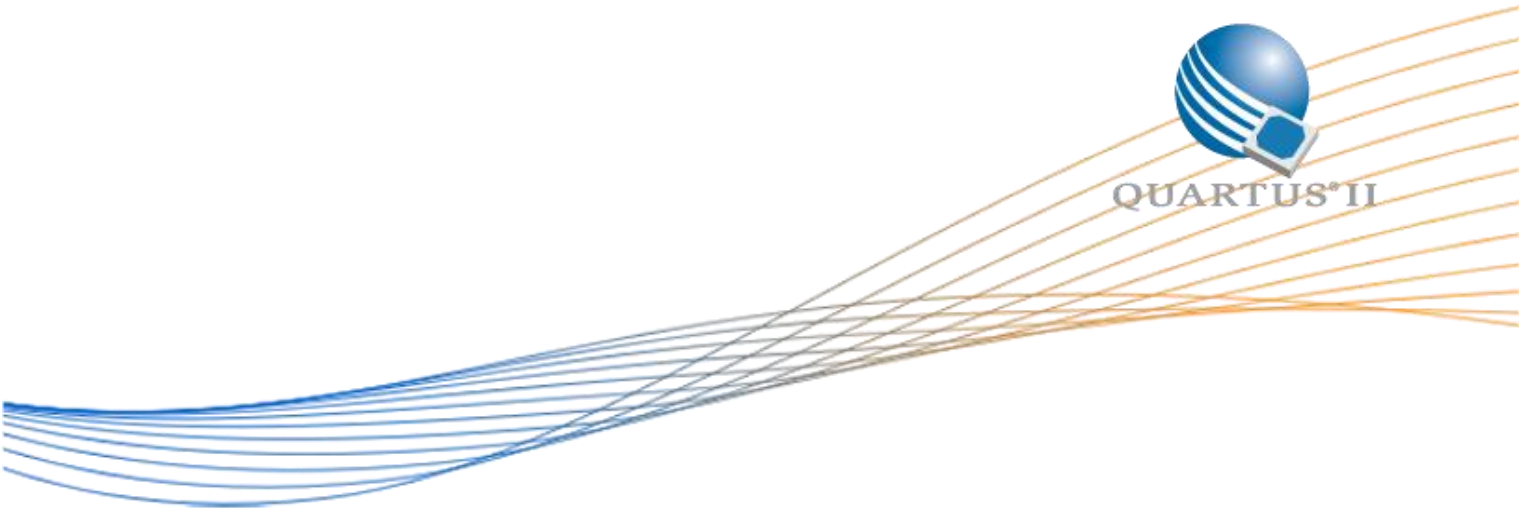


MAX 10 Single-Port Triple Speed Ethernet and On-Board PHY Chip Design Example User Guide

Date: Dec 2015

Revision: 1.0

Contents

Introduction 3

System Architecture 4

Design Components 5

Ethernet Packet Generator 7

Ethernet Packet Monitor 9

Base Addresses and Configuration Registers 10

Ethernet Packet Generator Configuration Registers 10

Ethernet Packet Monitor Configuration Registers 12

Interface Signals 13

Clock and Reset Signals 13

Triple-Speed Ethernet RGMII Interface Signals 13

RGMII Timing Constraints 14

At the FPGA Transmit Side 15

At the FPGA Receive Side 17

Using the Design Example 20

Hardware and Software Requirements 20

Setting Up the Development Board 20

Testing the Design Example 20

Tcl Script 22

Configuration Script 22

Ethernet Packet Generator Script 22

Document Revision History 23

Introduction

This design example demonstrates Triple Speed Ethernet IP solution for MAX 10® device family using Altera® Triple Speed Ethernet MAC and Marvell 88E1111 PHY chip on MAX 10 FPGA Development Kit. It provides flexible test and demonstration platforms on which you can control, test, and monitor the Ethernet operations using system loopbacks. In this design, the Single-Port Triple-Speed Ethernet MAC connects to the on-board PHY chip through the Reduce Gigabit Media Independent Interface (RGMII).

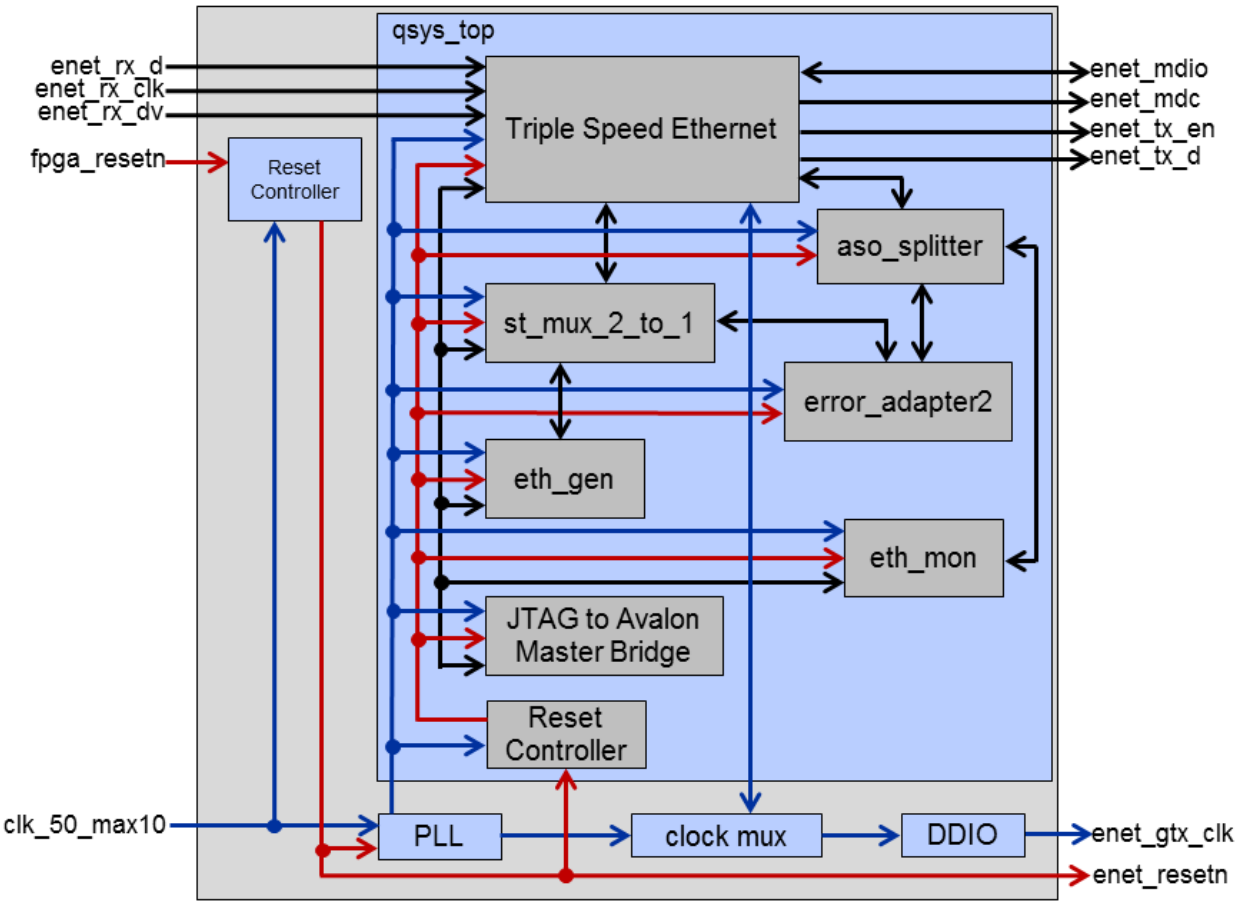
The reference design offers the following features:

