Lab Number: 2861

Shreyas Telkar

12/1/22

Section B

CSE100 Lab6 Writeup: Watch Your Step

Description

The goal of this lab was to create a platform game in which a player must collect moving coins while avoiding falling into a hazardous hole in the ground that would make the player flash as they are falling and end the game. To collect and score the coins, the player must jump and make contact with the moving coin and when this happens, the coin flashes, disappears, and then the score count on the Basys board is incremented.

This lab uses the VGA port of the board and a VGA monitor to display the running game. Time counters with a frame signal and separate **Hsync** and **Vsync** signals are set to synchronize the proper playing board dimensions and display objects in the proper locations.

Design

Border:

(x and y coordinates):

In order to print objects on the screen, two pixel address counters (time counters) must be specified to keep track of the x and y coordinates of the playing area. The x counter resets at 799 and the y position only increments whenever x hits 799 since pixels are encountered left to right and the row is incremented after every column is. The y coordinate is then reset at 524 which is the end of the board. Both of these counters are incremented every frame.

(Region):

As shown by the diagram, to specify the region of the border, the x region is specified and that is “&ed” with the y region for each one of the 4 sides of the border. Then, this logic is “&ed” with each vga color signal and passed into flip flops to the output.

Bottom Platform:

Similar to the border, the x and y coordinates from the time counters are used to specify the region of the platform. I.e. x >= left_num, & x <= right_num & y >= top_num & y <= bottom_num. This is then passed into each vga signal which is "&ed" with the hex color representation passed into flip flops. This result is then "or'ed" with the other vga signals to display all of the vga color signals from all of the modules.

Vga_border | Vga_platform | …

Top Platform (moving):

To create the top platform (not-moving) the same logic can be used above with the bottom platform. A Shift left input is "&ed" with the logic to draw to either show the moving platform or stationary.

