University at Buffalo
The State University of New York

EE450/550: Networked Systems Design
Lab. Assignment 3: Ethernet implementation using the Nios II in the DE2i-150

Given: October 3, 2016
Laboratory sessions: 2 hours (2 weeks)
Assignment Report Due: October 23, 2015 (11:59 PM)

Instructions:

1. Follow this guide step by step. You can access this document from your computer in the lab. You do not need to print it.
2. Show your progress to the instructor when indicated in this guide. **Make sure that you understand what you have done and be ready to answer questions.** Your grade in the checkpoints will depend on that.
3. Make sure you answer all the questions. Include supporting screenshots when needed.
4. Submit your assignment report in electronic format directly on UB Learns

Elena Bernal Mor, Ph.D.
Lecturer
Department of Electrical Engineering
University at Buffalo, The State University of New York
Office: 228 Davis Hall
E-mail: elbermo@buffalo.edu

This guided assignment provides a basic introduction to the usage of the Triple-Speed Ethernet IP Core on Altera's DE2i-150 board. First, it demonstrates how to build a Nios II hardware system with the Triple-Speed Ethernet IP Core using Qsys. Then, it shows how to create a software program that runs on the Nios II system which interfaces with the Marvell 88E111 Ethernet PHY chip and other components of the DE2i-150 board. You will run a packet analyzer, that is an application program that analyzes the received packets to your Ethernet port. This guide is a good starting point to understand an Ethernet standard implementation.

1. Introduction

1.1 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 OSI model: the Data Link layer and the Physical layer. 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 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).

The DE2i-150 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 Altera 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.

1.2 What you will learn in this guided assignment

The design example you will build in this guided assignment demonstrates a small Nios II system that can communicate with a host computer, allowing the host computer to control the Ethernet port and other logic inside the FPGA.

The Nios II standard hardware system provides the following necessary components:

- Nios II processor core, that’s where the software will be executed
- On-chip memory to store and run the software
- JTAG link for communication between the host computer and target hardware (typically using a USB-Blaster cable)
- LED peripheral I/O (PIO) pins, be used as indicators
- Switch peripheral I/O (PIO) pins, be used as system controllers
- Triple-Speed Ethernet IP core, be used as a media access controller
- Scatter-Gather Direct Memory Access (SG-DMA) controllers, be used to transfer data from the on-chip memory to the Triple-speed Ethernet IP core and vice versa.

When you complete this guided assignment you will understand how the Marvell 88E1111 Ethernet PHY chip interacts with the Triple-Speed Ethernet IP core and how received packets are processed in the Nios II system. You will also have a hands-on experience connecting different elements of a network to transfer packets through them.

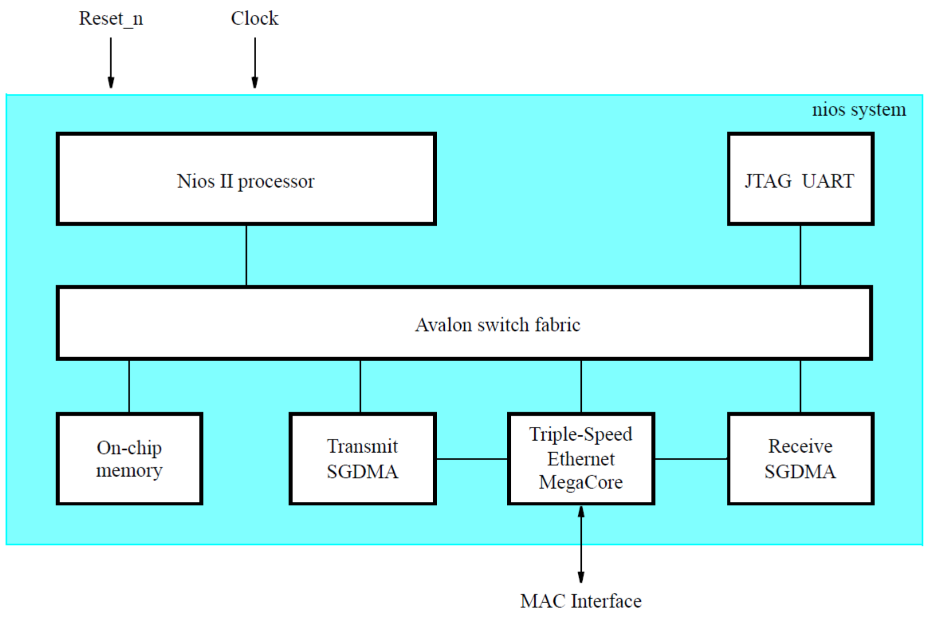
2. Implementing a Triple-Speed Ethernet System

This section describes how to implement the Triple-Speed Ethernet system.

