Project 2: Traffic Light

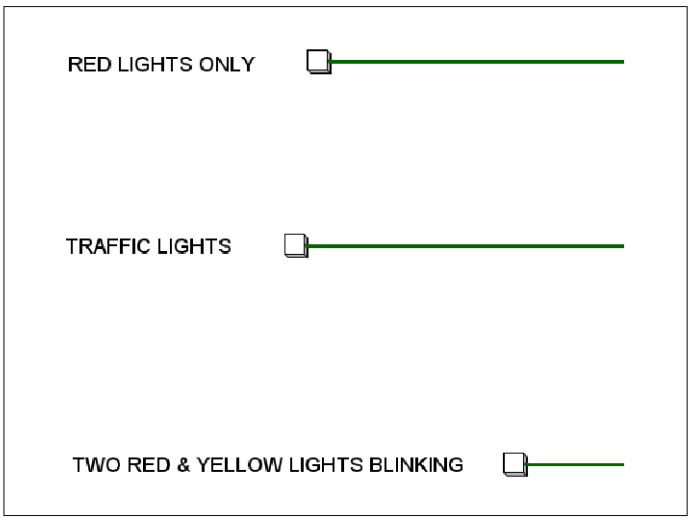
Ahmad Malik
The Cooper Union
Digital Logic Design - ECE150
Professor M. Billoo

Introduction and Project Description

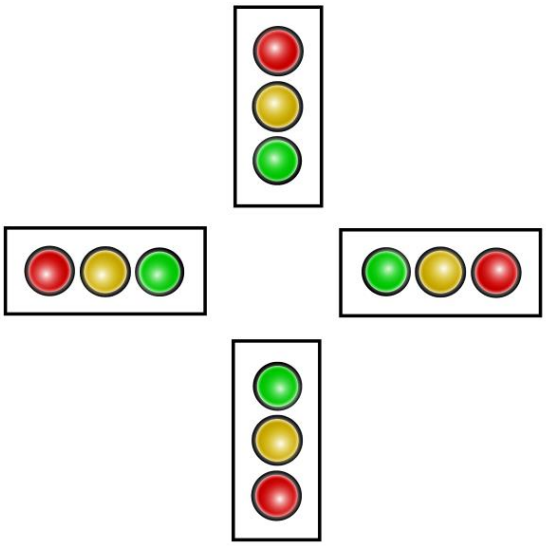
The purpose of this project was to design a circuit using logic gates to mimic the signal output of actual traffic lights, more specifically, a four-way intersection controlled by four traffic lights. Additionally, while the circuit operates as a traffic light, the project description required that two sets of traffic lights blink yellow and the other two sets blink red upon the push of a button; if the same button is pushed again, the circuit should resume normal traffic light operation. When the circuit is first powered, all four traffic lights should be red and there should be a corresponding button to return to this state at anytime.

All inputs are to be in the form of push buttons and not dip switches; the difference being that instead of a constant HIGH state transmitted by a dip switch, the push button only transmits a very short HIGH pulse. The outputs of this project will be shown through LEDs where red, yellow, and green LEDs correspond to the actual color outputs of a real traffic light.

During typical traffic light operation, the green light should be powered for ninety seconds, followed by a transition to the yellow light for three seconds. Once the three seconds are over, the traffic light should transition to red, while the adjacent traffic light should transition from red to green, and the cycle repeats. Thus, during typical traffic light operation, the red lights will always function for ninety-three seconds, the green for ninety seconds, and the yellows for three. If the button to activate the blinking lights is toggled, opposing pairs of traffic light will blink either yellow or red at one second intervals.



INPUT PUSH-BUTTONS



OUTPUT LED ARRANGEMENT

Theory

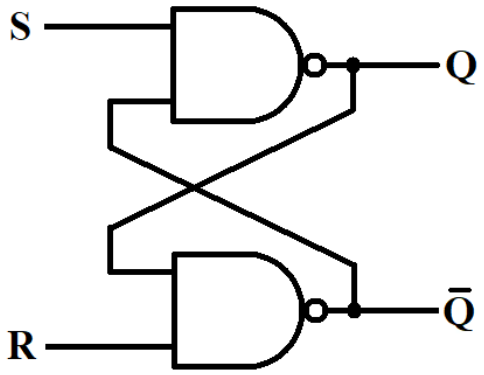
Preliminary Considerations

Because this project revolves around timing and repeated signal patterns, sequential logic is what will make up the majority of the circuit. The use of sequential logic such as Flip Flops and SR-Latches will allow the circuit to maintain a memory of the signal pattern it needs execute, and timers will enable the circuit to operate at a consistent pace.

Although it is possible to make our own sequential logic using basic gates (AND, OR, NAND, XOR, etc), we are provided with D-Flip Flops which saves the need to use more gates for the construction of individual Flip Flops. Naturally, it was in our best interest to simplify and reduce the number of gates we used. Lowering the number of gates means less time spent wiring, less supplies used, and greater energy efficiency. Thus, D-Flip Flops will be the extensively used in this project because they are simple, robust, and can serve the purpose of memory state storage. For timing, we are provided with 555 timers which will serve as the clock trigger for the sequential logic to operate under. The timer can be set to a specific frequency based on calculated capacitor and resistor values, of which we have a decent variety to choose from.

