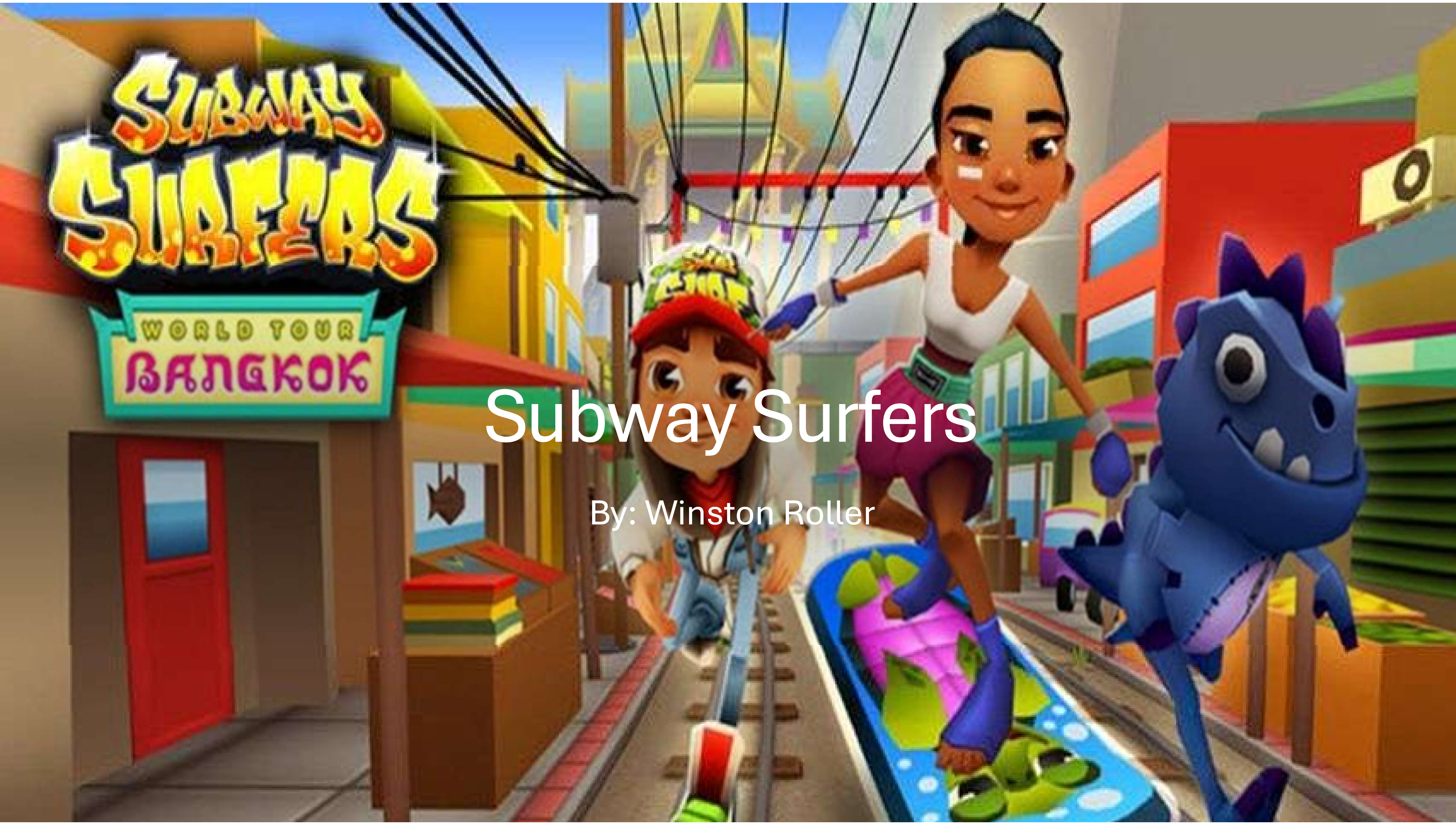


Subway Surfers

By: Winston Roller

Objective

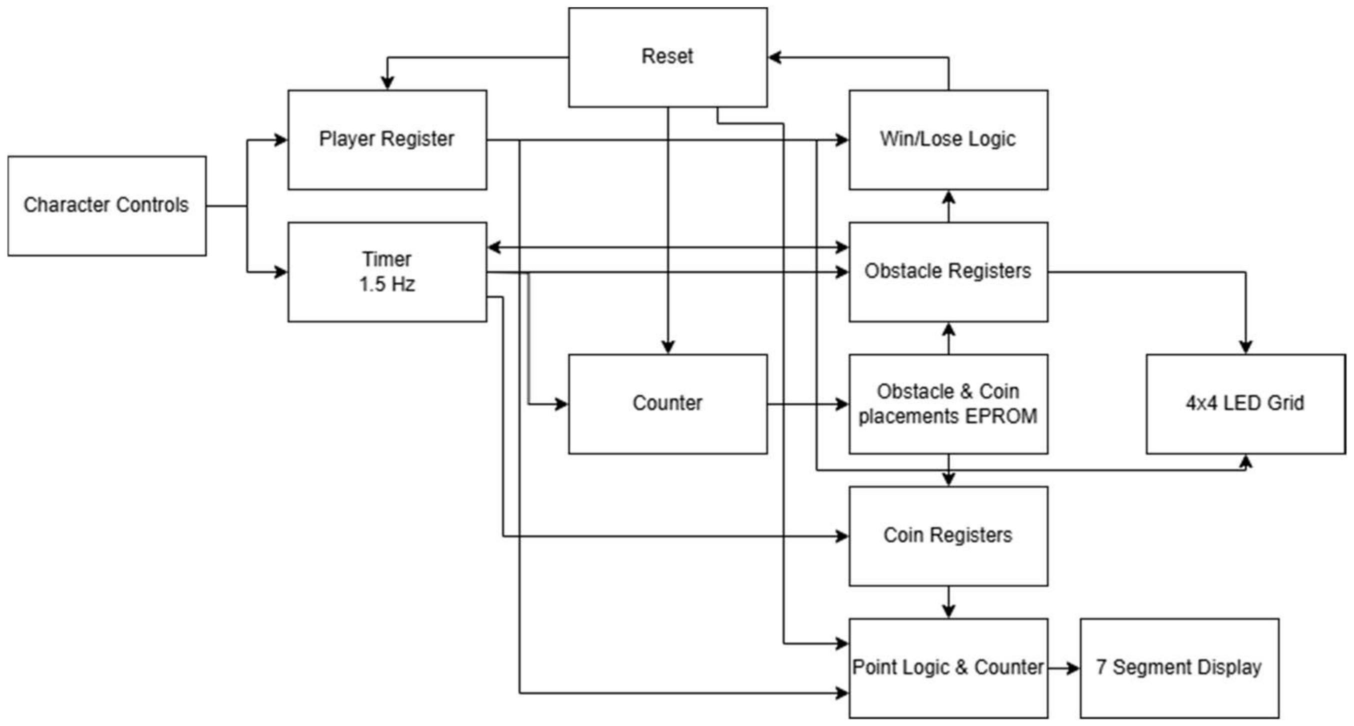
The goal of this project was the recreate the game Subway Surfers using digital logic. You move along the bottom row of a 4x4 LED display and dodge obstacles coming down at you by moving left and right and collect coins to increase your score. The obstacle and coin placements would be predetermined. The game also loops, so you could go forever if you don’t lose by colliding with an obstacle.

