Lab number: LAB 7
Name: Diya Saha
Date: 5/29/2022
Section: C

Description:

We built a lab called frog frenzy which imitates a frog game where the frog is controlled by an up and down button and the player has to avoid obstacles. This lab also contained a score displayed which shows how many leaves(obstacles) the player has successfully doged. This lab was a comprised of modules that were created in the previous labs.

Design:

VGA controller

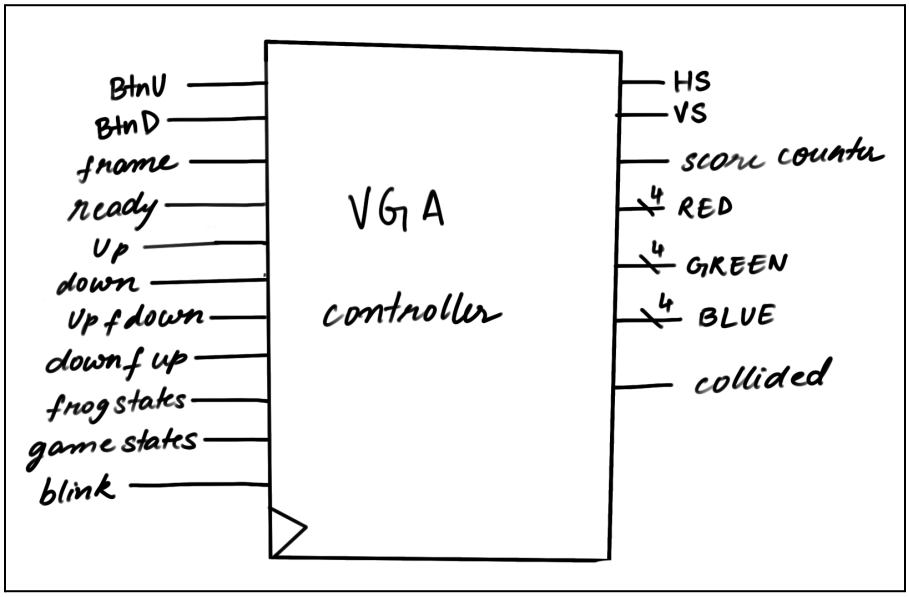

Fig.1. VGA control diagram

This module in the lab was my main draw module and it was in charge of displaying all the items onto the VGA monitor. This module controlled the RGB for each pixel. I had three outputs Red, Green, and Blue which then connected to the VGA red, blue, and green in the top level. I had one wire for each active region like the, water, plant, and frog and I defined each active region using my horizontal and vertical counters. The horizontal and vertical counters were the pixel address in this case.

To make the water region I defined the active region as the lower half of the screen, 240 onwards. To create the gradient of the water I calculated the difference from the middle point and then used the lower four bits to calculate the shade of the blue.

The plant part was easier to implement. I started by just hardcoding the region and getting a static green rectangle to appear on the monitor. Later while implementing the frog movement I applied the same logic to move the plants. I made three different 10-bit counters for each plant that load in different counter values depending on whether the plant reached the respawn area or the game reset.

This part of the project was the hardest to implement just because it took me a while to understand how the frames were being drawn. I first moved the frog at one pixel per frame and later used a pulse extender to change the speed to three pixels per frame. This helped me understand and visually see where each pixel was traveling. Understanding that the bounds of the frog's active region were constantly changing was what helped me understand the crucks of this lab. My frog moves using an offset counter that is synchronized with the states from my frog state machine. In every frame the offset keeps getting bigger and the frog moves further away from the center. Once the offset hits 96 pixels it starts counting down and the frog begins traveling back to the center until the offset is 0 pixels.