Logic Gates and Memory

Simple logic gates on their own cannot retain a stable state or "memory" because they require a constant input to maintain their output signal. With the introduction of sequential logic, multiple gates can be arranged so that an instantaneous input will be remembered. This is because sequential logic can change from one state to another if the input triggers it to a state which is equally stable as the one it was originally in. A primary example of this is the SR-Latch shown below. When both inputs are LOW, the output remains unchanged or in the "memory" state because the circuit remembers and maintains its previous HIGH or LOW state. When the SET input is instantaneously HIGH and RESET is kept LOW, the circuit assumes a new stable state in which Q is always HIGH. When the RESET input is instantaneously HIGH and SET is LOW, Q will always be LOW. If both inputs are triggered instantaneously, the output will depend on the last input that was HIGH; therefore, Q cannot be determined.



SR-Latch

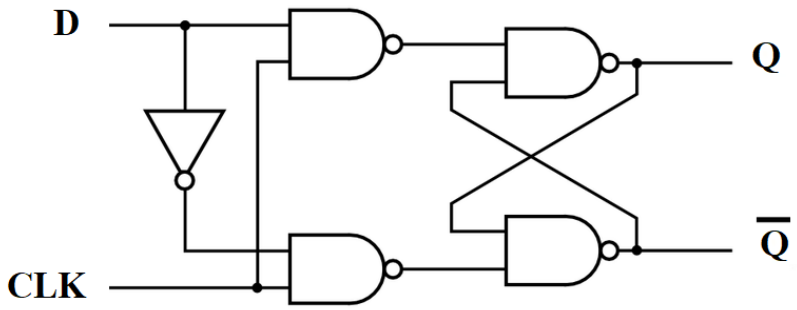
S	R	Q	State
0	0	NC	MEMORY
1	0	1	SET
0	1	0	RESET
1	1	N/A	UNKNOWN

Truth Table

D-Flip Flop

Building on the idea of an SR-Latch, the D-type Flip Flop is designed so that instead of acting on the High/Low state of the inputs, it only changes the output value based on the rising edge pulse of one of the inputs, CLK. The other input, D, also controls the value of Q because if the value of D is HIGH and CLK transmits a rising edge, Q will follow D's state and become HIGH. If D was in the LOW state when the rising edge of CLK was transmitted, Q would also become LOW. If the CLK pulse is anything but a rising edge, the flip flop won't change the state of Q, thus staying remaining in the "memory" state.

D- FLIP FLOP



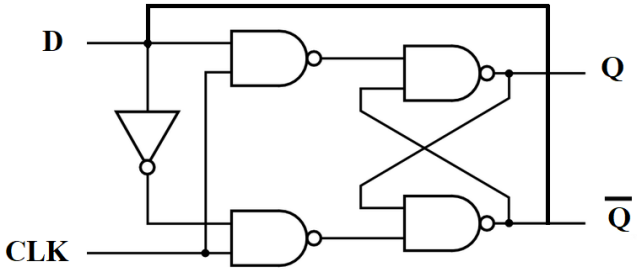
Truth Table

D	CLK	Q	State
0	↓	Q_{PREV}	MEMORY
1	↓	Q_{PREV}	MEMORY
0	↑	0	LOW
1	↑	1	HIGH

The D-Flip Flop incorporates the SR-latch discussed earlier, but the added NAND gates tie the inputs together. The NOT gate in combination with one of the NAND gates creates the rising edge detection needed for the functioning of this flip flop.

A Simple Trick

If the $\overline{Q}$ output of the D-Flip Flop is directly wired to the D input, the D input will be constantly changing states as the rising edge of CLK is transmitted. As a result, the Flip Flop changes the state of Q simply by the CLK input, and since it only changes states when a rising edge is detected, the period of Q's oscillations will be twice that of CLK's. This property comes in handy when using D- Flip Flops in creating asynchronous counters.



Modified D-Flip Flop

D	CLK	Q	State
0	0	1	HIGH
1	0	0	LOW
0	1	1	HIGH
1	1	0	LOW

Truth Table

Design

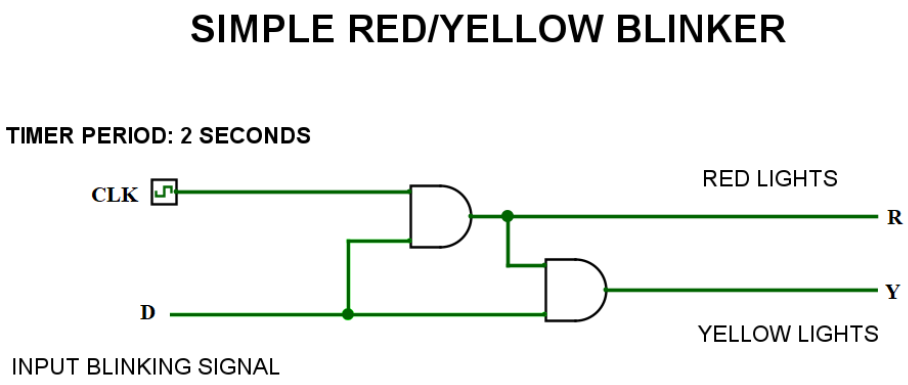
Timers

For traffic light to work there must be a reoccurring pattern with precisely timed signal outputs. This means that there must be a timer that keeps everything running. We are provided with 555 timers which can be used as astable timers to generate square waves whose frequency is based on the value of the capacitor and two resistors that are wired to it. Before deciding the value of the resistors and capacitors, I first need to decide on how many timers I need and what frequency they should operate at.

