EE 209 Lab 3 – Mind over Matter

1 Introduction

In this lab you will use the Xilinx CAD tools to complete the design of a kids game commonly referred to as “Mastermind” (simplified for easy implementation). In particular, you will design muxes, registers with enables, incrementers, counters, and a 7-segment decoder using a sum of minterms (decoder-based design) approach.

Note: In this lab we have provided the high level design, identifying the components and their connections needed to implement the game. Your job will be to build the individual components. In future labs you’ll also need to come up with some of the higher level design and we’ll assume you can then build the needed components on your own.

2 What you will learn

This lab is intended to show you the process of using an FPGA (Field-Programmable Gate Array) chip to implement digital HW designs while allowing you to practice your ability to implement an arbitrary logic function.

3 Background Information and Notes

The Mastermind Game: The mastermind game allows an "opponent" to input a secret combination (in this case, binary combination via switches) and then let the other player input guesses for the combination providing some kind of feedback (in this case a count of how many switches are in the right configuration). The normal game usually hides the secret combination and lets the player start guess from scratch with only a maximum number of guesses. Here we allow any number of guesses and you and a friend will have to take turns setting a secret combination while the other player doesn't look and then flipping the switches to some random combination to let the other player start guessing. The general flow of the game is as follows:

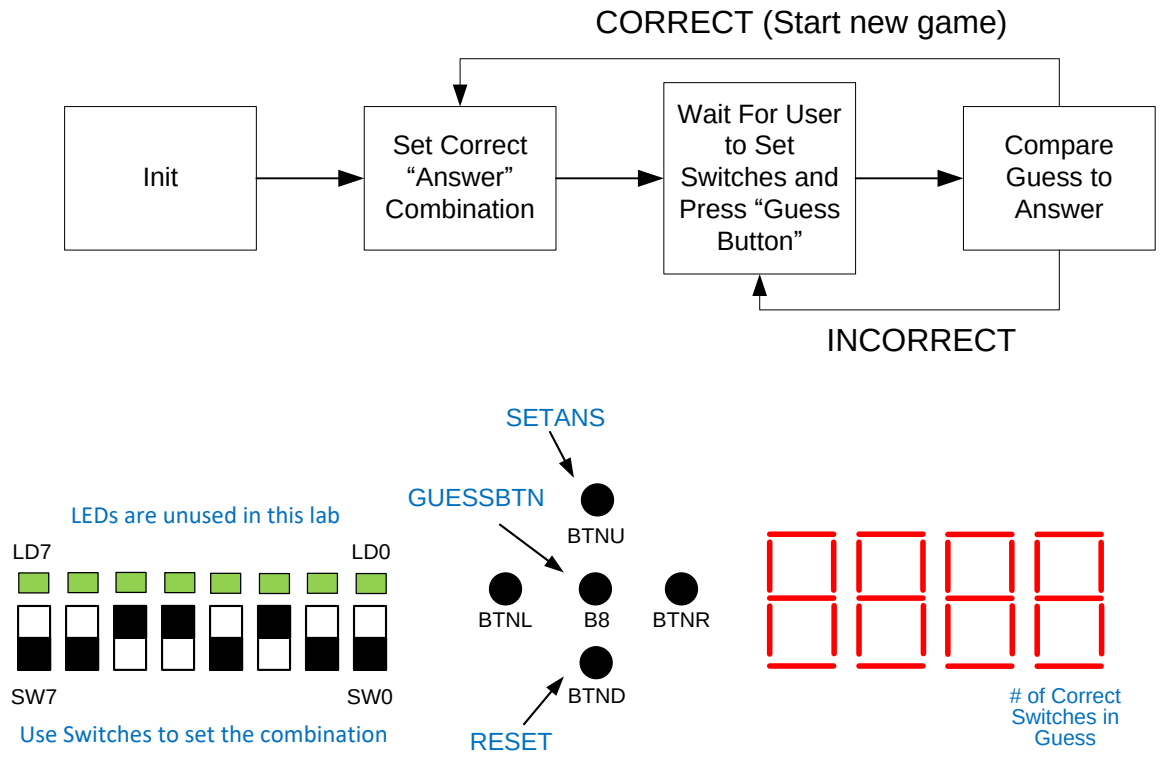


Figure 1 - Switches can be used to set both the secret combination and the guesses. Buttons can be used to capture the secret combination or indicate a guess is ready to be scored. The number of correct switch positions will be displayed on the 7-segment display.

