Microprocessor-Based Systems (E155)

Harris Fall 2004

Lab 1: Hexadecimal Display Driver

Due: Week of Sept 13

Introduction

As the first lab for E155 - Microprocessors, 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 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 circuit 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 4.2i 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 chip burning. 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.

The user has several options to program the design into a chip. In our case, we will use the PROM Programmer of the Xilinx tools. The FPGA is configured by loading static RAM within the chip. The RAM must be reloaded on powerup. The FPGA can automatically load its RAM on powerup from an external nonvolatile memory such as an electrically erasable programmable read-only memory (EEPROM). With a PROM, EPROM, or EEPROM, we can use a "chip burner" to record a data stream. The nice thing about EEPROMs is that they are reprogrammable. You will be using the same EEPROM throughout the course.

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 that we will have during the semester, 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. 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 responsible for designing the 7-segment display decoder that reads the four-bit number specified by the 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:

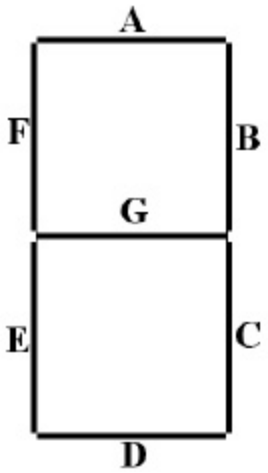

Figure 1 – Definition of outputs A through G for the 7-segment display

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 4.1i 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. Name it lab1_xx (where xx are your initials) and store it in your Kato directory. Set the device family, device, and flow to Spartan,

XCS10-3PC84, and Synplify Pro Verilog, respectively. This configures the Project Navigator to target the Xilinx 84-pin Spartan 10 FPGA that you will be using in the class and to use the 3rd party Synplify Pro tool to synthesize your Verilog code.

The Project Navigator window will come up with four panes. The Sources pane lists all the source files associated with your lab. To start with the design, create a new source file with the Project • New Source menu to invoke the New Source Wizard. Choose Verilog Module as the type and led as the file name, then choose next. Another window will appear to define the input/output (I/O) ports. Create 3 ports:

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

This defines a single-bit clock input, a 4-bit bus of switch inputs, and an 8-bit bus of led outputs. When you finish the wizard, the led.v file will appear selected in the sources pane. Below it, a set of processes are available for design entry, synthesis, implementation, and programming. At the bottom, the log pane should say "completed successfully." Watch the log window 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.

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));
```

To simulate your led module, be sure the led.v source is selected in the sources pane and click on the + symbol beside the Design Entry Utilities in the Processes window

to view the utilities. Double-click on the Launch ModelSim Simulator to invoke ModelSim, a 3rd-party simulator. Look through the log in the ModelSim window for any errors. One common error is the inability to open library aim; this may be ignored. If there are serious problems, the bottom of the window will 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. Once ModelSim launches correctly, a number of windows will appear; the most interesting are the main ModelSim command window and the wave window. In the command window, type the following commands:

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

Look at the waveform window and verify that the led waveforms match your expectations. Close the ModelSim command window when you are done.

Now you are ready to synthesize your led module. Being sure the led.v source is selected in the sources 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. You may also click on the + symbol to expand the Synthesize options and Compile suboptions, then double-click the View Compile Report item to view a report listing the synthesis errors.

Synplify Pro has a helpful capability of showing you how your Verilog compiles into gates and maps onto the FPGA. Under the Compile suboption, double-click Launch RTL Viewer. 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 Launch Technology Viewer from under the Mapping suboption. 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 an FPGA. Double-click Implement Design in the Processes pane and look at the messages in the log window. You should see warning #69 that an option overrides the effects of another option. To interpret this warning, right-click on the yellow web box next to the warning and choose Goto Solution Record. A web browser will pop up explaining that this is a rather dopey warning that should be ignored; remember that for future reference. 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 61 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 196 Configurable Logic Blocks (CLBs). Each CLB contains two lookup tables (LUTs); only one is needed for the AND gate. 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 14x14 matrix of CLBs in the core. Clicking on IOBs or CLBs shows how they are connected. You should see the 13 IOBs and one CLB in use. Click on the CLB and determine the three IOBs attached. Click on each one and observe that they correspond to s_pad[2], s_pad[3], and led_pad[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 attempt to wire pins to other circuits.

You have now been through the basic steps of entering a design, simulating it, and synthesizing and implementing. More documentation for the Xilinx tools is available by choosing Online Documentation from 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 switches and LEDs. Develop logic equations for each output and write the appropriate Verilog. 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. Simulate the circuit to see that it operates correctly and print a copy of your simulation waveforms. Paste or tape the Verilog and waveforms into your lab notebook and properly label your results. Do not staple a big pile onto one page; this is difficult to read and grade.

Look at the RTL Viewer and Technology Viewer schematics. Do they match what you would expect? How do they differ? How many CLBs does your hardware use? How many LUTs? 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

- Your lab manual documenting your design, including:
- Your design process for the hexadecimal display driver;
- The Verilog for your hex display driver;
- Simulation waveforms of the hex display driver;
- 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.