LogiCORE IP Quad Serial Gigabit Media Independent v3.1

Product Guide for Vivado Design Suite

PG029 December 18, 2013

XILINX®

XILINX

Table of Contents

Appendix E: Additional Resources

Introduction

The LogiCORE™ 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) [Ref 1].

Features

- Integrated transceiver interface using a Zynq®-7000 All Programmable SoC, Virtex®-7, and Kintex®-7 device GTX transceiver
- Integrated transceiver interface using Kintex UltraScale™ and 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

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family(1)	Kintex UltraScale, Zynq-7000, 7 Series
Supported User Interfaces	GMII/MII
Resources	See Table 2-1, Table 2-2, Table 2-3, and Table 2-4.
Provided with Core	
Design Files	Encrypted RTL
Example Design	VHDL and Verilog
Test Bench	Demonstration Test Bench in VHDL and Verilog
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Verilog and VHDL
Supported S/W Drivers	NA
Tested Design Flows(2)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The QSGMII IP core is designed to reduce significantly the number of signals that are needed between multi port 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

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

The QSGMII 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.

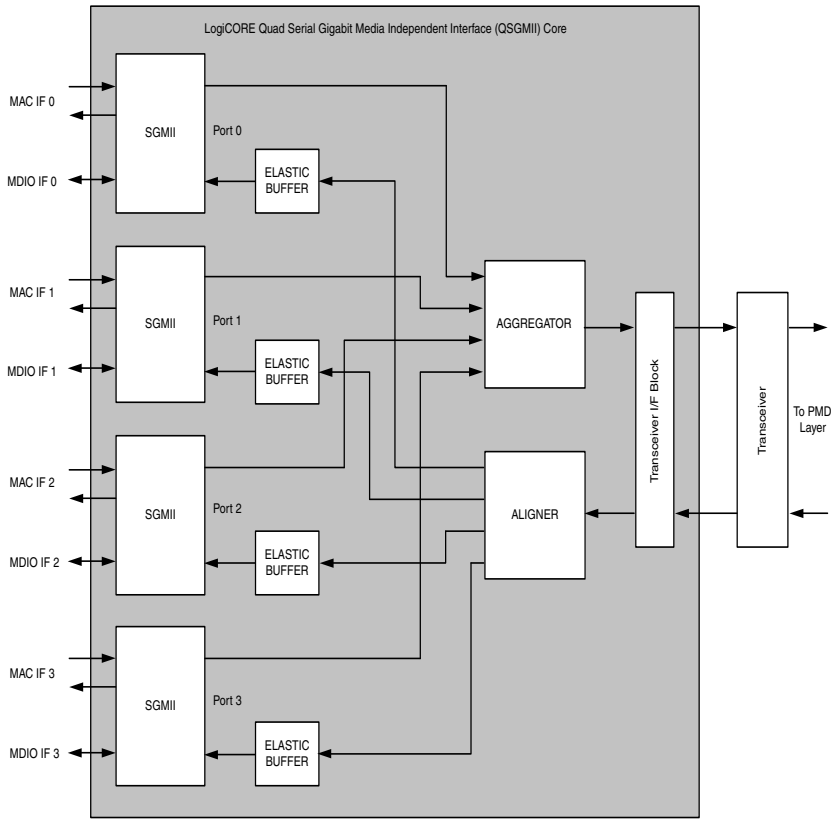


Figure 1-1: **QSGMII System Overview**

SGMII

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

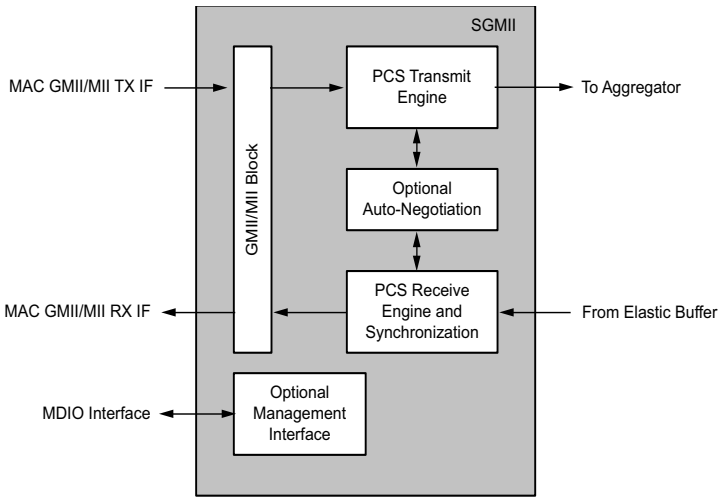


Figure 1-2: **Functional Diagram of SGMII Block**

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 Family home page. Zynq®-7000, 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 Zynq-7000, Virtex-7, Kintex-7, or Artix-7 device 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 Zynq-7000, Virtex-7, and Kintex-7 device 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.
- Supports maximum frame size of 2.8 KB for 10 Mb/s, 28 KB for 100 Mb/s and 280 KB for 1 Gb/s per single lane.

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.

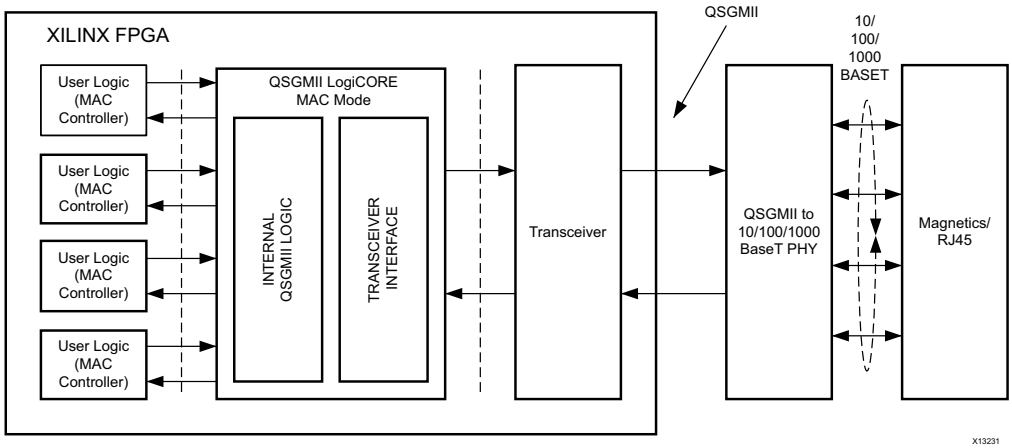


Figure 1-3: **Typical Application of QSGMII in MAC Mode**

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.

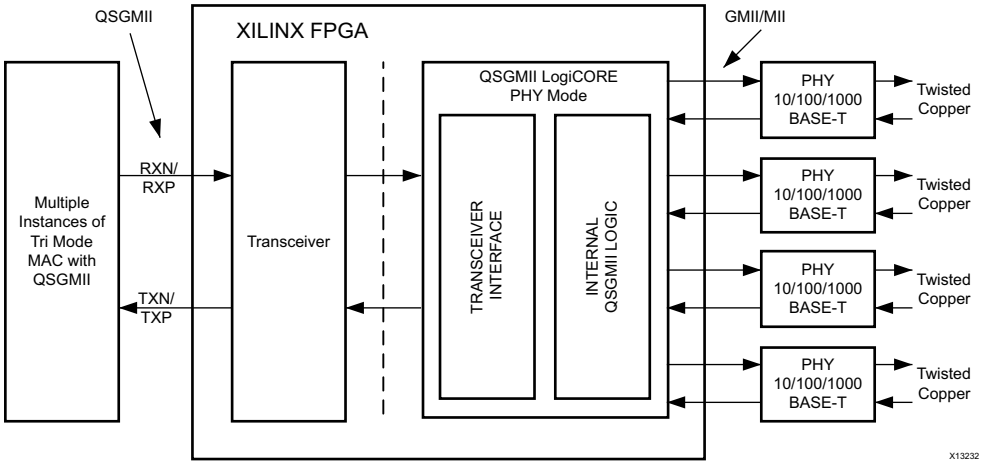


Figure 1-4: **Typical Application of QSGMII in PHY Mode**

Licensing and Ordering Information

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

- *Ethernet Standard 802.3-2008* Clauses 22, 35, 36 and 38 [Ref 2]
- Cisco *Serial GMII Specification* Revision 1.8 (SGMII) [Ref 3]
- Cisco *Quad SGMII Specification* Revision 1.2 (QSGMII) [Ref 4]

Performance

Latency

These measurements are for the core only; they do not include the latency through the Zynq®-7000, Virtex®-7, Kintex®-7, or Artix®-7 device transceiver, or the Transmitter Elastic Buffer added in the QSGMII core.

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

Resources required for this core have been estimated for different devices listed in Table 2-1 through Table 2-4. 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. These values were generated using Xilinx Vivado® IP catalog. They are derived from post-synthesis reports, and might change during MAP and PAR.

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.

Note: UltraScale™ architecture results are expected to be similar to 7 series device results.

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**

Parameter Values			Device Resources				
Mode	**MDIO Interface**	**Auto-Negotiation**	**Slices**	**FFs**	**LUTs**	**LUTRAM**	**BUFGs**
MAC MODE	Yes	Yes	1580	3353	2534	442	0
	Yes	No	1227	2481	1684	442	0
	No	Yes	1321	2841	2041	410	0
	No	No	1076	2237	1449	410	0
PHY GMII Mode	Yes	Yes	1570	4049	2549	442	0
	Yes	No	1190	3145	1684	442	0
	No	Yes	1294	3513	2012	410	0
	No	No	1107	2885	1440	410	0
PHY MII Mode	Yes	Yes	1574	3501	2657	442	0
	Yes	No	1164	2613	1795	442	0
	No	Yes	1306	2981	2131	410	0
	No	No	1068	2369	1562	410	0

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFG is required to drive the `txoutclk`. See Figure 5-1.
3. Additional BUFG is required to drive the free running `independent_clock`.
4. These two BUFGs can be shared across multiple instances of the core.
5. Additional BUFG can be added for `rxoutclk`. Alternately a BUFMR and BUFR in series can be used. BUFMR and BUFR are added by default if you selects **Include Shared Logic in Core**; otherwise you can manually instantiate the BUFGs.

Kintex-7 Devices

Table 2-2 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	1529	3353	2550	442	0
	Yes	No	1131	2481	1707	442	0
	No	Yes	1278	2841	2039	410	0
	No	No	1084	2237	1443	410	0
PHY GMII Mode	Yes	Yes	1662	4048	2547	442	0
	Yes	No	1275	3144	1688	442	0
	No	Yes	1407	3512	2013	410	0
	No	No	1137	2884	1439	410	0
PHY MII Mode	Yes	Yes	1641	3501	2675	442	0
	Yes	No	1153	2613	1821	442	0
	No	Yes	1291	2981	2135	410	0
	No	No	1066	2369	1558	410	0

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFG is required to drive the `txoutclk`. See Figure 5-2.
3. Additional BUFG is required to drive the free running `independent_clock`.
4. These 2 BUFGs can be shared across multiple instances of the core.
5. Additional BUFG can be added for `rxoutclk`. BUFG is added by default if you select **Include Shared Logic in Core**; otherwise you can manually instantiate the BUFGs.

Artix-7 Devices

Table 2-3 provides approximate utilization figures for various core options when a single instance of the core is instantiated in an 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	1456	3354	2525	442	0
	Yes	No	1120	2482	1678	442	0
	No	Yes	1212	2842	2035	410	0
	No	No	951	2238	1444	410	0
PHY GMII Mode	Yes	Yes	1490	4051	2536	442	0
	Yes	No	1162	3147	1711	442	0
	No	Yes	1339	3515	2026	410	0
	No	No	1076	2887	1447	410	0
PHY MII Mode	Yes	Yes	1498	3502	2658	442	0
	Yes	No	1133	2614	1794	442	0
	No	Yes	1264	2982	2139	410	0
	No	No	1005	2370	1562	410	0

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFG is required to feed the input of MMCM. See Figure 5-3.
3. Two additional BUFGs are required to drive the outputs of MMCM. See Figure 5-3.
4. Additional BUFG is required to drive the free running `independent_clock`.
5. These four BUFGs can be shared across multiple instances of the core.
6. Additional BUFR is used to divide the `rxoutclk` for driving `rxusrclk2`.
7. Additional BUFR is used to drive the `rxusrclk`.
8. The above BUFRs are added by default if you select **Include Shared Logic in Core**; otherwise you can manually instantiate the BUFRs. Additional BUFGs can be cascaded to the BUFRs if `rxoutclk` can be shared across multiple instances of the core.

Zynq-7000 All Programmable SoCs

Table 2-4 provides approximate utilization figures for various core options when a single instance of the core is instantiated in a Zynq-7000 device.

Table 2-4: **Resource Utilization for Zynq-7000**

Parameter Values			Device Resources				
Mode	**MDIO Interface**	**Auto-Negotiation**	**Slices**	**FFs**	**LUTs**	**LUTRAM**	**BUFGs**
MAC MODE	Yes	Yes	1596	3353	2536	442	0
	Yes	No	1248	2452	1657	442	0
	No	Yes	1330	2812	2005	410	0
	No	No	1147	2208	1416	410	0
PHY GMII Mode	Yes	Yes	1637	4048	2544	442	0
	Yes	No	1201	3144	1683	442	0
	No	Yes	1351	2981	2011	410	0
	No	No	1086	3512	1440	410	0
PHY MII Mode	Yes	Yes	1576	2884	2659	442	0
	Yes	No	1127	2613	1802	442	0
	No	Yes	1252	2981	2128	410	0
	No	No	992	2369	1561	410	0

1. The number of BUFGs indicated are at the block level of the core.
2. Additional BUFG is required to drive the `txoutclk`. See Figure 5-2.
3. Additional BUFG is required to drive the free running `independent_clock`.
4. These two BUFGs can be shared across multiple instances of the core.
5. Additional BUFG can be added for `rxoutclk`. BUFG is added by default if you select **Include Shared Logic in Core**; otherwise you can manually instantiate the BUFGs.

Port Descriptions

Internal Encrypted Hierarchy of the Core Level Ports

All ports in the encrypted hierarchy of the core are internal connections in FPGA logic. Un-encrypted HDL in the core and example design (delivered with the core) connect 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 the ports described here indicate the pins at the in the encrypted hierarchy of the core level. The block level design instantiates the core and transceiver.

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.

Figure 2-1 shows the pinout for the QSGMII core using a device-specific transceiver *with* the optional MDIO Management and optional Auto-Negotiation.

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

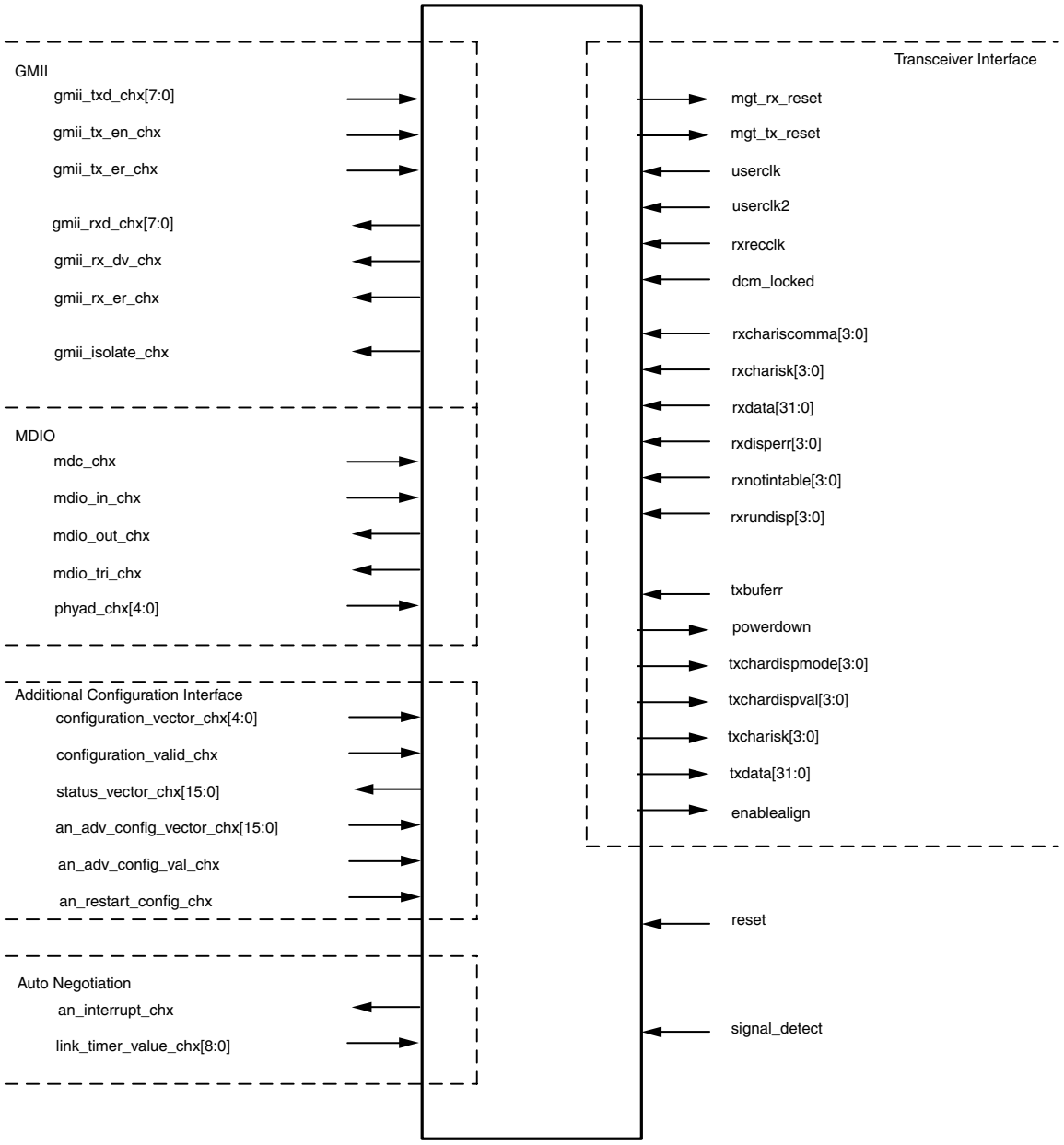


Figure 2-1: **Component Pinout of QSGMII Core *with* Optional MDIO and Auto-Negotiation**

Figure 2-2 shows the pinout for the QSGMII core using a device-specific transceiver *with only* the optional MDIO Management.The port name for multiple instances of an interface is generalized as "CHx". "CHx" takes the value "CH0", "CH1", "CH2", and "CH3".

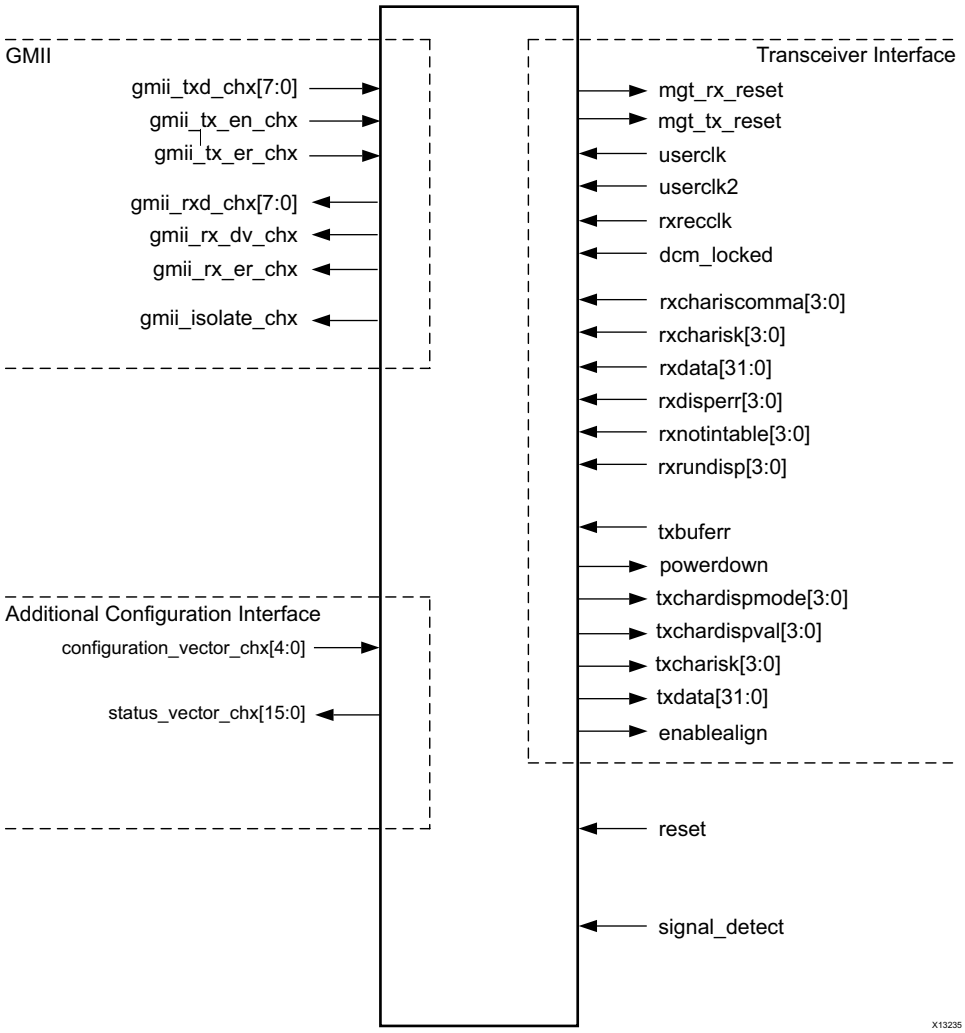


Figure 2-2: **Component Pinout of QSGMII Core *with* only Optional MDIO Management**

Figure 2-3 shows the pinout for the QSGMII core using a device-specific transceiver *with only* the optional Auto-Negotiation.

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

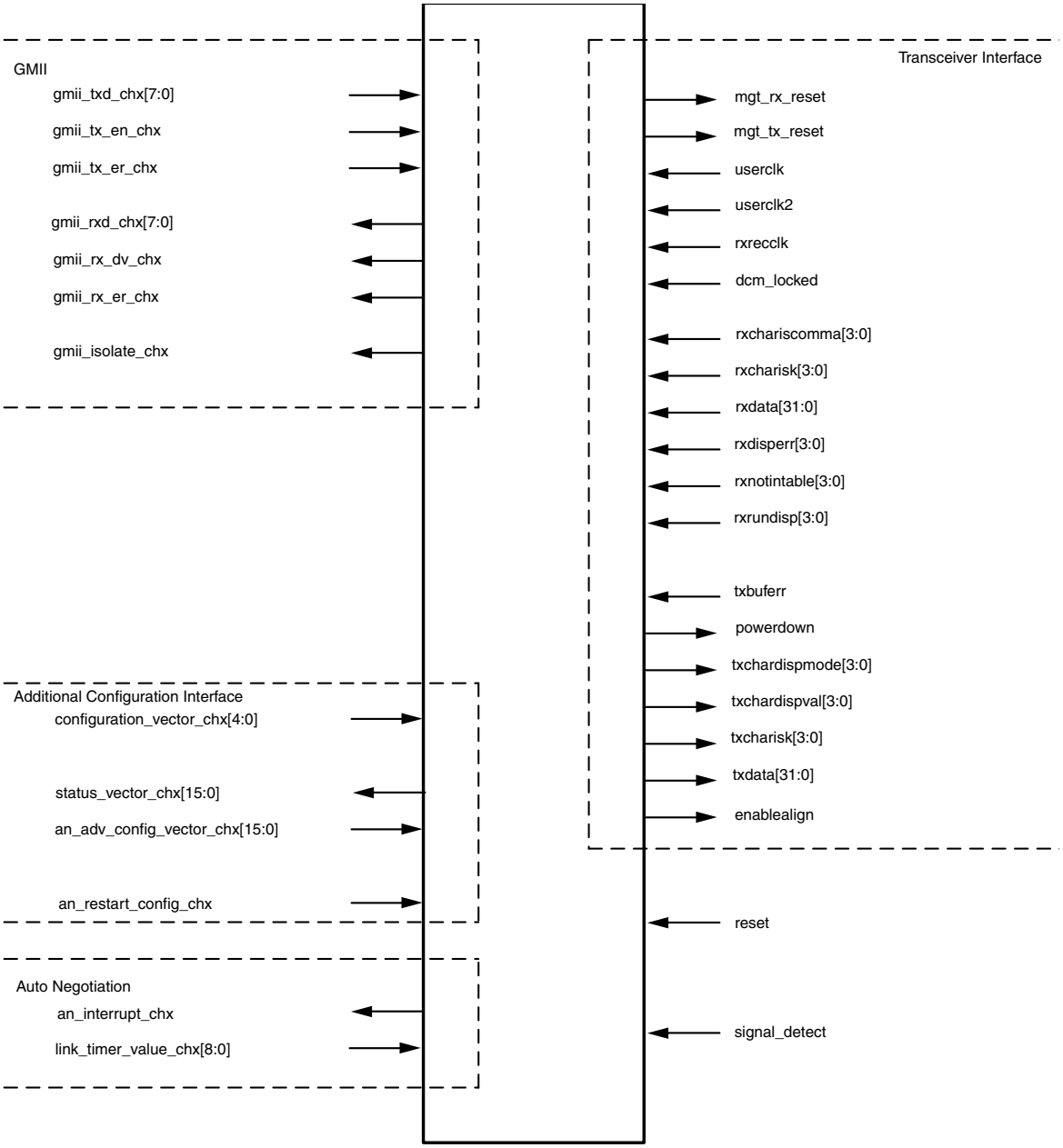


Figure 2-3: **Component Pinout of QSGMII Core *with* only Optional Auto-Negotiation**

Figure 2-4 shows the pinout for the QSGMII core using a device-specific transceiver *without* optional MDIO or Auto-Negotiation.

