

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

- Use Xilinx tools to add to the basic MicroBlaze system that was built in Module 1. A primary goal of this lab is to understand more details about adding IP into the system without Base System Builder and using device drivers.
- The General Purpose I/O (GPIO) IP core is used in this lab together with the associated device driver. The GPIO will be added to the project and used to turn on and off User LED 0 on the board. You will manipulate the state of the LED from XMD and by using a program.
- Get an introduction to the use and structure of driver code.

Requirements

Module 1 on building a base system.

Preparation

1. The GPIO is a simple parallel I/O port that has similar functions to the PIT used on the GIZMO board. You can have from 1 to 32 input/output ports, which corresponds to the processor having a 32-bit bus. If you use fewer than 32 ports, the advantage of working in the FPGA synthesis environment is that you will only instantiate logic for the number of ports that you require.

If you are on an installed system, find the overview of the OPB GPIO in the Processor IP Reference Guide. The Guide is provided in the EDK Online Documentation. The location of the documentation was described in Module 1. In the Guide you will find a link to the data sheet. Familiarize yourself with how this block works.
2. Also, only available via the documentation tree of an installed system is the Device Driver Programmer Guide. Xilinx provides device drivers so that the user may create applications quickly and easily. The Device Driver Programmer Guide documents many details of the device drivers and software infrastructure. You should get familiar with this section as you may eventually need to write your own drivers. In this lab you will be required to modify some code to properly use the existing drivers.
3. You will start modifying code in this lab. If you are not familiar with any source code control system, you can quickly learn one by doing `man rcsintro` on any unix system. Using such a system is highly recommended and just good practice. You will need to run the commands from a window connected to one of the ugsparcs. There is an ssh window in the Courseware folder.
4. You will need another 15MB of disk space. Limits will be adjusted once the final class list is known. You can `clean` the files created in Module 1 via the **Hardware** and **Software** menus to save some space.

Background

Processor IP is an integral part of a System-On-Chip (SOC) system. Xilinx provides a variety of processor IP cores that can quickly and easily be integrated into a system using XPS.

This module builds from Module 1 and assumes that the user has completed Module 1. A basic understanding of the EDK tools should have been gained from Module 1.

Setup

1. Copy the XPS project directory (lab 1 folder) of the previous lab and rename the copy to **lab2**. This will be the working project directory for this lab.
2. Start XPS by selecting Courseware → Xilinx 8.1i → Xilinx Embedded Development Kit 8.1 → Xilinx Platform Studio from the Windows desktop.
3. From the initial startup screen, select **Open a Recent Project** and click OK. Browse to the **system.xmp** file of in the **lab2** directory. Click Open to open the project. This is the project you completed in Module 1.

Adding A GPIO With User LED 0

4. Examine all the options in the **System Assembly View** to look at the various aspects of the system that you have already built. Think about how hard it would have been to do this from scratch! Having validated (tested) ready-to-use function blocks can significantly improve your design cycle.
5. From the **IP Catalog** tab on the left, expand the **General Purpose IO** tab. Double click **opb_gpio 3.01.b** to add it to the project.

6. From **Addresses**, set the base address of the **opb_gpio** to **0x80040000**. The base address is the 1st address that will be decoded by the peripheral on the On-chip Peripheral Bus (OPB).

Set the Size to be **512**. This will automatically set the High Address of the device to **0x800401ff**, which uses the minimum amount of memory space as displayed for the device.

What do you expect the tool to be doing with this information?

7. Select **Bus Interface**. The lines on the left show the peripheral bus interfaces in the design and how the peripherals interfaces are attached to the buses. To understand what the boxes and circles are, refer to the Legend at the bottom of the view. A filled shape means that the interface is connected, and a shape outline means that the interface is unconnected. Clicking a shape allows the interface to toggle its status between connected and unconnected. Select the **opb_gpio** instance to be a slave on OPB.
8. Right click **opb_gpio** and select **Configure IP...** Clicking the **Datasheet** button opens the specification for the IP. Review the parameters that can be set and signals that are to be input and output based on the application.

Parameters are implemented in the IP using VHDL generics. This feature of VHDL is a large advantage for IP that must fit many different applications.

9. Select the **GPIO Data Bus Width** parameter and change its value from **32** to **1** such that there is only **1** bit of general purpose I/O. The general purpose I/O will be an output to control an LED on the board. Click OK.
10. Select **Ports**. Depending on the filters you have set, this view can be very lengthy. Right click the top bar and make sure **Flat View** is not selected. Scroll to find the **opb_gpio** instance and select the **GPIO_IO** signal. In the **Net** dropdown box, select **Make External**. This should add the **GPIO_IO** signal to the list of **External Ports**.

What do you expect the tool to do with this information?

Notice that the GPIO signal is external such that it will be pinned out of the design and connected to a pin of the FPGA. That pin will be connected to the LED driver.

