Using Triple-Speed Ethernet on DE2-115 Boards

For Quartus® Prime 18.1

1 Introduction

This tutorial provides a basic introduction to the usage of Triple-Speed Ethernet on Intel's DE2-115 boards. It first demonstrates how to build a system with the Triple-Speed IP Core using Platform Designer software and then shows how to run an application program. The discussion assumes that the reader has a basic knowledge of Verilog hardware description language and is familiar with the Quartus® Prime software and the Intel® Platform Designer tool.

The screen captures in the tutorial were obtained using the Quartus Prime version 18.1; if other versions are used, some of the images may be slightly different.

Contents:

- Background
- Implementing a Triple-Speed Ethernet System
- Application Program for the Ethernet System
- Concluding Remarks

2 Background

Ethernet is a relatively inexpensive, reasonably fast and very popular Local Area Network (LAN) technology. When first deployed in the 1980s, it supported a maximum theoretical data rate of 10 megabits per second (Mbps). Later, Fast Ethernet extended the performance up to 100 Mbps and Gigabit Ethernet up to 1000 Mbps. Although products are not yet available to the average users, 10/100 Gigabit Ethernet (10000/100000 Mbps) are the latest published high-speed Ethernet standards.

Ethernet operates across two layers of the Open Systems Interconnect (OSI) model: the Data Link layer and the Physical layer as shown in Figure 1. The model provides a reference to which Ethernet can be related, but Ethernet is actually implemented in the lower half of the Data Link layer, which is known as the Media Access Control (MAC) sublayer, and the Physical layer only. The MAC sublayer has the responsibility for data encapsulation including frame assembly before transmission and frame parsing upon reception of a frame. The MAC also controls the initiation of frame transmission and recovery from transmission failure due to collisions. The upper half of the Data Link layer is called the Logical Link Control (LLC) sublayer. It handles the communication between the upper layers and the lower layers, the MAC sublayer in this case. Unlike the MAC sublayer, the LLC sublayer is implemented in software, and its implementation is independent of the physical equipment. In a computer, the LLC can be considered as the driver software for the Network Interface Card (NIC).

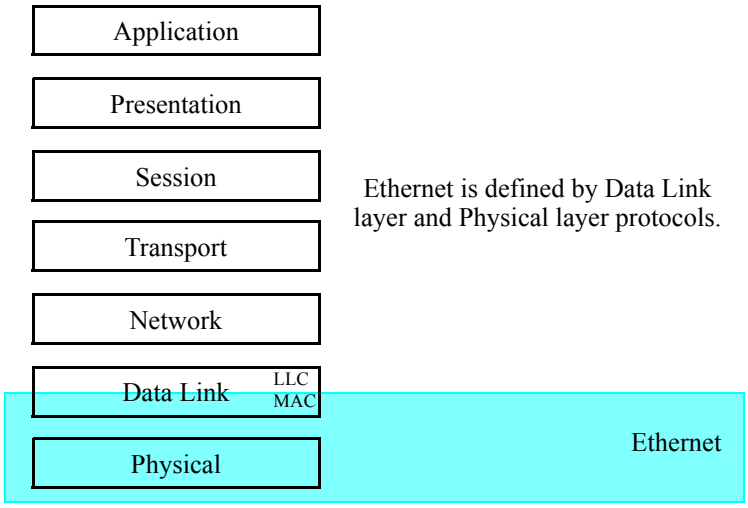

Figure 1. The block diagram that shows Ethernet standard inside the OSI model.

The DE2-115 board provides Ethernet support via the Marvell* 88E1111 Ethernet PHY chip, which is a physical layer device integrated with a 10/100/1000 Mbps Ethernet transceiver. To communicate with this chip, an Intel IP Core called Triple-Speed Ethernet can be used. This IP Core provides the features of a 10/100/1000-Mbps Ethernet Media Access Controller.

3 Implementing a Triple-Speed Ethernet System

This section describes how to implement the Triple-Speed Ethernet system shown in Figure 2. To start, create a new Quartus Prime project named *tse_tutorial* in a new directory of the same name. Select the Cyclone® IV E device EP4CE115F29C7, which is the FPGA on the DE2-115 board.

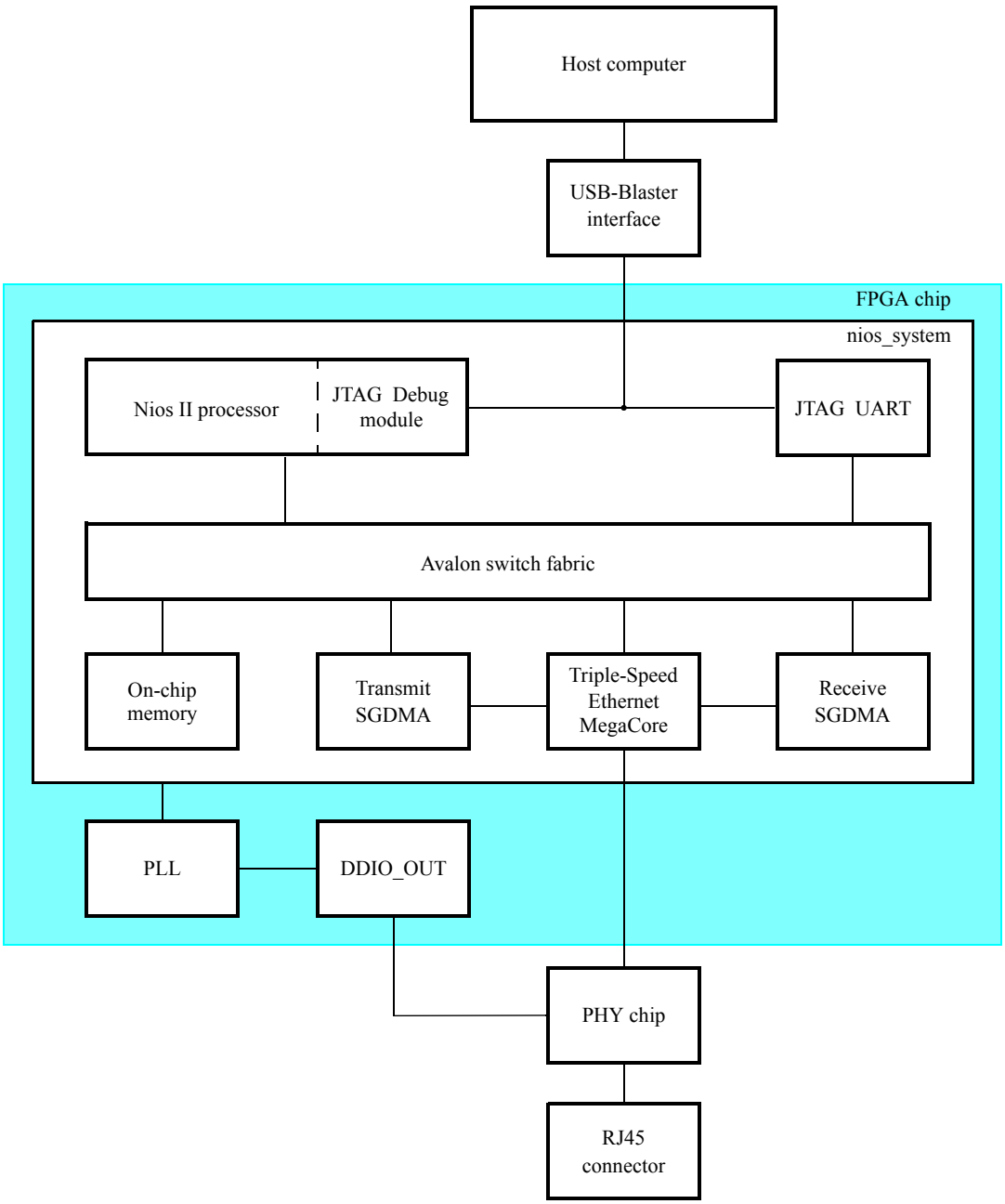

Figure 2. The block diagram of the Triple-Speed Ethernet system.

3.1 Creating the Nios® System

Most of the Triple-Speed Ethernet system can be built as a subsystem using the Platform Designer tool. The block diagram of this subsystem is shown in Figure 3.

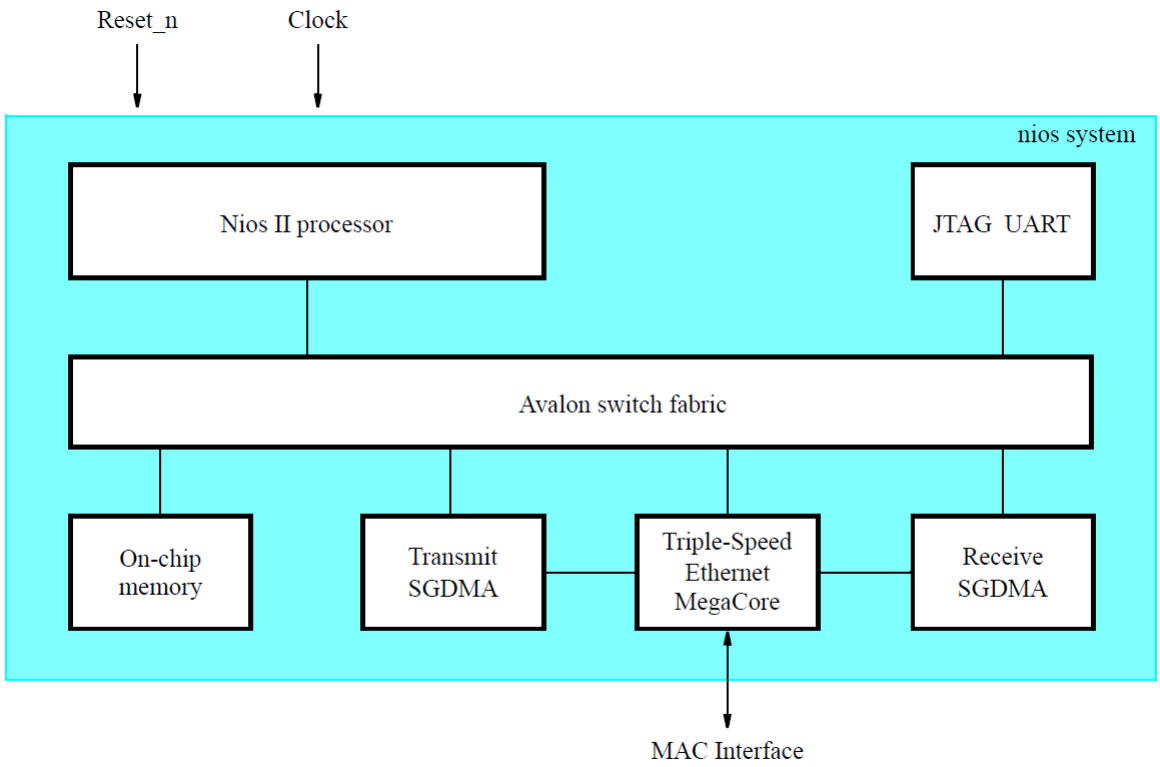

Figure 3. The block diagram of the Platform Designer subsystem.

