Microprocessor-Based Systems (E155)

Harris

Lab 1: Hexadecimal Display Driver

Introduction

As the first lab for E155, this assignment will be an introduction to what we will be doing throughout the semester. It will teach you how to use the Xilinx Foundation Series software to design circuits. In doing so, it will review knowledge acquired in Introduction to Computer Engineering. This handout will walk you step-by-step throughout this first lab, so that you can get used to the tools. This walk-through will also be done in the second lab, but from then on you will be mostly on your own.

Objectives

- To introduce the Xilinx Foundation Series software
- To review digital design;
- To construct a hexadecimal seven-segment display driver.
- To prepare a circuit that will test the FPGA board to be use throughout the course. This circuit should test the 8 LEDs, the DIP switches and the clock of the FPGA board.

Requirements

Design and simulate a circuit with 4 input bits (representing the hex numbers 0 through F) that outputs 7 bits to show that number (or character) on a 7-segment display. The circuit should also display the inputs and the clock on the 8 LEDs that will be provided on the FPGA board. Refer to **Tables 1 and 2** for details on the input/output requirements. *Remember:* No physical circuit will be designed in this lab; this will be only a simulation. In lab 2 you will assemble your FPGA board and test it with your design from this lab.

Background

The purpose of this first laboratory assignment is to introduce the Xilinx software and its tools. The Xilinx software allows the user to enter a design, simulate it, and then program it into a chip.

This year we will be using the Xilinx 8.2i Integrated Software Environment (ISE) tools, which are the leading tools used for commercial FPGA design. FPGA design consists of several stages: design entry, simulation, (possibly synthesis), implementation, and downloading the design to the chip. Some of these stages are supported directly by Xilinx, while others use third-party tools. Specifically, we will use the ModelSim tool from Mentor Graphics for simulation and the Synplify tool from Synplicity for Verilog synthesis.

Two methods of design entry include schematics and hardware description languages (HDLs). In E85 you used a schematic editor. In this class you will use the Verilog hardware description language to describe digital functions in a textual form. HDL code must be synthesized to convert it to a "netlist" before it can be programmed into an FPGA.

The Simulation in the Xilinx software allows the user to verify the design by assigning values to the inputs of the design and checking that the outputs correspond to what is expected. This way the user can save precious time before having to physically set up the circuit. Most digital systems are complex enough that it is a safe assumption they have bugs until proven correct.

This lab focuses on design entry, simulation, and synthesis. Lab 2 involves downloading the design and testing it on the real hardware.

Discussion

Now that you have an idea of what the Xilinx software can do, let's look at our requirement for this lab. This discussion will walk you through half of the lab, but then you will be on your own.

In order to check the 8 LEDs on the board, we'll use them in the following way:

Let S4, S3, S2 and S1 represent the 4 input bits and let CLK represent the clock. LED1 through LED8 will represent the 8 LEDs available. Table 1 shows the LED outputs you are required to program for this lab. These LED outputs are unrelated to the 7-segment display outputs you will also program. Note that building this table requires 3 inverters (for LEDs 2, 4, and 6) and an AND gate (for LED7).

S1	LED1	LED2
0	OFF	ON
1	ON	OFF

S2	LED3	LED4
0	OFF	ON
1	ON	OFF

S3	LED5	LED6
0	OFF	ON
1	ON	OFF

S4	S3	LED7
0	0	OFF
0	1	OFF
1	0	OFF
1	1	ON

CLK	LED8
0	OFF
1	ON

Table 1 – I/O requirements

You are also responsible for designing the 7-segment display decoder that reads the four-bit number specified by the input switches and illuminates the appropriate segments on the display. To understand the relationship between the 4-bit input and the output to the 7-segment display, you should complete **Table 2** with the necessary information. Again, the 4-bit input will be represented by S4 through S1, and the output to the display will be signals A through G in the following fashion:

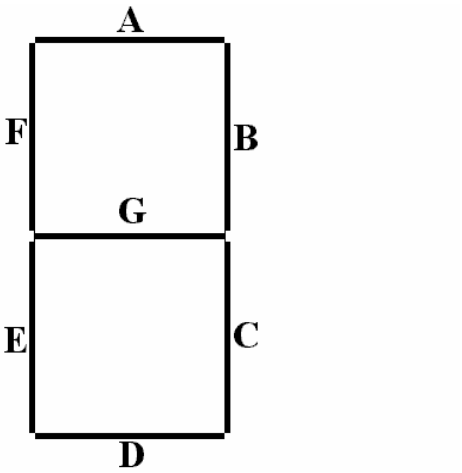

Figure 1 – Definition of outputs A through G for the 7-segment display

You should come up with the shape of the hexadecimal numbers based on the above standard for the outputs. Hint: Be careful to distinguish between the numbers 6 and 8 and the letter b.