Frog State Machine

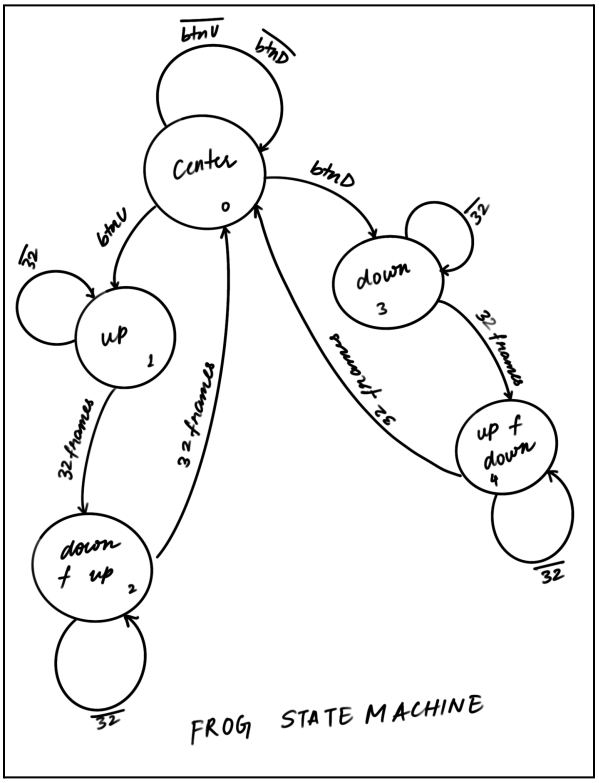

Fig.2. Hand drawn frog state machine

This state machine was purely used to control the mechanics of the frog. I had 5 states, center, up, down from up, down and up from down. The frog center state would remain in the same state as long as the player doesn't press the up(btnU) or the down(btnD) button. If the up button is pressed the play moves into the up stage, and a counter is starts counting upto 32. The player remains in this stage for 32 frames and then proceeds to the next stage, down from up. Similarly the play remains here for 32 frames and then proceeds back to the center. Each stage shift triggers the frame counter which lets the game keep track of how many frames have passed. When the down button is pressed the player goes through the same process, remains in the down state for 32 frames and remains in the up from down stage for 32 frames before finally travelling back to the center.

Game State Machine

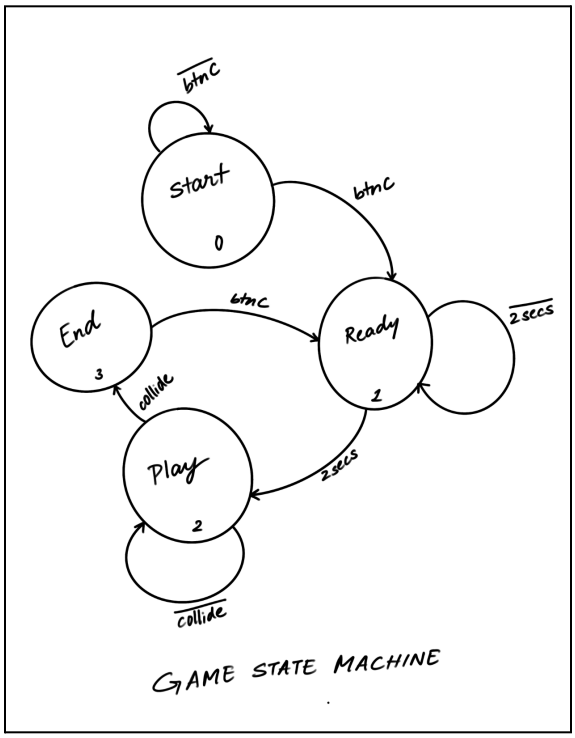

Fig.3. Hand drawn game state machine

This state machine was purely used to control the overall game mechanics. I had 4 states, start, ready, play and end. When the player did not press any button they remain in the start state. When the start (btnC) button is pressed the player is moved to the Ready state and remains there for 2 seconds before moving onto the Play state. The Play state is where all the playing happens so it keeps looping in that same stage until the player fails to cross an obstacle. If a collision occurs then the player is moved to the next state, End. The player remains in this state until they have pressed the restart(btnC) button and then they return to the Ready state.