What the project does currently is everything above except the point system. You can move around and dodge obstacles but there are no coins or a score.

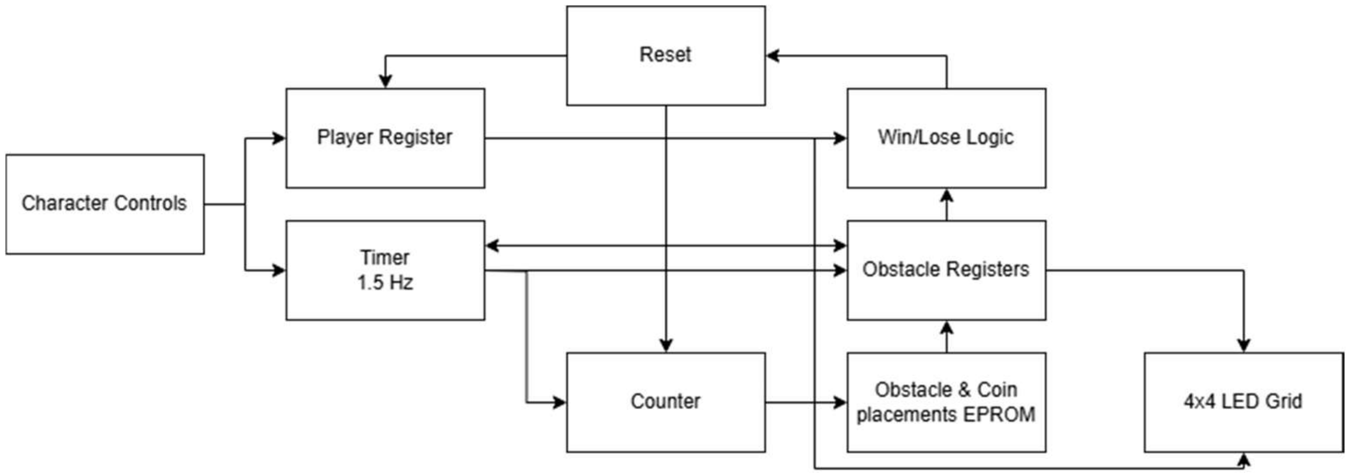
Instructions

There are 3 usable buttons for the game. The left most button is the start button. The buttons that move you left and right are the last 2 buttons from the start button. You first must click the start button and initialize the game by loading the player and starting the timer for the obstacles. Once you start you can use the left and right buttons to move around. You must be careful, as if you move to far to the left or right you get eaten by a dragon and you must reset the game by clicking the start button. The player is represented as a green LED along the bottom row and the obstacles are represented as red LEDs. So your goal is to make the green LED dodge the red LEDs. If you collide with a red LED, you will lose the game, and it will reset.

