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 Microblaze system that was built in Module 1 and Module 2. 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 such that 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 2.

Requirements

Module 2 Adding drivers and IP - GPIO.

Preparation

1. Find the overviews of the OPB Timer/Counter and the OPB Interrupt Controller in the Processor IP Reference Guide. From there you will find a link to the data sheets. Familiarize yourself with how these blocks work. The data sheets will also be posted on the course web page.
2. In this lab you will be modifying the file lab2b.c, which should be in the code directory of your Module 2 project. The file was in the m01.zip file you loaded for Module 1. 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.
3. You should look at the Xilinx Lecture slides starting at slide 119 for an overview of the driver organization and conventions. You can get more details in the Driver Reference Guide.

Background

This lab builds from Module 2 and assumes that the user has completed Module 2. A basic understanding of adding IP and device drivers should have been gained from Module 2.

Setup

1. Copy the XPS project directory (`lab2` folder) of the previous lab and rename the copy to `lab3`. This will be the working project directory for this lab.
2. Start XPS.
3. From the initial startup screen, select Open a Recent Project and click OK. Browse to the `system.xmp` file of `lab3`. Click Open to open the project. This is the project you completed in Lab2.

Adding A Timer/Counter And Interrupt Controller

4. Add an OPB Timer (opb_timer) and an OPB Interrupt Controller (opb_intc). Set the address range of the timer to `0x80040200--0x800402ff` and the range of the interrupt controller to `0x80040300--0x8004031f`. Attach both peripherals to the OPB bus as slaves.

5. Select Ports. The interrupt output signal of the timer should be connected to the interrupt input (Intr) of the interrupt controller. The interrupt output signal (Irq) of the interrupt controller should be connected to the interrupt input of the processor. Connect the signals by entering the same name into the Net fields for both signals. There are no parameters for each core that must be set differently than the default value.

6. Rebuild the bitstream and download it to the board.

Adding Software To Use The Timer And Interrupt Controller

7. Remove the source file `lab2a.c` from the project and add `lab2b.c`, which should be in the code directory of your project.

8. Make sure to have regenerated the Libraries. Why?

9. Using the source file `lab2b.c`, 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.

10. Compile the program and download using GDB. Verify the code works correctly.

11. If the bug with the Fast Download Methodology and GDB did not exist in EDK 8.1i, if you were to you stop your program and run it a second time, you would notice that it does not work and the message "Timer counter initialization error" would be printed to the serial port. Why do you think the initialization would fail? To find the answer, you would have to step into the `XTmrCtr_Initialize` function. However, first you would have to recompile the drivers in debug mode. Select the Software menu and the Software Platform Settings submenu. Select microblaze_0, and enter `-g -O0` in the extra_compiler_flags field would enable debug mode and turn off optimizations for the libraries.

    If you were able to step through the XTmrCtr_Initialize function after stopping and restarting the program in GDB, you would notice that it fails because the timer is already running from the last time your program was run and the function exits before completing the initialization. To fix this, you should make sure the timer is not running at the start of your program. However, you cannot call the XTmrCtr_Stop function before calling XTmrCtr_Initialize so you must use the level 0 driver interface to disable the timer. Look in `microblaze_0/include/xtmrctr_l.h` to determine how to do this and add it to the beginning of your program.

    Similarly, you should disable interrupts for the interrupt controller at the start of your program using its level 0 driver (`xintc_l.h`). Otherwise, an interrupt may occur before the drivers are initialized and your program will freeze while trying to service the interrupt.

Using A Switch To Control The Flashing Rate Of The LED

12. Add an additional bit (as input) to the GPIO to read the value of SW1 on the User Input DIP switch on the board. Change the GPIO Data Bus Width parameter to 2. Having changed this generic, the range of the opb_gpio_0_GPIO_IO signal would now become `[0:1]`. You must also make this change manually in the External Ports Connections section of the Ports tab. Add bit 1 of this signal to the `system.ucf` file and connect it to the pin for USER_INPUT1. Find the pin number in the Multimedia Board User Guide.

    Note: You must use SW1 and not SW0. SW0 is currently set as the system reset pin.

13. Rebuild the bitstream and download it to the board.

14. Use XMD to set bit 1 of the GPIO to be an input by writing to the GPIO_TRI register. Test the switch by reading from the GPIO_DATA register. **Pay close attention to the order and position of the bits. The system is big endian so the higher bit is on the right and because there are only two bits they are shifted to the right as well.** *(Hint: You may need to do some rearranging in the UCF file, or when you modify your* `lab2b.c` *file.)*

    Question: What value is in the GPIO register when SW1 is on? View the board schematic (page 25) to determine why this is true.

15. Edit `lab2b.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.

16. Compile the program and download using GDB. Verify the code works correctly.

Look At Next

Module 4: The EMAC (Ethernet MAC)