San Jose State University
Department of Electrical Engineering
EE178, Fall 2007, Crabill

Laboratory Assignment #5

Objectives

The objective of this lab is to expand your comfort zone with the PicoBlaze embedded processor from Xilinx. Figure 1 shows a block diagram of the system as it is provided to you. The omnipresent clock and reset signals have been omitted.

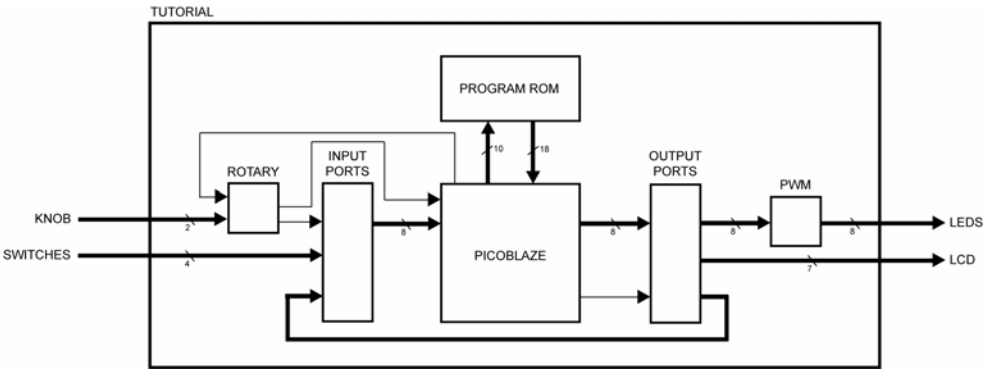

Figure 1: System Block Diagram

The main task for this lab is to write software in PicoBlaze assembly to implement a specified function. While this may sound familiar, you will find that you need to write substantially more code for this function than in previous lab assignments. The lab also introduces the concept of PicoBlaze interrupts and a method for rapidly reloading the program ROM to facilitate efficient code debugging.

When you successfully complete this lab, you have gained additional experience with PicoBlaze, hopefully enough to enable your successful completion of the final lab assignment.

Bibliography

This lab uses the Verilog-HDL version of PicoBlaze; you can read about PicoBlaze on the Xilinx website at http://www.xilinx.com/picoblaze. Additionally, the hardware system and some software routines used in this assignment are derived from PicoBlaze examples for the Spartan-3E Starter Kit board, also obtained from the Xilinx website. The files you need to complete the lab assignment are posted on the class website; you do not need to download anything from Xilinx.

Obtain and unzip the file archive from the class website and use the resulting directory as your project folder. You can move the directory anywhere you please as long as there are no spaces in the path to the directory.

System Description and Requirements

In this system, you are not allowed to use latches. You are allowed to use only one clock and only one asynchronous reset signal. The clock must be the 50 MHz clock signal available from the oscillator on the Spartan-3E Starter Kit board. **You will receive zero points if you do not follow these requirements.**

As shown in Figure 1, the system has a number of inputs. There are clock and reset inputs, plus a 4-bit switch input and two inputs from the rotary knob.

clk	clock signal, 50 MHz from oscillator
rst	reset signal
switches[3:0]	4-bit switch input
rot_a	rotary knob “a” signal
rot_b	rotary knob “b” signal

Also shown in Figure 1 are the outputs. There is an 8-bit LED output and several signals used to control the LCD display. One of these signals is permanently tied to logic one.

leds[7:0]	8-bit LED output
vcc	logic one, disables other devices on LCD data bus
lcd_db[7:4]	4-bit LCD data bus
lcd_e	LCD enable strobe
lcd_rs	LCD register/ram select
lcd_rw	LCD read/write select

You must successfully implement the system from the provided files, and then modify and complete a small program. Your final software implementation is required to display your name on the LCD and then allow the user to adjust the brightness of the LEDs using the rotary knob.

System Hardware Project

The first step is to create a hardware system using Project Navigator. To do this, you need to create a project and then “add existing source” files. This lab shares a great deal in common with the previous PicoBlaze based labs. However, there are some subtle differences, so do not attempt to reuse files from earlier labs. Use the files provided specifically for this lab.

In the subdirectory for the assembler, there is a template for an assembly program, software.psm, and a batch file to assemble it. This template is syntactically correct but incomplete – you will return to modify and then complete it later. Assemble the program template by double clicking on the “assemble” batch file. The assembler generates an additional source file, software.v. This source file contains a BlockROM initialized with executable code generated by the assembler from software.psm. Additionally, this batch file invokes a few additional programs that were not used in previous labs.

Now, start a new project with Project Navigator. Then, “add existing source” for the system from the project directory. There are seven necessary Verilog-HDL files, and one UCF file:

1. tutorial.v (top level system module)
2. testbench.v (testbench for behavioral simulation of top level system module)
3. kcpsm3.v (the PicoBlaze sub-module)
4. software.v (the PicoBlaze software sub-module)
5. pwm.v (the PWM controller)
6. spinner.v (the Knob processor)
7. synchro.v (a synchronizer)
8. tutorial.ucf (top level system constraints)

The resulting system is very similar to the original PicoBlaze tutorial design, but has two major differences. The first difference is the addition of a module to interface with the rotary knob. This interface consists of synchronizers, a quadrature decoder, and some random logic to create interrupts for the processor.

The second difference is the use of a single pulse width modulator circuit to drive the LEDs. The PWM circuit receives an 8-bit value and generates an output waveform with a duty cycle proportional to the 8-bit value. This is very similar to the DAC module used in a previous assignment, and allows for the intensity

of the LEDs to be adjusted. Note that the single output of the PWM is used to drive all eight LEDs in parallel so that it would be easy to see changes in intensity.

