CS 223 Digital Design Laboratory Assignment 5

Traffic Light System and BCD Counter

Preliminary Report Due: April 29, 2024 13:00

Lab Dates and Times

Section 1: May 02, 2024 Thur. 13:30-17:20 in EA-Z04
Section 2: April 29, 2024 Mon. 13:30-17:20 in EA-Z04

Location: EA-Z04 (in the EA building, straight ahead past the elevators)
Groups: Each student will do the lab individually. Group size = 1

Preliminary Work [30]

Note: This part must be completed before coming to the lab. Prepare a neatly organized report of your work and submit on Moodle before the start of the lab. Include your code as **plain text, not image!**

Traffic Light System

In a community, people need stop signs and traffic lights to organize the traffic flow. If there were no traffic lights or stop signs, people's lives would be in danger from divers going too fast. These devices also play a role in road safety. While accidents still occur at intersections, these crashes may be been prevented by the drivers yielding to the traffic lights and improving the traffic lights timings. To reduce danger at the intersections, time for switching red light to green light and green light to red light should be regulated carefully. Studies show if the both lights are red for 3 seconds before either light turns green again prevent huge amount of accidents in traffic.

Traffic light system you are going to implement is similar to the example in pages 124-129 in the text book, given in Fig. 1 for your convenience. The roads in which there is an intersection are Road A and Road B. There are sensors S_A and S_B installed in each road to detect the traffic. Each sensor will be `TRUE` if traffic is present and `FALSE` if the road is empty. There are two traffic lights L_A and L_B to control the traffic:

- The lights may change every 3 seconds depending on the sensors.
- If a sensor output is `TRUE` the lights will not change until it is set to `FALSE`.
- If a light is green and sensor is `FALSE` it will turn to yellow and then red. Both lights will be red for 3 seconds and then red light will turn yellow 3 seconds and then turn green.

You are going to design a state machine to model the traffic light controller of this intersection.

- **[6]** Sketch your Moore machine state transition diagram, state encodings, state transition table, output table, next state and output equations and your Finite State Machine schematic.
- **[2]** How many flip-flops you need to implement this problem?
- **[2]** Redesign your outputs using decoders.
- **[5]** Write a SystemVerilog module for your state machine and prepare a testbench for it to verify functionality.

Binary-Coded-Decimal Counter

In the previous lab, you utilized binary-coded-decimal (BCD) representation to drive the seven-segment display. This lab will build on that previous work. Therefore, you can re-purpose some of your former work. The introduction we made for the BCD concept is provided for you below:

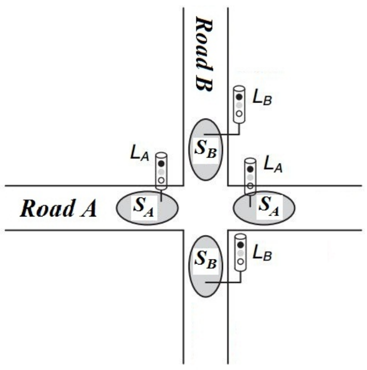

Figure 1: Model of the intersection.

Binary-coded decimal (BCD) is a form of representing numbers that is suitable for driving digit-oriented displays. The concept can be best understood through an example: Consider the decimal number "23". You need at least five bits to store it, but assume you store your numbers in groups of eight bits. Eight-bit representation of this number is "0001_0111". However, when you need to drive a digit-oriented display like the seven-segment display on the Basys3 board, you need to calculate each digit of this number, "2" and "3" whose binary representations are "0010" and "0011" respectively. These two binary numbers look nothing like the original two-digit decimal number at the first sight.

Mathematically, you need to perform $\mod 10$ *and* $\div 10$ *operation to get the first and second digits respectively. The division operation is not trivial to realize in FPGA logic unless you are using DSP slices. In this case, it might be preferable to store the number the way you are going to use it: in binary-coded decimal form. If you store the number as "0010_0011" where the* MSB *four-bits are "2", and the* LSB *four-bits are "3", you can directly drive the seven-segment display from these bits.*

Building on this concept, you will design a counter that will increment every second. You can easily implement a behavioral counter in Verilog by representing the total count in a binary vector and triggering the counter with a one-second period clock signal. However, you would need to implement the logic described above to actually be able to display the current count on the digit-oriented seven-segment display.

Instead, you are going to implement a BCD counter as we are only going to use this circuit to count seconds. Your circuit will store the counted seconds directly in BCD format and handle the necessary transitions without doing division and modulo arithmetic. Fig. 2 shows the i/o of a single-digit synchronous up-counter. The operation of your BCD counter should be as follows:

- When `Enable` $= 1$ and `Load` $= 0$, the count $Q_{[3:0]}$ is incremented on the rising edge of the clock.
- When `Enable` $= 1$ and `Load` $= 0$ and $Q_{[3:0]} = 4'b1001$, the count $Q_{[3:0]}$ rolls back to $4'b0000$ on the rising edge of the clock.
- When `Enable` $= 1$ and `Load` $= 1$, the input $D_{[3:0]}$ is loaded to the output $Q_{[3:0]} \leftarrow D_{[3:0]}$ on the rising edge of the clock.
- When `Enable` $= 0$, the output $Q_{[3:0]}$ remains the same $Q_{[3:0]} \leftarrow Q_{[3:0]}$ on the rising edge of the clock.

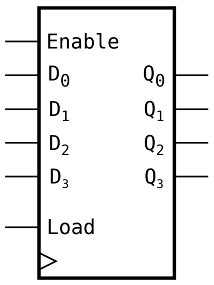

Figure 2: Binary counter with parallel load.