This subsystem takes a clock signal and an active-low reset signal as inputs, and communicates with the external PHY chip through a chosen interface. There is a Nios® II processor to run application programs, a JTAG* UART component to support communication between the processor and the host computer, and a Triple-Speed Ethernet IP Core to implement the MAC sublayer and a partial Physical layer when needed based on the interface type. The two SGDMA controllers are used for the transmit and receive functions of the core. The on-chip memory is used for the program code, data, as well as descriptors for the SGDMA controllers. Note that the example system of this tutorial uses on-chip memory to simplify the system. For a larger system, it is better to use the SDRAM to have a larger memory for application programs.

To design the desired system, you should use the Platform Designer tool to add the components mentioned above and make necessary connections between them. You may want to rename the components as well to make the names more descriptive. Perform the following steps:

1. Select **Tools > Platform Designer** to open the Platform Designer tool, and then save the file as *nios_system.qsys*.

2. Double-click on the clock source *clk_0* and change the **Clock frequency** to 100000000 Hz (100MHz). Then, right-click on *clk_0* and rename it as *sys_clk*.

3. Add a Nios II processor. The Nios II processor is used to run application programs that handle the data sent to or received from the Triple-Speed Ethernet IP Core.

   - Select **Processors and Peripherals > Embedded Processors > Nios II (Classic) Processor** and click **Add**.
   - Choose **Nios II/s**, which is the standard version of the processor.
   - Click **Finish** to add the Nios II processor to the design, and rename it as *nios2*.
   - Click on the **Clock** column and select *sys_clk* as the clock input to the processor, as shown in Figure 4.

   There may be some error messages at the bottom of the screen, because some parameters have not been specified yet. Ignore these messages as we will provide the necessary parameters later.

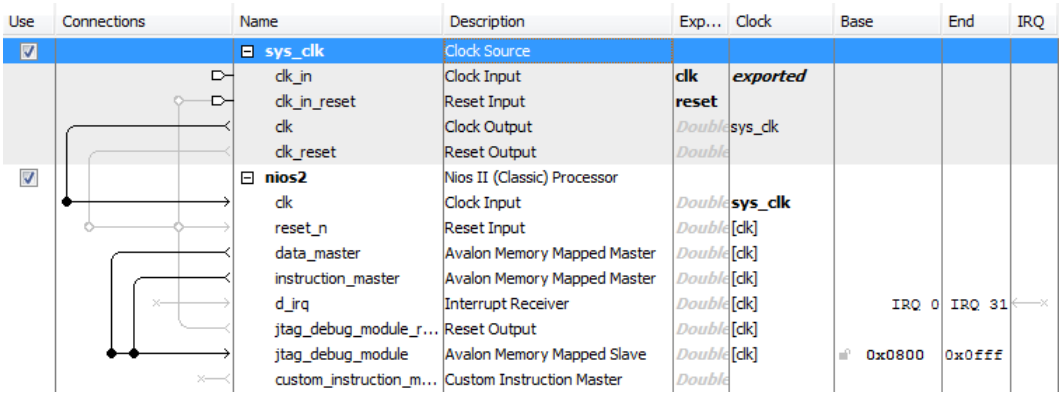

Figure 4. Select sys_clk as the clock input of the processor.

4. Add an on-chip memory, which will be used as the main memory to store programs and data for the processor. Any data received from the Triple-Speed Ethernet IP Core will be stored in this memory as well.

   - Select **Basic Functions > On Chip Memory > On-chip Memory(RAM or ROM) Intel FPGA IP** and click **Add**.
   - Set the **Total Memory Size** to 307200 bytes (300 KBytes), as shown in Figure 5.
   - Do not change the other default settings.
   - Click **Finish** and rename the on-chip memory as *main_memory*.
   - Select *sys_clk* as its clock input and connect its *s*1 slave port to both the *data_master* and *instruction_master* of the processor.

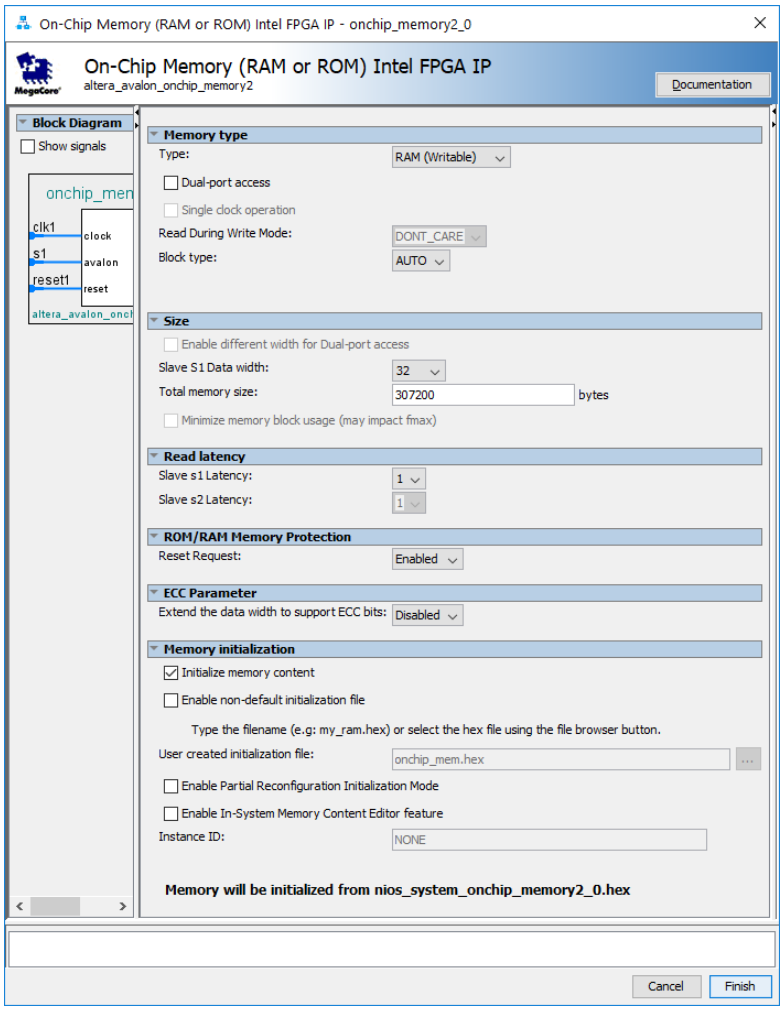

Figure 5. Settings for the on-chip memory.

5. Add a JTAG UART component. With the JTAG UART component, the Nios II processor is able to send data to the host computer, such as information that needs to be printed out to the terminal in the application program.
   - Select **Interface Protocols > Serial > JTAG UART Intel FPGA IP** and click **Add**.
   - Do not change the default settings.
   - Click **Finish** and rename it as *jtag_uart*.
   - Select *sys_clk* as its clock input and connect its *avalon_jtag_slave* port to the *data_master* port of the processor.
   - In the **IRQ** column, connect the interrupt sender port from the *Interrupt Sender* slave port to the *interrupt receiver* port of the processor and set the IRQ number to be 0.
6. Add a Triple-Speed Ethernet IP Core. It works as a Media Access Controller, which along with the Nios II processor and the external PHY chip are the key components of the Triple-Speed Ethernet system. For detailed information about this IP Core, refer to the *Triple-Speed Ethernet MegaCore Function User Guide*.

- Select **Interface Protocols > Ethernet > 1G Multi-rate Ethernet > Triple-Speed Ethernet Intel FPGA IP**.
- Change the interface to be **RGMII**, then select the **MAC Options** tab.
- Select the following options: **Enable MAC 10/100 half duplex support**, **Include statistics counters**, **Align packet headers to 32-bit boundary**, **Enable magic packet detection**, and **Include MDIO module (MDC/MDIO)**, as shown in Figure 6.
- Click **Finish** and rename it as *tse*.
- Select *sys_clk* as clock input for *receive_clock_connection*, *transmit_clock_connection* as well as *control_port_clock_connection*.
- Connect its *control_port* slave port to the *data_master* port of the processor.
- Export its *pcs_mac_tx_clock_connection*, *pcs_mac_rx_clock_connection*, *mac_status_connection*, *mac_rgmii_connection*, and *mac_mdio_connection* by double-clicking on the **Export** column.