Hex Number	Input bits				Output bits to display						
	S4	S3	S2	S1	A	B	C	D	E	F	G
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1							
2	0	0	1	0							
3	0	0	1	1							
4	0	1	0	0							
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0							
7	0	1	1	1							
8	1	0	0	0							
9	1	0	0	1							
A	1	0	1	0							
B	1	0	1	1							
C	1	1	0	0							
D	1	1	0	1							
E	1	1	1	0							
F	1	1	1	1							

Table 2 – Requirements for output to the display

Now that you know the inputs and outputs for the design let's create the schematic file for it. To do this, you will need to run the Xilinx 8.2i software, which is available on the workstations in the Microprocessor Lab. Enter the Xilinx software by invoking the Project Navigator.

Begin by creating a new project (File • New Project). Name it lab1_xx (where xx are your initials) and store it in your Charlie directory. Be sure there are no spaces in the

path name, or bad things will happen later. Set the Top-Level Source Type to HDL. Next select the Family to be Spartan3. Set select "XC3S400" as the Device, "TQ144" as the Package, and Speed Grade of "-4". This allows Xilinx to target the 144-pin Spartan 3 FPGA running at a standard speed. Set the Synthesis tool to Synplify Pro VHDL/Verilog and the Simulator to Modelsim-SE Verilog. This configures Project Navigator to use the 3rd party Modelsim for simulation and Synplify Pro to synthesize your Verilog code.

Create a New Source file (Project • New Source…). A wizard will walk you through the skeleton of the file. The source file should be a Verilog Module named led.v in the same directory as the project. Create 3 I/O ports: a single-bit clock input, a 4-bit bus of switch inputs, and an 8-bit bus of led outputs.

```
clk    input
s      input  3      0
led    output 7      0
```

When you finish the wizard, the led.v file will appear selected in the upper left sources pane. Below it, a set of processes are available for design entry, synthesis, implementation, and programming. Watch the log window at the bottom and learn what the normal messages are. If you encounter difficulties using Xilinx, look in the log for warnings and clues about your problem. To the right, the fourth pane should contain a skeleton of your led.v Verilog module. Xilinx adds a bunch of useless comments at the top. You may wish to delete all but your name and the creation date and any of your own comments about what the module does.

Add the following bold lines of Verilog code so you module should read as shown below. Note that in Verilog busses usually start at 0 instead of 1 so the numbering is shifted by one. Use File • Save to save your code when you are done

```
module led(clk,s,led);
    input clk;
    input [3:0] s;
    output [7:0] led;

       assign led[0] = s[0];
       assign led[1] = ~s[0];
       assign led[2] = s[1];
       assign led[3] = ~s[1];
       assign led[4] = s[2];
       assign led[5] = ~s[2];
       assign led[6] = s[2] & s[3];
       assign led[7] = clk;

endmodule
```

This example shows how to express combinational logic in Verilog. Common operators include ~ (NOT), & (AND), | (OR), and ^ (XOR). For example, to express

$$Y = \overline{(A+B)} \oplus C\left(D + \overline{E}\right)$$

write

```
assign y = (~(a|b))^(c&(d|~e));
```

Source files may be used for synthesis, simulation, or both. It is a good idea to simulate a file before synthesizing it and testing it on hardware because many bugs are

faster to find in simulation. There are several ways to apply stimulus to a design in simulation, including manually typing values into the simulator, creating test bench waveforms, and creating a Verilog testfixture.

