CSE 370 Introductory Laboratory Assignment

Applying Your Knowledge

Assigned: Friday, Novermber 14, 2008
Due: End of Next Lab Section

Objectives

You have gained many invaluable skills over the course of the quarter; you should now have a good understanding of combinatorial logic, sequential logic, registers, state diagrams, and other related skills. The final two labs of this course, lab eight and nine, will require you to combine all of your skills to develop a complete system on the FPGA. You will be provided with an output device, an LCD screen, and an input device, a magnetic stripe card reader, your task is to get the LCD to display the data stored on magnetic cards swiped through the magnetic card reader. This is not trivial to do, so it is highly suggested that you work with a partner. Furthermore, because of a shortage of LCD screens and magnetic card readers, you should probably work with a partner.

Since this task is relatively complicated it has been broken into two halves for you and your partner to work over the course of two lab sessions and time outside of class. The first half of the lab involves programming the logic that drives the LCD; like many devices the LCD has a series of initializations it must do, and a very specific set of data and commands it understands. For now, we will use the switches on the board to generate the data to be written to the LCD instead of reading data from the magnetic stripe reader. This will allow you to test to see if your LCD behaves as expected.

Before You Begin

The first thing you should before beginning this lab is finding yourself a partner.

At your disposal is also the very important datasheet for the LCD: LCD.pdf. This datasheet contains all of the information that is presented below in the lab, though in much greater detail. It is essential that you refer to the datasheet for questions about the LCD. You should take this opportunity in this lab to get some practice reading datasheets and interpreting the data they can provide.

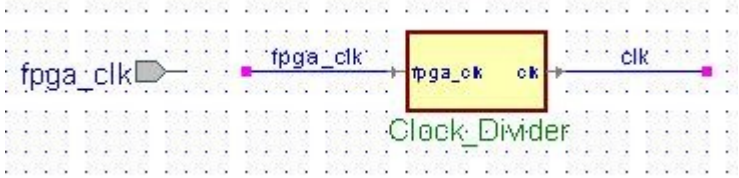
Tasks:

Since this is a complicated lab, this lab write-up will guide you through each component of the lab that you need to create for the final product to work. It is highly suggested you follow the tips and guidelines presented in the lab write-up. However, if you are feeling bold and confident, you are completely welcome to read the datasheet and design the entire lab independently of the write-up. Just make sure you complete all of the check off tasks at the bottom of the lab.

The Clock:

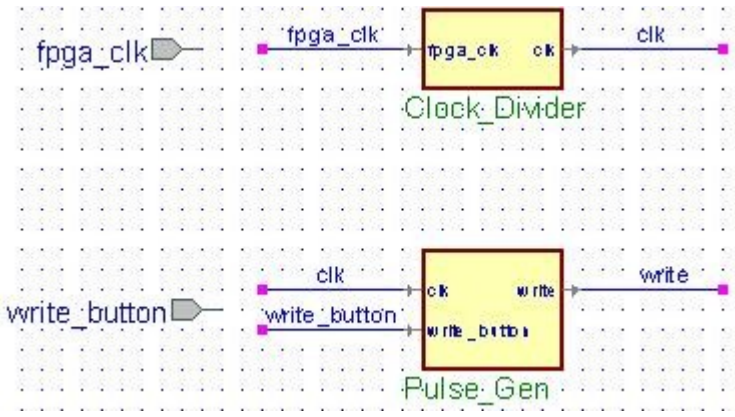
1. Now that we have done sequential logic we know how important the clock is for the entire system. So why are we concerned with the clock this time? Why can't we just use the clock provided by the FPGA? Take a look at the datasheet: LCD.pdf, under the section labeled as "Timing Characteristics" what do you see? The most important time is the Enable Pulse Width takes 450 nano-seconds minimum, however the FPGA's clock runs at 24Mhz. How fast is that? With that in mind, you should think about making a clock that has a clock cycle that is longer than the 450 nanosecond threshold so you don't have to worry about it anymore. Also note that some of these LCDs are old and require more than 450 nanoseconds to work properly and you don't need a very fast clock, so throw in some extra padding.

   If this is the problem, then how do we solve this problem? The most logical way would be to slow down the clock since we have no control over the internals of the LCD. Look at the provided clock_divider.v file, this verilog file divides down the clockspeed inputted into it and outputs a slower clock. Make sure you understand how it works because it is incomplete. Change the bits parameter and and the ratio parameter by filling in values for the x's such that it slows down the clock enough for the LCD to run properly. Have your TA check you off to confirm that you have an acceptable clock speed.
2. Now you should create a new design with design flow in Active-HDL. Add a new file called Lab_8.bde. Create an input called: fpga_clk and wire up your Clock Divider on the block diagram design. It should look like the image below when you are finished. Later on when we write up the .qsf file for the pin assignments we will assign the clock provided by the FPGA to the fpga_clk input and use the clk output that the Clock Divider provides as our actual clock.



