

Designing an Automatic Traffic Light Controller for a T-Junction Road.

James Gavin
2311749
ACDB5002

April 25, 2025

Contents

1	Introduction	6
2	Research	6
3	Circuit	6
3.1	Traffic Lights	8
3.1.1	Hardware	8
3.1.2	Software Interface	9
3.2	Counter	9
3.2.1	Hardware	10
3.2.2	Software Interface	11
3.3	Micro-controller	12
3.4	Voltage Regulator	12
3.4.1	Testing	12
3.5	Clock	15
4	Software	15
4.1	Pseudocode	15
4.2	Program	16
5	Testing	18
5.1	Initialisation	18
5.2	Lights Change	20
5.3	T2 Cycle Reset	22
5.4	T1 Cycle Reset	23
5.5	T0 Cycle Reset	24
6	Improvements	27
6.1	Timings	27
6.1.1	Software	27
6.2	Pedestrian Controls	27
6.2.1	Hardware	27
6.2.2	Software	30
7	Conclusion	32
8	Recommendations	32
9	References	33
A	C Code	34
A.1	Base Assignment	34
A.2	Timings	36
A.3	Pedestrian Buttons	38

List of Figures

1	Sheet 0 of the schematic	6
2	Sheet 1 of the schematic	7
3	Circuit diagram of a Traffic Light	8
4	Hardware Diagram of a Traffic Light	9
5	Circuit diagram of a single Counter	10
6	Hardware Diagram of the Counters	11
7	Hardware Diagram of the Micro-controller	12
8	Hardware diagram of the voltage regulator	12
9	Maximum current draw for each component.	12
10	Hardware diagram of the voltage regulator connected to PIC16F84A with optional load	13
11	Simulating the voltage regulator in ngspice with and without C_o	14
12	Hardware diagram of the crystal oscillator	15
13	Sequence of values for each traffic light.	15
14	Combined sequence of values for the traffic lights.	16
15	Circuit after initialisation	18
16	Circuit after initialisation and the LEDs updated	19
17	Circuit after change	20
18	Circuit after change and the LEDs updated	21
19	T2 before cycle reset	22
20	T2 after cycle reset	22
21	T1 before cycle reset	23
22	T1 after cycle reset	23
23	T0 before cycle reset	24
24	T0 after cycle reset	24
25	Circuit before T0 cycle reset	25
26	Circuit after T0 cycle reset	26
27	Sheet 0 of the schematic with the pedestrian controls	28
28	Sheet 1 of the schematic with the pedestrian controls	29
29	Circuit diagram of the buttons	30

List of Algorithms

1	Traffic Lights Modification Procedure	9
2	Counter Incrementing Procedure	11
3	Counter Resetting Procedure	11
4	Prototype Program	17
5	Timings adjusted Program	27
6	Pedestrian Button Program External Interrupt	30
7	Pedestrian Button Program	31

Listings

1	C code for the base assignment	34
2	C code for the correct timings	36
3	C code for the pedestrian buttons	38

1 Introduction

The task this report covers is to produce a prototype traffic system for a T-junction. Each side of the junction is to have a traffic light and a counter. The traffic lights are to each have a 10s interval during which their traffic can move, and an additional 10s interval for pedestrians to cross. During the pedestrian interval, all the traffic lights should be red. The counters should count from 0 to 39.

2 Research

A T-junction is a type of junction occurs when a road intersects perpendicular to another road, hence the ‘T’ shape. The traffic coming from road that is perpendicular, must ‘give way’ to traffic on the other [1, p. 68].

For this style of junction, pedestrian crossings can appear on all three sides. When a pedestrian is waiting to cross or is currently crossing, the traffic on the road must ‘give way’ to the pedestrians to allow them to cross [1, p. 5].

This type of junction can be configured with or without traffic lights.

3 Circuit

Figure 1 and 2 show the complete schematic for the traffic lights circuit.