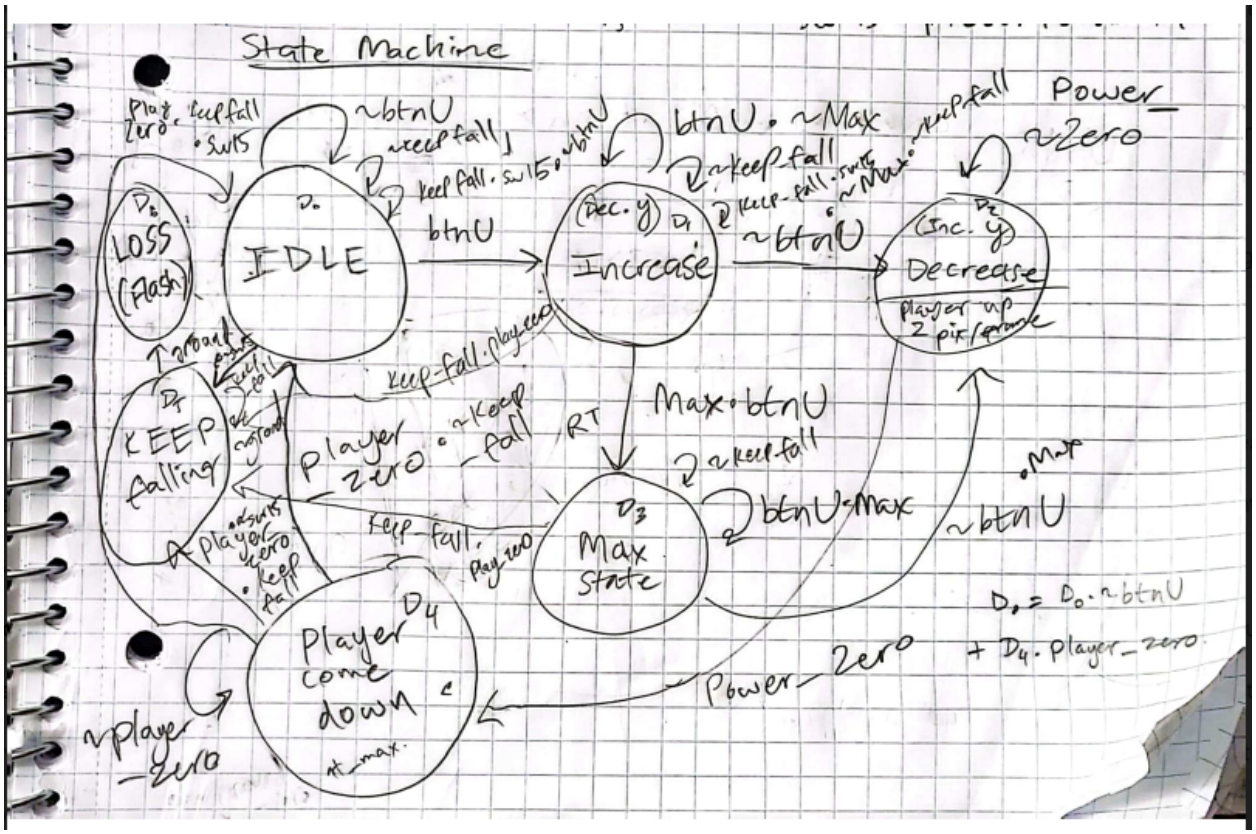
~SHL & not-moving | SHL & moving

To create the moving platform with the hole in it, a random width from 41 to 71 must be specified. This can be done by calling the LFSR which generates a random number from 0 to 31. To get the random number in the proper range, 40 is added to the number. Then to hold this value until the next one needs to be generated, the value is passed through a flip flop with reset being when the right edge hits the end (left side of the screen, after the left side of the border). Two counters are used to move the hole. To keep track of the current left and right side of the hole, two wires are used and are decremented by the counter output to move the hole left. The starting left and right sides of the hole are specified as the left edge being 631 and the right edge being 631 plus the output of the LFSR. This is then moved by the counters every frame (counter-increment input). The counter is reset when the right side hits the edge.

To detect when the left edge hits the left side so that the left side does not hit negative values, when the right edge is less than the width of the hole, the left edge is set as 7 so that it does not go below that when the right edge is close to the left side.

Player Jump and Power Bar State Machine:

The player is drawn the same way the platforms are drawn. However, when the game is in certain states, The player is drawn with the current top edge and bottom edge, depending on the jumping.



States Explanation:

Until btnU is pressed, the player is resting on the platform. Then when the button is pressed and held, it is in the increased state where the power bar increases. If the power bar reaches the max value, then it goes to the max state until btnU is released. In either case, when btnU is released, the game transitions to the decrease power bar state and the player jump up state until the power bar decreases to zero. Then, the player will come down either to the top platform or to the ground if the player is over a hole. If over the hole, the player flashes and stays in the last state. Switch 15 is used to prevent the player from falling into the hole. If switch15 is high and the player is under the hole, then it will stay in the start state.

Logic:

Two counters are used to find the current height of the power bar and two counters are used for the current height of the player. The power-bar counters are reset when they hit zero and are incremented or decremented in the specific states. The same goes with the player, the top and the bottom edges of the player get incremented by either counter depending on the state.

