Santosh Shrestha

#1841162

Section 1

Lab 6: Subway Surfers

Description

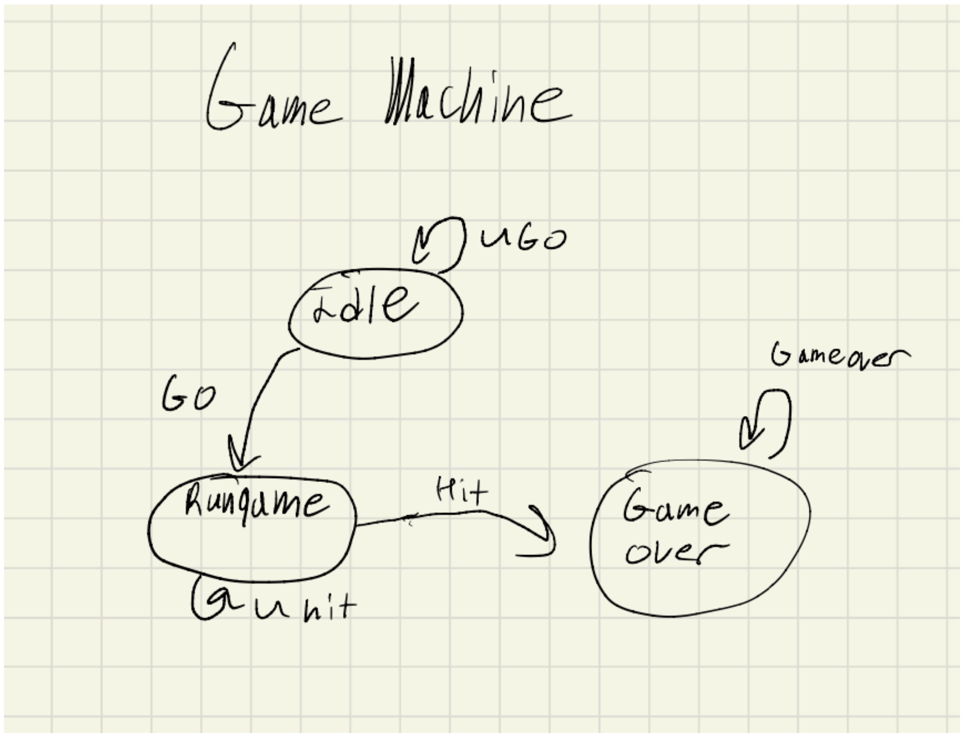
The objective of this lab was to create a 2d version of "Subway Surfers" with some differences from the original game. When the player turns on the game the player is idle and nothing happens until they press btnC which runs the game. The players' movement will be limited to the predefined Middle, Left, and Right positions on the screen meaning the players' slug will slide to the position they want with the btnR and btnL buttons. The player must dodge oncoming trains that vary in size and are randomized in the order they appear. The player can accumulate points by passing the oncoming busses, these points will be displayed on the 7-segment display of the board. Players are allowed to hover over these trains with btnU only when they are in the middle position and when their energy bar is not empty. Hovering allows players to glide over a train so they are not hit by the trains and the player will begin to flash a different color when they are pressing btnU. The energy bar will refill whenever the player is not pressing it or dead. The player will die once they hit any portion of the train or if they are hovering and run out of energy while hovering over a train. Once the player dies all trains, slug, and energy bar movement should stop and the player begins to flash a different color indicating their death. The only way for the player to continue playing is by restarting the game with the btnD button. The player can also turn on an invincibility switch (sw[3]) which allows them to play the game without ever dying.

Design/Modules

GameSM

The GameSM module was mainly created so that I could indicate to all of my CE in my flip flops and 15-bit counters when to start and stop. The player can only be in 3 states within this state machine Idle, Rungame, and GG.

Figure(1): Game State Machine Diagram



Inputs:

- Go: connected to btnC
- Hit: Indicates when the slug has come into contact with a train
- Clk: Used for the flip flops clocks

Output:

- StartGame: Becomes high when the player has entered the Rungame state.
- Idle: Becomes high while the player is in an Idle state.
- GG: Becomes high when the player has been hit by a bus and used to turn off every CE in every Counter and flip flop.