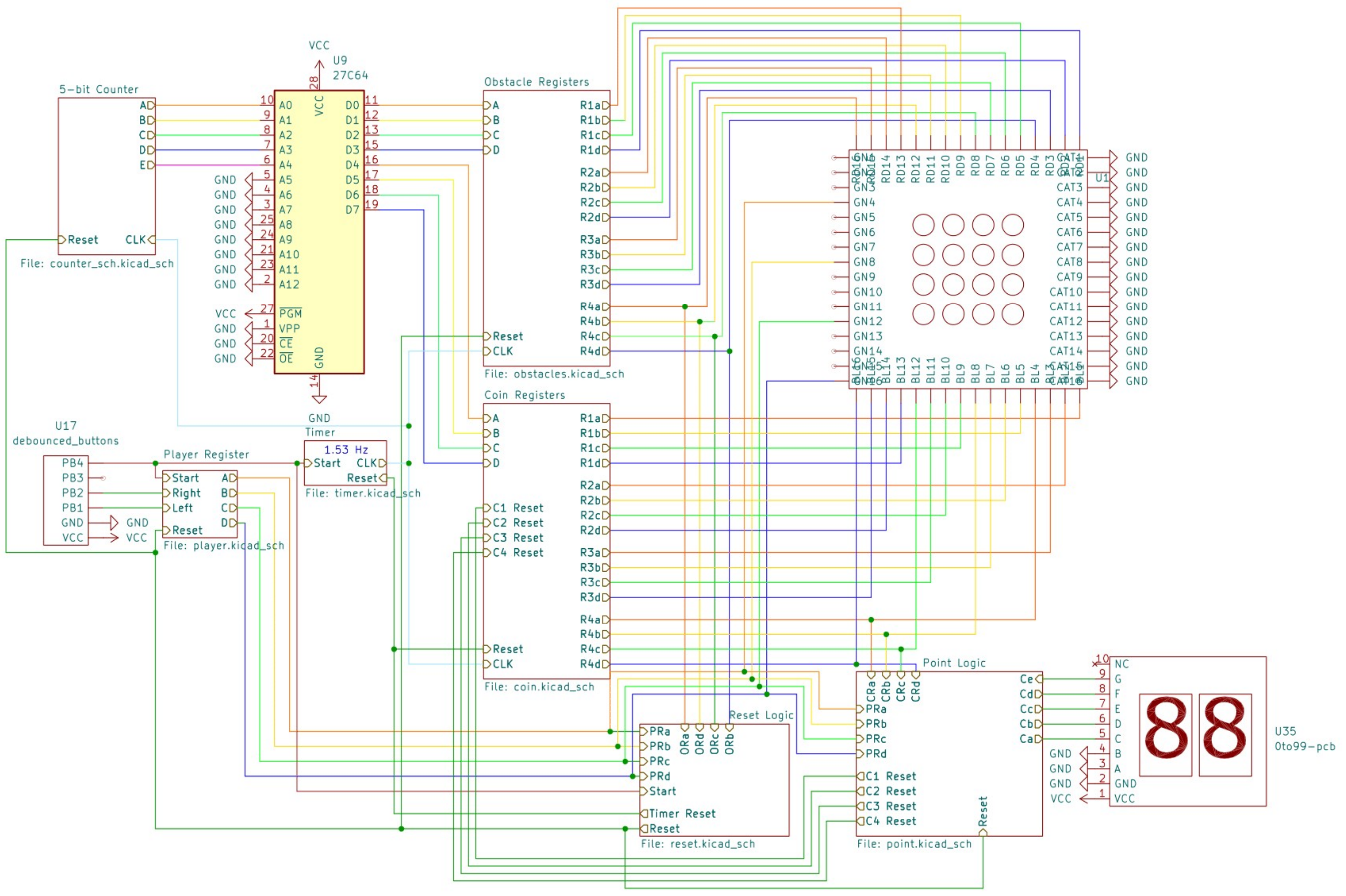
Block Diagram (Goal)



Block Diagram (Built)

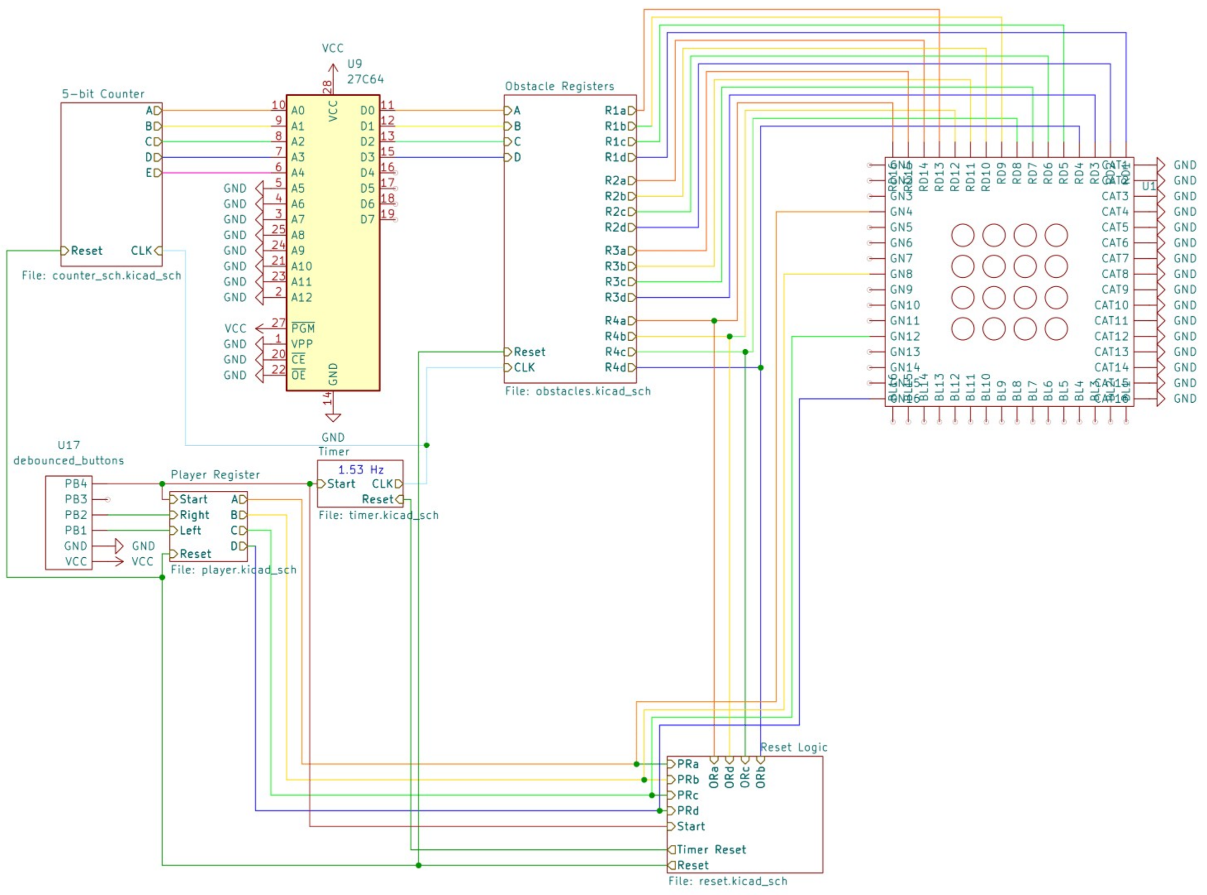


Main Schematic (Goal)

