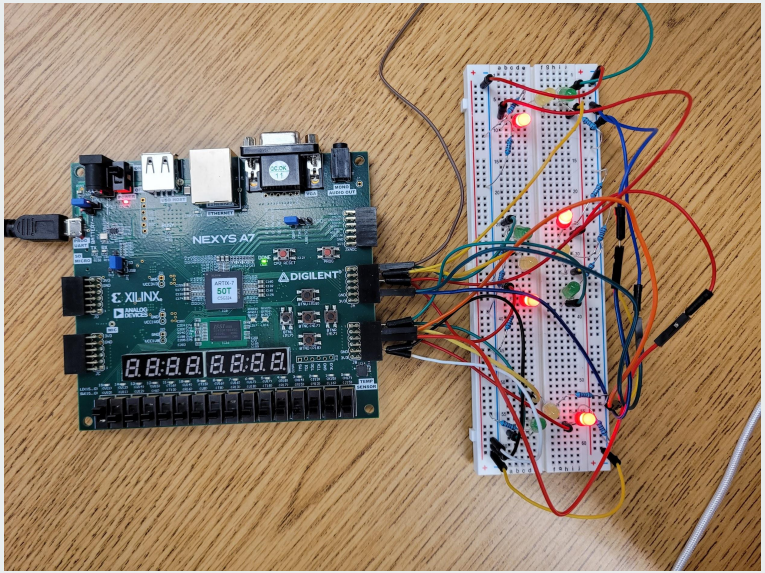
Four-Way Traffic Control:

ECE 2700 Final Project

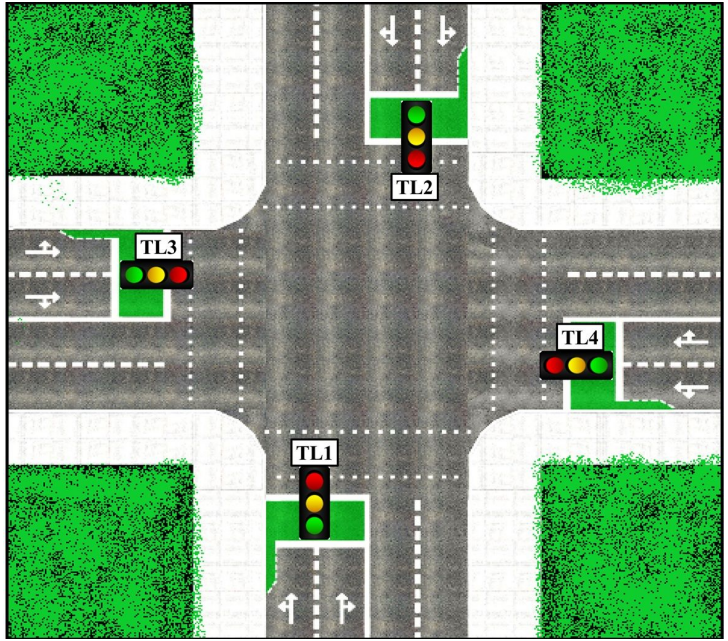
Isabella Bodmer: ibodmer@oakland.edu
Solana Brown: solanabrown@oakland.edu
Brikena Dulaj: brikenadulaj@oakland.edu
Dalena Vu: dvu@oakland.edu