To start, launch Quartus II and create a New Project Wizard. Choose an appropriate workspace in a new directory for this project (create a new folder). Choose Cyclone IV GX as device family and EP4CGX150DF31C7 as device. Choose NONE for simulation.

2.1 Creating the Nios II System

Most of the Triple-Speed Ethernet system can be built as a subsystem using the Qsys tool. The block diagram of this subsystem is shown above:



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.

Besides those components, we will also include some PIO ports that will be assigned to a LED and to a switch button in order to add more functions to our packet analyzer.

First of all check that your board is configured in **RGMII mode**. For that, check that the jumper for switching work modes from RGMII to MII has Pins 1 and 2 shorted.

To design the desired system, you should use the Qsys 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 > Qsys to open the Qsys 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 (100 MHz).** 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 Processor and click Add.
   - Choose Nios II/f.
   - Under the Advanced Features tab, disable the "Include reset_req signal for OCI RAM and Multi-Cycle Custom Instructions" setting by unchecking the checkbox.
   - 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.

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) and click Add.
   - Set the Total Memory Size to 307200 bytes (300 KBytes).
   - 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.

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 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 > Triple-Speed Ethernet.
   - 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**, and **Include MDIO module (MDC/MDIO)**.
   - 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.

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.
   - Select Stream To Memory as the transfer mode and 6 as the sink error width.
   - 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.

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.
   - 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.

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) 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. Add a PIO (Parallel I/O) that will be assigned to a LED

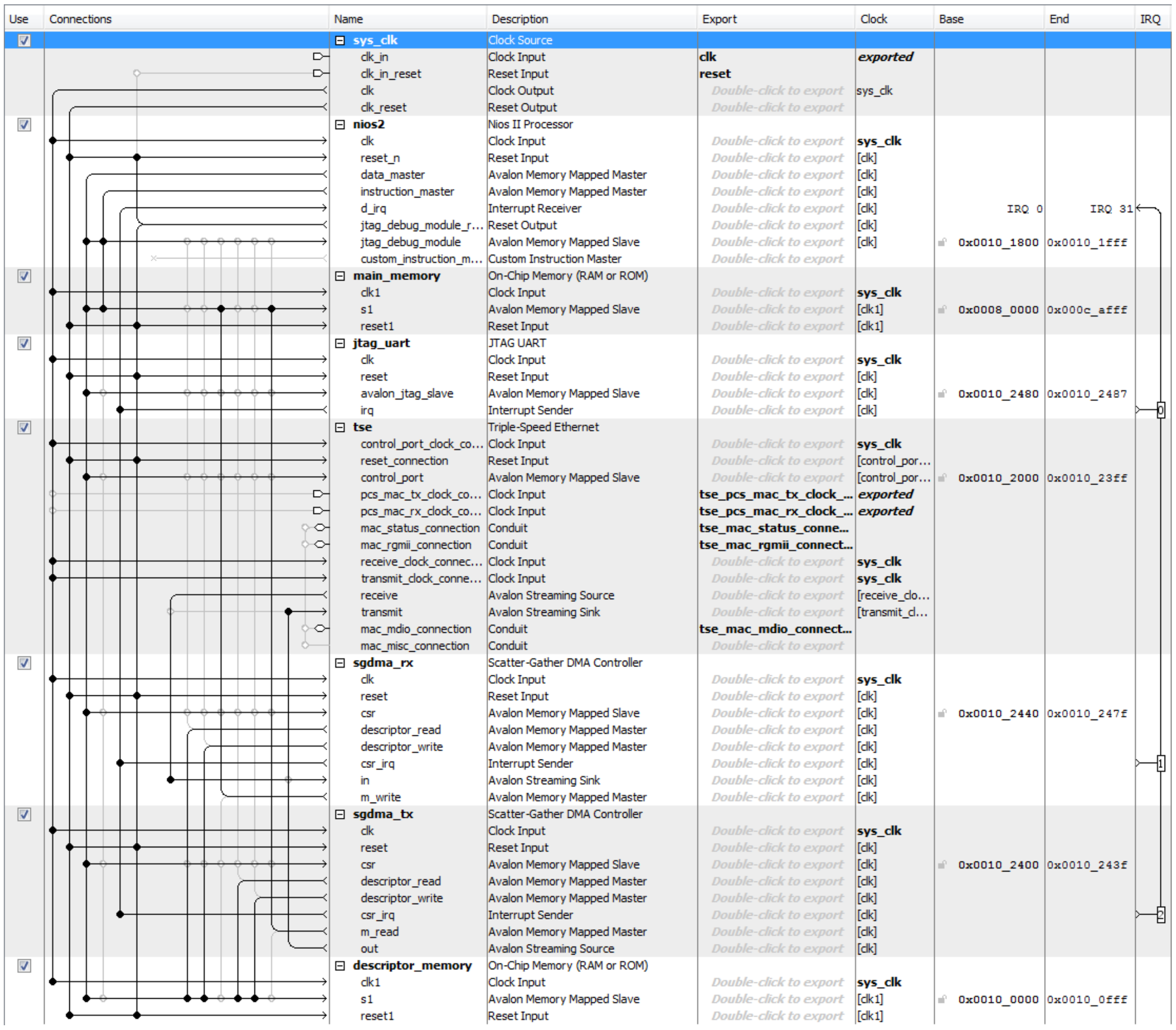
    - Select Processors and Peripherals > Peripherals > PIO (Parallel I/O). Choose 1 as width and output as Direction.
    - Rename pio_0 to led.
    - Connect the clk and reset port of the PIO to the clk and clk_reset of the clock source and to the jtag_debug_module_reset.
    - Connect the data_master port of the Nios II processor to the s1 port of the PIO.
    - To export the external_connection, double click in the column “export” and rename it to led.

