San Jose State University
Department of Electrical Engineering
EE178, Spring 2006, Crabill

Laboratory Assignment #4

Objectives

The objective of this lab is to gain familiarity with the PicoBlaze embedded processor from Xilinx. PicoBlaze is a piece of intellectual property owned by Xilinx. Xilinx makes PicoBlaze available to its customers for free, as what is called an IP core – or "core" for short. The idea is that PicoBlaze implements a function applicable in many systems, and Xilinx customers may be able to re-use it in their own designs. Xilinx provides a large variety of IP cores, many of which are free.

In this lab, you will implement an embedded processor system with several peripherals. From a hardware perspective, most of the system is provided for you. You are encouraged to read the hardware description of the system, however, so that you understand it. This is particularly important because you will likely encounter PicoBlaze in the future. Figure 1 shows a block diagram of the system. The omnipresent clock and reset signals have been omitted.

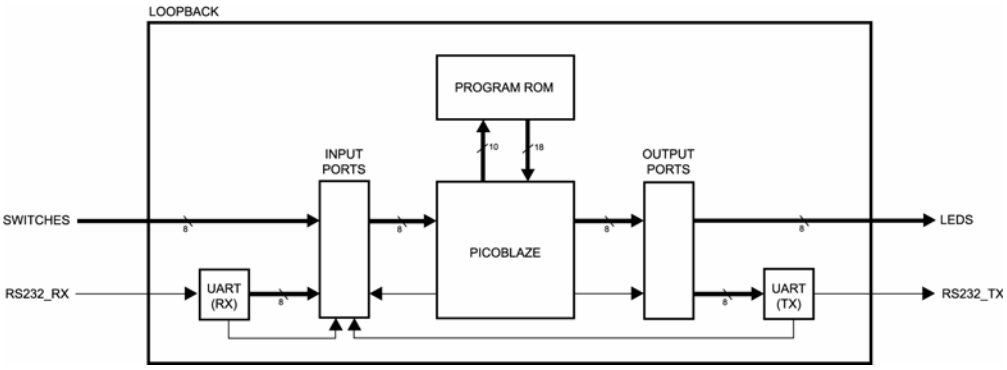

Figure 1: System Block Diagram

The main task for this lab is to write software in PicoBlaze assembly to implement a loopback test. A loopback test is a test in which a signal is sent to a device and returned back from that same device as a way to determine whether the device is working correctly.

The first loopback test will echo inverted switch settings on LEDs. Here, you are transmitting with your fingers, and receiving with your eyes what is looped back by the system. The second loopback test will echo serially received data over an RS232 serial port. Here, a desktop computer is transmitting with its serial port, and receiving with its serial port what is looped back by the system.

When you successfully complete this lab, you will have gained an understanding of how to use PicoBlaze to implement a small embedded processor system.

Bibliography

This lab uses the Verilog-HDL version of PicoBlaze, as well as the peripherals that are included with it. These items are available from http://www.xilinx.com/picoblaze. Additionally, the top-level system framework is derived from the PicoBlaze examples for the Spartan-3 Starter Kit board. Completion of this lab requires the Verilog-HDL source files and assembler provided with PicoBlaze for Spartan-3.

PicoBlaze Processor

You will need to download the PicoBlaze processor from the Xilinx website. There are several versions, make sure you obtain the version for use with Spartan-3. There is also a downloadable support package for this lab. Unzip the downloadable support package, and then unzip the PicoBlaze download inside the unzipped support package folder. You should carefully read the presentations and then skim the *PicoBlaze 8-bit Embedded Microcontroller User Guide* before proceeding.

