Version for EDK 14.2 as of January 9, 2013

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.sdfsdf

Goals

- Use Xilinx tools to add to the Microblaze system that was built in Modules m01 and m02. A primary goal of this lab is to understand the details of adding IP and device drivers that use interrupts.
- IP cores including the General Purpose I/O (GPIO), Timer, and Interrupt Controller are used in this lab together with the associated device drivers. Many complex embedded systems use interrupt driven I/O rather than polled I/O and as such, learning how to use interrupts is important.
- The Interrupt Controller and Timer will be added to the project and used to create a periodic interrupt that can be used to schedule other processing. The GPIO will be used to get input from switch User Input 1. This switch will control the flashing rate of User LED 0 from Module m02.

Prerequisites

Module m02: Adding IP and Device Drivers — GPIO and Polling

Preparation

- Find the overviews of the AXI Timer/Counter and the AXI Interrupt Controller in the EDK documentation (look in the IP Reference → Processor IP Catalog). From there you will find a link to the datasheets. Familiarize yourself with how these blocks work.
- Copy the project directory (`lab2` folder) of the previous lab and rename the copy to `lab3`. This will be the working project directory for this lab. Extract `lab3.zip` into the directory containing the `lab3` directory to add the source files required for this module to the `lab3/code/` directory. If you like, you can delete the existing `lab3/code/` directory before extracting m03.zip to clean up the files from Module m02.

  You should also remove the workspace directory to avoid any confusion with the previous projects. A new one will be created when you start the SDK.

  In this lab you will be modifying the file lab3.c, which you just extracted into your copy of your Module m02 project. Have a look at this file to understand what it does. In this lab you will be filling in the `<TO BE DONE>` sections after XPS generates the header files to support the hardware.
- Launch XPS and open the project for Module m03 (*i.e.*, `lab3/system.xmp`).

Background

This lab builds from Module m02 and assumes that the user has completed Module m02. A basic understanding of adding IP and device drivers should have been gained from Module m02.

Adding A Timer/Counter And Interrupt Controller

1. Go to the IP Catalog tab and add an AXI Timer (axi_timer) found in DMA and Timer and an AXI Interrupt Controller (axi_intc) found in (Clock, Reset, and Interrupt). Both peripherals will be automatically connected to the AXI bus as slaves(axi4lite_0), and assigned their own Base and High Addresses.

2. The XPS GUI is supposed to make it easier for you to make the connections. Some prefer a more direct method. When making or deleting connections, XPS is really just editing the mhs file for the design. It is possible for you to directly edit the mhs file if you know what you are doing. In the next step, observe the changes to the mhs file after you make each connection.

3. You now need to connect the interrupt control lines, which is done separately from the main bus connections that were done automatically in Step 1. Select the Ports tab. Click to expand the axi_timer and axi_intc entries to show the available ports.

   The Interrupt output signal of the timer should be connected to the interrupt input (Intr) of the interrupt controller. Since this is a connection to enable interrupts, the easiest is to bring up the Interrupt Connection Dialog. Click once under the Connected Port heading on the line for any of the interrupt signals you want to connect. The Interrupt Connection Dialog can then be used to connect all of the interrupt lines to a particular controller. In this case, there is only one controller to select. Select the axi_timer_0 instance under Unconnected Interrupts and move it to the Connected Interrupts. Click Ok and you should see the connection made.

   (Hint: When not connecting interrupt lines, you can connect two or more ports together by highlighting them, right-click one of the signals, then click Connect Ports.... Leave the name connection as default and click OK.)

   The INTERRUPT bus of the interrupt controller should be connected to the INTERRUPT bus of the processor. Note that this bus contains a number of signals such as the interrupt request line (Irq). What are the other signals? To find out, switch to the Bus Interfaces tab. Double click on axi_intc_0 to bring up the XPS Core Config window for the controller. Note that the INTERRUPT port has a + sign indicating it is a bus. Click on the wire stub under the + sign to expand it and see all the signals in that bus. Close the window and return to the Bus Interfaces tab. Find the INTERRUPT bus under microblaze_0. Note that is says No Connection under Bus Name. Click on that entry and select the only available bus in this case, which is axi_intc_0_INTERRUPT.

4. Rebuild the bitstream and launch the SDK. Make sure that your workspace will be in your `lab3` directory. You will see lab3_hw_platform in the Project Explorer of the SDK.

Adding Software To Use The Timer And Interrupt Controller

You may want to review the more detailed steps in Lab 2 for adding the software.

5. You now need to create a New Xilinx C Project and call it *blink_LED_intc_0*. Make it an Empty Application so that we can later point it at our program. Call the new board support package *blink_LED_intc_0*.

6. Add the `lab3.c` source file to the *blink_LED_intc_0* project.

   You will see some errors as it tries to compile the file. Why?

7. As you read through `lab3.c` you will see an example of how to set up a program that will use interrupts. It will help if you recall your previous experience of doing interrupts at the assembly language level. Here it is all done with C code with a heavy use of pointers, including pointers to functions.

   As you read through the following description, see where the various actions are done in the program.

   The interrupt driver structure works in two levels because of the use of the interrupt controller. The processor has only one interrupt input so an interrupt controller is used to receive interrupts from

multiple devices and the interrupt controller then interrupts the processor. (In this lab, we only have the timer interrupts, so, in principle, we do not need the interrupt controller.) When the interrupt controller interrupts the processor, the processor will jump to the interrupt handler for the interrupt controller. The interrupt vector is automagically initialized for you when the libraries are generated, so you do not explicitly have to do this. If you are curious, check out `microblaze_interrupts_g.c` in the libsrc/standalone directory.

The interrupt controller then figures out what device is requesting the interrupt and jumps to a handler for that specific device. For the timer used here, that handler is called `XTmrCtr_InterruptHandler`. Look for where it is *connected* to the interrupt controller. This is a general handler for the timer that calls different *user* handlers depending on the specific timer instance that caused the interrupt, i.e., there can be more than one timer in your system.

In `lab3.c`, the actual user action taken on a timer interrupt is expressed as `TimerCounterHandler`, which is written in C and can be found in the source for `lab3.c`. You will need to provide a pointer to this function in `XTmrCtr_SetHandler`.

8. Make the appropriate changes to the source file such that it will compile. Reference `xparameters.h` and the device driver documentation or header files for help. Looking at the source for the drivers can also help you understand what the code is doing. You can find the driver sources in the bsp tree under microblaze_0.

9. Program the FPGA and watch the LED blink! Make sure you select the proper Software Configuration.

Using a Switch to Control the Flashing Rate of the LED

10. Add an additional bit (as input) to the GPIO to read the value of switch **SW1** on the board. Change the GPIO Data Bus Width parameter to 2. Having changed this generic, the range of the axi_gpio_0_GPIO_IO signal would now become `[1:0]` (Under External Ports). Add bit 1 of this signal to the `system.ucf` file and connect it to the pin for **SW1** on the ZedBoard or Atlys Board. Make sure to check the schematics of the board to find the correct pin number.

11. Rebuild the bitstream and export it to the SDK.

12. Edit `lab3.c` to use SW1 to control the flashing rate of the LED such that there are two obvious flashing rates, slow and fast. Flipping the switch should change the rate.

13. Compile the program and download an updated bitstream. Verify that the code and hardware works correctly.