11. Add a PIO (Parallel I/O) that will be assigned to a switch button

    - Select Processors and Peripherals > Peripherals > PIO (Parallel I/O). Choose 1 as width and input as Direction.
    - Rename pio_0 to switch.
    - Connect the clk and reset port of the PIO to the clk and clk_reset of the clock source and to the jtag_debug_module_reset.
    - Connect the data_master port of the Nios II processor to the s1 port of the PIO.
    - To export the external_connection, double click in the column “export” and rename it to switch.

12. 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 below.



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		⊟ **sys_clk**	Clock Source					
		clk_in	Clock Input	**clk**	***exported***			
		clk_in_reset	Reset Input	**reset**				
		clk	Clock Output	*Double-click to export*	sys_clk			
		clk_reset	Reset Output	*Double-click to export*				
✓		⊟ **nios2**	Nios II Processor					
		clk	Clock Input	*Double-click to export*	**sys_clk**			
		reset_n	Reset Input	*Double-click to export*	[clk]			
		data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		d_irq	Interrupt Receiver	*Double-click to export*	[clk]	IRQ 0	IRQ 31	
		jtag_debug_module_r…	Reset Output	*Double-click to export*	[clk]			
		jtag_debug_module	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0010_1800	0x0010_1fff	
		custom_instruction_m…	Custom Instruction Master	*Double-click to export*				
✓		⊟ **main_memory**	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	*Double-click to export*	**sys_clk**			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	0x0008_0000	0x000c_afff	
		reset1	Reset Input	*Double-click to export*	[clk1]			
✓		⊟ **jtag_uart**	JTAG UART					
		clk	Clock Input	*Double-click to export*	**sys_clk**			
		reset	Reset Input	*Double-click to export*	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0010_2480	0x0010_2487	
		irq	Interrupt Sender	*Double-click to export*	[clk]			0
✓		⊟ **tse**	Triple-Speed Ethernet					
		control_port_clock_co…	Clock Input	*Double-click to export*	**sys_clk**			
		reset_connection	Reset Input	*Double-click to export*	[control_por…			
		control_port	Avalon Memory Mapped Slave	*Double-click to export*	[control_por…	0x0010_2000	0x0010_23ff	
		pcs_mac_tx_clock_co…	Clock Input	**tse_pcs_mac_tx_clock_…**	***exported***			