During traffic light operation, the red light should be on for 93 seconds, the green light for 90 seconds, and the yellow for 3 seconds. For the blinking state, the red and yellow light should turn on and off for 1 second. Having separate timers for each signal seems like the obvious and easy choice, but it becomes difficult to sync all the timers and keep them in order due to factors like propagation delays and timer error. Therefore I decided that the frequency would be transmitted from a single timer, at least for the traffic light operation; the timer for blinking light function could be isolated and used only when blinking function is called. The maximum period for the traffic light circuit would be 6 seconds, and maximum period for the blinking circuit would be 2 seconds.

Blinking Light Circuit

Since the lights for the blinking circuit blink at the same rate as the clock with a period of two seconds, their circuit can be easily constructed with two AND gates and a timer as shown below.



D	CLK	R	Y
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

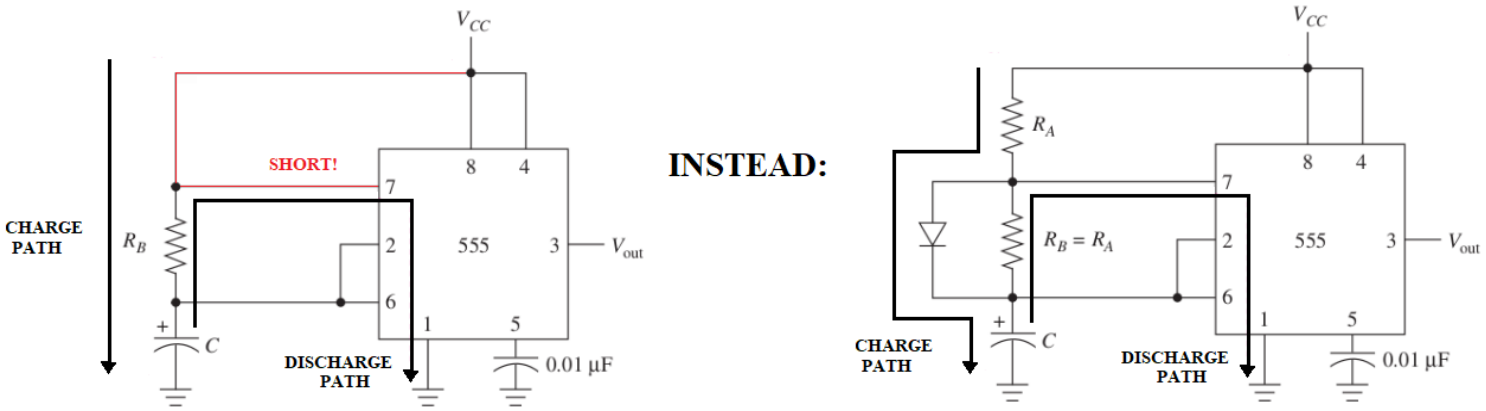
For the timer itself, we would need to calculate the resistor capacitor values to get the required period of two seconds. The formula below from the data sheet of the 555 timer shows how to calculate the value of the two resistors, given a capacitor value of 10µF (the capacitor we were supplied). The period is split between the HIGH and LOW portion of the cycle, but for the blinking circuit, we need those times to be equal or 50% duty cycle since the LEDs will be ON and OFF for the same amount of time (1s).

$$T_{High} = 0.693(R_A + R_B)C; \quad T_{LOW} = 0.693(R_B)C$$

$$1s = 0.693(R_A + R_B)10\mu F; \quad 1s = 0.693(R_B)10\mu F$$

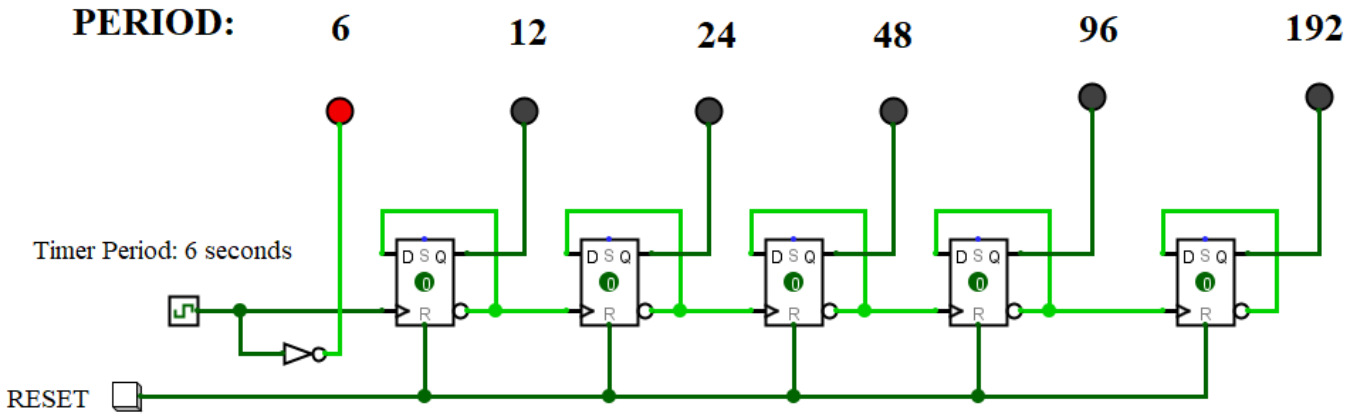
$$R_B = 144{,}300\Omega, \quad R_A = 0\Omega, \quad C = 10\mu F$$

Since $R_A = 0\Omega$, that would mean that V_{cc} is directly shorting with Pin7. To fix this issue, we set $R_A = R_B$, and short R_B with a diode so that the capacitor is discharging and charging at the same rate with a constant resistance. This ensures a 50% duty cycle.



