality:

- Instantiates the core from HDL.
- Connects the client interface through the QSGMII adaptation module. See Chapter 4, Using the Client Side GMII/MII Datapath.
- Connects the physical-side interface of the core to a Virtex-7, Kintex-7 or Artix-7 FPGA transceiver.

The logic implemented in the block level for the physical-side interface of the core is illustrated in all the figures and described in further detail for the remainder of this chapter.

Virtex-7 Devices

The core is designed to integrate with the 7 series FPGA transceiver. Figure 5-1 illustrates the connections and logic required between the core and the transceiver; the signal names and logic in the figure precisely match those delivered with the example design when a 7 series FPGA transceiver is used.

The 125 MHz differential reference clock is routed directly to the 7 series FPGA transceiver. The transceiver is configured to output a version of this clock (125 MHz) on the `TXOUTCLK` port; this is then placed onto global clock routing and is input back into the transceiver on the user interface clock ports `txusrclk` and `txusrclk2`. This clock is also used to source for all core logic.

The transceiver is configured to output a recovered clock (125 MHz) on the `RXOUTCLK` port; this is placed onto regional routing through BUFMR and BUFR. This clock is then used to source the receive logic from the transceiver receive side output to the rxelastic buffer in the core.

See also Virtex-7 FPGA GTX Transceivers in Chapter 10 for constraints.

7 Series FPGA Transceiver Wizard

The two wrapper files immediately around the transceiver pair, gtwizard and gtwizard_gt (Figure 5-1), are generated from the 7 series FPGA transceiver wizard. These files apply all the QSGMII attributes. Consequently, these files can be regenerated by customers. The CORE Generator™ tool log file (XCO file) that was created when the 7 series FPGA transceiver wizard project was generated is available in the following location:

ISE® Design Suite

```
<project_dir>/<component_name>/example_design/transceiver/<component_name>_gtwizard
.xco
```

Vivado™ Design Suite

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
<component_name>/example_design/transceiver/<component_name>_gtwizard.xco
```

This file can be used as an input to the CORE Generator tool to regenerate the device-specific transceiver wrapper files. This file can be used as an input to Vivado tools project by clicking on <Add Sources> in the Flow Navigator task bar and selecting the XCO file.

The XCO file itself contains a list of all of the transceiver wizard attributes that were used. For further information, see the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476).

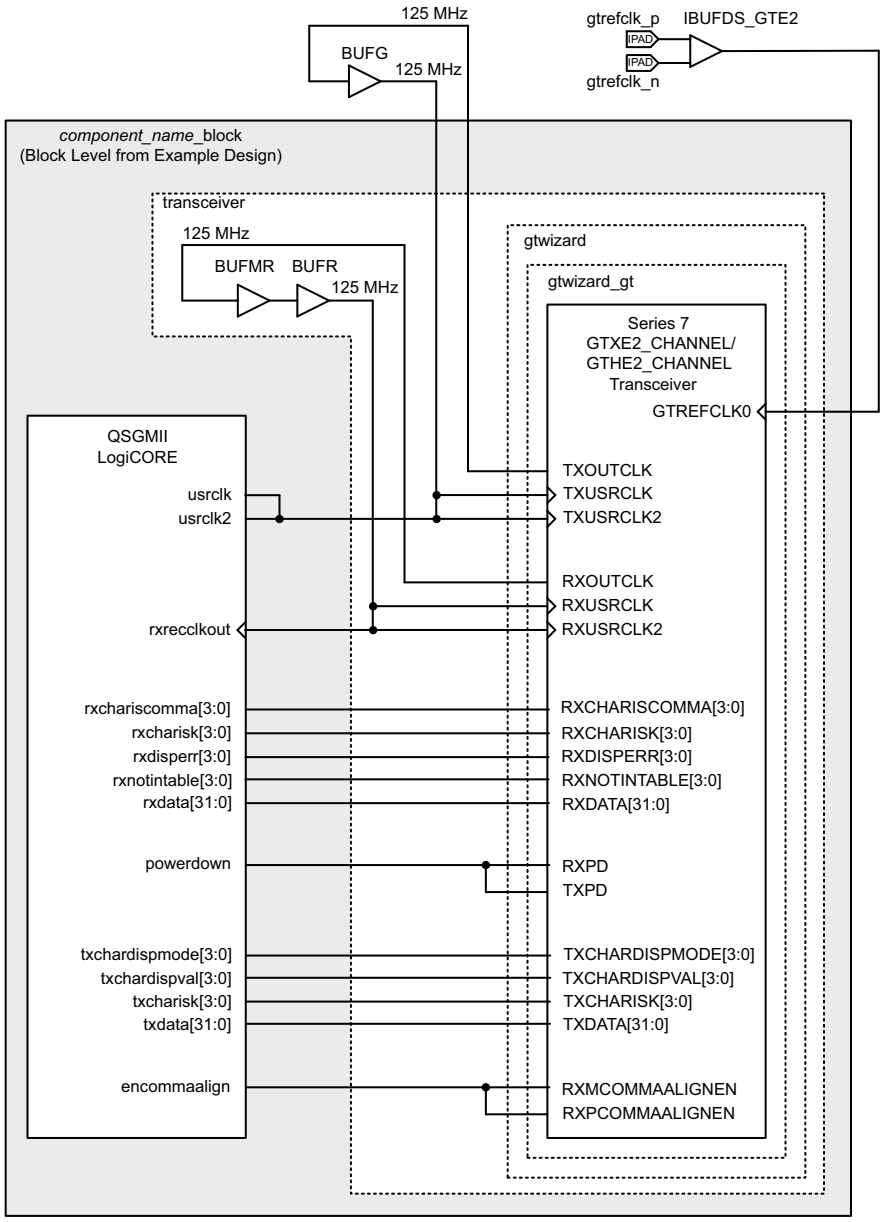

Figure 5-1: **QSGMII Connection to Virtex-7 FPGA Transceivers**

Kintex-7 Devices

The core is designed to integrate with the 7 series FPGA transceiver. Figure 5-2 illustrates the connections and logic required between the core and the transceiver; the signal names and logic in the figure precisely match those delivered with the example design when a 7 series FPGA transceiver is used.

The 125 MHz differential reference clock is routed directly to the 7 series transceiver. The transceiver is configured to output a version of this clock (125 MHz) on the `TXOUTCLK` port; this is then placed onto global clock routing and is input back into the GTXE2 transceiver on the user interface clock ports `txusrclk` and `txusrclk2`. This clock is also used to source for all core logic.

The transceiver is configured to output a recovered clock (125 MHz) on the `RXOUTCLK` port; this is placed onto global routing through BUFG. This clock is then used to source the receive logic from Transceiver receive side output to the rxelastic buffer in the core

See also Kintex-7 FPGA GTX Transceivers in Chapter 10 for constraints.

7 Series FPGA Transceiver Wizard

The two wrapper files immediately around the GTX transceiver pair, gtwizard and gtwizard_gt (Figure 5-2), are generated from the 7 series FPGA transceiver wizard. These files apply all the QSGMII attributes. Consequently, these files can be regenerated by customers. The CORE Generator tool log file (XCO file) that was created when the 7 series FPGA transceiver wizard project was generated is available in the following location:

ISE Design Suite

```
<project_dir>/<component_name>/example_design/transceiver/<component_name>_gtwizard
.xco
```

Vivado Design Suite

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
<component_name>/example_design/transceiver/<component_name>_gtwizard.xco
```

This file can be used as an input to the CORE Generator tool to regenerate the device-specific transceiver wrapper files. This file can be used as an input to Vivado project by clicking on <Add Sources> in the Flow Navigator task bar and selecting the XCO file. The XCO file itself contains a list of all of the transceiver wizard attributes that were used. For further information, see the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476).

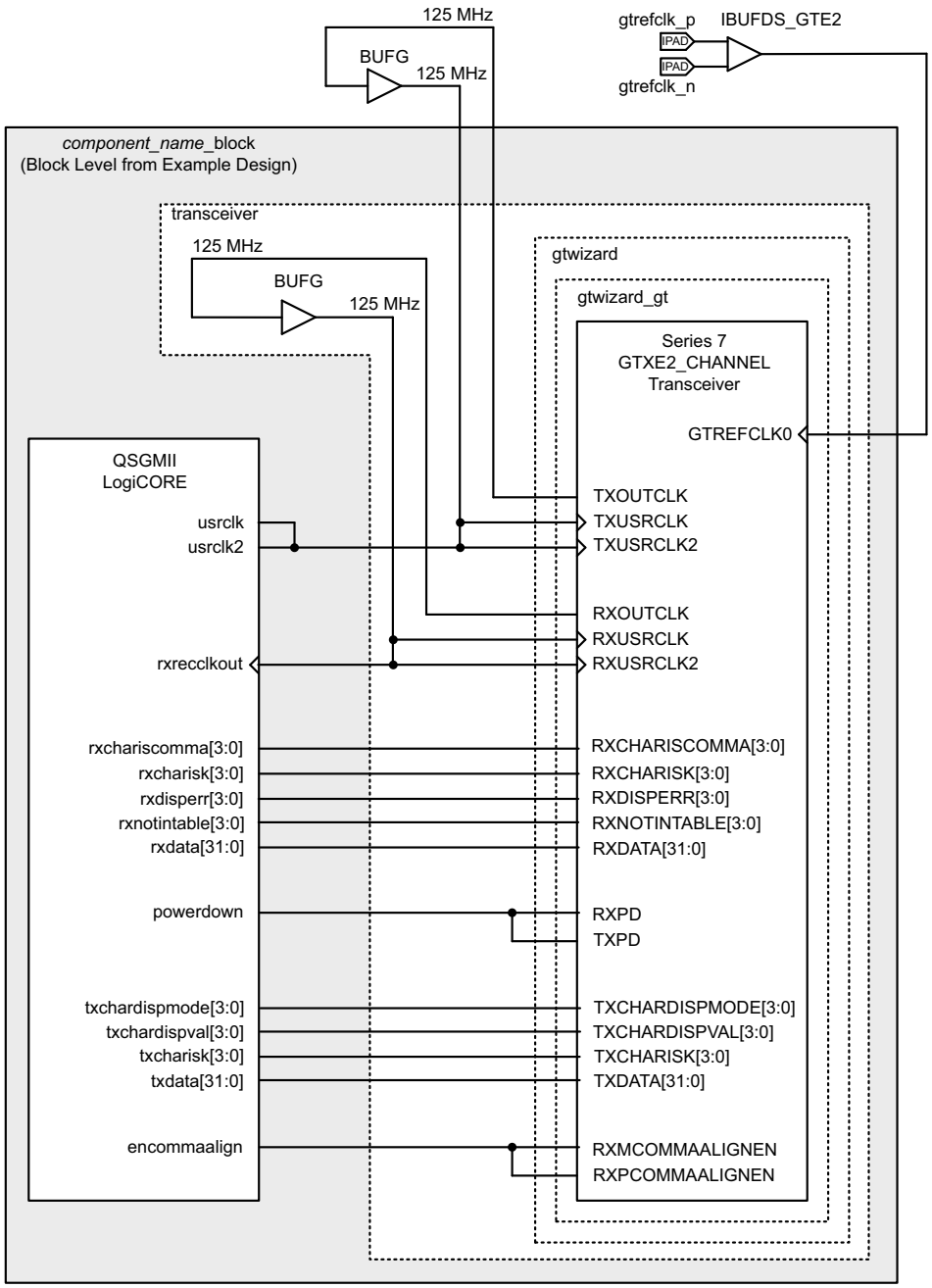

Figure 5-2: **QSGMII Connection to Kintex-7 FPGA Transceivers**

Artix-7 Devices

The core is designed to integrate with the 7 series FPGA transceiver. Figure 5-3 illustrates the connections and logic required between the core and the transceiver; the signal names and logic in the figure precisely match those delivered with the example design when a 7 series FPGA transceiver is used.

The 125 MHz differential reference clock is routed directly to the 7 series transceiver. The transceiver is configured to output a version of this clock (125 MHz) on the `TXOUTCLK` port; this is then placed onto global clock routing and passed to a MMCM which generates two clocks; one 125 MHz (`userclk2`) and other 250 MHz (`userclk`). `Userclk` and `userclk2` are input back into the GTXE2 transceiver on the user interface clock ports `txusrclk` and `txusrclk2`. `Userclk2` clock is also used to source for all core logic.

The transceiver is configured to output a recovered clock (250 MHz) on the `RXOUTCLK` port. This clock is divided down to 125 MHz using BUFR and is then used to source the receive logic from Transceiver receive side rxelastic buffer read in the core and `RXUSRCLK2` port of transceiver. The 250 MHz `RXOUTCLK` is placed on regional clock routing using BUFR and is routed back to `RXUSERCLK` port of transceiver.

See also Artix-7 FPGA GTP Transceivers in Chapter 10 for constraints.

7 Series FPGA Transceiver Wizard

The two wrapper files immediately around the GTX transceiver pair, gtwizard and gtwizard_gt (Figure 5-3), are generated from the 7 series FPGA transceiver wizard. These files apply all the QSGMII attributes. Consequently, these files can be regenerated by customers. The CORE Generator tool log file (XCO file) that was created when the 7 series FPGA transceiver wizard project was generated is available in the following location:

ISE Design Suite

```
<project_dir>/<component_name>/example_design/transceiver/<component_name>_gtwizard
.xco
```

