University of Toronto
ECE532 Digital Hardware
Module m02: Adding IP and Device Drivers — GPIO and Polling

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.

Goals

- Use Xilinx tools to add to the basic MicroBlaze system that was built in Module m01. A primary goal of this lab is to understand more details about adding IP into the system without the Base System Builder and using device drivers.
- Use the General Purpose I/O (GPIO) IP core 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.

Prerequisites

Module m01: Building a MicroBlaze System in XPS

Preparation

- The GPIO is a simple parallel I/O port for connecting to inputs and outputs outside of the chip. 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.

  The data sheet for the GPIO can be accessed through the IP Catalog tab in XPS. Look under General Purpose I/O and right click on the AXI version to open the PDF datasheet.
- The Device Driver Programmer Guide is only available via the documentation tree of an installed system. Look in `$XILINX/EDK/doc/usenglish` and open `xilinx_drivers_guide.pdf`. Xilinx provides device drivers so that the user may create applications quickly and easily. Open `xilinx_drivers.htm` to see all the drivers available.

  The Device Driver Programmer Guide documents many details of the device drivers and software infrastructure. You should get familiar with this document if you need to write your own drivers. In this lab you will be required to modify some code to properly use the existing drivers.
- You will need another 15MB of disk space. You can clean the files created in Module m01 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 m01 and assumes that the user has completed Module m01. A basic understanding of the EDK tools should have been gained from Module m01.

Setup

1. Copy the XPS project directory (`lab1` folder) of the previous lab and rename the copy to `lab2`. This will be the working project directory for this lab. Extract `lab2.zip` from the into the directory containing `lab2` to add the source files required for this module to the `lab2/code/` directory. If you like, you can delete the existing `lab2/code/` directory before extracting `lab2.zip` to clean up the files from Module m01. 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.

2. For Windows users, start XPS by going to Start → Programs → Xilinx ISE Design Suite 14.2 → EDK → Xilinx Platform Studio. For Linux users, open a terminal, run the environment scripts you created in Lab 0, and start XPS through the command xps.

3. From the initial startup screen, select Open Project. 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 m01.

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 AXI General Purpose IO 1.01.b to add it to the project. A XPS Core Config window will pop up. Clicking the PDF button opens the data sheet specification for the IP. Open the data sheet and look through what is in there. Review the parameters that can be set and the signals that constitute the hardware interface for this IP.

   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. Do not enable interrupts or enable Channel 2. In Channel 1, select the GPIO Data Channel 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. In the next Instantiate and Connect IP window, make sure microblaze_0 is selected. Click OK. You have now wired in the GPIO core to your system!

6. In the System Assembly View, select the Addresses tab. The Base Address and the High Address of axi_gpio_0 instance have been assigned automatically, change the Size of it to a smaller value, for example, 4K, which assigns the device 4K of memory space on the bus. The base address is the first address that will be decoded by the peripheral on the On-chip Peripheral Bus (AXI here).

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

7. Select the Bus Interface tab in System Assembly View. 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, place the cursor over the desired item. 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. The filled circle next to the axi_gpio_0 instance (make sure it is filled) means it is a slave on the axi4lite_0 AXI instance.

8. Select the Ports tab. Depending on the filters you have set, this view can be very lengthy. Hint: Right click the row containing the column headers and ensure Flat View is not selected. Scroll to find the axi_gpio_0 instance, expand its subtree, and select the (IO_IF)gpio_o signal. In the Connected Port dropdown box, make sure Connected to External Ports has been selected. This adds the GPIO_IO signal to the list of External Ports, assigning the name `axi_gpio_0_GPIO_IO_pin` to the external port by default. Scroll up to find the External Ports, expand its subtree to find the `axi_gpio_0_GPIO_IO_pin` instance. In the Range attribute, type [0:0]. 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. To control User LED 0 via the GPIO, the designer must ensure that the LED is connected to the same FPGA pin as the synthesized GPIO hardware.

Building the System with GPIO

9. Select the Project menu and the Project Options... submenu. Select the Design FLow tab and ensure that Verilog is selected as the HDL type in the HDL panel. Click Ok.

10. Select the Hardware menu and the Generate Netlist submenu. This will take a few minutes to complete. This step is updating the system and resynthesizing it, this time with the GPIO IP included. When the synthesis completes, the design is not yet ready to be programmed into the FPGA — recall that the netlist must be mapped, placed, routed, and converted into a bitstream first.

    Generating the netlist can be considered the *logical implementation* step of the synthesis flow, whereas the mapping, placement, and routing can be considered the *physical implementation* steps. In this step, we perform only the logical implementation step so that we can constrain our design prior to the physical implementation step — recall that the GPIO IP port must be connected to the same physical pin on the FPGA as User LED 0.

11. 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 axi_gpio_0_GPIO_IO_pin<0> LOC=*** | IOSTANDARD = LVCMOS33;
    ```

    where `***` is `T22` for the Zedboard and `U18` for the Atlys Board. Save this file once you've made your changes.

    The physical pin information of what FPGA pin is connected to USER LED 0 can be found on the board schematics. Open the schematic for your board. User LED 0 is labelled as `LD0` on the schematic. It is labeled that way on the board, so we made a good guess it would have the same name on the schematic! You can search the PDF file for this symbol. There are two pieces of information that you need to find out: (1) Verify the pin number of the User LED 0 connection; and (2) determine whether you need to drive a 1 or 0 to turn on the LED. For (1) you will find a schematic of how all the pins on the FPGA are assigned and for (2) there will be a circuit diagram on one of the schematics showing how the actual LEDs are driven from the FPGA.