- Supports programmable test parameters such as number of packets, packet length, source and destination MAC addresses, and payload-data type.
- Supports sequential random burst test, which enables the configuration of each burst for the number of packet, payload-data type, and payload size. A pseudo-random binary sequence (PRBS) generator generates the payload data type in fixed incremental values or in a random sequence.
- Demonstrates transmission and reception of Ethernet packets through internal loopback path at the maximum theoretical data rates without errors.
- Supports System Console user interface. This Tcl-based user interface allows you to dynamically configure, debug, and test the reference designs.

System Architecture

The design example demonstrates fully operational subsystems that integrate the Triple-Speed Ethernet IP Core for Ethernet applications. Figure 1 shows a top-level block diagram of the reference design implementing the RGMII interface.

Figure 1: Design Example Simplified Block Diagram



Design Components

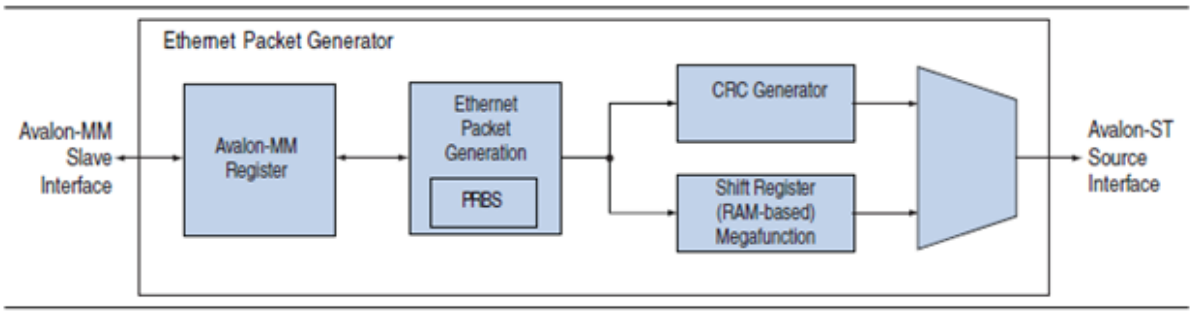
Table 1: Design example components.

Component	Description
Triple Speed Ethernet	**Triple-Speed Ethernet MegaCore Function** This IP core is configured for 10/100/1000 Ethernet MAC function to handle the flow of data between user applications and Ethernet network through an external Ethernet PHY.
st_mux_2_to_1	**Avalon-ST Multiplexer** This Qsys custom component accepts data on its two Avalon-ST sink interfaces and multiplexes the data for transmission on its Avalon-ST source interface. One Avalon-ST sink interface connects to the source of the Ethernet Packet Generator for forward loopback while the other sink interface connects to the source of the Error Adapter for reverse loopback. The Avalon-ST source interface sends Ethernet packets to the Triple-Speed Ethernet IP Core.
aso_splitter	**Avalon-ST Splitter** This Qsys custom component accepts data on its Avalon-ST sink interface and splits the data for transmission on its two Avalon-ST source interfaces. The Avalon-ST sink interface receives Ethernet packets from the Triple-Speed Ethernet IP Core. One Avalon-ST source interface connects to the sink of the Ethernet Packet Generator for forward loopback while the other source interface connects to the sink of the Error Adapter for reverse loopback.
error_adapter2	**Error Adapter** This Qsys custom component connects mismatched Avalon-ST source and sink interfaces. The adapter allows you to connect a data source to a data sink of differing byte sizes. For TX-to-RX Avalon-ST reverse loopback in these design examples, ff_tx_err is a 1-bit error signal while rx_err is a 6-bit error signal. The adapter ensures that the per-bit error information provided by ff_tx_err at the source interface connects correctly to the rx_err signal. The adapter connects matching error conditions that are handled by the source and the sink. For more information about the ff_tx_err and rx_err error signals, refer to the Triple Speed Ethernet User Guide.