Traffic Light Circuit

For the traffic light operation, the maximum period of the timer should be 6 seconds: 3 seconds on, and 3 seconds off. This ensures that the yellow lights can operate for 3 seconds. To increase the time of each cycle to around 186 seconds (the period of the entire traffic light) , you can add counters to the clock output so that the counters progressively oscillate at slower times as more counters are tied together. As mentioned earlier, a modified D- Flip Flop can be used for this process because it can take an oscillating input and double its period. The application of a D-Flip Flop as a counter is shown below. The reset button is an added advantage of using D-Flip Flops because it talks directly to the SR-latch inside the chip which can reset the value of Q output if the RESET is set to HIGH, causing a LOW Q output.



The logic diagram shows how a 6 second timer period can be doubled many times through the coupling of D-Flip Flops until it reaches about 192 seconds. This is close to the period needed for the functioning of the entire cycle of the traffic light which is 186 seconds. Since the rate of period doubling is linearly proportional to a constant (the clocks period), simply tweaking the clock period to slightly smaller value can give a closer value to the needed 186 second period. The calculation needed to find the clock period is shown below.

$$T_{TL} = T_c(2^n)$$

$$186 = T_c(2^5)$$

$$T_c = 5.8125$$

Since we need both traffic lights to function for the same amount of time, the high and low period of each cycle must be equal. This means that the timer should, like the blinking lights, function at a 50% duty cycle. Using the same formula discussed before, we can calculate the required resistor values given that we use a 10µF capacitor.

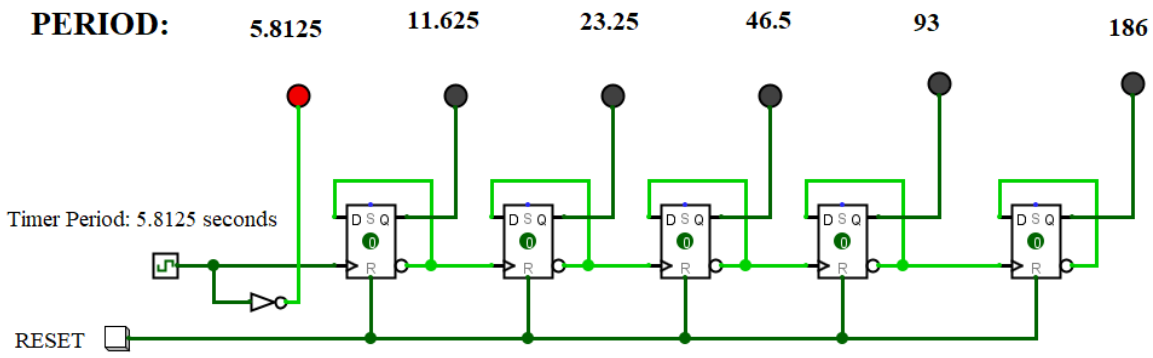
$$T_{High} = 0.693(R_A + R_B)C; \quad T_{LOW} = 0.693(R_B)C$$

$$3s = 0.693(R_A + R_B)10\mu F; \quad 3s = 0.693(R_B)10\mu F$$

$$R_B = 432{,}900\Omega, \quad R_A = 0\Omega, \quad C = 10\mu F$$

Again, since R_A cannot be equal to 0 Ω or else it would short Pin 7 of the timer, we short R_B with a diode and let $R_A = R_B$ so that the capacitor is discharging and charging at the same rate with a constant resistance.

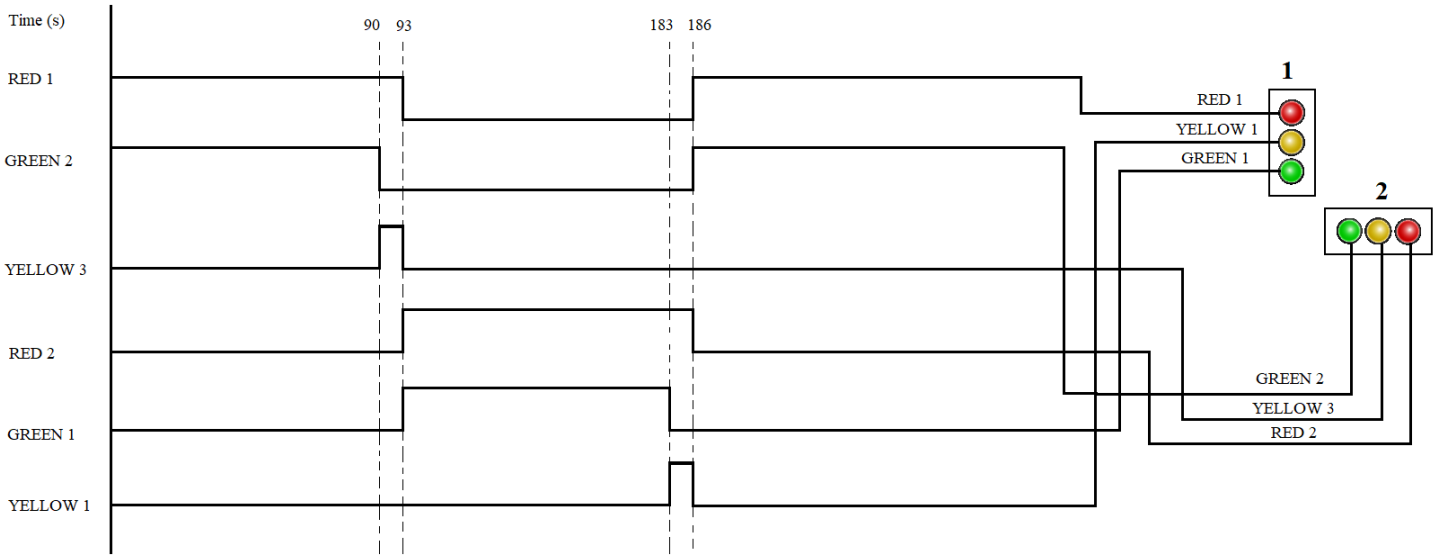
Using this new value clock value, we can update the previous period values