Intro and Overview

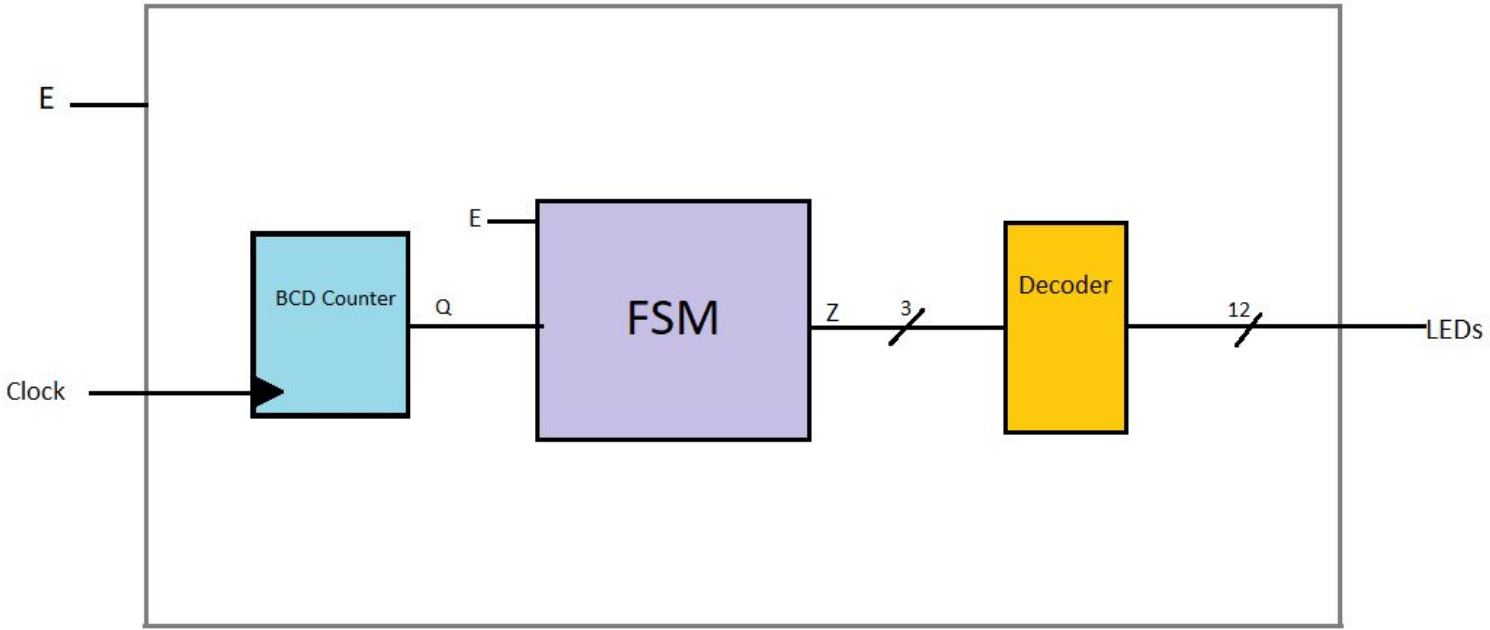
For this project we decided to replicate a four-way traffic control system by using both VHDL and our FPGA wired to a breadboard with LEDs in place of the traffic lights. Our project's intended way to work is to replicate how a real-life four-way traffic light would work. Our states include each cardinal direction, North, East, South, and West.

Our circuit is off until a switch representing enable is flipped. It starts with all light signals being red, while North and South states coincide with each other and West and East states coincide. We set up our circuit to run using a counter, finite state machine (FSM), and a decoder. The counter slows down the FPGAs internal clock to ensure our lights are left on for a reasonable amount of time, not just flashing. The FSM is used to determine which state the circuit is at and is then fed into a 3 to 12 decoder, which interprets the state and outputs the corresponding LED sequence deciding what LEDs are on and what color.