Component	Description
eth_gen	**Ethernet Packet Generator** This Qsys custom component generates Ethernet packets. Figure 3 shows a high-level block diagram of the Ethernet Packet Generator module. This module contains the following components: Avalon-MM registers, Ethernet packet generation block, CRC Generator, and Shift Register (RAM-based) megafunction.
eth_mon	**Ethernet Packet Monitor** This Qsys custom component verifies the payload of all receive packets, indicates the validity of the packets, and collects statistics about each packet, such as the number of bytes received. This module contains the following components: CRC Checker and Avalon-MM registers.
JTAG to Avalon Master Bridge	**JTAG to Avalon Master Bridge Core** This IP core provides a connection between the System Console and Qsys system through the physical interfaces. The System Console can initiate Avalon Memory-Mapped (Avalon-MM) transactions by sending encoded streams of bytes through the bridge's physical interfaces. For more information about the JTAG to Avalon Master Bridge Core, refer to the SPI Slave/JTAG to Avalon Master Bridge Cores chapter in the Embedded Peripherals IP User Guide.
Reset Controller	**Reset Controller** This module is used to synchronize and generate signals as per design requirements.
PLL	**Phase-Locked Loop (PLL) Core** This IP core takes an input clock from a 50-MHz crystal on the development board and generates a 125MHz, 25MHz and 2.5MHz PLL output clock (clk_125M). The 125MHz output clock is the system-wide clock source for the Qsys system. All the components in the reference designs use the 125-MHz clock from the PLL core.
clock mux	**Clock Multiplexer** This module is driven by the clock selector signals from Triple Speed Ethernet IP Core to select the 125MHz/25MHz/2.5MHz transmit reference clock for RGMII
DDIO	**Double data rate input/output** This digital component doubles the data-rate of a communication channel. This module is inserted to minimize the clock skew.

Ethernet Packet Generator

Figure 2: Ethernet Packet Generator Block Diagram



The Avalon-MM slave interface provides access to the Avalon-MM register interface. Using a Tcl script, you can configure the Avalon-MM configuration registers to specify the following:

- Number of packets to generate.
- MAC source and destination addresses—can be unicast, multicast, or broadcast addresses. If you use unicast addresses, set the source address to the address of the transmitting MAC and the destination address to the address of the receiving MAC. This setting ensures that the receiving Ethernet port in the reference designs receives valid packets.
- Packet length—you can specify a fixed or random packet length. A fixed length ranges between 24 to 9,600 bytes; a random length ranges between 24 and 1,518. The Triple-Speed Ethernet IP core pads undersized packets to meet the minimum required length, 64 bytes.
- Payload—you can specify the data type to be incremental or pseudo-random. Incremental data starts from zero and gets incremented by one in subsequent packets. The PRBS block generates the pseudo-random data.
- Random seed—specify the random seed used by the PRBS block to generate the pseudo-random data.

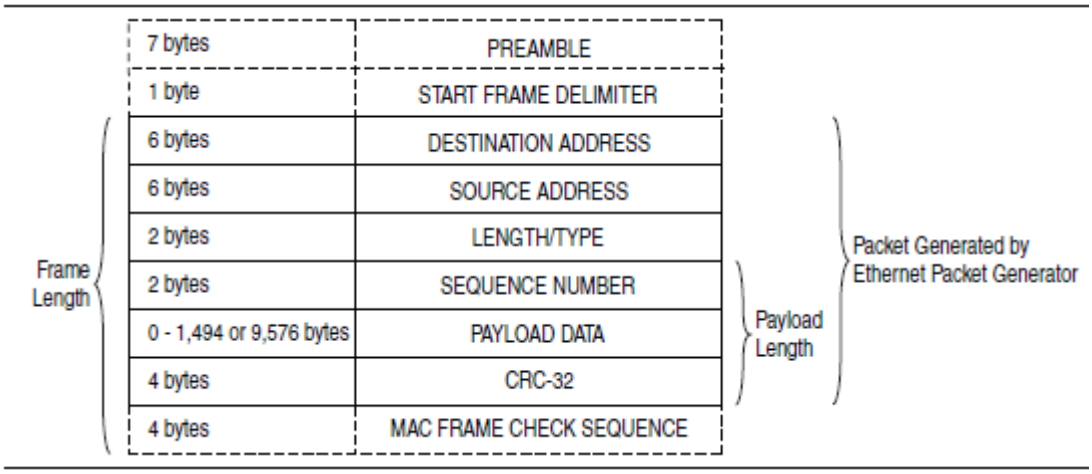
The Avalon-MM status registers provide the status of the transmit operation and report the number of packets that were successfully transmitted.

The Ethernet packet generation block generates an Ethernet packet header, data payload and running sequence number for each packet. The Ethernet packet generation block sends the packets to the CRC Generator and the RAM-based shift register megafunction.

The CRC Generator calculates the CRC-32 checksum for the packet and the RAM-based shift register megafunction stores the packet until the checksum is available. After the generator merges the valid CRC-32 checksum with the packet stream, it sends the complete packet to the Avalon-ST source interface.

The Avalon-ST source interface streams Ethernet packets in the format shown in Figure 3. The generated packets do not include the 7-byte preamble, 1-byte start frame delimiter (SFD) and 4-byte MAC-calculated Frame Check Sequence (FCS) fields.

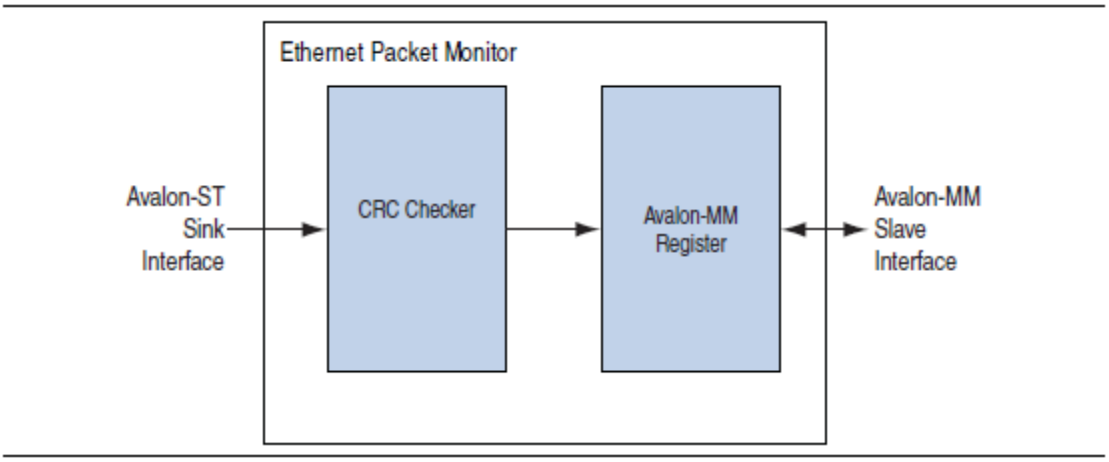
Figure 3: Ethernet Packet Generator Output Frame Format



Ethernet Packet Monitor

This Qsys custom component verifies the payload of all receive packets, indicates the validity of the packets, and collects statistics about each packet, such as the number of bytes received. Figure 4 shows a high-level block diagram of the Ethernet Packet Monitor module. This module contains the following components: CRC Checker and Avalon-MM registers.

