Version 1.1 1/29/2004 10:48 PM

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 OPB 10/100 EMAC peripheral to the MicroBlaze system.
- Search through more documentation to see where to find various bits of information and examples.

Preparation

1. Read through this lab first to get an idea of what you are about to do.
2. Find the data sheet for the OPB Ethernet Media Access Controller (EMAC) (v1.00m) and review it. A copy is also posted directly on the course web page.
3. 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 lab machines, look in

   O:\Xilinx\EDK6.1i\sw\XilinxProcessorIPLib\drivers\emac_v1_00_d\examples

4. The first step is to copy the previous lab. You can do this in advance by going to your ugsparc directory where you can find “lab3”:

   ```
   % cp -r lab3 lab4
   ```

   You should, of course, check to see if you have enough space first! You may need to clean up some files, i.e., do some cleaning. If you want to clean lab3, you can also do this from the ugsparcs using the system.make file in the lab3 directory. In the lab3 directory type:

```
% make -f system.make hwclean
```

to clean the hardware directory.

```
% make -f system.make
```

will give you all the options. But, this still does not work!! Why?

Try this first:

```
% dos2unix system.make usystem.make
```

and try the above commands but using usystem.make. What's going on? Just one of the many kinds of things you have to deal with when working with CAD tools.

5. During the lab, you will be working with the example called

   xemac_intr_fifo_example.c

   This file is also posted on the course web site for you to look at before the lab. 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 the lab doing this. You can also do some modifications and adding of print statements. Put your file in your ugsparc directory where you can access it later. If you did step 3 above, then put it in the code directory of the lab4 project directory.

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 (DCM). The DCM is a block in the FPGA that contains functions like 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 here. 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 OPB EMAC into an XPS project.

Outside of the FPGA is the physical layer interface (PHY) 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 OPB bus of the MicroBlaze allowing the processor to talk to the Ethernet chip.

This lab is built on top of the previous week's lab. 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 to last week's lab, you can build onto the simpler design.

Note that at 27MHz, the speed of the OPB bus, the EMAC will only function correctly at 10Mbps.

Using XPS Base System Builder

1. Copy the XPS project directory of the previous lab and rename the copy to "lab4". This will be the working project directory for this lab. You may need to delete the implementation directory of the previous lab to free up disk space. Use "clean" under the Tools menu.

2. Open the "lab4" project using XPS.

3. From the Project menu, select the Add/Edit Cores… (dialog) submenu item. Add the opb_ethernet peripheral to the system. Define the base address of the device to be a **16K aligned address** following the address of the previous peripheral in the list. Attach the EMAC as a slave to the OPB bus. Note that two devices appear. Configure only the sopb (slave) version.

   When would you attach the EMAC as a master to the OPB bus?

   The minimum address range of the OPB EMAC core is 16K (0x4000). The tools require that the base address begin at a 16K aligned address.

4. In the Add/Edit Cores… (ports tab) dialog box, add the PHY ports (except PHY_rx_en and PHY_rst_n) and the interrupt output port of the EMAC to the

design so that they can be connected to other signals or pins of the FPGA. Make sure all PHY ports are external such that they are connected to their corresponding FPGA pins. Later, you will use the board user's guide or schematic for information on pin assignments for each signal.

Note that the net names chosen for each port will need to match the net names in the system UCF file. The default net names will suffice for the PHY connections. You'll change the interrupt name later.

5. Add a vector range on the PHY receive and transmit data ports. Determine the width of the vector from the data sheet. For an n-bit bus, input the range of **[n-1:0]**.

6. There are two PHY signals connected to the FPGA that do not have corresponding ports in the EMAC device. These signals are inputs to the PHY and should be tied high (tied to **net_vcc**) in the design.

   Click Add Port in the Ports tab to create a system port. Name the port **PHY_slew1**, make it an output, and connect it to **net_vcc** in the Add External Port dialog box. Click OK.

   Do the same for **PHY_slew2**.

7. Make the interrupt output of the EMAC an internal port. We will be connecting this net to the interrupt input of the interrupt controller device. Rename the net name of the EMAC interrupt output port to **emac_intr**. This can be done by typing the new name in the Net Name box of the port in the Ports tab.

   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.

   The interrupt controller has a vector of interrupt input signals with each signal being assigned a specific priority. Concatenation of signals builds a vector of signals that will be connected to the input of the interrupt controller. The last signal in the list of concatenated signals is the highest priority. Each signal prior to the last decrements in priority.