States:

Idle: The player hasn't pressed Go

Rungame: The player has pressed Go and the slug hasn’t collided with any trains.

GG: The player has collided with a train and everything should stop moving and end the entire game.

Equations:

```
NS[0] = PS[0] & ~Go;

NS[1] = (PS[0] & Go) |(PS[1]& ~Hit);

NS[2] = (PS[1] & Hit) | PS[2];


StartGame = PS[1];

GG = PS[2
```

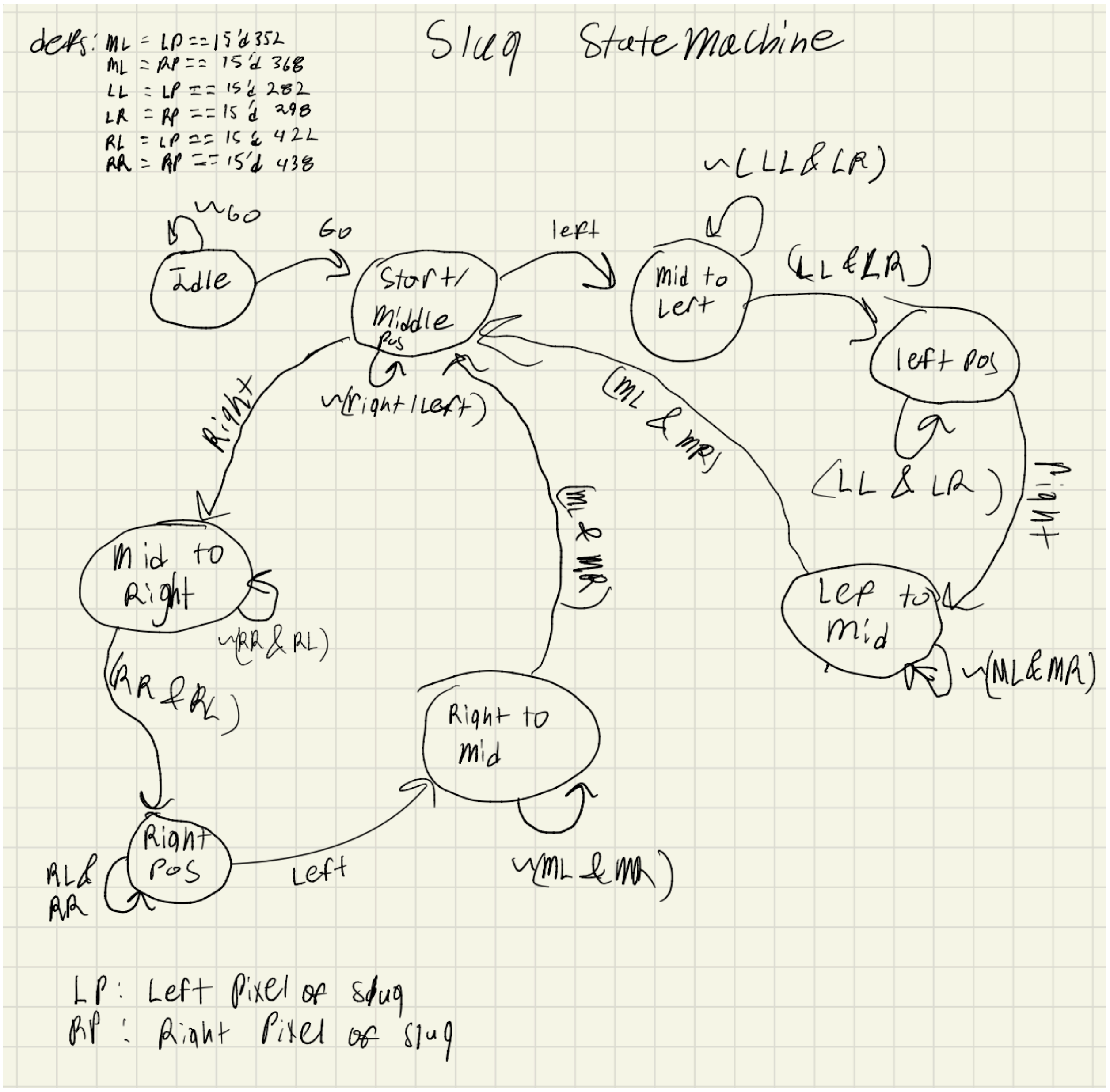
Slug

My Slug function implements the coloring of the slug and as well the movement of the slug as well as the coloring of the slug. I implemented two counters for the left and right pixels of the slug that would only turn on when the player has pressed left or right which is indicated from the output of the state machine (SL, and SR), when the game hasn’t ended and the if either of the edge detector outputs are true. I connected to two edge detectors so that the slugs are synced with the frames of the screen. The output of my pixel counters is connected to the H value which causes the slug movements

SlugSM

Slugsm was the module I created to control the Slugs" movement I took in the 15-bit counters I made within the Slug module that indicated where the slugs Left and Right pixels were located and I defined the location of what the Left, Right, and Middle positions of the slug should be. When I was creating my state machine I was thinking more in the order that the player shouldn’t be able to move left or right while they are moving so I wanted them to stay in a specific state if they were moving and that the slug shouldn’t be stationary in anywhere but the define pixel locations that are used to indicate the positions. The state machine has a total of eight states, with three corresponding to movement from the middle and right side of the screen, and another three corresponding to the left and middle movement of the screen.

Figure(2): Slug State Machine Diagram



My state machine takes in the inputs listed below:

btnR: indicates right movement

btnL: indicates left movement

btnC: indicates Go or start the game

btnU: indicates Hover

Clk: Used as the clock input for the flip-flops

Hit: Indicated when a player was hit by a train

GG: Indicates the game has ended