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

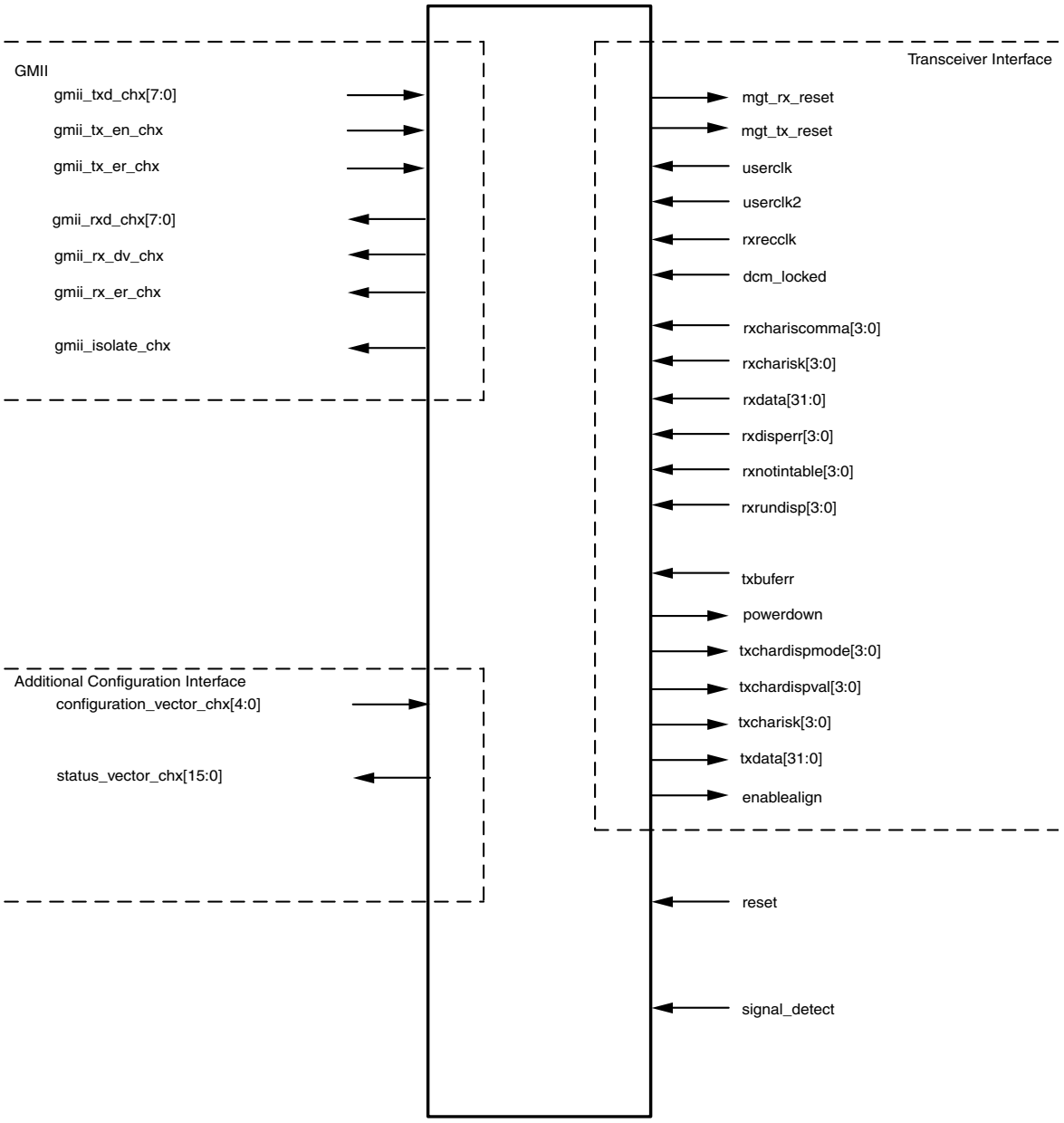


Figure 2-4: **Component Pinout for QSGMII Core *without* Optional MDIO or Optional Auto-Negotiation**

QSGMII Core 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. MII interface, if present on the QSGMII block, is converted to a GMII type interface. The interfaces end in “chx”, taking the values ch0 to ch3, indicating the port connection to the respective GMII interface.

GMII Pinout

Table 2-5 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 block level in PHY mode of operation for seamless connection to pads in IP Canvas delivered with the core connects these signals to IOBs to provide a place-and-route example.

Table 2-5: **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](1)	Output	GMII Received data to MAC
gmii_rx_dv_chx(1)	Output	GMII Received control signal to MAC
gmii_rx_er_chx(1)	Output	GMII Received control signal to MAC
gmii_isolate_chx(1)	Output	IOB 3-state control for GMII Isolation. Only of use when implementing an external GMII.

1. These signals are synchronous to the internal 125 MHz reference clock of the core. This is userclk2.

Common Signals

Table 2-6 describes the remaining signals common to all parameterizations of the core.

Table 2-6: **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 2-7 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 2-7: **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 2-8 describes the signals present when the optional Auto-Negotiation functionality is present.

Table 2-8: **Optional Auto-Negotiation Interface Signal Pinout**

Signal	Direction	Description
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 2-9: **Additional Configuration Interface Signal Pinout**

Signal	Direction	Description
configuration_vector_chx[5:0][(1)]	Input	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 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 Vivado IP catalog. 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.
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

Table 2-9: **Additional Configuration Interface Signal Pinout *(Cont'd)***

Signal	Direction	Description
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.
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 2-9: **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 2-9: **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.

QSGMII Core Physical Side Interface

Table 2-10 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 2-10: **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 `gtrxreset` signal of the device-specific transceiver. This reset is a combination of hard reset, soft reset and reset due to `rxbuffer` errors.
mgt_tx_reset[2]	Output	Reset signal issued by the core to the device-specific transceiver transmitter path. Connects to the `gttxreset` signal of the device-specific transceiver. This reset is a combination of hard reset, soft reset and reset due to `txbuffer` errors.
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.

Table 2-10: **Transceiver Interface Pinout *(Cont'd)***

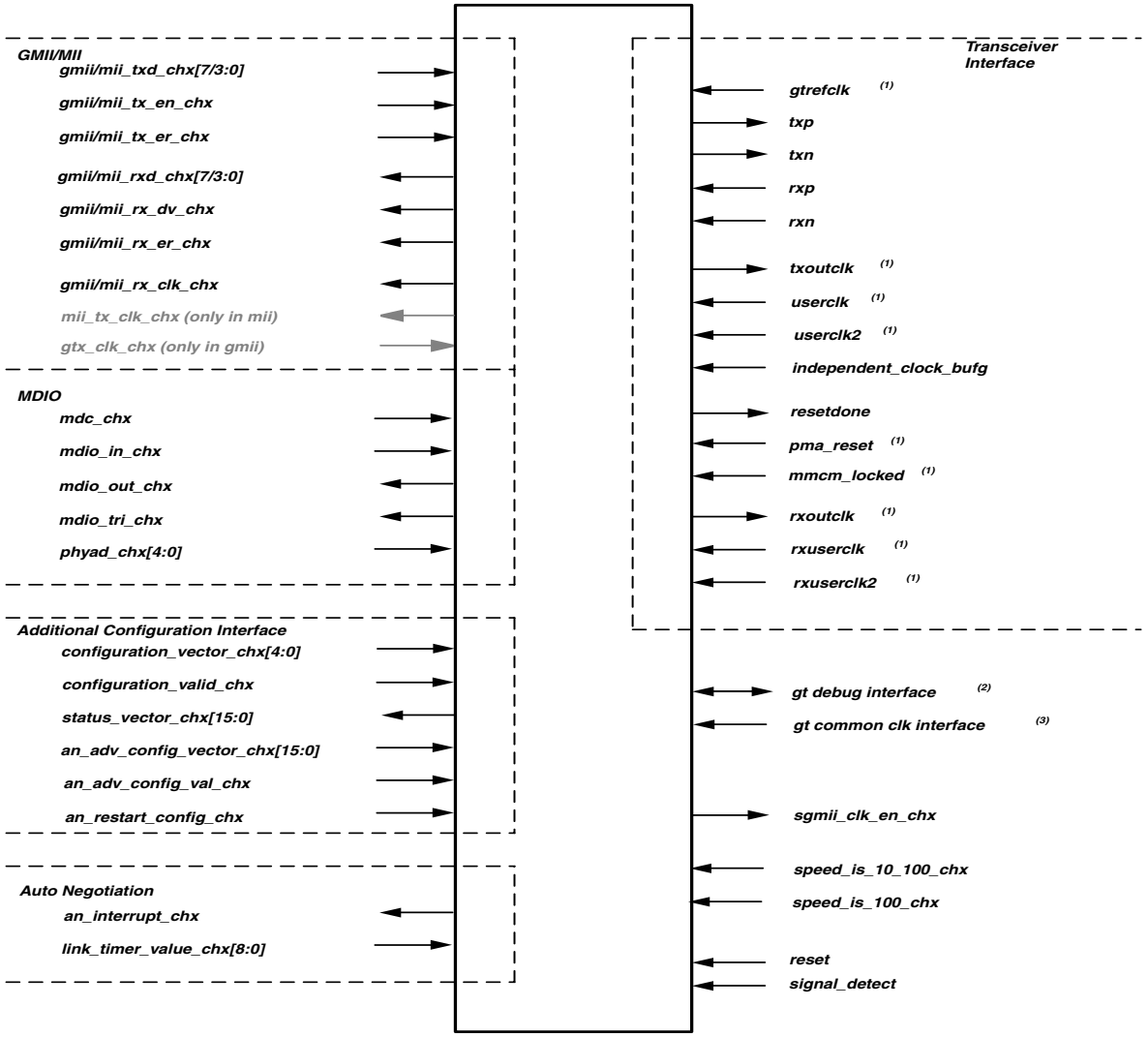
Signal	Direction	Description
rxrundisp[3:0](1)	Input	Connects to device-specific transceiver signal of the same name.
txbuferr(2)	Input	Connects to device-specific transceiver signal of the sam e name.
powerdown(2)	Output	Connects to device-specific transceiver signal of the same name.
txchardispmode[3:0](2)	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.

Block Hierarchy Level Ports

All the ports described here indicate the pins at the block level. The block level design instantiates the core and transceiver. The block level design is expected to be pulled from the IP Catalog into the IP Canvas.

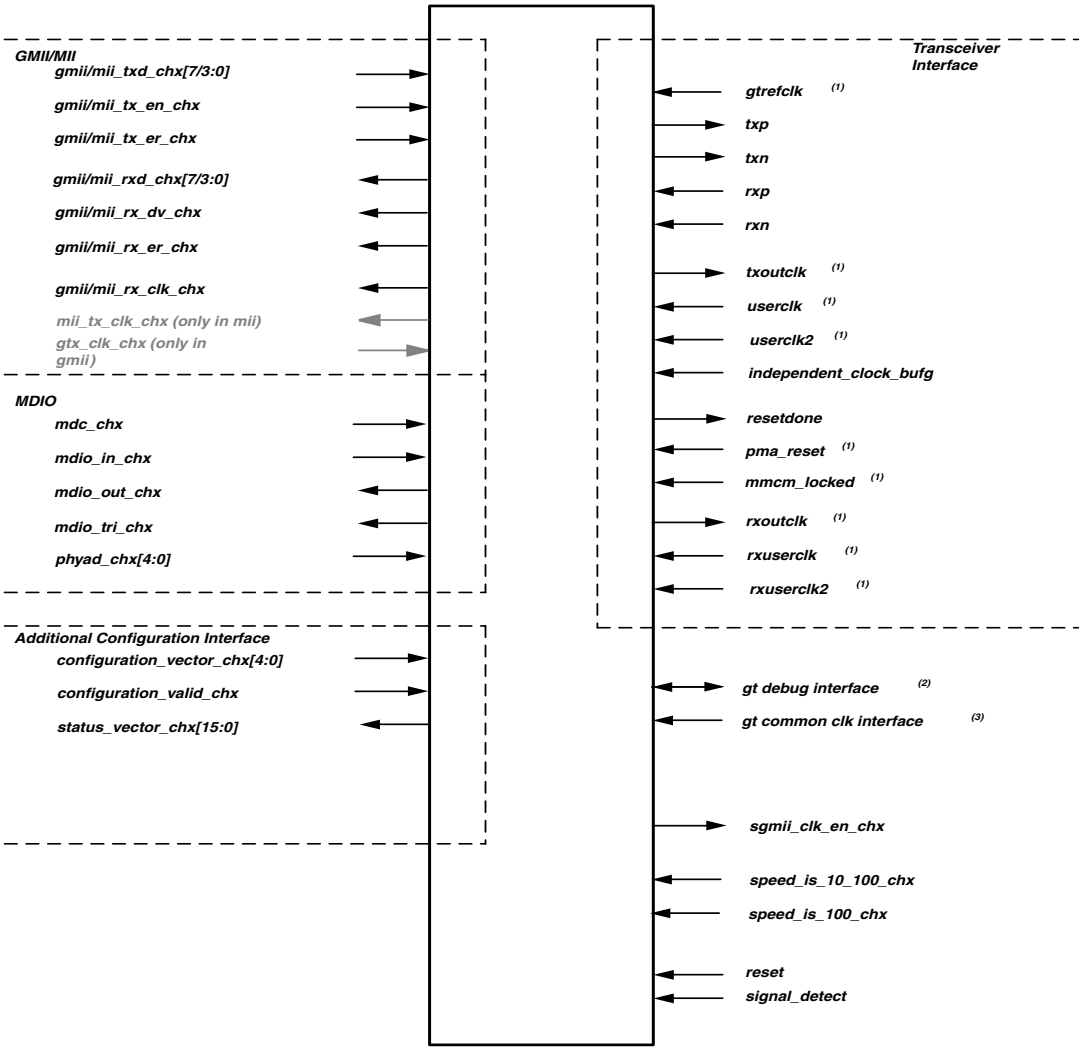
Figure 2-5 shows the pinout for the QSGMII block with the optional MDIO Management and optional Auto-Negotiation. The port name for multiple instances of an interface is generalized as "CHx". "CHx" takes the value "CH0", "CH1", "CH2", and "CH3".



1. These ports are available when **Include Shared Logic in Example Design** is selected. (Table 2-12). For ports available for **Include Shared Logic in Core** see to Table 2-13.
2. See Table 2-14 (Transceiver Control and Status Ports) .
3. See GT COMMON CLOCK INTERFACE section in Table 2-12 for **Include Shared Logic in Example Design** option. For **Include Shared Logic in Core** see Table 2-13.

Figure 2-5: **Component Pinout of QSGMII Block with Optional MDIO and Auto-Negotiation**

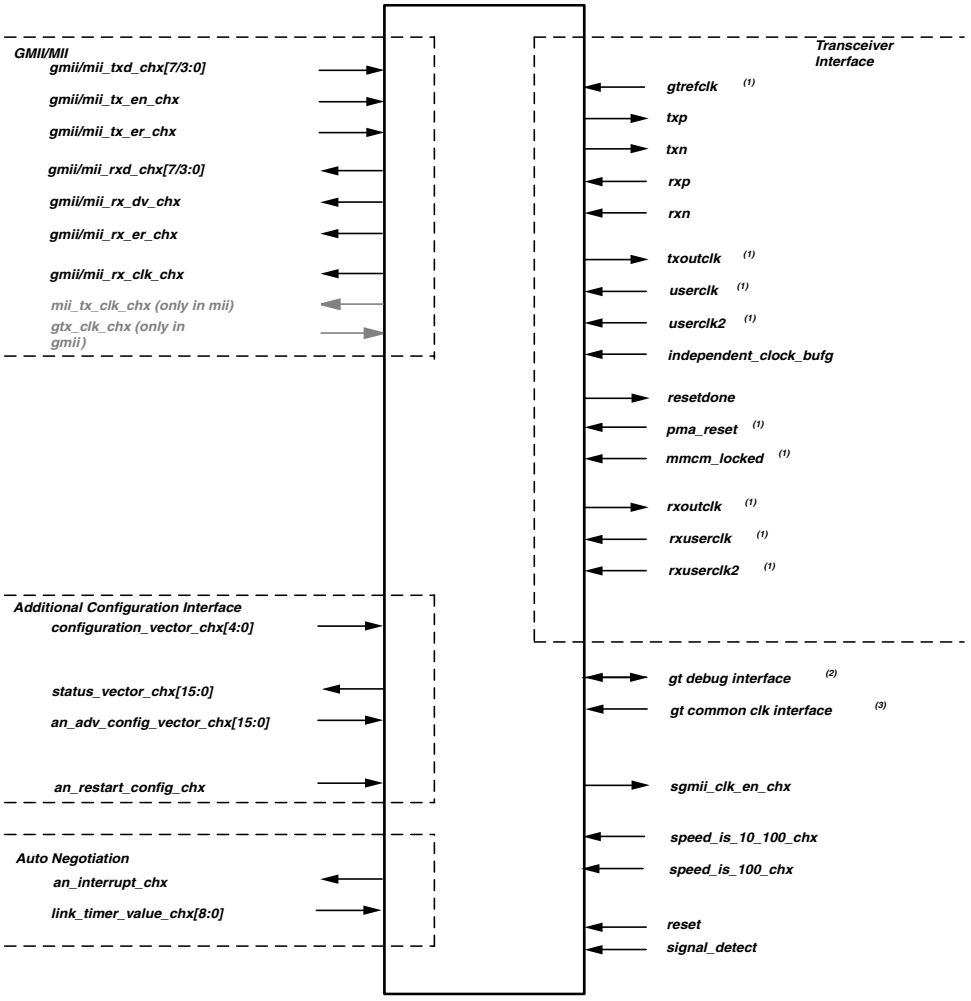
Figure 2-6 shows the pinout for the QSGMII block with only optional MDIO Management. The port name for multiple instances of an interface is generalized as "CHx". "CHx" takes the value "CH0", "CH1", "CH2", and "CH3".



1. These ports are available when **Include Shared Logic in Example Design** is selected. (Table 2-12). For ports available for **Include Shared Logic in Core** see to Table 2-13.
2. See Table 2-14 (Transceiver Control and Status Ports) .
3. See GT COMMON CLOCK INTERFACE section in Table 2-12 for **Include Shared Logic in Example Design** option. For **Include Shared Logic in Core** see Table 2-13.

Figure 2-6: **Component Pinout of QSGMII Block with only Optional MDIO Management**

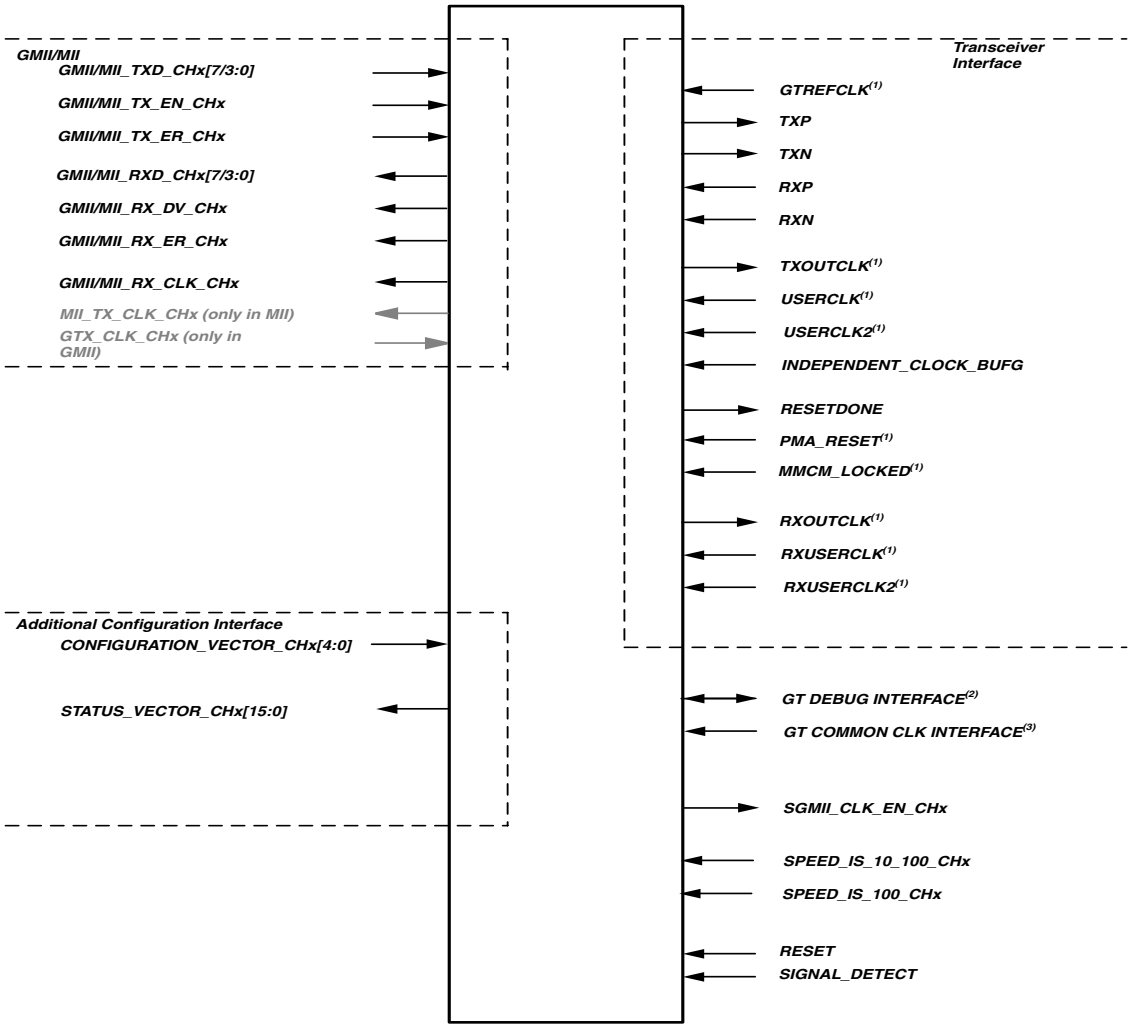
Figure 2-7 shows the pinout for the QSGMII block with only optional Auto-Negotiation. The port name for multiple instances of an interface is generalized as "CHx". "CHx" takes the value "CH0", "CH1", "CH2", and "CH3".



1. These ports are available when **Include Shared Logic in Example Design** is selected. (Table 2-12). For ports available for **Include Shared Logic in Core** see to Table 2-13.
2. See Table 2-14 (Transceiver Control and Status Ports) .
3. See GT COMMON CLOCK INTERFACE section in Table 2-12 for **Include Shared Logic in Example Design** option. For **Include Shared Logic in Core** see Table 2-13.

Figure 2-7: **Component Pinout of QSGMII Block with only Optional Auto-Negotiation**

Figure 2-8 shows the pinout for the QSGMII block without optional MDIO or Auto-Negotiation. The port name for multiple instances of an interface is generalized as "CHx". "CHx" takes the value "CH0", "CH1", "CH2", and "CH3".



1. These ports are available when **Include Shared Logic in Example Design** is selected. (Table 2-12). For ports available for **Include Shared Logic in Core** see to Table 2-13.
2. See Table 2-14 (Transceiver Control and Status Ports) .
3. See GT COMMON CLOCK INTERFACE section in Table 2-12 for **Include Shared Logic in Example Design** option. For **Include Shared Logic in Core** see Table 2-13.