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-3 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 is a clock and a reset input, plus an 8-bit switch input and a serial receive input. The serial receive input originates from the RS232 connector on the board and passes through a voltage level translator before reaching the FPGA device.

clk	clock signal, 50 MHz from oscillator
rst	reset signal
rs232_rx	serial receive input
switches[7:0]	8-bit switch input

Also shown in Figure 1 are the outputs. There is an 8-bit LED output and a serial transmit output. The serial transmit output originates from the FPGA and passes through a voltage level translator before reaching the RS232 connector on the board.

rs232_tx	serial transmit output
leds[7:0]	8-bit LED output

You must successfully implement the system from the provided source files, and then develop a small software program. The software development is split into three parts. Your final software implementation is required to transmit a message upon reset, and then concurrently perform two loopback functions:

- Echo switch settings, inverted, on LEDs
- Echo serially received data over an RS232 interface

In order to complete this lab, you will need to provide your own RS232 serial cable. It must be a standard DTE to DCE cable, often called a pass through cable, which is often used with an external modem. Do not use a null cable. The appropriate cable may be purchased at a variety of local electronics stores.

System Hardware Project

The first step is to create a hardware system using Project Navigator. To do this, you need to gather all the source files. While you were reading the documentation, you likely noticed the test bench and top level source files, as well as the subdirectories containing the assembler and PicoBlaze source files.

In the subdirectory for the assembler, there is a template for an assembly program, program.psm, and a batch file to assemble it. This template is syntactically correct but functionally useless – you will return to complete a meaningful program later. Assemble the program template by double clicking on the batch file. The assembler generates an additional source file, program.v. This source file contains a BlockROM initialized with executable code generated by the assembler from program.psm.

Now, start a new project with Project Navigator. Then, copy the complete set of source files for the system into the project directory. There are nine necessary Verilog-HDL files:

1. loopback.v (from support package)
2. testbench.v (from support package)
3. kcpsm3.v (from PicoBlaze Verilog directory)
4. program.v (from Assembler directory)
5. uart_rx.v (from PicoBlaze Verilog directory)
6. uart_tx.v (from PicoBlaze Verilog directory)
7. kcuart_rx.v (from PicoBlaze Verilog directory)
8. kcuart_tx.v (from PicoBlaze Verilog directory)
9. bbfifo_16x8.v (from PicoBlaze Verilog directory)

Using Project Navigator, add all the source files to the project in the order listed above. As you add the files, you will notice that Project Navigator actually reads the source and attempts to show you what modules it thinks are missing by marking them with a question mark in the Sources in Project window. When you are done with this step, no question marks should remain. At this point, also create a new implementation constraint file. You should have a project hierarchy matching that shown in Figure 2.

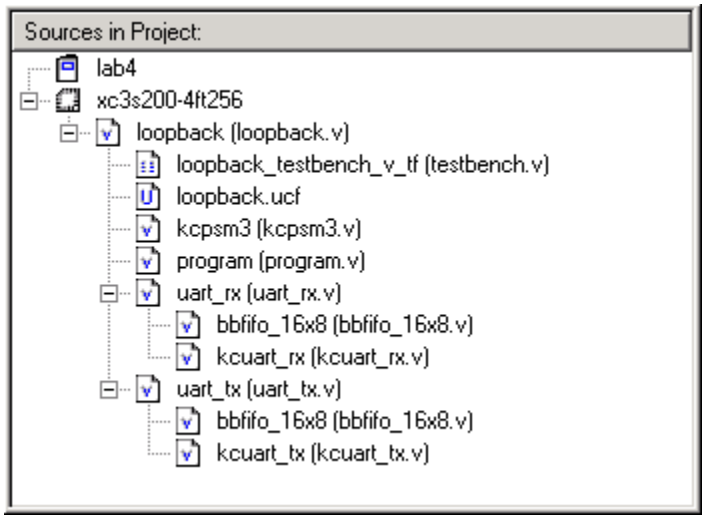

Figure 2: Sources in Project

As mentioned before, the system is designed for you. However, you will find that the top level module is missing a few details. Read the comments in the file and fill in the missing details. If you have further questions, or need clarification, consult the instructor.

Initial System Synthesis and Implementation

Synthesize the system as you have done before. Do not forget to check the synthesis report. Before you implement the system, you will need to edit the constraints file and assign I/O locations and properties. You must figure these out for yourself.

For all inputs and outputs, do not forget to properly specify the I/O standard as LVCMOS33. Once you have saved the constraint file, implement the system to make sure no errors occur. This implementation is only a test – do not try anything on the hardware yet!