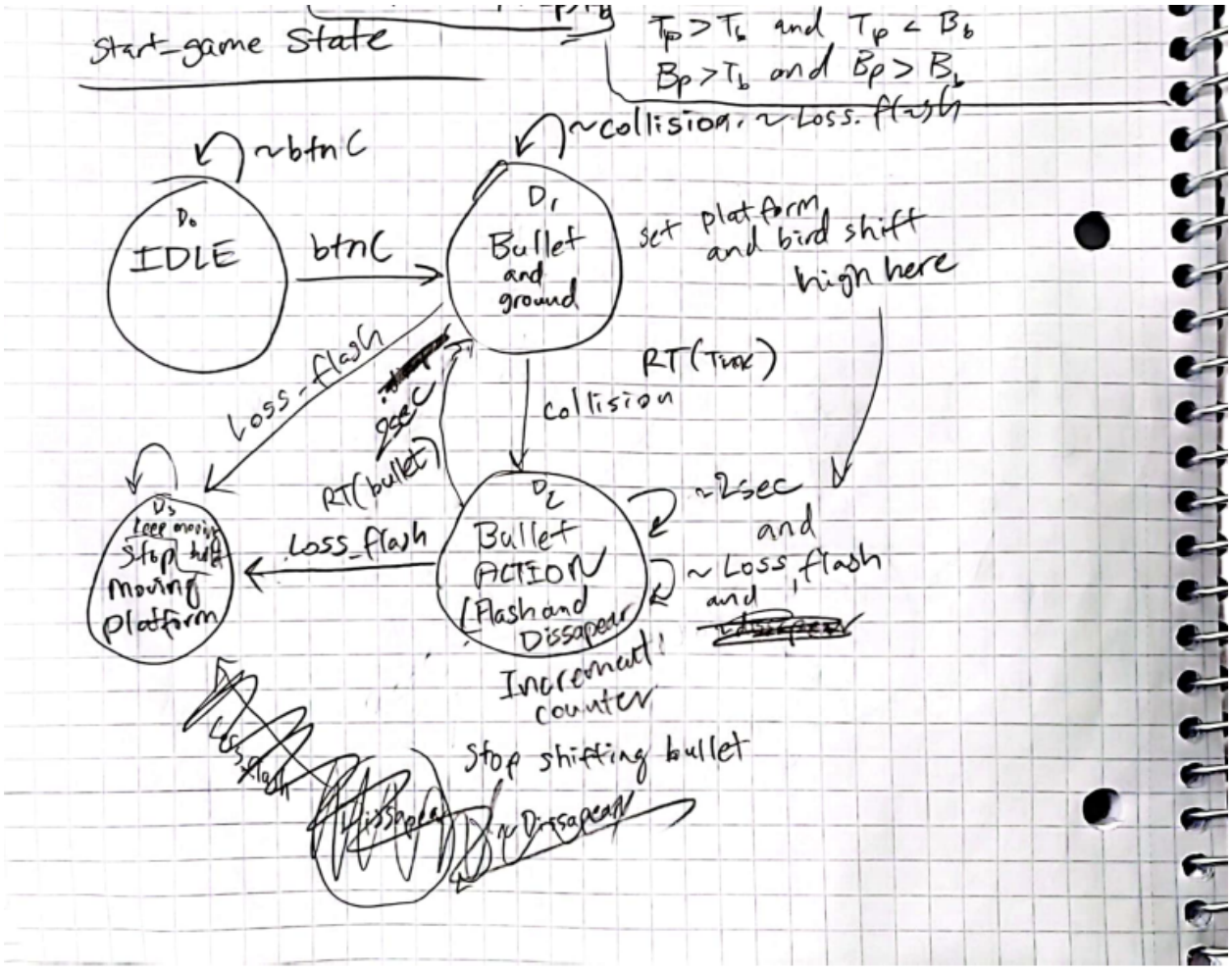
To make the player flash, a signal from a counter output is taken and counter[4] is anded with the draw logic of the player to make it flash when it is in the loss or falling state.

Ex)

Cur_left = start_left + counter_up;

Cur_right = start_right - counter_up;

Ball logic and state machine



To start the game (start moving the coin and platform), btnC must be pressed. Otherwise the player will be in the IDLE state still possible to jump from the other state machine. The inputs are passed in from the logic module. In D1, the floor and ground start moving until the player collides with the ball. Then it transitions to the state where the bullet must flash for 2 seconds and disappear. After 2 seconds, another ball is generated and there is a loop until the player loses at any of the two states. When the player hits the ground and loses from the other state, then the platform stops moving and the bullet continues to move.

To make the ball flash for 2 seconds, a counter output is taken with a frame signal input. Two seconds is 120 in decimal so, SOP is done with the counter output for the decimal value 120 in binary. Then it is passed in as an input to the state machine.