Figure 2-8: **Component Pinout of QSGMII Block without Optional MDIO or Auto-Negotiation**

QSGMII Block Client Side Interface

This interface contains four groups of interfaces, with each group containing a set of the GMII/MII 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

This interface is enabled in the MAC mode or GMII configuration of PHY mode. In the MAC mode this interface is expected to be connected to the GMII interface of Tri Mode Ethernet MAC core (TEMAC). In the GMII configuration of PHY mode, this interface is brought out onto the pads.

Table 2-11: **GMII Interface Signals Pinout**

Signal	Direction	Description
gmii_txd_chx[7:0](1)	Input	GMII Transmit data
gmii_txd_en_chx(1)	Input	GMII Transmit data enable
gmii_tx_er_chx(1)	Input	GMII Transmit error
gmii_rxd_chx[7:0] (2)	Output	GMII Receive data
gmii_rxd_dv_chx(2)	Output	GMII Receive data valid
gmii_rx_er_chx(2)	Output	GMII Receive error
gtx_clk_chx	Input	GMII TX clock. This is valid only in GMII configuration in PHY mode
gmii_rx_clk_chx	Output	GMII RX clock. This is valid only in GMII configuration in PHY mode
sgmii_clk_en_chx	Output	Clock enables. This valid only in MAC mode.

1. In MAC mode these signals are synchronous to 125 MHz reference clock of the core, that is, `userclk2`. In GMII configuration of PHY mode these signals are synchronous to `gtx_clk_chx` and these signals are synchronized to `userclk2` domain using Transmit Elastic Buffer present in the block
2. These signals are synchronous to 125 MHz reference clock of the core. This is `userclk2`.

Table 2-12: **Signal Direction Description**

Signal	Direction	Description
mii_txd_chx[7:0](1)	Input	MII Transmit data
mii_tx_en_chx(1)	Input	MII Transmit data enable
mii_tx_er_chx(1)	Input	MII Transmit error
mii_rxd_chx[7:0](2)	Output	MII Receive data
mii_rx_dv_chx(2)	Output	MII Receive data valid
mii_rx_er_chx(2)	Output	MII Receive data error
mii_tx_clk_chx	Output	MII TX clock. This is valid only in MII configuration in MII mode. The clock can be 2.5/25 MHz based on 10/100 Mbps mode of operation
mii_rx_clk_chx	Output	MII RX clock. This is valid only in MII configuration in MII mode. The clock can be 2.5/25 MHz based on 10/100 Mbps mode of operation

1. These signals should be driven on `mii_tx_clk_chx`.
2. These signals are synchronous to `mii_rx_clk_chx`.

Common Signals

See Table 2-6 of Common Signals section for these signals.

MDIO Management Interface Pinout (Optional)

See Table 2-7 of MDIO Management Interface Pinout section for these signals.

Auto-Negotiation Interface Pinout (Optional)

See Table 2-8 of Auto-Negotiation Interface Pinout section for these signals.

Additional Configuration Interface

See Table 2-9 of Additional Configuration Interface Pinout section for these signals.

QSGMII Block Physical Side Interface

Table 2-13 describes the interface to the device-specific transceiver for the case where shared logic is included in the example design.

Table 2-13: **QSGMII Block Physical Side Interface with Shared Logic in the Example Design**

Signal	Direction	Description
gtrefclk	Input	125 MHz reference clock from IBUFDS to the transceiver
txp	Output	Transmit differential
txn	Output	Transmit differential
rxp	Input	Receive differential
rxn	Input	Receive differential
txoutclk	Output	txoutclk from 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.
rxoutclk	Output	`rxoutclk` from transceiver.
rxuserclk	Input	Also connected to `rxusrclk` of the device-specific transceiver. Clock domain is not applicable.
rxuserclk2	Input	Also connected to `rxusrclk2` of the device-specific transceiver. Clock domain is not applicable.

Table 2-13: **QSGMII Block Physical Side Interface with Shared Logic in the Example Design**

Signal	Direction	Description
independent_clock_bufg	Input	200 MHz stable clock used as stable clock in transceiver and also as control clock for IDELAYCTRL.
resetdone	Output	Indication that reset sequence of the transceiver is complete.
pma_reset	Input	Hard reset synchronized to `independent_clock_bufg`.
mmcm_locked	Input	Indication from the MMCM that the outputs are stable.
GT COMMON CLOCK INTERFACE		
gt0_pll0outclk_in	Input	Valid only for Artix-7 families. Indicates out clock from PLL0 of GT Common.
gt0_pll0outrefclk_in	Input	Valid only for Artix-7 families. Indicates reference out clock from PLL0 of GT Common.
gt0_pll1outclk_in	Input	Valid only for Artix-7 families. Indicates out clock from PLL1 of GT Common.
gt0_pll1outrefclk_in	Input	Valid only for Artix-7 families. Indicates reference out clock from PLL1 of GT Common.
gt0_pll0lock_in	Input	Valid only for Artix-7 families. Indicates out PLL0 of GT Common has locked.
gt0_pll0refclklost_in	Input	Valid only for Artix-7 families. Indicates out reference clock for PLL0 of GT Common is lost.
gt0_pll0reset_out	Output	Valid only for Artix-7 families. Reset for PLL of GT Common from reset fsm in GT Wizard.
gt0_qplloutclk_in	Input	Valid only for non Artix-7 families. Indicates out clock from PLL of GT Common.
gt0_qplloutrefclk_in	Input	Valid only for non Artix-7 families. Indicates reference out clock from PLL of GT Common.

Table 2-14 describes the interface to the device-specific transceiver when Shared Logic is in the Core.

Table 2-14: **QSGMII Block Physical Side Interface with Shared Logic in the Core**

Signal	Direction	Description
gtrefclk_p	Input	125 MHz differential reference clock to IBUFDS
gtrefclk_p	Input	125 MHz differential reference clock to IBUFDS
gtrefclk_out	Output	125 MHz reference clock from IBUFDS
txp	Output	Transmit differential
txn	Output	Transmit differential
rxp	Input	Receive differential
rxn	Input	Receive differential
userclk_out	Output	Also connected to txusrclk of the device-specific transceiver. Clock domain is not applicable.
userclk2_out	Output	Also connected to txusrclk2 of the device-specific transceiver. Clock domain is not applicable.
rxuserclk_out	Output	Also connected to rxusrclk of the device-specific transceiver. Clock domain is not applicable.
rxuserclk2_out	Output	Also connected to rxusrclk2 of the device-specific transceiver. Clock domain is not applicable.
independent_clock_bufg	Input	200 MHz stable clock used as stable clock in transceiver and also as control clock for IDELAYCTRL.
resetdone	Output	Indication that reset sequence of the transceiver is complete.
pma_reset_out	Output	Hard reset synchronized to `independent_clock_bufg`.
mmcm_locked_out	Output	Indication from the MMCM that the outputs are stable.
GT COMMON CLOCK INTERFACE		
gt0_pll0outclk_out	Output	Valid only for Artix-7 families. Indicates out clock from PLL0 of GT Common.
gt0_pll0outrefclk_out	Output	Valid only for Artix-7 families. Indicates reference out clock from PLL0 of GT Common.

Table 2-14: **QSGMII Block Physical Side Interface with Shared Logic in the Core *(Cont'd)***

Signal	Direction	Description
gt0_pll1outclk_out	Output	Valid only for Artix-7 families. Indicates out clock from PLL1 of GT Common.
gt0_pll1outrefclk_out	Output	Valid only for Artix-7 families. Indicates reference out clock from PLL1 of GT Common.
gt0_pll0lock_out	Output	Valid only for Artix-7 families. Indicates out PLL0 of GT Common has locked.
gt0_pll0refclklost_out	Output	Valid only for Artix-7 families. Indicates out reference clock for PLL0 of GT Common is lost.
gt0_qplloutclk_out	Output	Valid only for non Artix-7 families. Indicates out clock from PLL of GT Common.
gt0_qplloutrefclk_out	Output	Valid only for non Artix-7 families. Indicates reference out clock from PLL of GT Common.

Transceiver Control and Status Ports

This section describes optional ports that, if enabled, allow the monitoring and control of certain important transceiver ports. When not selected, these ports are tied to their default values.

Note: The Dynamic Reconfiguration Port is only available if this option is selected. Also for UltraScale devices the prefix of ports in Table 2-15 are changed from "gt0" to "gt" and the postfix "_in" and "_out" are dropped.

Table 2-15: **Transceiver Control and Status Ports**

Signal	Direction	Description
gt0_drp_addr_in[8:0]	Input	DRP address bus `drp_en` IN DRP enable signal. 0: No read or write operation performed. 1: enables a read or write operation.
gt0_drpi_in[15:0]	Input	Data bus for writing configuration data to the transceiver.
gt0_drpo_out[15:0]	Output	Data bus for reading configuration data from the transceiver.

Table 2-15: **Transceiver Control and Status Ports *(Cont'd)***

Signal	Direction	Description
gt0_drprdy_out	Output	Indicates operation is complete for write operations and data is valid for read operations.
gt0_drp_busy_out	Output	Output valid only for Artix-7 family. Indicates that DRP interface is busy. This bit should be checked before any transaction is posted on DRP interface.
gt0_drpwe_in	Input	DRP write enable. 0: Read operation when DRPEN is 1. 1: Write operation when DRPEN is 1.
gt0_drpclk_in	Input	DRP Clock
gt0_rxchariscomma_out[3:0]	Output	GT Status
gt0_rxcharisk_out[3:0]	Output	
gt0_rxbyteisaligned_out	Output	
gt0_rxbyterealign_out	Output	
gt0_rxcommadet_out	Output	
gt0_txdiffctrl_in[3:0]	Input	GT TX Driver
gt0_txpostcursor_in[4:0]	Input	
gt0_txprecursor_in[4:0]	Input	
gt0_txpolarity_in	Input	GT Polarity
gt0_rxpolarity_in	Input	
gt0_txprbssel_in[2:0]	Input	GT PRBS
gt0_txprbsforceerr_in	Input	
gt0_rxprbscntreset_in	Input	
gt0_rxprbserr_out	Output	
gt0_rxprbssel_in[2:0]	Input	
gt0_loopback_in[2:0]	Input	GT Loopback
gt0_txresetdone_out	Output	GT Status
gt0_rxresetdone_out	Output	
gt0_rxdisperr_out[3:0]	Output	
gt0_rxnotintable_out[3:0]	Output	
gt0_eyescanreset_in[3:0]	Input	GT Eye Scan
gt0_eyescandataerror_out	Output	
gt0_eyescantrigger_in	Input	
gt0_rxrate_in[2:0]	Input	
gt0_rxcdrhold_in	Input	GT CDR
gt0_rxcdrlock_out	Output	

Table 2-15: **Transceiver Control and Status Ports *(Cont'd)***

Signal	Direction	Description
gt0_rxratedone_out	Output	GT Fabric Clock Output Control
gt0_rxlpmhfhold_in	Input	GT GTP Low-Power Mode (LPM)
gt0_rxlpmlfhold_in	Input	
gt0_rxlpmhfovrden_in	Input	
gt0_rxlpmreset_in	Input	
gt0_rxlpmen_in	Input	GT GTX/GTH RX Decision Feedback Equalizer (DFE)
gt0_rxdfelpmreset_in	Input	
gt0_rxdfeagcovrden_in	Input	
gt0_rxmonitorout_out[6:0]	Output	
gt0_rxmonitorsel_in[1:0]	Input	
gt0_dmonitorout_out[16:0]	Output	
gt0_gttxreset_in	Input	Tx Reset (`gt0_gttxreset_in` present in only non UltraScale devices)
gt0_txpcsreset_in	Input	
gt0_txpmareset_in	Input	
gt0_gtrxreset_in	Input	Rx Reset (`gt0_gtrxreset_in` present in only non UltraScale devices `gt0_rxpmaresetdone_out` is tied to '1' for devices supporting GTX transceivers)
gt0_rxpcsreset_in	Input	
gt0_rxpmareset_in	Input	
gt0_rxpmaresetdone_out	Output	
gt0_cplllock_out	Output	Channel PLL locked. Present only in non GTP transceiver devices
gt0_txbufstatus_out[1:0]	Output	Transmitter buffer status

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 2-9 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).

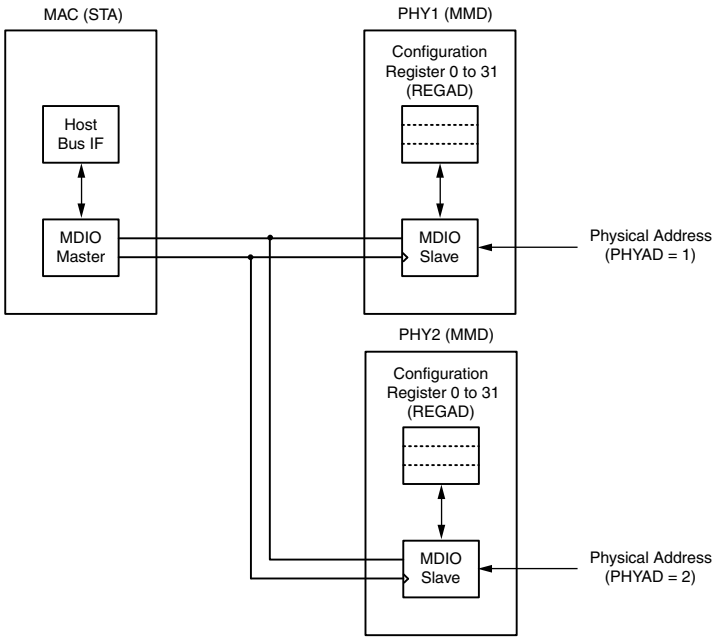


Figure 2-9: **Typical MDIO Managed System**

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 2-16.

Table 2-16: **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 2-10 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.

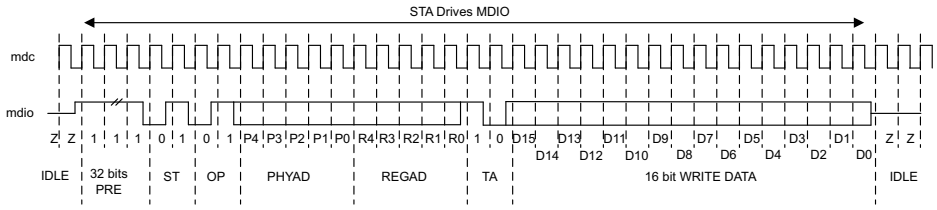


Figure 2-10: **MDIO Write Transaction**

Read Transaction

Figure 2-11 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.

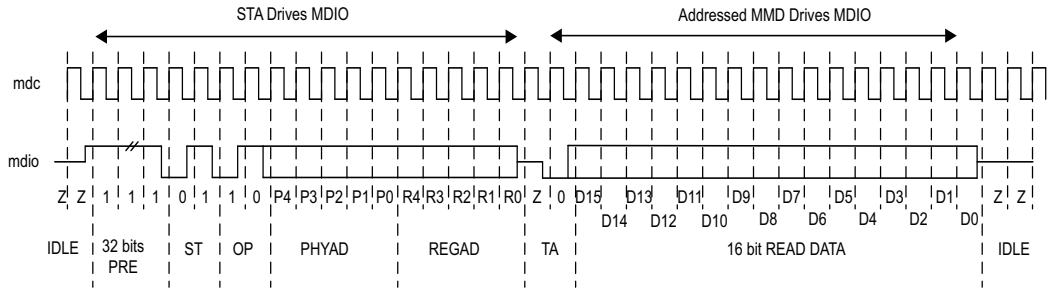


Figure 2-11: **MDIO Read Transaction**

MDIO Addressing

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

Physical Address (PHYAD)

As shown in Figure 2-9, 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 2-12 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.

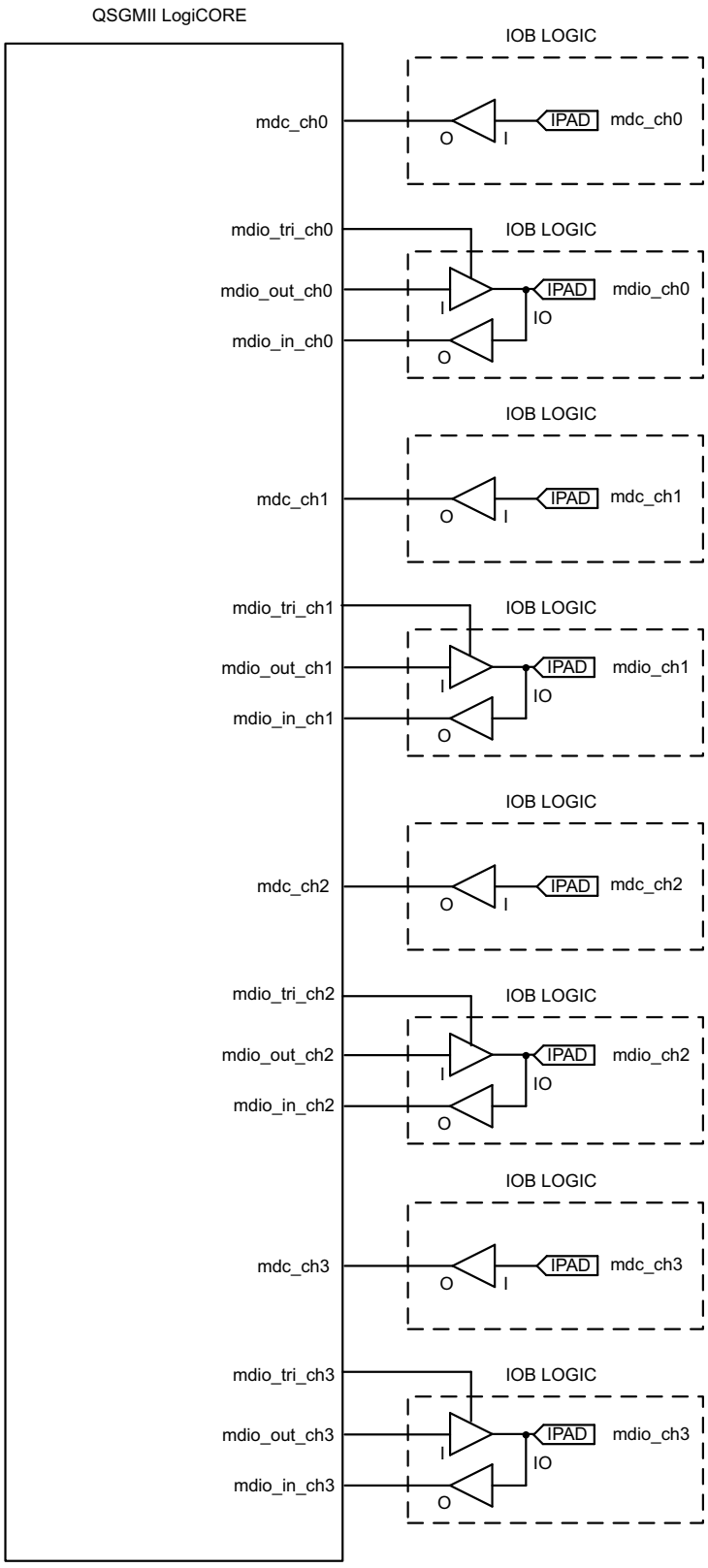


Figure 2-12: **Creating an External MDIO Interface**

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 Vivado design 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 registers described in Table 2-17 through Table 2-30. 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 2-17: **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

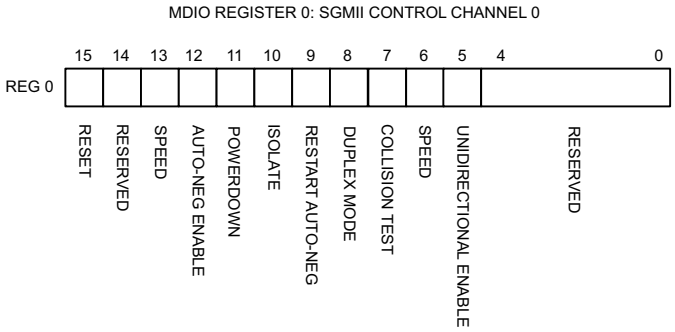


Figure 2-13: **MDIO Register 0: SGMII Control Register Channel/Module 0**

Table 2-18: **SGMII Control Register Channel/Module 0 (Register 0)**

Bits	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 2-18: **SGMII Control Register Channel/Module 0 (Register 0) *(Cont'd)***

Bits	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

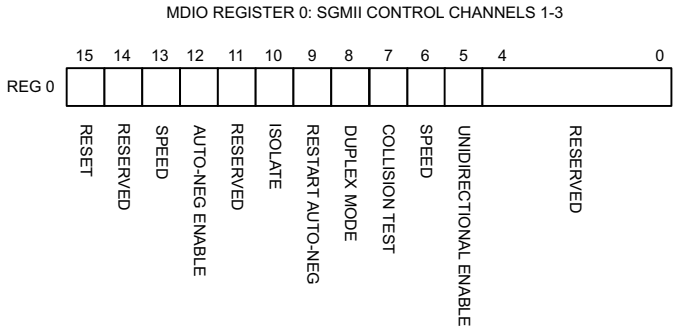


Figure 2-14: **MDIO Register 0: SGMII Control Channels/Modules 1-3**

Table 2-19: **SGMII Control Register Channels/Modules 1-3 (Register 0)**

Bits	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

Table 2-19: **SGMII Control Register Channels/Modules 1-3 (Register 0)**

Bits	Name	Description	Attributes	Default Value
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

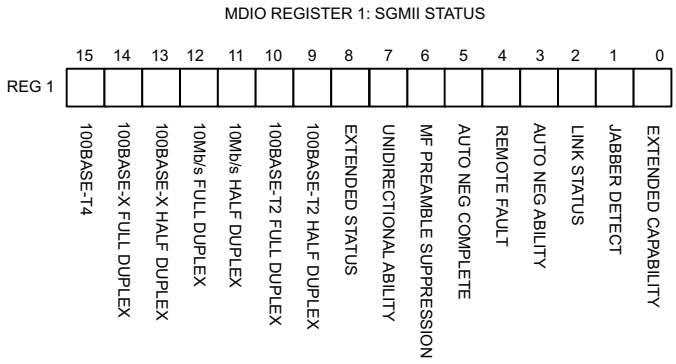


Figure 2-15: **MDIO Register 1: SGMII Status Register**

Table 2-20: **SGMII Status Register (Register 1)**

Bits	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
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

Table 2-20: **SGMII Status Register (Register 1) *(Cont'd)***

Bits	Name	Description	Attributes	Default Value
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)

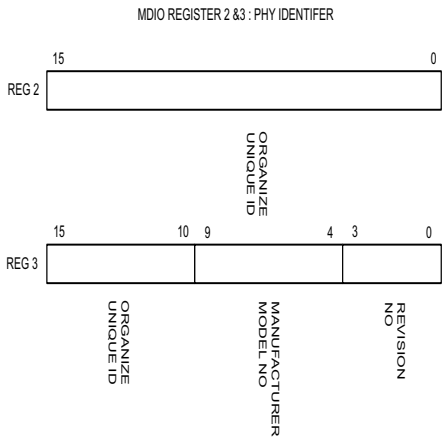


Figure 2-16: **MDIO Registers 2 and 3: (PHY IDENTIFIER)**

Table 2-21: **PHY Identifier (Registers 2 and 3)**

Bits	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

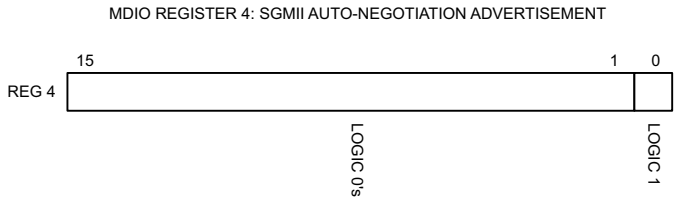


Figure 2-17: **MDIO Register 4: SGMII Auto-Negotiation Advertisement**

Table 2-22: **SGMII Auto-Negotiation Advertisement (Register 4)**

Bits	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

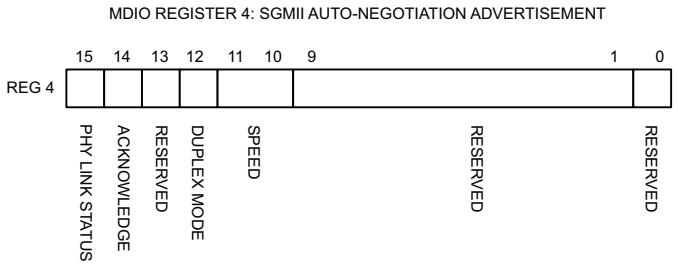


Figure 2-18: **MDIO Register 4: SGMII Auto-Negotiation Advertisement**

Table 2-23: **SGMII Auto-Negotiation Advertisement in PHY Mode (Register 4)**

Bits	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 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