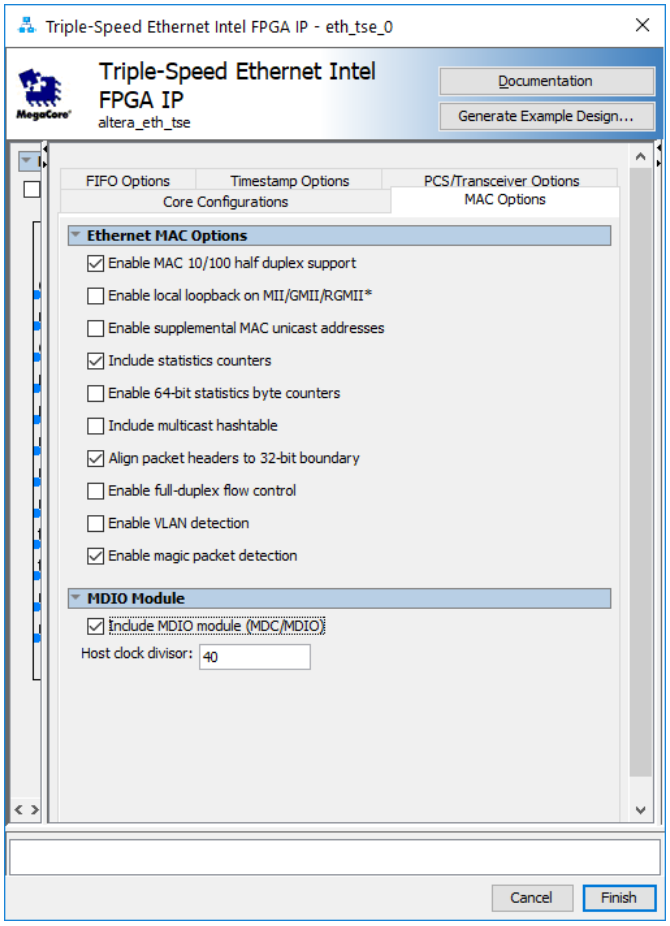

Figure 6. Settings for the Triple-Speed Ethernet IP Core.

7. Add an SGDMA controller for receive operation. This controller will be set to transfer data from a streaming interface to a memory-mapped interface, so that data can be transferred from the Triple-Speed Ethernet IP Core to the on-chip memory. The controller will interrupt the processor whenever it finishes the data transfer.

- Select **Basic Functions > DMA > Scatter-Gather DMA Controller Intel FPGA IP**.
- Select **Stream To Memory** as the transfer mode and **6** as the sink error width, as shown in Figure 7.
- Click **Finish** and rename it as *sgdma_rx*.
- Select *sys_clk* as its clock input and connect its *csr* slave port to the *data_master* port of the processor.
- Connect its *m_write* master port to the *s*1 port of the **main_memory** and its *in* streaming sink to the *receive* streaming source of the **tse** component.
- In the **IRQ** column, connect the interrupt sender port from the *csr* slave port to the interrupt receiver port of the processor and set the IRQ number to be 1.

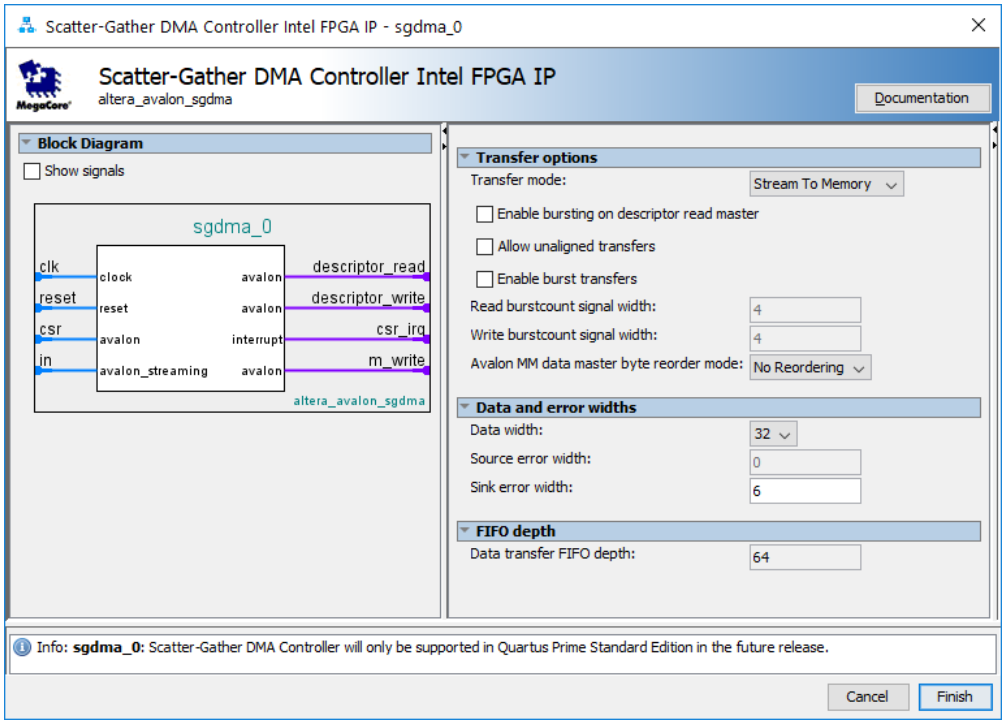

Figure 7. Settings for the SGDMA controller.

8. Add another SGDMA controller for transmit operation. This controller governs the reading of data from the on-chip memory *main_memory* and sending it to the Triple-Speed Ethernet IP Core.

- Select **Basic Functions > DMA > Scatter-Gather DMA Controller Intel FPGA IP**.
- As depicted in Figure 8, select **Memory To Stream** as the transfer mode and **1** as the source error width.
- Click **Finish** and rename it as *sgdma_tx*.
- Select *sys_clk* as its clock input and connect its *csr* slave port to the *data_master* port of the processor.
- Connect its *m_read* master port to the *s*1 port of the **main_memory** and its *out* streaming source to the *transmit* streaming sink of the **tse** component.
- In the **IRQ** column, connect its interrupt sender to the interrupt receiver of the processor and set the IRQ number to be 2.

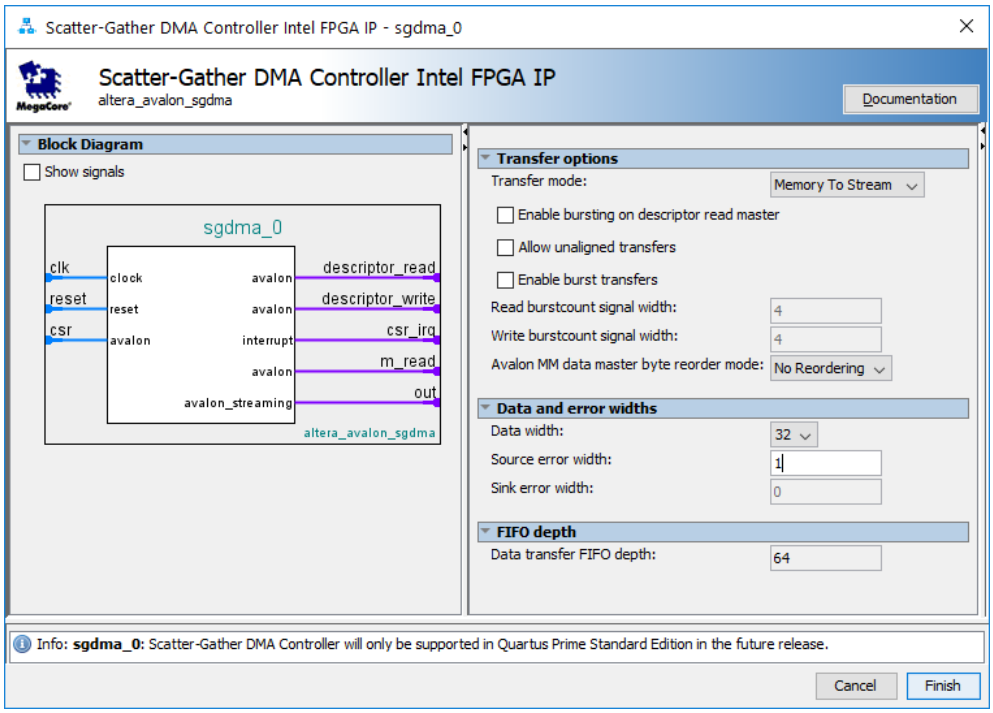

Figure 8. Settings for the other SGDMA controller.

9. Add another on-chip memory. Unlike the *main_memory* part, this on-chip memory is used to store only the descriptors of the SGDMA controllers.

   - Select **Basic Functions > On Chip Memory > On-chip Memory(RAM or ROM) Intel FPGA IP** and click **Add**.
   - Leave the default settings and click **Finish**.
   - Rename it as *descriptor_memory* and select *sys_clk* as its clock input
   - Connect its *s*1 slave port to the *data_master* of the processor, the *descriptor_read* and the *descriptor_write* ports for both **sgdma_tx** and **sgdma_rx**. This will guarantee that this on-chip memory is accessible for the processor and the two SGDMA controllers.

10. Now, all the components have been added, but the system is not complete as there are several error messages displayed.

    - Click on the drop-down menu **System** and click **Assign Base Address** to auto assign base addresses for all the components.
    - Under the same menu, click **Create Global Reset Network** to connect the reset signals to form a global reset network.
    - Double-click the Nios II processor *nios2* to edit its settings. Change both the reset and exception vector memory options to **main_memory.s1**, as shown in Figure 9. Then click **Finish**. The final system is shown in Figure 10.

