AN 477: Designing RGMII Interfaces with FPGAs and HardCopy ASICs

© January 2010 AN-477-2.0

This application note describes how to design a reduced gigabit media independent interface (RGMII) with Stratix®, Arria®, and Cyclone® FPGAs and HardCopy® ASICs.

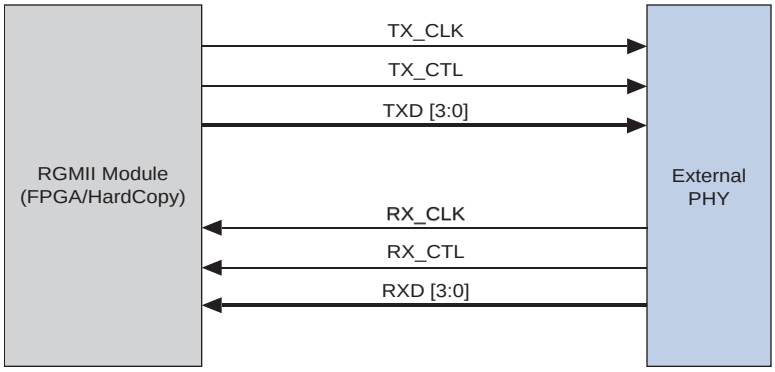
RGMII is an alternative to the IEEE 802.3z GMII with reduced pin count. Pin count reduction is achieved by clocking data on both the rising and falling edges of the clock and by multiplexing the control signals.

☞ You must be familiar with RGMII, Synopsys design constraints (SDC), and the TimeQuest Timing Analyzer before you read this application note.

System-Level Diagram

Figure 1 shows a block diagram of RGMII implementation. An RGMII interface module is implemented inside an FPGA or HardCopy ASIC and is connected to an external RGMII PHY. All signals are synchronous with a 125-MHz clock signal.

Figure 1. Signal Diagram of RGMII



RGMII data is sampled on both edges of the clock. Table 1 lists the signal descriptions. Typically, the clock and data from the RGMII PHY are generated simultaneously, that is, edge-aligned; thus the clocks must be routed with an added trace delay on the PCB.

Table 1. Signal Description of RGMII (Part 1 of 2)

Signal	I/O Type	Description
TX_CLK	Output	Transmit clock from an FPGA and HardCopy ASIC.
TXD	Output	Bits 3:0 on the positive edge of TX_CLK and bits 7:4 on the negative edge of TX_CLK.
TX_CTL	Output	TXEN on the positive edge of TX_CLK and a logical derivative of TXEN and TXERR on the negative edge of TX_CLK.
RX_CLK	Input	Receive reference clock from the external PHY.

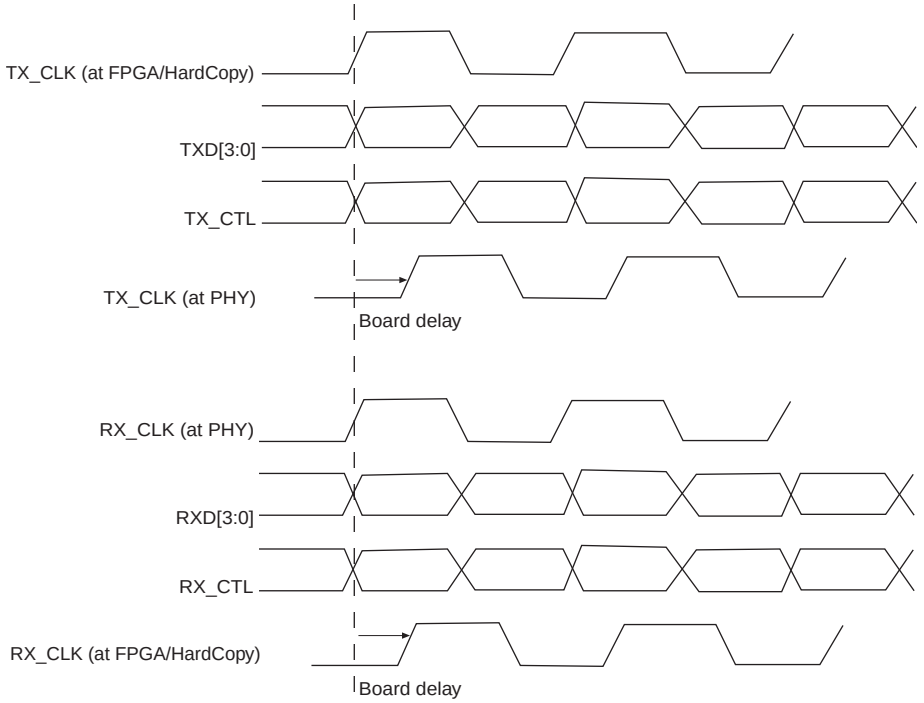
Table 1. Signal Description of RGMII (Part 2 of 2)

Signal	I/O Type	Description
RXD	Input	Bits 3:0 on the positive edge of RX_CLK and bits 7:4 on the negative edge of RX_CLK.
RX_CTL	Input	RXDV on the positive edge of RX_CLK and a derivative of RXDV and RXERR on the negative edge of RXC.

System Timing

Figure 2 shows the edge-aligned data and clock.