LP: Input of the Left pixels movement and helps keep track of where the slug is currently.

RP: Input of the Right pixels movement and helps keep track of where the slug is currently.

Outputs:

SL: Outputs when the player is currently in the “Middle to Left state” or “Right to Middle state” and the game hasn’t ended

SR: Outputs when the player is currently in the “Left to Middle state” or “Middle to right state”, and the game hasn’t ended

Mid: Outputs when the player is in the center this is used to indicate when the player is hovering in the top mod

- Chill state: In this state, nothing is happening and the slug is waiting for the player to press the btnC button (Go button)
- Middle state: Go has been pressed and the player is in this state when their Left and Right pixels (LP and RP) are at their current position of 352 and 368.
- Middle to left state: The player has pressed btnL while in the middle state and will stay in this state while the LP and RP haven’t moved exactly 60 pixels to the left
- Left state: The slug will enter this state when the counter values from the Slug module that are being inputted have counted to exactly 282 and 298 for the LP and RP. The slug

will remain in this position until the btnR button is pressed or while the pixels are still in this position

- Left to Middle state: the player has pressed btnR while in the Left state and will remain in this position while the LP and RP are not equivalent to 352 and 368
- Middle to right state: The player has pressed btnR while in the middle state and will stay in this state while the LP and RP haven't moved exactly 60 pixels to the right
- Right state: The slug will enter this state when the counter values from the Slug module that are being inputted have counted to exactly 412 and 428 for the LP and RP. The slug will remain in this position until the btnL button is pressed or while the pixels are still in this position
- Right to Middle state: the player has pressed btnR while in the Right state and will remain in this position while the LP and RP are not equivalent to 352 and 368

Equations:

NS[0] = PS[0] & ~btnC

NS[1] = (PS[0] & btnC) |(PS[1]& ~btnL & ~btnR) | ((PS[4]|PS[7]) & (MR & ML))

NS[2] = (~(LL & LR) & PS[2] & ~PS[4]) | (btnL & ~btnR & PS[1])