Figure 4: Ethernet Monitor Block Diagram



The Avalon-ST sink interface accepts Ethernet packets and sends the packets to the CRC Checker.
The CRC Checker computes the CRC-32 checksum of the receive packet and verifies it against the CRC-32 checksum field in the packet. It then outputs a status signal that identifies whether the packet received is good or corrupted, and updates the statistics counters accordingly.
The Avalon-MM slave interface provides access to the Avalon-MM register interface. Using a Tcl script, you can configure the Avalon-MM configuration registers to specify the number of packets the monitor expects to receive. The Avalon-MM status registers provide the status of the receive operation and report the number of good and bad packets received, the number of bytes received, and the number of clock cycles. This information is used to calculate the performance and throughput rate of the reference designs. For information about the Ethernet Packet Monitor registers, refer to“Ethernet Packet Monitor Configuration Registers” on page 12.

Base Addresses and Configuration Registers

Table 2 lists the base address for each component in the reference designs. To access the configuration registers of these components, use the base address of the component and the register offset.

Table 2: Base Address of Reference Design Components

Base Address	Name	Description
0x00000000	triple_speed_ethernet_0	Triple-Speed Ethernet
0x00000400	st_mux_2_to_1_0	Avalon-ST Multiplexer
0x00000800	eth_mon_0	Ethernet Packet Monitor
0x00000C00	eth_gen_0	Ethernet Packet Generator

Ethernet Packet Generator Configuration Registers

Table 3 describes the configuration registers of the Ethernet Packet Generator

Table 3: Ethernet Packet Generator Configuration Registers

Byte Offset	Name	Bits	Bit Name	R/W	H/W Reset	Description
0x00	number_packet	31:0	-	RW	0x0	Specifies the total number of packets to be generated.
0x04	config_setting	0	LENGTH_SEL	RW	0x0	0: Fixed packet length 1: Random packet length
		14:1	PKT_LENGTH	RW	0x0	Specifies the fixed packet length. Valid values are between 24 to 9,600. Applicable only when bit 0 of this register is set to 0.
		15	PATTERN_SEL	RW	0x0	Specifies the data pattern for the random packet length. 0: Incremental. Data starts from zero and is incremented by 1 in subsequent bytes. 1: Random.
		31:16	-	-	-	Reserved.
0x08	operation	0	START	RW	0x0	Set this bit to 1 to trigger packet generation. This bit clears as soon as packet generation starts.
		1	STOP	RW	0x0	Set this bit to 1 to stop packet generation. The generator completes the current packet before terminating packet generation.
		2	TX_DONE	RO	0x0	A value of 1 indicates that the packet generator completes generating the total number of packets specified in the number_packet register. This bit clears each time packet generation is triggered.
		31:3	-	-	-	Reserved.

Byte Offset	Name	Bits	Bit Name	R/W	H/W Reset	Description
0x10	source_addr0	31:0	-	RW	0x0	6-byte MAC address. • source/destination_addr0 = Last four bytes of the address • Bits 0 to 15 of source/destination_addr1 = First two bytes of the address • Bits 16 to 31 of source/destination_addr1 are unused. For example, if the source MAC address is 00-1C-23-17-4A-CB, the following assignments are made: source_addr0 = 0x17231C00 source_addr1 = 0x0000CB4A
0x14	source addr1	31:0	-	RW	0x0	
0x18	destination_addr0	31:0	-	RW	0x0	
0x1C	destination_addr1	31:0	-	RW	0x0	
0x24	packet_tx_count	31:0	-	-	-	Keeps track of the number of packets the generator successfully transmits. This register clears each time packet generation is triggered.
0x30	rand_seed0	31:0	-	RW	0x0	The lower 32 bits of the random seed. Occupies bits 31:0 of the PRBS generator when the data pattern is set to random (bit 15 of the configuration register).
0x34	rand_seed1	31:0	-	RW	0x0	The middle 32 bits of the random seed. Occupies bits 63:32 of the PRBS generator when the data pattern is set to random (bit 15 of the configuration register).
0x38	rand_seed2	31:0	-	RW	0x0	The upper 32 bits of the random seed. Occupies bits 91:64 of the PRBS generator when the data pattern is set to random (bit 15 of the configuration register).

Ethernet Packet Monitor Configuration Registers

Table 4 describes the configuration registers of the Ethernet Packet Monitor.

Table 4: Ethernet Packet Monitor Configuration Registers

Byte Offset	Name	Bits	Bit Name	R/W	H/W Reset	Description
0x00	number_packet	31:0	-	RO	0x0	Total number of packets the monitor is expected to receive.
0x04	packet_rx_ok	31:0	-	RO	0x0	Total number of good packets received.
0x08	packet_rx_error	31:0	-	RO	0x0	Total number of packets received with error.
0x0C	byte_rx_count_0	31:0	-	RO	0x0	64-bit counter that keeps track of the total number of bytes received. The byte_rx_count_0 register represents the lower 32 bits, byte_rx_count_1 represents the upper 32 bits. Read byte_rx_count_0 followed by byte_rx_count_1 in the subsequent cycle to get an accurate count.
0x10	byte_rx_count_1	31:0	-	RO	0x0	
0x14	cycle_rx_count_0	31:0	-	RO	0x0	64-bit counter that keeps track of the total number of cycles the monitor takes to receive all packets. The cycle_rx_count_0 register represents the lower 32 bits; cycle_rx_count_1 represents the upper 32 bits. Read byte_rx_count_0 followed by byte_rx_count_1 in the subsequent cycle to get an accurate count.
0x18	cycle_rx_count_1	31:0	-	RO	0x0	
0x1C	rx_control_status	0	START	RW	0x0	Set this bit to 1 to start packet reception. This bit clears when packet reception starts.
		1	STOP	RW	0x0	Set this bit to 1 to stop packet reception. This bit clears each time packet reception starts.
		2	RX_DONE	RO	0x0	A value of 1 indicates that the packet monitor has received the total number of packets specified in the number_packet register.
		3	CRCBAD	RO	0x0	A value of 1 indicates CRC error in the current packet received by the monitor.
		9:4	RX_ERR	RO	0x0	Receive error status. The rx_err[] signal of the Triple-Speed Ethernet IP core is mapped to this register.
		31:10	-	-	-	Reserved