Gameplay:

1. Player 1 should configure the 8 switches and press the SETANS button and then rearrange the switches in some random order before Player 2 starts to guess.
2. Player 2 may now configure the switches however he/she likes and then press the GUESS button. This will trigger the hardware to compare the switch configuration to the "answer" (1-bit at a time) and display how many switches are in the right place (though not which ones)
3. Player 2 may repeat step 2 as much as they like or if they want to start a new game simply start at step 1 (i.e. configure the switches and press SETANS).

FPGAs: Field-programmable Gate Arrays are chips that have many hardware resources (e.g. gates, registers, etc.) built on it but can be configured for how to wire them together. The idea is this one chip can be configured to wire the resources together to implement one design and then be reconfigured to wire the resources together differently to implement another design. This is handy for our lab since we don't want to buy separate chips for each lab, but instead reuse the chip. The designs you draw and describe in the Xilinx tools will be converted (i.e. synthesized) to produce the appropriate configuration to implement that exact

design on the FPGA. Your job is just to produce the write design description (i.e. schematic in this case).

Overall Design: A schematic of the overall HW design is shown below and only consists of a handful of components (primarily registers, muxes, and counter circuits). If you took EE 109 we want to draw a comparison and have you realize you could easily implement this in your Arduino but it would run significantly slower than this HW version can execute (though for a human user interacting at the seconds/milliseconds level that difference is undetectable). But we simply want to point out that what can be done in SW can be done in HW and vice versa.

However, let's take a look at the hardware design. We don't expect you to fully understand the operation and all the components yet but within a few weeks you should be able to and this may serve as a nice example/case study to refer back to. We start at the top with the ANSWER[7:0] register. When you press the SETANS button, this will enable the 'answer' register and capture whatever is on the SWITCHES and save it (recall registers save values each time you clock it). We then wait until the Guess button is pressed which starts a counter (producing CNT[2:0]) generating binary numbers 000-111 (one per clock). This 3-bit sequence (much like a for loop counter, i, and an array) walks through each current switch, SWITCHES[i], (on the bottom mux) and the corresponding ANSWER[i] bit and compares them to see if they are the same and, if so, increments a counter of how many correct positions the player has. This CORRECT count is then displayed on the 7-segment display. We need the 8-to-1 muxes to select individual bits to allow for sequential comparison and counting since comparing all 8-bits and counting them in parallel would require additional HW that we'd like to avoid for now.