NS[3] = (PS[2] & (LL & LR) & ~(RR & RL)) | (PS[3] & (LL & LR))

NS[4] = (~PS[2] & btnR & ~btnL & PS[3]) | (PS[4] & ~(MR & ML))

NS[5] = (btnR & ~btnL & PS[1]) | (PS[5] & ~(RR & RL) & ~PS[7])

NS[6] = (PS[6] & (RR & RL)) | (PS[5] & (RR & RL) & ~(LL & LR))

NS[7] = (PS[6] & btnL & ~btnR & ~PS[5]) | (PS[7] & ~(MR & ML))

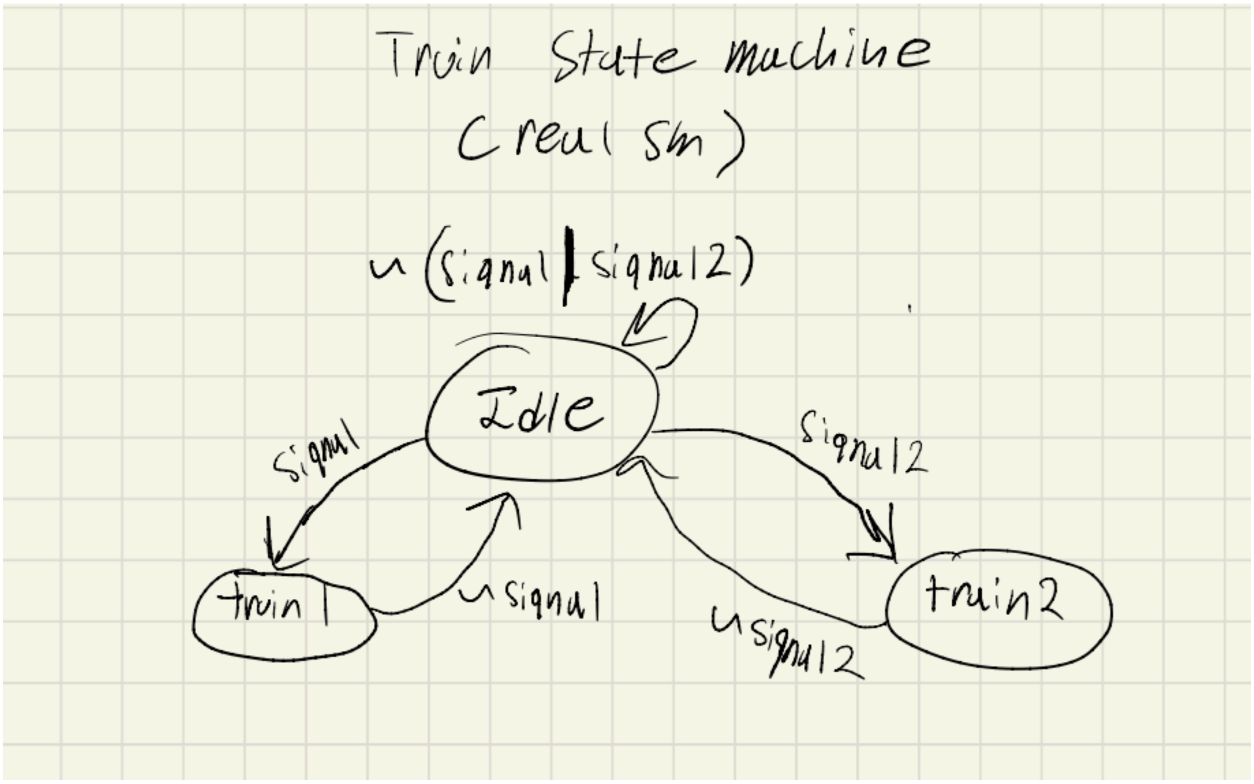
NS[8] = ((PS[1] |PS[2] |PS[3] |PS[4] |PS[5] |PS[6] |PS[7] | PS[8]) & Hit)| PS[8]

```
Mid = MR & ML;

SL  = (PS[2] | PS[7])& ~btnU  & ~GG ;

SR  = (PS[4]| PS[5]) & ~btnU  & ~GG;

Note:PS[8] was not used
```