Interface Signals

This section describes the top-level signals of the reference designs.

Clock and Reset Signals

Table 5: Top-level clock and reset signals

Name	I/O	Description
clk_50_max10	I	50MHz reference clock for PLL.
enet_rx_clk	I	125MHz, 25MHz or 2.5MHz RGMII receive clock derived from the received data stream and based on the selected speed. The clock is sourced from the on-board PHY chip.
enet_gtx_clk	O	125MHz, 25MHz or 2.5MHz RGMII transmit clock based on the selected speed. The clock is sourced from the PLL.
fpga_resetn	I	Reset signal for all logic in the design example. Connected to the User-Defined Push Button (USER_PB0).
enet_resetn	O	Active low hardware reset signal for on-board PHY chip

Triple-Speed Ethernet RGMII Interface Signals

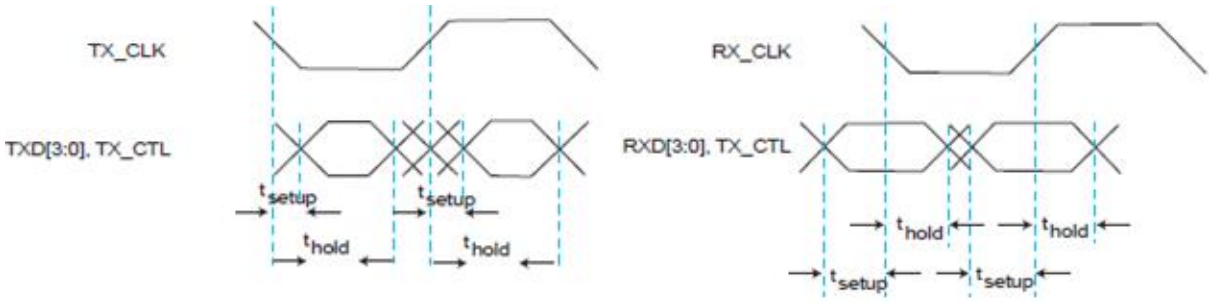
Table 6: Top-level RGMII signals for the RGMII interface

Name	I/O	Description
enet_rx_d[3:0]	I	RGMII receive data bus. Connected to the on-board PHY chip.
enet_rx_dv	I	RGMII receive control signal. Connected to the on-board PHY chip.
enet_tx_d[3:0]	O	RGMII transmit data bus. Connected to the on-board PHY chip.
enet_tx_en	O	RGMII transmit control signal. Connected to the on-board PHY chip.

RGMII Timing Constraints

The on-board PHY chip supports internal delay. It is enabled in this design example, so no additional logic is needed at the FPGA side for clock-data alignment.
The FPGA transmits edge-aligned data and receives center-aligned data.
The transmit clock can be driven by the same clock that clocks the transmit data; the receive clock, RX_CLK, can be used directly by the FPGA to capture the data.

Figure 5: Timing waveform when the delay option is enabled at the on-board PHY



On-board PHY I/O timing requirements are as follows:

- Minimum input setup time = -0.9 ns
- Minimum input hold time = 2.7 ns
- Minimum output setup time = 1.2 ns
- Minimum output hold time = 1.2 ns

I/O delay calculation (assume board trace, pin capacitance, and rise/fall time differences between the data and clock are negligible):

Output maximum delay = maximum trace delay for data - minimum trace delay for clock + t_{SU} of external register
= - 0.9ns

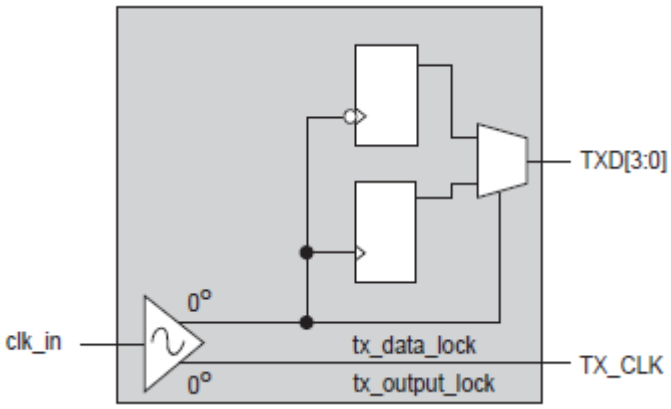
Output minimum delay = minimum trace delay for data - maximum trace delay for clock - t_H of external register
= -2.7ns

Input maximum delay = maximum trace delay for data - minimum trace delay for clock - t_{SU} of external register + unit interval
= -1.2 + 2
= 0.8ns

Input minimum delay = minimum trace delay for data - maximum trace delay for clock + t_H of external register - unit interval
= 1.2 - 2
= -0.8ns