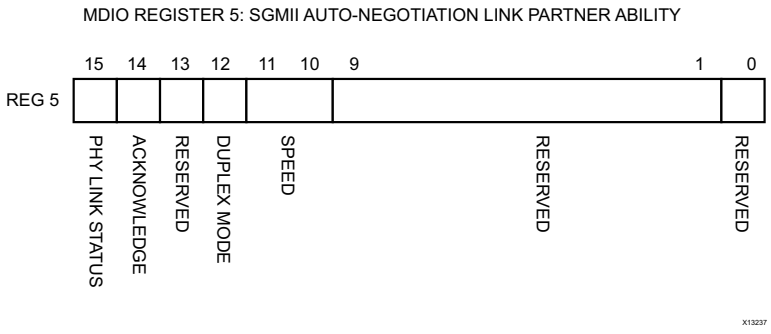


Figure 2-19: **MDIO Register 5: SGMII Auto-Negotiation Link Partner Ability**

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 2-24: **SGMII Auto-Negotiation Link Partner Ability Base (Register 5)**

Bits	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 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

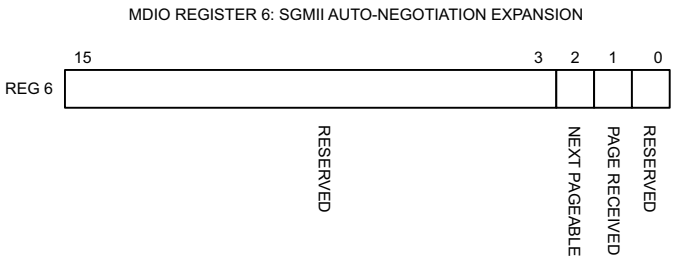


Figure 2-20: **MDIO Register 6: SGMII Auto-Negotiation Expansion**

Table 2-25: **SGMII Auto-Negotiation Expansion (Register 6)**

Bits	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

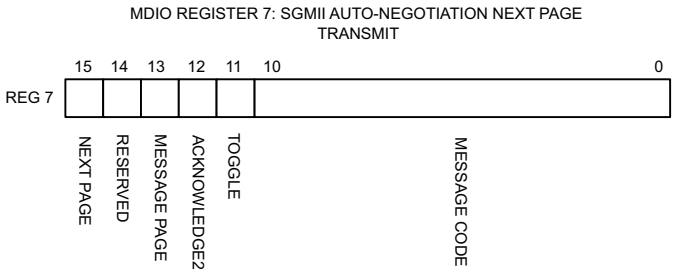


Figure 2-21: **MDIO Register 7: SGMII Auto-Negotiation Next Page Transmit**

Table 2-26: **SGMII Auto-Negotiation Next Page Transmit (Register 7)**

Bits	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

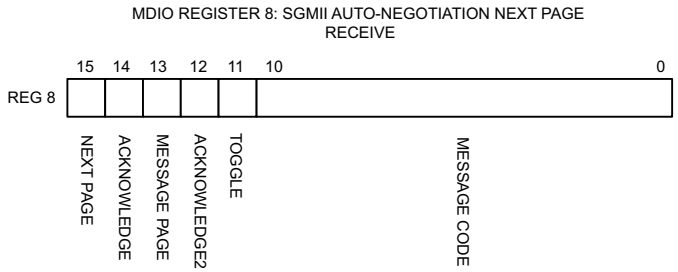


Figure 2-22: **MDIO Register 8: SGMII Auto-Negotiation Next Page Receive**

Table 2-27: **SGMII Auto-Negotiation Next Page Receive (Register 8)**

Bits	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 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

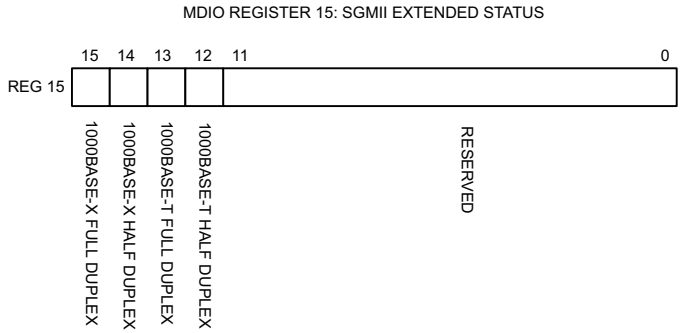


Figure 2-23: **MDIO Register 15: SGMII Extended Status**

Table 2-28: **SGMII Extended Status (Register 15)**

Bits	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

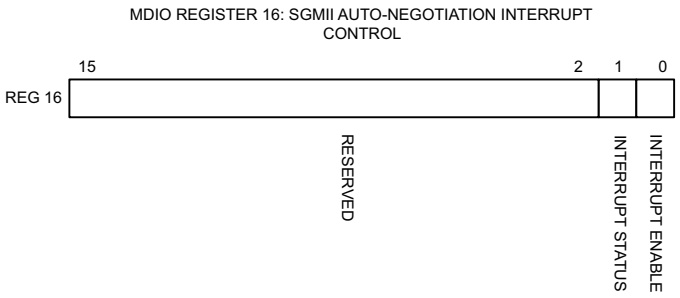


Figure 2-24: **MDIO Register 16: SGMII Auto-Negotiation Interrupt Control**

Table 2-29: **SGMII Auto-Negotiation Interrupt Control (Register 16)**

Bits	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

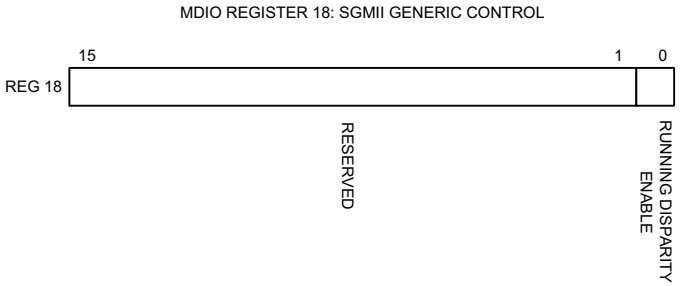


Figure 2-25: **MDIO Register 18: SGMII Generic Control**

Table 2-30: **SGMII Generic Control (Register 18)**

Bits	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 are described in this section.

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 2-31: **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

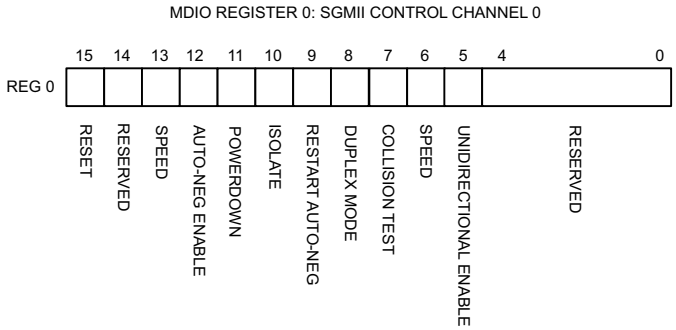


Figure 2-26: **MDIO Register 0: SGMII Control Register Channel/Module 0**

Table 2-32: **SGMII Control Register Channel/Module 0 (Register 0)**

Bits	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
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

Management Registers Channels/Modules 1-3

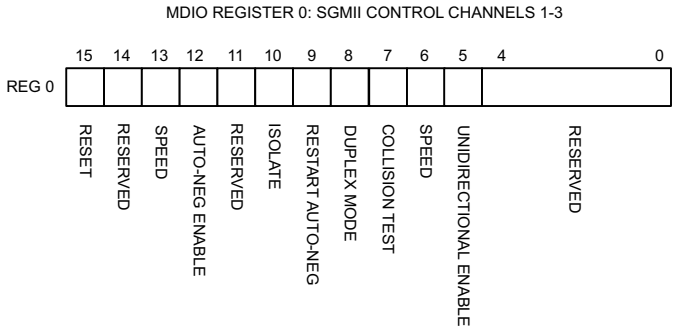


Figure 2-27: **MDIO Register 0: SGMII Control Channels/Modules 1-3**

Table 2-33: **SGMII Control Register Channels/Modules 1-3 (Register 0)**

Bits	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

Table 2-33: **SGMII Control Register Channels/Modules 1-3 (Register 0) *(Cont'd)***

Bits	Name	Description	Attributes	Default Value
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

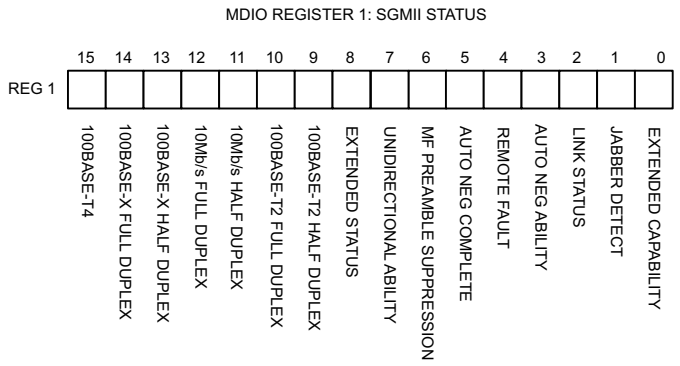


Figure 2-28: **MDIO Register 1: SGMII Status Register**

Table 2-34: **SGMII Status Register (Register 1)**

Bits	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

Table 2-34: **SGMII Status Register (Register 1) *(Cont'd)***

Bits	Name	Description	Attributes	Default Value
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)

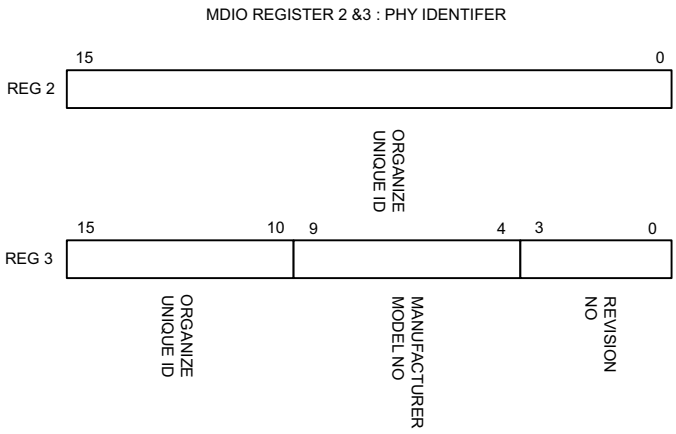


Figure 2-29: **MDIO Register 2 and 3: PHY Identifier**

Table 2-35: **PHY Identifier (Registers 2 and 3)**

Bits	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

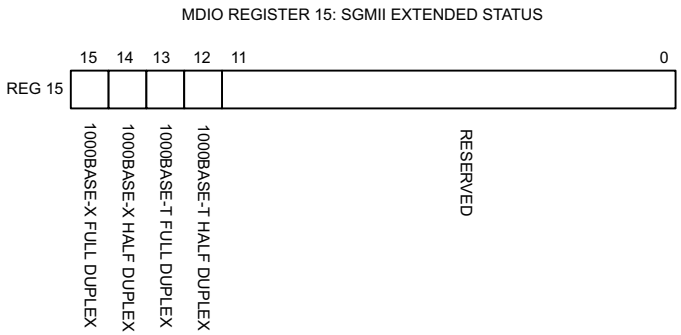


Figure 2-30: **MDIO Register 15: SGMII Extended Status**

Table 2-36: **SGMII Extended Status (Register 15)**

Bits	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)

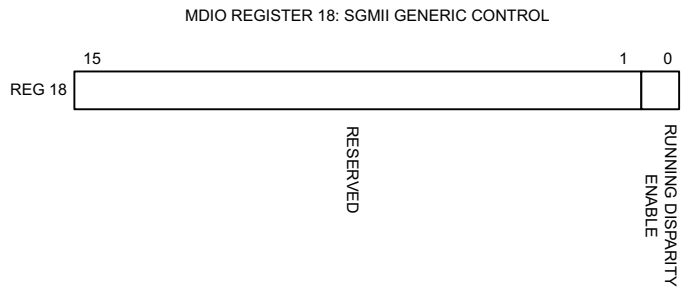


Figure 2-31: **MDIO Register 18: SGMII Generic Control**

Table 2-37: **SGMII Generic Control (Register 18)**

Bits	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, Product Specification 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 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 Vivado® design 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 10.

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
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 the Vivado Design Suite 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.

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 Vivado design tools when the core is generated. See the Vivado IP catalog — Chapter 6, Customizing and Generating the Core.

Shared Logic

Up to version 2 of the QSGMII core, the RTL hierarchy for the core was fixed. This resulted in some difficulty because shareable clocking and reset logic needed to be extracted from the core example design for use with a single instance, or multiple instances of the core.

Shared logic is a new feature that provides a more flexible architecture that works both as a standalone core and as a part of a larger design with one or more core instances. This minimizes the amount of HDL modifications required, but at the same time retains the flexibility to address more uses of the core.

The new level of hierarchy is called `<component_name>_support`. Figure 3-1 and Figure 3-2 show two hierarchies where the shared logic block is contained either in the core or in the example design. In these figures, `<component_name>` is the name of the generated core. The difference between the two hierarchies is the boundary of the core. It is controlled using the **Shared Logic** option in the Vivado IDE (see Figure 6-3).

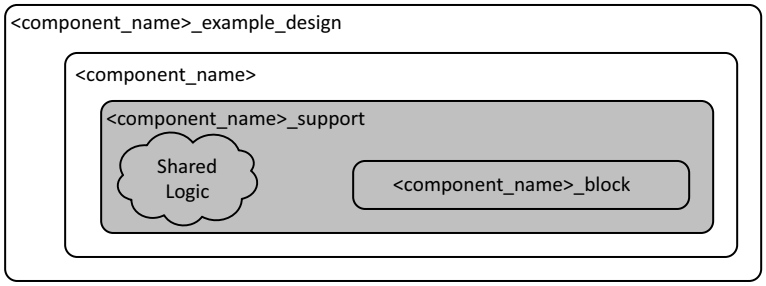


Figure 3-1: **Shared Logic Included in Core**

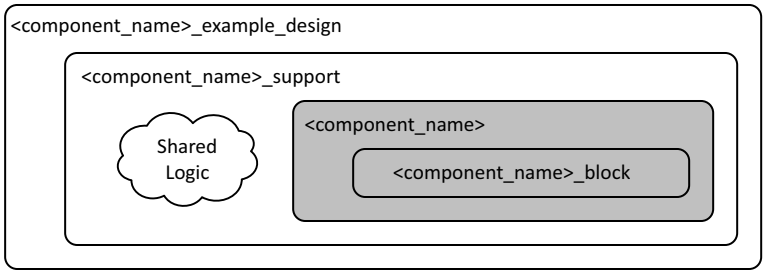


Figure 3-2: **Shared Logic Included in Example Design**

Clocking

- For clocking constraints see Chapter 7, Constraining the Core.
- 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, 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 3-3 shows the most common reset structure for the core.

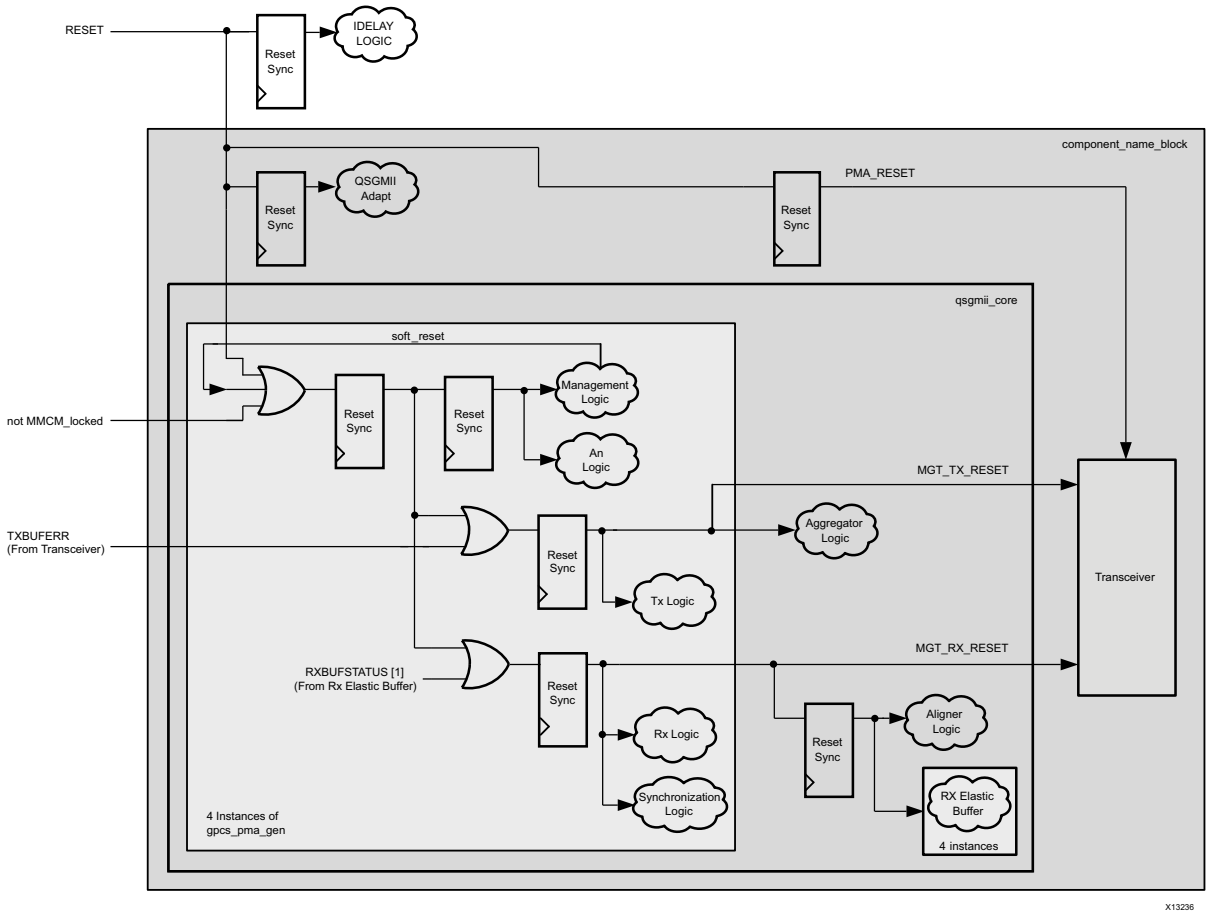


Figure 3-3: **Reset Structure**

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 Xilinx LogiCORE IP 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) [Ref 5].

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 Vivado® design 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 Encrypted Core Level 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.

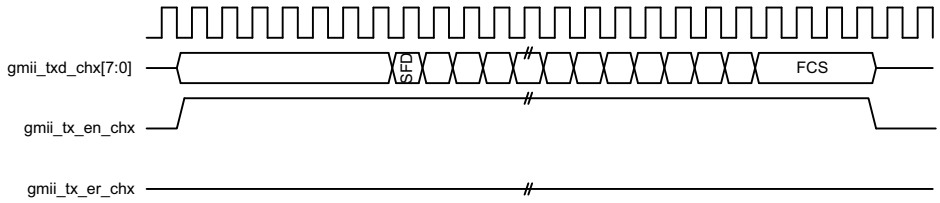


Figure 4-1: **GMII Normal Frame Transmission**

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.

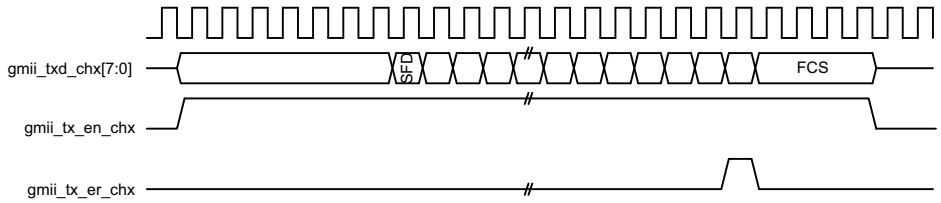


Figure 4-2: **GMII Error Propagation Within a Frame**

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.

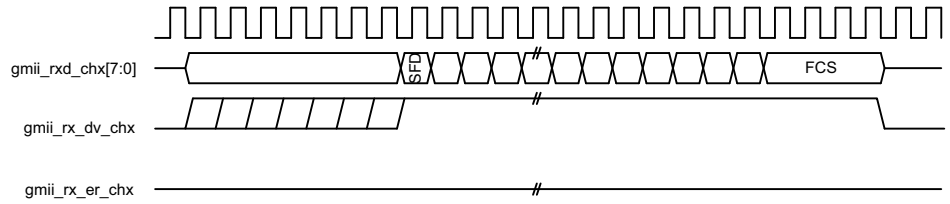


Figure 4-3: **GMII Normal Frame Reception**

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.

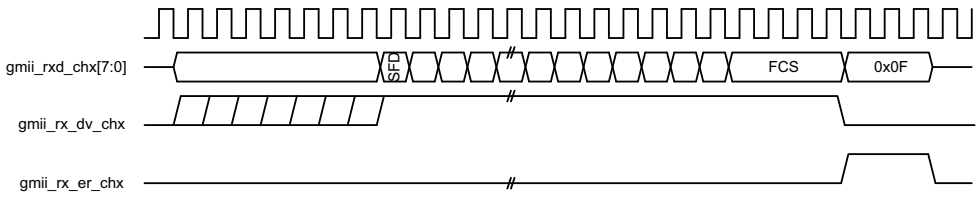


Figure 4-4: **GMII Normal Frame Reception WIth Carrier Extension**

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.

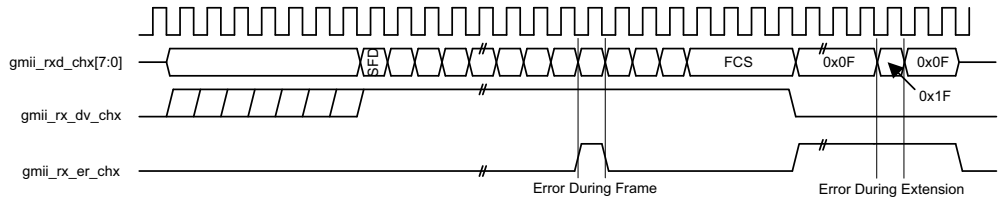


Figure 4-5: **GMII Frame Reception WIth Errors**

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.

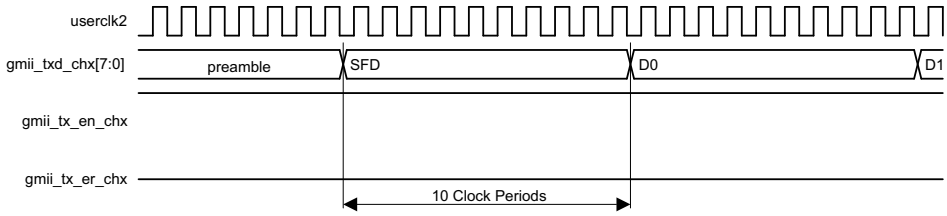


Figure 4-6: **100 Mb/s Frame Transmission**

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 QSGMII 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.

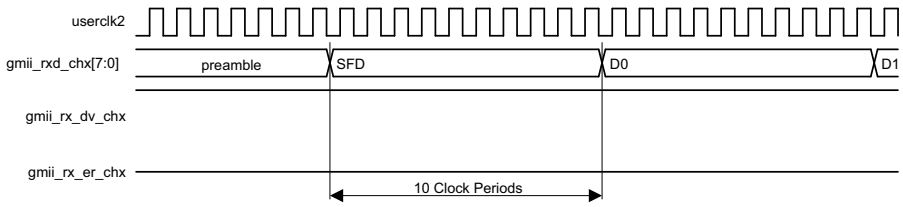


Figure 4-7: **100 Mb/s Frame Reception**

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

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.