System Software Development

Now you must develop a PicoBlaze assembly program to satisfy the three software requirements. The program template contains a number of constant definitions for your convenience and is structured so that

you can implement each of the three requirements independently. Each requirement is listed as a task, some are easier than others.

You are advised to work on the tasks one at a time; verify your results in hardware before moving on to the next task. If you follow this advice, you will go through at least three iterations of software development, functional simulation, synthesis, implementation, timing simulation, programming file generation, and hardware verification.

```
;===================================================================
; Actual assembly program goes here...
;===================================================================

cold_start:     LOAD     s0, s0           ; (nop)

                ; LAB TASK #1
                ; Write code to output a message to the
                ; serial port.  The message must be longer
                ; than 25 characters.

led_echo:       LOAD     s0, s0           ; (nop)

                ; LAB TASK #2
                ; Write code to read the switch state and
                ; write it, inverted, to the LED output port.

rs232_echo:     LOAD     s0, s0           ; (nop)

                ; LAB TASK #3
                ; Write code to check if a byte has been
                ; received by the uart.  If so, write it
                ; back to the uart transmit port.  Then...

                JUMP     led_echo         ; endlessly loop

;===================================================================
;
;===================================================================
```

After you have written the code to implement a given task, you must re-run the batch file to assemble the complete program. This will generate a new program.v file, and you will need to remember to copy it into the Project Navigator project directory every time you re-assemble the program. If you tend to forget these kinds of details, consider adding a copy command in the batch file which will copy the resulting program.v file to the project directory. Or, you can actually move the complete contents of the assembler directory into the project directory, and run it there – the program.v file will appear in the correct location by default.

System Functional Simulation

You must 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. The test bench can be used for functional simulation as well as timing simulation after the implementation step. 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. Serial port activity, however, is dreadfully slow because data is transferred at 9600 bits per second. You would have to run the hardware simulator for hours to see 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. Then verify the system in hardware.

System Synthesis, Implementation, and Timing Simulation

Synthesize and implement the project again. This is necessary every time you re-assemble the program because the resulting program.v source file will change. If you wish, you may perform a timing simulation at this point. You should see the same behavior in timing simulation as you did in functional simulation.

System Hardware Verification

Generate a programming file and proceed to verify the system in hardware. To verify the switch to LED loopback behavior, exercise the switches and observe the results on the LEDs. You should test that all eight switches and all eight LEDs behave as expected.

To verify the RS232 serial port behavior, connect the Spartan-3 Starter Kit board to the COM1 serial port on your desktop computer using a serial cable. Use the provided terminal.ht session file to launch the HyperTerminal application. If you need to use some other COM port you will need to change the HyperTerminal session options. Otherwise, leave the settings unchanged.

Upon hardware reset of the PicoBlaze system, you should receive the message in the HyperTerminal window. Note that the reset push button is not de-bounced; sometimes, when the button is released, you may receive a few garbage characters, in addition to the message, in the HyperTerminal window. This may be safely ignored as long as you are also receiving the message. If you do not receive the message, verify that you have correctly filled-in the missing information in the top-level module. If you still do not receive the message, ask the instructor to test your cable.

Afterwards, if serial loopback is implemented, anything you type in the HyperTerminal window should be echoed back to the window. If loopback is not implemented or not functioning properly, you will see nothing in response to typing in the HyperTerminal window.

When you have completed all three tasks, generate a programming file for the PROM and then load your system into the PROM.

Laboratory Hand-In Requirements

Once you have completed a working system, prepare for the submission process. You are required to demonstrate a working system which has been programmed into the PROM. Within six 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 l4_yourlastname.zip. For example, if I were to make a submission, it would be l4_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 xst subdirectory (temporary synthesis files)
- The work subdirectory (temporary simulation files)
- Any file with a .wlf extension (simulation waveform files)

Demonstrations must be made on or before the due date. If your circuit is not completely functional by the due date, you should turn in what you have to receive partial credit.