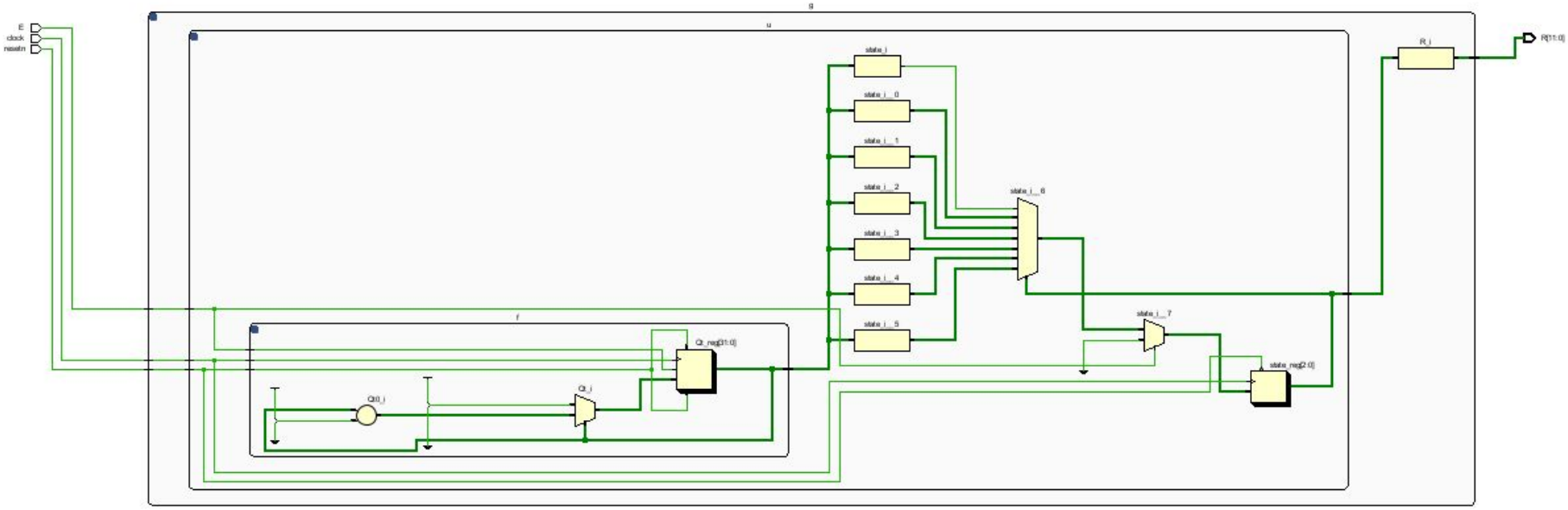


https://th.bing.com/th/id/R94cabd3e003bee423a6c5e31d35298de?rik=%2bGYgdI5eCjzJtg&riu=http%3a%2f%2f2.bp.blogspot.com%2f-L4BqbTWK84c%2fUP1I3YWUv6I%2fAAAAAAAAAOg%2fvFp3XFax_Uk%2fs1600%2f4-way%2bJunction.jpg&ehk=VINW2vNyuuNm3kFsx4qGWqPG0b0kzUg4TxOMuYqDN4A%3d&risl=&pid=ImgRaw