LSFR

The LSFR module was imported from the previous lab.

10-bit counter

The 10-bit counter was imported from a previous lab but there were some modifications that were made to it. My previous lab used an 8-bit counter and I just added 2 more flips flops to account for the 2 extra bits. Since I only used this counter to calculate the offset of the frog, I commented out the decrementing function and only kept the incrementing function of the counter. I did this to ensure that my counter did not decrement under any circumstance.

10-bit load counter

The 10-bit load counter was imported from a previous lab I made this counter to keep track of the right edges of the plants. I made this counter specifically to have a load function and a load in value. When I called this function in the VGA controlled I used muxes to decide what number to load in depending on the situation. If the frog died and the player pressed the restart button then the counter would load in the hard coded values of each right edge but if the plant had crossed the entire frame then the counter would load in a 0 to ensure that the plant smoothly transitioned into the screen.

Ring-counter

The ring counter module was imported from the previous lab.

Selector

The selector module was imported from the previous lab.

7-Segment display

The 7-segment display module was imported from the previous lab.

8X1 Multiplexor

The 8X1 mux module was imported from the previous lab.

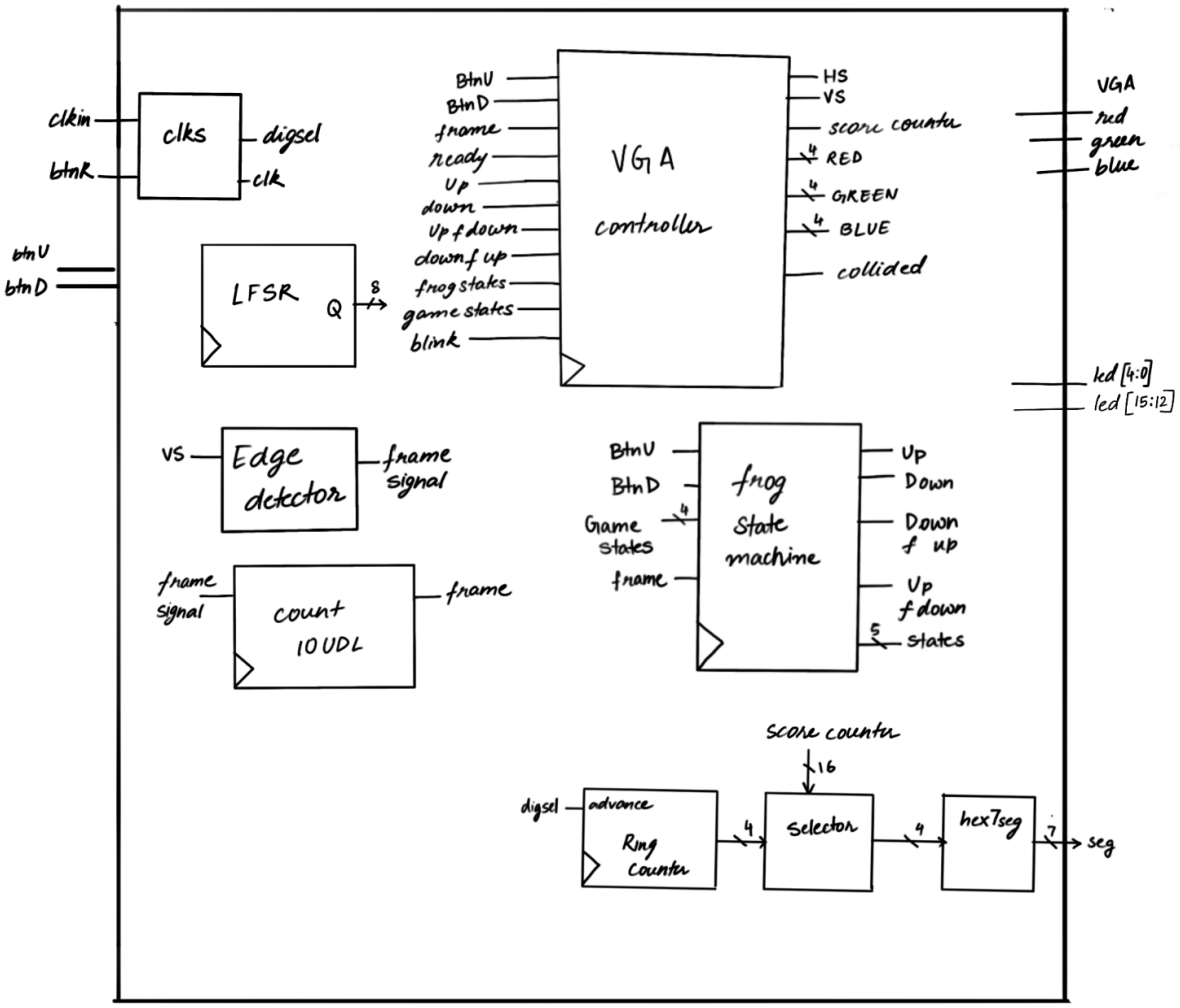

Fig.4. Drawn out top-level schematic

I started the top level by calling the VGA controller. I started with just the red, green, and blue outputs before I started adding inputs and outputs as needed by the module. The VGA controller has the states from both the frog and the game state machine going in and wires that specifically

indicated the transitions from some important states. For example, I needed to know when exactly the frog would start traveling away from the center and back, or I needed to know when the player pressed the restart button and the two seconds had passed. My outputs were the Hsync and the Vsync, the VGA red, green, and blue, and a wire that told the top level when the player had collided with an obstacle. The frog and the game state machine are explained above.