Figure 2. Timing Diagram for RGMII

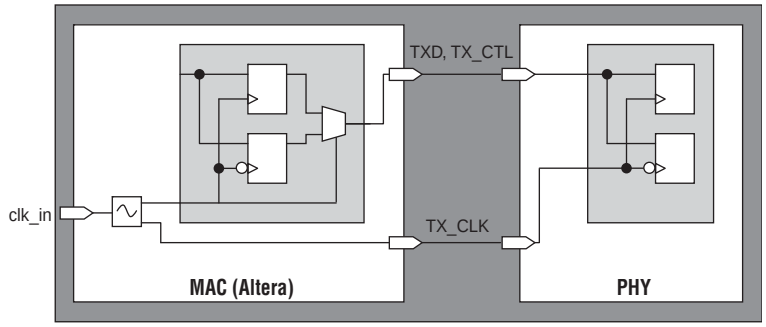


Edge-aligned data requirements complicate the PCB design, such that later revisions of the RGMII external PHY offer an option to operate with or without internal delay. This application note reviews the implementations of transmit and receive interfaces with the RGMII PHY.

Implementation of an FPGA and HardCopy ASIC Transmit Interface

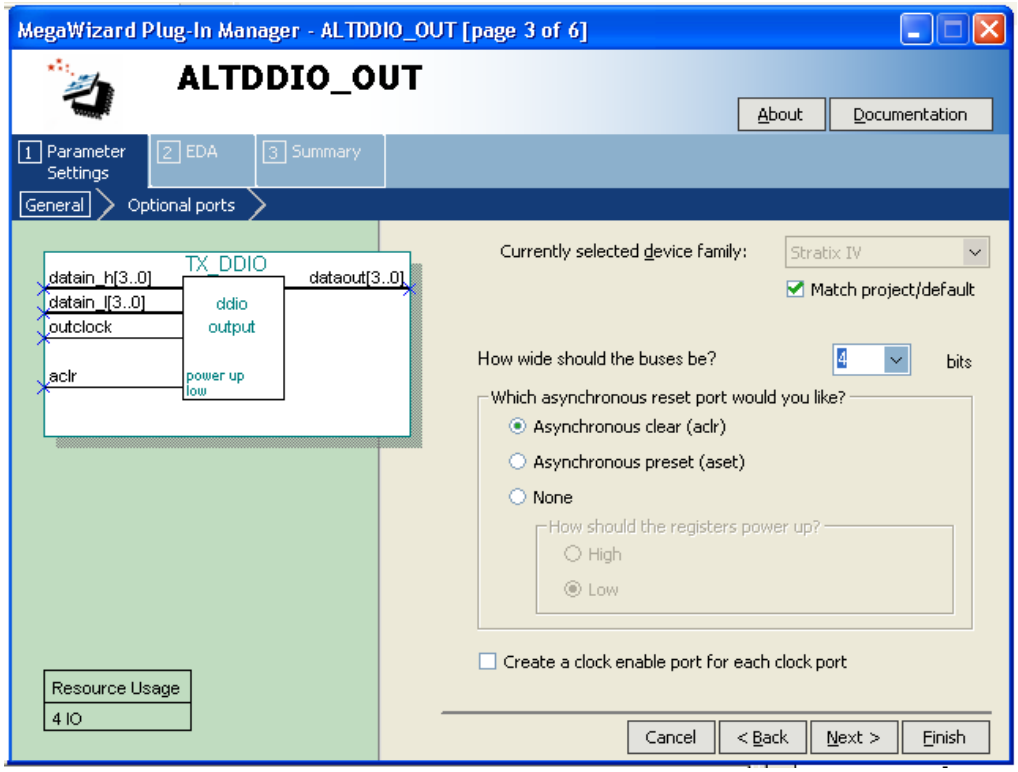
Implementing the transmit interface is a straight-forward process. Figure 3 shows a block diagram of the transmit interface.

Figure 3. Block Diagram of Transmit Interface



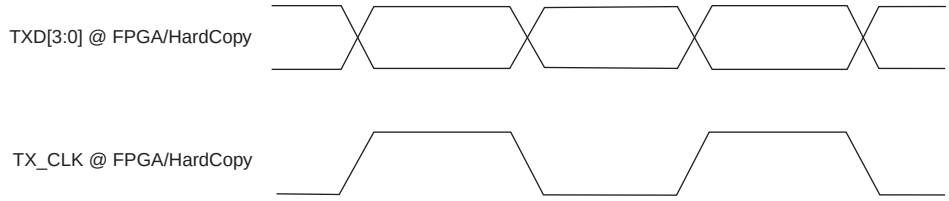
You can place the interface on any double data input/output (DDIO) I/O register, through the Altera® ALTDDIO_OUT megafunction, as shown in Figure 4.