		pcs_mac_rx_clock_co…	Clock Input	**tse_pcs_mac_rx_clock_…**	***exported***			
		mac_status_connection	Conduit	**tse_mac_status_conne…**				
		mac_rgmii_connection	Conduit	**tse_mac_rgmii_connect…**				
		receive_clock_connec…	Clock Input	*Double-click to export*	**sys_clk**			
		transmit_clock_conne…	Clock Input	*Double-click to export*	**sys_clk**			
		receive	Avalon Streaming Source	*Double-click to export*	[receive_clo…			
		transmit	Avalon Streaming Sink	*Double-click to export*	[transmit_cl…			
		mac_mdio_connection	Conduit	**tse_mac_mdio_connect…**				
		mac_misc_connection	Conduit	*Double-click to export*				
✓		⊟ **sgdma_rx**	Scatter-Gather DMA Controller					
		clk	Clock Input	*Double-click to export*	**sys_clk**			
		reset	Reset Input	*Double-click to export*	[clk]			
		csr	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0010_2440	0x0010_247f	
		descriptor_read	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		descriptor_write	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		csr_irq	Interrupt Sender	*Double-click to export*	[clk]			1
		in	Avalon Streaming Sink	*Double-click to export*	[clk]			
		m_write	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
✓		⊟ **sgdma_tx**	Scatter-Gather DMA Controller					
		clk	Clock Input	*Double-click to export*	**sys_clk**			
		reset	Reset Input	*Double-click to export*	[clk]			
		csr	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0010_2400	0x0010_243f	
		descriptor_read	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		descriptor_write	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		csr_irq	Interrupt Sender	*Double-click to export*	[clk]			2
		m_read	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		out	Avalon Streaming Source	*Double-click to export*	[clk]			
✓		⊟ **descriptor_memory**	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	*Double-click to export*	**sys_clk**			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	0x0010_0000	0x0010_0fff	
		reset1	Reset Input	*Double-click to export*	[clk1]			

13. After you have resolved all error messages, you can generate the system.

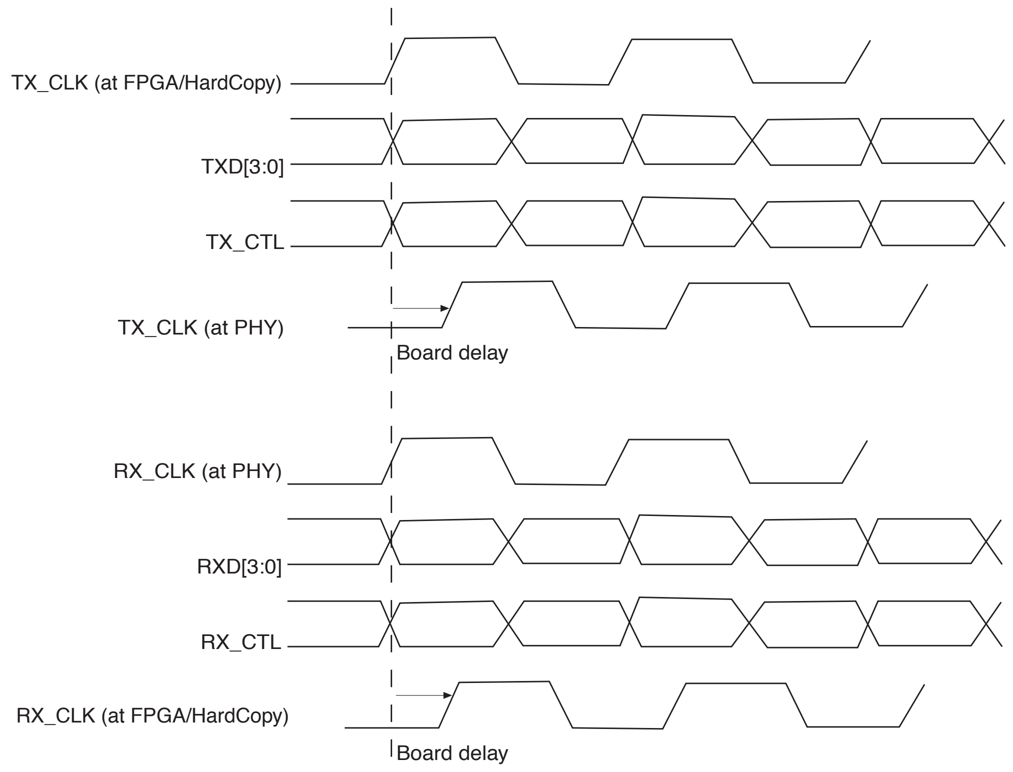
- Select Generate > Generate HDL....
- Click Generate on the bottom of the window.