Realsm

The Realsm function controls the feedback signal that is used to send from train 1 to train 2 of each train in the respective lane. This function is called three times to control each track and only has 3 states. The input signal refers to when the respective train has hit either 400 pixels for the right and left tracks or in the case of the middle track 440 pixels. When the train has hit that marker it outputs a signal I use to indicate that the other trains should begin its spawning process as well. As long as the game is playing and trains are being spawned the state machine will constantly swap between Idle, train1 signal, and train2 signal, states. The main reason I created a state machine for this was because I wasn't able to use the definition of the the train 400-440 pixels as the loopback/feedback signal since it didn't last long enough for a train to fully render on the screen.

Figure(3): Train state machine Diagram



Inputs:

- Clk: Used for the flip flops
- Signal: Used to indicate when the first train has been activated and when the second train has passed its marker
- Signal2: Used to indicate when the second train has been activated and when the first train has passed its marker
- GG: Indicates when the player has died and that all signals should be stopped being sent.

Outputs:

- S1: Singal sent out to train 1 so that it knows that train1 should start preparing to spawn
- S2: Singal sent out to train 1 so that it knows that train2 should start preparing to spawn

States:

- Idle: This state is entered if neither of the trains has sent a signal or if one of the trains has sent out a signal
- Train1 signal: This state is entered when train2 has passed the 400/440 marker or the game has started and they had waited for their 2 or six-second delay. Once the signal is turned off go back to idle.
- Train2 signal: Train 1 has passed 400 pixels and remains in that state while it's active. Once the signal is turned off go back to idle.

Equations:

NS[0] = (PS[0] | PS[1] | PS[2]) & (~Signal | ~Signal2);

NS[1] = (PS[0] & Signal);

NS[2] = (PS[0] & Signal2) ;

S1 = NS[1] & ~GG;

S2 = NS[2] & ~GG;

Traincall

Traincall is used to control a singular train's movements and is only active when the respective train has received its Relay value being high and the game hasn't ended to activate a flip flop so that the LFSR value will only be passed through when the relay has gone high and it will remain high for one clock cycle. I also added another flip-flop with the same CE as the prior flip-flop with the output variable name “fall” which will restart when the top of the train has exited the train which is 479 pixels. The “Fall” variable is used in both of my CEs for the top and bottom counters for the train.

In the 15-bit counter instantiations, I created a bottom train counter that would become active (CE) when the PC (pixel column or frame), falls, the game hasn’t ended all true and the output of the bottom train counter has exceeded 479 indicating the end of the screen. I also added a top train counter that would become active with the same CE as the prior train however the bottom train counter value has to be above 60 plus the first 5 bits of the LFSR value to randomize the train sizes. Both timers will also reset when the top of the train is past the top of the screen.

I made my train function a bit modular so that I could pass in the starting right and left pixels of where the train should spawn. This made me have to add another variable to let the train know when it should be sending its Signal output to the other trains' Loop input with whether the train should spawn past 400 or 440.

I also assign the bounds of the train to an output called color so I can then add it to the VGABlue output in the top mod and outputted the top train counter output to the top mod so I can use the position of the train for the score counter

Syncs

The Syncs module controls the active region of the screen the V is related to the columns and the H is the horizontal pixels of the screen. V is high when it's not below 490 and not above 480.

The H value is high when it is not above 655 and not below 750 and is then outputted as the Hsync and Vsync values

VGA

The VGA function I created encapsulates the screen and defines the borders of the screen making it display white borders and the output is sent to the VGARED, VGABLUE, and VGAGREEN outputs. I had mainly outputted the needed values for the coloring in each of my functions then added it to the VGARED etc output

Pixel Address

The pixel address module reveals where the columns and rows refresh with two 15-bit counters. One corresponds to the Horizontal portion of the screen and the other corresponds to the vertical portion of the screen and the screen resets at 799 pixels for both.