Vivado Design Suite

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
<component_name>/example_design/transceiver/<component_name>_gtwizard.xco
```

This file can be used as an input to the CORE Generator tool to regenerate the device specific transceiver wrapper files. This file can be used as an input to Vivado tools project by clicking on <Add Sources> in the Flow Navigator task bar and selecting the XCO file.The XCO file itself contains a list of all of the transceiver wizard attributes that were used. For further information, see the 7 *Series FPGAs GTX/GTH Transceivers User Guide* (UG476).

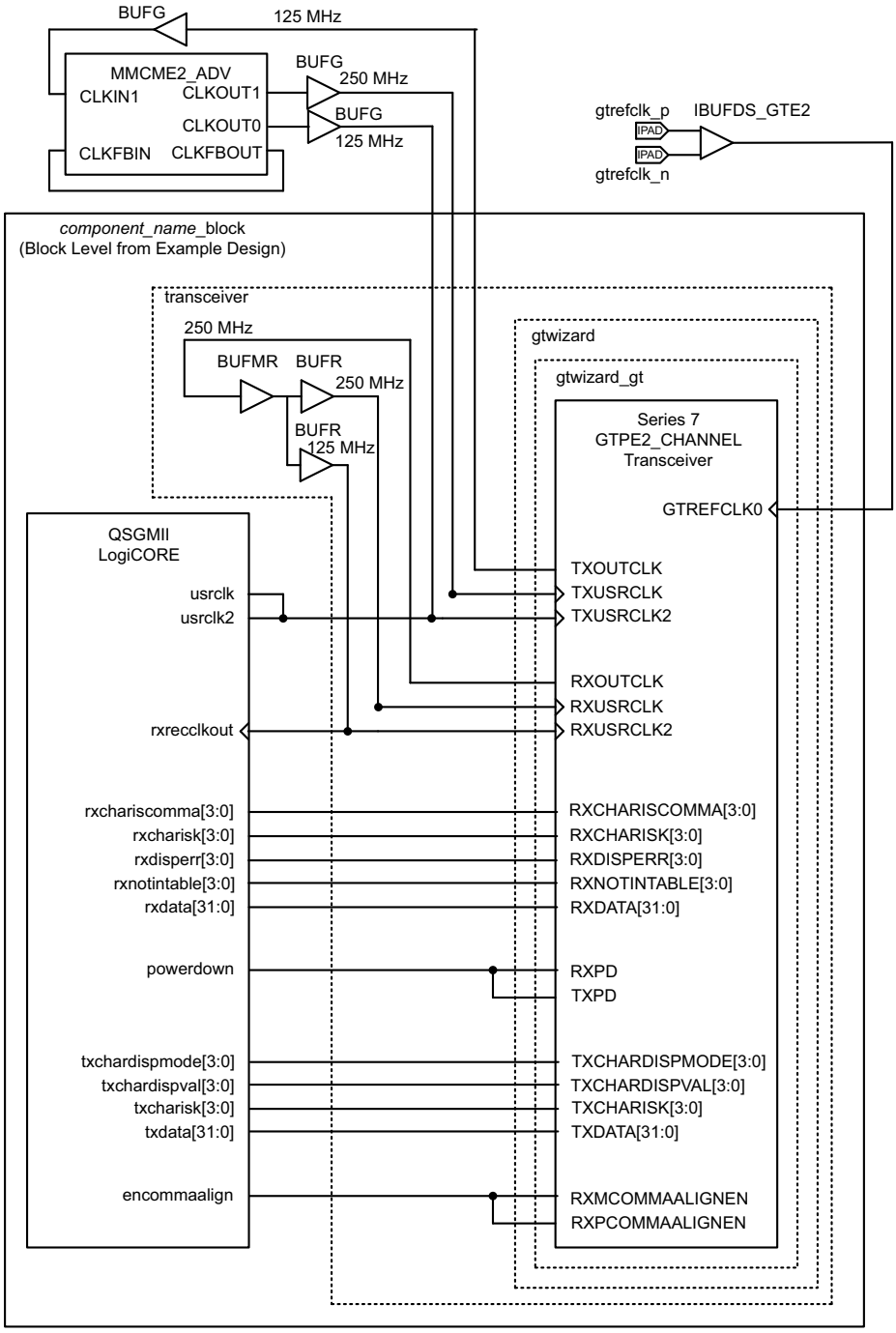

Figure 5-3: **QSGMII Connection to Artix-7 FPGA Transceivers**

Clock Sharing Across Multiple Cores with Transceivers

Virtex-7 Devices

Figure 5-4 illustrates sharing clock resources across two instantiations of the core when using 7 series FPGA transceivers. Additional cores can be added by continuing to instantiate extra block level modules.

To provide the FPGA logic clocks for all core instances, select a `TXOUTCLK` port from any transceiver and place it onto global clock routing using BUFGs; it can be shared across all core instances and transceivers as illustrated.

Each transceiver and core pair instantiated has its own independent clock domains synchronous to `RXOUTCLK`. These are placed on BUFMR followed by regional clock routing using a BUFR, as illustrated in Figure 5-4, and cannot be shared across multiple transceivers.

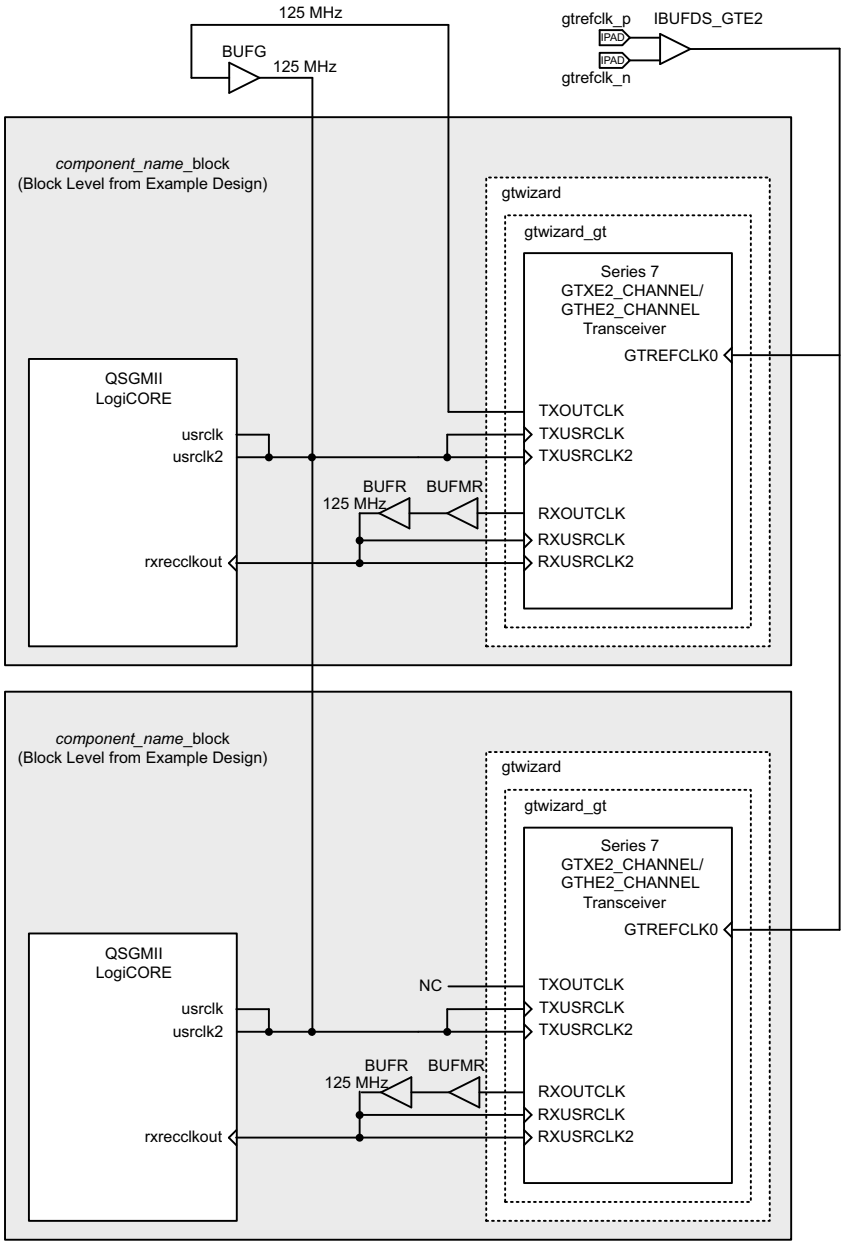

Figure 5-4: **Clock Management with Multiple Core Instances with Virtex-7 FPGA Transceivers**

Kintex-7 Devices

Figure 5-5 illustrates sharing clock resources across two instantiations of the core when using 7 series FPGAs transceivers. Additional cores can be added by continuing to instantiate extra block level modules.

To provide the FPGA logic clocks for all core instances, select a `TXOUTCLK` port from any transceiver and place it onto global clock routing using BUFGs; these can be shared across all core instances and transceivers as illustrated,

Each GTX transceiver and core pair instantiated has its own independent clock domains synchronous to `RXOUTCLK`. These are placed on global clock routing using a BUFG, as illustrated in Figure 5-5, and cannot be shared across multiple transceivers.

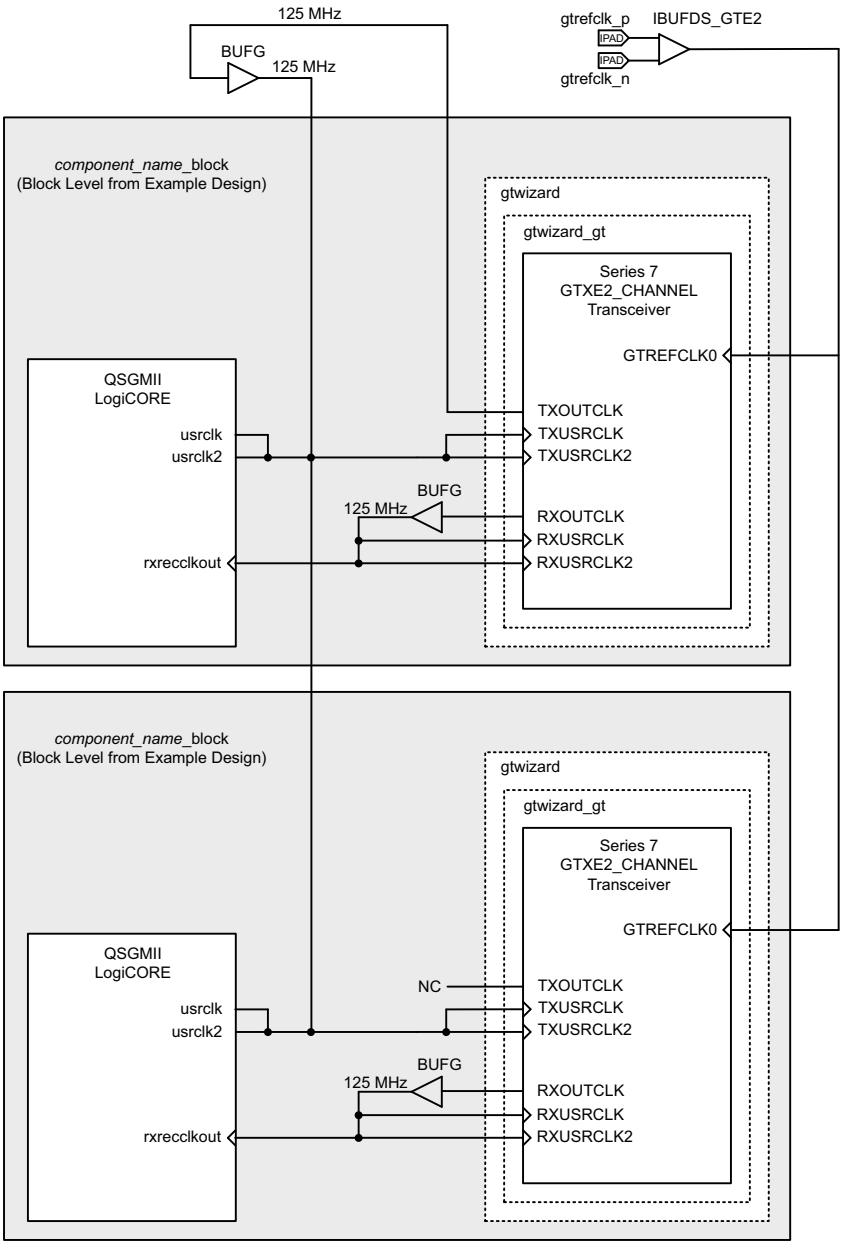

Figure 5-5: **Clock Management with Multiple Core Instances with Kintex-7 FPGA Transceivers**

Artix-7 Devices

Figure 5-6 illustrates sharing clock resources across two instantiations of the core when using 7 series FPGAs transceivers. Additional cores can be added by continuing to instantiate extra block level modules. To provide the FPGA logic clocks for all core instances, select a `TXOUTCLK` port from any transceiver and place it onto global clock routing using BUFGs. Pass this clock to an MMCM to generate 125 and 250 MHz clocks. The 125 MHz clock is used to clock the core logic and `TXUSRCLK2` of the transceivers. The 250 MHz clock is used to clock the `TXUSRCLK` of the transceivers. These can be shared across all core instances and transceivers as illustrated,

Each GTX transceiver and core pair instantiated has its own independent clock domains synchronous to `RXOUTCLK`. These are placed on global clock routing using a BUFMR and subdivided to generate 125 MHz and 250 MHz clocks though BUFRs respectively, as illustrated in Figure 5-6, and cannot be shared across multiple transceivers.

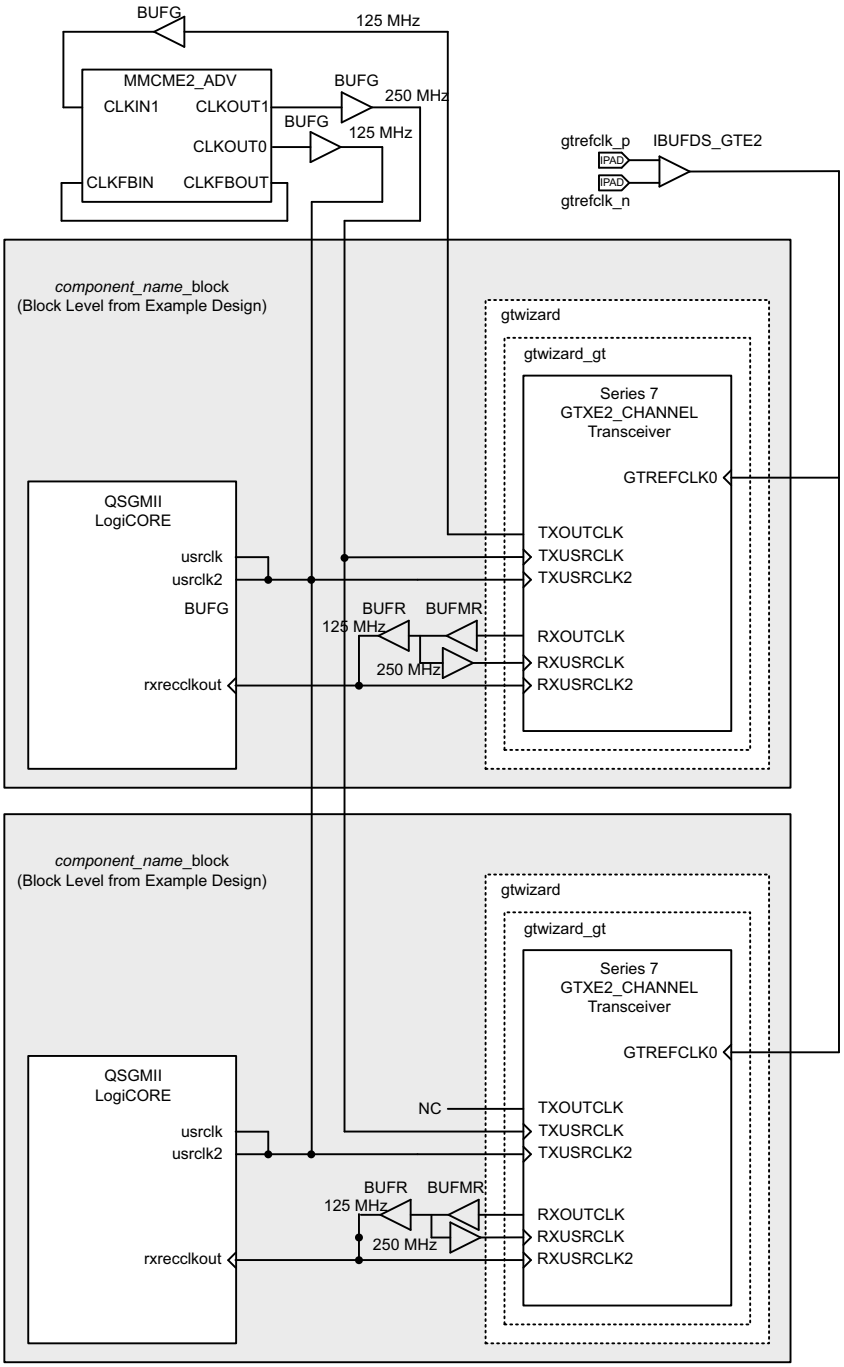

Figure 5-6: **Clock Management with Multiple Core Instances with Artix-7 FPGA Transceivers**

SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Chapter 6

Customizing and Generating the Core

The QSGMII core is generated using the Vivado™ IP Catalog tool. This chapter describes the GUI options used to generate and customize the core.

Graphical User Interface

Core Customization GUI

Figure 6-1 displays the QSGMII customization Graphical User Interface (GUI), used to set core parameters and options. For help starting and using Vivado tools on your system, see the documentation included with the Vivado design suite at www.xilinx.com/support/software_manuals.htm. See the Vivado Design Suite - 2012.2 User Guides web page.

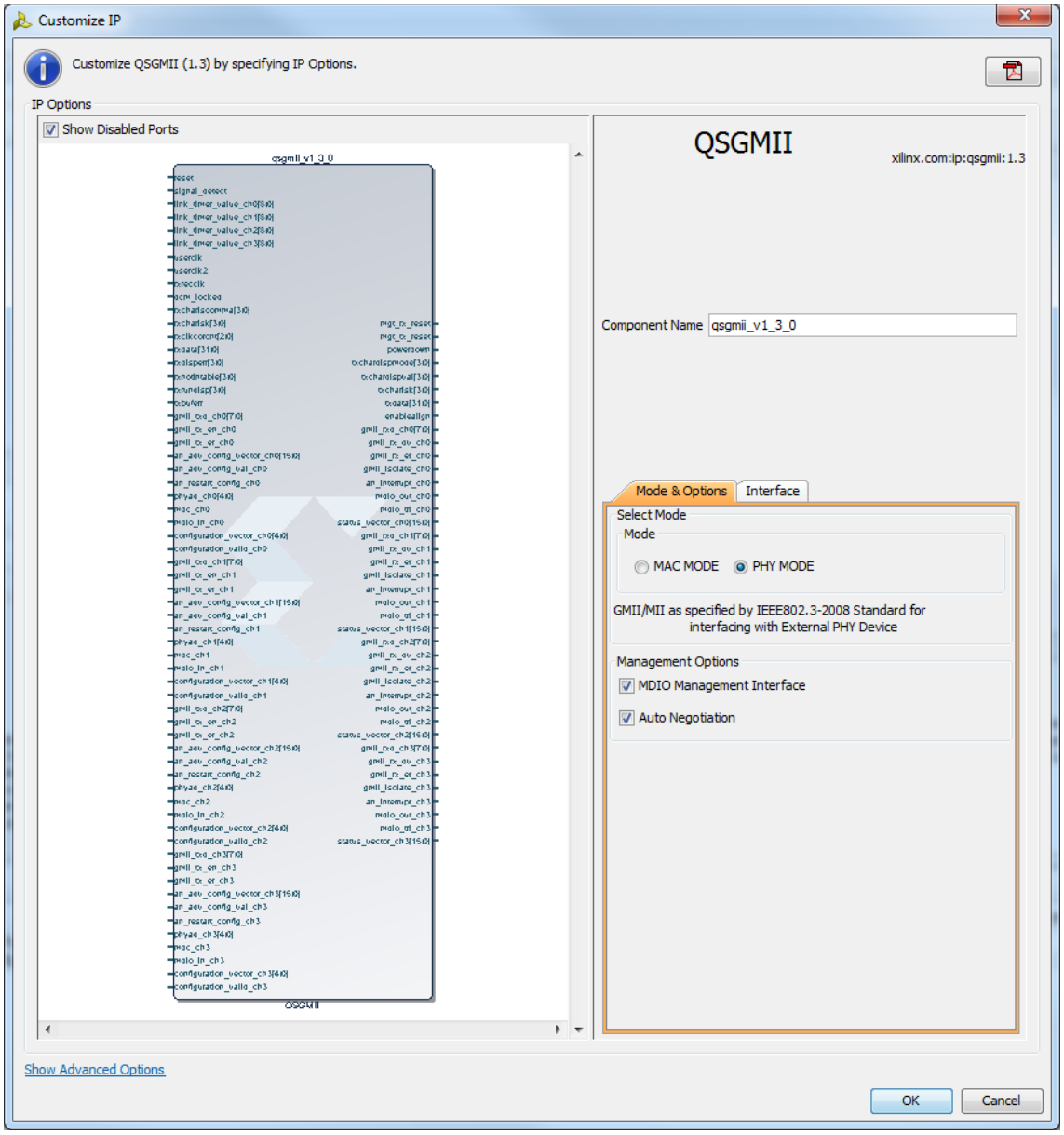

Figure 6-1: **Core Customization Screen**

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 "_."

Select Mode

QSGMII has two main modes of operation.

- MAC Mode
  When configured in this mode, QSGMII interfaces with Xilinx Vivado IP Catalog Tri-mode Ethernet IP on the GMII client side.
- PHY Mode
  When configured in this mode, QSGMII can interface with third-party Ethernet IPs. The client interface can further be selected as GMII or MII.

MDIO Management Interface

Select this option to include the MDIO Management Interface to access the PCS Configuration Registers. An additional configuration interface is provided independent of the MDIO Management Interface to program configuration registers 0 and 4. MDIO Management Interface is selected by default.

Auto-Negotiation

Select this option to include Auto-Negotiation functionality with the core. Auto-Negotiation functionality is selected by default.

Select Interface GUI

Figure 6-2 displays the QSGMII Interface selection GUI. This GUI is only displayed if PHY Mode is selected in the Select Mode section in the initial customization screen.

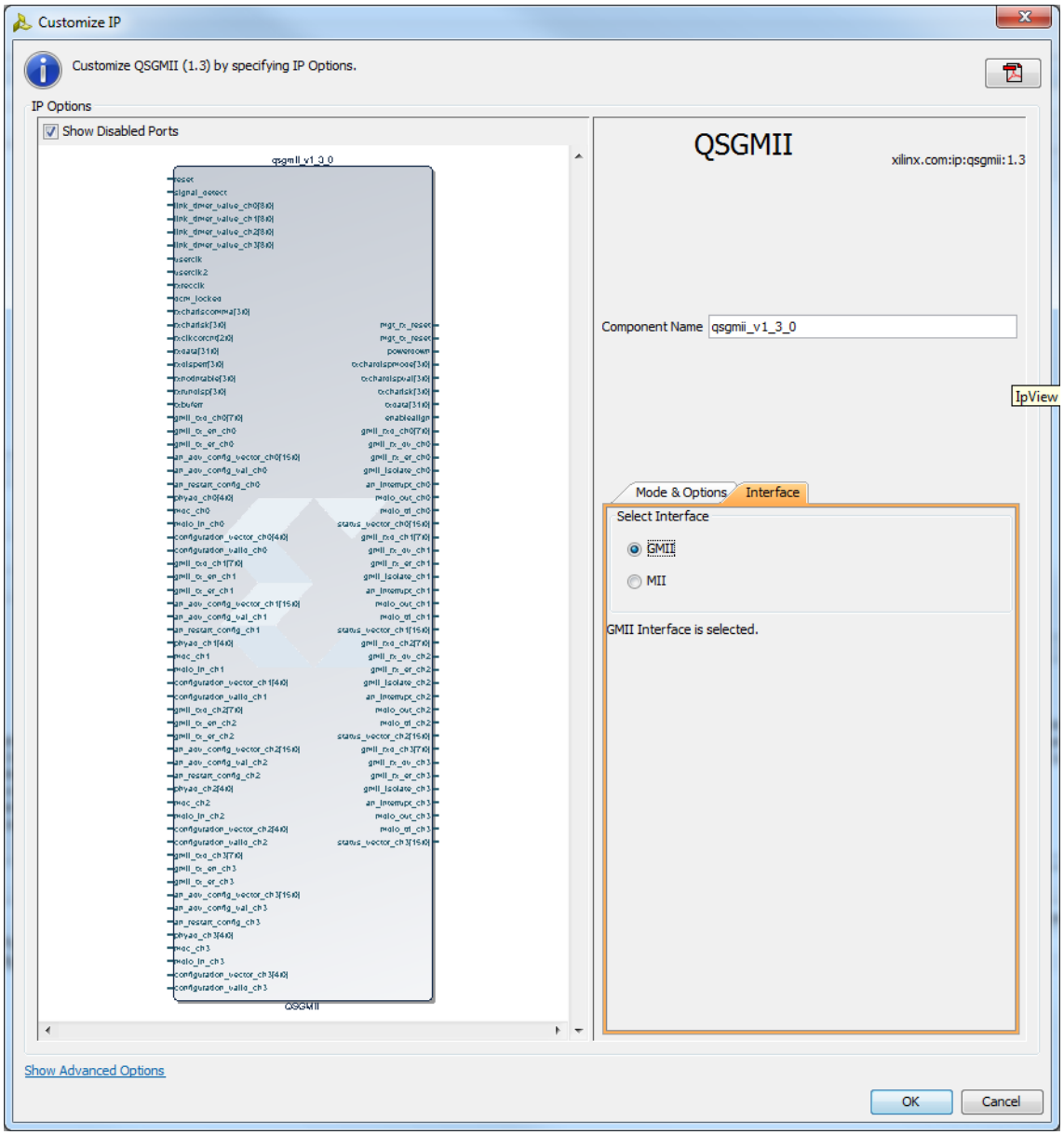

Figure 6-2: **Select Interface GUI**

Output Generation

The QSGMII solution delivers files into several filegroups. By default the filegroups necessary for use of the QSGMII or opening the IP Example design are generated when the core is generated. If additional filegroups are required these can be selected using the generate option.

The filegroups generated can be seen in the IP Sources tab of the Sources window where they are listed for each IP in the project. The filegroups available for the QSGMII solution are:

Examples

Includes all source required to be able to open and implement the IP example design project, that is, example design HDL and the example design xdc file.

Examples Simulation

Includes all source required to be able to simulate the IP example design project. This is the same list of HDL as the Examples filegroup with the addition of the demonstration test bench HDL.

Synthesis

Includes all synthesis sources required by the core. For the QSGMII solution this is a mix of both encrypted and unencrypted source. Only the unencrypted sources are visible.

Simulation

Includes all simulation sources required by the core. Simulation of the QSGMII solution at the core level is not supported without the addition of a test bench (not supplied). Simulation of the example design is supported.

Instantiation Template

Example instantiation template

Miscellaneous

This provides simulations scripts and support files required for running netlist based functional simulation. The files delivered as part of this filegroup are not used or understood by Vivado tools and as such this filegroup is not displayed. These files are delivered into the project source directory.

Constraining the Core

This chapter defines the constraint requirements of the QSGMII core. An example XDC file is also provided with the HDL example design to provide the board level constraints. This is specific to the example design and, as such, is only expected to be used as a template for the user design. See Chapter 8, Detailed Example Design. This XDC file, named `<component name>_example_design.xdc`, is found in the IP Sources tab of the Sources window in the Examples file group.

Device, Package, and Speed Grade Selections

The QSGMII core can be implemented in Virtex®-7, Kintex™-7, and Artix™-7 devices. When selecting a device, be aware of the following considerations:

- Device must be large enough to accommodate the core.
- Device must contain a sufficient number of IOBs.
- -1 speed grade for Virtex-7 and Kintex-7 devices
- -2 speed grade and faster for Artix-7 devices

I/O Location Constraints

No specific I/O location constraints are required.

However, when employing BUFIO and BUFR regional clock routing (Virtex-7, Kintex-7, and Artix-7 devices), ensure that a BUFIO capable clock input pin is selected for input clock sources, and that all related input synchronous data signals are placed in the respective BUFIO region. The user guide for the appropriate device family should be consulted.

Placement Constraints

No specific placement constraints are required.

Transceiver Placement

Virtex-7 FPGA GTH Transceivers

The constraints defined in this section are implemented in the XDC for the example designs delivered with the core. Sections from the XDC are copied into the following descriptions to serve as examples and should be studied with the HDL source code for the example design. Also see the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476).

Transceiver Placement Constraint

The provided XDC uses placement constraints to specify the serial transceiver that is used when the core is implemented. This can be moved around according to the application.

