Version for EDK 8.2.02i as of January 7, 2007

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 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 OPB Timer/Counter and the OPB 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 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. Extract `m03.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.

  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. 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.

2. 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. Both cores can be left with the default configuration values for their paramters.

3. Rebuild the bitstream and download it to the board. If you deleted `lab2.c` earlier, don't forget to remove it from Project: mb0_default → Sources on the Applications tab before building.

Adding Software To Use The Timer And Interrupt Controller

4. Remove the source file `lab2.c` from the project (if you haven't already done so) and add `lab3.c`, which should be in the code directory of your project.

5. Make sure you have regenerated the libraries. Why?

6. Using the source file `lab3.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.

7. Compile the program and download using GDB. Verify that the code works correctly with the software debugger.

8. Close the software debugger and switch to the XMD window. From the XMD window, enter `dow mb0_default/executable.elf` to download the binary to the MicroBlaze. Type `run` to start running the program and confirm that it is operating as it did when you ran it via the software debugger. Type `stop` to stop the program. Look at the PC — where did the program stop (*hint: use the* `rrd` *command*)? Reset the PC to the start address of your program (remember, it's `0x800`) using the `rwr` command and type `con` to start your program over from the beginning. Notice that this time, the timer initialization fails and the message "Timer counter initialization error" is printed to the serial port. Why do you suppose this is? Restart the program, this time using the `run` command and notice the difference. What changed? Why? Can you use the debugger to explain the different behaviour between continuing from the start address of the program and running the program?

   While trying to figure out what happened, you may find it difficult to debug the driver code. If you consult your system.log or libgen.log and look for the messages relating to building the libraries, you'll notice that they're still built with optimization enabled (`-O2`) even though you disabled optimizations for your application code in Module m02. To recompile the drivers in debug mode, select the Software menu and the Software Platform Settings submenu. Select microblaze_0 from the Processor Instance: drop-down list, and enter `-g -O0` in the extra_compiler_flags field. If you rebuild the libraries, you might notice that both the `-O2` and `-O0` flags are set. If you then look at the Makefile for the drivers, you'll notice that the `EXTRA_COMPILER_FLAGS` appears after `COMPILER_FLAGS` in the build command. If you further look at the man page for `gcc`, you'll note that if multiple `-O` flags are included in the `gcc` command line, the last one takes precendence.

   You should now be able to step through the XTmrCtr_Initialize function after stopping the program, resetting the PC, and continuing from 0x800. You'll now 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

9. 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 (on the Multimedia board) or SW_1 (on the XUPV2P board). Find the pin number in your board's User Guide.

   Note: Users of the Multimedia board have no choice but to use SW1 instead of SW0 — SW0 is set as the system reset pin by default.

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

11. 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 convention in this system is to number the bits from left to right, so the highest bit is the rightmost. Since there are only two bits used in the word, they are shifted to the right as well. As such, the LSB of GPIO_DATA corresponds to `opb_gpio_0_GPIO_IO_pin<1>`.

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

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 using GDB. Verify that the code works correctly.

Look At Next

Module m04: Adding the OPB EMAC Peripheral