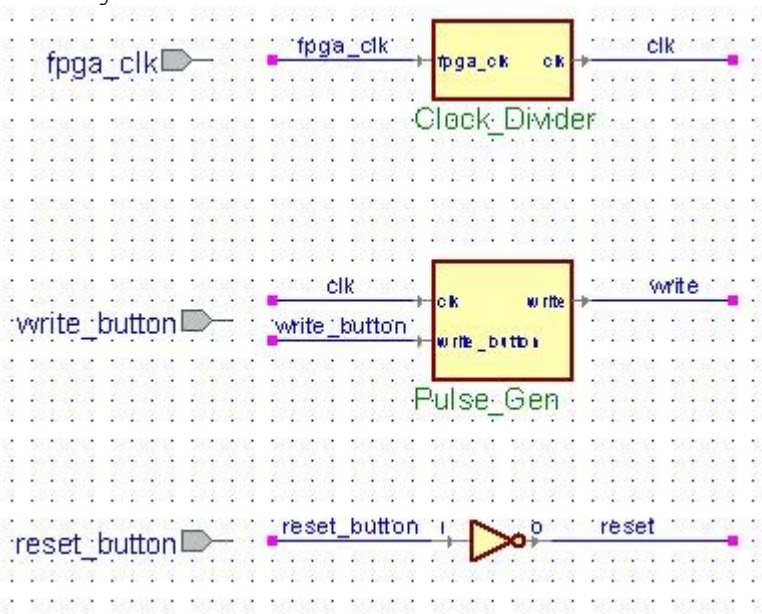
The Write Button:

3. For this lab you will need to control when characters are written to the LCD by using a button, but in order to write only a single character at a time it is helpful to bring back the Pulse_Gen module from the previous lab. This time we provide it: Pulse_Gen.v. Save it to your src/source folder and wire it up in your bde/block diagram design. Don't forget to add an in put in your design which represents the button. We will have to remember to assign a button to this input when we write the .qsf file later on. For simplicity, use the name: write_button for the name of the input. When you have wired it up correctly it should look like the image below.



The Reset Button:

4. We need to be able to reset the LCD to clear the screen whenever we want. The best way to do this is to create a button input that acts as a reset button. Create an input called: reset_button and wire it to an inverter with an output called reset. The inverter is used because the buttons are actually active low. When you press the button, instead of sending a logical 1, it sends a logical 0. Once you have wired it up, your diagram should look like the picture below. Well done, you've finished all the easy stuff!

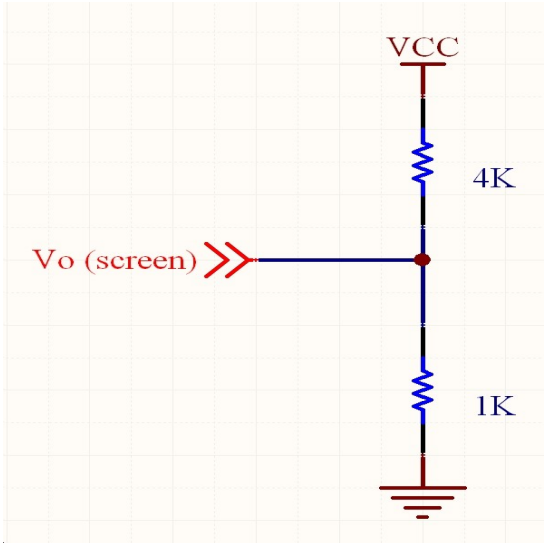


The LCD:

5. Now that we have finished the easy stuff it's time to actually create the logic that will drive the LCD and get it to do what we want it to do. This information is all contained within the datasheet: LCD.pdf. The LCD is capable of many commands; however you will only have to implement two commands: Reset and Clearing the LCD and writing a character to the LCD. To aid you, the pins and what they do are displayed below. The pins are numbered from right to left, when you hold the LCD so that the pins are closest to you.