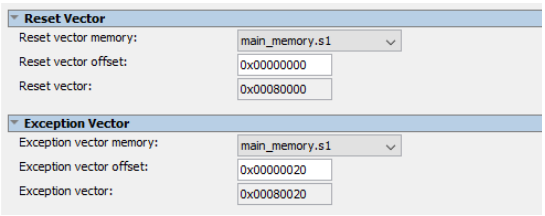

Figure 9. Settings for the Nios II processor.

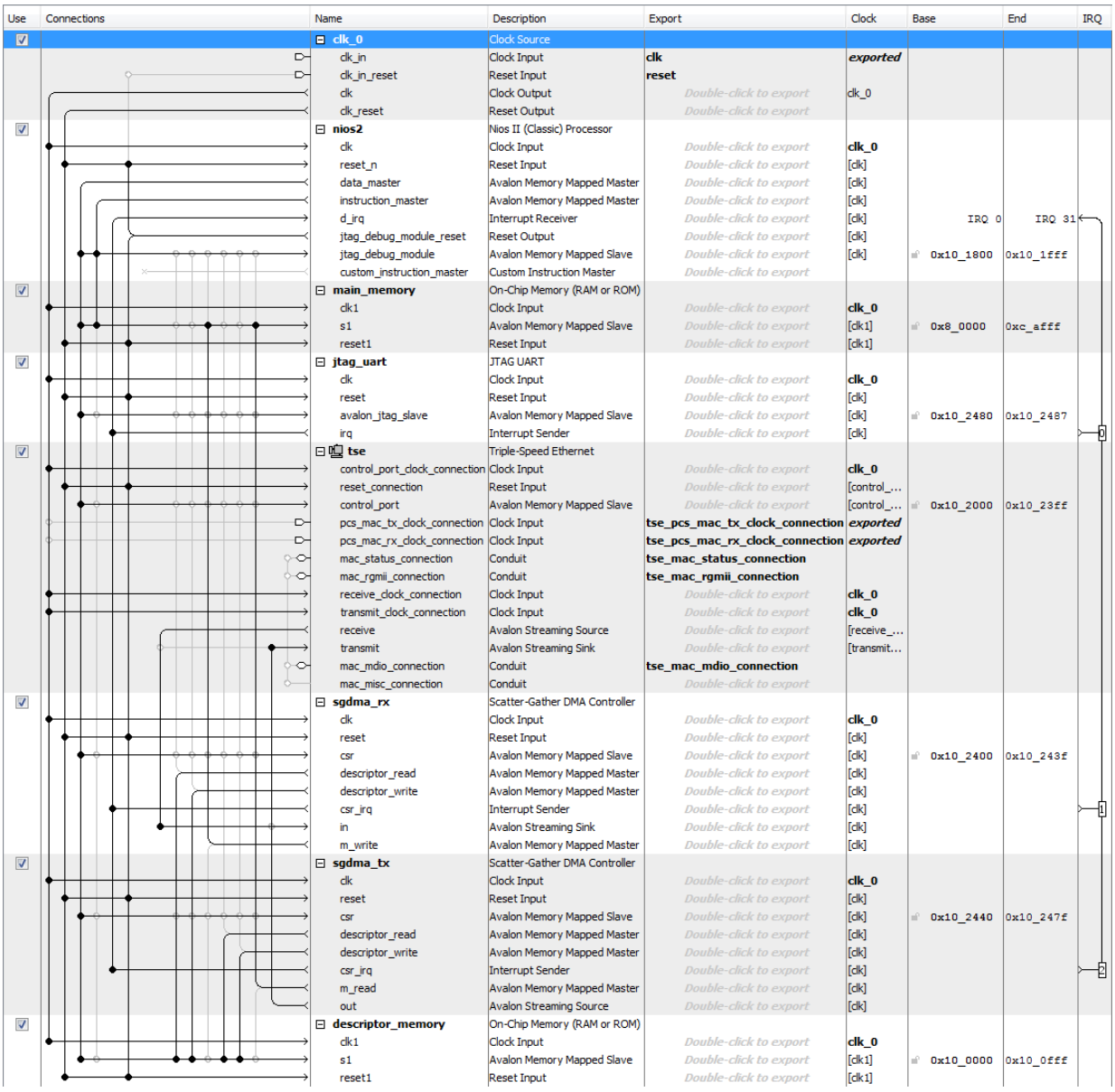

Figure 10. Nios II system with connections.

11. After you have resolved all error messages, you can generate the system.
    - Select **Generate > Generate HDL...**.
    - Uncheck the **Create block symbol file (.bsf)** in the **Synthesis** section as shown in Figure 11.
    - Click **Generate** on the bottom of the window.
    - When successfully completed, the generation process produces the message “Generate Completed”.

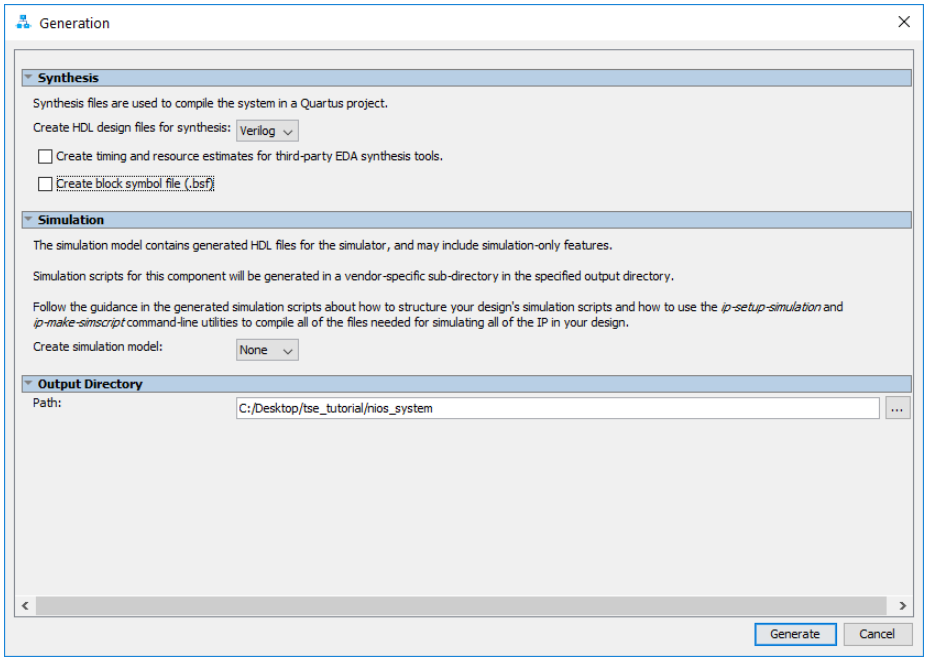

Figure 11. Settings for Generating Platform Designer.

After the generation is completed, you can have a look at the example of how to instantiate the system you built by selecting **Generate > Show Instantiation Template** as shown in Figure 12. Later you will use this template to instantiate this subsystem in your top-level module.

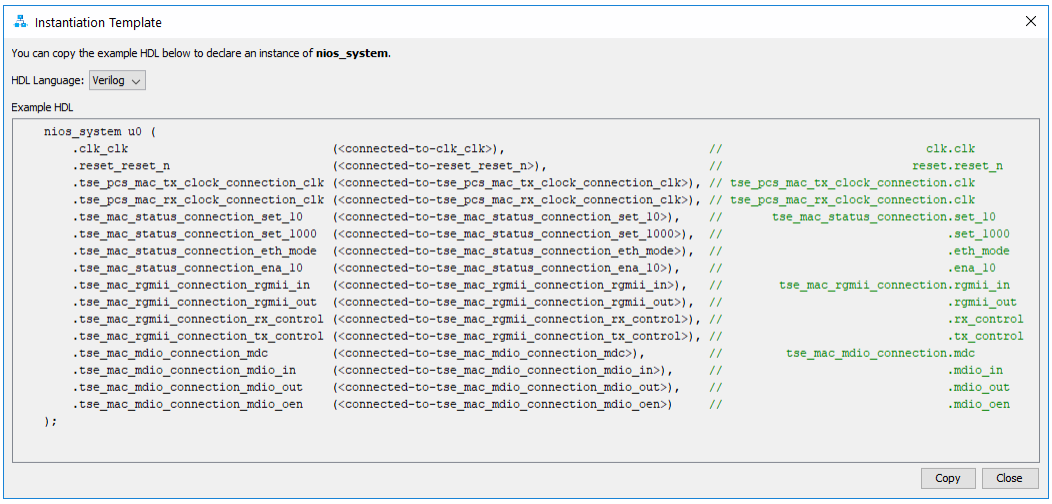

Figure 12. HDL example for the Platform Designer subsystem.

3.2 Adding Additional Modules

Besides the subsystem built using Platform Designer, you need a Phase-Locked Loop (PLL) module to generate clocks with different frequencies to make the Triple-Speed Ethernet system work properly. The PLL will take a 50 MHz input clock, and output the desired clocks. To add the PLL block, perform the following:

1. Select **Tools > IP Catalog** from the main Quartus window.
2. In the pop-up panel, select **Library > Basic Functions > Clocks; PLLs and Resets > PLL > ALTPLL** and click **Add...** button.
3. Choose the **Verilog** as the output file type for your design, and specify *my_pll.v* as the name of the output file as shown in Figure 13, and click **OK**.

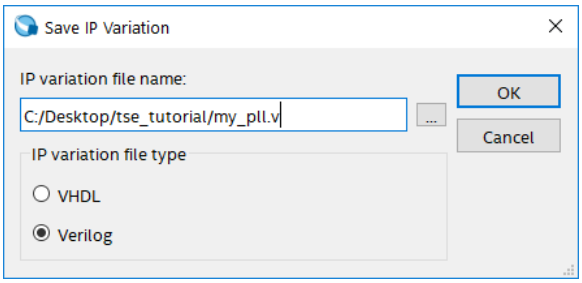

Figure 13. The IP Catalog.

