

Version for EDK 10.1.03 as of January 7, 2009

## Acknowledgement

This lab is derived from a Xilinx lab given at the University of Toronto EDK workshop in November 2003. Many thanks to Xilinx for allowing us to use and modify their material.

## Goals

- Understand the procedure for adding more complex peripherals to an XPS project.
- Use XPS to manually add the XPS 10/100 EMAC Lite peripheral to the MicroBlaze system.
- Search through more documentation to see where to find various bits of information and examples.

## Prerequisites

Module m03: Adding IP and Device Drivers — Timers and Interrupts

## Preparation

- Read through this module first to get an idea of what you are about to do.
- Find the data sheet for the XPS Ethernet Lite Media Access Controller (EMACLite) and review it.
- In this lab you will be modifying an example program that can be found in the Processor IP drivers library for the device. On the Microprocessor lab machines, look in:

```
%XILINX_EDK%\sw\XilinxProcessorIPLib\drivers\emaclite_v1_13_a\examples
```

On the ECF Linux machines, look in:

```
$XILINX_EDK$XILINX_EDK/sw/XilinxProcessorIPLib/drivers/emaclite_v1_13_a/examples
```

- The first step is to copy the previous lab into a directory called `lab4`. If you like, you can delete the `lab3.c` source file from your `lab4/code/` directory.

You should, of course, check to see if you have enough space first! You may need to clean up some files. You can do this using the XPS interface (*i.e.*, via `Project` → `Clean all generated files`), using the facilities provided by the `system.make` makefile (*e.g.*, start by trying `make -f system.make`; you might need to first use `dos2unix` on `system.make` if you're running `make` on an UGSPARC or an ECF Linux machine), or by simply deleting the `implementation` directory in your project (since it is automatically generated and since it consumes the bulk of your project's disk space).

- In this module, you will be working with the example called `xemaclite_intr_tapp_example.c`. Copy the file from the driver example directory.

You will be asked to modify it so that it will work with your system. You might want to try to understand what it does so that you will not have to spend time during a lab period doing this. You can also make some modifications or add some `xil_printf` statements. Put the copied (and possibly modified) `xemaclite_intr_tapp_example.c` in `lab4/code/`.

## Background

As peripherals become more complex, there are more signals to be brought out of the FPGA and possibly more timing issues, including the need for timing constraints and Digital Clock Managers (DCMs). The DCM is a block in the FPGA that contains functions like Delay-Locked Loops (DLLs) that can be used to help synchronize internal logic and clocks with external logic and their clocks. You will not have to deal with them in this lab. In this lab, you will be connecting the FPGA to an external Ethernet chip.

Ethernet is a widely used peripheral, so it is beneficial to learn how to properly include the XPS EMAC into an XPS project.

Outside of the FPGA is the physical layer interface (PHY) chip that actually connects to the Ethernet cable on one side, and the FPGA pins on the other side. The EMAC is the peripheral that is inside the FPGA that connects from the FPGA pins to the PLB bus of the MicroBlaze allowing the processor to talk to the Ethernet chip.

This module is built on top of Module m03. It expects a MicroBlaze system with an interrupt controller and serial Uart device for standard I/O. If you didn't successfully add the DIP switch in m03, you can build onto the simpler design.

Note that the clock frequency must be greater than or equal to 65MHz for the Ethernet core to be able to operate at 100Mbps; as such, the design implemented in this lab can only be expected to operate at 10Mbps (which requires a clock frequency of only 6.5MHz).

## Using XPS Base System Builder

1. Copy the XPS project directory of the previous lab and rename the copy to `lab4` if you have not done so already. This will be the working project directory for this lab. You may need to clean the project or delete the `implementation` directory of the previous module to free up disk space.
2. Open the `lab4` project using XPS.
3. Add an `xps_ethernetlite` peripheral to the system (use version 2.00.a). Define the base address of the device to be aligned at a 8K boundary following the address of the last peripheral in the list. Attach the EMAC as a slave to the PLB bus.
4. Make the `PHY_*` ports of the `xps_ethernetlite_0` peripheral instance external, except for `PHY_rx_en`, `PHY_rst_n` which should only be made external if you are using the XUPV2P board. **Be careful not to make both the bidirectional signal and the individual unidirectional components of that signal external!** For instance, make `PHY_Mii_data` external and leave `PHY_Mii_data_I`, `PHY_Mii_data_0`, and `PHY_Mii_data_T` as No Connection. Later, you will use the board user's guide or schematic for information on pin assignments for each signal.
5. There are two pins on the PHY that are connected to the FPGA but that do not have corresponding ports in the EMAC device. These signals are inputs to the PHY and should be tied high (*i.e.*, tied to `net_vcc`) in the design.  
  