Block Diagram

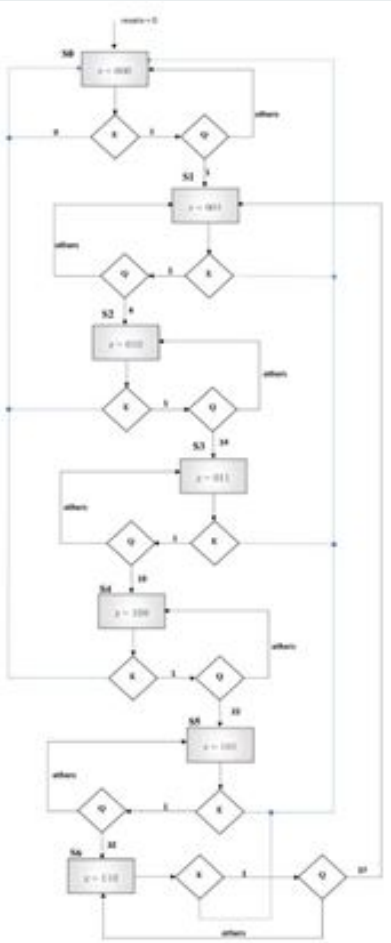


Circuit Description Diagram



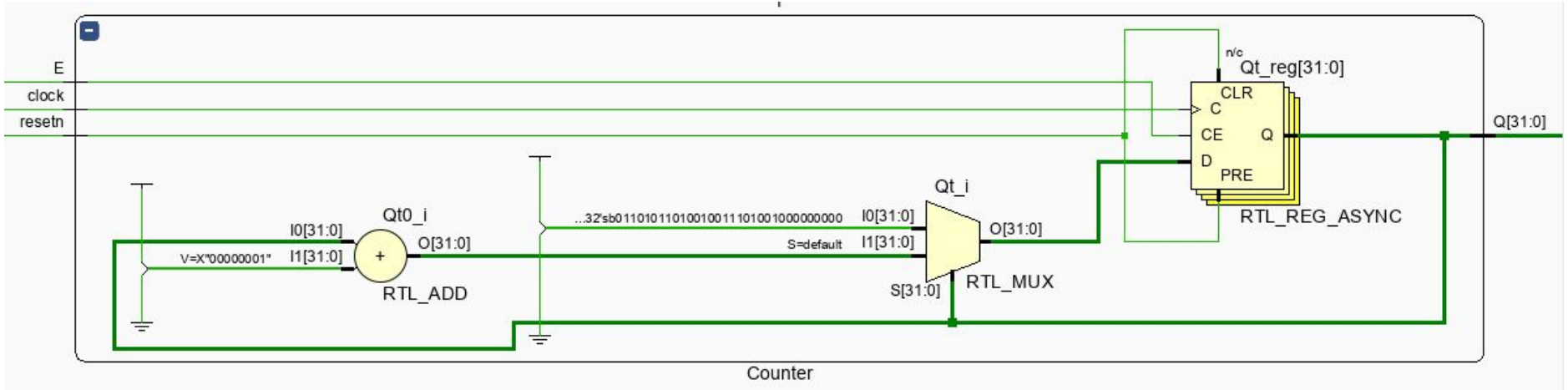
Algorithmic State Machine

The ASM describes the logic behind the finite state machine used for this project. The input to start the states cycle is resetn = ‘0’ which initializes the first state S0 that’s the moment when all the lights are off. Then an enabler is used to move the current state into another. Every state waits for the input from the counter to reach it value and then once that condition is met it moves to the next state where the same conditions apply. In the last state the process cycles back to state 1 unless the enabler is off where the default state is S0 where all the lights are off.



Counter

The counter intends to slow down the internal clock signal from nanoseconds to seconds. The clock signal will be fed into the counter first, and the counter's output is a slower clock signal that will be fed into the FSM, and then from that point used as that standard measurement of time. We decided to initially use a six-bit BCD counter but then increased the number of bits to 31 according to what the output would be in seconds. Since the default clock is 100MHz the default time of this counter would be nanoseconds which was too fast for us to see a difference. The output Q from the counter then goes into the FSM to signal the duration that the current state should last. While Q could have been left in BCD we agreed that converting it to integer using the formula (2^n)-1 would be easier to implement



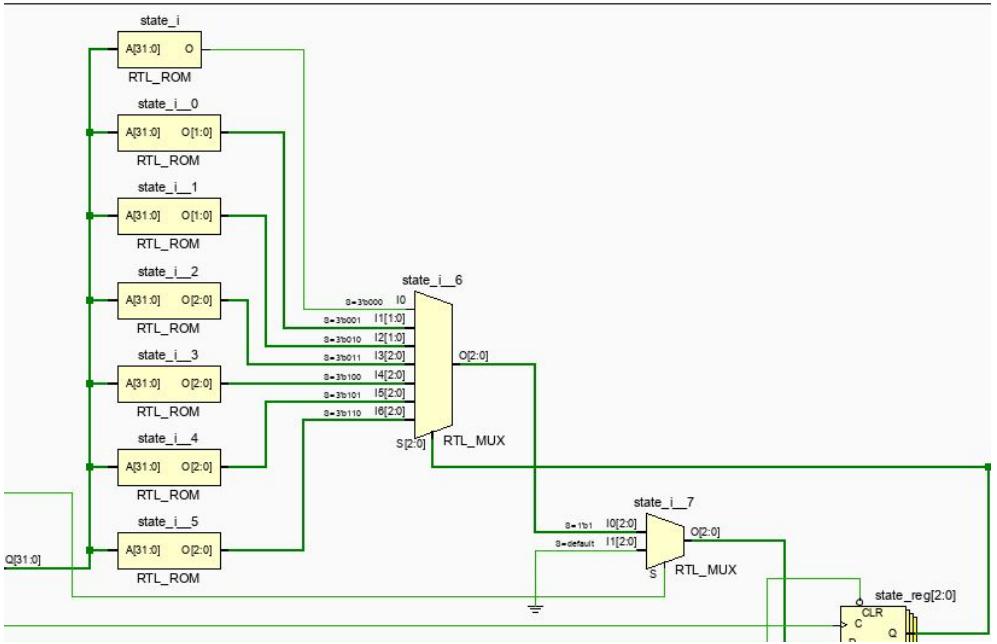
State Table

To describe our states we made a table showing the time duration, the name of the state, and which lights are on in that state. S0 is when enable is 0 and all lights are off. For the rest of the states enable =1. S1 is when all lights are red for 3 seconds. S2 is when north and south are green while east and west are red for 10s. S3 has north and south yellow and east and west red for 5s. For S4 all lights are red for 3 seconds again and after that the states repeat the process but this time for east and west.