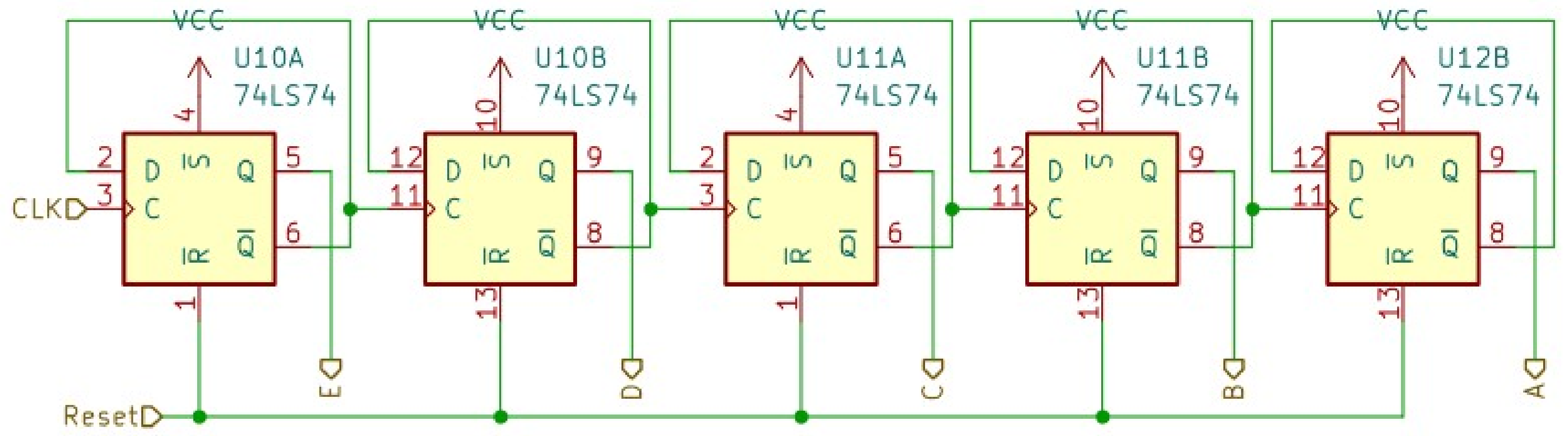


This is the main schematic of the project. More information on inputs/outputs on later slides

Main Schematic (Built)