```
set_property LOC GTHE2_CHANNEL_X0Y4 [get_cells
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gthe2_i]
```

Clock Period Constraints

The `gtrefclk` clock is provided to the GTH transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained. The `txoutclk` clock with a frequency of 125 MHz is provided by the GTH transceiver for the transmit path, which is placed onto global clock routing and is input back into the GTHE2 transceiver on the user interface clock ports `txusrclk` and `txusrclk2`. This clock also clocks the core logic and TX PCS logic of the transceiver.

The `rxrecclkoutp` clock of frequency 125 MHz is provided by the GTH transceiver for the receiver path which is placed onto global clock routing and is input back into the GTHE2 transceiver on the user interface clock ports `rxusrclk` and `rxusrclk2`.

```
#*********************************************************
# PCS/PMA Clock period Constraints: please do not relax *
#*********************************************************
create_clock -add -name gtrefclk -period 8.000 [get_pins ibufds_gtrefclk/O]
create_clock -add -name txoutclk -period 8.000 [get_pins
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gthe2_i/
TXOUTCLK]
create_clock -add -name rxrecclkoutp -period 8.000 [get_pins
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gthe2_i/
RXOUTCLK]
```

GTH Transceiver Attributes

The Virtex-7 FPGA GTH transceiver has many attributes that are set directly from the HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the gtwizard_gt.vhd file (for VHDL design entry) or the gtwizard_gt.v file (for Verilog design entry); these files were generated using the 7 series FPGA transceiver wizard.To change the attributes, re-run the wizard.

Virtex-7 FPGA GTX Transceivers

The constraints defined in this section are implemented in the XDC for the example designs delivered with the core. Sections from the XDC are copied into the following descriptions to serve as examples and should be studied with the HDL source code for the example design. Also see the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476).

Transceiver Placement Constraint

The provided XDC uses placement constraints to specify the serial transceiver that is used when the core is implemented. This can be moved around according to the application.

```
set_property LOC GTXE2_CHANNEL_X0Y1 [get_cells
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gtxe2_i]
```

Clock Period Constraints

The `gtrefclk` clock is provided to the GTX transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained.

The `txoutclk` clock of frequency 125 MHz is provided by the GTX transceiver for the transmit path which is placed onto global clock routing and is input back into the GTXE2 transceiver on the user interface clock ports `txusrclk` and `txusrclk2`. This clock also clocks the core logic and TX PCS logic of the transceiver.

The `rxrecclkoutp` clock of frequency 125 MHz is provided by the GTX transceiver for the receiver path which is placed onto global clock routing and is input back into the GTXE2 transceiver on the user interface clock ports `rxusrclk` and `rxusrclk2`.

```
#*********************************************************
# PCS/PMA Clock period Constraints: please do not relax    *
#*********************************************************

create_clock -add -name gtrefclk -period 8.000 [get_pins ibufds_gtrefclk/O]
create_clock -add -name txoutclk -period 8.000 [get_pins
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gtxe2_i/
TXOUTCLK]
create_clock -add -name rxrecclkoutp -period 8.000 [get_pins
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gtxe2_i/
RXOUTCLK]
```

GTX Transceiver Attributes

The Virtex-7 FPGA GTX transceiver has many attributes that are set directly from the HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the gtwizard_gt.vhd file (for VHDL design entry) or the gtwizard_gt.v file (for Verilog design entry); these files were generated using the 7 series FPGA transceiver wizard. To change the attributes, re-run the wizard.

Kintex-7 FPGA GTX Transceivers

The constraints defined in this section are implemented in the XDC for the example designs delivered with the core. Sections from the XDC are copied into the following descriptions to serve as examples, and should be studied with the HDL source code for the example design. Also see the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476).

Transceiver Placement Constraint

The provided XDC uses placement constraints to specify the serial transceiver that is used when the core is implemented. This can be moved around according to the application.

```
set_property LOC GTXE2_CHANNEL_X0Y10 [get_cells
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gtxe2_i]
```

Clock Period Constraints

The `gtrefclk` clock is provided to the GTX transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained.

The `txoutclk` clock of frequency 125 MHz is provided by the GTX transceiver for the transmit path which is placed onto global clock routing and is input back into the GTXE2 transceiver on the user interface clock ports `txusrclk` and `txusrclk2`. This clock also clocks the core logic and TX PCS logic of the transceiver.

The `rxrecclkoutp` clock of frequency 125 MHz is provided by the GTX transceiver for the receiver path which is placed onto global clock routing and is input back into the GTXE2 transceiver on the user interface clock ports rxusrclk and rxusrclk2.

```
#*********************************************************
# PCS/PMA Clock period Constraints: please do not relax    *
#*********************************************************

create_clock -add -name gtrefclk -period 8.000 [get_pins ibufds_gtrefclk/O]
create_clock -add -name txoutclk -period 8.000 [get_pins
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gtxe2_i/
TXOUTCLK]
create_clock -add -name rxrecclkoutp -period 8.000 [get_pins
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gtxe2_i/
RXOUTCLK]
```

GTX Transceiver Attributes

The Kintex-7 FPGA GTX transceiver has many attributes that are set directly from the HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the gtwizard_gt.vhd file (for VHDL design entry) or the gtwizard_gt.vhd.v file (for Verilog design entry); these files were generated using the 7 series FPGA transceiver wizard. To change the attributes, re-run the wizard.

Artix-7 FPGA GTP Transceivers

The constraints defined in this section are implemented in the XDC for the example designs delivered with the core. Sections from the XDC are copied into the following descriptions to serve as examples, and should be studied with the HDL source code for the example design. Also see the *7 Series FPGAs GTP Transceivers User Guide* (UG482).

Transceiver Placement Constraint

The provided XDC uses placement constraints to specify the serial transceiver that is used when the core is implemented. This can be moved around according to the application.

```
set_property LOC GTPE2_CHANNEL_X0Y4 [get_cells
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gtpe2_i]
```

Clock Period Constraints

The `gtrefclk` clock is provided to the GTP transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained. The `txoutclk` clock of frequency 125 MHz is provided by the GTP transceiver for the transmit path which is placed onto global clock routing and fed into a MMCM. The outputs of the MMCM are 125 MHz `clkout0` and 250 MHz `clkout1` which are input back into the GTPE2 transceiver on the user interface clock ports `txusrclk2` and `txusrclk`. The `txuserclk2` clock also clocks the core logic and TX PCS logic of the transceiver.

The `rxrecclkoutp` clock of frequency 250 MHz is provided by the GTP transceiver for the receiver path which is placed onto regional clock routing and is also used to generate a 125 MHz clock. Both these 125 MHz and 250 MHz clocks are input back into the GTPE2 transceiver on the user interface clock ports `rxusrclk2` and `rxusrclk`.

```
#*********************************************************
# PCS/PMA Clock period Constraints: please do not relax *
#*********************************************************
create_clock -add -name gtrefclk -period 8.000 [get_pins ibufds_gtrefclk/O]
create_clock -add -name txoutclk -period 8.000 [get_pins
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gtpe2_i/
TXOUTCLK]
create_clock -add -name rxrecclkoutp -period 4.000 [get_pins
core_wrapper/transceiver_inst/gtwizard_init_inst/GTWIZARD_i/gt0_GTWIZARD_i/gtpe2_i/
RXOUTCLK]
```

GTP Transceiver Attributes

The Artix-7 FPGA GTP transceiver has many attributes that are set directly from the HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the gtwizard_gt.vhd file (for VHDL design entry) or the gtwizard_gt.vhd.v file (for Verilog design entry); these files were generated using the 7 series FPGA transceiver wizard. To change the attributes, re-run the wizard.

Constraints When Using External GMII/MII

The constraints defined in this section are used when the core is operated in "PHY_MODE".

Clock Period Constraints

The core has four instances of SGMII cores. These constraints are valid only when the core is generated with the MODE parameter set to "PHY_MODE" and the Interface parameter set to "GMII" in the GUI. When implementing an external GMII, the Transmitter Elastic Buffer delivered with the example design (or similar logic) must be used. The input transmitter GMII signals are then synchronous to their own clock domain (`gtx_clk_chx` is used in the example design). These clocks must be constrained for a clock frequency of 125 MHz. The following XDC syntax shows the necessary constraints being applied to the example design.

```
#*********************************************************
# GMII GTX transceiver CLK for clocking in GMII TX Interface          *
#*********************************************************

create_clock -add -name gtx_clk_ch0 -period 8.000 [get_ports gtx_clk_ch0]

create_clock -add -name gtx_clk_ch1 -period 8.000 [get_ports gtx_clk_ch1]

create_clock -add -name gtx_clk_ch2 -period 8.000 [get_ports gtx_clk_ch2]

create_clock -add -name gtx_clk_ch3 -period 8.000 [get_ports gtx_clk_ch3]
```

GMII/MII IOB Constraints

The following constraints target the flip-flops that are inferred in the top-level HDL file for the example design. These constraints are defined for receive signals; the transmit GMII/MII interface passes through IDELAY modules to adjust for latency. See GMII Input Setup/Hold Timing. Constraints are set to ensure that these are placed in IOBs.