Figure 4. DDIO Megafunction for Transmit Interface



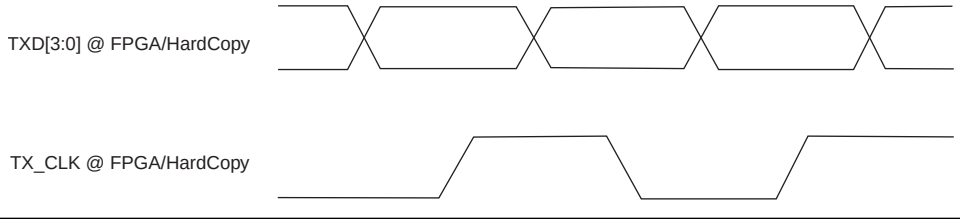
For data and clock transmission, most PHY devices that support RGMII offer an option to add delay to the transmit or receive clock. You can enable or disable this option based on your design requirements. When you enable the option to delay the TX_CLK inside the PHY device, the FPGA and HardCopy ASIC must generate a clock that is edge-aligned with the data and waveforms, as shown in Figure 5.

Figure 5. FPGA-Generated TX_CLK When TX_CLK Delay is Enabled in the PHY



In this case, the PHY device shifts the clock as necessary to capture the data. When you disable this option, the FPGA and HardCopy ASIC must generate a clock that is shifted with respect to the data (typically center-aligned with the data) with board delay consideration and waveforms, as shown in Figure 6. This shifted clock is used by the PHY device to capture the data.

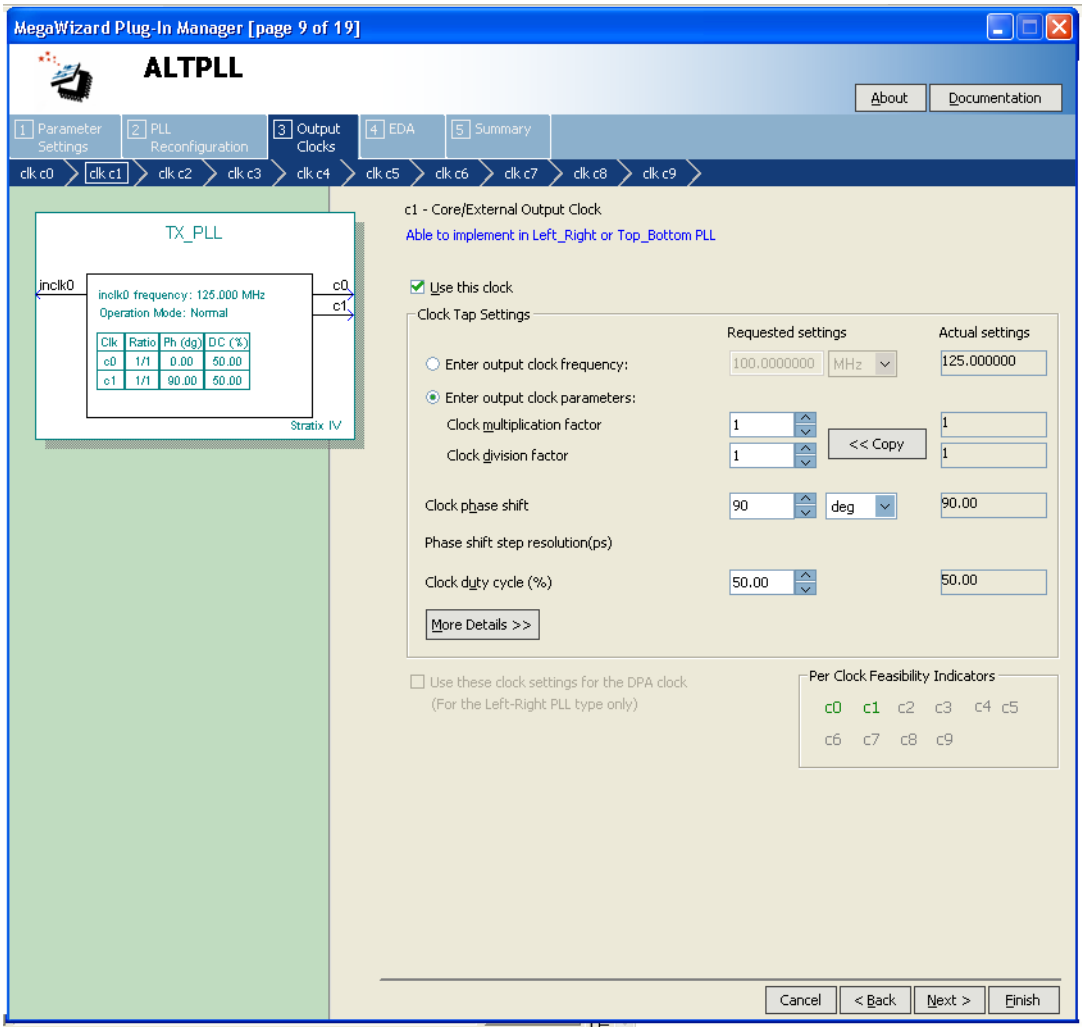
Figure 6. FPGA-Generated TX_CLK When TX_CLK Delay is Disabled in the PHY



You can use various methods to align the sourced clock with the data. You can drive clocks by the same clock that registers the data or that was created by a toggling clock output register, such as in the ALTDDIO_OUT megafunction. When the output clocks are generated independently from the data output register clocks (for example, two phase-locked loop [PLL] taps), you can change the clock and data timing relationship by adjusting the relationship between their clocks (for example, adjusting PLL phase). You can achieve this with Altera's PLL megafunction.

Figure 7 shows the ALTPLL megafunction that creates two separate clocks for TXD and TX_CLK.