According to the described operation:

[5] Write a SystemVerilog module for a single-digit up-counter with enable signal and parallel load. Prepare a testbench for it to verify functionality.

[5] Write a SystemVerilog module for a single-digit BCD counter with enable signal that resets after reaching the number nine using the up-counter you designed in the previous step. Prepare a testbench for it to verify functionality.

[5] Write a SystemVerilog module for a two-digit BCD counter using the single-digit BCD counter you implemented in the previous step. The count should automatically roll back to 0 after 99. Prepare a testbench for it to verify functionality.

Hint: You can implement the reset logic and extend to two-digit case by adding a few gates to the binary counter.

Part 1: Traffic Light System on FPGA [35]

In this part, you are going to implement your traffic light system on the FPGA and have a demo.

1. Slow down the clock to at 3 seconds to see the change in the lights.
2. Use LEDs on BASYS board for outputs of LA and LB traffic lights.

   Red : *** (three leds) Green: ** (two leds) Yellow: * (one led)
3. The SA and SB sensors will be two left most buttons. The sensor will be active when as long as the button set to 1.

   Now test your code and show the result to your TA.

Part 2: BCD Counter on FPGA [35]

In this part, you are going to implement your BCD counter on the FPGA and demonstrate your counter on the seven-segment display using the driver you previously created.

Note: You can directly implement the full four-digit BCD counter and get full points from this section. The single-digit binary and BCD, two-digit BCD counter implementations are for incremental work and for you to get partial points.

Slow down the clock to at 1 second so that the counter increments every second.

[5] **Test:** Synthesize, implement, generate bitstream file, and download your **binary counter** to Basys3 FPGA board. Assign four consecutive switches as your input and follow the convention where the leftmost bit is the MSB and rightmost bit is the LSB. Use any one of the buttons so that you can trigger a reset/load. You can choose any one of the four of the digits on the display.

[5] **Test:** Synthesize, implement, generate bitstream file, and download your **BCD counter** to Basys3 FPGA board. Assign four consecutive switches as your input and follow the convention where the leftmost bit is the MSB and rightmost bit is the LSB. Use any one of the buttons so that you can trigger a reset/load. You can choose any one of the four digits on the display. This counter resets to 0 after counting up to 9.

[10] **Test:** Synthesize, implement, generate bitstream file, and download your **two-digit BCD counter** to Basys3 FPGA board. Assign eight consecutive switches in groups of four as your inputs and follow the convention where the leftmost bit is the MSB and rightmost bit is the LSB. Use any one of the buttons so that you can trigger a reset/load. You can choose any two of the four digits on the display. This counter resets to 0 after counting up to 99.

Now, extend your circuit to create a four-digit BCD counter. The design flow should be very similar to the one where you extended to two-digit BCD counter.

[15] **Test:** Synthesize, implement, generate bitstream file, and download your **four-digit BCD counter** to Basys3 FPGA board. Assign all sixteen switches in groups of four as your inputs and follow the convention where the leftmost bit is the MSB and rightmost bit is the LSB. Use any one of the buttons so that you can trigger a reset/load. Use all four digits on the display. This counter resets to 0 after counting up to 9999.

Clean Up

1. Clean up your lab station, and return all the parts, wires, the Beti trainer board, etc. Leave your lab workstation for others the way you would like to find it.
2. CONGRATULATIONS! You are finished with Lab #5 and are one step closer to becoming a computer engineer.

Notes

- Advance work on this lab, and all labs, is strongly suggested.
- Be sure to read and follow the Policies and other related material for CS223 labs, posted in Moodle.

Lab Policies

1. There are three computers in each row in the lab. Don't use middle computers, unless you are allowed by lab coordinator.
2. You borrow a lab-board containing the development board, connectors, etc. in the beginning. The lab coordinator takes your signature. When you are done, return it to his/her, otherwise you will be responsible and lose points.
3. Each lab-board has a number. You must always use the same board throughout the semester.
4. You must be in the lab, working on the lab, from the time lab starts until you finish and leave. (bathroom and snack breaks are the exception to this rule). Absence from the lab, at any time, is counted as absence from the whole lab that day.
5. No cell phone usage during lab. Tell friends not to call during the lab hours–you are busy learning how digital circuits work!
6. Internet usage is permitted only to lab-related technical sites. No Facebook, Twitter, email, news, video games, etc–you are busy learning how digital circuits work!
7. If you come to lab later than 20 minutes, you will lose that session completely.
8. When you are done, DO NOT return IC parts into the IC boxes where you've taken them first. Just put them inside your Lab-board box. Lab coordinator will check and return them later.

Recommendations

When building circuits using IC's and FPGAs in CS223 labs, it is important to follow some simple guidelines to prevent damage to electronic parts or confusion during debugging. By following these rules, you can ensure that your circuit functions correctly and avoid potential problems. Here are some key points to keep in mind:

- Avoid touching IC or FPGA pins directly by your hand. Static electricity from your body can permanently damage them. If you have to touch the pins, first ground yourself by touching a nearby grounded surface.
- The white board which you setup your circuit on it, is called "breadboard". Research online and find out how its pins are connected internally and use this knowledge when building your circuit.
- Postpone connecting power pins (V_{cc} and ground) until the last step. Check all other connections first, then connect the power pins if everything seems correct.
- Use a consistent wire color convention for easier debugging of circuits. For example, always use black or white wires for ground and red wires for V_{cc}. This will help you quickly identify any issues that may arise during testing.
- If an LED's light is weak or the IC's package feels hot to touch (you can safely touch the plastic part), there might be a problem with power pin connections, such as a short circuit or connecting V_{cc} wire to ground pin.

Updated by: Barış Bilgin Şenol, February 2024.