Energy Bar

The overall idea of this module is that it should display a green bar on the left of the screen and decreases when the player is in the middle and is pressing btnU. That would cause the energy bar that is displayed to go down until it is empty.

I implemented this by setting the vertical bounds of the bar to be from V < 300 pixels and V >= 300 plus my 15-bit counter value. The 15-bit counter would help define the upper portion of the energy bar and when the counter began to increase the upper bound would begin to decrease the bar because the top portion of the screen is 0 and that number increases as you point at lower portions of the screen. This allowed me to use my 15-bit counter to decrease the energy bar by increasing the V greater than equal to the portion. I set the load of the 15-bit counter to always rise when btnU isn't pressed by setting it as the UD and having a condition where the CE is active when the user isn't pressing the button. I output the defined region of the energy bar for coloring as well as an indicator of when the energy bar is empty so that the top mod knows when to let hovering be high or low.

Top Module

My top module displays what is on the screen with the use of the given VGA clock function along with the VGA controller, Syncs, and the Pixel address.

Inputs:

Clkin: used for the clk input for multiple functions, flipflops, and counters

- btnR and btnL: Used for Right and Left movement of the slug
- btnC: Initializes/starts the game
- btnU: Allows the slug to flash while they have energy and hover over oncoming trains only when they are in the center
- btnD: Resets the entire game
- sw: overall they do nothing besides sw[3] which makes the character invincible meaning they will never die and allows the player to continue playing the game even in their death.

Outputs

- vgaRed, vgaBlue, vgaGreen: Used to change the colors of the trains, slug, flashing, energy bar, etc.
- Hsync and Vsync: Synchronizes the screen to refresh and display the screen and movement of trains etc
- seg, dp, an: Displays the active score of the player and lets them see the current score of the game
- Led: I mainly used this for testing to see what state I was in or to see if any value went high or low.

Logic

I use counters for the x and y movement control along with edge detectors to sync the counters with the important frame timing for anything that is displayed on the screen. This is mainly because I use specific pixel locations to indicate movement or scoring.

The majority of my game is controlled by my game state machine which just indicates if the game is running or if the player has died I had Rungame and GG into every module and then added instances of Rungame & ~ GG into every CE within a counter or flip flop to make everything stop moving or working this includes my slug/train state machine my energy bar movement. I output when the slug was in the center from the slugsm to indicate to the energy bar when the player is in the middle so it knows when not to increase or decrease by adding it into its counter CE.

I created two timers for trains one for 2 seconds and another for 6 seconds and I use these to start the signals for right and middle trains since I couldn't connect them to the go button, then that way they once the signal is sent from the first instance of train on each track the signals outputted from each train will be sent to the realSM (train state machine) to allow that signal to be held and sent to the second summoning of trains which will be inputted as their Relay input which enables them to begin spawning and vice versa. This is done times with 2 calls of traincall for each train on each track and 3 calls of realSM one for each train track.

I outputted the live location of the top pixels of the trains so I could then create an assignment so that when the top of a train passes 360 pixels which will allow another 15-bit counter to increase, as a bus passes that point to increase the score counter after. I also used a 15-bit counter for the use of flashing so that it would turn on when the player has either died or they are hovering and inputted that into the VGA output.

Once the player has collided with a train the game will instantly end due to the "& ~GG" I had placed in every CE so that nothing can be moved by the player unless they decide to restart the game.