En			N	E	S	W
0		S0				
1	3s	S1	R	R	R	R
1	10s	S2	G	R	G	R
1	5s	S3	Y	R	Y	R
1	3s	S4	R	R	R	R
1	10s	S5	R	G	R	G
1	5s	S6	R	Y	R	Y

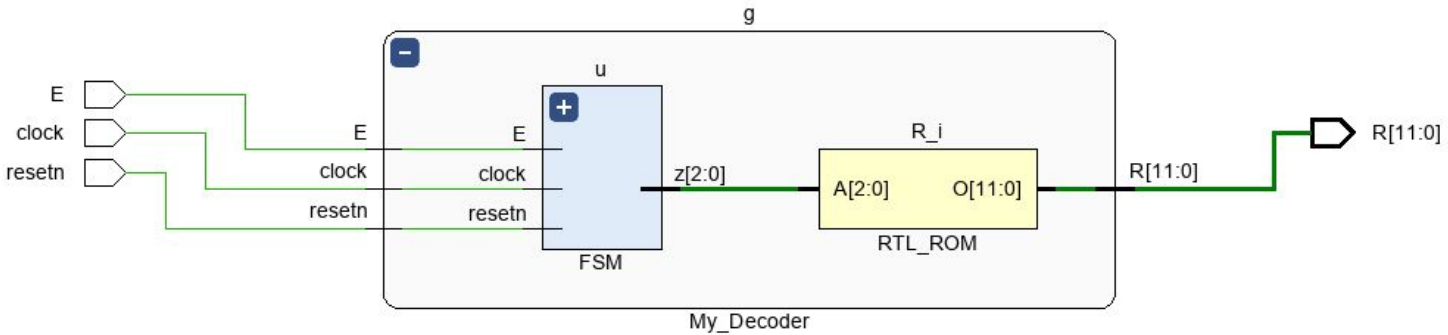
FSM

The timing and settings for the lights are based on which state the device is in, making the FSM the central part of our design. For our FSM, we had all agreed on having seven states. It uses a switch representing enable as its input. If the E=0, all lights are off, and it's S0. If the switch is on, the circuit will progress through the states cycling from S1 to S6 and back to S1.



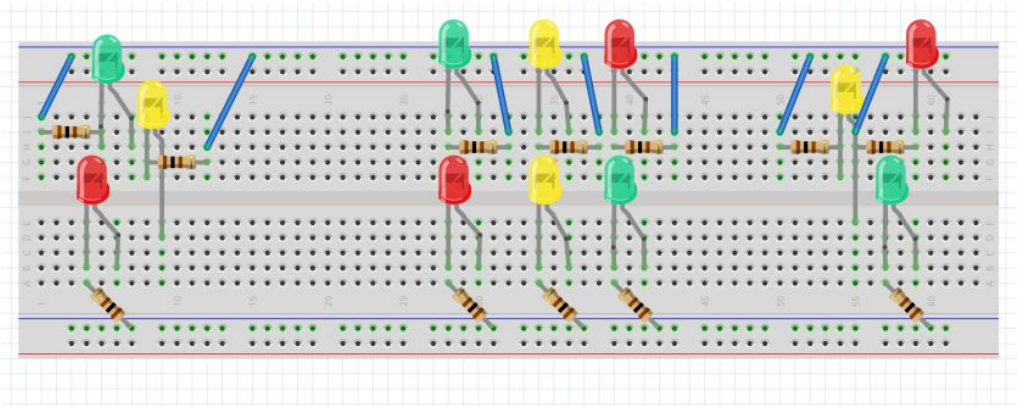
Decoder

For our decoder we decided to use a 3 to 12 decoder. That way when the FSM outputs the three-bit value corresponding to the state it's in, the value would then feed into the decoder causing it to output the corresponding number and the correct twelve-bit value can then be sent to the LEDs.