    The net name is the same as the signal XPS added to the External Ports list when you selected Connected to External Ports in Step ??. The net was specified as a bus (okay, a 1-bit bus in this case, but we'll add to this in another lab so doing this now makes it easier) and the `<0>` suffix specifies bit 0 of that bus. For more information on the UCF file and constraints in general, consult the *Constraints Guide* included in the ISE User Guides documentation set.

    The `IOSTANDARD` configures the pin to use `LVCMOS33` voltage levels. Note that FPGA pins can be configured to many IO standards so that they can interface to many types of chips and devices. However, the FPGA is set up in *Banks* of pins and all of the pins in a bank must use compatible voltage standards according to the voltages used to power the bank. Otherwise, you will get an error at the placement stage. We use `LVCMOS33` to be compatible with the other signals such as the others in the UCF file, and it is enough to drive the LED.

12. Once the UCF file is ready, it's time to do the *physical implementation* part. Select the Hardware menu and the Generate Bitstream submenu. It will take several minutes.

Testing the GPIO in Hardware

13. After the bitstream is successfully downloaded, you can use XMD to test the system to ensure that the GPIO is working. The GPIO registers are memory mapped, so reads and writes of the GPIO registers can be done using the read (`mrd`) and write (`mwr`) memory commands. The GPIO datasheet states that the GPIO_DATA register is at offset `0x00` and the GPIO_TRI register is at `0x04`. In Step **??**, the base address of the GPIO was assigned. You can open the Addresses tab again in XPS to check its value. Add the offsets to the based address to determine the actual register addresses.

    The tri-state register for the GPIO must be loaded with a value to configure the LSB of the register as an output as described in the GPIO datasheet. There is a way to specify the value that the tri-state register takes at power-up; did you notice it when configuring the GPIO peripheral in Step **??**? Double click on axi_gpio_0 in the Bus Interfaces tab to open up the Core Config window for the GPIO core. Click on Channel 1 to open its parameters to see the default values. Is the default an input or output?

    Read the GPIO_TRI register and see whether it is configured correctly. If not, write the correct value.

    Write the GPIO_DATA register with the appropriate values to turn the LED on and off. This should validate whether the hardware is correctly configured.

Adding Software to Use GPIO

14. As you did in Lab 1, after you finish generating the bitstream export your design to SDK. Change the workspace directory to point to your `lab2` directory. In the Xilinx SDK tool, create a New Xilinx C project called *blink_LED_0* and choose Empty Application as the Project Template. Click Next. Change the Project name of the Board Support Package to *blink_LED_bsp_0*. Click Finish.

15. Next, you need to add the source file into the *blink_led* project. In the Project Explorer expand the *blink_LED_0* project tree, right click on the src, create a new file by going to New → File. Click Advanced and link the source file you downloaded from the course website (lab2.c). Click Finish.

    In the console window, you will see that there is an error. That is because the file does not yet compile. You will need to make some changes.

16. The goal of this exercise is to get you to examine some of the relevant files dealing with the drivers.

    Reference the device driver documentation in the *Device Driver Programmer Guide* to determine the naming convention of the device driver header files. Look under Preparation above to find the location of the guide. You can browse to the drivers included in the XPS project by looking in the `blink_LED_bps_0/microblaze_0/include` subdirectory of the project.

    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. You'll need to figure out the driver include file and some of the values can be found by referencing the `xparameters.h` file for system-wide constant definitions. Also, you will need to recall the offsets for the registers in the axi_gpio block. The `xparameters.h` file is also in the `blink_LED_bps_0/microblaze_0/include` directory.

    The `GPIO INSTANCE POINTER` is a structure of type `XGpio`. It is defined in the driver include file. Find the declaration of this structure and read the comment for a further hint as to how to modify your source file. You will need to have some familiarity with C programming for this. Take care that you are correctly passing a pointer.

    The `blink_LED_bps_0/microblaze_0/libsrc` directory will have the source code for the driver that you may also wish to consult.

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 more 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.

The `lab2.c` source code uses the GPIO driver to turn on and off the LED on the board at a visible rate. Save the file and SDK will auto-build the program. Note that XPS won't automatically save a changed file before building, so don't forget to save your changes before you build... always!

18. If an overflow error occurs when the project is building you have to reduce the size of the heap and stack in the linker script. Right click on the project (`blink_LED_0`) and select Generate Linker Script. Change the size of the stack and heap to half the value (from 1024 bytes to 512 bytes). Click Generate and Yes to overwrite the file.

    If the problem still persists, simply keep on reducing the size of the stack and heap by half until the error no longer occurs.

19. After you finish building the software without any compilation error, you should see the message *"Finished building: blink_LED.elf.elfcheck"* in the Console tab of the message window (typically in the bottom).

    From the SDK, you should now be able to go to Xilinx Tools→Program FPGA. Be sure to select the correct Software Configuration.

20. Download the bitstream to the board and you should see LED_0 blinking on your board. Hit the reset button if you want to start the program again. You will have to watch carefully to see the Layer 0 example fo turning the LED on and off.

Other Things To Consider

- What version of the MicroBlaze processor IP is being used?
- What is the base address of the UartLite peripheral (*hint: its instance name is RS232*)?
- What is the instance name of the xps_mdm peripheral?
- What is the net name used for the system clock in the design?
- 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 bus controllers*)?

Look At Next

Module m03: Adding IP and Device Drivers — Timers and Interrupts