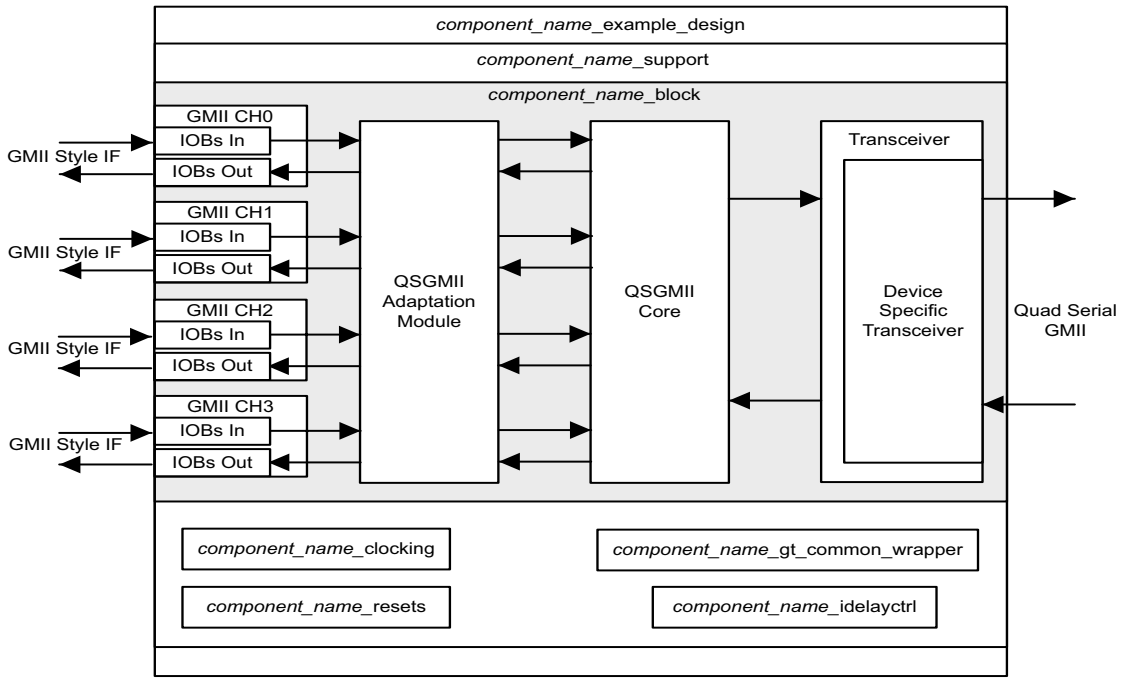


Figure 4-8: **Example Design HDL for QSGMII**

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 Encrypted Core Level 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.

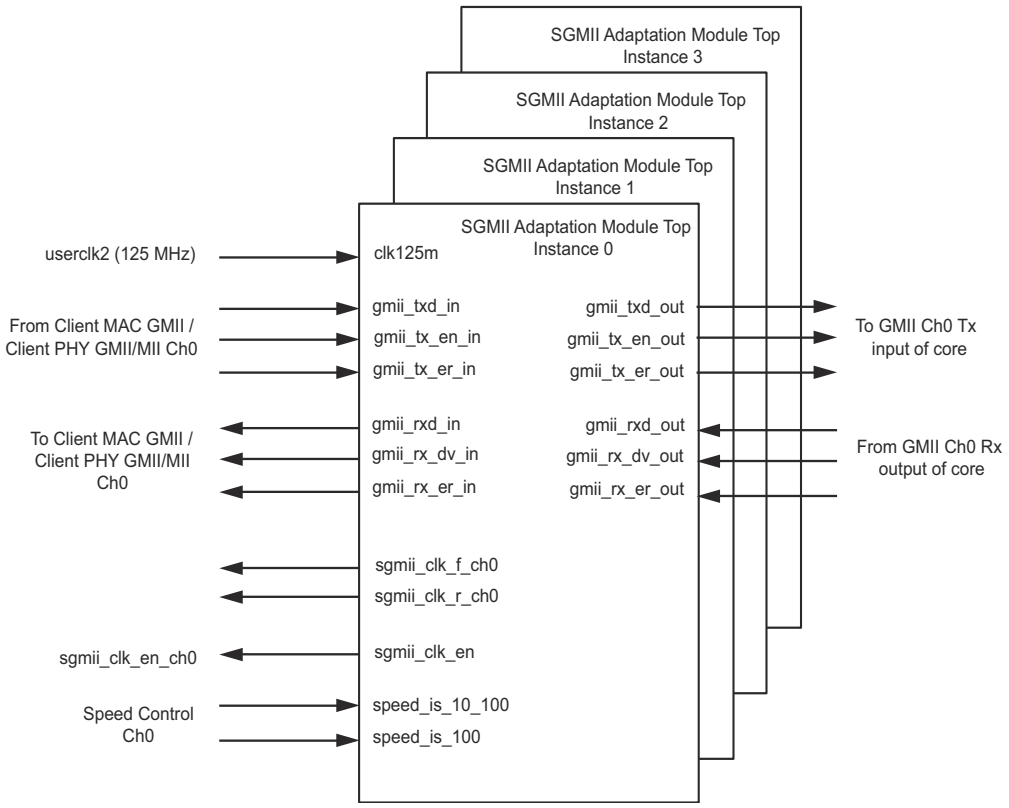


Figure 4-9: **QSGMII Adaptation Module**

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.

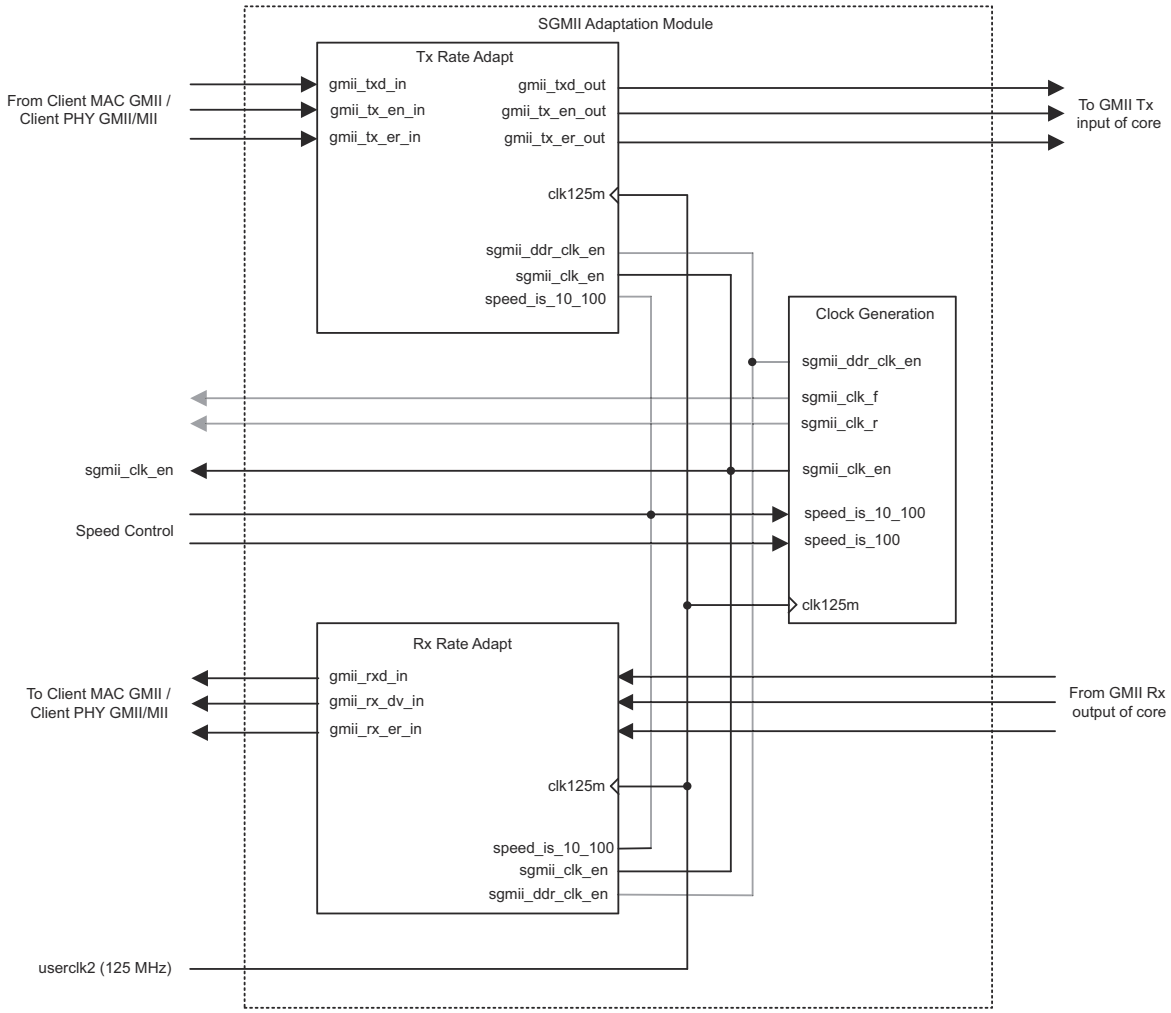


Figure 4-10: **SGMII Adaptation Module**

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.

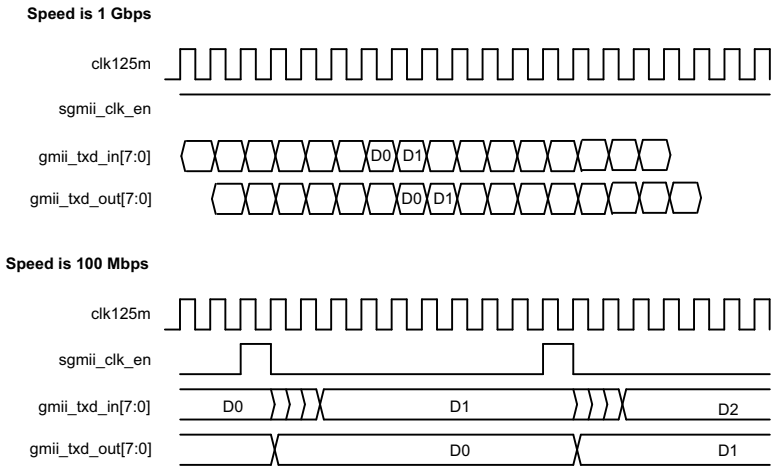


Figure 4-11: **Transmitter Rate Adaptation Module Sampling in MAC Mode**

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.

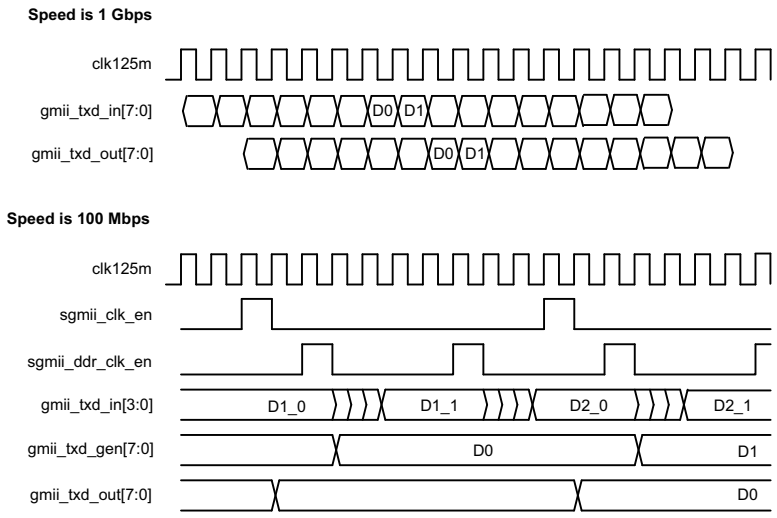


Figure 4-12: **Transmitter Rate Adaptation Module Sampling in PHY Mode**

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.

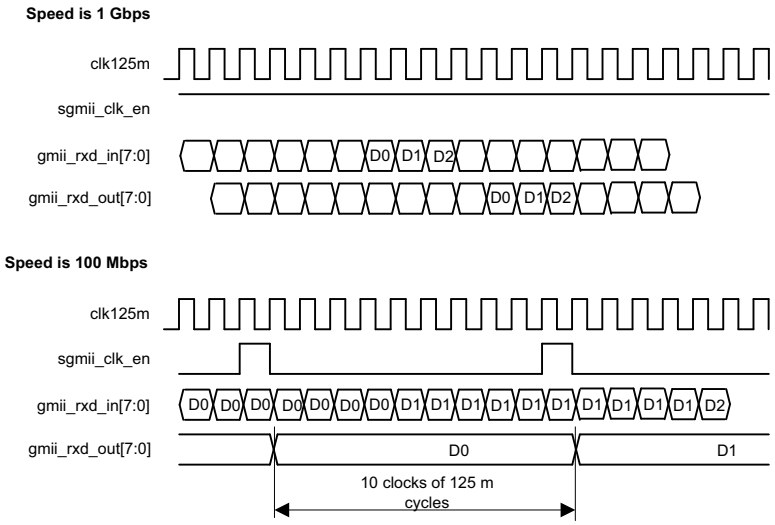


Figure 4-13: **Receiver Rate Adaptation Module Sampling in MAC Mode**

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`.

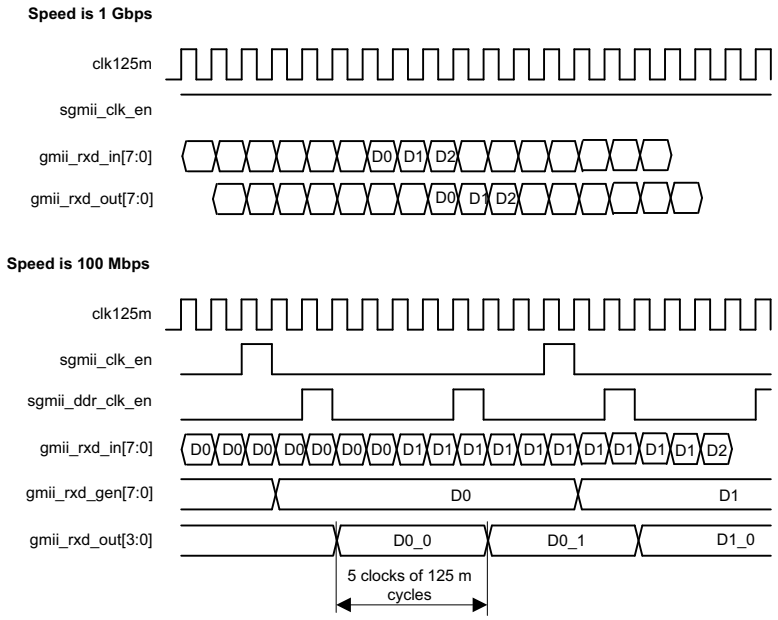


Figure 4-14: **Receiver Rate Adaptation Module Sampling in PHY Mode**

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.

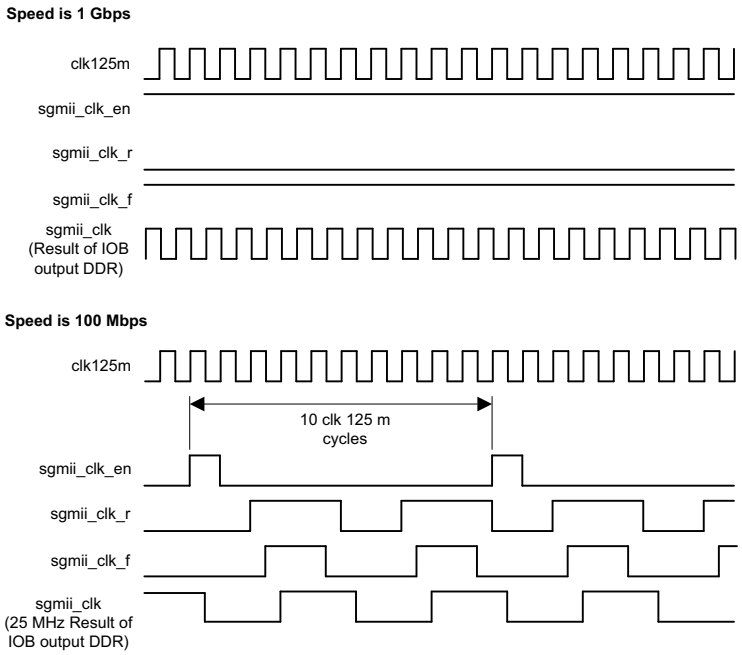


Figure 4-15: **Clock Generator Output Clock and Clock Enables**

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 Zynq®-7000, Virtex®-7, Kintex®-7, or Artix®-7 devices and 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 Kintex UltraScale, Zynq-7000, Virtex-7, Kintex-7 or Artix-7 device 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. The clocking logic is included in a separate module <component_name>_clocking which is instantiated in the <component_name>_support module.

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 tool log file for Vivado® Design Suite (XCI file) that was created when the 7 series FPGA transceiver wizard project was generated is available in the following location:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/transceiver/<component_name>_gtwizard.xci
```

The XCI file can be used as an input to the Vivado tools project by clicking on **Add Sources** in the Flow Navigator task bar and selecting the XCI file.

The XCI 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) [Ref 6].

Note: The optional Transceiver Control and Status ports are not shown here. These ports have been brought up to the <component_name> module level

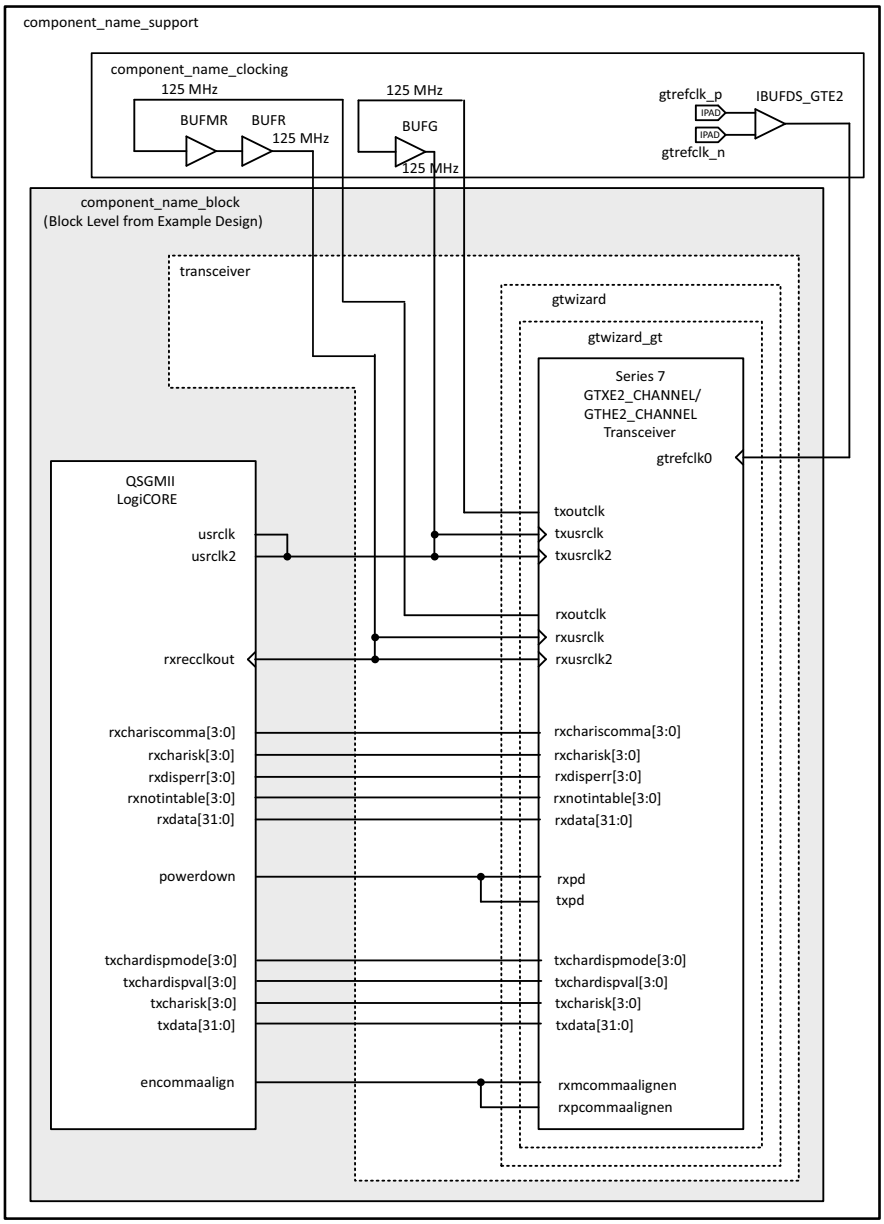


Figure 5-1: **QSGMII Connection to Virtex-7 FPGA Transceivers**

Zynq-7000/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. The clocking logic is included in a separate module, <component_name>_clocking, which is instantiated in the <component_name>_support module.

The two wrapper files immediately around the GTP 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 tool log file for Vivado Design Suite (XCI file) that was created when the 7 series FPGA transceiver wizard project was generated is available in the following location:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/transceiver/<component_name>_gtwizard.xci
```

The XCI file can be used as an input to Vivado project by clicking on **Add Sources** in the Flow Navigator task bar and selecting the XCI file. The XCI 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) [Ref 6].

Note: The optional Transceiver Control and Status ports are not shown here. These ports have been brought up to the <component_name> module level.

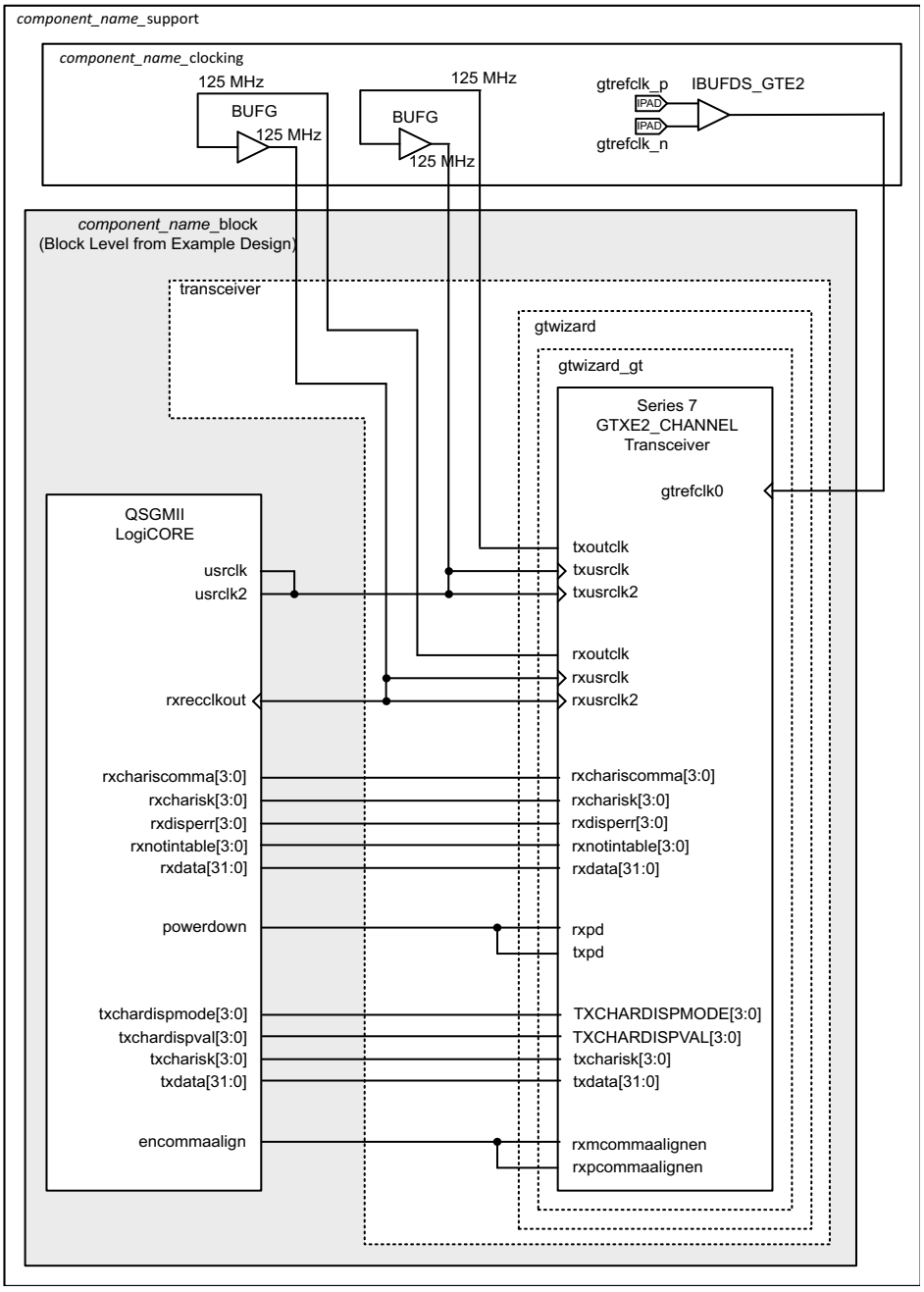


Figure 5-2: **QSGMII Connection to Zynq-7000/Kintex-7 FPGA Transceivers**

Artix-7 Devices

The QSGMII 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`). The `userclk` and `userclk2` signals are input back into the GTPE2 transceiver on the user interface clock ports `txusrclk` and `txusrclk2`. The `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. The clocking logic is included in separate module, <component_name>_clocking, which is instantiated in the <component_name>_support module. In Figure 5-3 clocking resources BUFMR and BUFR are used on `rxoutclk`. In certain devices BUFMRs are not available. In this case other clocking schemes need to be used with the core generated with the **Include Shared Logic in Example Design** option.