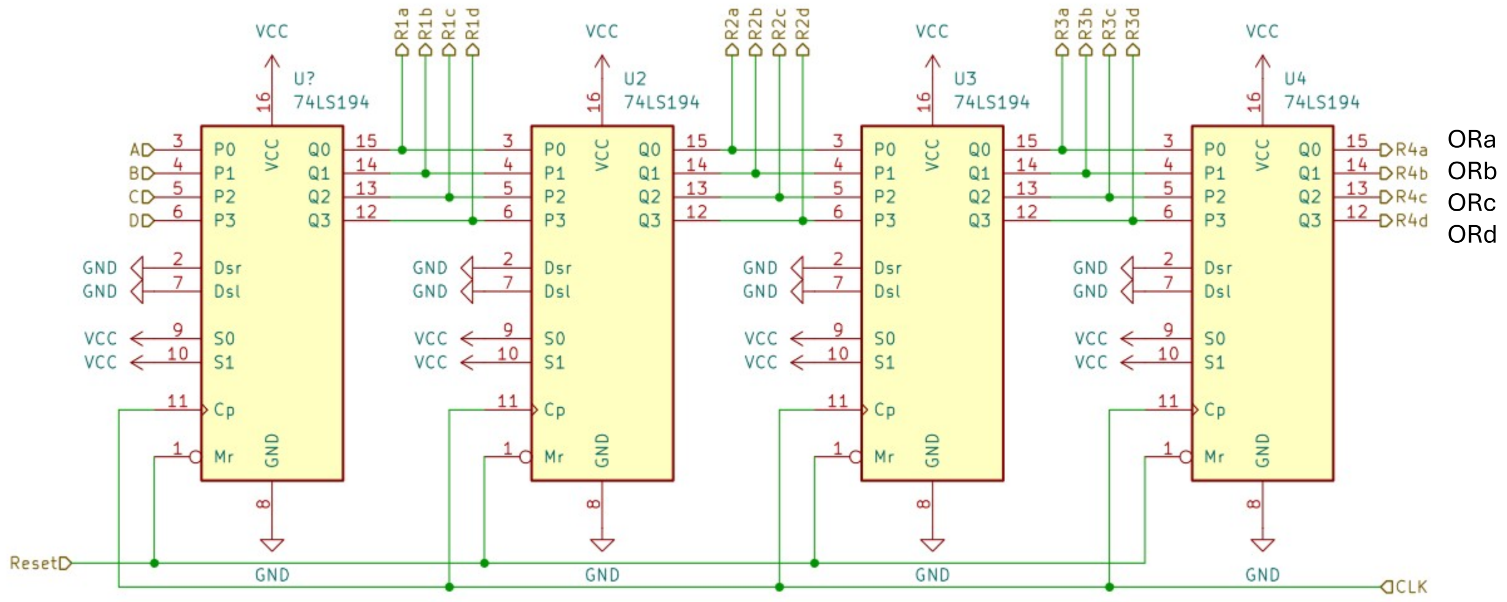


5-bit Counter



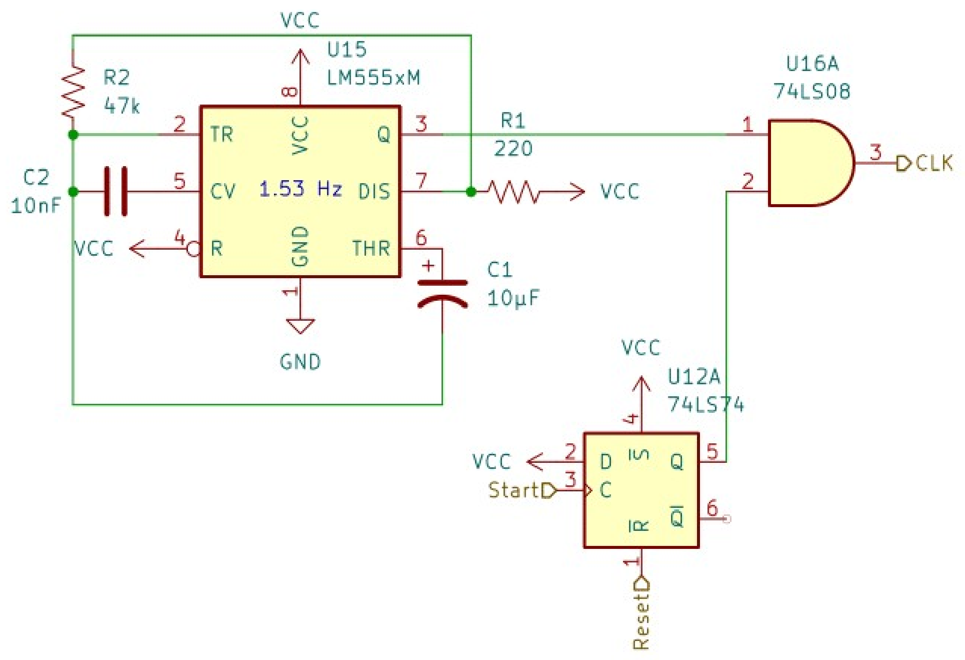
5-Bit ripple counter. Resets when either the start button is pressed or when you lose.
E is the Least Significant Bit, and A is the Most Significant Bit
The outputs of this counter control the EPROM

Obstacle Registers



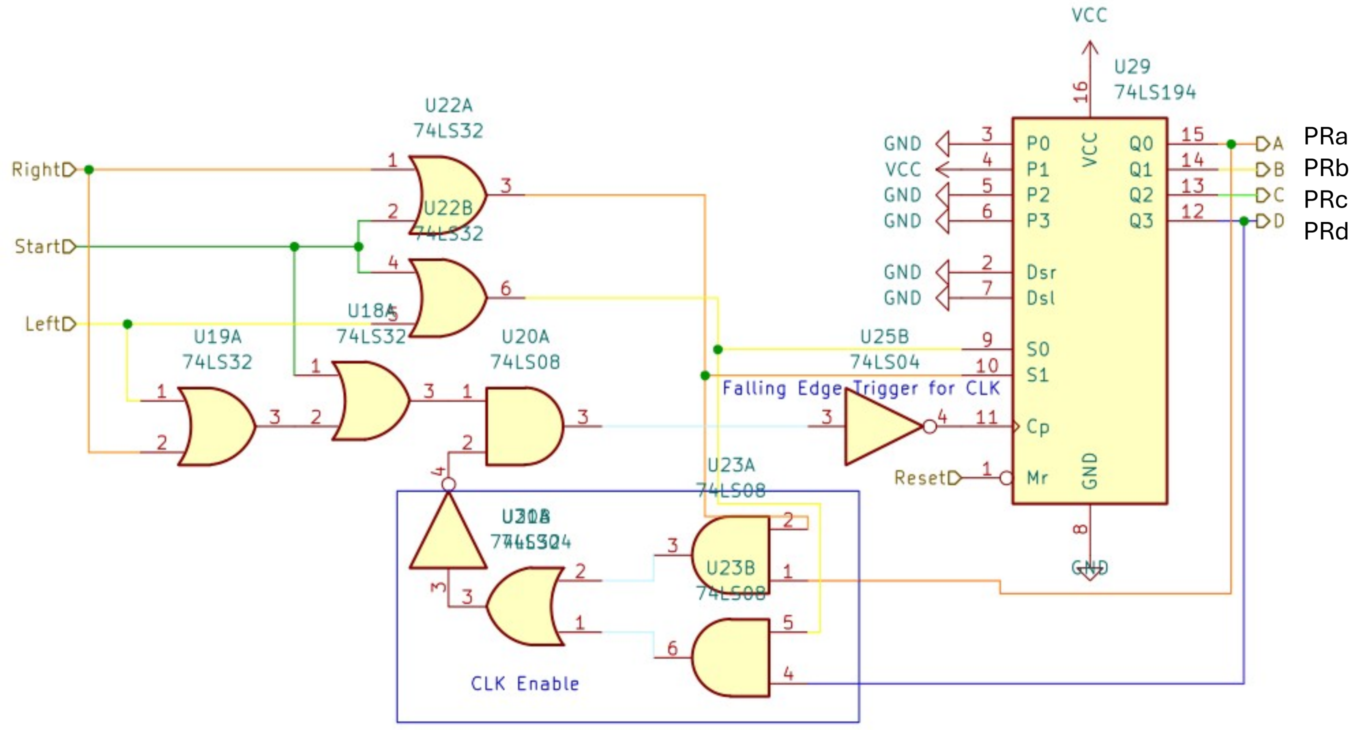
Each register will load what was in the previous one, and the first will load values from EPROM.
CLK starts at the last register because I want the last one to load first to avoid timing issues. The outputs of the last registers appear later as ORx instead of R4x.