Figure 7. ALTPLL Megafunction to Generate Clocks for TXD and TX_CLK



Enabling the TX_CLK delay option inside the PHY simplifies the transmit interface because you do not have to add logic to center-align data on the FPGA.

For more information about implementing source-synchronous interfaces, refer to *AN 433: Constraining and Analyzing Source-Synchronous Interfaces*.

Implementing an FPGA and HardCopy ASIC Receive Interface

You can implement the receive interface on any DDIO I/O register, as shown in Figure 8.

Figure 8. Block Diagram of the Receive Interface

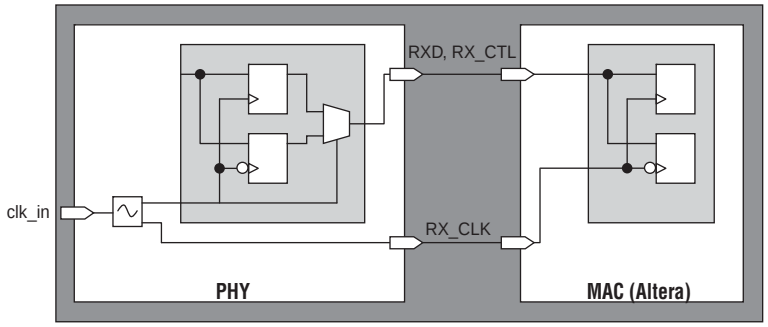
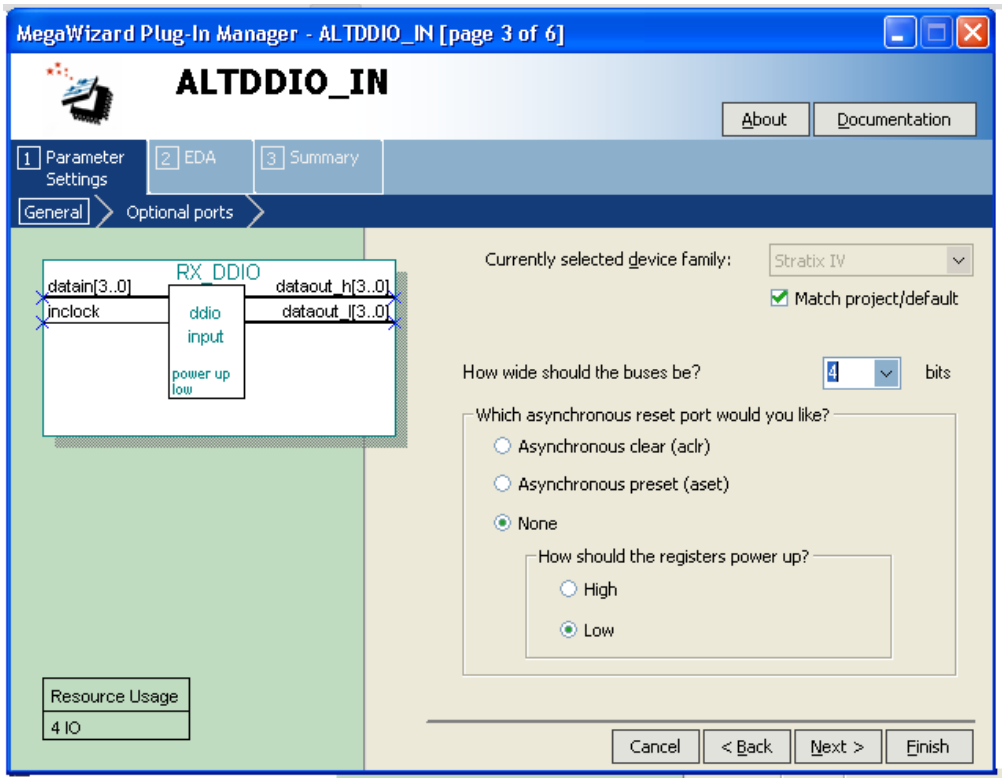


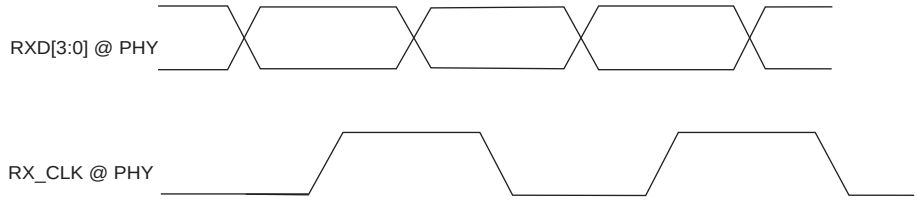
Figure 9 shows the ALTDDIO_IN megafunction for the receive interface.

Figure 9. ALTDDIO_IN Megafunction for the Receive Interface



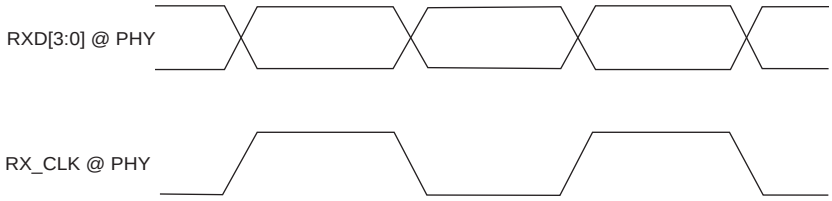
For data capture, the RGMII external PHY offers an option to add delay to RX_CLK. When you enable the option to delay RX_CLK at the external PHY device, the PHY device transmits a clock that is center-aligned with the data and waveforms, as shown in Figure 10. Thus, the FPGA and HardCopy ASIC can capture the incoming data directly using RX_CLK without additional board delays and internal delays in the FPGA or HardCopy ASIC.