4. Under the **Parameter Settings** tab, set **What is the frequency of the inclk0 input?** to 50 MHz as shown in Figure 14.

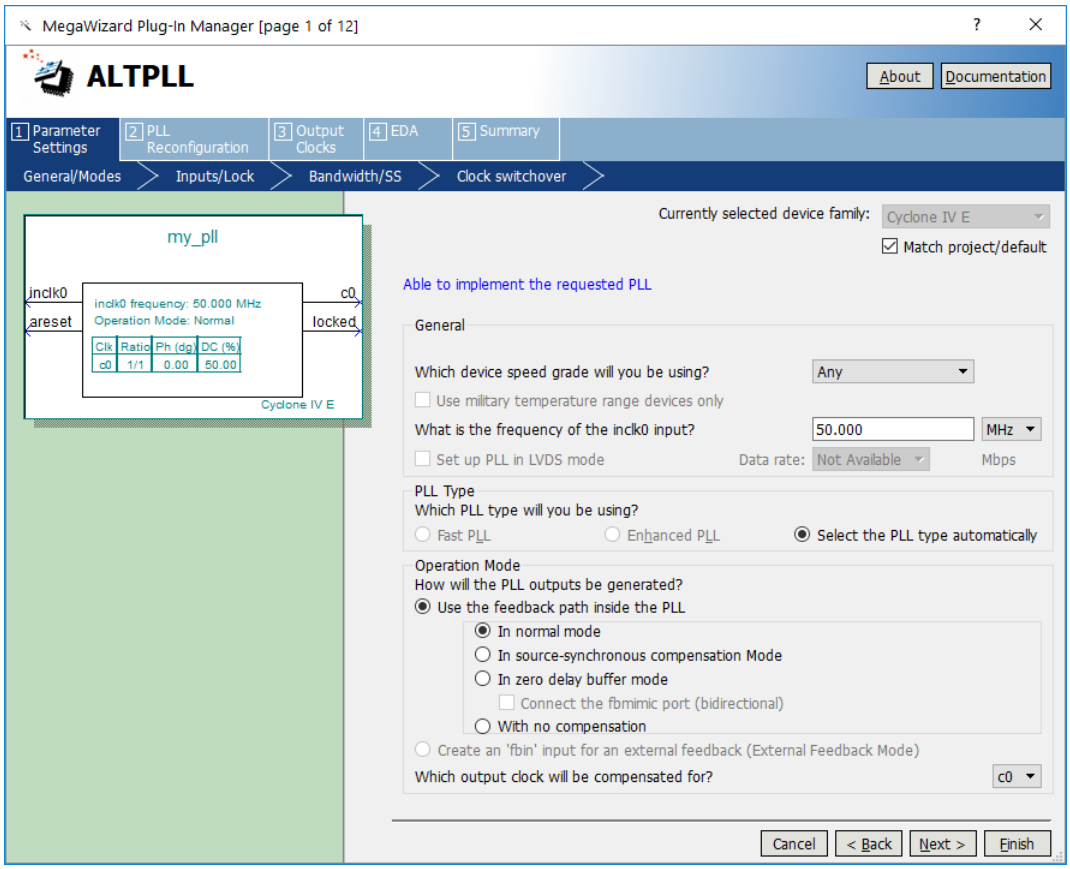

Figure 14. Settings for General/Mode section.

5. Under the **Output Clocks** tab, enter 100 MHz after selecting **Enter output clock frequency** as shown in Figure 15. This will cause the PLL to output a 100-MHz clock for the system clock *sys_clk* . Click **Next** to go to the **clk_c1** section.

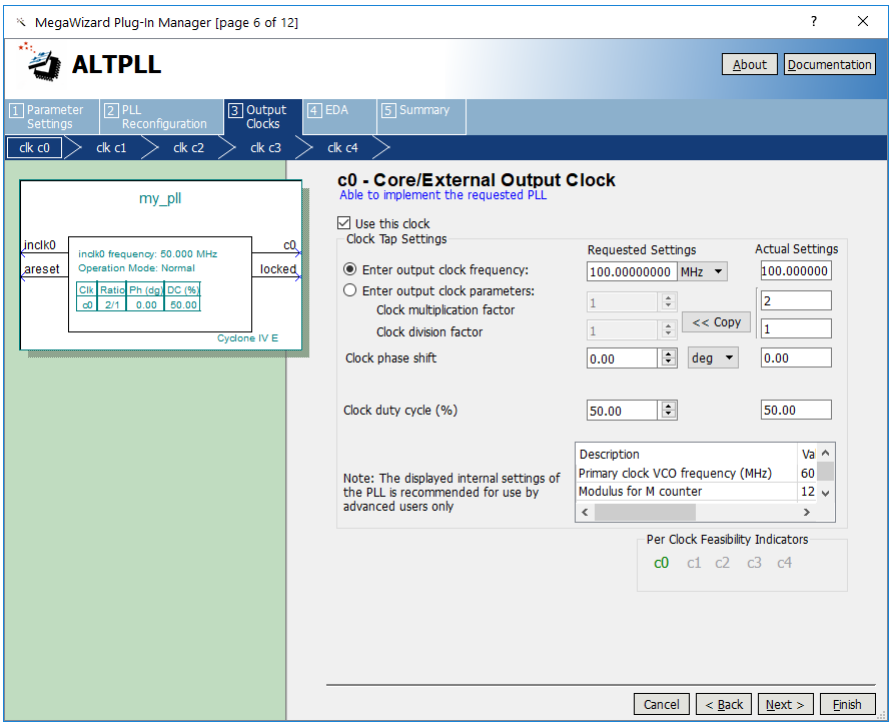

Figure 15. Settings for output c0.

6. Select the checkbox **Use this clock** and set the output clock frequency to 125 MHz using the same procedure as in the last step. This 125-MHz clock will serve as the transmission clock for the Triple-Speed Ethernet IP Core when operating in the Gigabit mode.

7. Repeat the same procedure to create a clock with 25 MHz and one with 2.5 MHz. These clocks will be used when the IP Core operates in 100 Mbps mode and 10 Mbps mode respectively.

8. Under the **Summary** tab, uncheck **my_pll_bb.v** as shown in Figure 16, and then click **Finish** to generate the files.

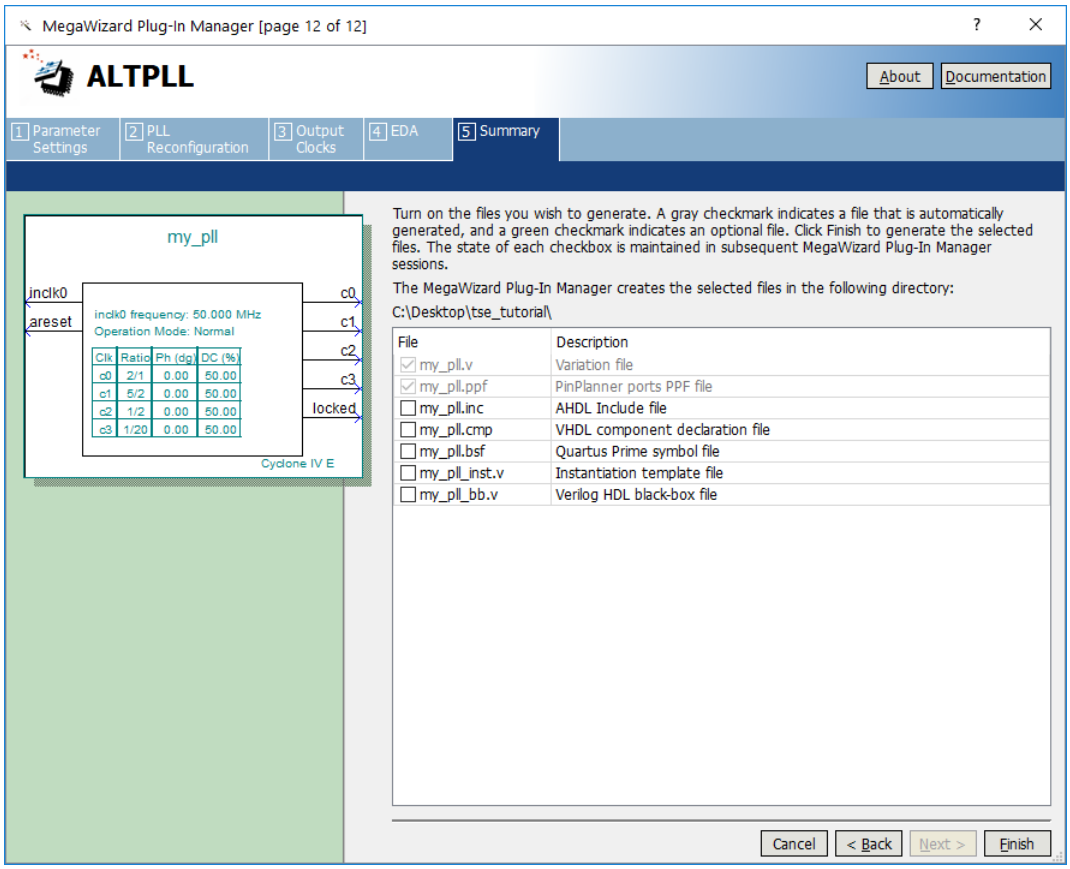

Figure 16. Settings for Summary tab.

9. If a pop-up box appears, click **Yes** to include the generated IP file in the project.

Another module needed is a DDIO_OUT register. By using the DDIO_OUT register for the transmission clock, outputting to the external PHY chip will create a very accurate edge-aligned clock-data relationship. Thus, we can maximize the timing margin on the data capture to improve the performance of the system.