```
#*************************************************************
# GMII Receiver Constraints:  place flip-flops in IOB      *
#*************************************************************

set_property IOB TRUE [get_cells gmii_rxd_ch0_obuf_reg*]
set_property IOB TRUE [get_cells gmii_rx_dv_ch0_obuf_reg*]
set_property IOB TRUE [get_cells gmii_rx_er_ch0_obuf_reg*]

set_property IOB TRUE [get_cells gmii_rxd_ch1_obuf_reg*]
set_property IOB TRUE [get_cells gmii_rx_dv_ch1_obuf_reg*]
set_property IOB TRUE [get_cells gmii_rx_er_ch1_obuf_reg*]

set_property IOB TRUE [get_cells gmii_rxd_ch2_obuf_reg*]
set_property IOB TRUE [get_cells gmii_rx_dv_ch2_obuf_reg*]
set_property IOB TRUE [get_cells gmii_rx_er_ch2_obuf_reg*]

set_property IOB TRUE [get_cells gmii_rxd_ch3_obuf_reg*]
set_property IOB TRUE [get_cells gmii_rx_dv_ch3_obuf_reg*]
set_property IOB TRUE [get_cells gmii_rx_er_ch3_obuf_reg*]
```

Virtex-7 devices support GMII at 3.3 V or lower only in certain parts and packages. See the Virtex-7 device documentation. GMII/MII by default is supported at 3.3 V and the XDC contains the following syntax. Use this syntax together with the device I/O Banking rules.

```
#*************************************************************
# GMII IOSTANDARD Constraints: please select an I/O        *
# Standard (LVTTL is suggested).                           *
#*************************************************************

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_txd_ch0[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_en_ch0]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_er_ch0]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_rxd_ch0[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_dv_ch0]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_er_ch0]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_txd_ch1[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_en_ch1]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_er_ch1]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_rxd_ch1[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_dv_ch1]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_er_ch1]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_txd_ch2[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_en_ch2]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_er_ch2]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_rxd_ch2[*]}]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_rxd_ch0[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_dv_ch0]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_er_ch0]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_txd_ch1[*]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_en_ch1]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_er_ch1]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_rxd_ch1[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_dv_ch1]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_er_ch1]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_txd_ch2[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_en_ch2]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_er_ch2]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_rxd_ch2[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_dv_ch2]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_er_ch2]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_txd_ch3[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_en_ch3]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_tx_er_ch3]

set_property IOSTANDARD LVCMOS33 [get_ports {gmii_rxd_ch3[*]}]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_dv_ch3]
set_property IOSTANDARD LVCMOS33 [get_ports gmii_rx_er_ch3]

set_property IOSTANDARD LVCMOS33 [get_ports gtx_clk_ch0]
set_property IOSTANDARD LVCMOS33 [get_ports gtx_clk_ch1]
set_property IOSTANDARD LVCMOS33 [get_ports gtx_clk_ch2]
set_property IOSTANDARD LVCMOS33 [get_ports gtx_clk_ch3]
```

GMII Input Setup/Hold Timing

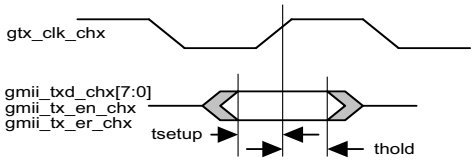

Figure 7-1: **Input GMII Timing Specification**

Figure 7-1 and Table 7-1 illustrate the setup and hold time window for the input GMII signals. These are the worst-case data valid window presented to the FPGA pins.

Observe that there is, in total, a 2 ns data valid window of guaranteed data that is presented across the GMII input bus. This must be correctly sampled by the FPGA devices.

Table 7-1: **Input GMII Timings**

Symbol	Min	Max	Units
t_{SETUP}	2.00	-	ns
t_{HOLD}	0.00	-	ns

Virtex-7, Kintex-7, and Artix-7 Devices

Figure C-1 in Appendix C illustrates the GMII input logic provided by the example design for the Virtex-7, Kintex-7, and Artix-7 family.

Figure C-2 in Appendix C illustrates the MII input logic provided by the example design for the Virtex-7, Kintex-7, and Artix-7 family.

IODELAY elements are instantiated on the GMII/MII data input path as illustrated. Fixed tap delays are applied to these IODELAY elements to delay the GMII/MII input data signals so that data is correctly sampled at the IOB IDDR registers, thereby meeting GMII/MII input setup and hold timing constraints.

The number of tap delays are applied using the following XDC syntax.

```
#*************************************************************
# To Adjust GMII Tx Input Setup/Hold Timing                 *
#*************************************************************
# These constraints will be set at a later date when device speed files have matured
set_property IDELAY_VALUE 0 [get_cells delay_gmii_tx_en_ch0]
set_property IDELAY_VALUE 0 [get_cells delay_gmii_tx_er_ch0]

set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch0[7].delay_gmii_txd_ch0}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch0[6].delay_gmii_txd_ch0}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch0[5].delay_gmii_txd_ch0}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch0[4].delay_gmii_txd_ch0}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch0[3].delay_gmii_txd_ch0}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch0[2].delay_gmii_txd_ch0}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch0[1].delay_gmii_txd_ch0}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch0[0].delay_gmii_txd_ch0}]

set_property IDELAY_VALUE 0 [get_cells delay_gmii_tx_en_ch1]
set_property IDELAY_VALUE 0 [get_cells delay_gmii_tx_er_ch1]

set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch1[7].delay_gmii_txd_ch1}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch1[6].delay_gmii_txd_ch1}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch1[5].delay_gmii_txd_ch1}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch1[4].delay_gmii_txd_ch1}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch1[3].delay_gmii_txd_ch1}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch1[2].delay_gmii_txd_ch1}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch1[1].delay_gmii_txd_ch1}]
set_property IDELAY_VALUE 0 [get_cells {gmii_data_bus_ch1[0].delay_gmii_txd_ch1}]
```

The number of tap delays are preconfigured in the example designs to meet the setup and hold constraints for the example GMII/MII pinout in the particular device.

Detailed Example Design

This chapter provides detailed information about the deliverables provided by the Vivado™ Design Suite for the QSGMII core.

Example Design

Figure 8-1 illustrates an example design for top-level HDL for the QSGMII using a device-specific transceiver (Virtex®-7, Kintex™-7, or Artix™-7 FPGAs).

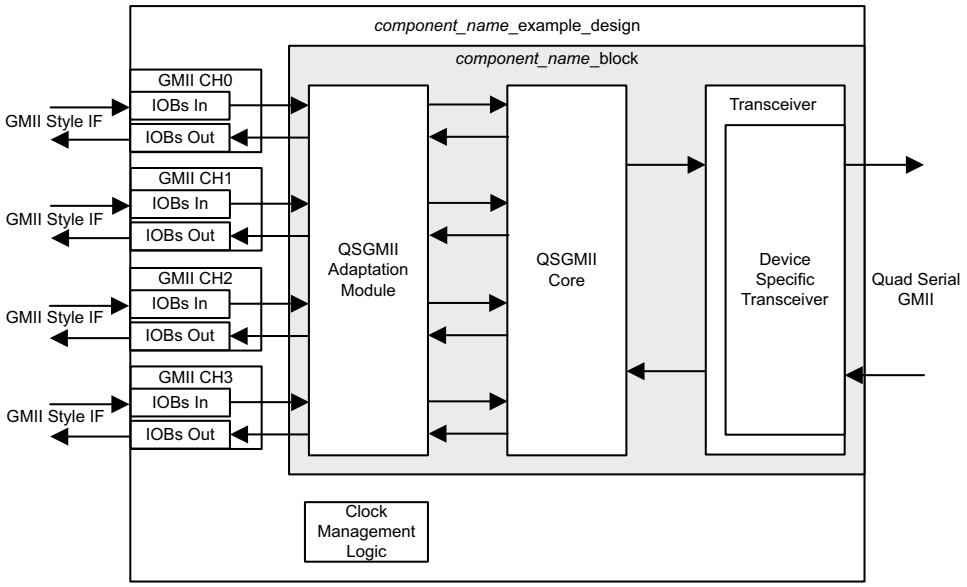

Figure 8-1: **Example Design HDL for QSGMII**

As illustrated, the example is split between two hierarchical layers. The block level is designed so that it can be instantiated directly into customer designs and performs the following functions:

- Instantiates the core from HDL
- Connects the physical-side interface of the core to a device-specific transceiver
- Connects the client side GMII of the core to an QSGMII Adaptation Module, which provides the functionality to operate at speeds of 1 Gb/s, 100 Mb/s and 10 Mb/s

The top level of the example design creates a specific example which can be simulated, synthesized and implemented. The top level of the example design performs the following functions:

- Instantiates the block level from HDL
- Derives the clock management logic for device-specific transceiver and the core
- Implements an external GMII-style interface

The next few pages in this section describe each of the example design blocks (and associated HDL files) in detail.

Top-Level Example Design HDL

The top-level example design for the QSGMII core is described in the following files:

VHDL

```
/example_design/<component_name>_example_design.vhd
```

Verilog

```
/example_design/<component_name>_example_design.v
```

The example design HDL top level contains the following:

- An instance of the QSGMII block level
- Clock management logic for the core and the device-specific transceiver, including DCM (if required) and Global Clock Buffer instances
- External GMII logic, including IOB and Double Data Rate (DDR) register instances, where required

The example design HDL top level connects the GMII interfaces of the block level to external IOBs. This allows the functionality of the core to be demonstrated using a simulation package, as described in this guide.

Block Level HDL

The following files describe the block level for the QSGMII core:

VHDL

```
/example_design/<component_name>_block.vhd
```

Verilog

```
/example_design/<component_name>_block.v
```

The block level contains the following:

- An instance of the QSGMII core
- An instance of a transceiver specific to the target device (Virtex-7, Kintex-7, or Artix-7)
- An QSGMII adaptation module containing four instances of SGMII adaptation module. Each instance of SGMII adaptation module contains
  - The clock management logic required to enable the instance of SGMII operate at 10 Mb/s, 100 Mb/s, and 1 Gb/s.
  - GMII logic for both transmitter and receiver paths.
    - In MAC mode the GMII style 8-bit interface is run at 125 MHz for 1 Gb/s operation; 12.5 MHz for 100 Mb/s operation; 1.25 MHz for 10 Mb/s operation.
    - In PHY mode the GMII style 8 bit interface is run at 125 MHz for 1 Gb/s operation; 25 MHz for 100 Mb/s operation; 2.5 MHz for 10 Mb/s operation. For 100/10 Mb/s operation 4 bits of the MII are mapped to the LSB 4 bits of the GMII style interface.

The block-level HDL connects the PHY side interface of the core to a device-specific transceiver instance and the client side to QSGMII adaptation logic as illustrated in Figure 8-1. This is the most useful part of the example design and should be instantiated in all customer designs that use the core.

Transceiver Files for Virtex-7, Kintex-7 or Artix-7 Devices

Transceiver Wrapper

This device-specific transceiver wrapper is instantiated from the block-level HDL file of the example design and is described in the following files:

VHDL

```
/example_design/transceiver/<component_name>_transceiver.vhd
```

Verilog

```
/example_design/transceiver/<component_name>_transceiver.v
```

This file instances output source files from the transceiver wizard (used with QSGMII attributes).

Virtex-7 and Kintex-7 FPGA Transceiver Wizard Files

For Virtex-7 and Kintex-7 devices, the transceiver wrapper file directly instantiates device-specific transceiver wrapper files created from the transceiver wizard. These files tie off (or leave unconnected) unused I/O for the transceiver, and apply the QSGMII attributes. The files can be edited/tailored by rerunning the wizard and swapping these files. The files include the following:

VHDL

```
/example_design/transceiver/<component_name>_gtwizard_init.vhd
/example_design/transceiver/<component_name>_tx_startup_fsm.vhd
/example_design/transceiver/<component_name>_rx_startup_fsm.vhd

/example_design/transceiver/<component_name>_gtwizard.vhd/example_design/
transceiver/<component_name>_gtwizard_gt.vhd
```

Verilog

```
/example_design/transceiver/<component_name>_gtwizard_init.v
/example_design/transceiver/<component_name>_tx_startup_fsm.v
/example_design/transceiver/<component_name>_rx_startup_fsm.v
/example_design/transceiver/<component_name>_gtwizard.v/example_design/transceiver/
<component_name>_gtwizard_gt.v
```

To re-run the transceiver wizard, a CORE Generator™ tool XCO file for the wizard is included. This file defines all the required wizard attributes used to generate the preceding files. The XCO file is in the following location:

```
/example_design/transceiver/<component_name>_gtwizard.xco
```

QSGMII Adaptation Module

An QSGMII adaptation module containing four instances of SGMII adaptation module. Each instance of SGMII adaptation module contains the following:

- The clock management logic required to enable the instance of SGMII operate at 10 Mb/s, 100 Mb/s, and 1 Gb/s.
- GMII logic for both transmitter and receiver paths.
  - In MAC mode the GMII style 8-bit interface is run at 125 MHz for 1 Gb/s operation; 12.5 MHz for 100 Mb/s operation; 1.25 MHz for 10 Mb/s operation. The speed of operation is controlled by clock enables. The reference clock out (`sgmii_clk_chx`) is always 125 MHz.
  - In PHY mode the GMII style 8 bit interface is run at 125 MHz for 1 Gb/s operation; 25 MHz for 100 Mb/s operation; 2.5 MHz for 10 Mb/s operation. For 100/10 Mb/s operation LSB 4 bits of the GMII style interface is mapped to 4 bits of the MII.

The QSGMII Adaptation Module is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/sgmii_adapt/
    <component_name>_qsgmii_adapt.vhd
    <component_name>_sgmii_adapt.vhd
    <component_name>_clk_gen.vhd
    <component_name>_clk_div.vhd
    <component_name>_johnson_cntr.vhd
    <component_name>_tx_rate_adapt.vhd
    <component_name>_rx_rate_adapt.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/sgmii_adapt/
    <component_name>_qsgmii_adapt.v
    <component_name>_sgmii_adapt.v
    <component_name>_clk_gen.v
    <component_name>_clk_div.v
    <component_name>_johnson_cntr.v
    <component_name>_tx_rate_adapt.v
    <component_name>_rx_rate_adapt.v
```

The GMII of the core always operates at 125 MHz. The core makes no differentiation between the three speeds of operation; it always effectively operates at 1 Gb/s. However, at 100 Mb/s, every data byte run through the core should be repeated 10 times to achieve the required bit rate; at 10 Mb/s, each data byte run through the core should be repeated 100 times to achieve the required bit rate. Dealing with this repetition of bytes is the function of the SGMII adaptation module and its component blocks.

Demonstration Test Bench

Figure 8-2 illustrates the demonstration test bench for the QSGMII core. The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself.

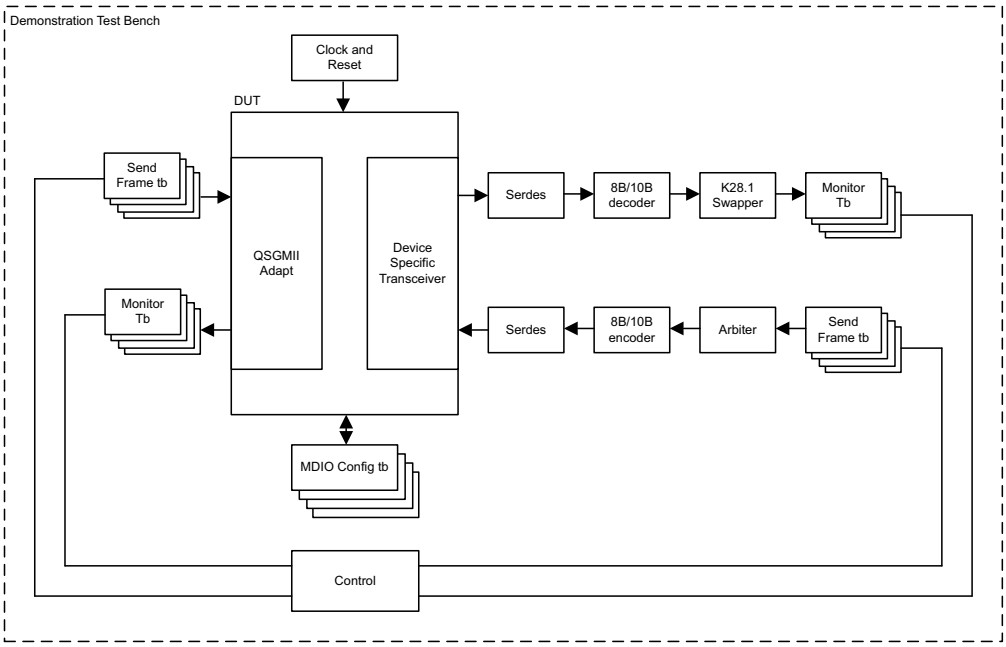

Figure 8-2: **Demonstration Test Bench for QSGMII**

The top-level test bench entity instantiates the example design for the core, which is the Device Under Test (DUT). Other modules needed to provide stimulus, clocks, resets and test bench semaphores are also instantiated in the top-level test bench. The following files describe the top-level of the demonstration test bench.

VHDL

```
/simulation/demo_tb.vhd
```

Verilog

```
/simulation/demo_tb.v
```

Send frame test bench generates the stimulus to excite the transceiver on the DUT receive side and data input on the DUT QSGMII adapt side. Four instances of the send frame test bench are instantiated, with each instance representing one channel.

VHDL

```
/simulation/send_frame_tb.vhd
```

Verilog

```
/simulation/send_frame_tb.v
```

The Arbiter module selects one byte from each instance of the send frame test bench and passes it on to the 8B/10B encoder module.

VHDL

```
/simulation/arbiter_tb.vhd
```

Verilog

```
/simulation/arbiter_tb.v
```

The 8B/10B encoder test bench module converts 8-bit data from arbiter to 10 bits as specified by *IEEE 802.3-2008* standard clause 36.

VHDL

```
/simulation/encode_8b10b_tb.vhd
```

Verilog

```
/simulation/encode_8b10b_tb.v
```

The 8B/10B decoder test bench module converts 10-bit data from SERDES on the transceiver transmit interface to 10 bits as specified by *IEEE 802.3-2008* standard clause 36.