Timer



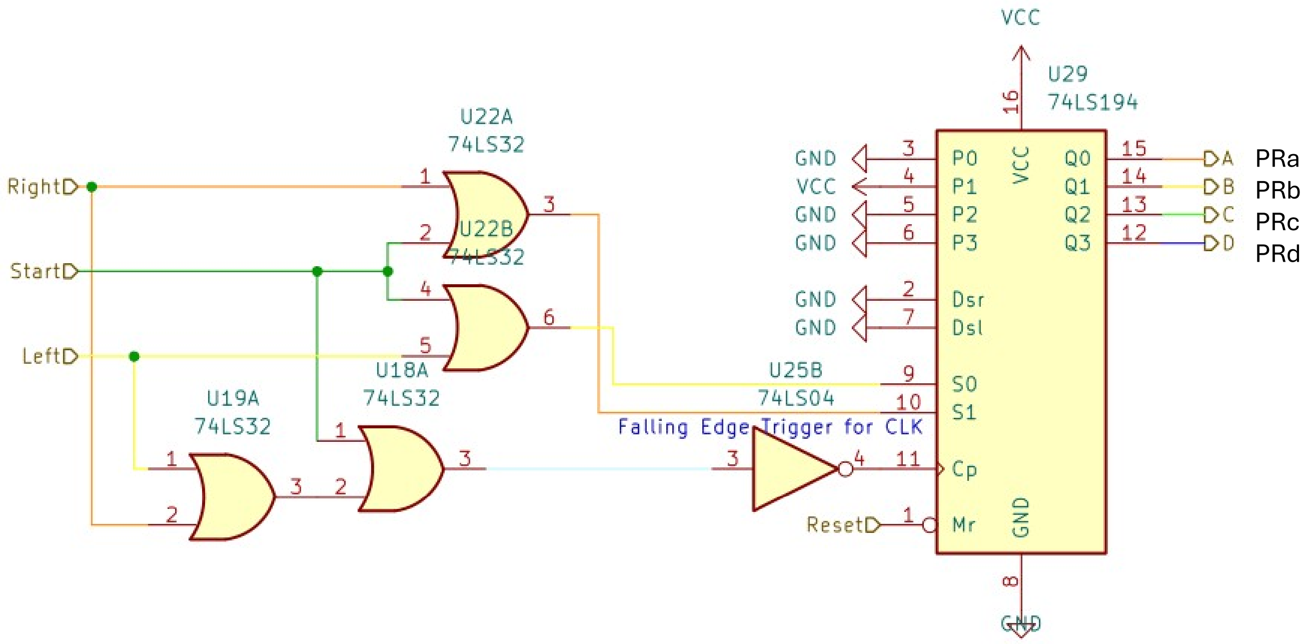
Flip-Flop inhibits the timer from affecting the game until you press start. The flip-flop resets when you lose.

Player Register (Goal)



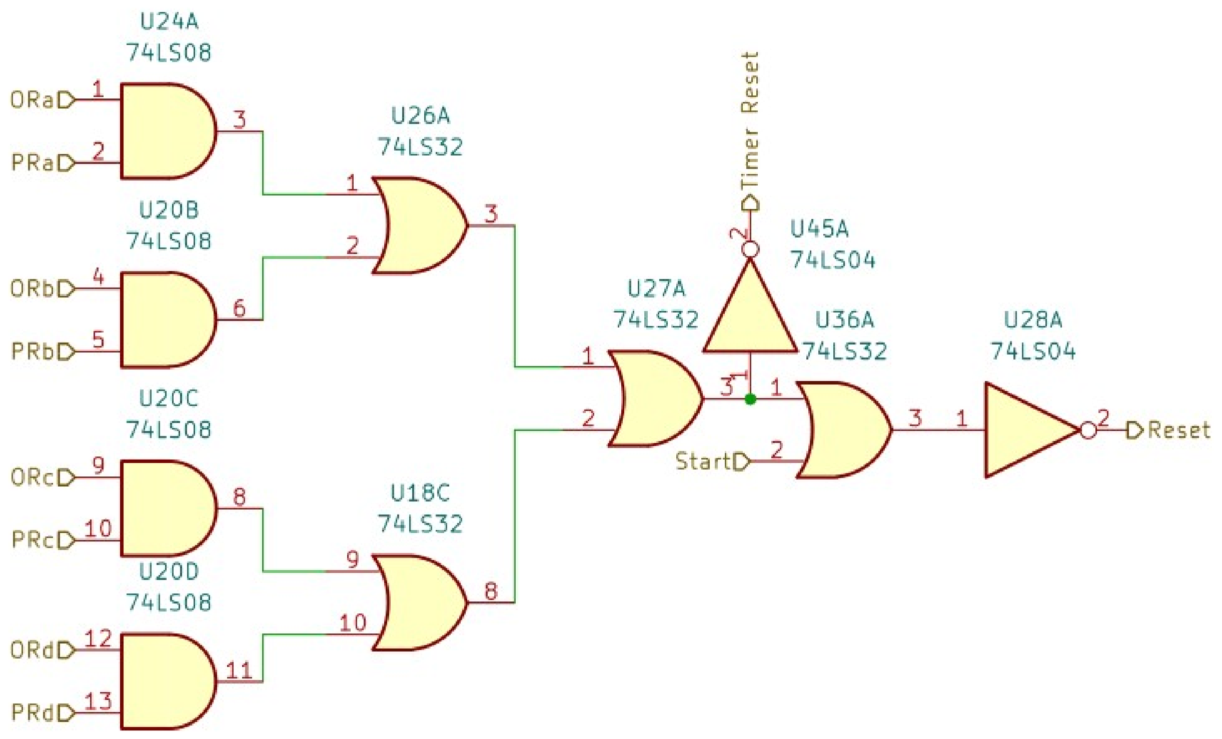
Right/Left/Start are from the control buttons. The CLK Enable allows the clock signal when there isn't risk of the player being deleted. It will inhibit the signal when S0 is one and the player is in the left most spot, or if S1 is one and the player is in the right most spot.

Player Register (Built)