Pin	Name	Function
1	GND	Ground - wire GND to it
2	VDD	5V - Powers the LCD, wire the 5V red VDD to it
3	Vo	1V - Contrast, The LCD needs exactly 1V on this pin so characters are visible
4	RS	High: When RS is high the LCD expects data to come over the data pins Low: When RS is low the LCD expects one of the hard-encoded commands to come over the data pins
5	R/W	High: When R/W is high the LCD can be read from Low: When R/W is low the LCD can be written to
6	E	This is the Enable, it executes command/data is passed to the LCD on the negative edge of E
7	DB0	Data Bus 0, this is one bit of data
8	DB1	Data Bus 1, this is one bit of data
9	DB2	Data Bus 2, this is one bit of data
10	DB3	Data Bus 3, this is one bit of data
11	DB4	Data Bus 4, this is one bit of data
12	DB5	Data Bus 5, this is one bit of data
13	DB6	Data Bus 6, this is one bit of data
14	DB7	Data Bus 7, this data bit also acts as an output for the LCD. When R/W is high and RS is low, the LCD will output a high signal if the LCD is busy and low signal if the LCD is not busy.

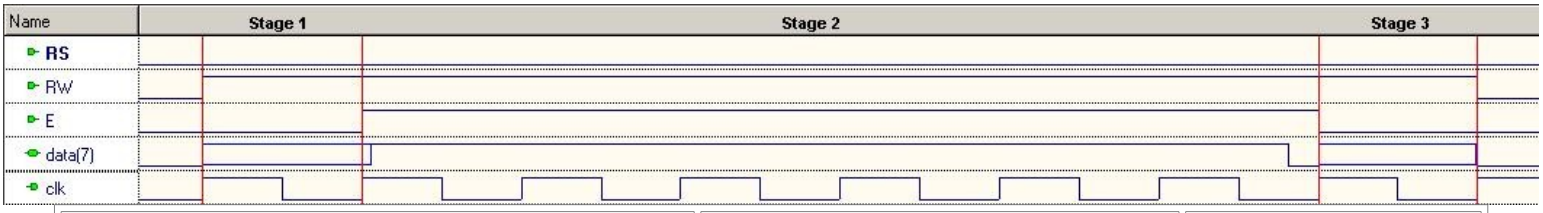
6. Make sure you understand each one of these pins and exactly what they do. You will have to provide a signal for all 14 of these pins. The first 3 are easy because they are only Ground, VDD, and Vo. To generate a 1V signal for Vo you will be provided with a 3.9K and 1K resistor. You can use these two resistors to divide the voltage down to provide 1V to pin 3 of the LCD. If you don't understand exactly how to do this, refer to the picture below. You can visit this site: Resistor Color Codings to help you identify which resistor is which. Go ahead and wire up the first 3 pins on your board now. Since the FPGA can arbitrarily assign inputs and outputs to any of the I/O inputs at the bottom of your board you don't have to worry about putting your LCD near any particular pins.



LCD Control Logic:

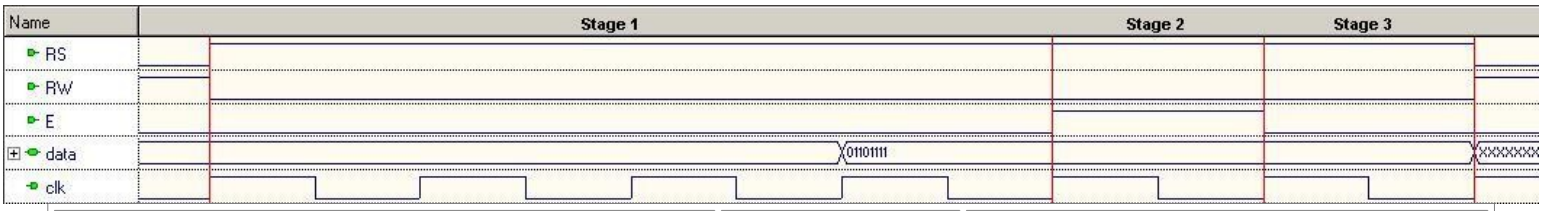
7. Now that we have set up power, ground, and contrast for the LCD we need to begin to work on the logic that drives the LCD. There are four key signals that you should keep in mind, RS, R/W, E, and Busy. From above you know that RS is driven on Pin 4, R/W is driven on Pin 5, E is driven on Pin 6, and Busy is driven on Pin 14. The goal of the control logic is to pass the correct signal for RS, R/W, and E based on the Busy signal. The idea here is: you can't write to the LCD if it's busy, and you can't pass commands to it either while it's busy. Note that according to the timing diagrams on the datasheet, RS and RW must remain fixed for a cycle before E goes high, while E is high, and a cycle after E goes low, so keep that in mind when you're considering your logic. Take a moment to consider what combinations of signals will be necessary to execute the following signals: Clear Display (Command), Function Set (Command), Display On (Command), Entry Mode Set (Command), Write Character (Data). Make sure you account for the kind of signal and set RS appropriately. Take a look at the waveform below, familiarize yourself with the timings of the LCD control logic.