VHDL

```
/simulation/decode_8b10b_tb.vhd
```

Verilog

```
/simulation/decode_8b10b_tb.v
```

The SERDES module serializes the 10-bit data from the 8B/10B encoder and maps it to the receive interface of the DUT transceiver. This module de-serializes the serial bitstream from the transmit interface of the DUT transceiver and maps it to the 8B/10B decoder.

VHDL

```
/simulation/serdes_tb.vhd
```

Verilog

```
/simulation/serdes_tb.v
```

The K28.1 swapper module swaps K28.1 characters received on port 0 with K28.5 as specified in the QSGMII specification.

VHDL

```
/simulation/k28p1_swapper_tb.vhd
```

Verilog

```
/simulation/k28p1_swapper_tb.v
```

The Monitor test bench module monitors the output from the DUT and verifies the data with preloaded data structures present in the module.

VHDL

```
/simulation/monitor_tb.vhd
```

Verilog

```
/simulation/monitor_tb.v
```

The programing of per channel configuration registers in the DUT is performed through MDIO configuration test bench. There are four instances of this module with each instance representing one channel.

VHDL

```
/simulation/mdio_cfg_tb.vhd
```

Verilog

```
/simulation/mdio_cfg_tb.v
```

Test Bench Functionality

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- Each channel of the QSGMII core is configured through the MDIO interface by injecting an MDIO frame into the example design. This disables Auto-Negotiation and takes the core out of Isolate state.
- The speed of the interface is programmed as follows
  - MAC mode
    - Channel 0 - 1 Gb/s
    - Channel 1 - 100 Mb/s
    - Channel 2 - 10 Mb/s
    - Channel 3 - 1 Gb/s

  - PHY mode with GMII; all channels at 1 Gb/s
  - PHY mode with MII
    - Channel 0 - 10 Mb/s
    - Channel 1- 100 Mb/s
    - Channel 2 - 10 Mb/s
    - Channel 3 - 100 Mb/s
- The following frames are injected into the transmitter by the send frame block.
  - the first is a minimum length frame
  - the second is a type frame
  - the third is an errored frame
  - the fourth is a padded frame
- The serial data received at the device-specific transceiver transmitter interface is converted to 10-bit parallel data, then 8B/10B decoded. The resultant byte is aligned to the corresponding channel based on the K28.1 character set and also the K28.1 character set is replaced with K28.5. The resulting frames are checked by in the monitor test bench against the stimulus frames injected into the transmitter to ensure data integrity.
- The same four frames are generated by the frame generator module in the receive side of the transceiver. The data from all four instances are aggregated into 32 bits, are 8B/10B encoded, converted to serial data, and injected into the device-specific transceiver receiver interface at 5 Gb/s.
- Data frames received at the receiver GMII interface are checked by the Monitor against the stimulus frames injected into the device-specific transceiver receiver to ensure data integrity.

Customizing the Test Bench

Changing Frame Data

You can change the contents of the four frames used by the demonstration test bench by changing the *data* and *valid* fields for each frame defined in the send frame module. New frames can be added by defining a new frame of data. Modified frames are automatically updated in both the stimulus and monitor functions.

Changing Frame Error Status

Errors can be inserted into any of the predefined frames in any position by setting the *error* field to ‘1’ in any column of that frame. Injected errors are automatically updated in both the stimulus and monitor functions.

Changing the Core Configuration

The configuration of the QSGMII core used in the demonstration test bench can be altered.

CAUTION! *Certain configurations of the core cause the test bench to fail, or to cause processes to run indefinitely. For example, the demonstration test bench will not Auto-Negotiate with the design example. Determine the configurations that can safely be used with the test bench.*

The core can be reconfigured by editing the injected MDIO frame in the demonstration test bench top level.

Changing the Operational Speed

The speed of both the example design and test bench can be set to the desired operational speed by editing the following settings, recompiling the test bench, and then running the simulation again. The changes also need to be implemented in the *example design*.

1 Gb/s Operation

```
set speed_is_10_100_chx to logic 0
```

100 Mb/s Operation

```
set speed_is_10_100_chx to logic 1
set speed_is_100_chx to logic 1
```

10 Mb/s Operation

```
set speed_is_10_100_chx to logic 1
set speed_is_100_chx to logic 0
```

SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

The QSGMII core is generated using the CORE Generator™ tool. This chapter describes the GUI options used to generate and customize the core.

GUI

Core Customization Screen

Figure 9-1 displays the QSGMII customization screen, used to set core parameters and options. For help starting and using CORE Generator tool on your system, see the documentation included with the ISE® design suite, including the *CORE Generator Guide*, at www.xilinx.com/support/software_manuals.htm.

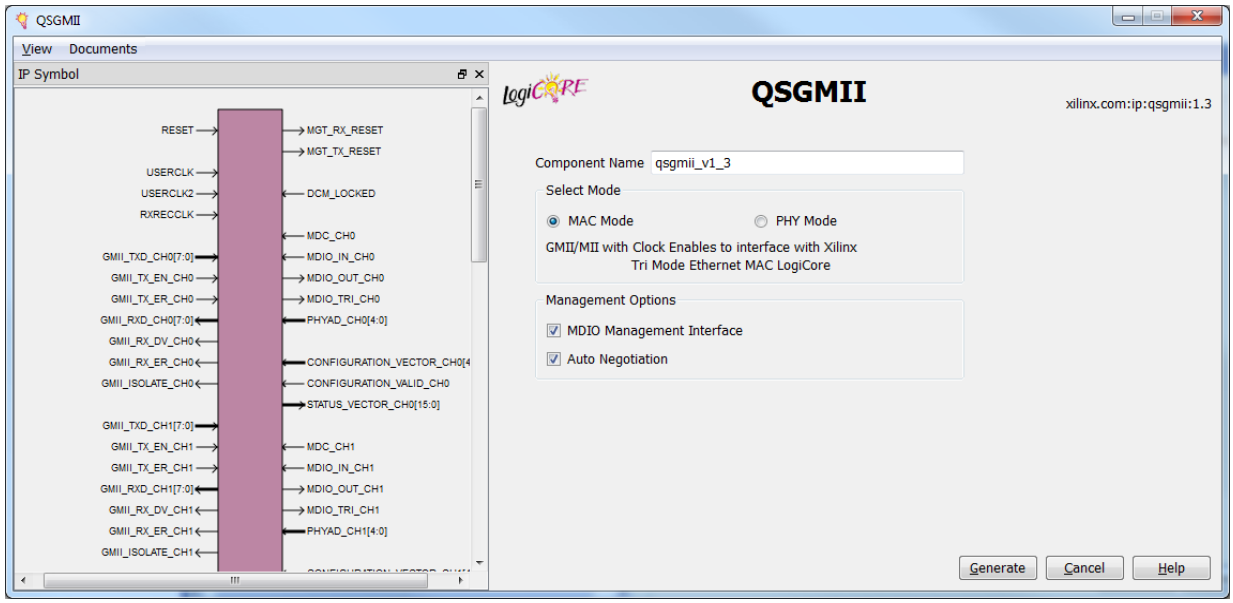

Figure 9-1: **Core Customization Screen**

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 "_."

Select Mode

QSGMII has two main modes of operation.

- MAC Mode

  When configured in this mode, QSGMII interfaces with Xilinx CORE Generator tool Tri-mode Ethernet IP on the GMII client side.

- PHY Mode

  When configured in this mode, QSGMII can interface with third-party Ethernet IPs. The client interface can further be selected as GMII or MII.

MDIO Management Interface

Select this option to include the MDIO Management Interface to access the PCS Configuration Registers. An additional configuration interface is provided independent of the MDIO Management Interface to program configuration registers 0 and 4. MDIO Management Interface is selected by default.

Auto-Negotiation

Select this option to include Auto-Negotiation functionality with the core. Auto-Negotiation functionality is selected by default.

Select Interface Screen

Figure 9-2 displays the QSGMII Interface selection screen. This screen is *only* displayed if "PHY" Mode is selected in the "Select Mode" section in the initial customization screen

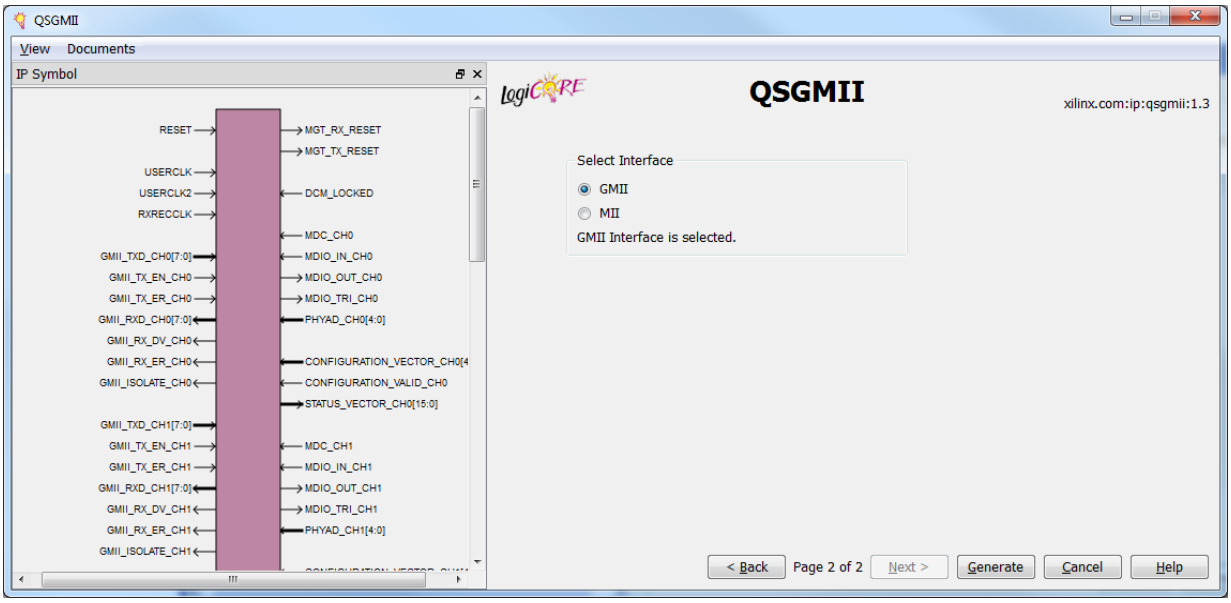

Figure 9-2: **Select Interface Screen**

Parameter Values in the XCO File

XCO file parameters are used to run the CORE Generator tool from the command line. The text in an Xilinx CORE Generator (XCO) file is not case sensitive.

Table 9-1 describes the XCO file parameters and values and summarizes the GUI defaults. The following is an example of the CSET parameters in an XCO file.

```
CSET component_name=qsgmii_v1_3
CSET mode=MAC_MODE
CSET management_interface=true
CSET auto_negotiation=true
CSET gmii_or_mii_mode=GMII
```

Table 9-1: **Parameters**

Parameter	XCO File Values	Default GUI Setting
component_name	ASCII text starting with a letter and based upon the following character set: a..z, 0..9 and _	qsgmii_v1_2
mode	One of the following keywords: MAC_MODE, PHY_MODE	MAC_MODE
management_interface	One of the following keywords: true, false	true
auto_negotiation	One of the following keywords: true, false	true
gmii_or_mii_mode	One of the following keywords: GMII, MII	GMII

Output Generation

See Directory and File Contents in Chapter 11 for details about the files and directories created when generating the core.

Constraining the Core

This chapter defines the constraint requirements of the QSGMII core. An example UCF is provided with the HDL example design for the core to implement the constraints defined in this chapter.

Device, Package, and Speed Grade Selections

The QSGMII core can be implemented in Virtex®-7, Kintex™-7, and Artix™-7 devices. When selecting a device, be aware of the following considerations:

- Device must be large enough to accommodate the core.
- Device must contain a sufficient number of IOBs.
- -1 speed grade for Virtex-7 and Kintex-7 devices.
- -2 speed grade or faster for Artix-7 devices.

I/O Location Constraints

No specific I/O location constraints are required.

However, when employing BUFIO and BUFR regional clock routing (Virtex-7, Kintex-7, and Artix™-7 devices), ensure that a BUFIO capable clock input pin is selected for input clock sources, and that all related input synchronous data signals are placed in the respective BUFIO region. The device user guide should be consulted.

Placement Constraints

No specific placement constraints are required.

Transceiver Placement

Virtex-7 FPGA GTH Transceivers

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from the UCF are copied into the following descriptions to serve as examples and should be studied with the HDL source code for the example design. Also see the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476).

Transceiver Placement Constraint

The provided UCF uses placement constraints to specify the serial transceiver that is used when the core is implemented. This can be moved around according to the application.

```
INST "core_wrapper?transceiver_inst?gtwizard_inst?gt0_gtwizard_i?gthe2_i" LOC =
"GTXE2_CHANNEL_X0Y4"
```

Clock Period Constraints

The `gtrefclk` clock is provided to the GTH transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained. The `txoutclk` clock with a frequency of 125 MHz is provided by the GTH transceiver for the transmit path, which is placed onto global clock routing and is input back into the GTHE2 transceiver on the user interface clock ports `txusrclk` and `txusrclk2`. This clock also clocks the core logic and TX PCS logic of the transceiver.

The `rxrecclk` clock of frequency 125 MHz is provided by the GTH transceiver for the receiver path which is placed onto global clock routing and is input back into the GTHE2 transceiver on the user interface clock ports `rxusrclk` and `rxusrclk2`.

```
#*********************************************************
# PCS/PMA Clock period Constraints: please do not relax *
#*********************************************************
NET "gtrefclk" TNM_NET = "gtrefclk";
TIMESPEC "ts_gtrefclk" = PERIOD "gtrefclk" 8 ns HIGH 50 %;
NET "txoutclk" TNM_NET = "txoutclk";
TIMESPEC "TS_txoutclk" = PERIOD "txoutclk" 8 ns HIGH 50 %;
NET "core_wrapper/qsgmii_core/rxrecclk" TNM_NET = "rxrecclk";
TIMESPEC "ts_rxrecclk" = PERIOD "rxrecclk" 8 ns HIGH 50 %;
```

GTH Transceiver Attributes

The Virtex-7 FPGA GTH transceiver has many attributes that are set directly from the HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the gtwizard_gt.vhd file (for VHDL design entry) or the gtwizard_gt.v file (for Verilog design entry); these files were generated using the 7 series FPGA transceiver wizard.To change the attributes, re-run the wizard.

Virtex-7 FPGA GTX Transceivers

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from the UCF are copied into the following descriptions to serve as examples and should be studied with the HDL source code for the example design. Also see the *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476).

Transceiver Placement Constraint

The provided UCF uses placement constraints to specify the serial transceiver that is used when the core is implemented. This can be moved around according to the application.

```
INST "core_wrapper?transceiver_inst?gtwizard_inst?gt0_gtwizard_i?gtxe2_i" LOC =
"GTXE2_CHANNEL_X0Y1"
```

Clock Period Constraints

The `gtrefclk` clock is provided to the GTX transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained.

The `txoutclk` clock of frequency 125 MHz is provided by the GTX transceiver for the transmit path which is placed onto global clock routing and is input back into the GTXE2 transceiver on the user interface clock ports `txusrclk` and `txusrclk2`. This clock also clocks the core logic and TX PCS logic of the transceiver.

The `rxrecclk` clock of frequency 125 MHz is provided by the GTX transceiver for the receiver path which is placed onto global clock routing and is input back into the GTXE2 transceiver on the user interface clock ports `rxusrclk` and `rxusrclk2`.

```
#*********************************************************
# PCS/PMA Clock period Constraints: please do not relax    *
#*********************************************************

NET "gtrefclk" TNM_NET = "gtrefclk";
TIMESPEC "ts_gtrefclk" = PERIOD "gtrefclk" 8 ns HIGH 50 %;

NET "txoutclk" TNM_NET = "txoutclk";
TIMESPEC "TS_txoutclk" = PERIOD "txoutclk" 8 ns HIGH 50 %;

NET "core_wrapper/qsgmii_core/rxrecclk" TNM_NET = "rxrecclk";
TIMESPEC "ts_rxrecclk" = PERIOD "rxrecclk" 8 ns HIGH 50 %;
```

GTX Transceiver Attributes

The Virtex-7 FPGA GTX transceiver has many attributes that are set directly from the HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the gtwizard_gt.vhd file (for VHDL design entry) or the gtwizard_gt.v file (for Verilog design entry); these files were generated using the 7 series FPGA transceiver wizard. To change the attributes, re-run the wizard.

Kintex-7 FPGA GTX Transceivers

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from the UCF are copied into the following descriptions to serve as examples, and should be studied with the HDL source code for the example design. Also see the *7 Series FPGAs GTP Transceivers User Guide* (UG482).

Transceiver Placement Constraint

The provided UCF uses placement constraints to specify the serial transceiver that is used when the core is implemented. This can be moved around according to the application.

```
INST "core_wrapper?transceiver_inst?gtwizard_inst?gt0_gtwizard_i?gtxe2_i" LOC =
"GTXE2_CHANNEL_X0Y10"
```

Clock Period Constraints

The `gtrefclk` clock is provided to the GTX transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained.

