CS 223 Digital Design Laboratory Assignment 5
Traffic Light System and Gray Code Counter

Preliminary Report Due: December 9, 2024 08:00

Lab Dates and Times

Section 1: December 09, 2024 Mon. 13:30-17:20 in EA-Z04
Section 2: December 10, 2024 Tue. 13:30-17:20 in EA-Z04
Section 3: December 11, 2024 Wed. 08:30-12:20 in EA-Z04
Section 4: December 09, 2024 Mon. 08:30-12:20 in EA-Z04
Section 5: December 13, 2024 Fri. 08:30-12:20 in EA-Z04
Section 6: December 10, 2024 Tue. 08:30-12:20 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.

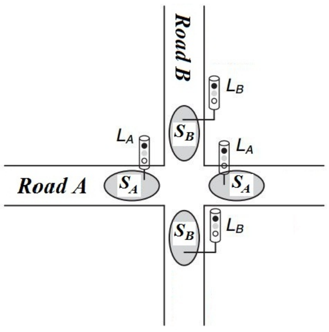

Figure 1: Model of the intersection.

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.

Gray Code

In the previous lab, you created logic to drive the seven-segment display. This lab will build on that previous work. Therefore, you can re-purpose some of your former work.

Consider a 4-bit binary counter currently storing the decimal number 7; its internal binary representation is `0111`. When the count rolls over to 8, the internal bits become `1000`, requiring all 4 bits to switch simultaneously. Although numbers 7 and 8 have a distance of 1 between them, the binary representations bear little resemblance to each other. This phenomenon extends to n-bits: whenever the number rolls over from $2^n - 1$ to 2^n, $n+1$ bits must change state at once. In electronics, current is drawn whenever a bit transitions from low to high or vice versa; however, power consumption is minimized when the state of each bit is preserved. Consequently, it might be desirable to develop a design that minimizes the amount of bit flips during a "+1" counter operation. Furthermore, numerous simultaneous bit flips can introduce errors into the circuit.

Building on this concept, you will implement a Gray Code counter based on the principles established by Frank Gray [1]. Examine the numbers presented in Table 1, comparing their binary and Gray Codes in detail. Notice that the Gray Code necessitates only a single bit to transition between states at each increment.

Table 1: Horizontal timing.

DECIMAL	BINARY	GRAY
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

In binary and Gray Code systems, the counts of zeros and ones at each bit position are identical and comprise half of the total number of decimal values that can be represented within a given range. For example, in the 4-bit scenario, there are eight logic-level lows and eight logic-level highs for all bits, differing only in their order of arrangement. Therefore, there is a pattern to generate a n-bit Gray Code counter. Figure out how to design a circuit that increments in Gray Code domain.

According to the described operation:

- **[5]** Describe the algorithm to generate n-bit Gray Code recursively.
- **[5]** Write a SystemVerilog module for a 4-bit Gray Code up-counter with enable signal and parallel load. Prepare a testbench for it to verify functionality.
- **[5]** Write a SystemVerilog module for a 8-bit Gray Code up-counter with enable signal and parallel load. Prepare a testbench for it to verify functionality.

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: Gray Code Counter on FPGA [35]

In this part, you are going to implement your Gray Code counter on the FPGA and demonstrate your counter on the seven-segment display using the driver you created in the previous lab.

Note: You can directly implement the full four-digit Gray Code counter and get full points from this section. The bitwise, single-digit, two-digit Gray Code 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] **<u>Test</u>:** Synthesize, implement, generate bitstream file, and download your **4-digit bitwise Gray Code 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. Wire each bit in your counter to individual digits so that you can clearly visualize your counter follows the pattern in Table 1. In this step, each digit in your display shows either 0 or 1.

[5] **<u>Test</u>:** Synthesize, implement, generate bitstream file, and download your **Gray Code 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 to show the hex value.

[10] **<u>Test</u>:** Synthesize, implement, generate bitstream file, and download your **two-digit Gray Code 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 to show the hex value.

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

[15] **<u>Test</u>:** Synthesize, implement, generate bitstream file, and download your **four-digit Gray Code 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.

Important Note: While your counter increments bits according to the Gray Code, your display should represent these changes as if they were occurring within a standard binary counter. This mismatch will create an appearance of erratic jumps in the count value, allowing you to observe cyclic movements in bit transitions. As an example, assuming you are starting from 0: your counter should display: `0000` → `0001` → `0003` → `0002` → `0006` → `0007` → `0005` → `0004` → `000C` → `000D` → `000F` → · · ·

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 30 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.

References

[1] Frank Gray. Pulse code communication. *United States Patent Number 2632058*, 1953.