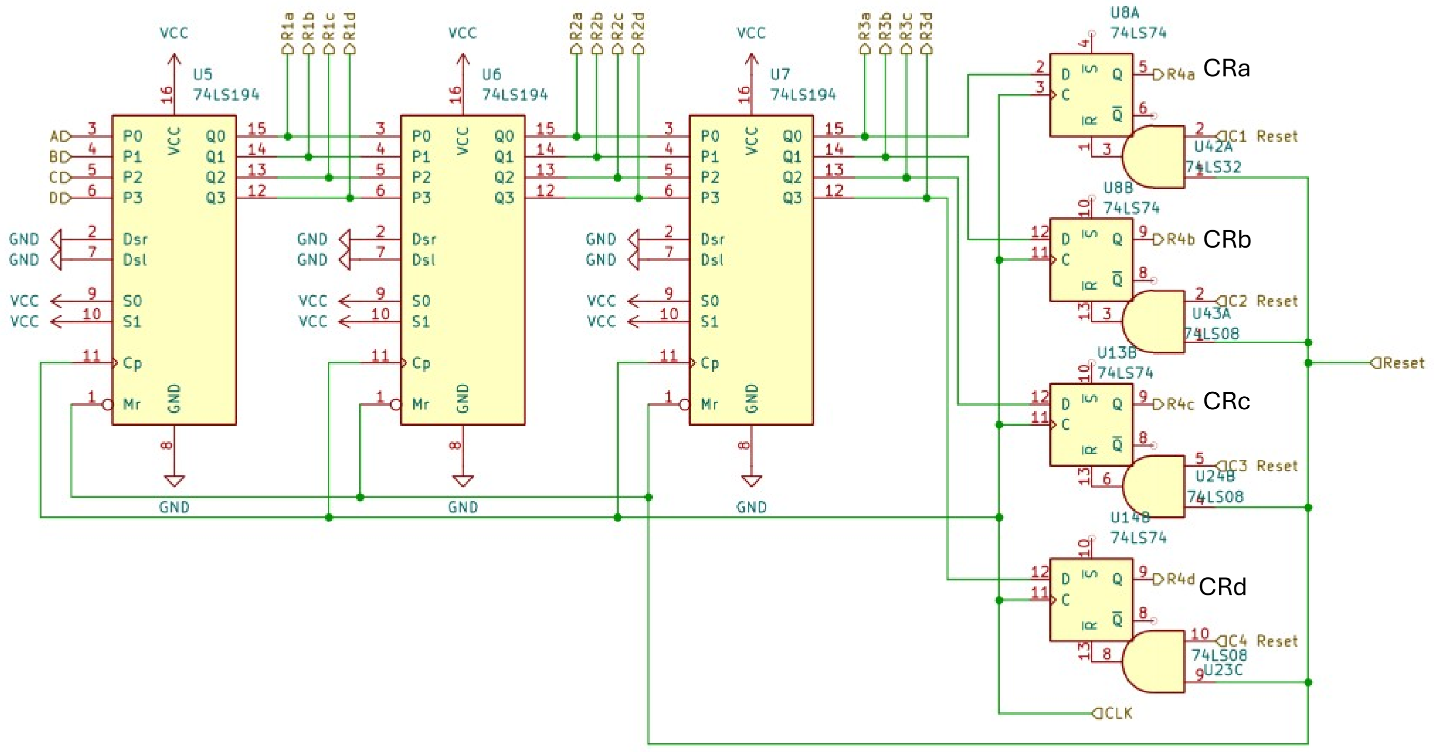
The same as before except no CLK Enable

Reset/Lose Logic



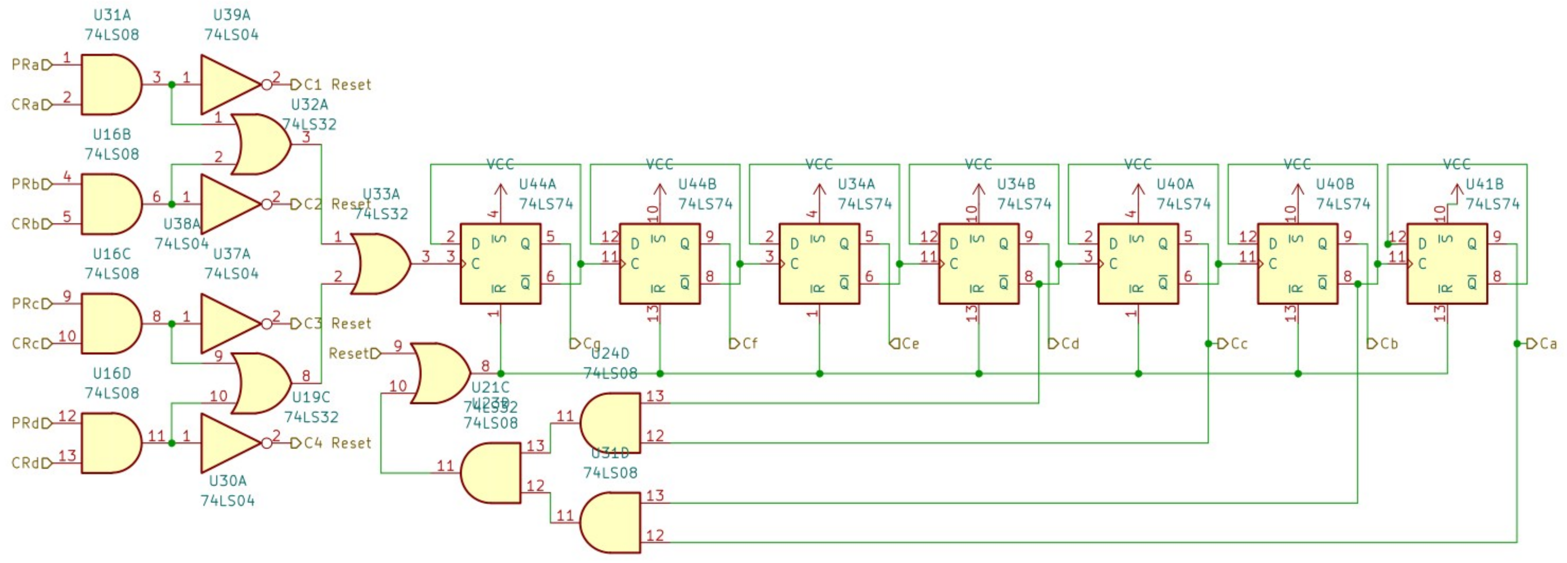
If the player and an obstacle are in the same spot it resets the timer and the rest of the components. The reason the timer reset does not include the Start button is because it would reset the flip-flop which goes to one when Start is pressed. Start will reset everything but the timer.
Both resets are inverted because all reset input pins are active low.

Coin Registers (Goal)



Same layout as the obstacle registers expect it uses flip-flops for the last register instead. The reason I did this is explained in the Design section

Point Counter (Goal)



7-bit counter will clock every time the player is in the same spot as a coin. It counts up to 99 then resets. If you get a coin, that coin flip-flop will set to 0, removing the coin from the display.