The `txoutclk` clock of frequency 125 MHz is provided by the GTX transceiver for the transmit path which is placed onto global clock routing and is input back into the GTXE2 transceiver on the user interface clock ports `txusrclk` and `txusrclk2`. This clock also clocks the core logic and TX PCS logic of the transceiver.

The `rxrecclk` clock of frequency 125 MHz is provided by the GTX transceiver for the receiver path which is placed onto global clock routing and is input back into the GTXE2 transceiver on the user interface clock ports `rxusrclk` and `rxusrclk2`.

```
#*********************************************************
# PCS/PMA Clock period Constraints: please do not relax    *
#*********************************************************

NET "gtrefclk" TNM_NET = "gtrefclk";
TIMESPEC "ts_gtrefclk" = PERIOD "gtrefclk" 8 ns HIGH 50 %;

NET "txoutclk" TNM_NET = "txoutclk";
TIMESPEC "TS_txoutclk" = PERIOD "txoutclk" 8 ns HIGH 50 %;

NET "core_wrapper/qsgmii_core/rxrecclk" TNM_NET = "rxrecclk";
TIMESPEC "ts_rxrecclk" = PERIOD "rxrecclk" 8 ns HIGH 50 %;
```

GTX Transceiver Attributes

The Kintex-7 FPGA GTX transceiver has many attributes that are set directly from the HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the gtwizard_gt.vhd file (for VHDL design entry) or the gtwizard_gt.vhd.v file (for Verilog design entry); these files were generated using the 7 series FPGA transceiver wizard. To change the attributes, re-run the wizard.

Artix-7 FPGA GTP Transceivers

The constraints defined in this section are implemented in the UCF for the example designs delivered with the core. Sections from the UCF are copied into the following descriptions to serve as examples, and should be studied with the HDL source code for the example design. Also see the *7 Series FPGAs GTP Transceivers User Guide* (UG482).

Transceiver Placement Constraint

The provided UCF uses placement constraints to specify the serial transceiver that is used when the core is implemented. This can be moved around according to the application.

```
INST
"core_wrapper?transceiver_inst?gtwizard_init_inst?gtwizard_i?gt0_gtwizard_i?gtpe2_i
" LOC = "GTPE2_CHANNEL_X0Y4";
```

Clock Period Constraints

The `gtrefclk` clock is provided to the GTP transceiver. It is a high-quality reference clock with a frequency of 125 MHz and should be constrained. The `txoutclk` clock of frequency 125 MHz is provided by the GTP transceiver for the transmit path which is placed onto global clock routing and fed into a MMCM. The outputs of the MMCM are 125 MHz `clkout0` and 250 MHz `clkout1` which are input back into the GTPE2 transceiver on the user interface clock ports `txusrclk2` and `txusrclk`. The `txuserclk2` clock also clocks the core logic and TX PCS logic of the transceiver.

The `rxoutclk` clock of frequency 250 MHz is provided by the GTP transceiver for the receiver path which is placed onto regional clock routing and is also used to generate 125 MHz clock. Both these 125 MHz and 250 MHz clocks are input back into the GTPE2 transceiver on the user interface clock ports `rxusrclk2` and `rxusrclk`.

```
#*********************************************************
# PCS/PMA Clock period Constraints: please do not relax *
#*********************************************************
NET "gtrefclk" TNM_NET = "gtrefclk";
TIMESPEC "ts_gtrefclk" = PERIOD "gtrefclk" 8 ns HIGH 50 %;
NET "txoutclk" TNM_NET = "txoutclk";
TIMESPEC "TS_txoutclk" = PERIOD "txoutclk" 8 ns HIGH 50 %;
NET "core_wrapper/qsgmii_core/rxrecclk" TNM_NET = "rxrecclk";
TIMESPEC "ts_rxrecclk" = PERIOD "rxrecclk" 8 ns HIGH 50 %;
```

GTP Transceiver Attributes

The Artix-7 FPGA GTP transceiver has many attributes that are set directly from the HDL source code for the transceiver wrapper file delivered with the example design. These can be found in the gtwizard_gt.vhd file (for VHDL design entry) or the gtwizard_gt.vhd.v file (for Verilog design entry); these files were generated using the 7 series FPGA transceiver wizard. To change the attributes, re-run the wizard.

Constraints When Using External GMII/MII

The constraints defined in this section are used when the core is operated in "PHY_MODE".

Clock Period Constraints

The core has four instances of SGMII cores. These constraints are valid only when the core is generated with the MODE parameter set to "PHY_MODE" and the Interface parameter set to "GMII" in the GUI. When implementing an external GMII, the Transmitter Elastic Buffer delivered with the example design (or similar logic) must be used. The input transmitter GMII signals are then synchronous to their own clock domain (`gtx_clk_chx` is used in the example design). These clocks must be constrained for a clock frequency of 125 MHz. The following UCF syntax shows the necessary constraints being applied to the example design.

```
#*********************************************************
# GMII GTX transceiver CLK for clocking in GMII TX Interface          *
#*********************************************************

NET "gtx_clk_ch0" TNM_NET = "gtx_clk_ch0";
TIMESPEC "ts_gtx_clk_ch0" = PERIOD "gtx_clk_ch0" 8 ns HIGH 50 %;

NET "gtx_clk_ch1" TNM_NET = "gtx_clk_ch1";
TIMESPEC "ts_gtx_clk_ch1" = PERIOD "gtx_clk_ch1" 8 ns HIGH 50 %;

NET "gtx_clk_ch2" TNM_NET = "gtx_clk_ch2";
TIMESPEC "ts_gtx_clk_ch2" = PERIOD "gtx_clk_ch2" 8 ns HIGH 50 %;

NET "gtx_clk_ch3" TNM_NET = "gtx_clk_ch3";
TIMESPEC "ts_gtx_clk_ch3" = PERIOD "gtx_clk_ch3" 8 ns HIGH 50 %;
```

GMII/MII IOB Constraints

The following constraints target the flip-flops that are inferred in the top-level HDL file for the example design. These constraints are defined for receive signals; the transmit GMII/MII interface passes through IDELAY modules to adjust for latency. See GMII Input Setup/Hold Timing. Constraints are set to ensure that these are placed in IOBs.

```
#*************************************************************
# GMII Receiver Constraints:  place flip-flops in IOB      *
#*************************************************************
INST "gmii_rxd_ch0*"  IOB = true;
INST "gmii_rx_dv_ch0" IOB = true;
INST "gmii_rx_er_ch0" IOB = true;


INST "gmii_rxd_ch1*"  IOB = true;
INST "gmii_rx_dv_ch1" IOB = true;
INST "gmii_rx_er_ch1" IOB = true;

INST "gmii_rxd_ch2*"  IOB = true;
INST "gmii_rx_dv_ch2" IOB = true;
INST "gmii_rx_er_ch2" IOB = true;

INST "gmii_rxd_ch3*"  IOB = true;
INST "gmii_rx_dv_ch3" IOB = true;
INST "gmii_rx_er_ch3" IOB = true;
```

Virtex-7 devices support GMII at 3.3 V or lower only in certain parts and packages. See the Virtex-7 device documentation. GMII/MII by default is supported at 3.3 V and the UCF contains the following syntax. Use this syntax together with the device I/O Banking rules.

```
#*************************************************************
# GMII IOSTANDARD Constraints: please select an I/O        *
# Standard (LVTTL is suggested).                           *
#*************************************************************

INST "gmii_txd_ch0<?>"      IOSTANDARD = LVCMOS33;
INST "gmii_tx_en_ch0"       IOSTANDARD = LVCMOS33;
INST "gmii_tx_er_ch0"       IOSTANDARD = LVCMOS33;


INST "gmii_rxd_ch0<?>"      IOSTANDARD = LVCMOS33;
INST "gmii_rx_dv_ch0"       IOSTANDARD = LVCMOS33;
INST "gmii_rx_er_ch0"       IOSTANDARD = LVCMOS33;

INST "gmii_txd_ch1<?>"      IOSTANDARD = LVCMOS33;
INST "gmii_tx_en_ch1"       IOSTANDARD = LVCMOS33;
INST "gmii_tx_er_ch1"       IOSTANDARD = LVCMOS33;

INST "gmii_rxd_ch1<?>"      IOSTANDARD = LVCMOS33;
INST "gmii_rx_dv_ch1"       IOSTANDARD = LVCMOS33;
INST "gmii_rx_er_ch1"       IOSTANDARD = LVCMOS33;

INST "gmii_txd_ch2<?>"      IOSTANDARD = LVCMOS33;
INST "gmii_tx_en_ch2"       IOSTANDARD = LVCMOS33;
INST "gmii_tx_er_ch2"       IOSTANDARD = LVCMOS33;

INST "gmii_rxd_ch2<?>"      IOSTANDARD = LVCMOS33;
INST "gmii_rx_dv_ch2"       IOSTANDARD = LVCMOS33;
INST "gmii_rx_er_ch2"       IOSTANDARD = LVCMOS33;

INST "gmii_txd_ch3<?>"      IOSTANDARD = LVCMOS33;
```

```
INST "gmii_tx_en_ch3"        IOSTANDARD = LVCMOS33;
INST "gmii_tx_er_ch3"        IOSTANDARD = LVCMOS33;

INST "gmii_rxd_ch3<?>"       IOSTANDARD = LVCMOS33;
INST "gmii_rx_dv_ch3"        IOSTANDARD = LVCMOS33;
INST "gmii_rx_er_ch3"        IOSTANDARD = LVCMOS33;

INST "gtx_clk_ch0"           IOSTANDARD = LVCMOS33;
INST "gtx_clk_ch1"           IOSTANDARD = LVCMOS33;
INST "gtx_clk_ch2"           IOSTANDARD = LVCMOS33;
INST "gtx_clk_ch3"           IOSTANDARD = LVCMOS33;
```

GMII Input Setup/Hold Timing

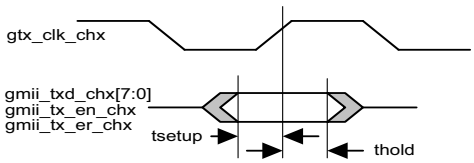

Figure 10-1: **Input GMII Timing Specification**

Figure 10-1 and Table 10-1 illustrate the setup and hold time window for the input GMII signals. These are the worst-case data valid window presented to the FPGA pins.

Observe that there is, in total, a 2 ns data valid window of guaranteed data that is presented across the GMII input bus. This must be correctly sampled by the FPGA devices.

Table 10-1: **Input GMII Timings**

Symbol	Min	Max	Units
t_{SETUP}	2.00	-	ns
t_{HOLD}	0.00	-	ns

Virtex-7, Kintex-7 and Artix-7 Devices

Figure C-1 in Appendix C illustrates the GMII input logic provided by the example design for the Virtex-7, Kintex-7, and Artix-7 families.

Figure C-2 in Appendix C illustrates the MII input logic provided by the example design for the Virtex-7, Kintex-7, and Artix-7 families.

IODELAY elements are instantiated on the GMII/MII data input path as illustrated. Fixed tap delays are applied to these IODELAY elements to delay the GMII/MII input data signals so that data is correctly sampled at the IOB IDDR registers, thereby meeting GMII/MII input setup and hold timing constraints.

The number of tap delays are applied using the following UCF syntax.

```
#*************************************************************
# To Adjust GMII Tx Input Setup/Hold Timing                 *
#*************************************************************
# These constraints will be set at a later date when device speed files have matured

INST "delay_gmii_tx_en_ch0"  IDELAY_VALUE = 0;
INST "delay_gmii_tx_er_ch0"  IDELAY_VALUE = 0;

INST "gmii_data_bus_ch0[7].delay_gmii_txd_ch0"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch0[6].delay_gmii_txd_ch0"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch0[5].delay_gmii_txd_ch0"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch0[4].delay_gmii_txd_ch0"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch0[3].delay_gmii_txd_ch0"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch0[2].delay_gmii_txd_ch0"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch0[1].delay_gmii_txd_ch0"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch0[0].delay_gmii_txd_ch0"  IDELAY_VALUE = 0;

INST "delay_gmii_tx_en_ch1"  IDELAY_VALUE = 0;
INST "delay_gmii_tx_er_ch1"  IDELAY_VALUE = 0;

INST "gmii_data_bus_ch1[7].delay_gmii_txd_ch1"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch1[6].delay_gmii_txd_ch1"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch1[5].delay_gmii_txd_ch1"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch1[4].delay_gmii_txd_ch1"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch1[3].delay_gmii_txd_ch1"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch1[2].delay_gmii_txd_ch1"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch1[1].delay_gmii_txd_ch1"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch1[0].delay_gmii_txd_ch1"  IDELAY_VALUE = 0;

INST "delay_gmii_tx_en_ch2"  IDELAY_VALUE = 0;
INST "delay_gmii_tx_er_ch2"  IDELAY_VALUE = 0;

INST "gmii_data_bus_ch2[7].delay_gmii_txd_ch2"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch2[6].delay_gmii_txd_ch2"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch2[5].delay_gmii_txd_ch2"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch2[4].delay_gmii_txd_ch2"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch2[3].delay_gmii_txd_ch2"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch2[2].delay_gmii_txd_ch2"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch2[1].delay_gmii_txd_ch2"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch2[0].delay_gmii_txd_ch2"  IDELAY_VALUE = 0;

INST "delay_gmii_tx_en_ch3"  IDELAY_VALUE = 0;
INST "delay_gmii_tx_er_ch3"  IDELAY_VALUE = 0;

INST "gmii_data_bus_ch3[7].delay_gmii_txd_ch3"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch3[6].delay_gmii_txd_ch3"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch3[5].delay_gmii_txd_ch3"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch3[4].delay_gmii_txd_ch3"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch3[3].delay_gmii_txd_ch3"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch3[2].delay_gmii_txd_ch3"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch3[1].delay_gmii_txd_ch3"  IDELAY_VALUE = 0;
INST "gmii_data_bus_ch3[0].delay_gmii_txd_ch3"  IDELAY_VALUE = 0;
```

The number of tap delays are preconfigured in the example designs to meet the setup and hold constraints for the example GMII/MII pinout in the particular device. The setup/hold timing, which is achieved after place-and-route, is reported in the data sheet section of the TRCE report (created by the implement script).

Chapter 11

Detailed Example Design

This chapter provides detailed information about the deliverables provided by the CORE Generator™ tool for the QSGMII core.

Directory Structure

- <project directory>

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

  - <project directory>/<component name>

    Core release notes file

    - <component name>/doc

      Product documentation

    - <component name>/example design

      Verilog and VHDL design files

    - <component name>/implement

      Implementation script files

      - implement/results

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

    - <component name>/simulation

      Simulation scripts

      - simulation/functional

        Functional simulation files

Directory and File Contents

The core directories and their associated files are defined in the following tables.

<project directory>

The project directory contains all the CORE Generator tool project files.

Table 11-1: **Project Directory**

Name	Description
<project_dir>	
<component_name>.ngc	Top-level netlist. This is instantiated by the Verilog or VHDL example design.
<component_name>.v[hd]	Verilog or VHDL simulation model; UNISIM-based
<component_name>.v{ho|eo}	Verilog or VHDL instantiation template for the core
<component_name>.xco	Log file that records the settings used to generate a core. An XCO file is generated by the CORE Generator tool for each core that it creates in the current project directory. An XCO file can also be used as an input to the CORE Generator tool.
<component_name>_flist.txt	List of files delivered with the core

<project directory>/<component name>

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

Table 11-2: **Component Name Directory**

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

<component name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 11-3: **Doc Directory**

Name	Description
<project_dir>/<component_name>/doc	
pg029_qsgmii.pdf	QSGMII Product Guide

<component name>/example design

The example design directory contains the example design files provided with the core, and can contain files and subdirectories other than those defined in Table 11-4.

Table 11-4: **Example Design Directory**

Name	Description
	<project_dir>/<component_name>/example_design
sync_block.v[hd]	This is a synchronization flip-flop pair, used for passing signals across a clock domain.
reset_sync.v[hd]	This is a reset synchronization module for creating a synchronous reset output signal from an asynchronous input.
_example_design.ucf	Example User Constraints File (UCF) provided for the example design.
_example_design.v[hd]	Top-level file that allows example design to be implemented in a device as a standalone design.
_block.v[hd]	A block-level file that is a useful part of example design and should be instantiated in all customer designs.

<component name>/implement

The implement directory contains the core implementation script files.

Table 11-5: **Implement Directory**

Name	Description
	<project_dir>/<component_name>/implement
implement.sh	Linux shell script that processes the example design through the Xilinx tool flow. See Implementation for more information.
implement.bat	Windows batch file that processes the example design through the Xilinx tool flow. See Implementation for more information.
xst.prj	XST project file for the example design (VHDL only); it enumerates all of the VHDL files that need to be synthesized.
xst.scr	XST script file for the example design

implement/results

The results directory is created by the implement script, after which the implement script results are placed in the results directory.

Table 11-6: **Results Directory**

Name	Description
<project_dir>/<component_name>/implement/results	
routed.v[hd]	Back-annotated SIMPRIM-based model used for timing simulation
routed.sdf	Timing information for simulation

<component name>/simulation

The simulation directory and subdirectories that provide the files necessary to test a Verilog or VHDL implementation of the example design.