Figure 10. PHY-Generated RX_CLK When RX_CLK Delay is Enabled in the PHY



However, when you disable this delay option (for the timing diagram, refer to Figure 11), the FPGA and HardCopy ASIC must shift the clock as necessary to capture the data. You can achieve this with a DLL and by assigning RX_CLK to a DQS pin or promoting RX_CLK to a global or regional net if you need a small clock latency.

Figure 11. PHY-Generated RX_CLK When RX_CLK Delay is Disabled in the PHY



Enabling the **RX_CLK delay** option inside the PHY is generally preferred because it reduces the complexity of the receive logic in the FPGA device and provides better timing margin.

Altera Recommendation

The following sections describe Altera recommendations for designing RGMII interfaces.

Clock Pin Placement

If you plan to migrate your design to a HardCopy ASIC, Altera recommends using primary clock inputs in the FPGA whenever possible for RX_CLK. If you choose a non-clock input pin for RX_CLK, the clock latency—and thus, the timing results reported by the TimeQuest Timing Analyzer—may be different between the FPGA and the HardCopy ASIC. If primary clock inputs are not available, you can place the I/O close to the clock pins on the pin planner to reduce this discrepancy.

Data and Clock Alignment

When an internal delay option is available in external PHY devices, Altera recommends enabling this option for the receive interface. This added delay significantly reduces design complexity and offers improved timing margins for HardCopy ASIC migration. For the transmit interface, Altera recommends disabling the delay option and using a PLL inside the FPGA or HardCopy ASIC to center-align the data if the time slack of the transmit interface is tight, because PLLs provide superior timing accuracy with their internal compensation capability.

Table 2 lists the data and clock alignment recommendation.

Table 2. Data and Clock Alignment Recommendation

Interface	Clock Alignment	PHY Delay Enabled	Aligned By
Transmit	Center	No	FPGA / HardCopy ASIC
Receive	Center	Yes	PHY

Clock Uncertainty Value

In addition, the HardCopy ASIC design requires you to insert clock uncertainty values (refer to "Constraining Clock Uncertainty" on page 15).

Connecting an FPGA to an External PHY Device When the Delay Option Is Disabled

This section presents examples of RGMII timing constraints.

In the following example, the options of the external PHY device to delay both TX_CLK and RX_CLK are disabled. Figure 12 shows the timing waveform at the external PHY device. To meet the timing requirements of the external device, both TX_CLK and RX_CLK are delayed by a PLL inside the FPGA, as shown in Figure 13 and Figure 14.

The external PHY I/O timing requirements are as follows:

- Setup time for TXD = 1.0 ns
- Hold time for TXD = 0.8 ns
- Incoming clock-to-data skew (RX_CLK-to-RXD) = ± 500 ps

Assume trace delay, pin capacitance, and rise/fall time differences between data and clock are negligible.

The calculated FPGA and HardCopy ASIC timing constraints are as follows:

- Output maximum delay = t_{su} of external register = 1.0 ns
- Output minimum delay = $-t_h$ of external register = –0.8 ns
- Input maximum delay = t_{co} of external device = 0.5 ns
- Input minimum delay = t_{co} of external device = –0.5 ns

For more information about constraining source-synchronous interfaces, refer to *AN 433: Constraining and Analyzing Source-Synchronous Interfaces*.

Figure 12 shows the timing waveform at the external PHY device.

Figure 12. Timing Waveform When the Delay Option at the External PHY is Disabled

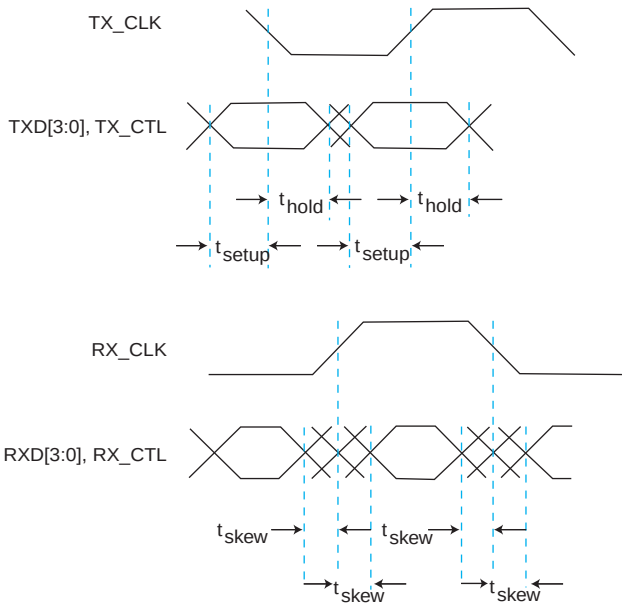


Figure 13 shows TX_CLK delayed by a PLL inside the FPGA.