1. Open the IP Catalog to create a new custom IP Core variation.
2. Select **Basic Functions > I/O > ALTDDIO_OUT** and enter *my_ddio_out.v* as the file name. Click **Next**.
3. In the window in Figure 17, set the **width** to **1** bit. Also select **Not used** for the **asynchronous clear and asynchronous set ports** options.

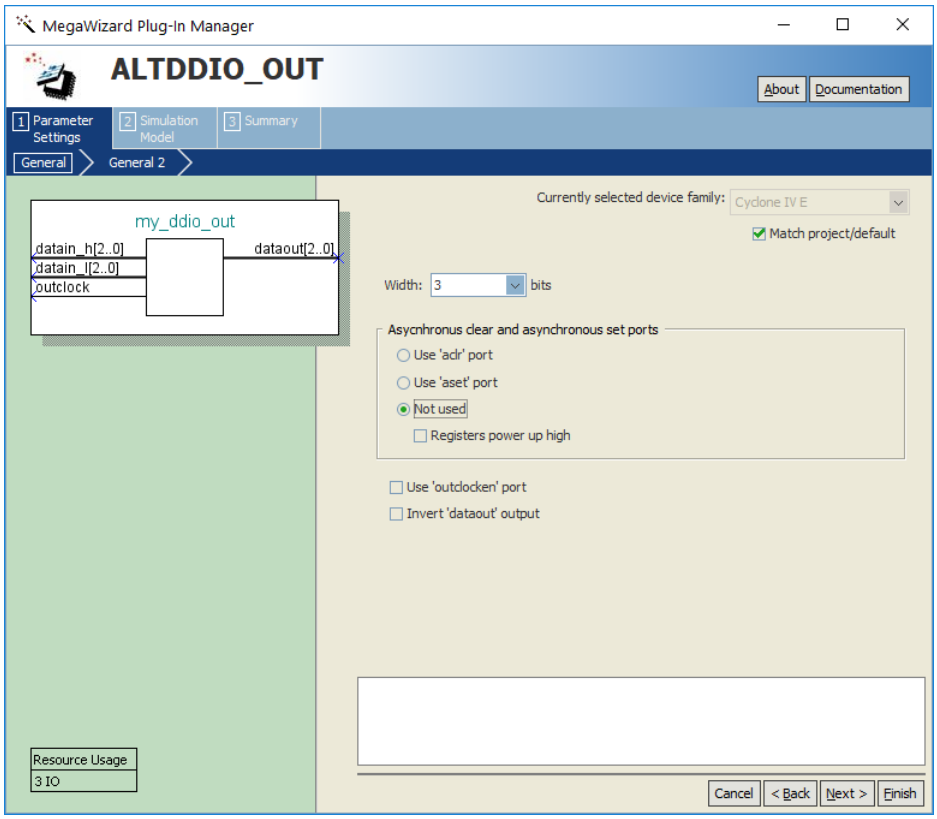

Figure 17. Settings for the DDIO_OUT register.

4. Navigate to the **Summary** tab and uncheck all the uncheckable files as shown in Figure 18. Then click **Finish**.

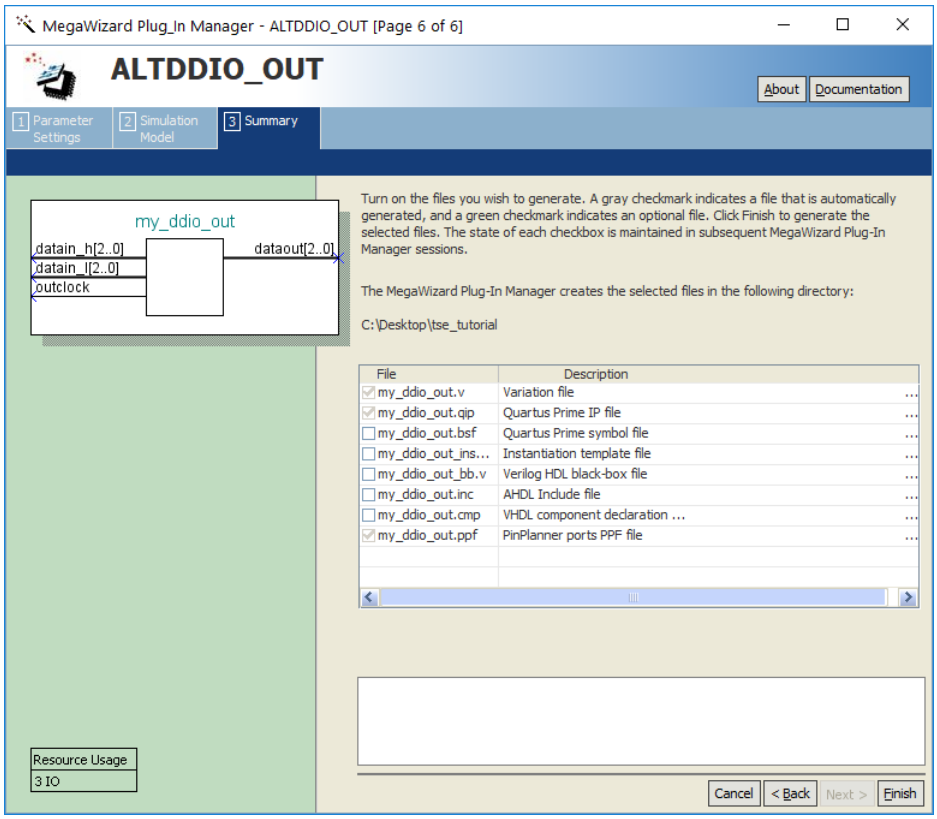

Figure 18. Settings for the Summary tab of DDIO_OUT register.

3.3 Integrating Modules into the Quartus® Prime Project

Next, you have to instantiate the nios_system and the two additional modules within the top-level module. The complete Verilog file for top-level module is provided in the design_files folder along with this tutorial. You may notice that the signals from *tse_mac_conduit* are connected to the pins related to Ethernet1, but the MDIO signals are connected to Ethernet0 as well. This is because an extra Ethernet port is required to implement a loopback functionality to mimic the actual communication between ports for the demonstration application in the next section.

To complete the hardware design, perform the following:

1. Copy the provided Verilog file *tse_tutorial.v* and the timing constraint file *tse_tutorial.sdc* to project directory. The timing constraint file is required by the TimeQuest Tool to properly evaluate the timing of your system.
2. Select **Assignments > Settings...** to add the *tse_tutorial.v*, *tse_tutorial.sdc* and *nios_system.qip* files to the Quartus Prime project. Along with the *.qip* files for the two modules generated by the IP Catalog before, all the files included in the project are shown in Figure 19.

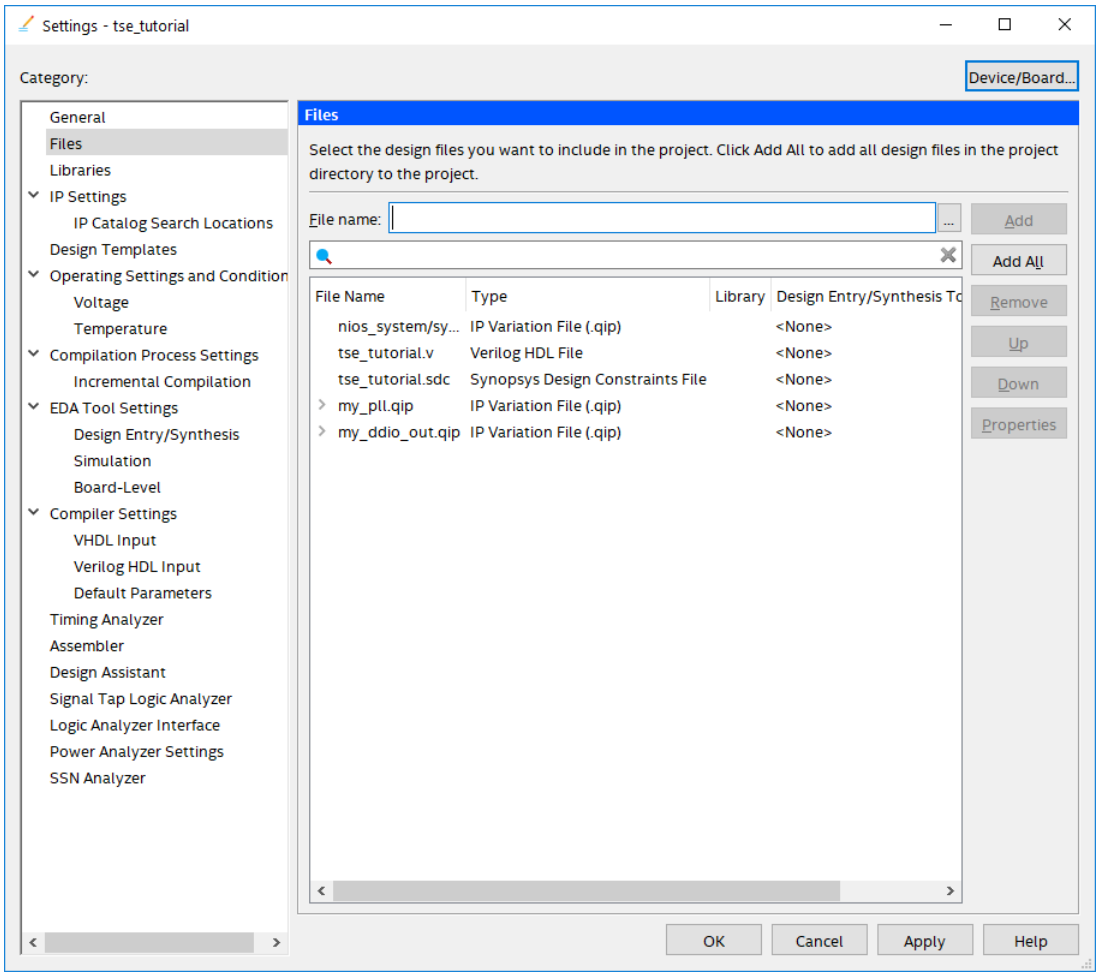

Figure 19. Settings for Files included in the Project.