The two wrapper files immediately around the GTP 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 tool log file for Vivado Design Suite (XCI file) that was created when the 7 series FPGA transceiver wizard project was generated is available in the following location:

```
<project_dir>/<project_name>/<project_name>.srcs/sources1/ip/<component_name>/
synth/transceiver/<component_name>_gtwizard.xci
```

The XCI 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 XCI file.The XCI file itself contains a list of all of the transceiver wizard attributes that were used. For further information, see the 7 *Series FPGAs GTP Transceivers User Guide* (UG482) [Ref 7].

Note: The optional Transceiver Control and Status ports are not shown here. These ports have been brought up to the <component_name> module level.

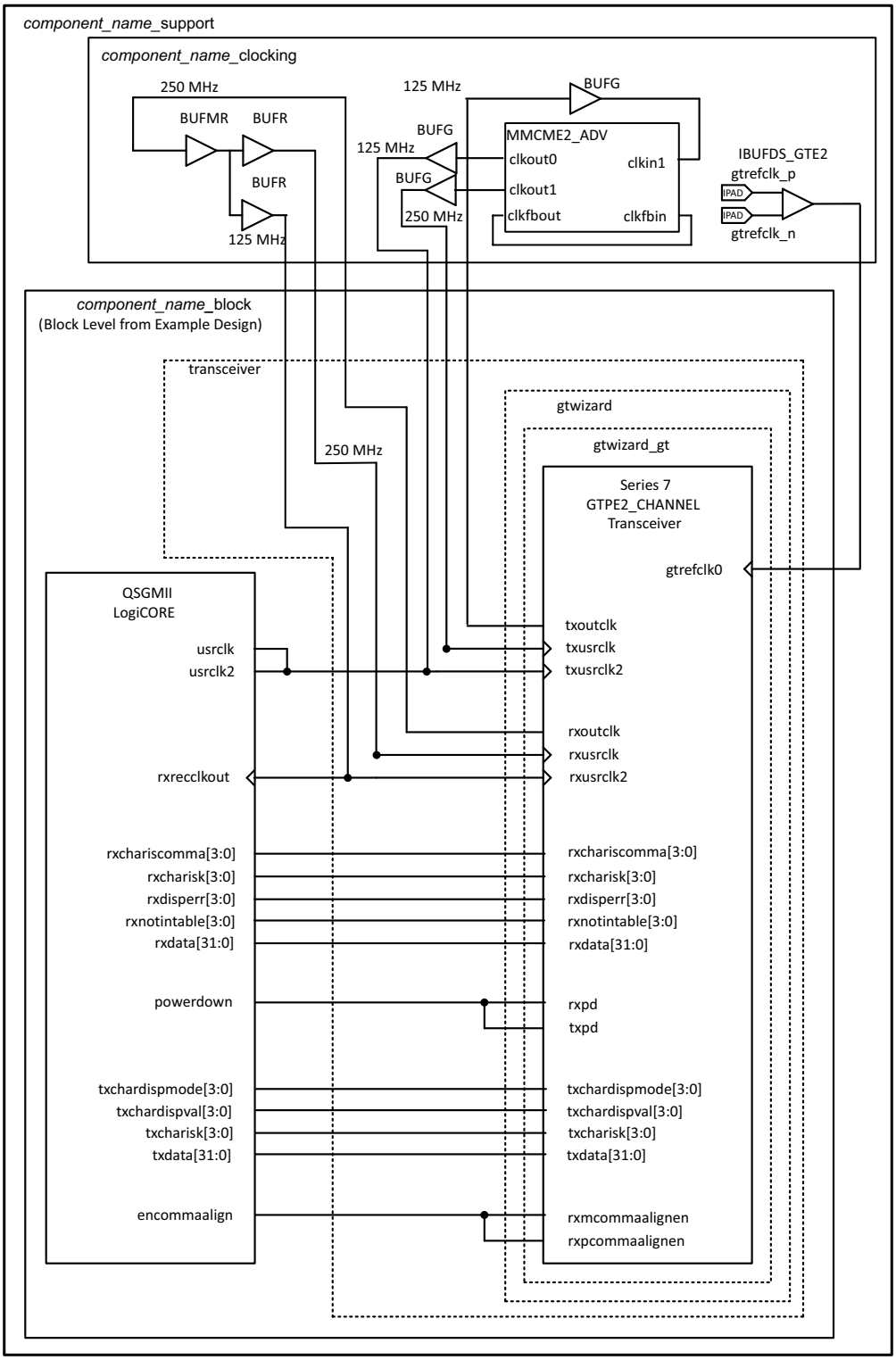


Figure 5-3: **QSGMII Connection to Artix-7 FPGA Transceivers**

UltraScale Devices

The core is designed to integrate with the UltraScale device transceiver. Figure 5-4 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 core hdl when a UltraScale device transceiver is used.

The 125 MHz differential reference clock is routed directly to the 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 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_GT. This clock is then used to source the receive logic from Transceiver receive side output to the rxelastic buffer in the core. The clocking logic is included in a separate module, <component_name>_clocking, which is instantiated in the <component_name>_support module.

The transceiver subcore is auto generated and instantiated when the QSGMII core is generated.

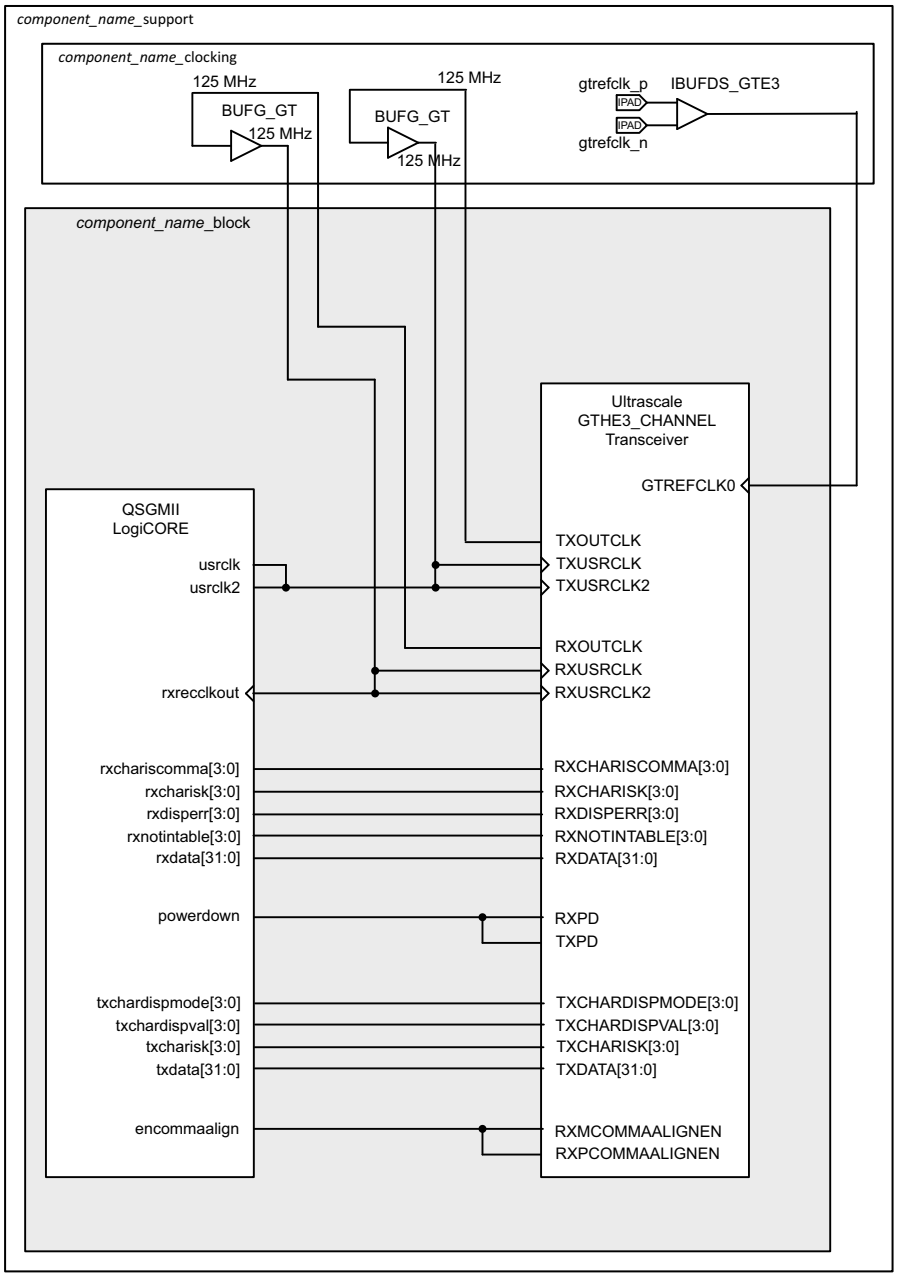


Figure 5-4: **QSGMII Connection to UltraScale Device Transceivers**

Clock Sharing Across Multiple Cores with Transceivers

Virtex-7 Devices

Figure 5-5 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. One instance of the core is generated with the **Include Shared Logic in Core** option. This instance contains all the clocking logic that can be shared. The remaining instances can be generated using the **Include Shared Logic in Example Design** option. This method of using shared logic core is limited to a GT Quad.

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 5-5, and cannot normally be shared across multiple transceivers. The clocking logic for `rxoutclk` can only be shared if it is known that the transceiver and core pairs across QSGMII instances are synchronous.

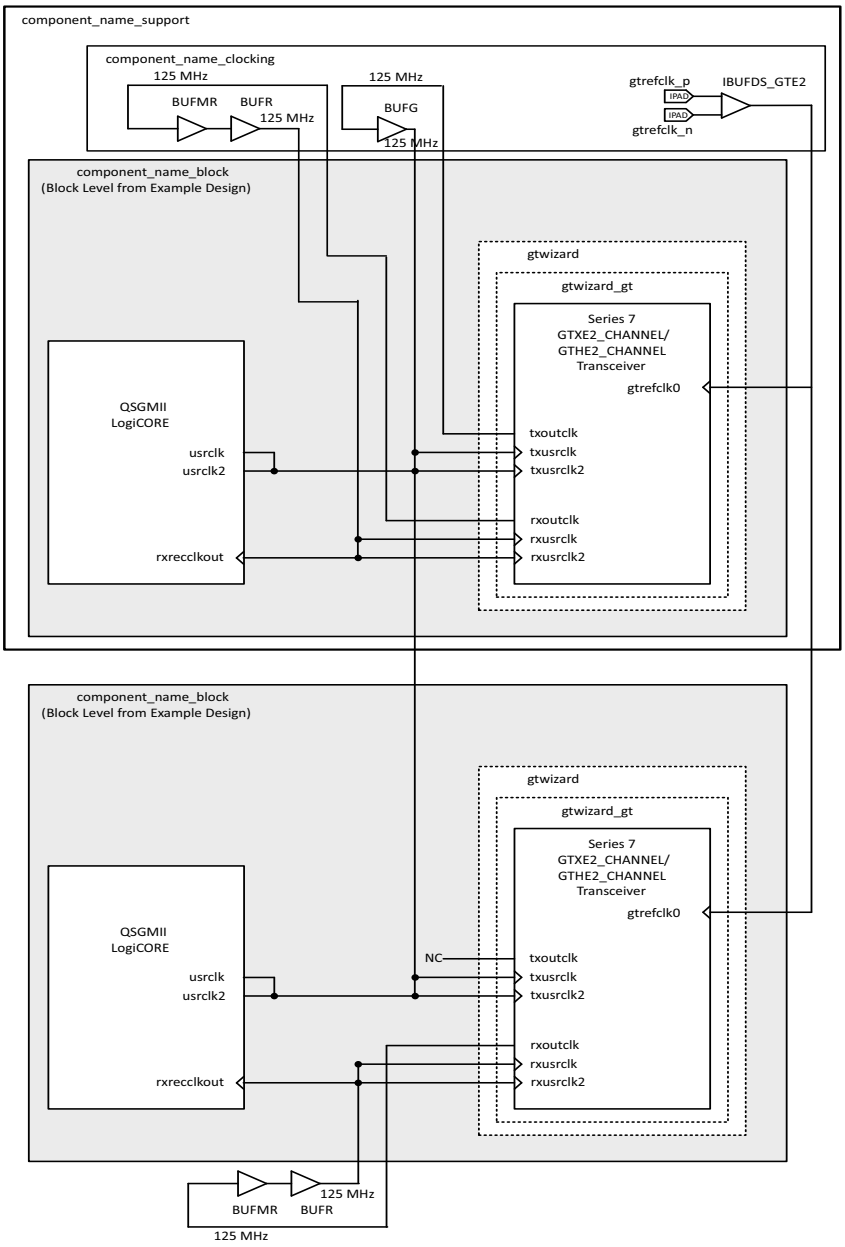


Figure 5-5: **Clock Management with Multiple Core Instances with Virtex-7 FPGA Transceivers**

Zynq-7000 and Kintex-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. One instance of the core is generated with the **Include Shared Logic in Core** option. This instance contains all the clocking logic that can be shared.

The remaining instances can be generated using the **Include Shared Logic in Example Design** option. This method of using shared logic core is limited to a GT Quad.

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 5-6, and cannot be normally shared across multiple transceivers. The clocking logic for `rxoutclk` can only be shared if it is known that the transceiver and core pairs across QSGMII instances are synchronous. In this case the receive clock outputs of clocking module can be used.

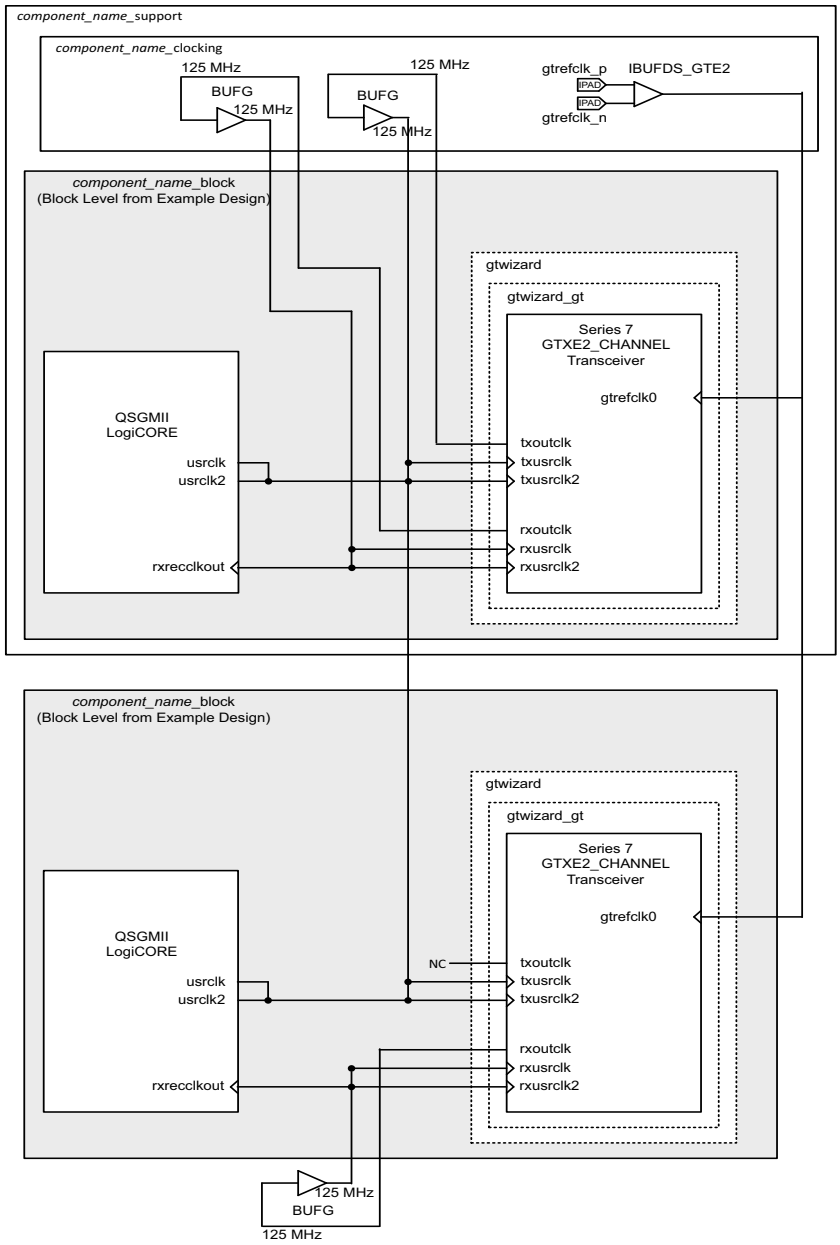


Figure 5-6: **Clock Management with Multiple Core Instances with Zynq-7000/Kintex-7 FPGA Transceivers**

Artix-7 Devices

Figure 5-7 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. One instance of the core is generated with the **Include Shared Logic in Core** option. This instance contains all the clocking logic that can be shared. The remaining instances can be generated using the **Include Shared Logic in Example Design** option. The method of using the shared logic core is limited to a GT Quad.

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 GTP 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 5-7, and cannot normally be shared across multiple transceivers. The clocking logic for `rxoutclk` can only be shared if it is known that the transceiver and core pairs across QSGMII instances are synchronous. In this case the receive clock outputs of clocking module can be used. In some devices BUFMRs are not available and this scheme might not be feasible. Other schemes with a core just generated using the **Include Shared Logic in Example Design** option should be considered.

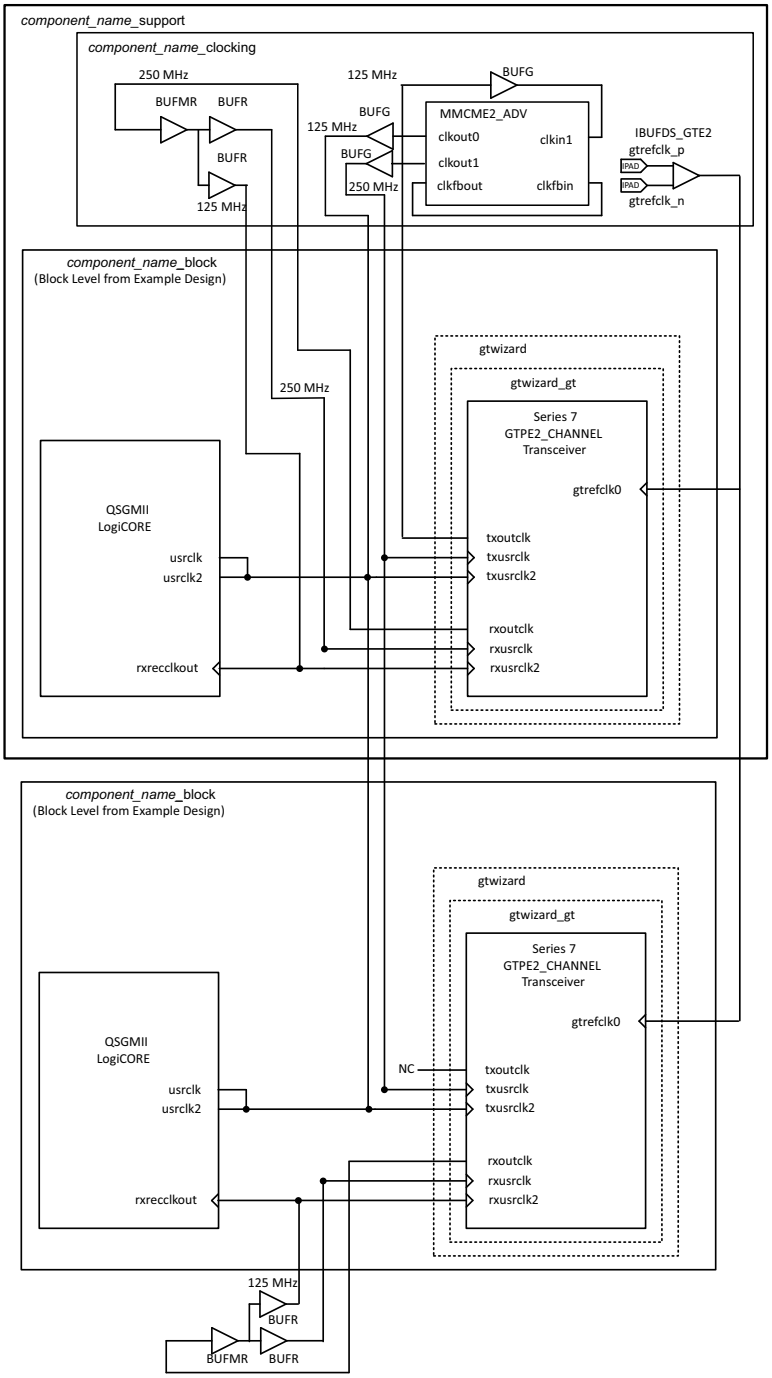


Figure 5-7: **Clock Management with Multiple Core Instances with Artix-7 FPGA Transceivers**

UltraScale Devices

Figure 5-8 illustrates sharing clock resources across two instantiations of the core when using UltraScale device transceivers. Additional cores can be added by continuing to instantiate extra block level modules. One instance of the core is generated with the **Include Shared Logic in Core** option. This instance contains all the clocking logic that can be shared. The remaining instances can be generated using the Include Shared Logic in Example Design option. This method of using shared logic core is limited to a GT Quad.

To provide the device logic clocks for all core instances, select a `txoutclk` port from any transceiver and place it onto the global clock routing using BUFG_GTs; these 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 the global clock routing using a BUFG_GT, as illustrated in Figure 5-8, and cannot be normally shared across multiple transceivers. The clocking logic for `rxoutclk` can only be shared if it is known that the transceiver and core pairs across QSGMII instances are synchronous. In this case the receive clock outputs of clocking module can be used.

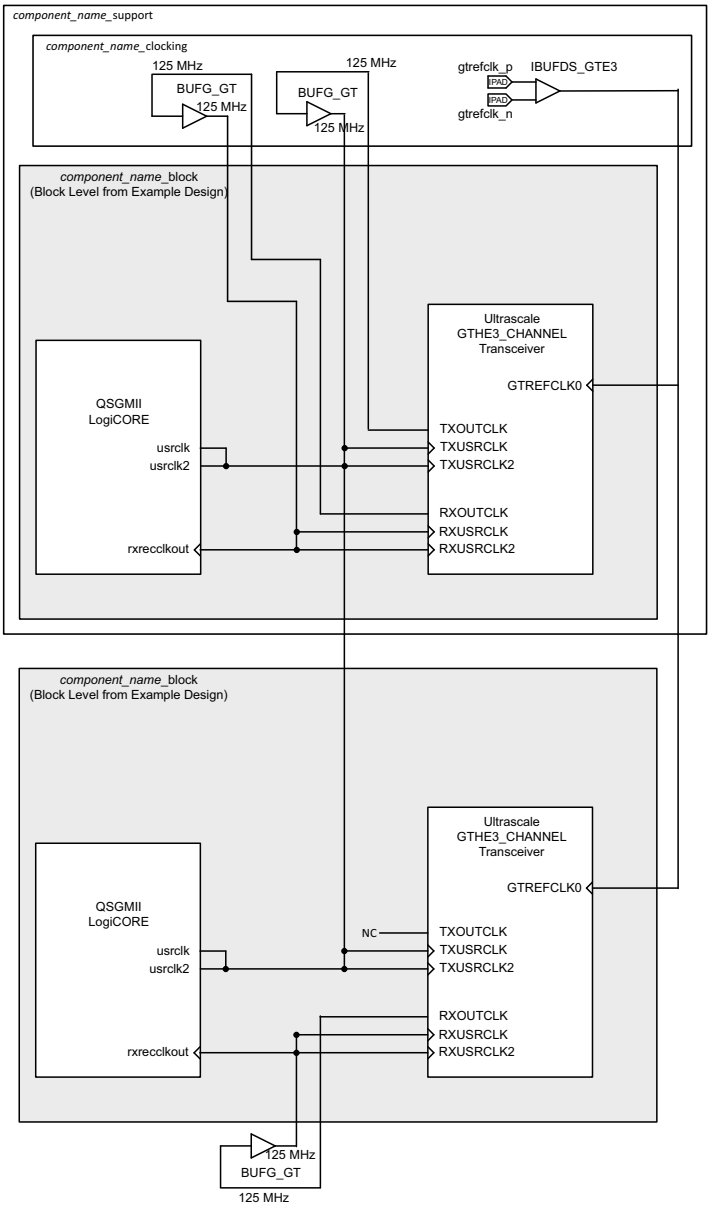


Figure 5-8: **Clock Management with Multiple Core Instances with UltraScale Device Transceivers**

Customizing and Generating the Core

The QSGMII core is generated using the Vivado® IP catalog tool. This chapter describes the Vivado Integrated Design Environment (IDE) options used to generate and customize the core.