Figure 13. RGMII Transmit Implementation When the Delay Option is Disabled

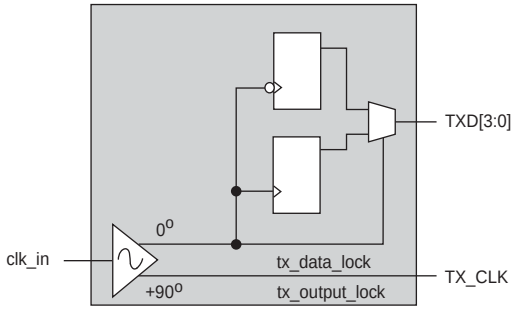
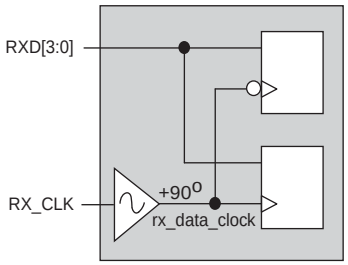


Figure 14 shows RX_CLK delayed by a PLL inside the FPGA.

Figure 14. RGMII Receive Implementation When the Delay Option is Disabled



At the FPGA (Transmit) Side

Use the code in Example 1 on the transmit side.

Example 1.

```
# Create a 125MHz clock and apply generated clocks to the PLL outputs
# pll|clk[0]: clock for TXD registers
# pll|clk[1]: TX_CLK with 90 degree phase shift--this delay is design-dependent
create_clock -name input_clock -period 8 [get_ports clk_in]

# Below are the pll derived clocks, which can also be generated by typing
# derive_pll_clocks in the TCL console
# You can refer to The Quartus II TimeQuest Timing Analyzer chapter in Quartus II handbook
create_generated_clock -name tx_data_clock -source [get_pins \
{tx_pll|altpll_component|pll|inclk[0]}] [get_pins \
{tx_pll|altpll_component|pll|clk[0]}]
create_generated_clock -name pll_output -phase 90 -source [get_pins \
{tx_pll|altpll_component|pll|inclk[0]}] [get_pins \
{tx_pll|altpll_component|pll|clk[1]}]

# Apply a generated clock to the clk_out port
create_generated_clock -name tx_output_clock -source [get_pins \
{tx_pll|altpll_component|pll|clk[1]}] [get_ports {TX_CLK}]

# Set output delay based on the requirements mentioned previously
set_output_delay -clock tx_output_clock -max 1.0 [get_ports TXD*]
set_output_delay -clock tx_output_clock -min -0.8 [get_ports TXD*] -add_delay
set_output_delay -clock tx_output_clock -clock_fall -max 1.0 [get_ports TXD*] -add_delay
set_output_delay -clock tx_output_clock -clock_fall -min -0.8 [get_ports TXD*] \
-add_delay
set_output_delay -clock tx_output_clock -max 1.0 [get_ports {TX_CTL}]
set_output_delay -clock tx_output_clock -min -0.8 [get_ports {TX_CTL}] -add_delay
set_output_delay -clock tx_output_clock -clock_fall -max 1.0 [get_ports {TX_CTL}]
set_output_delay -clock tx_output_clock -clock_fall -min -0.8 [get_ports {TX_CTL}]

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

At the FPGA (Receive) Side

Use the code in Example 2 on the receive side.

Example 2.

```
# Create a 125MHz clock and apply generated clocks to the PLL outputs
# virtual_source: an ideal clock in the sourcing device
# RX_CLK: input clock port of the interface
# pll|clk[0]: clock for capturing RXD and RX_CTL with 90 degree phase shift - this delay
# is design-dependent
create_clock -name virtual_source -period 8
create_clock -name RX_CLK -period 8 [get_ports RX_CLK]

# Below is the pll derived clock, which can also be generated by derive_pll_clocks

# Refer to the Quartus II TimeQuest Timing Analyzer chapter in Quartus II Handbook.
create_generated_clock -name rx_data_clk -phase 90 -source [get_pins \
{rx_pll|altpll_component|pll|inclk[0]}] [get_pins \
{rx_pll|altpll_component|pll|clk[0]}]

# Set multicycle paths to align the launch edge with the latch edge
set_multicycle_path 0 -setup -end -rise_from [get_clocks virtual_source] -rise_to \
[get_clocks {RX_CLK}]
set_multicycle_path 0 -setup -end -fall_from [get_clocks virtual_source] -fall_to \
[get_clocks {RX_CLK}]

# Set input delay based on the requirements mentioned previously
set_input_delay -max 0.5 -clock [get_clocks virtual_source] -add_delay [get_ports RXD*]
set_input_delay -min -0.5 -clock [get_clocks virtual_source] -add_delay [get_ports RXD*]
set_input_delay -max 0.5 -clock [get_clocks virtual_source] -clock_fall -add_delay \
[get_ports RXD*]
set_input_delay -min -0.5 -clock [get_clocks virtual_source] -clock_fall -add_delay \
[get_ports RXD*]
set_input_delay -max 0.5 -clock [get_clocks virtual_source] -add_delay [get_ports \
{RX_CTL}]
set_input_delay -min -0.5 -clock [get_clocks virtual_source] -add_delay [get_ports \
{RX_CTL}]
set_input_delay -max 0.5 -clock [get_clocks virtual_source] -clock_fall -add_delay \
[get_ports {RX_CTL}]
set_input_delay -min -0.5 -clock [get_clocks virtual_source] -clock_fall -add_delay \
[get_ports {RX_CTL}]

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

Connecting an FPGA to an External PHY Device When the Delay Option Is Enabled

If an external PHY device supports internal delay, you do not need additional logic. The FPGA transmits edge-aligned data and receives center-aligned data. Figure 15 shows the timing waveform at the external PHY device. 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 or HardCopy ASIC to capture the data, as shown in Figure 16 and Figure 17.