To move the objects on the screen I moved everything at 1 pixel per frame and then finally used a pulse extender using three flip flops to hold the coordinate. I had the frame signal in the top level that was synced to the Vsync. I made another signal, game frame signal that was synced to the game state machine and the frog state machine. In the VGA to move the items I used this game frame signal, which handled all the blinking, pausing and restarting the game. The ringcounter, selector and the hex7segment module were taken from the previous lab and instead of the game counters output feeding into the selector I made a score counter instead. Everytime the player crossed a plant I incremented the score by one.

Testing and simulation:

In this lab, we were given a test bench to test if our Hsync and our Vsync were working properly and if the RGB coloring was being done in the specified given active region. I used this test bench and made a few modifications. Since this test bench was simulating only one row and we needed 479 rows to complete and frame I synced my frame signal to go high whenever the Hsync went high. This meant that in one row I would have 639 frames and I could watch how each of my modules was working in accordance with the frames. I used this method to check my offsets for all the objects like the frogs, water, and, plant.

Once I got my static objects to move on the screen I programmed my LEDs to match the states from both my state machines. I used my LEDs[4:0] to match the frog states and my LEDs[15:12] to match the game states. That way when I was playtesting my game I could see exactly how the states were transitioning and which states they were getting stuck in.

Towards the end of the lab, I also coded my 7-segment display to display the output from my counters to check if the transitions were happening exactly at 32 frames or not.

Results:

The workings of the state machines and the modules have been defined above. The design can run at $\frac{1}{|T - WNS|}$. This is the equation of max frequency. I got the values of T as 10ns and the value of WNS as 30.290ns from the timing report. My calculated frequency was 47.8mHz.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	30.920 ns	Worst Hold Slack (WHS):	0.198 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	310	Total Number of Endpoints:	310	Total Number of Endpoints:	143

All user specified timing constraints are met.