To simulate the led module manually, use the pull-down menu in the Sources pane to choose Sources for Behavioral Simulation. Click on led.v to select that source. In the Processes Pane, expand the ModelSim Simulator item. Double-click on the Simulate Behavioral Model. ModelSim will start up in a new window with several panes, including the command/transcript pane and the wave pane. Look through the log in the transcript pane for any errors. If there are serious problems, the bottom of the window may say <No Design Loaded> and messages in the window may give a clue about your typo. Close ModelSim, fix your Verilog, and launch ModelSim again. In the command pane, type the following commands:

```
force led/s 0000
force led/clk 0
run 100
force led/clk 1
run 100
force led/s 1111
run 100
```

Look at the waveform window and verify that the led waveforms match your expectations. You may have to zoom in to read all the bits in the waveforms. Close the ModelSim command window when you are done.

Next, to simulate the led module with a test bench waveform, create a New Source of type Test Bench WaveForm. Name it led_tbw.tbw and put it in the same directory with the rest of your project. Associate the new source with led. When the Initial Timing and Clock Wizard pops up, click Finish to accept the default clock information. A graphical waveform editor will appear. The clk signal is automatically associated with a clock. Click on the s wave at various points to set some values for it. If you wish to change the length of the simulation, use the Test Bench • Set End of Test Bench menu. Save your waveform file when you are done.

Look at the Sources pane. led_tbw should appear under Sources for Behavioral Simulation but not Sources for Synthesis/Implementation because it is only for simulation. In the Sources for Behavioral Simulation view, it should have a Unit Under Test (UUT) called led hierarchically under the test bench. Select led_tbw and choose Simulate Behavioral Model in the Processes pane. ModelSim will pop up and simulate the led module using the waveforms you have specified. Again, check for any problems in the transcript pane. Zoom until you can clearly read the waves, then check for the correct outputs. Close ModelSim when you are done.

An advantage of the test bench waveform technique is that it is easy to view the waveforms being specified. You can also click on outputs to specify expected values. ModelSim will report when the actual outputs don’t match your expectation. Test bench waveforms are the easiest way to create stimulus for simple designs.

For more complex designs, it is helpful to have more control over the simulation stimulus. Testfixtures are usually used for these designs. To make a test fixture, create a new source of type Verilog Test Fixture. Name it led_tf.v and associate it with led. A

Verilog template will appear that instantiates the led module and applies some initial input values. Add your own stimulus as shown in bold below. 4'b1111 indicates the 4-bit binary number 1111. #100 indicates that the testfixture should wait for 100 ns before performing the next command.

```
module led_tf_v;

	// Inputs
	reg clk;
	reg [3:0] s;

	// Outputs
	wire [7:0] led;

	// Instantiate the Unit Under Test (UUT)
	led uut (
		.clk(clk),
		.s(s),
		.led(led)
	);

	initial begin
		// Initialize Inputs
		clk = 0;
		s = 0;

		// Wait 100 ns for global reset to finish
		#100;

		// Add stimulus here
		clk = 1;
		#100;
		s = 4'b1111;
		#100;

	end

endmodule
```

Save the testfixture. Select it in the Sources for Behavioral Simulation panel, then in the Processes Panel choose simulate Behavioral Model. ModelSim will again pop up and perform the simulation. Check for problems in the transcript, then click on the Zoom Full button to see the entire results and check if they match expectations.

As we will discuss later, it is also possible to create testfixtures that read stimulus from a file and self-check against intended outputs.

Now you are ready to synthesize your led module. Being sure the led.v source is selected in the Sources for Synthesis/Implementation pane, double-click on Synthesize in the processes pane. This launches Synplify Pro to synthesize your Verilog into hardware. Look at the log pane and check that synthesis completed successfully. The Synthesize process should also be marked with a green check if it was successful. If you encounter errors, look through your led.v file for any typos.

Click on the + symbol to expand the Synthesize options, then double-click the View Synthesis Report item. It is wise to check this report to make sure there are no errors and to see what your circuit produces. The report appears as the led.srr file in the Xilinx Project Manager. The target clock frequency defaults to 1 MHz. You should find that one LUT2 (a two-input Lookup Table) is used to perform the AND operation. There

are also five input buffers (IBUFs) for clk and s[3:0] and eight output buffers (OBUFs) for led[7:0]. In summary, only one LUT and no registers should be used in this design.