Vivado IDE

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or popup menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 10].

Note: Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version.

Core Customization Vivado IDE

Figure 6-1 displays the QSGMII customization Vivado IDE, used to set core parameters and options. For help starting and using Vivado design tools on your system, see the documentation included with the Vivado design suite at www.xilinx.com/support/software_manuals.htm.

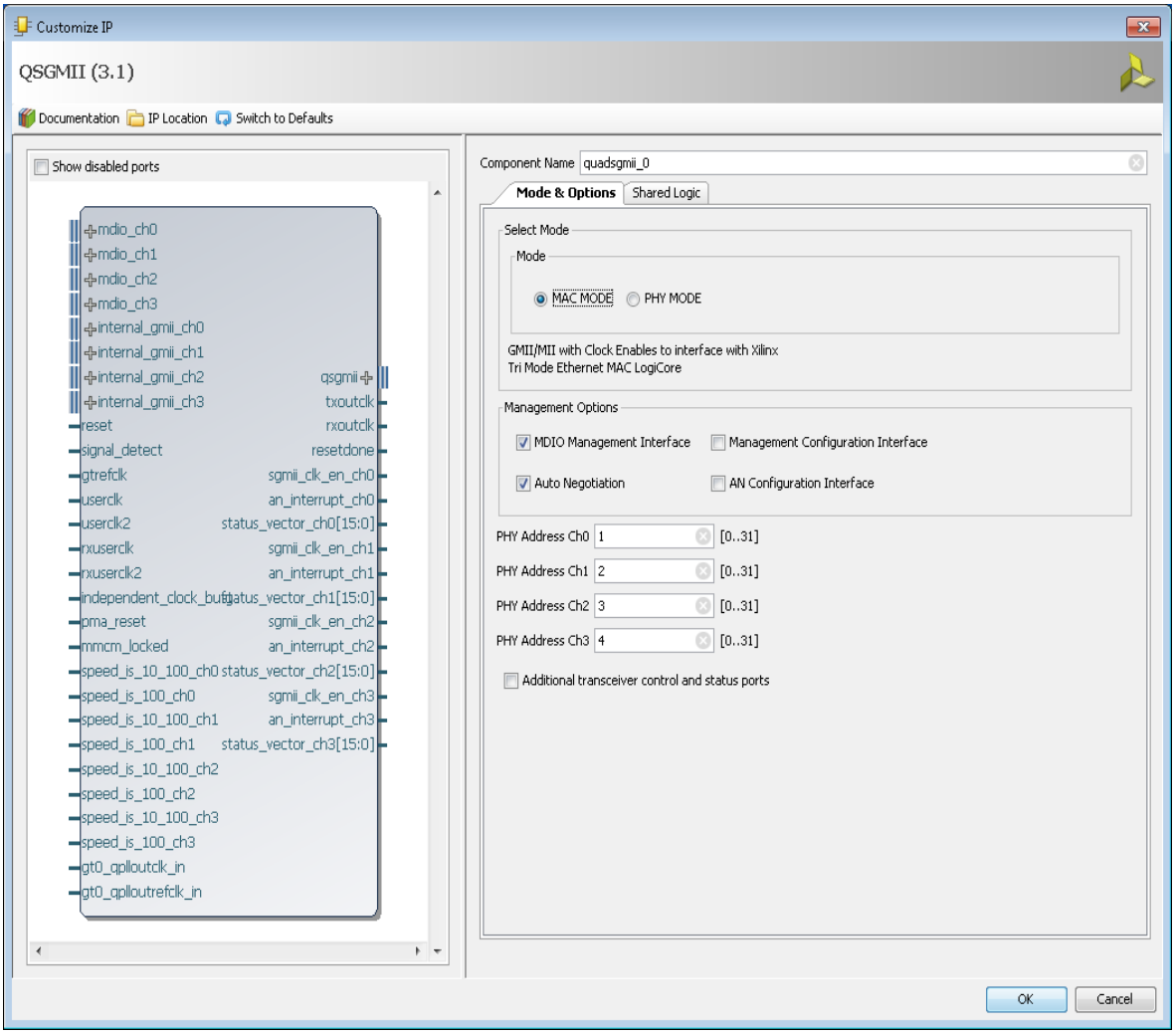


Figure 6-1: **Core Customization Screen**

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 the 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 IP cores. 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. MDIO Management Interface is selected by default.

MDIO Configuration Interface

Select this option to include additional configuration interface to program configuration register 0. This is in addition to the MDIO Management Interface. This option is always selected if MDIO Management Interface is disabled.

Auto-Negotiation

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

AN Configuration Interface

Select this option to include additional configuration interface to program the AN Advertisement register (Register 4). This option is valid only if Auto-Negotiation is enabled.

PHY Address Chx

This defines the PHY address of the MDIO interface of individual channels in the core. The value should be in the range of 0 to 31.

Additional Transceiver Control and Status Ports

If selected, enables additional transceiver control ports for DRP, Tx Driver, Rx Equalization, and other features such as PRBS.

Select Interface Vivado IDE

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

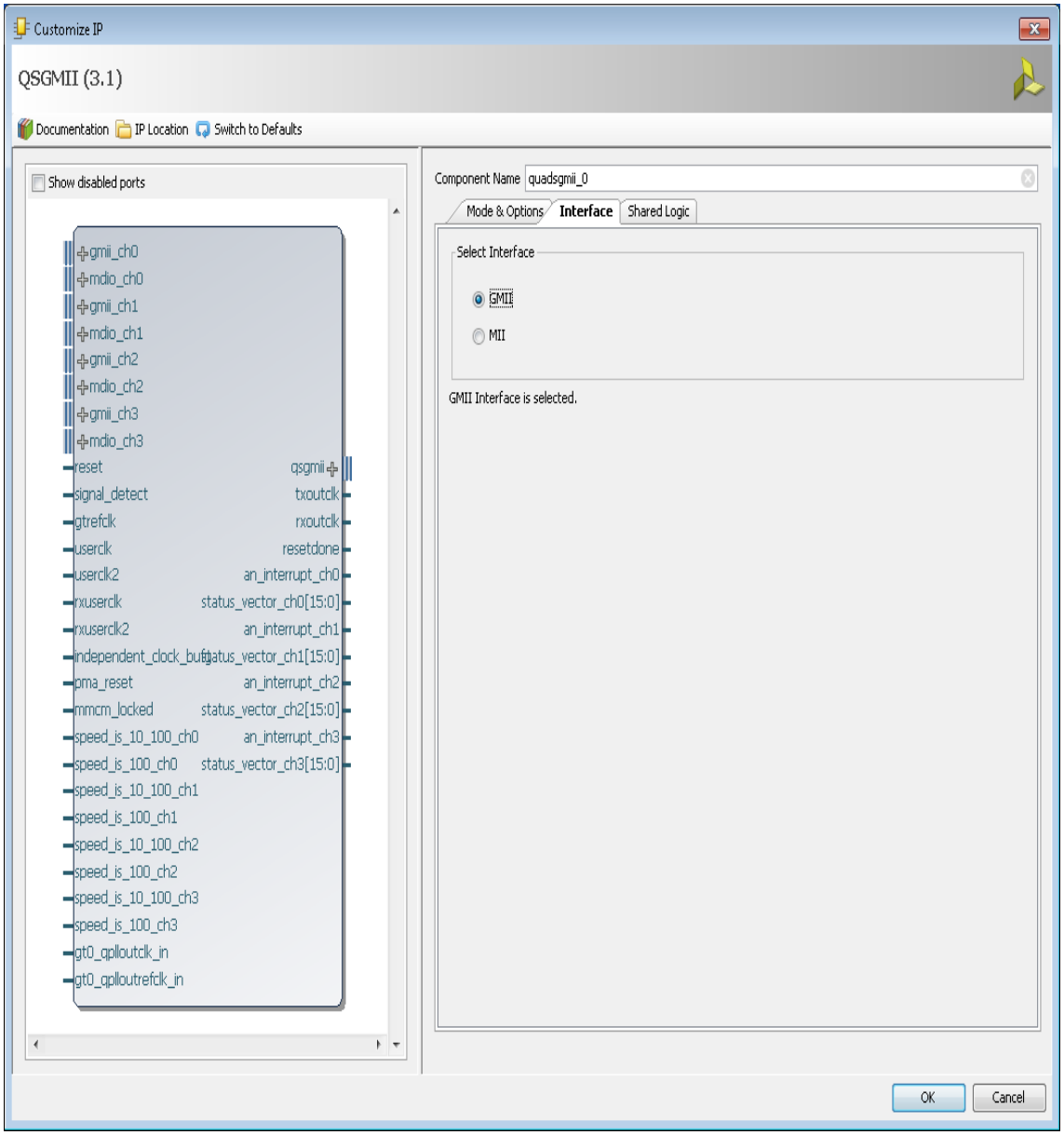


Figure 6-2: **Select Interface Vivado IDE**

Shared Logic Options in Vivado IDE

Figure 6-3 displays the shared logic options as well as transceiver control and status selection in the Vivado IDE.

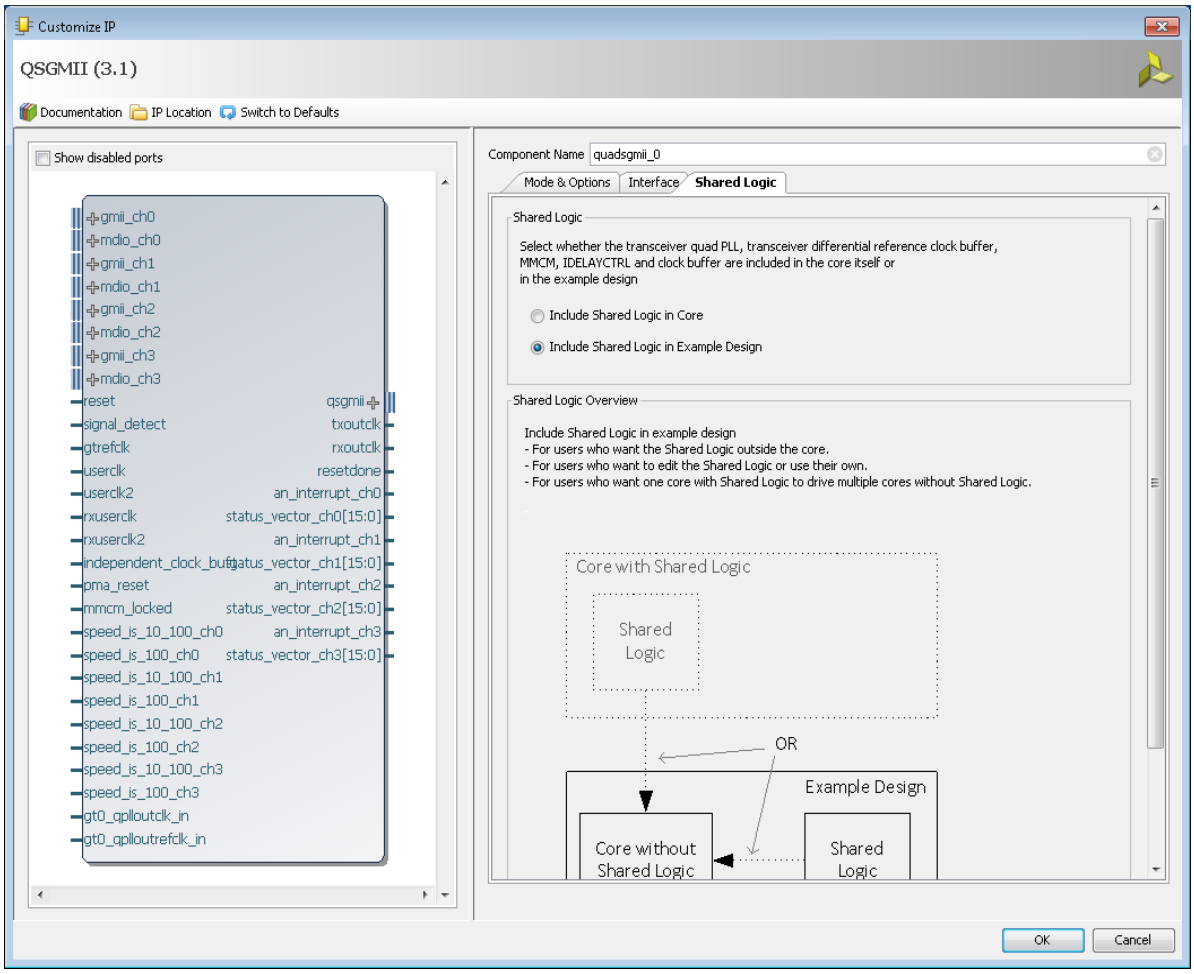


Figure 6-3: **Shared Logic Options**

Shared Logic

Determines whether some shared clocking logic is being included as part of the core itself or as part of the example design.

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.

For details, see "Generating IP Output Products" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

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 design tools and as such this filegroup is not displayed. These files are delivered into the project source directory.

Constraining the Core

This chapter contains information about constraining the core in the Vivado® Design Suite environment and 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 10, 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 Zynq®-7000, 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 Zynq-7000 and Artix-7 devices

I/O Location Constraints

No specific I/O location constraints are required.

However, when employing BUFIO and BUFR regional clock routing, 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) [Ref 6].

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
*/*/*/transceiver_inst/gtwizard_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.

```
#*********************************************************
# PCS/PMA Clock period Constraints: please do not relax *
#*********************************************************
create_clock -add -name gtrefclk -period 8.000 [get_ports gtrefclk_p]
```

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) [Ref 6].

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 get_cells
*/*/*/transceiver_inst/gtwizard_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.

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

create_clock -add -name gtrefclk -period 8.000 [get_ports gtrefclk_p]
```

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.

Zynq-7000 and 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) [Ref 6].

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 get_cells
*/*/*/transceiver_inst/gtwizard_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.

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

create_clock -add -name gtrefclk -period 8.000 [get_ports gtrefclk_p]
```

GTX Transceiver Attributes

The Zynq-7000/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) [Ref 7].

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 get_cells
*/*/*/transceiver_inst/gtwizard_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.

```
#*********************************************************
# PCS/PMA Clock period Constraints: please do not relax *
#*********************************************************
create_clock -add -name gtrefclk -period 8.000 [get_ports gtrefclk_p]
```

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 Vivado IDE. When implementing an external GMII, the Transmitter Elastic Buffer embedded in the QSGMII block will 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

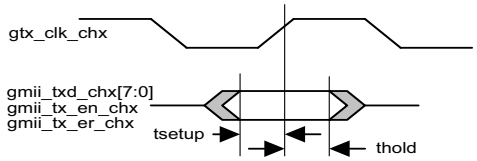


Figure 7-1: **Input GMII Timing Specification**

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

Zynq-7000, Virtex-7, Kintex-7, and Artix-7 Devices

Figure C-1 in Appendix C illustrates the GMII input logic provided in the core for the Zynq-7000, Virtex-7, Kintex-7, and Artix-7 family.

Figure C-2 in Appendix C illustrates the MII input logic provided in the core for the Zynq-7000, 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.

Simulation

For comprehensive information about Vivado® simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 9].

All simulation sources are included that are required by the core. Simulation of the QSGMII core at the core level is not supported without the addition of a test bench (not supplied). Simulation of the example design is supported.

Synthesis and Implementation

This chapter contains information about synthesis and implementation in the Vivado® Design Suite.

For details about synthesis and implementation, see "Synthesizing IP" and "Implementing IP" in the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

All synthesis sources are included that are required by the core. For the QSGMII core this is a mix of both encrypted and unencrypted source. Only the unencrypted sources are visible and optionally editable by using the **Unlink IP Vivado** option.

Detailed Example Design

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

Example Design

Figure 10-1 illustrates an example design for top-level HDL for the QSGMII using a device-specific transceiver (Zynq®-7000, Virtex®-7, Kintex®-7, or Artix®-7 devices).

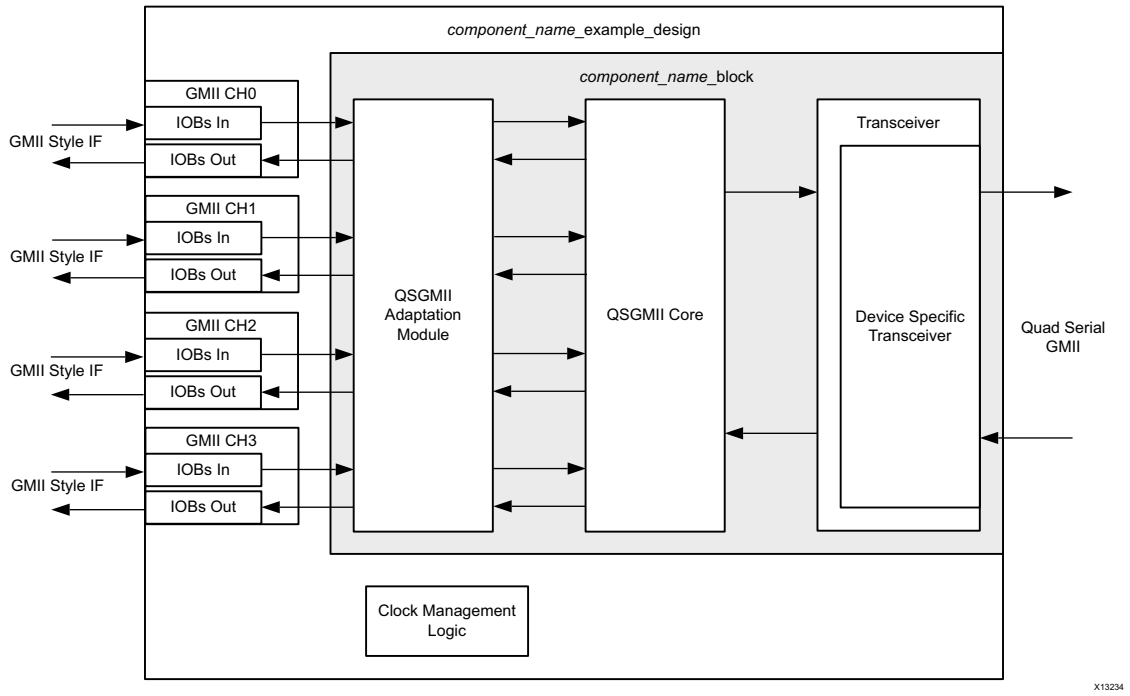


Figure 10-1: **Example Design HDL for QSGMII**

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
- Implements an external GMII-style interface
- 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 in case shared logic in the core is selected, otherwise support level.
- Derives the clock management logic for device-specific transceiver and the core.

The next few pages in this section describe each of the example design blocks (and associated HDL files) in detail. The example design can be opened in a separate project by generating the **Examples** output product, then right clicking the core instance and choosing **Open IP Example Design..**.

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 in case shared logic in the core is selected, otherwise support level.
- Clock management logic for the core and the device-specific transceiver, including DCM (if required) and Global Clock Buffer instances.

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.

Support Level HDL

The following files describe the block level for the QSGMII core. The files can be found in the /synth directory if shared logic in the core is selected or /example_design if shared logic in the example design is selected while generating the core.

VHDL

```
/synth/<component_name>_support.vhd or
/example_design/support/<component_name>_support.vhd
```

Verilog

```
/synth/<component_name>_support.vhd or
/example_design/support/<component_name>_support.vhd
```

<component_name>_support module instantiates idelayctrl, clocking and reset modules.

VHDL

```
/synth/<component_name>_idelayctrl.vhd or
/example_design/support/<component_name>_idelayctrl.vhd

/synth/<component_name>_clocking.vhd or
/example_design/support/<component_name>_clocking.vhd

/synth/<component_name>_resets.vhd or
/example_design/support/<component_name>_resets.vhd
```

Verilog

```
/synth/<component_name>_idelayctrl.v or
/example_design/support/<component_name>_idelayctrl.v

/synth/<component_name>_clocking.v or
/example_design/support/<component_name>_clocking.v

/synth/<component_name>_resets.v or
/example_design/support/<component_name>_resets.v
```

Block Level HDL

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

VHDL

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

Verilog

```
/synth/<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

- External GMII logic, including IOB and Double Data Rate (DDR) register instances, where required
- 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 10-1. 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 Zynq-7000, 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

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

Verilog

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

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

Zynq-7000, Virtex-7, Kintex-7 and Artix-7 Device Transceiver Wizard Files

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 delivered might include some or all of the following:

VHDL

```
/synth/transceiver/<component_name>_gtwizard_init.vhd
/synth/transceiver/<component_name>_gtwizard_tx_startup_fsm.vhd
/synth/transceiver/<component_name>_gtwizard_rx_startup_fsm.vhd
/synth/transceiver/<component_name>_gtwizard_gtp_gtrxreset_seq.vhd
/synth/transceiver/<component_name>_gtwizard_gtp_rxpmarst_seq_vhd.vhd
/synth/transceiver/<component_name>_gtwizard_gtp_rxrate_seq.vhd
/synth/transceiver/<component_name>_gtwizard.vhd
/synth/transceiver/<component_name>_gtwizard_gt.vhd
/synth/transceiver/component_name>_gtwizard_multi_gt.vhd
```

Verilog

```
/synth/transceiver/<component_name>_gtwizard_init.v
/synth/transceiver/<component_name>_gtwizard_tx_startup_fsm.v
/synth/transceiver/<component_name>_gtwizard_rx_startup_fsm.v
/synth/transceiver/<component_name>_gtwizard_gtp_gtrxreset_seq.v
/synth/transceiver/<component_name>_gtwizard_gtp_rxpmarst_seq_vhd.v
/synth/transceiver/<component_name>_gtwizard_gtp_rxrate_seq.v
/synth/transceiver/<component_name>_gtwizard.v
/synth/transceiver/<component_name>_gtwizard_gt.v
/synth/transceiver/component_name>_gtwizard_multi_gt.v
```

To re-run the transceiver wizard, a Vivado Design Suite tool XCI file for the wizard is included. This file defines all the required wizard attributes used to generate the preceding files. The XCI file is in the following location:

```
/synth/transceiver/<component_name>_gtwizard.xci
```

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 four bits of the MII.

The QSGMII Adaptation Module is described in the following files:

VHDL

```
<project_dir>/<component_name>/synth/qsgmii_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>/synth/qsgmii_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.

Test Bench

This chapter contains information about the provided test bench in the Vivado® Design Suite.

Figure 11-1 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.