3. Select **Assignments > Import Assignments...** to add the necessary pin assignments for the DE2-115 board to your project. The pin assignment file *DE2_115_pin_assignments.qsf* can be found in the design_files folder.

4. Select **Processing > Start Compilation** to compile your project in Quartus Prime software.

4 Application Program for the Ethernet System

After building the hardware system, you can now download the circuit onto the FPGA and run an application program. A demonstration application is provided with this tutorial. The program will print what the user types in the terminal window and send the message through the Ethernet port when the user presses the **Enter** key. If at any time a message is received, the received message will be printed in the terminal window.

4.1 A C-Language Demonstration Program

The C-language source file *tse_tutorial.c* is given in the design_files folder. To help you understand the source code, some sections of the code are explained below.

At the beginning of the source code, some necessary variables are declared as global variables. As shown in Figure 20, this section of the code allocates memory to store both the transmission and received frames. Also, it performs the necessary initialization for the transmission frames. The all 1's destination address means that this frame is a broadcast frame, while the source address indicates the MAC address of the source hardware device. The length of the payload data is a fixed value of 46 characters indicated by the 0x2E under the length section. A termination character '\0' is used to determine the actual end of the message sent.

```
unsigned char tx_frame[1024] = {
    0x00,0x00,                      // for 32-bit alignment
    0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,  // destination address (broadcast)
    0x01,0x60,0x6E,0x11,0x02,0x0F,  // source address
    0x00,0x2E,                      // length or type of the payload data
    '\0'                            // payload data (ended with termination char)
};

unsigned char rx_frame[1024] = { 0 };
```

Figure 20. Allocating memory for transmission and received frames.

Pay attention to the keyword **__attribute__** shown in Figure 21. By adding **__attribute__ ((section (".descriptor_memory")))** to the end of the declaration, you tell the compiler to place certain variables into the ".descriptor_memory" section. This matches the requirement of the previous Ethernet system that the descriptors for the SGDMA controllers should be put in the memory called *descriptor_memory*. Variables without this keyword will be placed along with the program code into the other memory called the *main_memory*.

```
alt_sgdma_descriptor tx_descriptor __attribute__ ((section(".descriptor_memory")));
alt_sgdma_descriptor tx_descriptor_end __attribute__
    ((section(".descriptor_memory")));

alt_sgdma_descriptor rx_descriptor __attribute__ ((section(".descriptor_memory")));
alt_sgdma_descriptor rx_descriptor_end __attribute__
    ((section(".descriptor_memory")));
```

Figure 21. Allocating memory for descriptors in the "descriptor_memory".

In the **main** function, the initialization of the SGDMA controllers is performed first. This involves opening the SGDMA devices, setting up the interrupt routine, creating a descriptor, and setting up transfers for the descriptor. This is done by using the Application Programming Interface (API) for the SGDMA controller. For more details about how to use these API functions, refer to the chapter *Scatter-Gather DMA Controller Core* in *Embedded Peripherals IP User Guide*.

The next piece of code is the process of initializing the Triple-Speed Ethernet IP Core and the external PHY chips. This code is shown in Figure 22. It is performing the following:

1. Define a pointer to the base address of the Triple-Speed Ethernet IP Core.
2. Give the Triple-Speed Ethernet IP Core its MAC address (0x0F02116E6001), which matches the source address of the transmission frame.
3. Write the PHY addresses to the IP Core for access to the specific external PHY chips using MDIO interface.
4. Write to register 20 of the first PHY chip, which is the PHY chip for the Ethernet 0 port, to set up the line loopback functionality. This will enable the Ethernet 0 port to send back all the frames it received.
5. Write to register 16 of the second PHY chip to enable automatic crossover for all modes. With this option turned on, you can use any Ethernet cable to connect the Ethernet ports rather than using a cross-over cable only.
6. Write to register 20 of the second PHY chip to set up the phase-shift delay for the input and output clocks. This is important because the Gigabit Ethernet is working at the double data rate. By shifting the input and output clocks, the clock edges will be aligned with the center of the data to ensure correct data capturing.
7. After turning the phase-shift delay option on, you need to software reset the PHY chip and then wait for the reset bit to be cleared.
8. Finally, enable read and write transfers, Gigabit Ethernet operation, and CRC forwarding. You can write different values into this register to tell the IP Core to operate at 100 Mbps or 10 Mbps mode. For example, you can write 0x00000043 for operating at 100 Mbps and write 0x02000043 for 10 Mbps. More details can be found in *Triple-Speed Ethernet IP Core Function User Guide*.

```
// Triple-speed Ethernet IP Core base address
volatile int *tse =(int *) 0x00102000;

// Initialize the MAC address
*(tse + 3) = 0x116E6001;
*(tse + 4) = 0x00000F02;

// Specify the addresses of the PHY devices to be accessed through MDIO interface
*(tse + 0x0F) = 0x10;
*tse + 0x10 = 0x11;

// Write to register 20 of the PHY chip for Ethernet port 0 to set up line loopback
*(tse + 0x94) = 0x4000;

// Write to register 16 of the PHY chip for Ethernet port 1 to enable automatic
   crossover for all modes
*(tse + 0xB0) = *(tse + 0xB0) | 0x0060;

// Write to register 20 of the PHY chip for Ethernet port 2 to set up delay for
   input/output clk
*(tse + 0xB4) = *(tse + 0xB4) | 0x0082;

// Software reset the second PHY chip and wait
*(tse + 0xA0) = *(tse + 0xA0) | 0x8000;
while (*(tse + 0xA0) & 0x8000)
    ;

// Enable read and write transfers, gigabit Ethernet operation, and CRC forwarding
*(tse + 2) = *(tse + 2) | 0x0000004B;
```

Figure 22. Initialization Process for Triple-Speed Ethernet and external PHY chips

After you enable the read and write transfers, the Triple-Speed Ethernet IP Core will start working. Then the main loop is entered which sends data typed by the user and prints messages that are received. The last part of the code is the interrupt routine. Whenever an interrupt is raised, the interrupt routine will be executed to:

- Erase the user's current message from the screen.
- Print the received message.
- Reprint the user's message that was erased.

4.2 Running the Application Program

In this section we will use the *Intel FPGA Monitor Program*, provided by the Intel FPGA University Program for use with the DE-series boards. The Monitor Program provides a simple means for compiling, assembling and

downloading of programs onto a DE-series board. It also makes it easy for the user to perform debugging tasks. A description of this software is available in the *Intel FPGA Monitor Program* tutorial.

To run the demonstration application program above, perform the following steps:

1. Connect the DE2-115 board to the host computer and make sure it is powered on.
2. Create a new subdirectory named *app_software* within the *tse_tutorial* project directory. Copy the provided file *tse_tutorial.c* into this new directory.
3. Open the Monitor Program and create a project named *tse_tutorial* in the above directory.
4. Select **Nios II** as the architecture. Click **Next**.
5. In the window in Figure 23, select **Custom System** as the system type and browse to find the appropriate *.sopcinfo* and *.sof* files. Select **Not Required** for the preloader. Click **Next**.

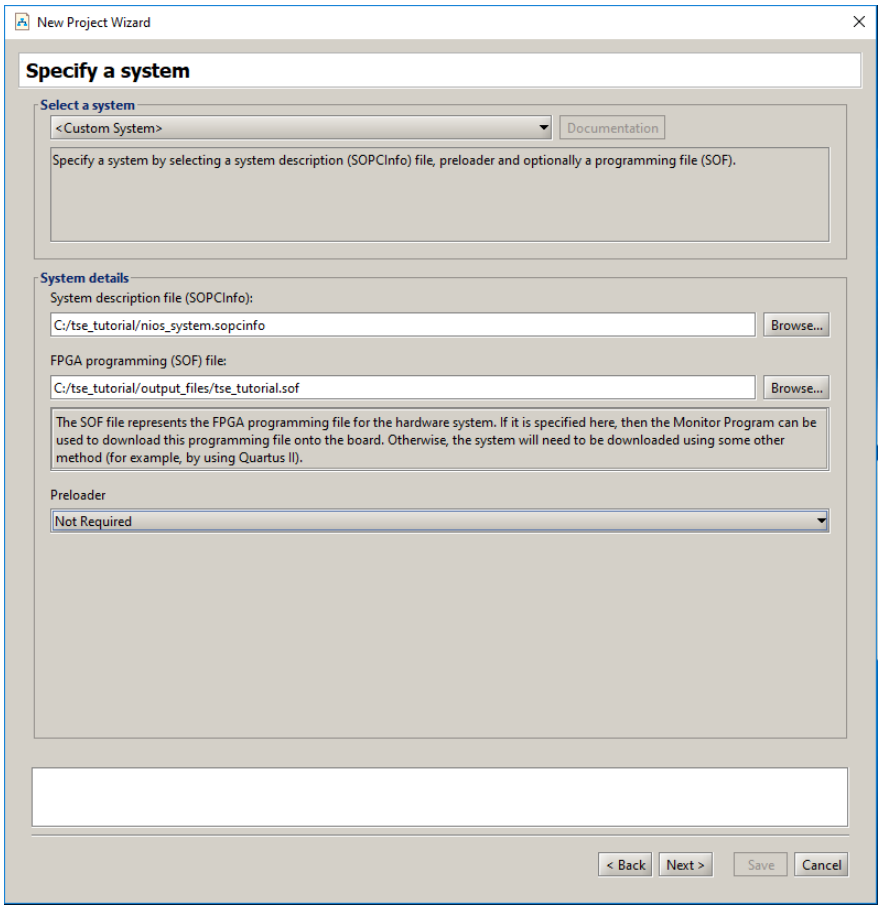

Figure 23. Specifying a system.