External I/O timing requirements are as follows:

- Minimum setup time = 0.9 ns
- Minimum hold time = 2.7 ns
- Minimum clock delay added internally by the PHY device, Tdelay_min = 1.2 ns
- Maximum clock delay added internally by the PHY device, Tdelay_max = 2.8 ns

Assume trace delay, pin capacitance, and rise/fall time differences between the data and clock are negligible.

The calculated FPGA and HardCopy ASIC timing constraints are as follows:

- Rising edge of RX_CLK (90° phase shift) = 2 ns
- Calculated output maximum delay = t_{su} of external register = –0.9 ns
- Calculated output minimum delay = -t_h of external register = –2.7 ns
- Calculated input maximum delay = t_{co} of external device = 2.8 ns – 2 ns (rising edge of RX_CLK) = 0.8 ns
- Calculated input minimum delay = t_{co} of external device = 1.2 ns – 2 ns (rising edge of RX_CLK) = –0.8 ns

Figure 15 shows the timing waveform at the external PHY device.

Figure 15. Timing Waveform When the Delay Option Is Enabled at the External PHY Device

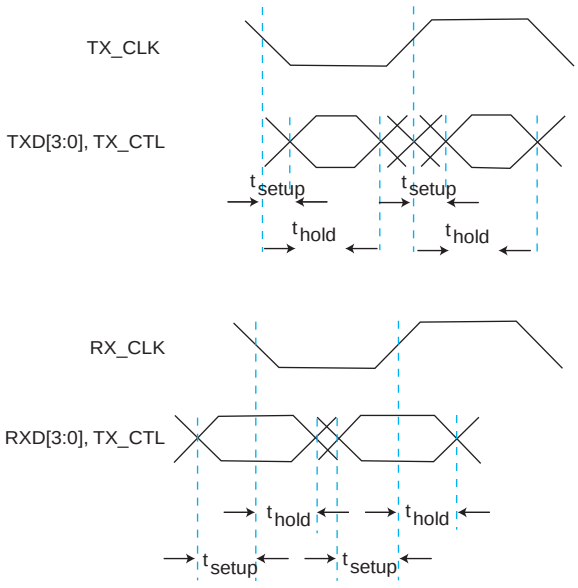
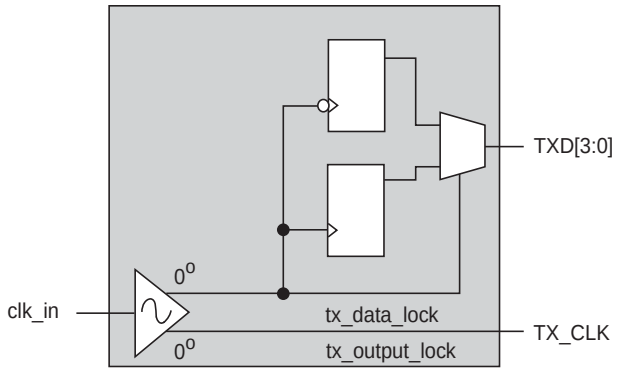


Figure 16 shows the RGMII transmit implementation when the delay option is enabled.

Figure 16. RGMII Transmit Implementation when the Delay Option is Enabled



At the FPGA (Transmit) Side

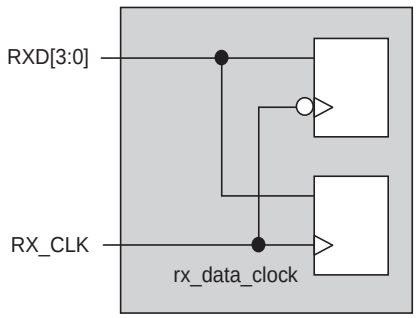
Use the code in Example 3 on the transmit side.