Fig.5.WNS timing summary

Name	Waveform	Period (ns)	Frequency (MHz)
sys_clk_pin	{0.000 5.000}	10.000	100.000
clk_out1_clk_wiz_0	{0.000 20.000}	40.000	25.000
clkfbout_clk_wiz_0	{0.000 5.000}	10.000	100.000

Fig.6. Timing summary

Conclusion:

This lab was very interesting as I got to pull my existing knowledge from all the previous lab and then combine it to form a working model of a game. I think the given time for the lab was a little too short and if I had more time then I would be less stressed in the process. I appreciate this lab a lot because I got to see my code working in real-time and playing the game was a lot of fun too. I appreciated the end result a lot more because I understood the time and effort put into it.

Appendix:

1. Code Modules
    a. Top Level [top_level.v]
    b. VGA controller [VGA_controller.v]
    c. Edge Detector [Edge_detector.v]
    d. Frog State Machine [frog_sm.v]
    e. Game State Machine [Game_state_machine.v]
    f. 10-bit Counter [count10UDL.v]
    g. 10-bit Load Counter [leaf_counter.v]
    h. Ring Counter [RingCounter.v]
    i. Selector [Selector.v]
    j. 7 segment display [hex7seg.v]
    k. 8-to-1 multiplexer [mux8_1.v]
2. Schematic
    a. Top Level schematic
    b. VGA controller schematic
    c. Frog State Machine schematic
    d. Game State Machine schematic
    e. 10-bit Counter schematic
    f. 10-bit Load Counter schematic

3. Waveform simulation displaying all my outputs
4. Lab testbench code for the simulation
5. Scanned copy of my lab notebook

Waveform output from the simulation

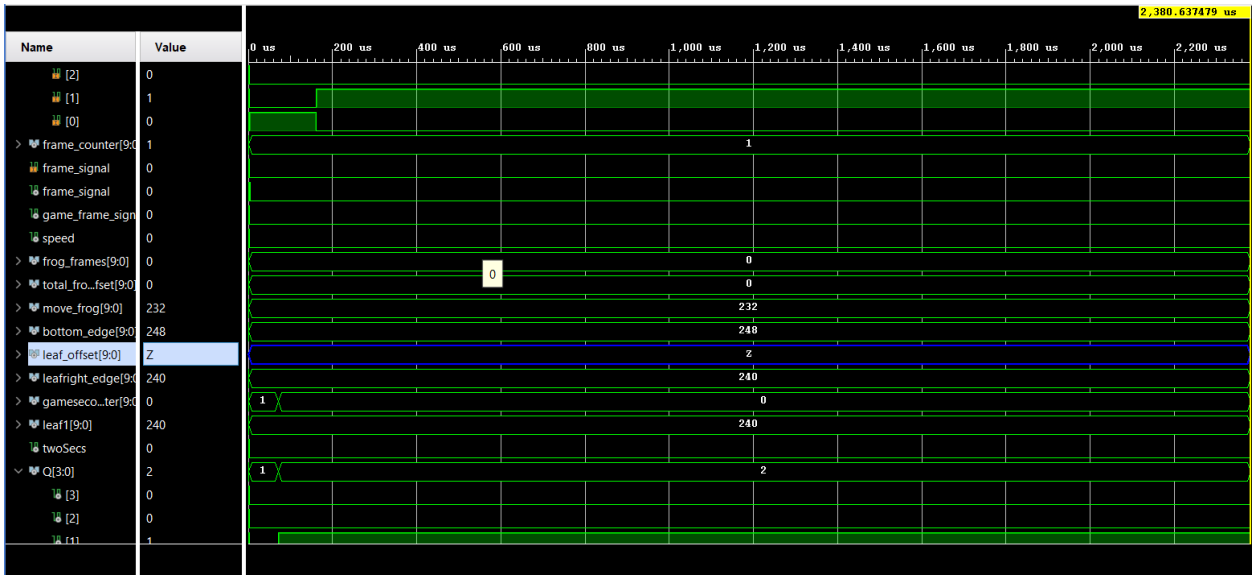

Fig.7. Waveform output of the top-level testbench