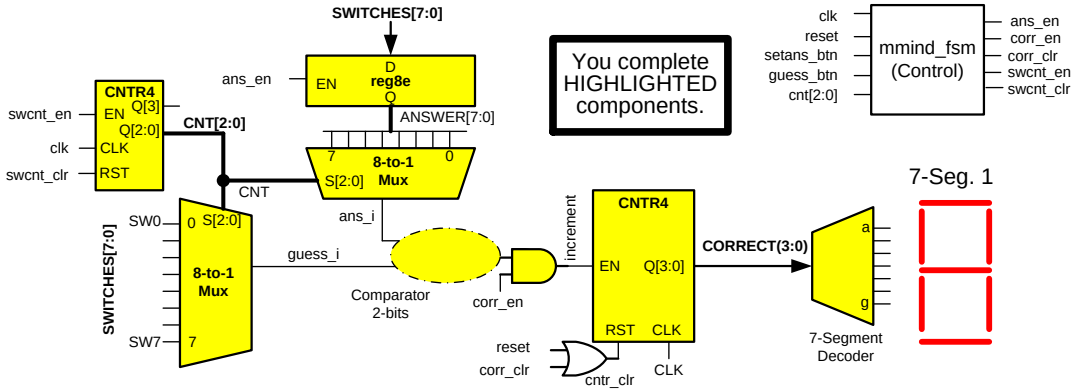
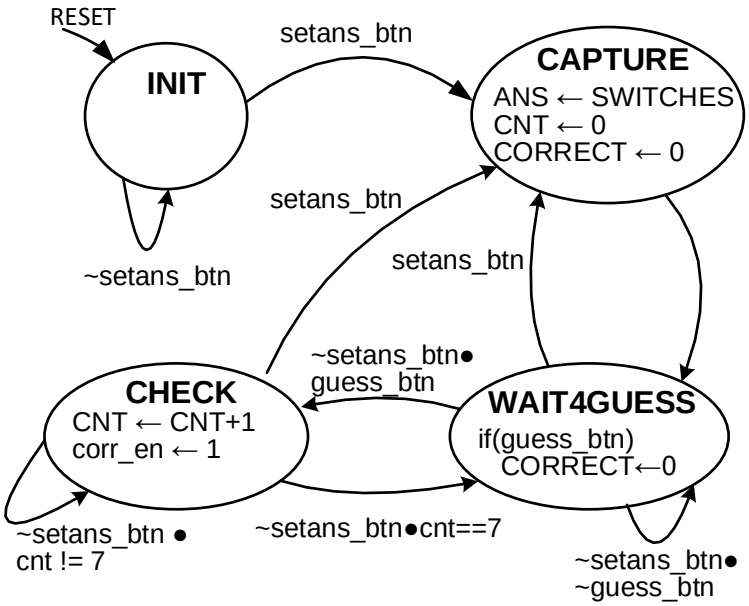


Figure 1 – Datapath of the MasterMind Game

IMPORTANT: Keep coming back to this picture and map each component here to the components in **mmind.v**

Governing the various control signals to clock a register, increment a counter, reset a counter to 0, etc. is a state machine (which you've also learned about in EE 109) but this state machine is implemented in HW. You will learn how to implement state machines in HW in a few weeks. Below is the state diagram…See if you can understand its basic operation.



Your task: Your job will be to build all the individual components. In doing this you should review how each component is built. But another learning goal is to understand how to use hierarchy (take 1 small component and combine them with others to build large components). This includes:

- An 8-to-1 mux that can be used for the two muxes shown in Figure 1
- Determine **what gate** can be used in the blank oval that is shown comparing the bits from the two muxes to see if they are equal (i.e. the comparator of 2-bits) and the **AND gate** producing the 'increment' signal
- The 7-segment decoder that will convert a 4-bit binary number whose value is (0-8) to the appropriate segments to light up the 7-segment display.
- A 4-bit incrementer used in the 4-bit counter blocks shown in Figure 1.
- Complete the design of reg4e (4-bit register with enable) that will be used in the 8-bit register and the two counters shown in Figure 1.
- Use hierarchy to create the 8-bit register with enable that is used to store ANSWER[7:0] from two 4-bit registers with enable.