Example 3.

```
# Create a 125MHz clock and apply generated clocks to # the PLL outputs
# pll|clk[0]: TX_CLK
# pll|clk[1]: clock for TXD registers
create_clock -name input_clock -period 8 [get_ports {clk_in}]

# Below are the pll derived clocks, which can also be generated by typing
# derive_pll_clocks in the TCL console
# Refer to the Quartus II TimeQuest Timing Analyzer chapter in Quartus II Handbook.
create_generated_clock -name tx_data_clock -source [get_pins \
{tx_pll|altpll_component|pll|inclk[0]}] [get_pins \
{tx_pll|altpll_component|pll|clk[0]}]
create_generated_clock -name tx_output_clock -source [get_pins \
{tx_pll|altpll_component|pll|clk[0]}] [get_ports {TX_CLK}]

# Set output delay based on the requirements mentioned previously
set_output_delay -clock tx_output_clock -max -0.9 [get_ports TXD*]
set_output_delay -clock tx_output_clock -min -2.7 [get_ports TXD*] -add_delay
set_output_delay -clock tx_output_clock -clock_fall -max -0.9 [get_ports TXD*]
set_output_delay -clock tx_output_clock -clock_fall -min -2.7 [get_ports TXD*]
set_output_delay -clock tx_output_clock -max -0.9 [get_ports {TX_CTL}]
set_output_delay -clock tx_output_clock -min -2.7 [get_ports {TX_CTL}] -add_delay
set_output_delay -clock tx_output_clock -clock_fall -max -0.9 [get_ports {TX_CTL}]
set_output_delay -clock tx_output_clock -clock_fall -min -2.7 [get_ports {TX_CTL}]

# Set multicycle paths to align the launch edge with the latch edge
set_multicycle_path 0 -setup -end -rise_from [get_clocks tx_data_clock] -rise_to \
[get_clocks tx_output_clock]
set_multicycle_path 0 -setup -end -fall_from [get_clocks tx_data_clock] -fall_to \
[get_clocks tx_output_clock]

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

Figure 17 shows the RGMII receive implementation with the delay option enabled.

Figure 17. RGMII Receive Implementation When the Delay Option is Enabled



At the FPGA (Receive) Side

Use the code in Example 4 on the receive side.

Example 4.

```
# Create a 125MHz clock
# virtual_source: an ideal clock in the sourcing device
# RX_CLK: input clock port of the interface; 90 deg phase shifted
create_clock -name virtual_source -period 8
create_clock -name RX_CLK -period 8 -waveform { 2 6 } [get_ports {RX_CLK}]

# Set input delay based on the requirements mentioned previously
# RX_CLK is 90 deg phase shifted
# Input delay is relative to the rising and falling edges of the clock
set_input_delay -max 0.8 -clock [get_clocks virtual_source] -add_delay [get_ports RXD*]
set_input_delay -min -0.8 -clock [get_clocks virtual_source] -add_delay [get_ports RXD*]
set_input_delay -max 0.8 -clock_fall -clock [get_clocks virtual_source] -add_delay \
[get_ports RXD*]
set_input_delay -min -0.8 -clock_fall -clock [get_clocks virtual_source] -add_delay \
[get_ports RXD*]
set_input_delay -max 0.8 -clock [get_clocks virtual_source] -add_delay [get_ports \
{RX_CTL}]
set_input_delay -min -0.8 -clock [get_clocks virtual_source] -add_delay [get_ports \
{RX_CTL}]
set_input_delay -max 0.8 -clock_fall -clock [get_clocks virtual_source] -add_delay \
[get_ports {RX_CTL}]
set_input_delay -min -0.8 -clock_fall -clock [get_clocks virtual_source] -add_delay \
[get_ports {RX_CTL}]

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

Constraining Clock Uncertainty

For all 65 nm and beyond FPGAs and HardCopy ASICs, including 90 nm HardCopy II ASICs, you are required to insert clock uncertainty values in the timing constraint file. This can be readily obtained by typing the `derive_clock_uncertainty` command in the TimeQuest Timing Analyzer console.

For more information about clock uncertainty, refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Clock Uncertainty for HardCopy ASICs

Example 5 shows the clock uncertainty constraints generated by the `derive_clock_uncertainty` command. Clock uncertainty values may vary depending on the resource usage. The uncertainty values in Example 5 are for illustration purposes only.

Example 5.

```
set_clock_uncertainty -from tx_data_clock -to tx_output_clock -setup 0.100
set_clock_uncertainty -from tx_data_clock -to tx_output_clock -hold 0.050
set_clock_uncertainty -from tx_data_clock -to tx_data_clock -setup 0.100
set_clock_uncertainty -from tx_data_clock -to tx_data_clock -hold 0.050
set_clock_uncertainty -from virtual_source -to rx_data_clk -setup 0.130
set_clock_uncertainty -from virtual_source -to rx_data_clk -hold 0.130
set_clock_uncertainty -from rx_data_clk -to rx_data_clk -setup 0.100
set_clock_uncertainty -from rx_data_clk -to rx_data_clk -hold 0.050
set_clock_uncertainty -from virtual_source -to RX_CLK -setup 0.100
set_clock_uncertainty -from virtual_source -to RX_CLK -hold 0.100
set_clock_uncertainty -from RX_CLK -to RX_CLK -setup 0.100
```

Conclusion

Designing the RGMII interface with an Altera FPGA is a straight-forward process. Stratix, Arria, and Cyclone FPGAs and HardCopy ASICs offer the complete solution, including hardware, software, and documentation to help you build a robust RGMII interface.

When you follow this design guideline and the recommendations in this application note, you can implement the interface with confidence, especially when converting your design to a HardCopy ASIC.

Document Revision History

Table 3 shows the revision history for this document.

Table 3. Document Revision History

Date and Document Version	Changes Made
January 2010, v2.0	■ Changed all “HardCopy device(s)” references to “HardCopy ASIC(s)”. ■ Changed all “Stratix II and Cyclone II” references to “Stratix and Cyclone FPGAs”. ■ Updated Figure 4, Figure 7, and Figure 9. ■ Minor text edits. ■ Converted to the updated template.
November 2007, v1.0	Initial release.

101 Innovation Drive
San Jose, CA 95134
www.altera.com
Technical Support
www.altera.com/support