Synplify Pro has a helpful capability of showing you how your Verilog synthesizes into gates and maps onto the FPGA. Under the Launch Tools suboption, double-click View RTL Schematic. Look at the window where you should see the input and output terminals and the AND gate and inverters. See how the bus labeling is illustrated and how the inverter is marked [2:0] to indicate that there are actually 3 inverters. When you understand this view, close the Synplify window. Then choose View Technology Schematic. This shows how the Verilog code will map onto hardware within your FPGA. You should see IBUF and OBUF cells corresponding to the input and output buffers attached to pins on an FPGA. There are also three INV cells and an AND gate. Again look at the bus labeling and explore the diagram until you understand it. By looking at the schematics produced by synthesis, you may be able to debug some errors you make in your Verilog in later labs. Close the window when you are done.

The last step is to implement your project, which maps the gates onto lookup tables on an FPGA. Double-click Implement Design in the Processes pane and look at the messages in the console window. There should be no warnings or errors. Don't take warnings lightly unless you understand them enough to know they can be ignored; many engineers have lost countless hours of sleep chasing problems that the CAD tools had warned them about.

The log has a number of other interesting tidbits of information. It tells you that 13 of the 97 Input/Output blocks (IOBs) are used, corresponding to your thirteen signals. The IOBs include inverters, so the only hardware required is the AND gate. This occupies one of the 7168 lookup tables (LUTs).

Under the Place & Route suboption of Implement, double-click the View/Edit Placed Design (FloorPlanner). This brings up a floorplan showing the IOBs around the periphery of the chip and the large matrix of LUTs in the core. Clicking on IOBs or LUTs in the panel on the left shows how they are connected. You may have to zoom in to see anything. You should see the 13 IOBs and one CLB in use. Click on the LUT (a multiplexer symbol) and determine the three IOBs attached. Click on each one and observe that they correspond to s[2], s[3], and led[6], as one would expect. Close the FloorPlanner and double-click View/Edit Routed Design (FPGA Editor). This shows the routing of the entire FPGA. Enjoy, then close the window. Another interesting report is the Pad Report under the Place & Route suboption. It tells you which signals are connected to which pins, which will be useful in the next lab as you wire pins to other circuits.

You have now been through the basic steps of entering a design, simulating it, and synthesizing and implementing it. More documentation for the Xilinx tools is available under the Help menu.

Now you must go on and finish the lab. Create another Verilog module named sevenseg that has four inputs s[3:0] and seven outputs seg[6:0] corresponding to the input switches and LEDs of the 7-segment display. Develop logic equations for each output and write the appropriate Verilog using assign statements. Do not use always blocks in this lab. You may use any logic gates you want for the design. There are many ways to

consider generating and optimizing your design. The Verilog book and references in your lab notes may be helpful.

It is good practice to carefully and legibly document your design process in your personal lab notebook at the same level of detail that you would use showing your work on a homework assignment. The act of planning and writing down your work often helps you catch errors that would have been time-consuming to catch on the computer for a nontrivial design. Of course, it helps you refresh your memory when you need to use your work later (for example, you will use your sevenseg module in the next three labs). And when you become a practicing engineer, your notebook carries legal weight to prove conception of inventions during patent litigation. This course is a good time to get into the habit of keeping a good notebook and finding what elements of the notebook are personally most valuable. However, your notebook will not be graded.

Simulate the circuit to see that it operates correctly. Synthesize and implement your design. Look at the RTL Viewer and Technology Viewer schematics. Do they match what you would expect? How do they differ? How many LUTs does your design use? Does the number match your expectations? (Hint: look at the Spartan FPGA datasheet for more information on CLBs and LUTs). Look at the floorplan. Can you relate the hardware appearing in the floorplan to that in the technology schematic?

What to turn in

For each lab in this class, turn in a concise lab writeup containing enough information for another engineer to duplicate your work and describing the key results of the lab. The report may be typed or handwritten legibly. All supporting printouts should be labeled appropriately to indicate their significance.

For this lab, your writeup should include:

- Your design process for the hexadecimal display driver;
- The Verilog for your hex display driver;
- Printed simulation waveforms of the hex display driver;
- Printed schematics of the synthesized hex display driver;
- A discussion of the synthesis and implementation results.
- How many hours did you spend on the lab? This will not count toward your grade.

Acknowledgments

An earlier version of this lab was developed by Fernando Mattos, Fall 1999.