At the FPGA Transmit Side

Figure 6: RGMII Transmit Implementation when the Delay Option is enabled




```
#Create PLL reference clock
create_clock -name {clk_50_max10} -period "50 MHz" [get_ports {clk_50_max10}]
#PLL clocks for triple speed Ethernet
create_generated_clock -name enet_tx_125 -source [get_pins
{i_pll|altpll_component|auto_generated|pll1|inclk[0]}] -duty_cycle 50.000 -multiply_by 5 -divide_by 2 -
master_clock {clk_50_max10} [get_pins {i_pll|altpll_component|auto_generated|pll1|clk[0]}]
##Shifted 125MHz clock for timing closure in MAX10 device
create_generated_clock -name enet_tx_125_shift -source [get_pins
{i_pll|altpll_component|auto_generated|pll1|inclk[0]}] -duty_cycle 50.000 -multiply_by 5 -divide_by 2 -
phase 11.25 -master_clock {clk_50_max10} [get_pins
{i_pll|altpll_component|auto_generated|pll1|clk[3]}]
create_generated_clock -name enet_tx_25 -source [get_pins
{i_pll|altpll_component|auto_generated|pll1|inclk[0]}] -duty_cycle 50.000 -multiply_by 1 -divide_by 2 -
master_clock {clk_50_max10} [get_pins {i_pll|altpll_component|auto_generated|pll1|clk[1]}]
create_generated_clock -name enet_tx_2p5 -source [get_pins
{i_pll|altpll_component|auto_generated|pll1|inclk[0]}] -duty_cycle 50.000 -multiply_by 1 -divide_by 20 -
master_clock {clk_50_max10} [get_pins {i_pll|altpll_component|auto_generated|pll1|clk[2]}]
#GTX clocks to PHY
create_generated_clock -name gtx_125_clk -source [get_pins {gtx_clk125_shift~1clkctrl|outclk}] -
master_clock {enet_tx_125_shift} [get_ports {enet_gtx_clk}]
create_generated_clock -name gtx_25_clk -source [get_pins
{clkctrl_inst0|altclkctrl_0|clkctrl_altclkctrl_0_sub_component|clkctrl1|outclk}] -master_clock
{enet_tx_25} [get_ports {enet_gtx_clk}] -add
create_generated_clock -name gtx_2p5_clk -source [get_pins
{clkctrl_inst0|altclkctrl_0|clkctrl_altclkctrl_0_sub_component|clkctrl1|outclk}] -master_clock
{enet_tx_2p5} [get_ports {enet_gtx_clk}] -add
derive_clock_uncertainty
```

```
# RGMII TX channel
# Output Delay Constraints (Edge Aligned, Same Edge Capture)
set output_max_delay -0.9
set output_min_delay -2.7
set_output_delay -clock gtx_125_clk -max $output_max_delay [get_ports "enet_tx_d* enet_tx_en"]
set_output_delay -clock gtx_125_clk -max $output_max_delay [get_ports "enet_tx_d* enet_tx_en"] -clock_fall -add_delay
set_output_delay -clock gtx_125_clk -min $output_min_delay [get_ports "enet_tx_d* enet_tx_en"] -add_delay
set_output_delay -clock gtx_125_clk -min $output_min_delay [get_ports "enet_tx_d* enet_tx_en"] -clock_fall -add_delay

set_output_delay -clock gtx_25_clk -max $output_max_delay [get_ports "enet_tx_d* enet_tx_en"] -add_delay
set_output_delay -clock gtx_25_clk -max $output_max_delay [get_ports "enet_tx_d* enet_tx_en"] -clock_fall -add_delay
set_output_delay -clock gtx_25_clk -min $output_min_delay [get_ports "enet_tx_d* enet_tx_en"] -add_delay
set_output_delay -clock gtx_25_clk -min $output_min_delay [get_ports "enet_tx_d* enet_tx_en"] -clock_fall -add_delay

set_output_delay -clock gtx_2p5_clk -max $output_max_delay [get_ports "enet_tx_d* enet_tx_en"] -add_delay
set_output_delay -clock gtx_2p5_clk -max $output_max_delay [get_ports "enet_tx_d* enet_tx_en"] -clock_fall -add_delay
set_output_delay -clock gtx_2p5_clk -min $output_min_delay [get_ports "enet_tx_d* enet_tx_en"] -add_delay
set_output_delay -clock gtx_2p5_clk -min $output_min_delay [get_ports "enet_tx_d* enet_tx_en"] -clock_fall -add_delay

# Set multicycle paths to align the launch edge with the latch edge
# Not needed for 125MHz GTX clock as the clock was shifted
set_multicycle_path -setup -end -rise_from [get_clocks enet_tx_25] -rise_to [get_clocks gtx_25_clk] 0
set_multicycle_path -setup -end -fall_from [get_clocks enet_tx_25] -fall_to [get_clocks gtx_25_clk] 0
set_multicycle_path -hold -end -rise_from [get_clocks enet_tx_25] -rise_to [get_clocks gtx_25_clk] -1
set_multicycle_path -hold -end -fall_from [get_clocks enet_tx_25] -fall_to [get_clocks gtx_25_clk] -1

set_multicycle_path -setup -end -rise_from [get_clocks enet_tx_2p5] -rise_to [get_clocks gtx_2p5_clk] 0
set_multicycle_path -setup -end -fall_from [get_clocks enet_tx_2p5] -fall_to [get_clocks gtx_2p5_clk] 0
set_multicycle_path -hold -end -rise_from [get_clocks enet_tx_2p5] -rise_to [get_clocks gtx_2p5_clk] -1
set_multicycle_path -hold -end -fall_from [get_clocks enet_tx_2p5] -fall_to [get_clocks gtx_2p5_clk] -1
```