6. In the next window select **Program with Device Driver Support** as the program type. Click **Next**.

7. Then add the *tse_tutorial.c* file and click **Next**.
8. In the window in Figure 24, if the physical connection exists between the host computer and your DE2-115 board you should see **USB-Blaster [USB-0]** under the Host Connection drop-down list. Otherwise, you should check the connection and make sure that the USB-Blaster driver is correctly installed. Then click **Save**.

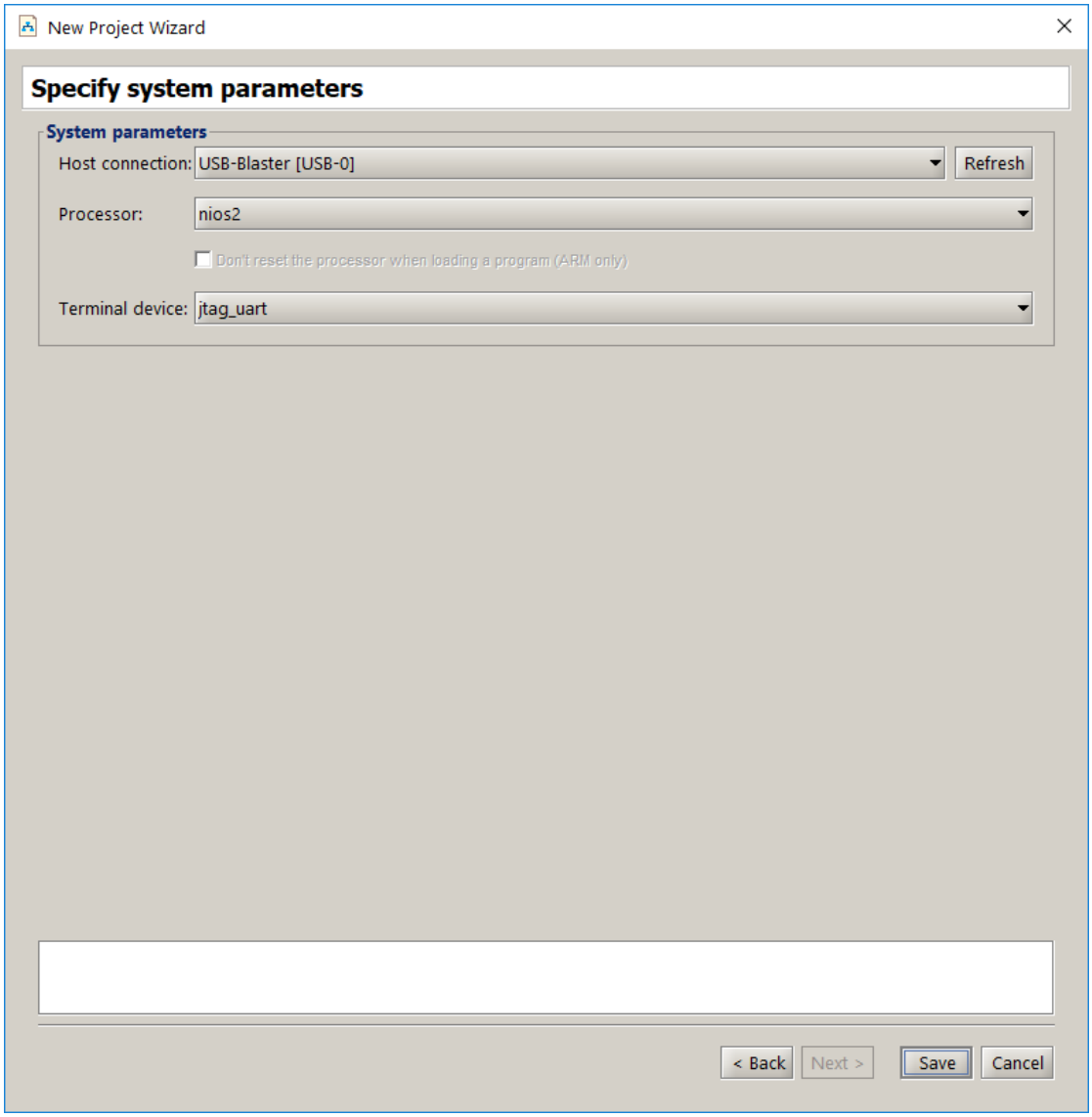

Figure 24. Check physical connection between the host computer and the DE2-115 board.

9. In the pop-up window in Figure 25, click **Yes** to download the system to the DE2-115 board.

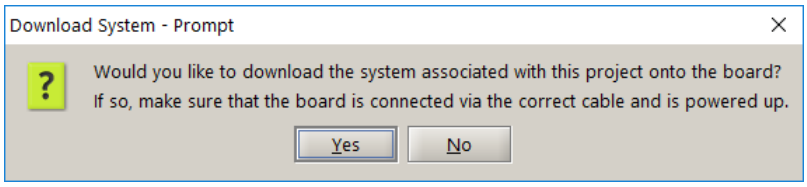

Figure 25. Pop-up window to download the system to the board.

10. Click **Actions > Compile & Load** to compile the source code of your program. If it's your first time compiling the code, it will take a while because the Monitor Program will generate the necessary files for the API functions of SGDMA controllers.

11. Then click **Actions > Continue** to start running the program on the board. You should see the message on the Monitor Program terminal screen as shown in Figure 26.

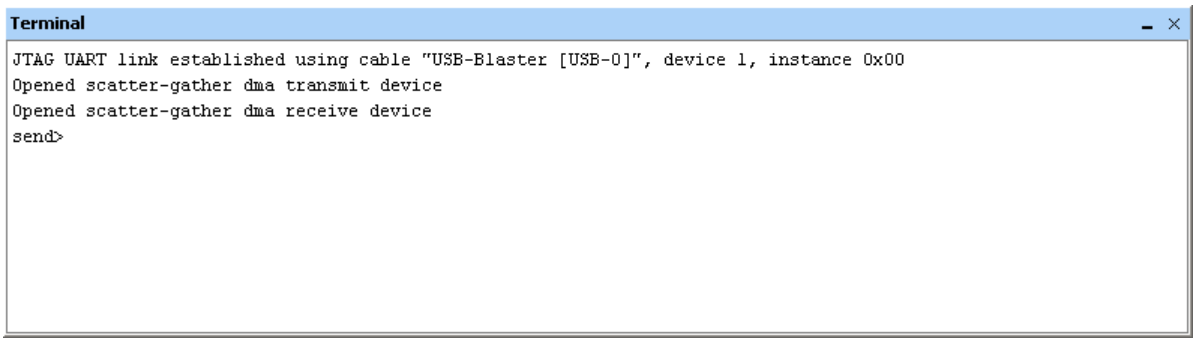

Figure 26. Running the application program by using the Monitor Program.

12. To make the demonstration program work properly, connect both Ethernet ports of the DE2-115 with an Ethernet cable. After connecting the ports, check the green LEDs besides the PHY chips to see in which mode the PHY chips are operating. Make sure the Triple-Speed Ethernet IP Core is operating in the same mode as the PHY chips.

13. Place the cursor in the terminal screen after the **send>** label and type a message, then press the **Enter** key to transmit the message. The message will be transmitted from the Ethernet1 port to the Ethernet0 port. Since the Ethernet0 port is set to perform a loopback, the message is transmitted back to the Ethernet1 port. Then, the message you type should be printed on the terminal screen after the label **receive>**. An example is shown in Figure 27.

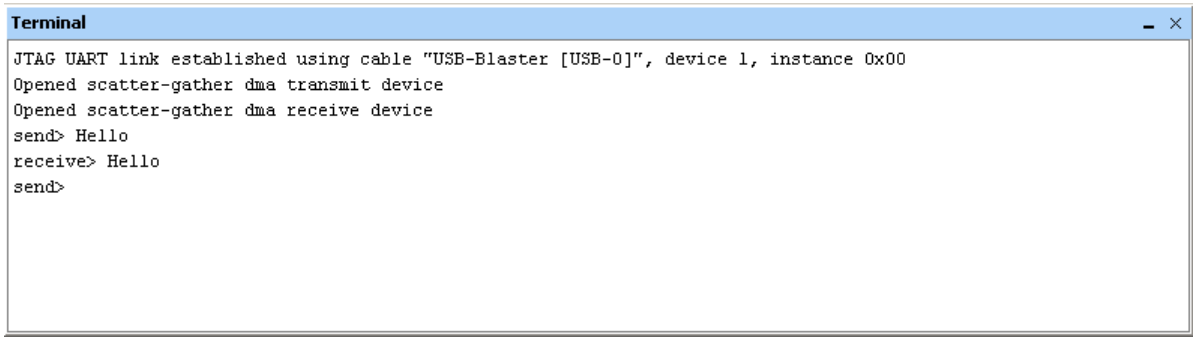

Figure 27. Sending messages.

5 Concluding Remarks

This tutorial shows how to build a Triple-Speed Ethernet system and run a simple application program on it. In most applications, using the Ethernet allows the system to communicate with any other device that supports Ethernet standard. While these details are beyond the scope of this tutorial, completing the tutorial provides the basic knowledge. If you are going to connect your system to some other devices, the next step should be to write software for the Nios II processor to handle the necessary protocols of higher layers.

Copyright © Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Avalon, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

*Other names and brands may be claimed as the property of others.