After the generation is completed, you can have a look at the example of how to instantiate the system you built by selecting Generate > HDL Example.... Later you will use this template to instantiate this subsystem in your top-level module.

2.2 Adding Additional Modules

When RGMII is implemented, data is sampled on both edges of the clock. If the clock and data are generated simultaneously, that is, edge-aligned, the clock must be routed with an added trace delay on the external PHY chip in order to capture the data correctly in the receiver.

Timing Diagram for RGMII



For data and clock transmission, some PHY chips that support RGMII offer an option to add delay to the transmit or receive clock. When the option to delay the TX_CLK inside the PHY chip is enabled, the MAC function in the FPGA must generate a clock that is edge-aligned with the data and waveform. In this case, the PHY chip shifts the clock as necessary to capture the data. When this option is disabled, the FPGA (where the MAC function is implemented) 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 the Figure above. This shifted clock is the RGMII transmit clock and is used by the PHY chip to capture the data in the receiver.

Actually, our Ethernet PHY chip does not delay the transmit clock (it is disabled). Therefore, our design must generate a clock for the RGMII transmit clock used by the Ethernet PHY chip (ENET_GTX_CLK) that is delayed with respect to the data (90º shift).

If the option to add delay to the transmit clock were possible and enabled in the PHY chip, an IP core called “my_ddio_out” would be used to generate a clock for the PHY chip (ENET_GTX_CLK) that is edge-aligned with the data.

Then, besides the subsystem built using Qsys, you need:

1. A Phase-Locked Loop (my_pll) module to generate clocks with different frequencies to make the Triple-Speed Ethernet system (which implements the MAC function) work properly at 10/100/1000 Mbps.
2. A Phase-Locked Loop (pll_clocks_PHY) module to generate 90º shifted clocks with different frequencies used by the Ethernet PHY chip (RGMII transmit clock) at 10/100/1000 Mbps.

First, create the first PLL named “my_pll”. It will take a 50 MHz input clock, and output the desired clocks. You need a 100 MHZ clock for the clock system, a 2.5 MHz clock for 10 Mbps Ethernet, a 25 MHz clock for 100 Mbps Ethernet and a 125 MHz clock for Gigabit Ethernet. We will use the output port of the PLL that indicates whether the PLL is locked or not as a reset signal for our Nios II system. To add the PLL block, perform the following:

1. In the IP Catalog, select Library > Basic Functions > Clocks; PLLs and Resets > PLL > ALTPLL and click Add... button.
2. Choose the Verilog as the output file type for your design, and specify *my_pll.v* as the name of the output file and click OK.
3. Do not disable any box if it is not specifically said in this guide.
4. Under the Parameter Settings tab, set the frequency of the inclk0 input to 50 MHz
5. Under the Output Clocks tab, select *Enter output clock frequency* and enter 100 MHz. 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.
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, and then click Finish to generate the files.
9. If a pop-up box appears, click Yes to include the generated IP file in the project.

Now, create the other PLL named “pll_clock_PHY”. It will take a 50 MHz input clock, and output the desired 90º shifted clocks. You do not need to shift the 100 MHZ clock system. You need 90º shifted clocks with 2.5 MHz, a 25 MHz and 125 MHz. We will not use the output port of the PLL that indicates whether the PLL is locked or not. To add the PLL block, perform the following:

1. In the IP Catalog, select Library > Basic Functions > Clocks; PLLs and Resets > PLL > ALTPLL and click Add... button.
2. Choose the Verilog as the output file type for your design, and specify *pll_clock_PHY.v* as the name of the output file and click OK.