Important Notes: As you design the 7-segment decoder, it is important to note that the actual display is connected in such a way that an output of 0 will light up the segment and a 1 will turn it off. We refer to this as an active-low convention because the light will turn on or activate when we output a 0 (or low value). The easiest way to deal with this is to just design the circuits normally (producing

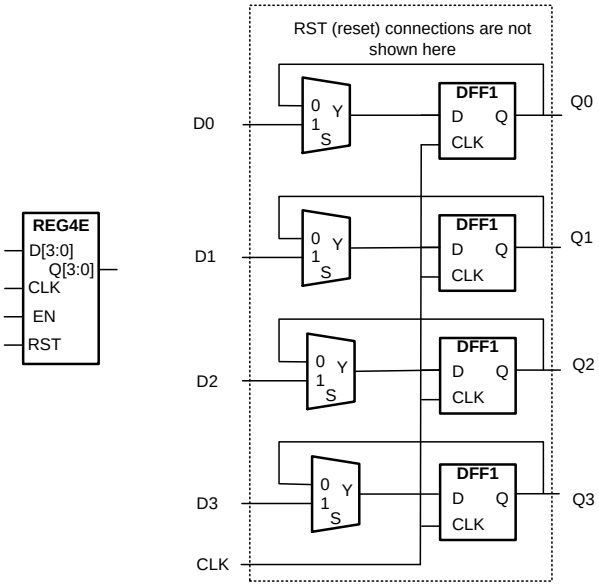
outputs of 1 when you want the LED to light up) and then use an inverter right at the output to flip the 1 you want to light up the segment to the actual 0 that is needed by the display.

Building the components

8-to-1 mux (mux8.v): This should be straightforward. Do not use hierarchy here (i.e. do not cascade many 2-to-1 muxes, etc.). Instead, design the 8-to-1 mux directly from AND, OR, NOT operations.

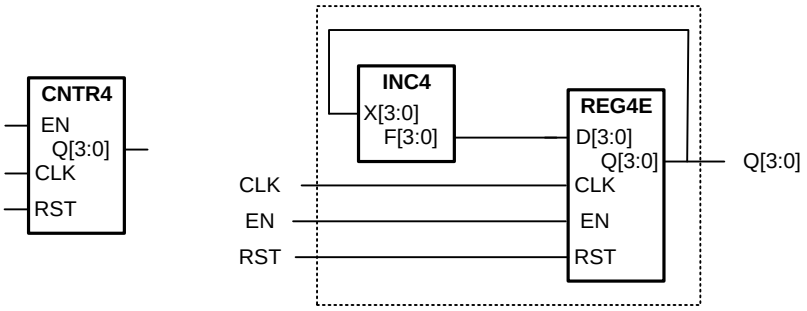
7-segment Decoder (sevenseg_decoder.v): Given a 4-bit input design logic for the 7 outputs (a-g). Use the attached worksheet at the end to help. Use a sum-of-minterm (or product-of-maxterm) approach. Think about how you can just generate all the minterms (or maxterms) and share them to produce the 7 outputs (i.e. you should not have to reproduce all the minterms for EACH of the 7 outputs).

4-bit Register with Enable (reg4e.v): A register with enable is simply D flip-flops with a 2-to-1 mux produce the D-FF's D input. The mux allows us to choose whether we recycle the old Q value and retain it on the next clock cycle OR if we should take in and remember a new value (the overall D-inputs). The schematic is shown below. Use the provided 2-to-1 mux in **mux.v** and the provided D-FF's to complete the design of the 4-bit register with enable.