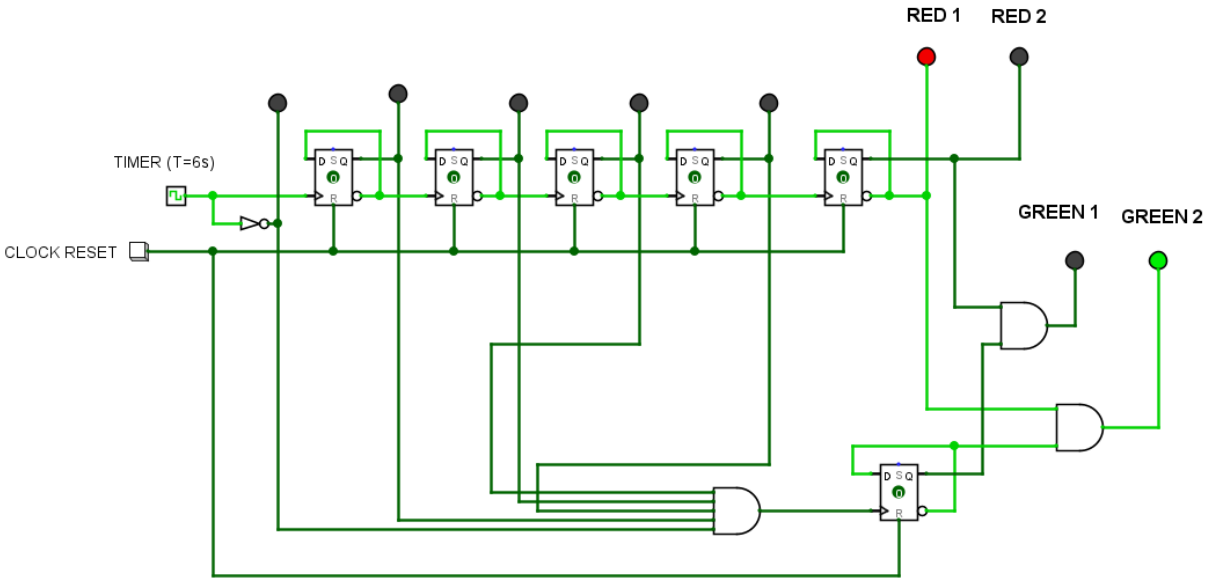
Although the new overall period value of the traffic light is exactly what it needs to be, 186s, the time that the yellow light will operate for will be closer 2.9 seconds instead of 3, and the green light will be on for 90.1 seconds instead of 90. Despite these drawbacks, given the amount of small amount of ICs used and the allowance of a little time error in the project description, I think this error is justified.

Speaking of error, I had initially planned on the counter to be a synchronous counter but I decided to keep it asynchronous. The reason why synchronous timers are preferred for applications that are time sensitive is because each counter is directly tied to the initial clock signal so there is almost no error in the time. With asynchronous counters, the clock pulse has to travel through all of the individual counters which create propagation delay; however this is only relevant if the initial clock is running at a very high frequency and/or if there are many Flip Flops involved. Since my traffic light is running at an extremely low frequency (~1/6hz) and is using only five flip flops, the asynchronous counter is more practical choice than the highly complex and gate demanding synchronous counter.

Now that the counter and clock is set to the right periods, the next step is to create the order in which all of the traffics lights will function. The timing diagram below shows exactly what signal output the circuit should give.



As you can see, the green and yellow lights of the second traffic light are related to the red light of the first traffic light. When the Red1 turns on for 93 seconds, the Green2 light should run for 90 seconds then turn off. The modified counter circuit is shown below.



When counter starts, Red1 and Green2 should be on. I use a multi input AND gate to see when the counter reaches 90 seconds based on the Q output of the individual counters. Since the AND gate will be activated only for a brief moment, I need a D-Flip Flop to remember and maintain that output. The triggering of the multi-input AND gate will change the state of the D-Flip Flop so Green2 will turn off and Green1 will turn on. However, this is not desired because Green1

should wait 3 seconds until Red 2 is activated in the second half of the Traffic Period. That is why there are two extra AND gates to make sure the green lights are only activated when their corresponding red lights are activated. But what about the yellow lights? Since the yellow2 is activated in the absences of Green2, but in the presence of Red1, I need something to regulate that. I used a truth table and some Boolean algebra to figure this out.