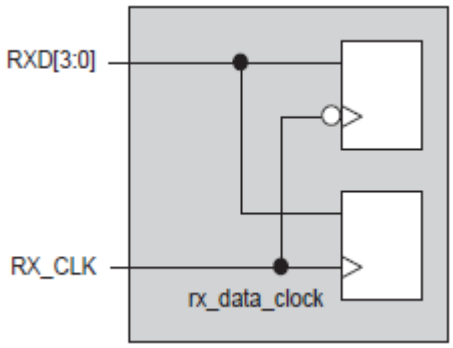
```
# Set false paths to remove irrelevant setup and hold analysis
set_false_path -fall_from [get_clocks enet_tx_125] -rise_to [get_clocks gtx_125_clk] -setup
set_false_path -rise_from [get_clocks enet_tx_125] -fall_to [get_clocks gtx_125_clk] -setup
set_false_path -fall_from [get_clocks enet_tx_125] -fall_to [get_clocks gtx_125_clk] -hold
set_false_path -rise_from [get_clocks enet_tx_125] -rise_to [get_clocks gtx_125_clk] -hold

set_false_path -fall_from [get_clocks enet_tx_25] -rise_to [get_clocks gtx_25_clk] -setup
set_false_path -rise_from [get_clocks enet_tx_25] -fall_to [get_clocks gtx_25_clk] -setup
set_false_path -fall_from [get_clocks enet_tx_25] -fall_to [get_clocks gtx_25_clk] -hold
set_false_path -rise_from [get_clocks enet_tx_25] -rise_to [get_clocks gtx_25_clk] -hold

set_false_path -fall_from [get_clocks enet_tx_2p5] -rise_to [get_clocks gtx_2p5_clk] -setup
set_false_path -rise_from [get_clocks enet_tx_2p5] -fall_to [get_clocks gtx_2p5_clk] -setup
set_false_path -fall_from [get_clocks enet_tx_2p5] -fall_to [get_clocks gtx_2p5_clk] -hold
set_false_path -rise_from [get_clocks enet_tx_2p5] -rise_to [get_clocks gtx_2p5_clk] -hold
```

At the FPGA Receive Side

Figure 7: RGMII Receive Implementation When the Delay Option is enabled




```
# RGMII RX channel
#Virtual clocks for input constraints
create_clock -name virtual_clk_125 -period 8
create_clock -name virtual_clk_25 -period 40
create_clock -name virtual_clk_2p5 -period 400
#Define RX clocks from PHY
create_clock -period 8 -waveform { 2 6 } -name  {rx_clk_125} [get_ports enet_rx_clk]
create_clock -period 40 -waveform { 10 30 } -name  {rx_clk_25} [get_ports enet_rx_clk] -add
create_clock -period 400 -waveform { 100 300 } -name  {rx_clk_2p5} [get_ports enet_rx_clk] -add
```

```
# Input Delay Constraints (Center aligned, Same Edge Analysis)
# 125MHz input constraints
set input_max_delay_125 0.8
set input_min_delay_125 -0.8
set_input_delay -max [expr $input_max_delay_125 ] -clock [get_clocks virtual_clk_125] [get_ports "enet_rx_d* enet_rx_dv"]
set_input_delay -max [expr $input_max_delay_125 ] -clock [get_clocks virtual_clk_125] [get_ports "enet_rx_d* enet_rx_dv"] -clock_fall -add_delay
set_input_delay -min [expr $input_min_delay_125 ] -clock [get_clocks virtual_clk_125] [get_ports "enet_rx_d* enet_rx_dv"] -add_delay
set_input_delay -min [expr $input_min_delay_125 ] -clock [get_clocks virtual_clk_125] [get_ports "enet_rx_d* enet_rx_dv"] -clock_fall -add_delay

# Set false paths to remove irrelevant setup and hold analysis
set_false_path -fall_from [get_clocks virtual_clk_125] -rise_to [get_clocks {rx_clk_125}] -setup
set_false_path -rise_from [get_clocks virtual_clk_125] -fall_to [get_clocks {rx_clk_125}] -setup
set_false_path -fall_from [get_clocks virtual_clk_125] -fall_to [get_clocks {rx_clk_125}] -hold
set_false_path -rise_from [get_clocks virtual_clk_125] -rise_to [get_clocks {rx_clk_125}] -hold

# 25MHz input constraints
set input_max_delay_25 8.8
set input_min_delay_25 -8.8
set_input_delay -max [expr $input_max_delay_25 ] -clock [get_clocks virtual_clk_25] [get_ports "enet_rx_d* enet_rx_dv"] -add_delay
set_input_delay -max [expr $input_max_delay_25 ] -clock [get_clocks virtual_clk_25] [get_ports "enet_rx_d* enet_rx_dv"] -clock_fall -add_delay
set_input_delay -min [expr $input_min_delay_25 ] -clock [get_clocks virtual_clk_25] [get_ports "enet_rx_d* enet_rx_dv"] -add_delay
set_input_delay -min [expr $input_min_delay_25 ] -clock [get_clocks virtual_clk_25] [get_ports "enet_rx_d* enet_rx_dv"] -clock_fall -add_delay

# Set false paths to remove irrelevant setup and hold analysis
set_false_path -fall_from [get_clocks virtual_clk_25] -rise_to [get_clocks {rx_clk_25}] -setup
set_false_path -rise_from [get_clocks virtual_clk_25] -fall_to [get_clocks {rx_clk_25}] -setup
set_false_path -fall_from [get_clocks virtual_clk_25] -fall_to [get_clocks {rx_clk_25}] -hold
set_false_path -rise_from [get_clocks virtual_clk_25] -rise_to [get_clocks {rx_clk_25}] -hold

# 2.5MHz input constraints
set input_max_delay_2p5 98.8
set input_min_delay_2p5 -98.8
set_input_delay -max [expr $input_max_delay_2p5 ] -clock [get_clocks virtual_clk_2p5] [get_ports "enet_rx_d* enet_rx_dv"] -add_delay
set_input_delay -max [expr $input_max_delay_2p5 ] -clock [get_clocks virtual_clk_2p5] [get_ports "enet_rx_d* enet_rx_dv"] -clock_fall -add_delay
set_input_delay -min [expr $input_min_delay_2p5 ] -clock [get_clocks virtual_clk_2p5] [get_ports "enet_rx_d* enet_rx_dv"] -add_delay
set_input_delay -min [expr $input_min_delay_2p5 ] -clock [get_clocks virtual_clk_2p5] [get_ports "enet_rx_d* enet_rx_dv"] -clock_fall -add_delay
```

```
# Set false paths to remove irrelevant setup and hold analysis
set_false_path -fall_from [get_clocks virtual_clk_2p5] -rise_to [get_clocks {rx_clk_2p5}] -setup
set_false_path -rise_from [get_clocks virtual_clk_2p5] -fall_to [get_clocks {rx_clk_2p5}] -setup
set_false_path -fall_from [get_clocks virtual_clk_2p5] -fall_to [get_clocks {rx_clk_2p5}] -hold
set_false_path -rise_from [get_clocks virtual_clk_2p5] -rise_to [get_clocks {rx_clk_2p5}] -hold

#Remove irrelevant clock domain crossing analysis
set_clock_groups -exclusive -group {enet_tx_125 gtx_125_clk virtual_clk_125 rx_clk_125} -group
{enet_tx_25 gtx_25_clk virtual_clk_25 rx_clk_25} -group {enet_tx_2p5 gtx_2p5_clk virtual_clk_2p5
rx_clk_2p5}
```

Using the Design Example

This section describes the required hardware and software setup.