3. We will not use the output pin that indicates whether the PLL is locked or not.
4. Under the Parameter Settings tab, set the frequency of the inclk0 input to 50 MHz
5. Under the Output Clocks tab, select *Enter output clock frequency* and enter 125 MHz. and 90º shift. Click Next to go to the clk_c1 section.
6. Select the checkbox Use this clock and set the output clock frequency to 25 MHz and 90º shift.
7. Repeat the same procedure to create a clock with 2.5 MHz and 90º shift.
8. Under the Summary tab, uncheck my_pll_bb.v, and then click Finish to generate the files.
9. If a pop-up box appears, click Yes to include the generated IP file in the project.

Checkpoint! Show your current status to the instructor (20%)

2.3 Integrating Modules into the Quartus II Project

Next, you have to instantiate the nios_system and the two additional modules within the top-level module. A template Verilog file for top-level module is provided in UBlearns along with this guided assignment. This top_level module will be controlling the signals sent to the Gigabit Ethernet Transceiver, so open the system manual and go to the corresponding section.

To complete the hardware design, perform the following:

1. Copy the template file inside your project folder and change its name to the same name of your project. Change the name of the module also.
2. Fill the gaps on the port declaration taking into account the connection setup between the Gigabit Ethernet external PHY chip and the FPGA (go to the manual).
3. As you can see, after the port declaration there are several variable declarations that will be used for the different output clocks of the PLL's, reset and outputs of the Nios II system generated with Qsys.
4. Next, the MDIO and MDC signals are obtained.

   - First, the wire *mdio_in* is assigned the value of the management data inout signal. The wire *mdio_in* will be used as an input in the Nios II instance.
   - Second, the management data clock is assigned the value of the wire *mdc*. The wire *mdc* will be obtained from the Nios II system instance.
   - Finally, the management data inout signal, *ENET_MDIO*, is assigned a value of "z" if the control signal *mdio_oen* is 1. Otherwise *ENET_MDIO* is *mdio_out*. The signals *mdio_oen* and *mdio_out* are obtained from the Nios II instance.

5. You can see that the system reset is assigned to a wire called *core_reset*_n. This wire will be obtained from the PLL *my_pll* as it was explained above.
6. Instance the PLL, *my_pll*.

   - Use as input signal reset the inverse signal given by the push-button KEY, and for the input clock the 50 MHz clock of the board

- The output clock signals correspond to the different clocks that we need for the FPGA, where the MAC function is implemented in the Triple Speed Ethernet IP core (remember which output corresponds to each clock)
- The output locked signal is used as a reset, so it will correspond to the core_reset_n variable

7. Instance the PLL, "pll_clock_PHY".

    - Use as input signal reset the inverse signal given by the push-button KEY, and for the input clock the 50 MHz clock of the board
    - The output clock signals correspond to the three clocks shifted 90º, which are used in the external PHY chip

8. Next we have to decide which clock to assign to the transmit clock in the Triple Speed Ethernet IP core. We will use here an intermediate variable tx_clk. For that, we check two signals that we are obtained from the Nios II system (qsys). This two signals are *eth_mode* and *ena_10*. These two signals indicate:

    - *eth_mode*=1 → Gigabit Ethernet
    - *eth_mode*=0 and ena_10=1 → 10Mbps Ethernet
    - *eth_mode*=0 and ena_10=0 → 100Mbps Etherne

    Fill the gaps in the continuous assignment to do:

    - *eth_mode=1? → tx_clk=clk_125; eth_mode=0? → (ena_10=1? → tx_clk=clk_2p5; ena_10=0 → tx_clk=clk_25*)

9. Next we have to decide which clock to assign to the transmit clock in the Ethernet PHY chip, which is the RGMII transmit clock ENET_GTX_CLK. For that, we check again *eth_mode* and *ena_10*. Fill the gaps in the continuous assignment to do:

    - *eth_mode=1? → ENET_GTX_CLK=clk_125; eth_mode=0? → (ena_10=1? → ENET_GTX_CLK=clk_2p5; ena_10=0 → ENET_GTX_CLK =clk_25)*

10. Finally, instance the Nios II system (obtained with qsys). Read the comments in the template file to assign correctly the different ports.