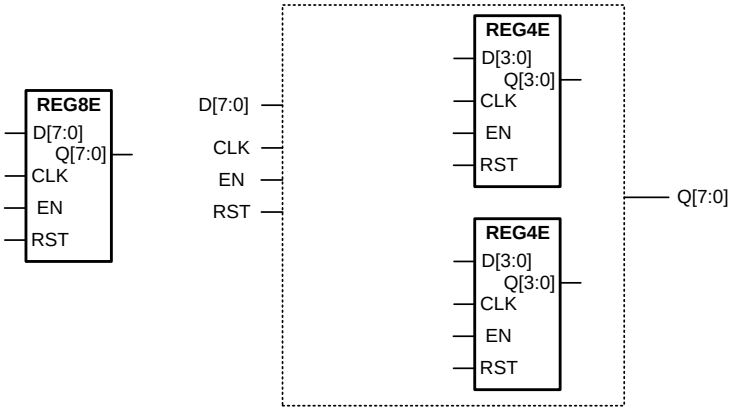


4-bit Incrementer (inc4.v): A 4-bit counter is a circuit that will count up in binary one combination per clock cycle. We can build a counter as an incrementer and a 4-bit register (with enable) that you designed above. Write out a truth table for a 4-bit incrementer where each output is 1 more than the input combination. Note that the input combination 1111 should cause an output of 0000 (i.e. the output wraps to 0). Use an approach similar to the 7-segment decoder by generating minterms and then producing each output as a function of those minterms.

4-bit Counter (cnt4.v): Design the 4-bit counter by simply instantiating and wiring a 4-bit incrementer and 4-bit register with enable as shown in the schematic below.



8-bit Register w/ Enable: Design an 8-bit register with enable by running two 4-bit registers together in parallel. Use the incomplete schematic below to think about what connections are needed.



4 Prelab

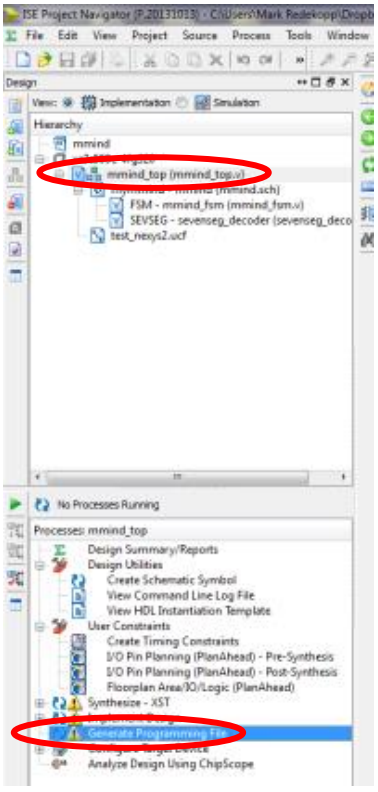
None.

5 Procedure

1. Download the project skeleton zip file from our website and extract it to a folder. Then load the project file (the file with the .xise extension) in Xilinx's Project Navigator
2. Open the **mmind.v** Verilog file (not the mmind_top.v). You will see a structural description of the components in our design. You must now implement the missing designs.
3. Start with the 8-to-1 mux. Open **mux8.v** and complete the logic by instantiating AND, OR, and NOT gates or using assign statements. Do NOT use 2-to-1 muxes to build the larger 8-to-1 mux. Practice how to build a mux "from scratch".
4. In the **mmind.v** file (this is the top-level design file where all components are instantiated), scroll down to find the blank area where you should add the gate(s) that are needed to compare the current guess and answer bits and then the AND gate to produce the **increment** signal. Be sure you use the appropriate signal names so that the design will correctly wire the gates and counter

together. Your inputs should be **ans_i** and **guess_i**. The final output of the AND gate should be **increment**.