Hardware and Software Requirements

To run the reference designs, you need the following hardware:

- MAX 10 FPGA Development Kit
- Quartus Prime v15.1
- External Ethernet Packet Generator (only for Avalon-ST reverse loopback test)

Setting Up the Development Board

For internal loopback test:

1. Connect the development board to computer using programming cable thru JTAG connection port or USB cable thru Type-B USB connector (Embedded USB-Blaster II).
2. Connect the power supply cord to the power supply input.

For Avalon-ST reverse loopback test:

1. The Avalon-ST reverse loopback requires an external Ethernet packet generator. Using the Ethernet cable assembly, connect the external generator to the RJ-45 port, Ethernet Port A (Bottom) of the FPGA development board.
3. Connect the development board to computer using programming cable thru JTAG connection port or USB cable thru Type-B USB connector (Embedded USB-Blaster II).
4. Connect the power supply cord to the power supply input.

Testing the Design Example

To perform hardware test, follow these steps:

1. Set up or restore the design example.
2. Launch the Quartus Prime software and open the project file (top.qpf).
3. Click **Start Compilation** on the Processing menu to compile the design example.
4. Configure the FPGA on MAX 10 FPGA Development Board using the generated configuration file (top.sof).
5. Press **FPGA_RESETN** push button to reset the system.
   Note: System should be hard reset after configuration done.
6. On Quartus Prime Tools menu, click on **System Debugging Tools** and then launch **System Console**.
7. In the **System Console** command shell, change the directory to “sc_tcl” directory.

8. Run the command “source main.tcl” to initialize the design example command list.
9. Perform the following test by running the command in the **System Console** command shell:

   a. To Perform Internal MAC Loopback Test (local loopback on the RGMII of the MAC function to exercise the transmit and receive paths) for 10M/100M/1000M speed:

   Command: TEST_MAC_LB {speed_test}
   Example: TEST_MAC_LB 1000M

   - The System Console displays the copper link connection status and the PHY’s operating speed and mode.
   - Run “stats_chk” command to view the MAC statistic counters.

   b. To Perform External MAC Loopback Test (data sent from Triple Speed Ethernet MAC **to** on-board PHY chip is not transmitted out on the media interface. Instead, the data is looped back and sent to the MAC) for 10M/100M/1000M speed:

   Command: TEST_MAC_PHY_LB {speed_test}
   Example: TEST_MAC_PHY_LB 1000M

   - The System Console displays the copper link connection status and the PHY’s operating speed and mode.
   - Run “stats_chk” command to view the MAC statistic counters.

   c. Avalon-ST Reverse Loopback Test (external packet generator sends Ethernet packets to Triple Speed Ethernet MAC thru the RJ45 port, loop back at the Avalon-ST interface of Triple Speed Ethernet MAC and then return back to the external packet generator) for 10M/100M/1000M speed:

   Command: TEST_ST_LB {speed_test}
   Example: TEST_ST_LB 1000M

   - The System Console displays the copper link connection status and the PHY’s operating speed and mode.
   - Start sending Ethernet packets from the external packet generator to the FPGA development board and verify that the packets are correctly looped back to the external packet generator.
   - Run “stats_chk” command to view the MAC statistic counters.

Tcl Script

This section describes the Tcl scripts inside the **sc_tcl** folder of the design example. You can use any plain text editor to edit the Tcl scripts. Altera recommends that you do not modify the **tse_mac_config.tcl**, **tse_marvel_phy.tcl**, **eth_gen_mon.tcl** and **tse_stat_read.tcl** scripts inside the **sc_tcl** folder.

Configuration Script

The **tse_mac_config.tcl** and **tse_marvel_phy.tcl** configuration script contains the parameters to configure the MAC and Marvell PHY registers in the reference designs. You can configure the following settings in the Tcl script:

- MAC configuration setting—allows you to configure the MAC registers. For more information about the MAC configuration registers, refer to the *Configuration Register Space* chapter in the *Triple Speed Ethernet User Guide*.
- Marvell PHY configuration setting—allows you to configure the on-board PHY chip registers.
    - **PHY_ENABLE**—use this parameter to enable or disable the on-board PHY chip.
    - **PHY_ETH_SPEED**—select the PHY's operating speed.
    - **PHY_ENABLE_AN**—use this parameter to enable or disable auto-negotiation on the PHY.
    - **PHY_COPPER_DUPLEX**—select the PHY's operating mode.

Statistic Counter Script.

The **tse_stat_read.tcl** script reads the values of the MAC statistic counters after you execute the design example. For more information about the MAC statistic counters, refer to the *Configuration Register Space* chapter in the *Triple Speed Ethernet User Guide*.

Ethernet Packet Generator Script

The **eth_gen_start.tcl** configuration script contains the parameters to configure the Ethernet Packet Generator registers in the reference designs. You can configure the following settings in the Tcl script:

- **number_packet**—set the total number of packets to be generated by the packet generator.
- **eth_gen**—use this parameter to enable or disable the packet generator.
- **length_sel**—select fixed or random packet length.
- **pkt_length**—set the fixed packet length. The packet length can be a value between 24 to 9,600 bytes.
- **pattern_sel**—select the data pattern for the random packet length.
- **rand_seed**—set the initial random seed for the PRBS generator. This parameter is only valid when you select random packet length.
- **source_addr**—set the source MAC address.
- **destination_addr**—set the destination MAC address.

Document Revision History

Date	Version	Changes
Dec 2015	1.0	Initial release