Design

For the player to move, I needed 2 buttons. One to move left and one to move right. I also needed a start button to load the register. These need to control the *S1* and *S0*. ***S1 = Right + Start*** and ***S0 = Left + Start***. The player register will clock when any of the buttons are pressed, so ***CLK = CLKen(Right + Left + Start)***. An issue that I wanted to avoid was that if you shifted to far left/right the player would be deleted, and you would have to restart. I considered looping because it was simpler to design, as you just connect *Qa* with *Serial Left* and *Qd* with *Serial Right*. However, I felt that it was not fitting as I don't want the player teleporting from one side to the other. So, I decided to try to stop the player from moving to far left/right by inhibiting the clock signal and adding a clock enable. If you press the right button while the player is in the right most spot or if you press the left button while the player is in the left most spot, it will inhibit the clock signal. ***CLKen = (QaS1 + QdS0)'***.

Design

The obstacles are created and displayed using a timer, a 5-bit ripple counter, an EPROM, and a series of 4 shift registers. The timer has a frequency of about 1.5 Hz. To inhibit the timer, I used a flip-flop that sets when you press the start button. The timer and the output of the flip-flop go through an AND gate, and that is the CLK signal for the counter and registers. The 5-bit ripple counter represents the stage that's going to be loaded into the top register of the obstacles. The outputs of the counter go into an EPROM which converts it into the placements of the obstacles and coins. The programmed hex is on the next slide. The most significant digit of the hex is the coin placements and the least significant is the obstacle placements.

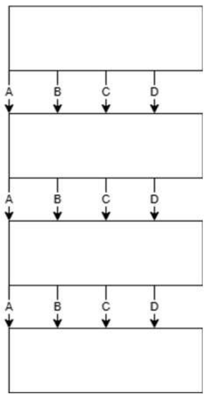
Design

Counter Value	Coin Placements	Obstacles Placements	Hex
00000	0000	1001	09
00001	0000	1011	0B
00010	0000	0001	01
00011	1010	0100	A4
00100	1000	0100	84
00101	0001	0110	16
00110	0000	0010	02
00111	0010	1000	28
01000	0000	0011	03
01001	1000	0111	87
01010	0000	0010	02
01011	0100	1010	4A
01100	0000	1001	09
01101	0010	1101	2D
01110	0000	1001	09
01111	0100	1011	4B

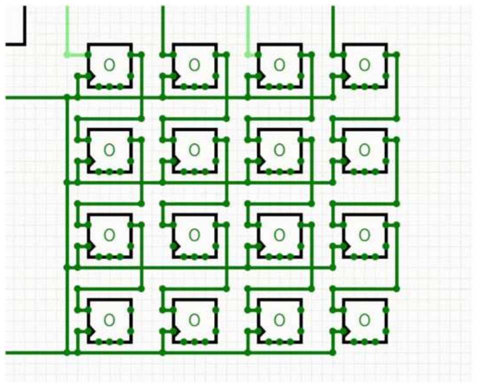
Counter Value	Coin Placements	Obstacles Placements	Hex
10000	0000	0000	00
10001	1010	0101	A5
10010	0010	0101	25
10011	1000	0000	80
10100	0001	1010	1A
10101	0101	1010	5A
10110	0000	1100	0C
10111	0010	0101	25
11000	0010	0101	25
11001	1000	0001	81
11010	0010	0100	24
11011	0001	0110	16
11100	0000	0111	07
11101	1101	0000	D0
11110	0000	0111	07
11111	1000	0111	87

Design

The outputs of the EPROM get loaded into 2 different series of 4 shift registers. One for the coins and one for the Obstacles. Every time the timer cycles, the data moves down to the next, and the new placements are loaded into its place. To determine if the player lost, the outputs of the final row of obstacle registers and the player register is compared. ***Lose = PRaORa + PRbORb + PRcORc + PRdORd***. *PRx = Player Regster, ORx = Obstacle Register*



Once you lose the timer will be inhibited once again by clearing the flip-flop by the timer. ***Timer Reset = Lose***. The rest of the components all reset when ***Reset = Lose + Start***. So, losing or pressing the start button resets the player, counter, and obstacles.



Design

The coin system works similar to the obstacle system, but instead the last register is replaced with 4 flip-flops. The reason I did this is because on the 74194 register, you can only clear the entire row, now just 1 spot. What I wanted to happen is that when you pick up a coin, it gets removed from the display, and your points go up. To determine if the player gets a point, the outputs of the final coin register is compared to the player register. ***Point = PRaCRa + PRbCRa + PRcCRc + PRdCRd***. To clear each coin, you just compare the coin flip flop and the corresponding player bit. ***C1 Reset = (PRaCRa)’***, ***C2 Reset = (PRbCRb)’***, ***C3 Reset = (PRcCRc)’***, and ***C4 Reset = (PRdCRd)’***. Since the reset pins on the flip-flops are active low, they need to be inverted. *Point* clocks a 5-bit counter which counts how many points you have. This displays on two 7 segment displays. I chose a 7-bit counter because the game loops so theoretically the amount you could get would be infinite, and since the two 7-segents go to 99, I made the counter reset at 99. ***Counter Reset = Reset + (CaCb’CcCd’)***

Design Tools

I used CircuitVerse to experiment with and test different components of the project. KiCad was used for making the schematic of the circuit, which I used to assist in building the project on the board. I used the digikey 555 timer calculator to get the values for the resisters and capacitors for my timer.

Complete vs. Incomplete work

What I was able to complete was the having a functional obstacle system, the obstacles get placed and move down. The player is also mostly functional, you can move left and right, but one issue is that if you move to far left or right you will delete yourself and will have to press the start button again to reset. I also wasn’t able to build the point/coin system of the game. So, what’s built is the Obstacle and player systems and their interactions. I really don’t know why the CLK Enable wasn’t working. I’ve built it before on a portfolio circuit and it worked just fine. Using the logic probe I was getting the expecting behavior on the S0, S1, and CLK pins on the register. The only issue I can think of is a timing issue I couldn’t see using the logic probe. If I had more time I would just rebuild the player register and the CLK Enable logic from scratch to potentially fix it. I would also build the coin/point system.