Top

To get a frame signal, a random point outside of the active region is taken and passed in as the frame. Multiple frame signals are ored to make objects move at greater than one frame per second. Every module is called in the top module and inputs and outputs are passed in and out of these modules. Eventually, the vga outputs are ored and displayed. Then, to display the score on the board, the collision transition in the ball state machine is used as the increment in a counter in the top module. This is then passed into the selector and then the hex7seg. An1 and 0 are high only.

Testing and Simulation

Initially, to test the Hysnc and Vsync values, a simulation was run and the expected results were shown. To test the modules, experimenting with reset inputs and state outputs took me the longest to solve. Working around issues with setting values at different states and setting edge coordinates to their appropriate values solved most of the issues after tinkering for a while.

Results

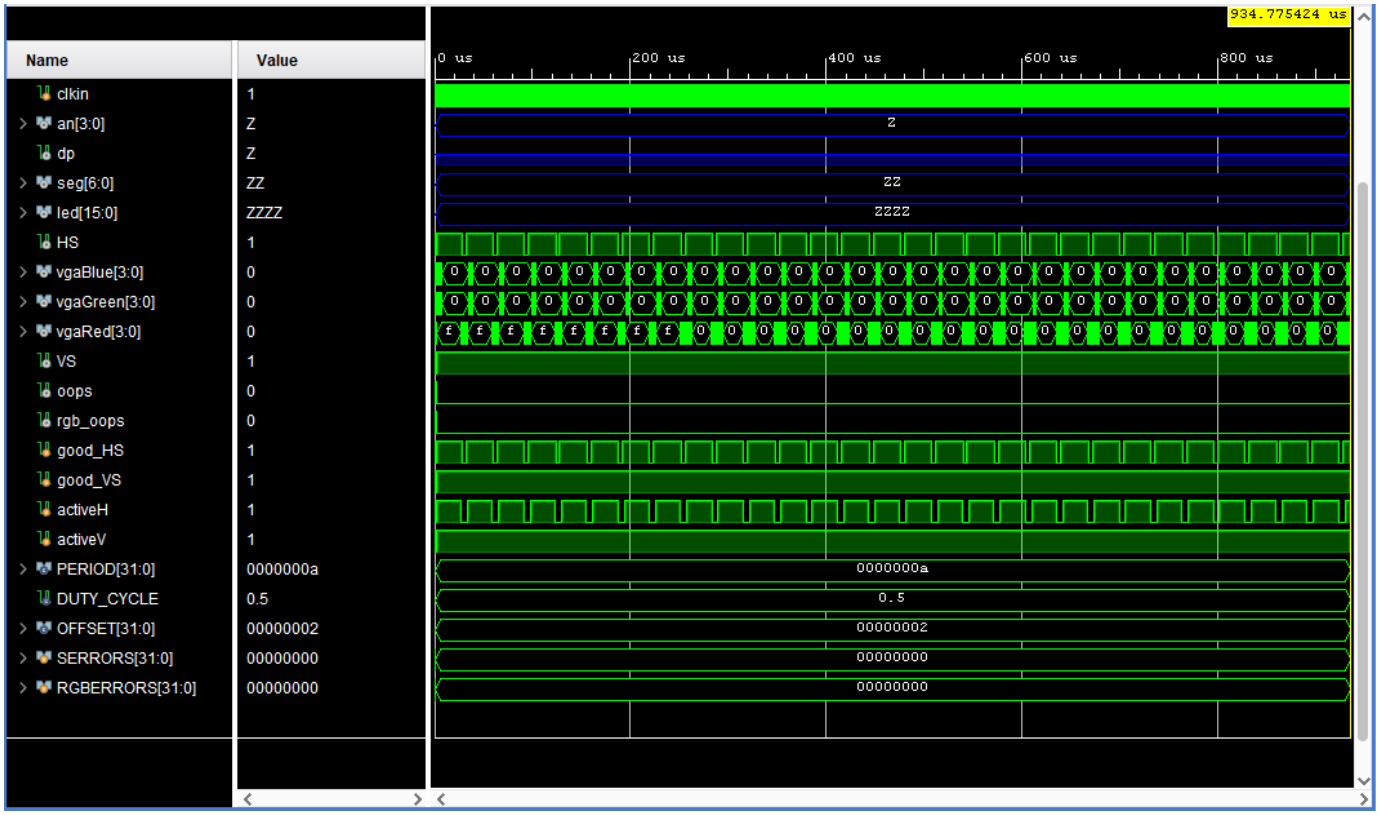
The schematics were explained above and specific inputs are shown in the code as well as described above. The game functioned properly after implementing all of the components of the game.

