Version for EDK 8.1i as of May 10 2006

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.

Requirements

Module 3 – Drivers, IP, and Interrupts.

Preparation

1. Read through this module 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) and review it.
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 Microprocessor lab machines, look in

   `O:\Xilinx\EDK6.3i\sw\XilinxProcessorIPLib\drivers\emac_v1_00_d\examples`

   On the ugsparcs, look in

   `/cad2/Xilinx/EDK6.3i/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`. In an ugsparc shell, type:

   ```
   % 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. Keep this in mind if you first generate your system on the PCs and then try to run the scripts on the ugsparcs.

5. In this module, you will be working with the example called `xemac_intr_fifo_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 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 4 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) 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 OPB 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 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` if you have not done so already. This will be the working project directory for this lab. You may need to delete the implementation directory of the previous module to free up disk space. Use the Clean submenus under the Hardware and Software menus.

2. Open the `lab4` project using XPS.

3. 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. Add the PHY ports (except PHY_rx_en and PHY_rst_n) and the IP2INTC 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 when labeling buses as external ports, the polarity is reversed. If the polarity is [3:0] in the Internal Ports Connection list, it must be reversed to [0:3] in the External Ports Connection list. This is yet another peculiarity of working with new tools.

   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. 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`.

6. 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`.

   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 Intr input of the Interrupt Controller. In the Net dropdown list, select the interrupt signal you wish to add. Do this multiple times, each time adding one signal. The interrupt signals will be ranked high to low in the interrupt list.

   Note that the connections made under this Ports tab are done by the names. Boxes with the same names indicate ports to connect.

7. For the OPB Ethernet core, set the DMA Present parameter to a value of No DMA, to disable the direct memory access for the Ethernet core.

8. 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. Make sure the angle brackets appear as greater and less than symbols).*

```
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 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.

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. Step 5 of the Preparation describes where to find the file.

    You can look through the example code while XPS generates a hardware bitstream but you should have done this prior to the working on this 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 modules.

13. 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. (You may have to close and reopen the 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

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

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

Downloading the Bitstream to the FPGA

16. 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

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

Debugging Software

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

19. 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 `xil_printf` statements to the code and looking at the terminal output. Why do we prefer `xil_printf` over `printf`?

    Note: You can save your breakpoint locations for subsequent runs. In the GDB Source window, select the View menu and the Breakpoints submenu. The Breakpoints window will open. Once you have set all your breakpoints in the Breakpoints window, select the Global menu and the Store Breakpoints submenu. You can then specify a filename that you can restore in a subsequent run of GDB.

Look At Next

Module 5: Adding a User Core - Snoopy
Module 6: Using ISE