Table 11-7: **Simulation Directory**

Name	Description
<project_dir>/<component_name>/simulation	
arbiter_tb.v[hd]	This module outputs the input of the module in a round robin fashion.
clk_en_gen_tb.v[hd]	This module generates clock enables according to the configured speed.
clk_rst_xlnx_tb.v[hd]	This module generates clocks and resets.
decode_8b10b_tb.v[hd]	This module decodes 10 bits to 8 bits.
encode_8b10b_tb.v[hd]	This module encodes 8 bits to 10 bits.
frame_typ_pack.vhd	Functions to convert std_logic to boolean
k28p1_swapper_tb.v[hd]	Detects K28.1 and assigns data to correct channel.
mdio_cfg_tb.v[hd]	This module contains the frame for MDIO transaction.
monitor_tb.v[hd]	Monitors and reports the frames received from example design (Device Under Test (DUT))
send_frame_tb.v[hd]	Frame generation module for exciting the DUT
serdes_tb.v[hd]	Serializes and de-serializes the 10 bit data
demo_tb.v[hd]	Top-level file of the demonstration test bench for the example design. Instantiates the example design (DUT), generates clocks, resets, and test bench control semaphores.

simulation/functional

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

Table 11-8: **Functional Directory**

Name	Description
<project_dir>/<component_name>/simulation/functional	
simulate_mti.do	ModelSim macro file that compiles Verilog or VHDL sources and runs the functional simulation to completion.
wave_mti.do	ModelSim macro file that opens a wave window and adds signals of interest to it. It is called by the simulate_mti.do macro file.
simulate_ncsim.sh	IES script file that compiles the Verilog or VHDL sources and runs the functional simulation to completion.
wave_ncsim.sv	IES macro file that opens a wave window and adds signals of interest to it. It is called by the simulate_ncsim.sh script file.
simulate_vcs.sh	VCS script file that compiles the Verilog sources and runs the functional simulation to completion.
vcs_commands.key	This file is sourced by VCS at the start of simulation; it configures the simulator.
vcs_session.tcl	VCS macro file that opens a wave window and adds signals of interest to it. It is called by the simulate_vcs.sh script file.

Example Design

Figure 11-1 illustrates an example design for top-level HDL for the QSGMII using a device-specific transceiver (Virtex®-7, Kintex™-7, or Artix™-7 FPGAs).

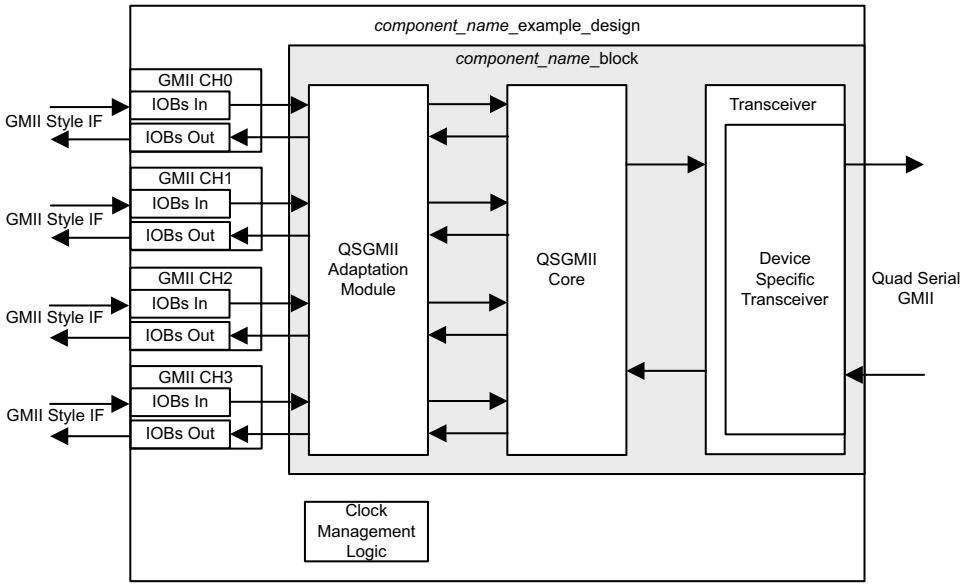

Figure 11-1: **Example Design HDL for QSGMII**

As illustrated, the example is split between two hierarchical layers. The block level is designed so that it can be instantiated directly into customer designs and performs the following functions.

- Instantiates the core from HDL
- Connects the physical-side interface of the core to a device-specific transceiver
- Connects the client side GMII of the core to an QSGMII Adaptation Module, which provides the functionality to operate at speeds of 1 Gb/s, 100 Mb/s and 10 Mb/s

The top level of the example design creates a specific example which can be simulated, synthesized and implemented. The top level of the example design performs the following functions.

- Instantiates the block level from HDL
- Derives the clock management logic for device-specific transceiver and the core
- Implements an external GMII-style interface

The next few pages in this section describe each of the example design blocks (and associated HDL files) in detail.

Top-Level Example Design HDL

The top-level example design for the QSGMII core is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/<component_name>_example_design.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/<component_name>_example_design.v
```

The example design HDL top level contains the following:

- An instance of the QSGMII block level
- Clock management logic for the core and the device-specific transceiver, including DCM (if required) and Global Clock Buffer instances
- External GMII logic, including IOB and Double Data Rate (DDR) register instances, where required

The example design HDL top level connects the GMII interfaces of the block level to external IOBs. This allows the functionality of the core to be demonstrated using a simulation package, as described in this guide.

Block Level HDL

The following files describe the block level for the QSGMII core:

VHDL

```
<project_dir>/<component_name>/example_design/<component_name>_block.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/<component_name>_block.v
```

The block level contains the following:

- An instance of the QSGMII core
- An instance of a transceiver specific to the target device (Virtex-7, Kintex-7, or Artix-7)
- An QSGMII adaptation module containing four instances of SGMII adaptation module. Each instance of SGMII adaptation module contains
  - The clock management logic required to enable the instance of SGMII operate at 10 Mb/s, 100 Mb/s, and 1 Gb/s.
  - GMII logic for both transmitter and receiver paths.

- In MAC mode the GMII style 8-bit interface is run at 125 MHz for 1 Gb/s operation; 12.5 MHz for 100 Mb/s operation; 1.25 MHz for 10 Mb/s operation.
- In PHY mode the GMII style 8 bit interface is run at 125 MHz for 1 Gb/s operation; 25 MHz for 100 Mb/s operation; 2.5 MHz for 10 Mb/s operation. For 100/10 Mb/s operation 4 bits of the MII are mapped to the LSB 4 bits of the GMII style interface.

The block-level HDL connects the PHY side interface of the core to a device-specific transceiver instance and the client side to QSGMII adaptation logic as illustrated in Figure 11-1. This is the most useful part of the example design and should be instantiated in all customer designs that use the core.

Transceiver Files for Virtex-7, Kintex-7, and Artix-7 Devices

Transceiver Wrapper

This device-specific transceiver wrapper is instantiated from the block-level HDL file of the example design and is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_transceiver.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_transceiver.v
```

This file instances output source files from the transceiver wizard (used with QSGMII attributes).

Virtex-7 and Kintex-7 FPGA Transceiver Wizard Files

For Virtex-7 and Kintex-7 devices, the transceiver wrapper file directly instantiates device-specific transceiver wrapper files created from the transceiver wizard. These files tie off (or leave unconnected) unused I/O for the transceiver, and apply the QSGMII attributes. The files can be edited/tailored by rerunning the wizard and swapping these files. The files include the following:

VHDL

```
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_gtwizard_init.vhd
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_tx_startup_fsm.vhd
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_rx_startup_fsm.vhd
```

```
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_gtwizard.vhd
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_gtwizard_gt.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_gtwizard_init.v
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_tx_startup_fsm.v
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_rx_startup_fsm.v
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_gtwizard.v
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_gtwizard_gt.v
```

To re-run the transceiver wizard, a CORE Generator tool XCO file for the wizard is included. This file defines all the required wizard attributes used to generate the preceding files. See the CORE Generator tool documentation for further information about XCO files. The XCO file is in the following location:

```
<project_dir>/<component_name>/example_design/transceiver/
<component_name>_gtwizard.xco
```

QSGMII Adaptation Module

An QSGMII adaptation module containing four instances of SGMII adaptation module. Each instance of SGMII adaptation module contains the following:

- The clock management logic required to enable the instance of SGMII operate at 10 Mb/s, 100 Mb/s, and 1 Gb/s.
- GMII logic for both transmitter and receiver paths.
  - In MAC mode the GMII style 8-bit interface is run at 125 MHz for 1 Gb/s operation; 12.5 MHz for 100 Mb/s operation; 1.25 MHz for 10 Mb/s operation. The speed of operation is controlled by clock enables. The reference clock out (`sgmii_clk_chx`) is always 125 MHz.
  - In PHY mode the GMII style 8 bit interface is run at 125 MHz for 1 Gb/s operation; 25 MHz for 100 Mb/s operation; 2.5 MHz for 10 Mb/s operation. For 100/10 Mb/s operation LSB 4 bits of the GMII style interface is mapped to 4 bits of the MII.

The QSGMII Adaptation Module is described in the following files:

VHDL

```
<project_dir>/<component_name>/example_design/sgmii_adapt/
    <component_name>_qsgmii_adapt.vhd
    <component_name>_sgmii_adapt.vhd
    <component_name>_clk_gen.vhd
    <component_name>_clk_div.vhd
    <component_name>_johnson_cntr.vhd
    <component_name>_tx_rate_adapt.vhd
    <component_name>_rx_rate_adapt.vhd
```

Verilog

```
<project_dir>/<component_name>/example_design/sgmii_adapt/
    <component_name>_qsgmii_adapt.v
    <component_name>_sgmii_adapt.v
    <component_name>_clk_gen.v
    <component_name>_clk_div.v
    <component_name>_johnson_cntr.v
    <component_name>_tx_rate_adapt.v
    <component_name>_rx_rate_adapt.v
```

The GMII of the core always operates at 125 MHz. The core makes no differentiation between the three speeds of operation; it always effectively operates at 1 Gb/s. However, at 100 Mb/s, every data byte run through the core should be repeated 10 times to achieve the required bit rate; at 10 Mb/s, each data byte run through the core should be repeated 100 times to achieve the required bit rate. Dealing with this repetition of bytes is the function of the SGMII adaptation module and its component blocks.

Demonstration Test Bench

Figure 11-2 illustrates the demonstration test bench for the QSGMII core. The demonstration test bench is a simple VHDL or Verilog program to exercise the example design and the core itself.

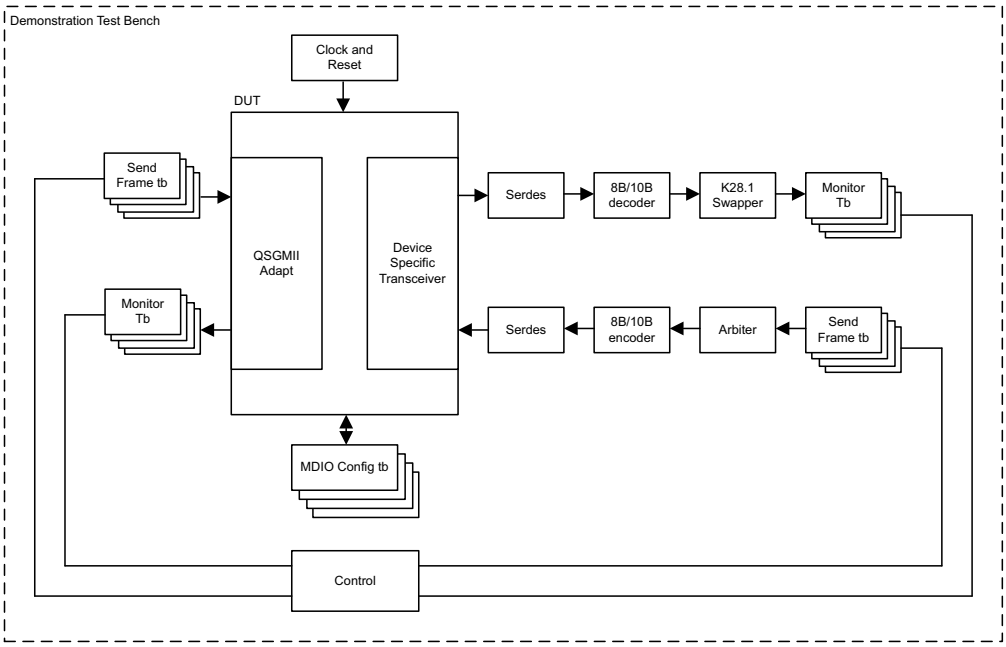

Figure 11-2: **Demonstration Test Bench for QSGMII**

The top-level test bench entity instantiates the example design for the core, which is the Device Under Test (DUT). Other modules needed to provide stimulus, clocks, resets and test bench semaphores are also instantiated in the top-level test bench. The following files describe the top-level of the demonstration test bench.

VHDL

```
<project_dir>/<component_name>/simulation/demo_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/demo_tb.v
```

Send frame test bench generates the stimulus to excite the transceiver on the DUT receive side and data input on the DUT QSGMII adapt side. Four instances of the send frame test bench are instantiated, with each instance representing one channel.

VHDL

```
<project_dir>/<component_name>/simulation/send_frame_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/send_frame_tb.v
```

The Arbiter module selects one byte from each instance of the send frame test bench and passes it on to the 8B/10B encoder module.

VHDL

```
<project_dir>/<component_name>/simulation/arbiter_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/arbiter_tb.v
```

The 8B/10B encoder test bench module converts 8-bit data from arbiter to 10 bits as specified by *IEEE 802.3-2008* standard clause 36.

VHDL

```
<project_dir>/<component_name>/simulation/encode_8b10b_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/encode_8b10b_tb.v
```

The 8B/10B decoder test bench module converts 10-bit data from SERDES on the transceiver transmit interface to 10 bits as specified by *IEEE 802.3-2008* standard clause 36.

VHDL

```
<project_dir>/<component_name>/simulation/decode_8b10b_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/decode_8b10b_tb.v
```

The SERDES module serializes the 10-bit data from the 8B/10B encoder and maps it to the receive interface of the DUT transceiver. This module de-serializes the serial bitstream from the transmit interface of the DUT transceiver and maps it to the 8B/10B decoder.

VHDL

```
<project_dir>/<component_name>/simulation/serdes_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/serdes_tb.v
```

The K28.1 swapper module swaps K28.1 characters received on port 0 with K28.5 as specified in the QSGMII specification.

VHDL

```
<project_dir>/<component_name>/simulation/k28p1_swapper_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/k28p1_swapper_tb.v
```

The Monitor test bench module monitors the output from the DUT and verifies the data with preloaded data structures present in the module.

VHDL

```
<project_dir>/<component_name>/simulation/monitor_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/monitor_tb.v
```

The programing of per channel configuration registers in the DUT is performed through MDIO configuration test bench. There are four instances of this module with each instance representing one channel.

VHDL

```
<project_dir>/<component_name>/simulation/mdio_cfg_tb.vhd
```

Verilog

```
<project_dir>/<component_name>/simulation/mdio_cfg_tb.v
```

Test Bench Functionality

The demonstration test bench performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- Each channel of the QSGMII core is configured through the MDIO interface by injecting an MDIO frame into the example design. This disables Auto-Negotiation and takes the core out of Isolate state.
- The speed of the interface is programmed as follows
  - MAC mode
    - Channel 0 - 1 Gb/s
    - Channel 1 - 100 Mb/s
    - Channel 2 - 10 Mb/s
    - Channel 3 - 1 Gb/s

  - PHY mode with GMII; all channels at 1 Gb/s
  - PHY mode with MII
    - Channel 0 - 10 Mb/s
    - Channel 1- 100 Mb/s
    - Channel 2 - 10 Mb/s
    - Channel 3 - 100 Mb/s
- The following frames are injected into the transmitter by the send frame block.
  - the first is a minimum length frame
  - the second is a type frame
  - the third is an errored frame
  - the fourth is a padded frame
- The serial data received at the device-specific transceiver transmitter interface is converted to 10-bit parallel data, then 8B/10B decoded. The resultant byte is aligned to the corresponding channel based on the K28.1 character set and also the K28.1 character set is replaced with K28.5. The resulting frames are checked by in the monitor test bench against the stimulus frames injected into the transmitter to ensure data integrity.
- The same four frames are generated by the frame generator module in the receive side of the transceiver. The data from all four instances are aggregated into 32 bits, are 8B/10B encoded, converted to serial data, and injected into the device-specific transceiver receiver interface at 5 Gb/s.
- Data frames received at the receiver GMII interface are checked by the Monitor against the stimulus frames injected into the device-specific transceiver receiver to ensure data integrity.

Customizing the Test Bench

Changing Frame Data

You can change the contents of the four frames used by the demonstration test bench by changing the *data* and *valid* fields for each frame defined in the send frame module. New frames can be added by defining a new frame of data. Modified frames are automatically updated in both the stimulus and monitor functions.

Changing Frame Error Status

Errors can be inserted into any of the predefined frames in any position by setting the *error* field to ‘1’ in any column of that frame. Injected errors are automatically updated in both the stimulus and monitor functions.

Changing the Core Configuration

The configuration of the QSGMII core used in the demonstration test bench can be altered.

CAUTION! *Certain configurations of the core cause the test bench to fail, or to cause processes to run indefinitely. For example, the demonstration test bench will not Auto-Negotiate with the design example. Determine the configurations that can safely be used with the test bench.*

The core can be reconfigured by editing the injected MDIO frame in the demonstration test bench top level.

Changing the Operational Speed

The speed of both the example design and test bench can be set to the desired operational speed by editing the following settings, recompiling the test bench, and then running the simulation again. The changes also need to be implemented in the *example design*.