Checking Busy Signal Waveform:



Stage 1: During this stage, because we need to read the busy signal, RS needs to be low, and R/W needs to be high. We need to wait one clock cycle before changing E to high to check for the busy signal. The reason why we need to wait is because the specification says there is an address set up time before changing E.	Stage 2: The address setup time has passed so you can now raise E and start checking DataBus[7] for the busy signal. When you finally see DataBus[7]/Busy go low, you can move to the next stage since you are done checking the busy signal.	Stage 3: This stage is necessary in order to meet the timing requirments. E gets set low which ends the busy signal test

Command/Character Write Waveform:



Stage 1: Since you are preparing for a write operation R/W has to be set low. The value of RS depends on whether a character or a command is being written to the LCD. If it is a character that is being written then you must pause here until you have a valid charachter to write	Stage 2: At this stage you raise E to write the character or command to the LCD	Stage 3: One clock stage later the state machine can lower E, creating a negative edge. RS and R/W needs to be held for one clock cycle after E is lowered according to the specifications of the LCD.

LCD Setup Commands:

8. The commands: Clear Display, Display On, Entry Mode Set, and Function Set are the initialization commands that must be passed to the LCD to prepare the LCD to write characters. Since these are the initialization commands, they must be executed every single time the LCD is reset. You should refer to the table below for the execution order, RS signal, and command encoding for each of these commands.

Order	Operation	RS	DB[7:0]
1	Clear Display	0	0000 0001
2	Function Set	0	0011 0011
3	Display On	0	0000 1100
4	Entry Mode Set	0	0000 0110
5+	Write Character	1	???? ????

Before you write any character or command to the LCD you must check the busy signal first. In fact your circiut should always be alternating between checking the busy signal and writing something to the LCD. A state machine is a very good way to control what stage of the above waveforms you are in. You should also think about making a state machine to keep track of what command you are on or whether you've finished sending the commands and are now sending characters. Note that for every command/character the second state machine goes through, the first state machine must complete a full cycle. This means that the first state machine must have an output that is used as an input to the second state machine. Also note that because there is not a steady stream of characters to write there will be times when you will have to wait in stage 1 of the write waveform. Whether or not to pause is something the second state machine must tell the first. There is also the question of how to deal with RS because its value depends on the state of both machines. Based on the information that you have now, you should be able to design two simple state machines/state diagrams that carry out all of the logic described above. Show your TA your completed state diagrams before moving on.

Programming States in Verilog:

9. Now that you have created two state diagrams it is time to convert this state diagram into Verilog. The first thing to note is that all of your logic will probably be sequential, so you will want to do most of your coding inside of an always block. Within always blocks you can set up case statements, which allow you to easily transition from state to state. Most of your logic will involve transitioning from one state to another to achieve the desired effect. You can refer to the code below on the usage of case statements in Verilog.

```
module case_state_example (CLK, Next, Reset, Fib_Out);
  input CLK, Next, Reset;
  output reg [4:0] Fib_Out;
  reg [2:0] state;

  intial
    state=0;

  always @(posedge CLK) begin
    if (Reset)
      state = 0;
    else case (state)
      0: if (Next) state = 1;
      1: if (Next) state = 2;
      2: if (Next) state = 3;
      3: if (Next) state = 4;
      4: if (Next) state = 5;
      5: if (Next) state = 6;
      6: if (Next) state = 7;
      7: if (Next) state = 0;
    endcase
  end

  always @(state) begin
    case (state)
      0: Fib_Out = 1;
      1: Fib_Out = 1;
      2: Fib_Out = 2;
      3: Fib_Out = 3;
      4: Fib_Out = 5;
      5: Fib_Out = 8;
      6: Fib_Out = 13;
      7: Fib_Out = 21;
    endcase
  end
endmodule
```

If you look over the code, it is actually very redundant, the output given is just the next Fibonacci number in the sequence, which resets or cycles back to the first Fibonacci number. The technique is relatively simple, you only need a register to remember which state you are in, and then a case statement that determines the logic to advance states. You can include an intial block as well to state what value the registers begin with. Another always block can be used to change the output based on the changing state, as the second always block demonstrates in the code. You can use this system to write the logic for your LCD.