Truth Table

R1	G2	Y 2
0	0	0
1	0	1
0	1	0
1	1	0

Boolean Algebra

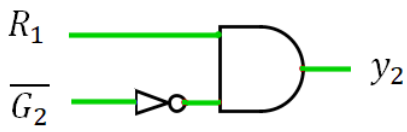
$$\overline{y_2} = \overline{R_1}\,\overline{G_2} + \overline{R_1}\,G_2 + R_1 G_2$$

$$\overline{y_2} = \overline{R_1}\,(\overline{G_2} + G_2) + R_1 G_2$$

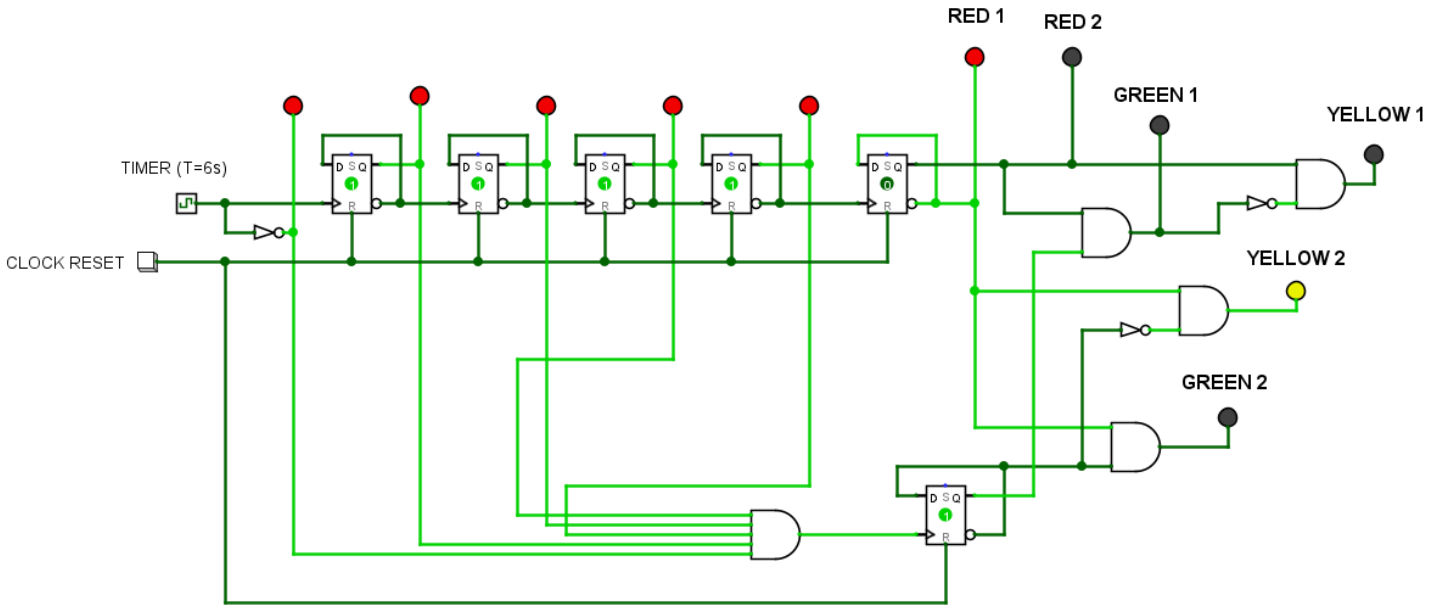
$$\overline{y_2} = \overline{R_1} + R_1 G_2$$