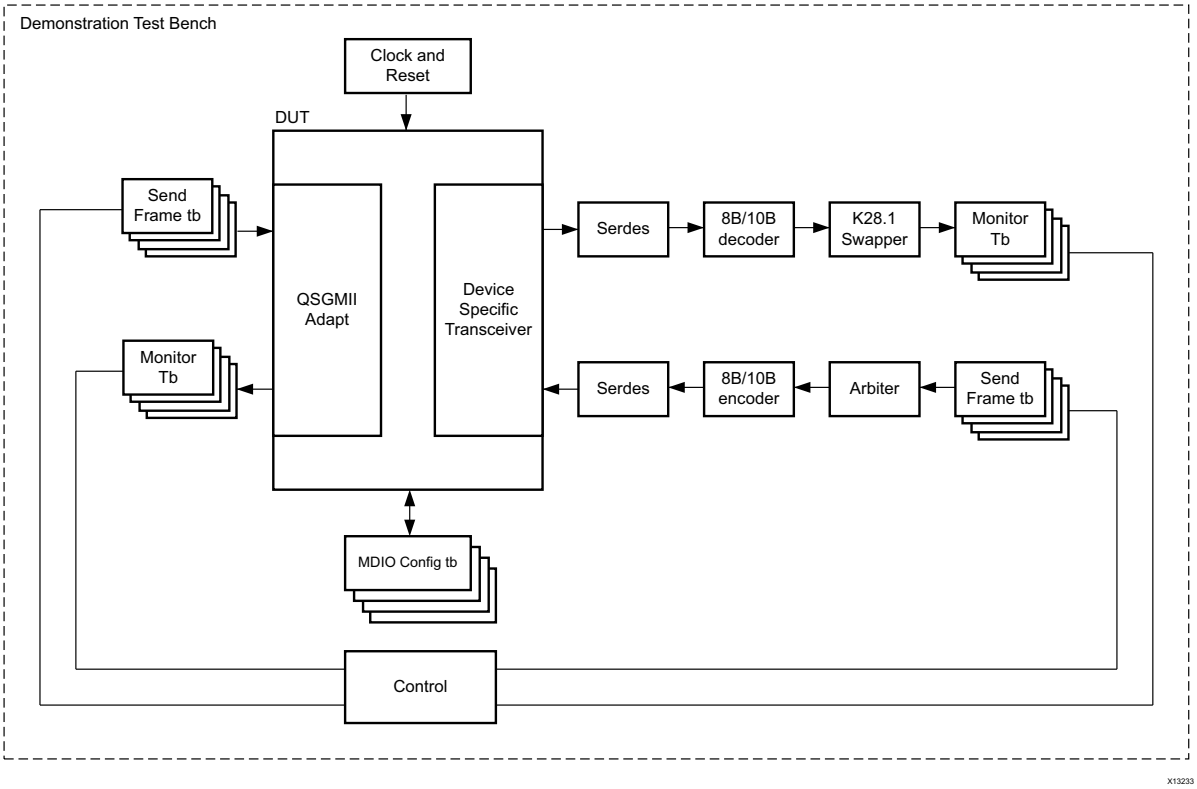


Figure 11-1: **Demonstration Test Bench for QSGMII**

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 pre loaded 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
```

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 been validated with the KC705 board in only external loopback mode. Please note this core has not been validated for compliance with other QSGMII vendors.

Compliance Testing

This core has not been validated for compliance with other QSGMII vendors.

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating

For information on migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 11].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

In the 3.0 version of the core, there have been several changes that make the core pin-incompatible with the previous version(s). These changes were required as part of the general one-off hierarchical changes to enhance the customer experience and are not likely to occur again.

Shared Logic

As part of the hierarchical changes to the core, it is now possible to have the core itself include all of the logic that can be shared between multiple cores, which was previously exposed in the example design for the core.

If you are updating a previous version to the 3.0 version with Shared Logic, there is no simple upgrade path; it is recommended to consult the Shared Logic sections of this document for more guidance.

Port Changes from v2.0 to v3.0

Ports Added

The following ports were added to the core (non-shared logic).

Table B-1: **Ports Added (non-shared logic)**

Port name and width	In/Out	Description	What to do
rxoutclk	Output	`rxoutclk` from the Transceiver	This was previously connected internally to clocking elements and routed to `rxuserclk` and `rxuserclk2`. This can be left open if `rxoutclk` can be shared across instances or if not should drive clocking elements.
rxuserclk	Input	Signal from the shared logic block to the transceiver	If `rxoutclk` can be shared across instances, connect O/P of shared logic block. If not connect to `rxoutclk` after passing through additional clocking elements.
rxuserclk2	Input	Signal from the shared logic block to the transceiver	If `rxoutclk` can be shared across instances, connect O/P of shared logic block. If not connect to `rxoutclk` after passing through additional clocking elements.
gt0_pll0outclk_in	Input	Valid only for Artix®-7 families. Indicates out clock from PLL0 of GT Common.	Should be connected to signal of same name from GT Common
gt0_pll0outrefclk_in	Input	Valid only for Artix-7 families. Indicates reference out clock from PLL0 of GT Common.	Should be connected to signal of same name from GT Common
gt0_pll1outclk_in	Input	Valid only for Artix-7 families. Indicates out clock from PLL1 of GT Common.	Should be connected to signal of same name from GT Common
gt0_pll1outrefclk_in	Input	Valid only for Artix-7 families. Indicates reference out clock from PLL1 of GT Common.	Should be connected to signal of same name from GT Common
gt0_pll0lock_in	Input	Valid only for Artix-7 families. Indicates out PLL0 of GT Common has locked.	Should be connected to signal of same name from GT Common
gt0_pll0refclklost_in	Input	Valid only for Artix-7 families. Indicates out reference clock for PLL0 of GT Common is lost.	Should be connected to signal of same name from GT Common

Table B-1: **Ports Added (non-shared logic) *(Cont'd)***

Port name and width	In/Out	Description	What to do
gt0_pll0reset_out	output	Valid only for Artix-7 families. Reset for PLL of GT Common from reset fsm in GT Wizard	Should be connected to signal of same name from GT Common or can be left open if not needed
gt0_qplloutclk_in	Input	Valid only for non Artix-7 families. Indicates out clock from PLL of GT Common.	Should be connected to signal of same name from GT Common
gt0_qplloutrefclk_in	Input	Valid only for non Artix-7 Families. Indicates reference out clock from PLL of GT Common.	Should be connected to signal of same name from GT Common

The following ports were added to the core, but only if the transceiver debug feature was requested during core customization. Consult the relevant transceiver user guide for more information on using these control/status ports

Table B-2: **Ports Added for Transceiver Debug Feature**

Port name and width	In/Out	Description	What to do
gt0_rxchariscomma_out[3:0]	Output	RX Character is Comma indication	If you want to be more compatible with the previous version of the core and also if DRP interface was not used, do not request the Transceiver Debug Feature.
gt0_rxcharisk_out[3:0]	Output	RX Character is K indication	
gt0_rxbyteisaligned_out	Output	RX Byte is aligned indication	
gt0_rxbyterealign_out	Output	Rx Byte is realigned indication	
gt0_rxcommadet_out	Output	RX Comma is detected indication	
gt0_txpolarity	Input	Switch the sense of the TXN/P pins	If you want to be more compatible with the previous version of the core and also if DRP interface was not used, do not request the Transceiver Debug Feature. Otherwise, drive this signal according to the relevant transceiver user guide.
gt0_txdiffctrl[3:0]	Input	Can be used to tune the transceiver TX waveform	
gt0_txprecursor[4:0]	Input	Can be used to tune the transceiver TX waveform	
gt0_txpostcursor[4:0]	Input	Can be used to tune the transceiver TX waveform	
gt0_rxpolarity	Input	Switch the sense of the RXN/P pins	
gt0_txprbssel_in[2:0]	Input	TX Pattern Generator control signals to test signal integrity	
gt0_txprbsforceerr_in	Input	TX Pattern Generator control signals to test signal integrity	
gt0_rxprbscntreset_in	Input	RX Pattern Checker reset	
gt0_rxprbserr_out	Output	RX Pattern Checker error output	

Table B-2: **Ports Added for Transceiver Debug Feature *(Cont'd)***

Port name and width	In/Out	Description	What to do
gt0_rxprbssel_in[2:0]	Input	RX Pattern Checker control signals to test signal integrity	
gt0_loopback_in[2:0]	Input	Loopback within transceiver	
gt0_txresetdone_out	Output	Transmitter Reset Done	
gt0_rxresetdone_out	Output	Receiver Reset Done	
gt0_rxdisperr_out[3:0]	Output	Indicates there is disparity error in received data	
gt0_rxnotintable_out[3:0]	Output	Indicates received 10 bit pattern was not found in 8B/10B decode table	
gt0_eyescanreset	Input	Reset the EYE Scan logic	
gt0_eyescantrigger	Input	Trigger the EYE Scan logic	
gt0_eyescandataerror	Output	Signals an error during Eye Scan	
gt0_rxrate[2:0]	Input	Change the PLL Divider value	
gt0_rxcdrhold	Input	Freeze the CDR loop	
gt0_rxcdrlock_out	Output	CDR loop has locked	
gt0_rxratedone_out	Output	Asserted in response to change in RXRATE	
gt0_rxlpmhfhold_in	Input	GTP Low power mode signal	
gt0_rxlpmlfhold_in	Input	GTP Low power mode signal	
gt0_rxmonitorout_out[6:0]	Output	GTX/GTH RX DFE Signal	
gt0_rxmonitorsel_in[1:0]	Input	GTX/GTH RX DFE Signal	

Ports Moved

The following ports were moved under the Transceiver Debug Feature of the core (non-shared logic). If these signals were used in the previous version, the Transceiver Debug Feature needs to be enabled and the appropriate signals mapped and remaining signals tied off to default values as specified in the relevant transceiver user guide.

Table B-3: **Ports Moved (non-shared logic)**

In/Out	Port name and width	Description	What to do
Outputs	gt0_drp_busy_out, gt0_drpdo_out, gt0_drprdy_out	These signals come from the transceiver and should be connected either to an external arbiter or to the signals described in the next row.	If there is no external arbiter, connect these signals directly to the associated signals. If they interface is not used. Can be left open
Inputs	gt0_drpen_in, gt0_drpwe_in, gt0_drpaddr_in[8:0], gt0_drpdi_i[15:0], gt0_drpclk_in	These signals go to the transceiver, either from an external arbiter or from the signals described in the previous row	If there is no external arbiter, connect these signals directly to the associated core signals. If the interface is not used, tie off the signals to ground and `gt0_drpclk_in` to `txusrclk2`.

Ports Removed

The following ports were removed from the core interface and are driven internally in the core.

Table B-4: **Ports Removed**

In/Out	Port name and width	Description	What to do
Input	link_timer_value_chx[8:0]	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).	For normal operation the value of link timer is set internally to "000110010" as specified in QSGMII specification for 1.6 ms link time. To speed up simulation this value can be set internally to "000000100". This is done though xci file by the parameter EXAMPLE_SIMULATION.

Port Changes from v3.0 to v3.1

The following ports were added to the core, but only if the transceiver debug feature was requested during core customization. Consult the relevant transceiver user guide for more information on using these control/status ports.

Table B-5: **Ports Added**

Port name and width	In/Out	Description	What to Do
gt0_rxlpmreset_in	Input	RX LPM Reset (valid only for GTP transceivers)	If not used, should be tied off as specified in the transceiver guide
gt0_rxlpmhfovrden_in	Input	Valid only for GTP transceivers	
gt0_dmonitorout_out[16:0]	Output	Digital Monitor to monitor state of LPM/DFE loops	
gt0_gttxreset_in	Input	Reset to start full transmit reset sequence. Present in only non UltraScale devices.	
gt0_txpcsreset_in	Input	Reset for TX PCS	
gt0_txpmareset_in	Input	Reset for TX PMA	
gt0_gtrxreset_in	Input	Reset to start full receive reset sequence. Present in only non UltraScale devices.	
gt0_rxpcsreset_in	Input	Reset for RX PCS	
gt0_rxpmareset_in	Input	Reset for RX PMA	
gt0_rxpmaresetdone_out	Output	Indication that RX PMA reset sequence is complete. Available only for non GTX transceivers.	
gt0_cplllock_out	Output	Indication that CPLL has locked	
gt0_txbufstatus_out[1:0]	Output	Indicates the status of TX buffer.	

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 block level and is seamlessly delivered with the core by the 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. Zynq®-7000, Kintex®-7, and Artix®-7 devices support GMII at 3.3 V or lower.

External GMII Transmitter Logic (Zynq-7000, 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 embedded in the block level of 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 XDC; these can be edited if desired.

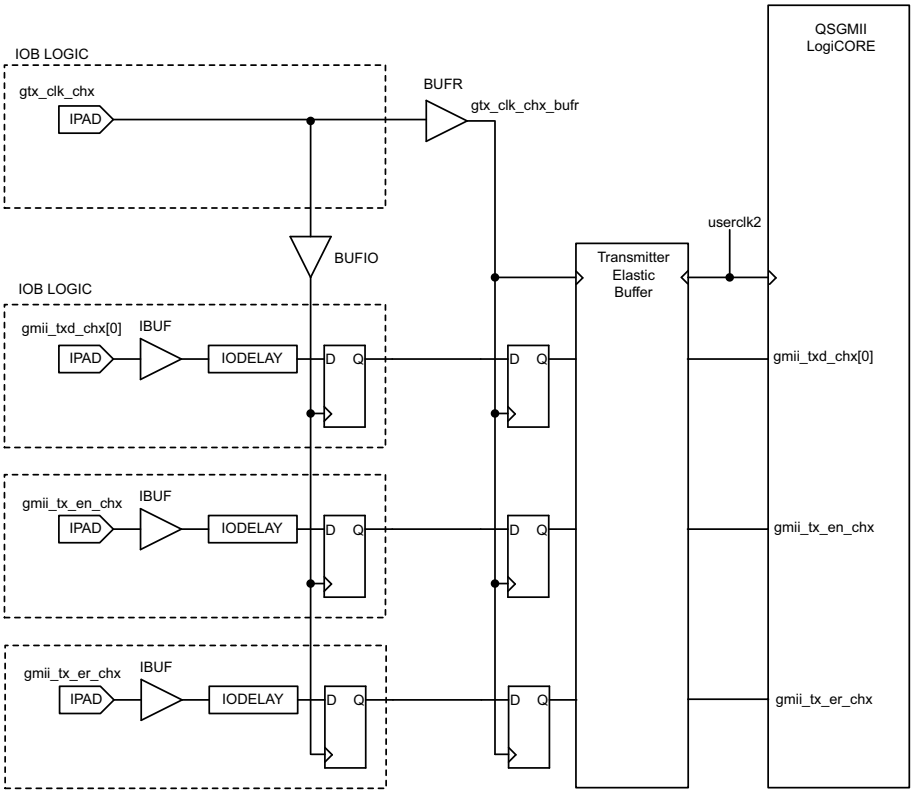


Figure C-1: **External GMII Transmitter Logic**

External MII Transmitter Logic (Zynq-7000, 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 Example Design level XDC; these can be edited if desired.

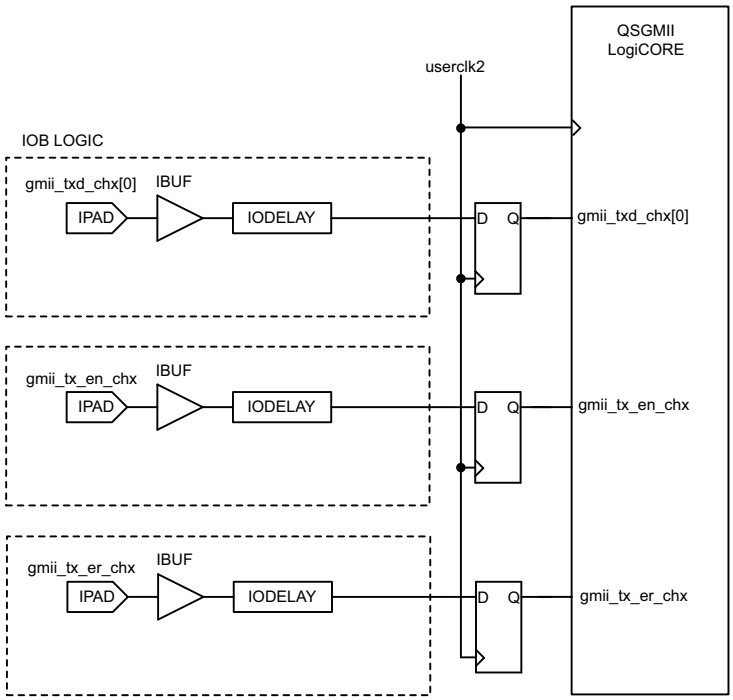


Figure C-2: **External MII Transmitter Logic**

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 in the QSGMII block when the GMII is selected. If other families are selected, equivalent primitives and logic specific to that family are automatically used in the QSGMII block.

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.

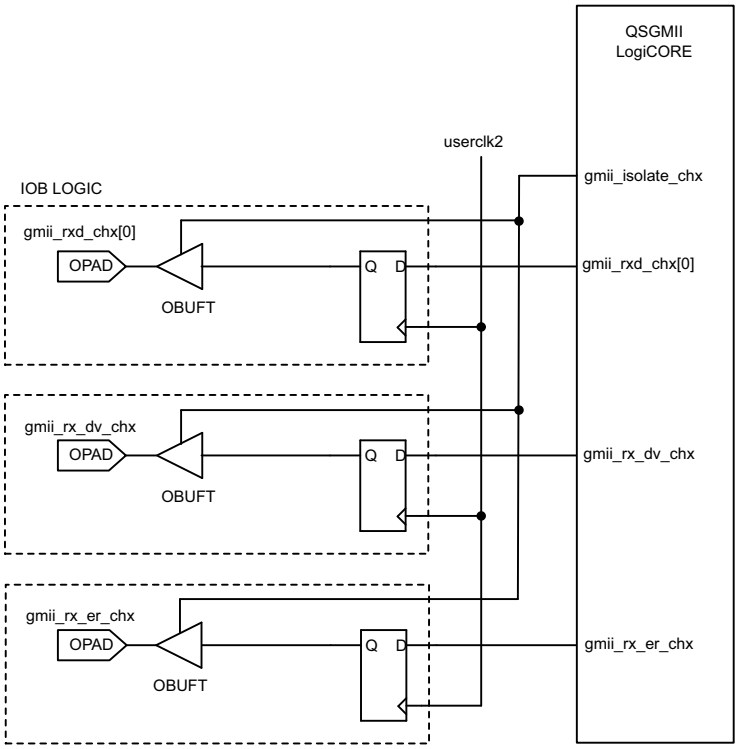


Figure C-3: **External GMII/MII Receiver Logic**

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

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

Documentation

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

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

Solution Centers

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

The Solution Center specific to this core is located at Xilinx Ethernet IP Solution Center

Answer Records

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

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

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

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

Master Answer Record for the QSGMII Core

AR 54668

Contacting Technical Support

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

To contact Xilinx Technical Support:

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

When opening a WebCase, include:

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

Note: Access to WebCase is not available in all cases. Login to the WebCase tool to see your specific support options.

Debug Tools

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

Vivado Lab Tools

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

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

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

See *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 12].

Reference Boards

Various Xilinx development boards support QSGMII core. These boards can be used to prototype designs and establish that the core can communicate with the system.

- 7 series FPGA evaluation boards
  - KC705
  - VC707
  - AC701

Simulation Debug

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

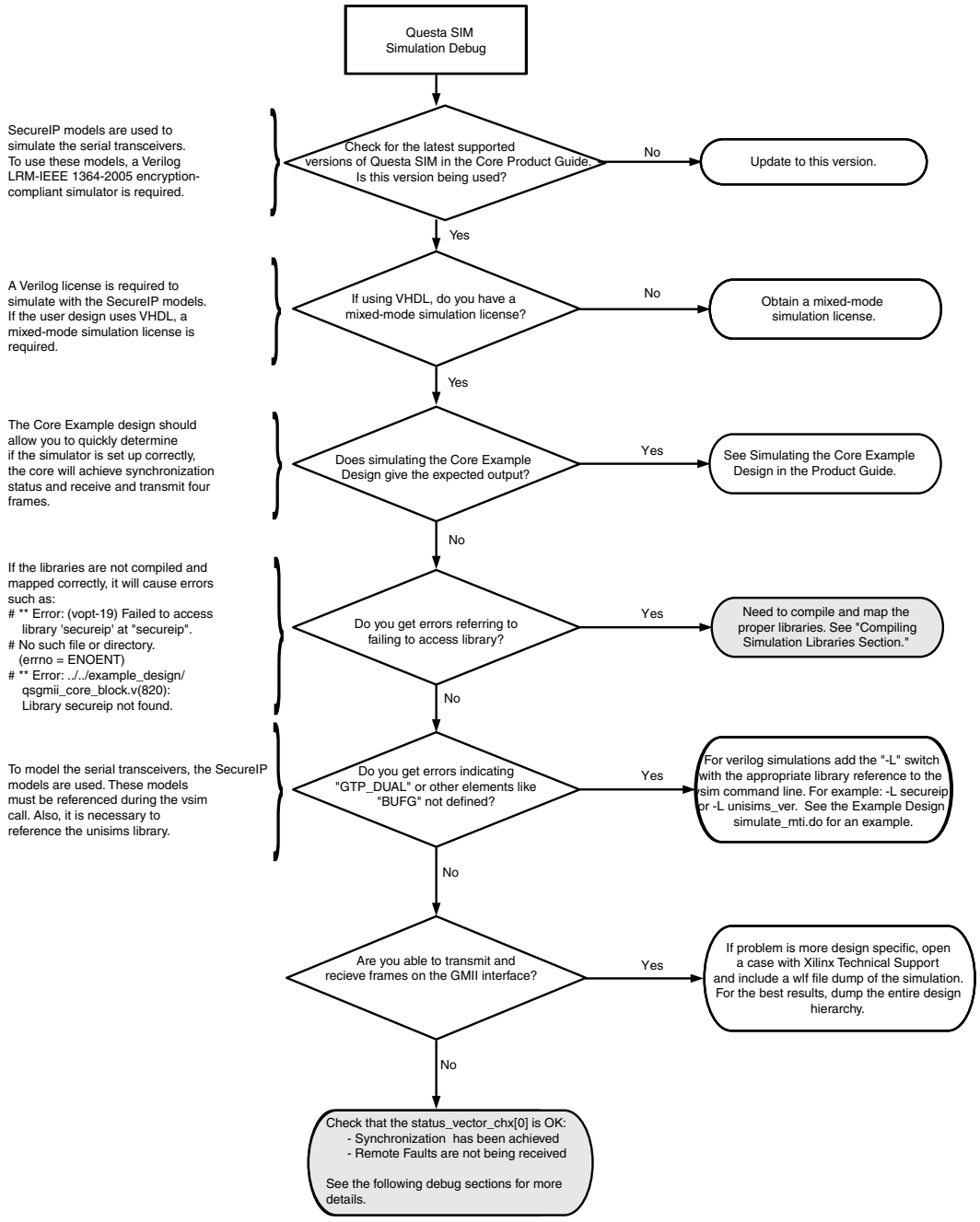


Figure D-1: **Simulation Debug Flow**

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The Vivado lab tools are a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the Vivado lab tools for debugging the specific problems.

Many of these common issues can also be applied to debugging design simulations. Details are provided on:

- Problems with the MDIO
- Problems with Data Reception or Transmission
- Problems with Auto-Negotiation
- Problems in Obtaining a Link (Auto-Negotiation Disabled)
- Problems with a High Bit Error Rate

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 Vivado lab tools 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.

References

Unless otherwise noted, IP references are for the product documentation page. To search for Xilinx documentation, go to www.xilinx.com/support

1. *Cisco QSGMII Specification Version 1.2* (EDCS-540123)
2. *Ethernet Standard 802.3-2008 Clauses 35, 36 and 38* (Part 3)
3. Cisco Serial GMII Specification Revision 1.8 (SGMII) (ftp://ftp-eng.cisco.com/smii/sgmii.pdf)
4. *Cisco Quad SGMII Specification Revision 1.2 (QSGMII)* (EDCS-540123)
5. *LogiCORE IP Ethernet 1000BASE-X PCS/PMA or SGMII Product Guide* (PG047)
6. *7 Series FPGAs GTX/GTH Transceivers User Guide* (UG476)
7. *7 Series FPGAs GTP Transceivers User Guide* (UG482)
8. *Vivado Design Suite User Guide: Designing with IP* (UG896)
9. *Vivado Design Suite User Guide - Logic Simulation* (UG900)
10. *Vivado Design Suite User Guide: Getting Started* (UG910)
11. *ISE to Vivado Design Suite Migration Guide* (UG911)

12. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
13. *Tri-Mode Ethernet MAC Product Guide* (PG051)
14. *XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices* (UG687)
15. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)

Specifications

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

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/18/2012	1.0	Initial Xilinx release
04/24/2012	1.2	Added support for Virtex®-7 FPGA GTH Transceiver.
07/25/2012	1.3	Added Vivado tools support. Added support for Artix®-7 devices.
10/16/2012	1.4	Updated for 14.3 and 2012.3 support Added support for Zynq®-7000 devices
03/20/2013	2.0	Revision number advanced to N.N to align with core version number. Removed ISE-related information. Updated resource numbers. Updated Vivado IDEs
10/02/2013	3.0	Removed static MDIO PHY Address ports and made programmable while generation through Vivado IDE. Removed Link Timer ports and tied to 1.64 ms for synthesis and 0.14 ms for simulation. Updated for 7 series transceivers (Termination settings updates, attribute updates, hierarchy update). Enhanced support for IP Integrator. Reduced warnings in synthesis and simulation. Updated clock synchronizers to improve Mean Time Between Failures (MTBF) for metastability. Added support for the out-of-context flow. Added Vivado IDE option to include or exclude shareable logic resources in the core. Added Vivado IDE option to include or exclude configuration vector ports. Added optional transceiver control and status ports.
12/18/2013	3.1	Added UltraScale™ architecture support. Updated serial transceivers for 7 series devices (RX/TX Startup FSM updates.) Increased the number of optional transceiver control and status ports Updated screen captures in Chapter 6. Changed all signal and port names in figures to all lowercase.

Notice of Disclaimer