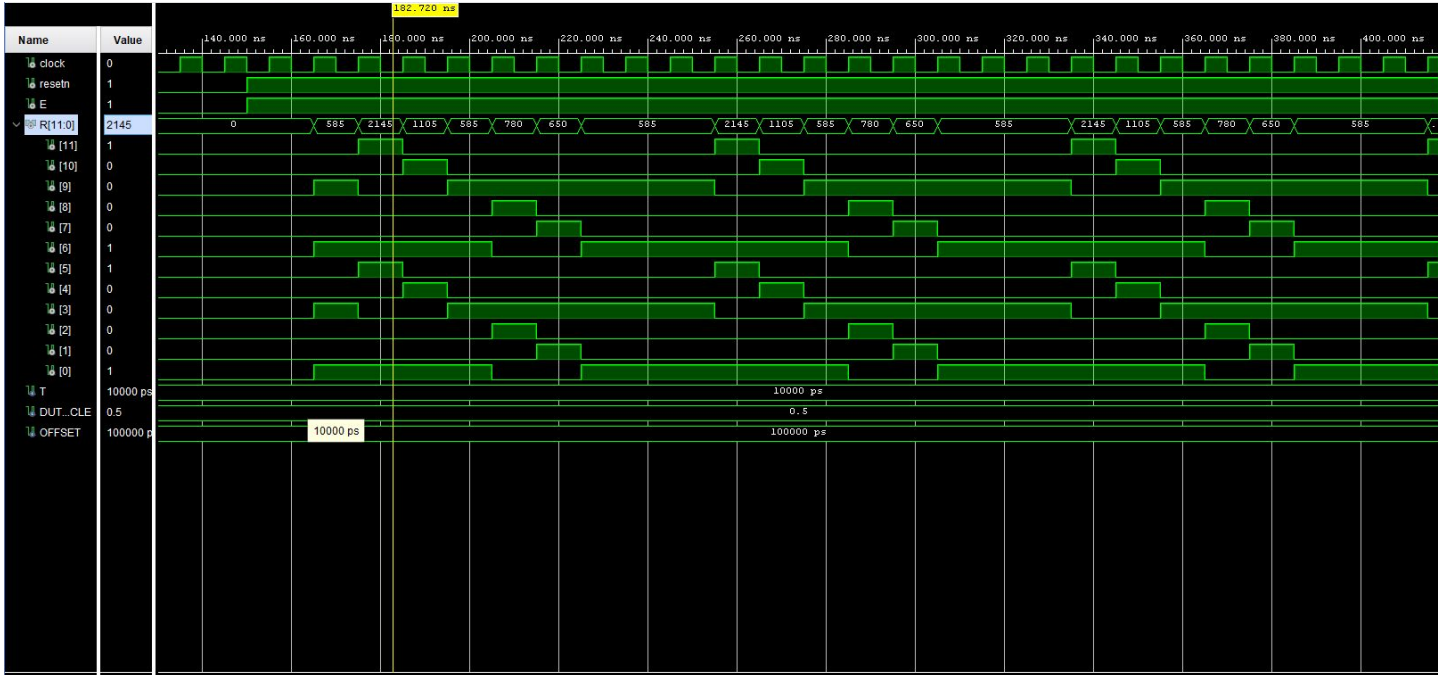


Breadboard Setup

After looking at the manual for the Nexys A7, we decided to use the JA and JB pins to interface to the breadboard. The top half of the board was grounded through the JA ground port, and the lights on the top half were then wired to JA pins 1-4 and 7-8. Meanwhile, the bottom half of the board was grounded through the JB ground port, and the lights on the bottom half were wired to JB pins 1-4 and 7-8. Each LED is wired to a ballast resistor to limit the current running through it. After taking the LEDs recommended current and our boards output voltage into account it we decided to use 100 ohm resistors to accomplish.



Simulation Results (simulated in ns for better view)



Demo

https://linksharing.samsungcloud.com/qELnwCAHSxzO

Board is activated by the flipping of the enable switch (switch 15)