$$y_2 = R_1 \overline{G_2}$$

Resulting Gates

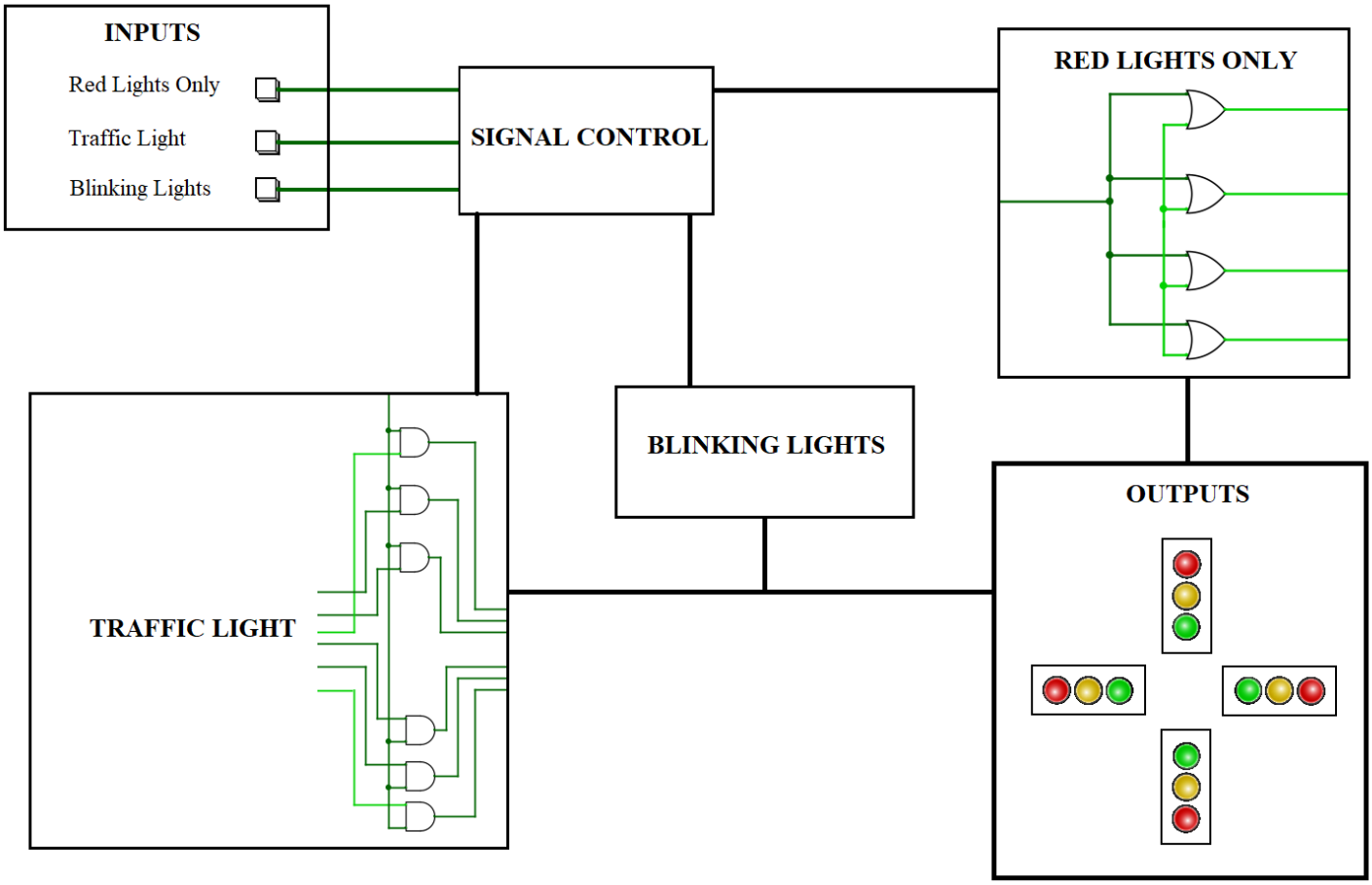


The resulting gates can be used to determine the state of yellow2 and can also be attached to other side of the counter to get the state of yellow1. This completes the final piece of the traffic light puzzle, and the resulting circuit is shown below.

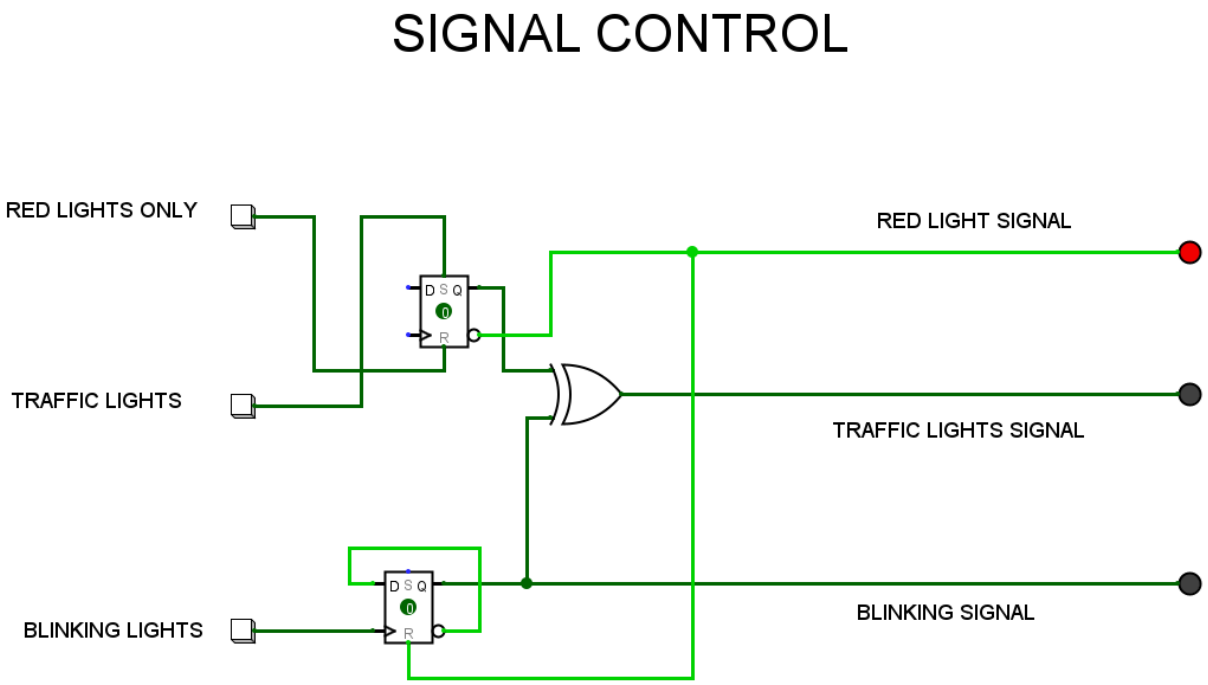


Power and Control

This project required that there be three operations that control the output of the traffic lights. The three operations were: all red lights, red and yellow blinking, and normal traffic light operation. To change states the circuit needs inputs in the form of short pulses created by a push button. Since I already have all the individual circuits created (the all red circuit wasn't discussed because it's fairly obvious as you will see), the only thing left to do is to tie it all together with the control circuit and traffic light output. The block diagram shows how this is done below.