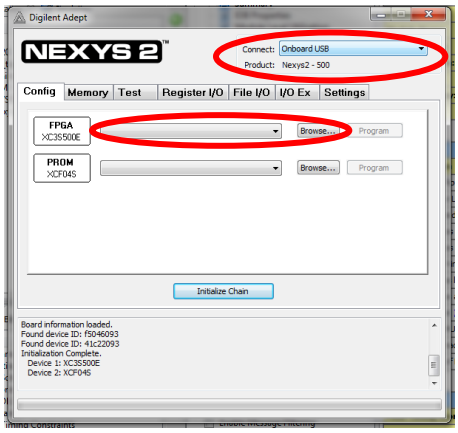
5. Now complete the 7-segment decoder design by opening the **sevenseg_decoder.v** file.
    a. Complete the truth table for the outputs a-g in terms of the 4-bit input on the **attached sheet** which you will submit with your lab report.
    b. In the Verilog file, implement minterms for inputs 0-9. Then, for each output a-g, sum together the appropriate minterms (use your truth table to guide you) and produce that output by **NOR**'ing the appropriate minterms together (**we normally OR minterms but remember the display wants 0's to light up the segment so we will just invert at the output…**)
6. Complete the implementation of **reg4e.v** to implement the 4-bit register with enable. (You do not need to have studied registers with enable to implement one. It is just wiring some 2-to-1 muxes and D flip flops together. See the earlier schematic).
7. Complete the implementation of **inc4.v** to implement a 4-bit incrementer.
8. Complete the implementation of **cntr4.v** to implement a 4-bit counter. (You do not need to have studied counters to implement one. It is just wiring a register and incrementer together. See the earlier schematic).
9. Finally, implement the **reg8e.v** design. Do so by wiring up two 4-bit registers w/ enable.
10. Simulate your design using the provided testbench by clicking the Simulation radio box in the upper right of the window, then choosing **mmind_tb.v** in the Hierarchy area, and finally Simulate Behavioral Model in the Processes area. We have included a wave configuration file that will display the most important signals for you though you are welcome to add more.



[**Important**] The best way to make sure you understand the design is to painstakingly trace through the simulation waveform and see what happens. Think about what the input stimulus is: the switches, the setans button, the guess button, etc. Look for those to get an idea of what scenario we are inputting. Then from there look at our circuit diagram and think about what the important signals should do and when. Confirm them on the simulation waveform. It takes time to really understand what is happening but you will be well rewarded with what you learn from it and it will grow your ability to debug digital circuits, so please invest that time.

If you find the simulation doesn't behave as intended, add more intermediate signals, rerun your simulation, etc. until you can figure out the problem.

11. Once you are satisfied the design seems to work in simulation we will now implement it on the FPGA. To do so, go to the Design/Hierarchy tab in the top right and select the top-level file **mmind_top.v**.
12. **Important**: In the Processes pane, right-click "Generate Programming File", click "Properties" and under "Startup Options" ensure that the "FPGA Start-Up Clock" is set to "JTAG Clock". This is necessary for your design to work properly but only needs to be done once (the project settings will be saved).
13. Now double click the **Generate Programming File**. It will take some time to synthesize the design and implement it but when it is done double check that there are no errors (look at the errors tab in the bottom console area) or at the color of the icon next to the Synthesize and Implement processes. Warnings (yellow triangles) are fine however.
14. At this point the hardware configuration file (.bit) file has been generated and is ready to configure the hardware on the FPGA boards. Get a Nexys-2 board from your TA and connect it via USB to your laptop. If you are running on the Remote Desktop (VDI) you'll need have the virtual machine "take control" of the Nexys board by selecting "Connect USB Device" and then choosing "Digilent Onboard USB".
15. If you are running on your own PC you'll need to download the Digilent Adept software using the link on the Tools page of our course website.
16. Start the Digilent Adept software which is used to download the HW configuration. Make sure the Connect field in the upper right says Onboard USB and the Product is recognized as Nexys3. Finally, in the FPGA row, select the Browse button and find the **mmind_top.bit** file in your project folder. Then click Program which should configure your hardware and start the program running.



17. Try playing the game on the board and ensure the display is showing appropriate digits. If you got your logic wrong go back and examine it and try to fix it. You will then need to repeat steps 13-16.
18. Demonstrate your working game to a TA and get their signatures/initials. Submit your files online.