Once you have completed your top_level file you can add the necessary files to your project to complete the hardware design. Perform the following steps:

1. If you have not added your top_level file in the Quartus II project yet, add it now.
2. Add to the Quartus II project the .qip file generated by Qsys.
3. Add the .qip files for the two modules generated by the IP Catalog before, i.e., *my_pll.qip* and *pll_clocks_PHY.qip*.

4. Add also to the Quartus II project an .sdc file. You can use the same than in other assignments (use the system clock). Remember to change the name of this file. It must have the same name as your project.

2.4 Pin Assignment, compilation and download of the Hardware Design to the FPGA

Now you can assign the FPGA pin locations. Remember that before making pin assignments, you have to perform the analysis & elaboration. After analysis & elaboration completes, you can use the pin assignment information in the “User Manual” in UBlearns to find the location of the pins for all the pins that you need.

Set the unused pins to “As input tri-state”. Now that you have created a complete Quartus II project and entered all pin assignments, you can compile the design.

Download the configuration file (i.e. the SRAM Object File (.sof) that contains the Nios II standard system) to the board.

When .sof file is successfully downloaded to the FPGA, a window pops up. Leave the Quartus II Programmer open in the background while you are using your Nios II system.

At this point, the Nios II system is configured and running in the FPGA, but it does not yet have a program in memory to execute.

Checkpoint! Show your current status to the instructor (20%)

3. 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. The software program that we will develop will show the received frames from the Ethernet port of the board in the terminal window. That is, we will program a frame analyzer.

We need to connect the Ethernet port of the DE2i-150 board to a device that can generate frames, i.e., a computer in our case. We can use switches or crossover cables to properly connect our boards to the computers. Check what option is available in your workstation and connect the Ethernet port of your board to a computer using a switch or a crossover cable. In the case you are using a switch, remember that several boards can be connected to the same computer.

Our application will work as follows. The Ethernet chip will be configured to work in promiscuous mode. If the switch button you chose in your hardware design is up, the terminal window will show the source and destination addresses of the received frames. Moreover the LED that you chose in your hardware design will be switched on. If the switch button is down, no new messages are shown in the terminal window and the LED is switched off.

3.1 A C-Language Demonstration Program

Create a new folder named “software” inside your project folder. A template of the C-language source file is given in UBlearns. Copy this file in the “software folder”.
Create a new NIOS II C/C++ application project with a blank file. To begin, perform the following steps in the NIOS II SBT for Eclipse:

1. Go to the NIOS II Software Build Tools for Eclipse.
2. Choose File > New > Nios II Application and BSP from Template. The Nios II Application and BSP from Template wizard appears.

   - Under **Target hardware information**, next to **SOPC Information File name**, browse to locate the <design files directory> where the previously hardware project resides.
   - Select **Use default location**
   - Select the **blank project** template. Click Finish

3. The Nios II SBT for Eclipse creates the new project and returns to the Nios II C/C++ project perspective.
4. Open the “software” folder and drag with the mouse the template file above the first folder of the project explorer. Select copy files and click ok. Now your .c file is included in your project.
5. Now you have to fill the gaps and what is more important, understand the source code.

At the beginning of the source code, some header files are included. Some others need to be included. Read the comments and include the necessary header files.

After including the header files, the function rx_ehernet_isr is declared. This function is executed every time an interruption from sgdma_rx arrives due to the reception of a new frame.
Then, some necessary variables are declared as global variables. This section of the code allocates memory to store received frames (we use a vector for that) and for the sgdma_tx and sgdma_rx devices (we use pointers). The sgdma descriptors are also declared.

Pay attention to the keyword __attribute__ that comes next to the descriptor declarations. 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*.

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*.

Next, define a pointer to the base address of the Triple-Speed Ethernet IP Core. The value of the MecaCore base address is stored in the variable TSE_BASE, which can be found in system.h.