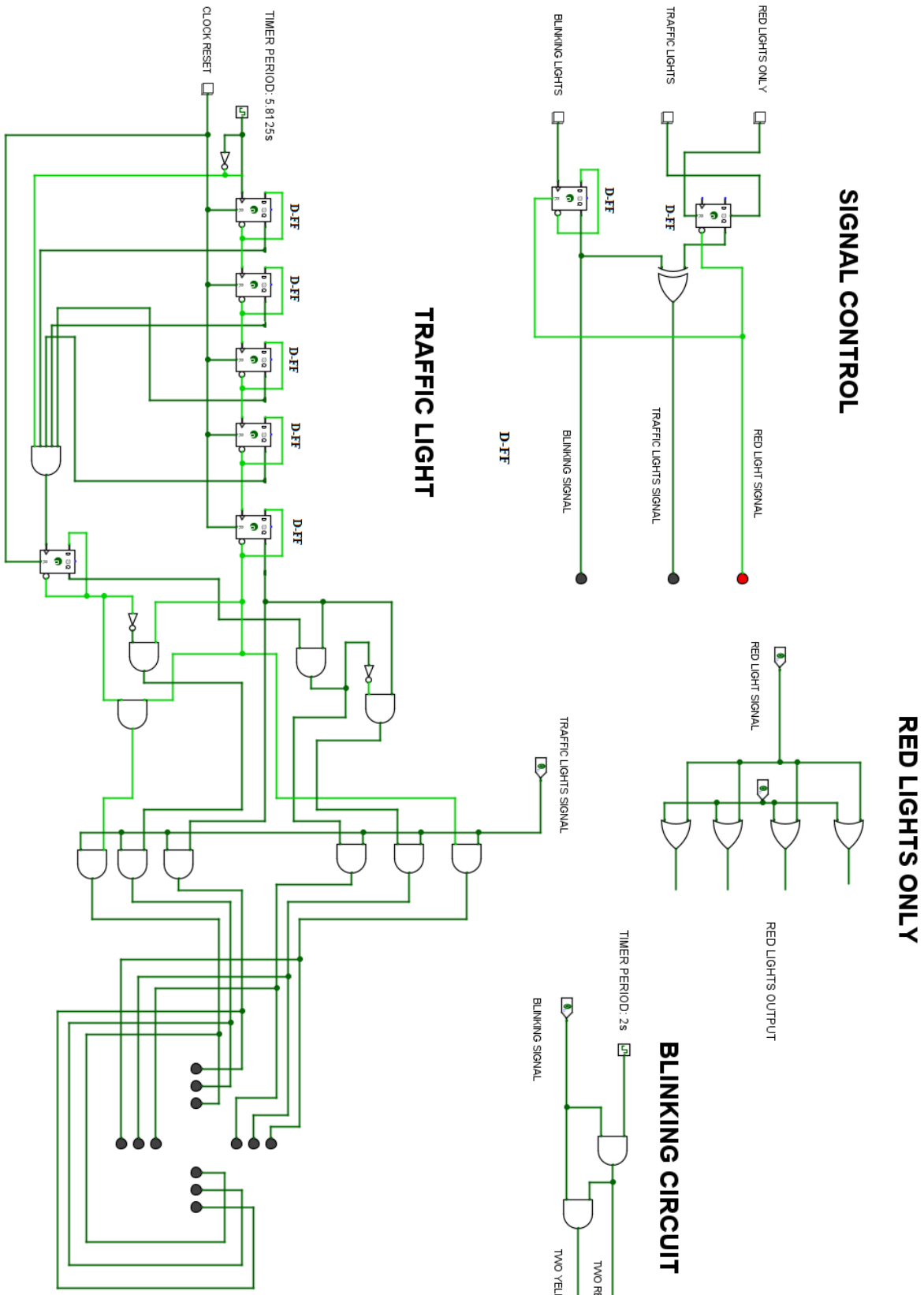


The one circuit that has not been discussed yet, as shown by the block diagram, is the Signal Control Circuit which is suppose to control which process is being executed. Initially, I had designed some SR latches in combination with a few gates to get the right output, but I then realized that I could D-Flip Flops to do this because they are essentially SR-latches with a few extra gates. After some experimentation on Logisim, I arrived at this logic circuit:



This circuit ensures that upon power up, the red lights are only activated since they are wired to the $\overline{Q}$ output which is on when nothing is toggled. Once the Traffic Light push button is toggled, it reroutes the signal from the red lights to the master AND gates (as seen in block diagram) that controls the signal output of the traffic light. If the Blinking Lights signal is toggled when Traffic Light is enabled, through the use of an XOR gate and a D-Flip Flop, the traffic light signal is suspended and blinking lights signal is activated. Toggling the blinking light push button again will reactivate normal traffic light operation. If at any time Red Light push button is toggled, the red lights will be activated.

Final Logic Circuit



Conclusion

This project was much more involved than the last project because it required a deep a understanding of sequential logic. I've learned a lot about how certain configurations of logic gates can hold a steady state given the opportunity to change states. I guess it ties into how potential difference works in IC's where electrons prefer to remain at a lower potential, thus creating a constant flow of them at specific pins.

I've also learned about how timers work and how we can use RC circuits to control the rate at which a timer "tiks" based on how fast the capacitor discharges and charges simply by changing resistor values. The addition of a timer in our logic circuits has helped me realize the vast applications a timer can have on sequential logic. For traffic light in particular, the addition of a timer meant a reoccurring signal pattern that could manipulated to output lights at specific times. The combination of Flip Flops and timers to create counters taught me how a single circuit can function on different frequencies even if the main timer oscillates at constant frequency.

If I were to do this again, I would think more about how I can use a single timer to control all of the circuits; in this project, I used two timers to control the blinking circuit and the traffic light. I'm also wondering if there was another way to combine the inputs of the three different circuits into one circuit since mine requires holding the output of the other circuits to prevent signal output overlap.