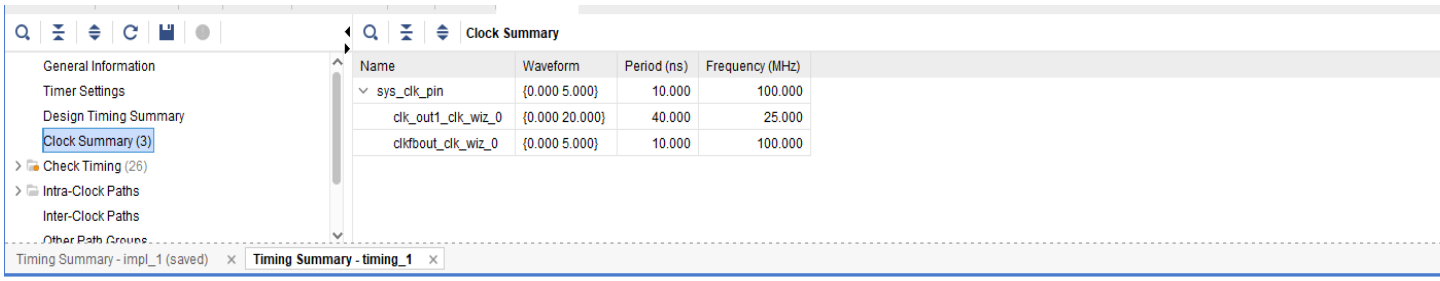
The worst negative slack is 25.615ns and the period is 40ns. The maximum clk frequency is 25 MHz.

Conclusion

This lab taught me a lot about how vga inputs work and how to create moving parts on a vga display by applying what we learned in previous labs about state machines and heavily used modules such as counters. I learned how to make a game using multiple state machines and connected inputs and outputs. I had significant difficulty with the power bar as it would display max values at random times and the character would not jump with the power bar too. I spent a lot of time debugging my counters and state machine inputs and outputs to resolve the issues I had. If I could do this lab again, I would plan out my inputs and outputs better, I kept making new wires trying to connect everything and kept changing inputs and outputs when I could have just planned it out better by writing it out. Many modules can be optimized as I made sure to check all of the cases in my code which led to long lines of code. Minimizing this code can optimize the readability, simplicity, and run of the program.

Appendix


934.775424 us
Name
Value
0 us
200 us
400 us
600 us
800 us
clkin
1
an[3:0]
Z
dp
Z
seg[6:0]
ZZ
led[15:0]
ZZZZ
HS
1
vgaBlue[3:0]
0
vgaGreen[3:0]
0
vgaRed[3:0]
0
VS
1
oops
0
rgb_oops
0
good_HS
1
good_VS
1
activeH
1
activeV
1
PERIOD[31:0]
0000000a
DUTY_CYCLE
0.5
OFFSET[31:0]
00000002
SERRORS[31:0]
00000000
RGBERRORS[31:0]
00000000


General Information
Timer Settings
Design Timing Summary
Clock Summary (3)
Check Timing (26)
Intra-Clock Paths
Inter-Clock Paths
Clock 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
Timing Summary - impl_1 (saved)
Timing Summary - timing_1