1 Gb/s Operation

```
set speed_is_10_100_chx to logic 0
```

100 Mb/s Operation

```
set speed_is_10_100_chx to logic 1
set speed_is_100_chx to logic 1
```

10 Mb/s Operation

```
set speed_is_10_100_chx to logic 1
set speed_is_100_chx to logic 0
```

Implementation

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

Linux

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

Windows

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

The implement script performs the following steps:

1. The HDL example design files are synthesized using XST.
2. NGDBuild is run to consolidate the core netlist and the example design netlist into the NGD file containing the entire design.
3. The design is mapped to the target technology.
4. The design is placed-and-routed on the target device.
5. Static timing analysis is performed on the routed design using `trce`.
6. A bitstream is generated.
7. Netgen runs on the routed design to generate a VHDL or Verilog netlist (as appropriate for the Design Entry project setting) and timing information in the form of SDF files.

The Xilinx tool flow generates several output and report files. These are saved in the following directory, which is created by the implement script:

```
<project_dir>/<component_name>/implement/results
```

Functional Simulation

The test script is a ModelSim, IES or VCS macro that automates the simulation of the test bench and is in the following location:

```
<project_dir>/<component_name>/simulation/functional/
```

The test script performs the following tasks:

- Compiles the structural UNISIM simulation model
- Compiles HDL example design source code
- Compiles the demonstration test bench
- Starts a simulation of the test bench
- Opens a Wave window and adds signals of interest (`wave_mti.do/wave_ncsim.sv`)
- Runs the simulation to completion

SECTION IV: APPENDICES

Verification, Compliance, and Interoperability

Migrating

Implementing External GMII/MII

Debugging

Additional Resources

Appendix A

Verification, Compliance, and Interoperability

Simulation

A highly parameterizable transaction-based test bench was used to test the core. Testing included the following:

- Register Access
- Loss of Synchronization
- Auto-Negotiation and error handling
- Frame Transmission and error handling
- Frame Reception and error handling
- Clock Compensation in the Elastic Buffers

Hardware Testing

This core has not been validated in hardware.

Compliance Testing

This core has not been validated in hardware.

Appendix B

Migrating

See *Vivado Design Suite Migration Methodology Guide* (UG911).

For a complete list of Vivado User and Methodology Guides, see the Vivado Design Suite - 2012.2 User Guides web page.

Implementing External GMII/MII

In certain applications, the Client-Side GMII/MII datapath can be used as a true GMII/MII to connect externally off-device across a PCB. This external GMII/MII functionality is included in the HDL example design delivered with the core by the Xilinx CORE Generator™ tool or Vivado™ IP Catalog. This extra logic required to accomplish this is described in this Appendix.

Note: Virtex®-7 devices support GMII at 3.3 V or lower only in certain parts and packages; see the Virtex-7 Device Documentation. Kintex™-7 devices support GMII at 3.3 V or lower.

External GMII Transmitter Logic (Virtex-7, Kintex-7, and Artix-7 Devices)

When implementing an external GMII, the GMII transmitter signals will be synchronous to their own clock domain. The core must be used with a Transmitter Elastic Buffer to transfer these GMII transmitter signals onto the cores internal 125 MHz reference clock (`userclk2`). A Transmitter Elastic Buffer is provided by the example design that is with the core.

Using a combination of IODELAY elements on the data and using BUFIO and BUFR regional clock routing for the `gtx_clk_chx` input clock, are illustrated in Figure C-1.

In this implementation, a BUFIO is used to provide the lowest form of clock routing delay from the input clock to the input GMII Tx signal sampling at the device IOBs. Note, however, that this creates placement constraints; a BUFIO capable clock input pin must be selected, and all other input GMII Tx signals must be placed in the respective BUFIO region. The device FPGA user guides should be consulted.

The clock is then placed onto regional clock routing using the BUFR component and the input GMII Tx data immediately resampled as illustrated.

The IODELAY elements can be adjusted to fine-tune the setup and hold times at the GMII IOB input flip-flops. The delay is applied to the IODELAY element using constraints in the UCF; these can be edited if desired. See Constraints When Using External GMII/MII for more information.

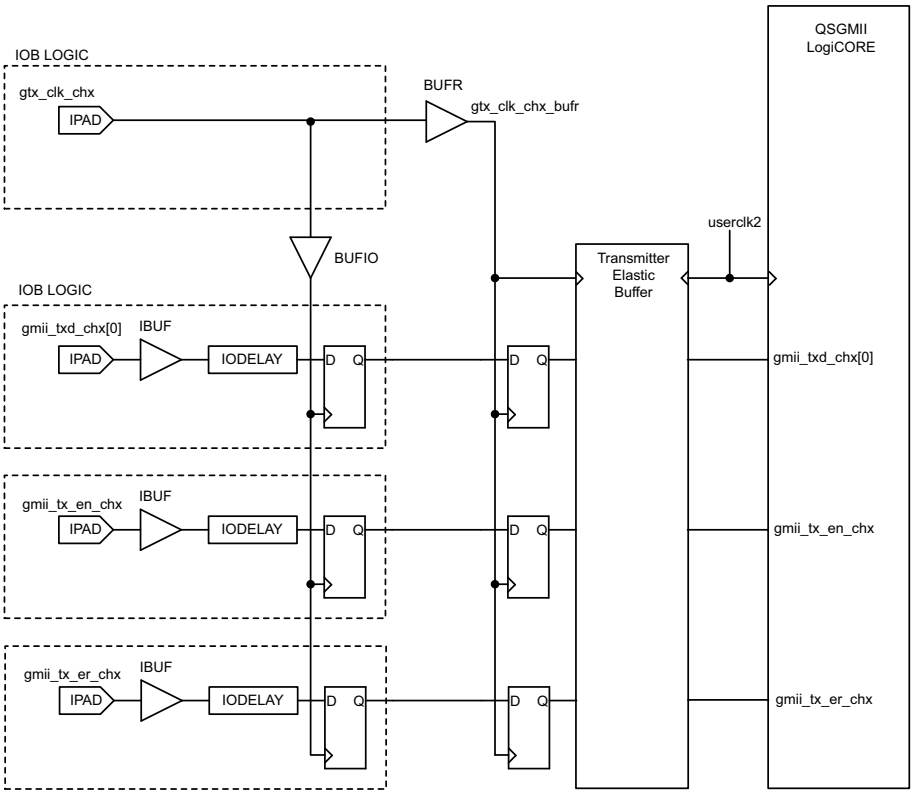

Figure C-1: **External GMII Transmitter Logic**

External MII Transmitter Logic (Virtex-7, Kintex-7, and Artix-7 Devices)

When implementing an external MII, the MII transmitter signals will be synchronous to the core's internal 125 MHz clock (`userclk2`).

IODELAY elements on the data are used to delay the data by fixed duration depending on the application. The delayed data is then sampled on core internal clock (`userclk2`), as illustrated in Figure C-2.

The IODELAY elements can be adjusted to fine-tune the setup and hold times at the MII IOB input flip-flops. The delay is applied to the IODELAY element using constraints in the UCF; these can be edited if desired. See Constraints When Using External GMII/MII for more information.

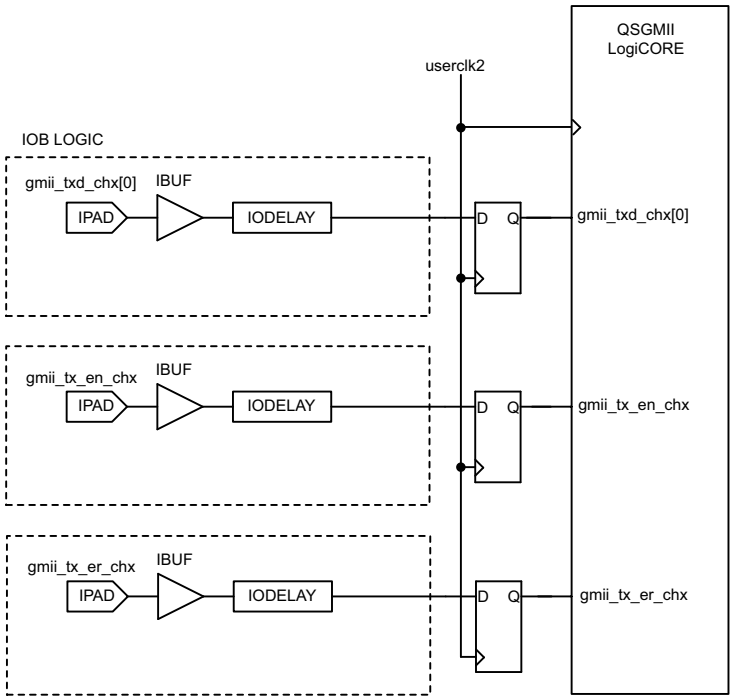

Figure C-2: **External MII Transmitter Logic**

External GMII/MII Receiver Logic

Figure C-3 illustrates an external GMII receiver created in a Virtex-7 device. The signal names and logic shown in the figure exactly match those delivered with the example design when the GMII is selected. If other families are selected, equivalent primitives and logic specific to that family are automatically used in the example design.

Figure C-3 also shows that the output receiver signals are registered in device IOBs before driving them to the device pads. All receiver logic is synchronous to a single clock domain.

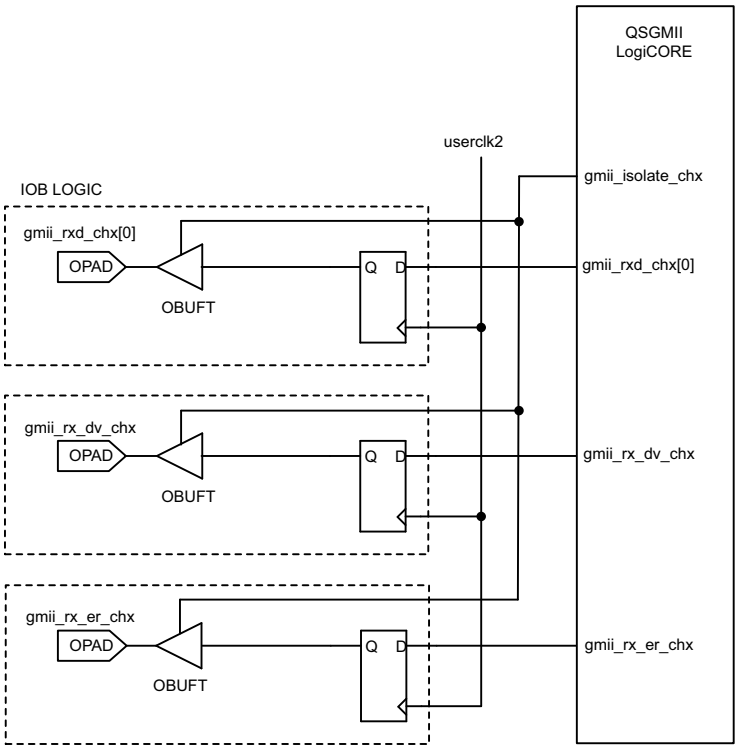

Figure C-3: **External GMII/MII Receiver Logic**

Debugging

This appendix provides assistance for debugging the core within a system. For additional help, contact Xilinx by submitting a WebCase at support.xilinx.com/.

General Checks

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

Problems with the MDIO

- Ensure that the MDIO is driven properly. See MDIO Management System for detailed information about performing MDIO transactions.
- Check that the `mdc` clock is running and that the frequency is 2.5 MHz or less.
- 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 PHYAD field placed into the MDIO frame matches the value placed on the `phyad[4:0]` port of the core.

Problems with Data Reception or Transmission

When no data is being received or transmitted on a channel:

- Ensure that a valid link has been established between the core and its link partner, either by Auto-Negotiation or Manual Configuration; `status_vector_chx[0]` and `status_vector_chx[1]` should both be high. If no link has been established, see the topics discussed in the next section.
  - Problems with Auto-Negotiation
  - Problems in Obtaining a Link (Auto-Negotiation Disabled)

Note: Transmission through the core is not allowed unless a link has been established. This behavior can be overridden by setting the Unidirectional Enable bit.

- Ensure that the Isolate state has been disabled.

  By default, the Isolate state is enabled after power-up. In the PHY mode, the PHY is electrically isolated from the GMII; for an internal GMII, it behaves as if it is isolated. This results in no data transfer across the GMII.

Problems with Auto-Negotiation

Determine whether Auto-Negotiation has completed successfully by doing one of the following.

- Poll the Auto-Negotiation completion bit 1.5 in MDIO Register 1: Status Register.
- Use the Auto-Negotiation interrupt port of the core.

If Auto-Negotiation is not completing:

1. Ensure that Auto-Negotiation is enabled in *both* the core and in the link partner (the device or test equipment connected to the core). Auto-Negotiation cannot complete successfully unless both devices are configured to perform Auto-Negotiation.

   The Auto-Negotiation procedure requires that the Auto-Negotiation handshaking protocol between the core and its link partner, which lasts for several link timer periods, occur without a bit error. A detected bit error causes Auto-Negotiation to go back to the beginning and restart. Therefore, a link with an exceptionally high bit error rate might not be capable of completing Auto-Negotiation, or might lead to a long Auto-Negotiation period caused by the numerous Auto-Negotiation restarts. If this appears to be the case, try the next step and see Problems with a High Bit Error Rate.

2. Try disabling Auto-Negotiation in both the core and the link partner and see if both devices report a valid link and are able to pass traffic. If they do, it proves that the core and link partner are otherwise configured correctly. If they do not pass traffic, see Problems in Obtaining a Link (Auto-Negotiation Disabled)).

Problems in Obtaining a Link (Auto-Negotiation Disabled)

Determine whether the device has successfully obtained a link with its link partner by doing the following:

- Reading bit 1.2, Link Status, in MDIO Register 1: Status Register using the optional MDIO management interface (or look at `status_vector_chx[1]`).
- Monitoring the state of `status_vector_chx[0]`. If this is logic 1, then synchronization, and therefore a link, has been established. See Bit[0]: Link Status.

If the devices have failed to form a link then do the following:

- Ensure that Auto-Negotiation is disabled in *both* the core and in the link partner (the device or test equipment connected to the core).
- Monitor the state of the `signal_detect` signal 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.
  - Signal must be 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.
  - See Problems with a High Bit Error Rate in a subsequent section.

When using a device-specific transceiver, perform these additional checks:

- Ensure that the polarities of the `TXN/TXP` and `RXN/RXP` lines are not reversed. If they are, this can be fixed by using the `TXPOLARITY` and `RXPOLARITY` ports of the device-specific transceiver.
- Check that the device-specific transceiver is not being held in reset by monitoring the `mgt_tx_reset` and `mgt_rx_reset` signals between the core and the device-specific transceiver. If these are asserted, this indicates that the PMA Phase-Locked Loop (PLL) circuitry in the device-specific transceiver has not obtained lock; check the PLL Lock signals output from the device-specific transceiver.

Problems with a High Bit Error Rate

Symptoms

The severity of a high-bit error rate can vary and cause any of the following symptoms:

- Failure to complete Auto-Negotiation when Auto-Negotiation is enabled.
- Failure to obtain a link when Auto-Negotiation is disabled in both the core and the link partner.
- High proportion of lost packets when passed between two connected devices that are capable of obtaining a link through Auto-Negotiation or otherwise. This can usually be accurately measured if the Ethernet MAC attached to the core contains statistic counters.

  Note: All bit errors detected by the QSGMII Receive channel logic during frame reception show up as Frame Check Sequence Errors in an attached Ethernet MAC.

Debugging

- Compare the problem across several devices or PCBs to ensure that the problem is not a one-off case.
- Try using an alternative link partner or test equipment and then compare results.
- Try swapping the optical module on a misperforming device and repeat the tests.

Perform these additional checks when using a device-specific transceiver:

- Directly monitor the following ports of the device-specific transceiver by attaching error counters to them, or by triggering on them using the ChipScope™ debugging tool or an external logic analyzer.

  ```
  RXDISPERR

  RXNOTINTABLE
  ```

  These signals should not be asserted over the duration of a few seconds, minutes or even hours. If they are frequently asserted, it might indicate a problem with the device-specific transceiver.

- Place the device-specific transceiver into parallel or serial loopback.
  - If the core exhibits correct operation in device-specific transceiver serial loopback, but not when loopback is performed through an optical cable, it might indicate a faulty optical module.
  - If the core exhibits correct operation in device-specific transceiver parallel loopback but not in serial loopback, this can indicate a device-specific transceiver problem.

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.

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.

References

To search for Xilinx documentation, go to http://www.xilinx.com/support

- *7 Series FPGAs GTP Transceivers User Guide* (UG482)
- *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)
- *Tri-Mode Ethernet MAC User Guide* (UG138)
- *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices* (UG687)
- Vivado Design Suite - 2012.2 User Guides web page

Specifications

- *IEEE 802.3-2008*
- *Serial-GMII Specification* Revision 1.7 (CISCO SYSTEMS, ENG-46158)
- *QSGMII Specification* Revision 1.2 (CISCO SYSTEMS, EDCS-540123)

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 the product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation or if changes are made to any section of the design labeled DO NOT MODIFY.

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

- New Features
- Resolved Issues
- Known Issues

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/18/12	1.0	Initial Xilinx release
04/24/12	1.2	Added support for Virtex-7 FPGA GTH Transceiver.
07/25/12	1.3	Added Vivado tools support. Added support for Artix-7 devices.

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 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.