Testing & Simulation/Results

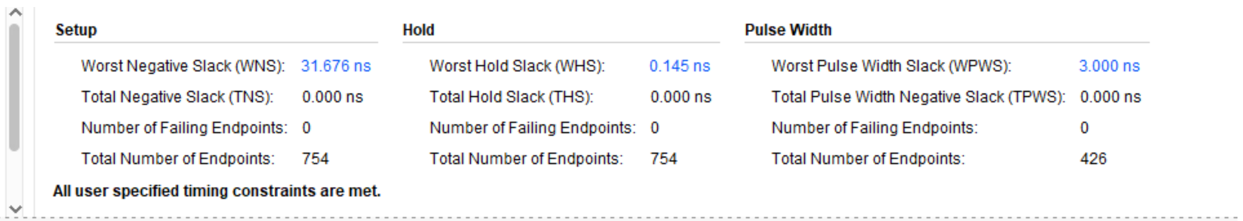
I would mainly test my code manually but would test my state machines with simulations to see if they were entering the proper state or not I had to do this a lot for my slug state machine since it was entering multiple states at once and would even leave the state machine. I would also test my code by seeing if things would display properly or with LEDs to see if a value was high or low and to see if I was entering the proper state. I had issues with my slug state machine also did this for my slug state machines as I was playing the game to see what would cause it to break. However, I was able to quickly fix the issues by testing the state machine with LEDs adding more definitions to each state, and adding more states to it since I originally 2 movement states instead of 4 which in the end was easier to implement and test.

Another big issue was making trains render and it wasn't until I started to see if my counters for my trains were going high with LEDs I realized the issue was that the CE was never becoming true. I then had one of the TAs review my code and told me structurally it looked fine but he wasn't able to find my issue. One of my classmates told me to just recreate the train module on the top mod and then try to work from there I was able to make a train render on the screen after 5 days of being stuck on it and I realized it was because I had messed up the V comparison values for the trains

Conclusion

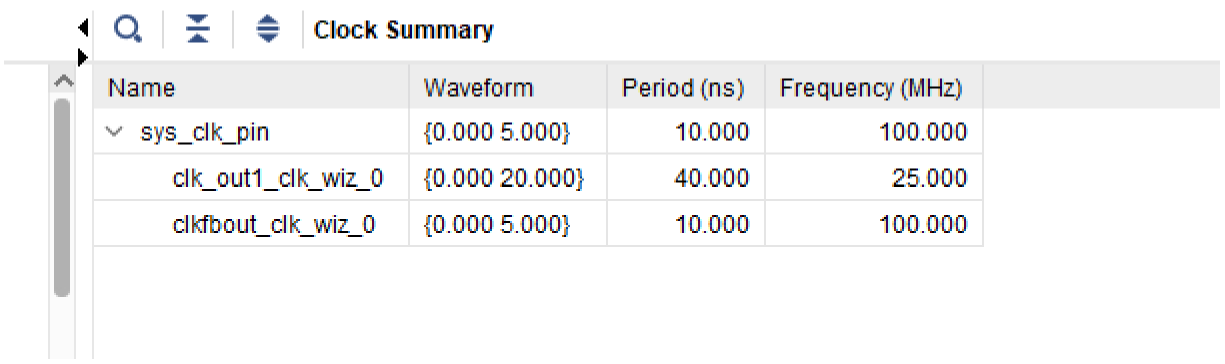
In this lab, I learned how to display things on a screen and learned how VGA pixel addressing and frame syncing work. I feel like those were the harder concepts I didn't grasp at first which was what made me dislike this lab at first. It had made me feel like I didn't understand anything and I was making no traction whatsoever especially when I was having trouble trying to make a single train display on the screen. It wasn't until one of my classmates broke it down compartmentally I could understand how the pixel address framing and as well as how the screen conditionals worked. I also had a lot of difficulty making the trains feedback from one to another because I would just use a boolean however the boolean wouldn't stay high long enough so I implemented a state machine since the outputs of it entering a state and staying there would allow the signal to stay high for long enough. If I were to redo this lab I would probably just tell myself to work on things one at a time because I was trying to do everything for trains all at once and then running the program to see if it would work but it never did until I made my function not care about looping back and the more complicated aspects. Once I was able to make a train to render I was able to make a lot more progress than I did in a week within a few days. There are a lot of components that I would optimize like my slug state machine since it has an extra state that I don't use or even my top mod which has wires or function calls that I don't even use and overall it is a bit hard to follow and understand.

Appendix



Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	31.676 ns	Worst Hold Slack (WHS):	0.145 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:	754	Total Number of Endpoints:	754	Total Number of Endpoints:	426

All user specified timing constraints are met.



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