Tri-State Command and Data:

10. By now you must have realized that the 8-Pin DataBus is shared by two different sets of signals.

One is the command signals to initialize the LCD and the other is the data signals that specify which character to write to the LCD. We need to tri-state these two sets of signals so that they don't interfere with one another. You can do this in Verilog by specifying the output z, z stands for high impedance, which basically means the connection is temporarily disconnected from the rest of the circuit. You can refer to this article for some more information on Tri-State Logic . You will need to do this when writing the Verilog for your LCD logic. An example is provided for you below on how to tri-state your command signals and data signals.

```
module tri_state_example (Select, Data, DataBus);
    input Select;
    input [7:0] Data;
    output [7:0] DataBus;
    assign DataBus = (Select) ? Data : 8'zzzzzzzz;
endmodule
```

As you can see from the code above, the Select input acts just like a multiplexer, choosing from the Data that comes in or the high impedance z signal. The LCD is already designed to tri-state its data bus outputs when RW is low so that you can drive the bus. You have to make sure to tri-state your data bus outputs when RW is high so that the LCD can drive the bus.

In-Out Buses/Wires

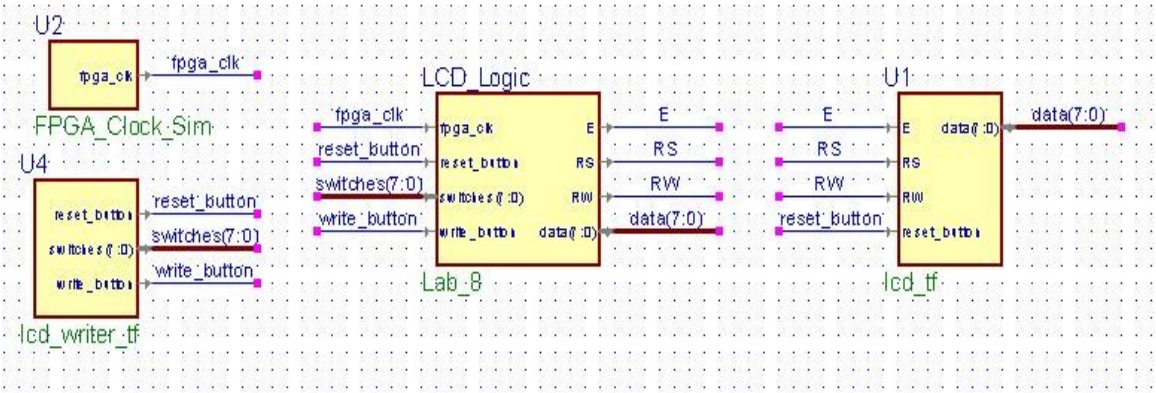
11. If you have been paying attention you will notice that in your block diagram design you need to make your DataBus an output too because the busy signal comes over on DataBus[7]. If you check under the icon that usually has inputs and outputs, however there is another option to make your input/output an in-out instead. This will allow your code to work when you program it. Just in case you aren't sure, check the picture below to make sure it looks right.

Testing Your Logic:

12. Now that you have worked out your logic your LCD logic should consist of 4 inputs and 4 outputs. The four inputs should consist of the clock, reset, write, and an 8-bit switch input. Your four outputs should consist of E, RS, R/W, and an 8-bit data output. Wire up the following three pieces in a new test.bde around your LCD logic as shown in the picture below. You will need these three pieces:

fpga_clock_sim.v
lcd_writer_tf.v
lcd_tf.v

Make sure that the expected message prints out: "Hooray!



Synthesize and Demonstrate:

13. Now that everything has been tested and seems to work in simulation you are ready to go ahead and synthesize it and wire it up on your prototyping board. Use SW0-7 to drive your Switch inputs, use the 24MHz clock to drive your clock input that is divided down. Drive your write with KEY1 and drive your reset with KEY0. You should run your outputs to the output pins which you used in the earlier labs. When writing your .qsf file for this lab, you will have to make connections from the LCD pins to the IO pins on your prototyping board. The name of the pins are written on the label on the side of the IO pins. These are like "A13, B13, etc." You can refer to this page: IO Pins, if you are confused. Make sure everything is working and demonstrate your working LCD to your TA.

Lab Demonstration/Turn-In Requirements

A TA needs to "Check You Off" for each of the tasks listed below.

1. Have a TA check you off for your Clock Divider.
2. Have a TA check you off for your LCD Logic State Diagrams.
3. Have a TA check off your working LCD.

Comments to: cse370- webmaster@cs.washington.edu