6 Lab Report

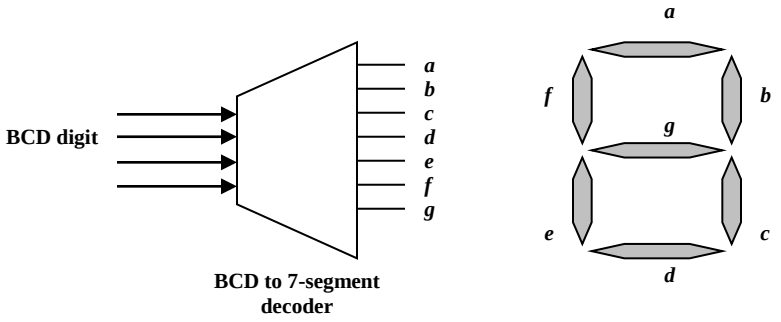
Name: _____ Score: _____

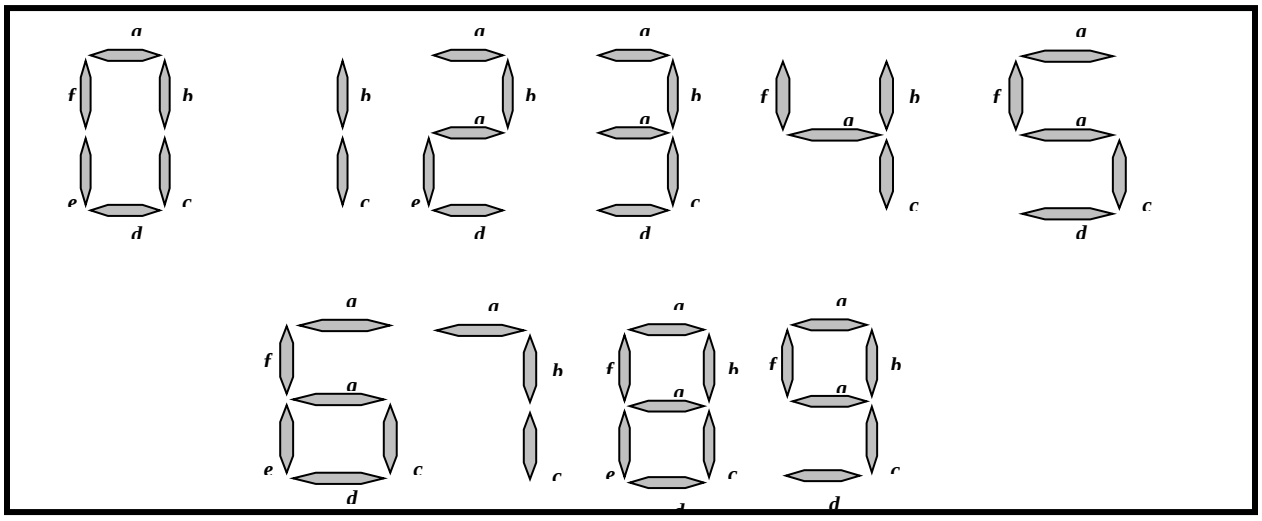
Due: _____

(Detach and turn this sheet along with any other requested work or printouts)

Turn in the following items:

1. TA Signature: _____
2. Submit your mmind.v, inc4.v, reg4e.v, cntr4.v, reg8e.v, mux8.v, and sevenseg_decoder.v at the online submission link on the lab webpage.
3. Completed truth table below.





COR[3]	COR[2]	COR[1]	COR[0]	a	b	c	d	e	f	g
0	0	0	0							
0	0	0	1							
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							

7 EE 209 Lab 3 Grading Rubric

Student Name: _____

Item	Outcome	Score	Max.
Design • Correct Truth Tables for a-g	Yes / No		1
• Correct mux8.v implementation	Yes / No		2
• Correct reg4e.v implementation	Yes / No		2
• Correct inc4.v implementation	Yes / No		2
• Correct cntr4.v implementation	Yes / No		2
• Correct reg8e.v implementation	Yes / No		2
• Correct seven segment decoder implementation	Yes / No		2
• Correct logic for comparing ans_i and guess_i	Yes / No		1
• Circuit works correctly on the FPGA	Yes / No		1
SubTotal			15
Late Deductions (-1 pts. per day)			
Total			15
Open Ended Comments:			