Building The System With GPIO

11. Select the **Project** menu and the **Project Options...** submenu. Select the **HDL and Simulation** tab. If it is not already set, select **Verilog** as the HDL type.
12. Select the **Hardware** menu and the **Generate Netlist** submenu. This will take a few minutes to complete. This step is updating the system and synthesizing it to use the GPIO IP. When it completes, it is not yet ready to put in the FPGA because it is only a netlist, which is only a description of the logic elements and how it is connected. A bitstream will need to be generated by doing a place and route of the design for the FPGA.
13. In the **Project** tab on the left, expand the **Project Files** tree and double click on UCF File: **data/system.ucf**. This opens the file in an editor window. This file contains the design inputs and outputs to specific pins of the FPGA. Edit the UCF file to connect bit 0 of the **GPIO_IO** signal to **User LED 0** by adding the line

```
Net opb_gpio_0_GPIO_IO LOC=B27;
```

and save the file.

The net name is the same as the signal you added to the **External Ports** section. Pin B27 of the FPGA connects to **User LED 0**. The pins of the FPGA are described in the **Multimedia Board User Guide** which is available on the UofT EDK page. You can find pin B27 on page 23. See if you can find the pin on the **Multimedia Board Schematics** available on the UofT EDK page. You will observe that the schematic is quite large. Use the Acrobat Find feature to look for the string **user_led**. You should find it on two pages. One showing the pins of the FPGA and the other showing the connection to the LEDs. Remember how the LEDs are driven as a later step will ask you about this.

14. Right click on **Project: mb0_default** in the **Applications** tab and select the **Set Compiler Options** submenu. A dialog box is displayed. Verify that the **XmdStub** mode of operation is selected and the **Program Start Address** is **0x800**. This will build the ROM monitor stub, called **XmdStub**, into the software and hardware so that debugging can be done. Executable mode assumes that the software has already been debugged and executes it on reset.
15. Select the **Device Configuration** menu followed by the **Download Bitstream** submenu. This will run the ISE place and route tools to create a bitstream, then download the bitstream to the target board. This will take a few minutes.

Testing The GPIO In Hardware

16. After the hardware and software have successfully downloaded, use XMD to test the system to ensure that the GPIO is working. Reads and writes of the GPIO registers can be done using the read (**mrd**) and write (**mwr**) memory commands of the XMD because the registers are memory mapped. The GPIO data sheet states that the **GPIO_DATA** register is at address **0x00** and the **GPIO_TRI** register is at **0x04**. Previously, you set the base address of the GPIO to **0x80040000**. Add these numbers to determine the register addresses.

The tri-state register of the GPIO must be cleared such that the LSB of the register is an output as described in the GPIO data sheet. A zero must be written to the **GPIO_DATA** register to turn on the **User LED**. Can you explain this? Recall what you saw on the schematics.

Adding Software To Use GPIO

17. Layer 1 drivers are high-level drivers. Their interfaces are defined in `<driver>.h` files. Each file includes the low-level (layer 0) driver interface defined in `<driver>_l.h`. The layer 1 driver has a larger memory footprint and robust error checking that is not part of the layer 0 driver. The high level driver also supports more device features and interrupt driven I/O. Each function of the layer 0 driver takes the base address of the device as the first argument. Each function of the layer 1 driver takes an instance pointer as the first argument.
18. Remove the `lab1.c` source file from the previous lab by selecting it under **Sources** in the **Applications** tab and pressing delete.

Add the source file `lab2a.c` by right clicking on **Sources** in the **Applications** tab, selecting **Add File**, and navigating to the `code` directory of your project. Before making changes to the file, a good practice is to use a source code control system to help you archive the multiple versions of your code as you change it. Use RCS, or another system, to archive the original file before you change it.

Reference the device driver documentation in the Driver Reference Guide to determine the names of the device driver header files. The Driver Reference Guide is available in the EDK Online documentation. You can also browse to the drivers included in the XPS project by looking in the `microblaze_0/libsrc` subdirectory of the project. First, note that the files are copied there when you select **Software** → **Generate Libraries and BSPs** from the menus.

Make the appropriate changes to the source file such that it will compile. There are a number of places marked with `<TO BE DONE>` that need to be modified. Some of the values can be found by referencing the `xparameters.h` file for system-wide constant definitions. This header file is generated by double clicking **Project: mb0_default** → **Processor: microblaze_0** → **Generate Header: microblaze_0/include/xparameters.h** in the **Applications** tab.

This source code uses the GPIO driver to turn on and off the LED on the board at a visible rate. Save and compile the program. Connect to XmdStub with XMD. Open XILINXPORT and use the Software Debugger to verify that the code works correctly. Remember to avoid using the Run submenu of the Run menu and the running man icon to run the program.

Other things to consider

1. What version of the MicroBlaze processor is being used? (*hint: look in Project->Add/Edit Cores*)
2. What is the base address of the `UartLite` peripheral (its instance name is `RS232`)?
3. What is the instance name of the `opb_mdm` peripheral?
4. What is the net name used for the system clock in the design?
5. What is the name of the `UartLite` receive data signal? Is the external reset signal active high or active low? (*hint: look at the parameters for the LMB and OPB bus controllers*)

Look At Next

Module 3: Drivers, IP, and Interrupts