Here is the syntax for the net name of the interrupt input of the interrupt controller, assuming there is already a signal named **timer_intr** connected to the interrupt controller input that comes from the interrupt request output of the timer peripheral. The line below indicates the concatenation of two signals to form a two-bit input vector for the interrupt input.

timer_intr & emac_intr

Note that the connections made under this Ports tab are done by the names. Boxes with the same names indicate ports to connect. The Connect button is only a quick way to connect ports with the same name, i.e., to save some typing. You cannot use it here.

8. Click OK on the Add/Edit Cores… (dialog) dialog box to save changes and return to the main XPS GUI.

9. Add the following entries to the UCF file in the /data subdirectory of the project. This is describing the physical connections between the PHY device and the FPGA pins. Use the board user's guide to verify the correct pin location. Note that one of the pin locations below (****) still needs to be assigned. *(Hint: You can cut and paste these values from the online document source by changing the select cursor in Acroread to select text.).*

```
Net PHY_slew1 LOC=G16;
Net PHY_slew2 LOC=C16;
Net opb_ethernet_0_PHY_crs LOC=F20;
Net opb_ethernet_0_PHY_col LOC=C23;
Net opb_ethernet_0_PHY_tx_data<3> LOC=C22;
Net opb_ethernet_0_PHY_tx_data<2> LOC=B20;
Net opb_ethernet_0_PHY_tx_data<1> LOC=B21;
Net opb_ethernet_0_PHY_tx_data<0> LOC=G20;
Net opb_ethernet_0_PHY_tx_en LOC=G19;
Net opb_ethernet_0_PHY_tx_clk LOC=****;
Net opb_ethernet_0_PHY_tx_er LOC=D21;
Net opb_ethernet_0_PHY_rx_er LOC=D22;
Net opb_ethernet_0_PHY_rx_clk LOC=C17;
Net opb_ethernet_0_PHY_dv LOC=B17;
Net opb_ethernet_0_PHY_rx_data<0> LOC=B16;
Net opb_ethernet_0_PHY_rx_data<1> LOC=F17;
Net opb_ethernet_0_PHY_rx_data<2> LOC=F16;
Net opb_ethernet_0_PHY_rx_data<3> LOC=D16;
Net opb_ethernet_0_PHY_Mii_clk LOC=D17;
Net opb_ethernet_0_PHY_Mii_data LOC=A17;
```

Building The Hardware

10. Select the Tools 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.

Defining The Software

11. Create a simple loopback application that sends an Ethernet frame and receives the same frame while the EMAC is in internal loopback mode. Begin with the xemac_intr_fifo_example.c example provided in the EDK installation area by copying this file to your lab4/code subdirectory. The example resides at:

    O:\Xilinx\EDK6.1i\sw\XilinxProcessorIPLib\drivers\emac_v1_00_d\examples

You can look through the example code while XPS generates a hardware bitstream but you should have done this prior to the lab as preparation!

12. Add the xemac_intr_fifo_example.c file to the XPS project as program source. Be sure to remove any other source files that are leftover from previous labs.

13. Select the Level 1 driver interface for the EMAC device's S/W Settings.

14. Next you will edit the xemac_intr_fifo_example.c source to match your system.

    The example supports both the Microblaze and PPC405 processors as well as the VxWorks operating system. Be sure to include support only for MicroBlaze.

    You should reference the xparameters.h file for the correct constant names needed by the application. Open the xparameters.h file. When looking through the file, you should notice that there are no references to the Ethernet core. The generation of the bitstream did not update your xparameters.h file. Goto the Tools menu and generate the libraries. This should update your xparameters.h file. Now you can edit the example file to match your system by providing the correct names from the xparameters.h file. *(Hint: the constants should all begin with XPAR so you can do a search)*

Compiling the Drivers and Program

15. In XPS, make sure that the compiler options are set so there is no optimization and debug flags are generated. Compile the application.

16. Select the Tools menu and the Update Bitstream submenu to update the hardware bitstream with the xmdstub elf.

Downloading the Bitstream to the FPGA

17. Ensure that power is on to the board and the parallel 4 cable is connected to the PC. The Status light on the parallel 4 pod should be green. Select the

Tools menu and Download submenu. This will download the hardware and software contained in the bitstream to the FPGA. The ROM monitor software will begin executing after the download completes.

Getting Ready to Debug

18. Use XMD to connect to the stub (ROM monitor software) running on the target board.

Debugging Software

19. Use GDB to connect to the XMD gdb server and download the executable elf file that contains the EMAC loopback application.

20. Use GDB to step through the program. Verify the program runs by setting breakpoints in the interrupt handlers and looking at return values with the debugger or by adding printf statements to the code and looking at the terminal output.