System Functional Simulation

You should perform some minimal functional simulation of the system. This is important for two reasons. First, it will give you confidence your system is working properly before you implement it. Second, if the system does not behave as expected when you download it, you will have a mechanism to quickly create additional test cases to help debug the problem. The instructor will not help you debug software problems (incorrect system behavior) unless you are able to run a simulation. A simple test bench is included with the downloadable support package for this lab. Feel free to enhance the basic test bench as you see fit.

When you run a simulation, you should be able to watch the PicoBlaze processor fetch instructions from the program ROM. The software-driven LCD activity, however, is dreadfully slow. You would have to run the hardware simulator for a substantial amount of time to see waveform results that are useful. For this reason, consider running a short simulation as a sanity check to make sure you have some reason to expect the system might work, but perform extensive verification of the system in hardware.

System Synthesis and Implementation

Synthesize the system as you have done before. Do not forget to check the synthesis report. You should expect to see the PicoBlaze module generate a fair number of warnings about "simulation mismatch" which may be safely ignored.

Implement the design, and then generate a programming file. Next, create a PROM programming file, and then program your design into the PROM. When the PROM has been successfully programmed, close iMPACT and then also close Project Navigator. Cycle the power on the board, but leave the USB programming cable attached. You should see the instructor's name and the lab information appear on the LCD display.

System Software Development

You must modify and complete the PicoBlaze assembly program to satisfy the software requirements. The program template contains a number of constant definitions for your convenience and is structured with the "skeleton" of an interrupt service routine (ISR) at the very end of the software.psm file.

First – the easy part. Edit the existing LCD message routines in the software.psm file to display your name instead of the instructor's name. Assemble the program by double clicking on the "assemble" batch file. Assuming the assembly was successful, download the new program to your board by double clicking on the "download" batch file. You should see your modifications reflected on the LCD display.

Notice how you did not need to invoke Project Navigator to re-run the full implementation flow. Now you are empowered to try out code changes and quickly evaluate the results. Note, however, that these changes are temporary and not stored in the non-volatile configuration PROM. They are "lost" when you turn off the power.

Second – the more complex part. Open the software.psm file and scroll to the bottom where you will find the interrupt service routine. When the processor receives an interrupt, it will temporarily suspend what it was executing in the main loop and jump to the interrupt service routine. A typical interrupt service routine will "service" the external source that generated the interrupt, clear the interrupt, and then exit – allowing the processor to resume what it was doing before the interrupt occurred.

In this system, the knob processor creates an interrupt event every time the knob is rotated. You must fill-in the interrupt service routine with code that reads the knob direction status and then updates the value provided to the PWM circuit which is driving the LEDs. Use the following hex values for programming

the LEDs: 00, 01, 02, 04, 08, 10, 20, 40, 80, FF. This sequence begins at 00 (LEDs completely off) and ends at FF (LEDs at full brightness). The intermediate values yield intermediate intensities for the LEDs. Rotating the knob clockwise should increase the intensity, and rotating the knob counter-clockwise should decrease the intensity. The interrupt service routine must include bounds checking to make sure that the value provided to the PWM circuit never "rolls around". In other words, any user attempt to decrement past 00 has no effect and any user attempt to increment past FF has no effect.

It is important to note that an interrupt service routine truly does "interrupt" the execution of the main program loop. If any actual work is being done by the main loop, the interrupt service routine must leave the processor registers "as it found them" so that when the interrupt service routine exits, and the main loop is resumed, the main loop is unaware that it was suspended. Obviously, the interrupt service routine will need to use the registers to accomplish its task, so it would have to save the existing register values when it starts and then restore them when it is done.

One way to handle this is to have the interrupt service routine perform what is called a context save. For instance, the interrupt service routine might start with some code that stores all 16 processor registers "somewhere" before proceeding to modify any of them. Immediately before the interrupt service routine exits, it would then perform what is called a context restore. For instance, the interrupt service routine would load all 16 processor registers from "somewhere", thereby restoring the registers. An example of a safe "somewhere" for storing the context is in the PicoBlaze scratchpad RAM. If you are not familiar with this, consult the PicoBlaze manual to read more about it.

In this programming assignment, the main loop does no work (it's a tight loop consisting of a single jump instruction). Therefore, it is not necessary for the interrupt service routine to do any context save/restore.

Final Steps

Once you have arrived at a program that satisfies the requirements – open Project Navigator and re-implement the design. This will create a new bitstream that contains your final version of the PicoBlaze program. Then generate a programming file. Next, create a PROM programming file, and then program your design into the PROM. When the PROM has been successfully programmed, close iMPACT and then also close Project Navigator. Remove the USB programming cable, cycle the power on the board, and test your design.

Laboratory Hand-In Requirements

Once you have completed a working design, prepare for the submission process. You are required to demonstrate a working design which has been programmed into the PROM. Within nine hours of your demonstration, you are required to submit your entire project directory in the form of a compressed ZIP archive. Use WinZIP to archive the entire project directory, and name the archive l5_yourlastname.zip. That is "l5" as in Lab 5, **not** "15" as in fifteen. For example, if I were to make a submission, it would be l5_crabill.zip. Then email the archive to the instructor. Only WinZIP archives will be accepted. If your archive is too large, you may remove the subdirectories in the project folder.

Demonstrations must be made on the due date. If your circuit is not completely functional by the due date, you should demonstrate and submit what you have to receive partial credit.