Click **Add External Port** in the Ports tab to create a system port. Name the port `xps_ethernetlite_0_PHY_slew0_pin`, make it an output, and connect it to `net_vcc`. Do the same for `xps_ethernetlite_0_PHY_slew1_pin`.
6. Connect the `IP2INTC_Irpt` signal of the `xps_ethernetlite_0` peripheral instance to a new net called `xps_ethernetlite_0_IP2INTc_Irpt`.

Recall that the interrupt controller can handle a number of interrupt input request lines. The interrupt input of the controller is really a vector of signals, not a single wire. Since there may be other interrupt signals already connected to the interrupt input of the interrupt controller, the EMAC interrupt output may need to be concatenated to those signals.

To add a new signal to the interrupt input of the controller, click on the entry in the **Net** column beside the `Intr` input of the `xps_intc_0` peripheral instance. In the dialog that opens, select the interrupt signal

you wish to add. Click on the plus symbol to add signals, the scissors symbol to remove signals, and the arrows to change the order of signals in the list on the right. Note that the order of the list reflects the priority of the interrupt sources.

7. Add the following entries to the `system.ucf` file for the project. This configures the physical connections between the PHY device and the FPGA pins. Use the board user's guide to find the correct pin locations. (*Hint: You can cut and paste the template below from the PDF into your `system.ucf` by changing the select cursor in Acrobat to select text. Make sure the angle brackets appear as greater and less than symbols after pasting.*)

```
Net xps_ethernetlite_0_PHY_tx_data_pin<0> LOC=***; # TX_DATA0
Net xps_ethernetlite_0_PHY_tx_data_pin<1> LOC=***; # TX_DATA1
Net xps_ethernetlite_0_PHY_tx_data_pin<2> LOC=***; # TX_DATA2
Net xps_ethernetlite_0_PHY_tx_data_pin<3> LOC=***; # TX_DATA3
Net xps_ethernetlite_0_PHY_tx_en_pin LOC=***; # TX_ENABLE
Net xps_ethernetlite_0_PHY_tx_clk_pin LOC=***; # TX_CLOCK
Net xps_ethernetlite_0_PHY_rx_data_pin<0> LOC=***; # RX_DATA0
Net xps_ethernetlite_0_PHY_rx_data_pin<1> LOC=***; # RX_DATA1
Net xps_ethernetlite_0_PHY_rx_data_pin<2> LOC=***; # RX_DATA2
Net xps_ethernetlite_0_PHY_rx_data_pin<3> LOC=***; # RX_DATA3
Net xps_ethernetlite_0_PHY_rx_clk_pin LOC=***; # RX_CLOCK
Net xps_ethernetlite_0_PHY_dv_pin LOC=***; # RX_DATA_VALID
Net xps_ethernetlite_0_PHY_rx_er_pin LOC=***; # RX_ERROR
Net xps_ethernetlite_0_PHY_col_pin LOC=***; # COLLISION_DETECTED
Net xps_ethernetlite_0_PHY_crs_pin LOC=***; # CARRIER_SENSE
Net xps_ethernetlite_0_PHY_rst_n_pin LOC=***; # ENET_RESET_Z -- XUPV2P only
```

## Building The Hardware

8. Select the **Hardware** menu and the **Generate Bitstream** submenu in XPS to start building the hardware system. This will take about 10-15 minutes as the system is compiled, placed and routed for the FPGA. During the build process, a lot of information will be displayed in the bottom window pane of XPS. The first step of the software design may be done while the bitstream is being generated.

## Look At Next

Module m05: Adding a User-Designed Peripheral

Module m06: Using ISE