The next piece of code is the process of initializing the Triple-Speed Ethernet IP Core and the external PHY chips. In order to understand and be able to fill the gaps, go to the *Triple-Speed Ethernet megacore function user guide*. You will need to configure some command_config registers. Look up on the tables which registers correspond which each one of the initializations. Use the following recommended initialization sequences for this example:

1. External PHY Initialization using MDIO: (external PHY Address is assigned 0x10.

   - External PHY Address is assigned 0x10 → mdio_addr0 = 0x10

2. MAC Configuration Register Initialization:

   - Disable MAC Transmit and Receive Datapath and wait
     Disable the MAC transmit and receive datapath before performing any changes to configuration.
     Set TX_ENA and RX_ENA bits to 0 in Command Config Register.
     Wait that Command_config register has the correct value.
   - MAC FIFO Configuration:
     Tx_section_empty = Max FIFO size - 16
     Tx_almost_full = 3
     Tx_almost_empty = 8
     Rx_section_empty = Max FIFO size - 16
     Rx_almost_full = 8
     Rx_almost_empty = 8
     Enable Store and Forward mode, Tx_section_full=0
     Enable Store and Forward mode, Rx_section_full=0
   - MAC address Configuration: Assume that the MAC address is 00-1C-23-17-4A-CB
     mac_0 = 0x17231C00
     mac_1 = 0x0000CB4A
   - MAC function configuration
     Maximum Frame Length is 1518 bytes: Frm_length = 1518
     Minimum Inter Packet Gap is 12 bytes: Tx_ipg_length = 12
     Maximum Pause Quanta Value for Flow Control: Pause_quant = 0xFFFF
     Update the Command_config register to set the MAC with the following options: 100Mbps, Full Duplex, Padding Removal on Receive, CRC Removal and TX MAC Address Insertion on Transmit Packet (Select mac_0 and mac_1 as the source MAC Address): Command_config Register = 0x00800220

- Reset MAC
  Altera recommends that you perform a software reset when there is a change in the MAC speed or duplex. The MAC software reset bit self-clears when the software reset is complete. Set SW_RESET bit in the Command_config to 1:
  Command_config Register = 0x00802220
  Wait for the reset bit to be cleared, i.e., Command_config Register = 0x00800220
- Enable MAC transmit and receive datapath, Gigabit Ethernet and promiscuous mode
  Set TX_ENA and RX_ENA to 1 in Command Config Register
  Set ETH_SPEED to 1 in Command Config Register
  Set PROMIS_EN to 1 in Command Config Register
  Command_config Register = 0x0080023B
  Wait Command_config Register = 0x0080023B

You can write different values into command_config register to tell the IP Core to operate at 100 Mbps or 10 Mbps mode. For example, you can write 0x00000033 for operating at 100 Mbps and write 0x02000033 for 10 Mbps. More details can be found in *Triple-Speed Ethernet Megacore Function User Guide*.

After you enable the read and write transfers, the Triple-Speed Ethernet IP Core will start working. Then the main loop is entered. In this loop, the signal received from the switch button is checked. If it is 1, switched on the LED. Otherwise, switched off the LED.

The last part of the code is the interrupt routine. Whenever an interrupt is raised, i.e., whenever a new frame is received, the interrupt routine will be executed to print in the terminal window the source and destination addresses of the received frame if the switch button is up.

Once you have added all the necessary lines in the .c file, build and run the program. Check that your program is working as expected.

Checkpoint! Show your current status to the instructor (20%)

4. EXTRA POINT. Showing Statistics in the terminal window

Add a new function to your software program. When the switch button is down and a new frame is received, print in the terminal window some statistics, such as how many frames have been received correctly, how many valid broadcast frames have been received or how many frames have been received with errors.

Checkpoint! Show your current status to the instructor (G: 5%)

5. Guided Assignment Report (G:40%)

Answer these questions:

1. Explain with your own words, what you have done in this guided assignment and what you have learnt. You only need one paragraph, please be concise. (10%)
2. Can you connect two machines with only one regular cable? Why? What can you use to connect two machines? Give two options. (15%)
3. Suppose that your board is working in MII mode and it is connected to a network using a hub. Suppose that at a given moment the Command_config register has this value 0x01000E43. What MAC functions are configured? Is it working in promiscuous mode? Which is the Ethernet speed operation? Do you know if any collision has been detected? (15%)

Submission materials:

- A folder with all the .v files (*nios_system*, *my_pll*, *pll_clock_PHY*, *top_level*), the .sdc file, the .sof file and the .sopcinfo file. Inside this folder include another folder called "software" with the files generated with the NiosII SBT for Eclipse.
- A brief report (1—2 pages at most), answering the specific questions in section 5. Guided Assignment Report.
- For convenience, create a ZIP/RAR file with the materials.