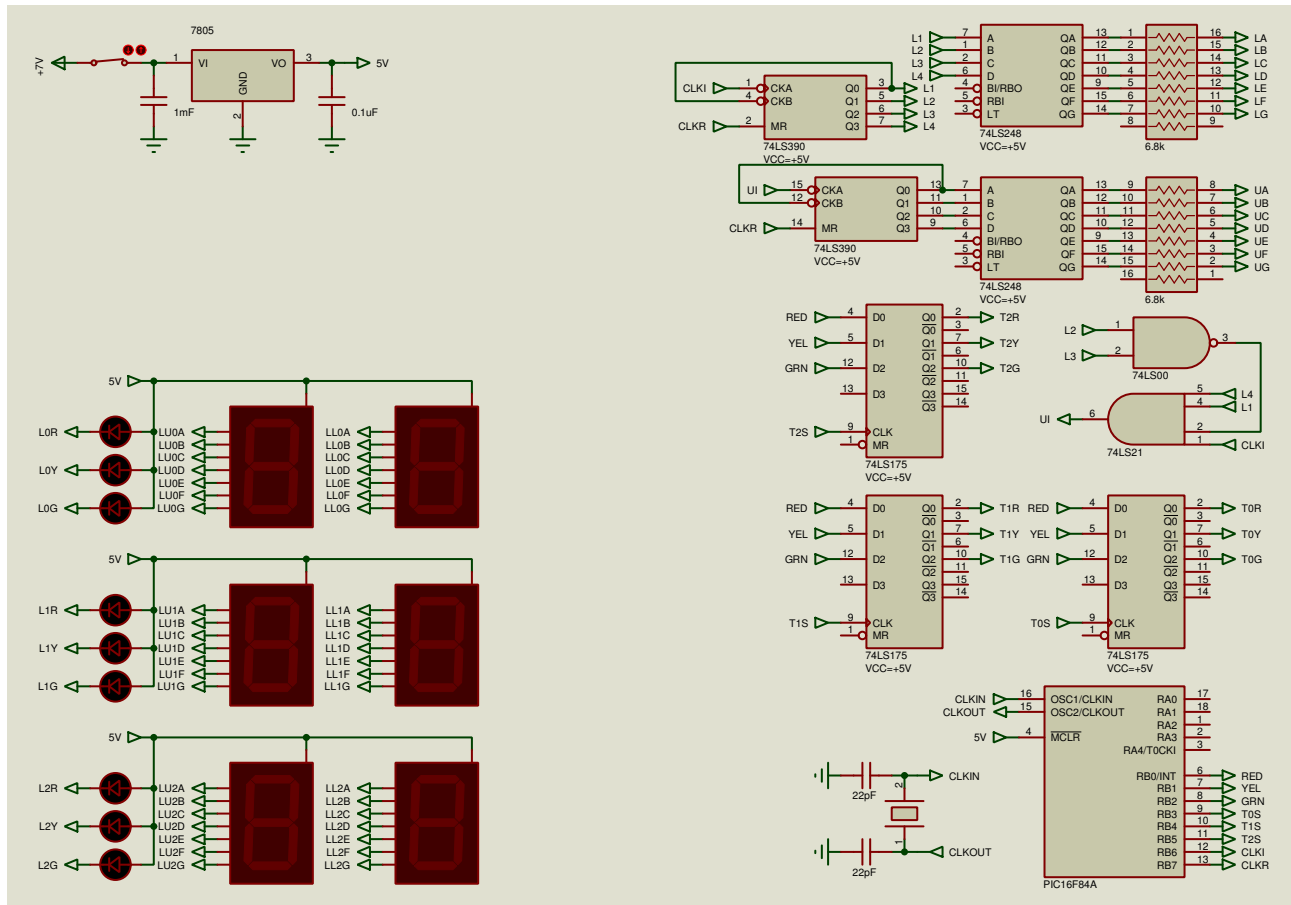


Figure 1: Sheet 0 of the schematic

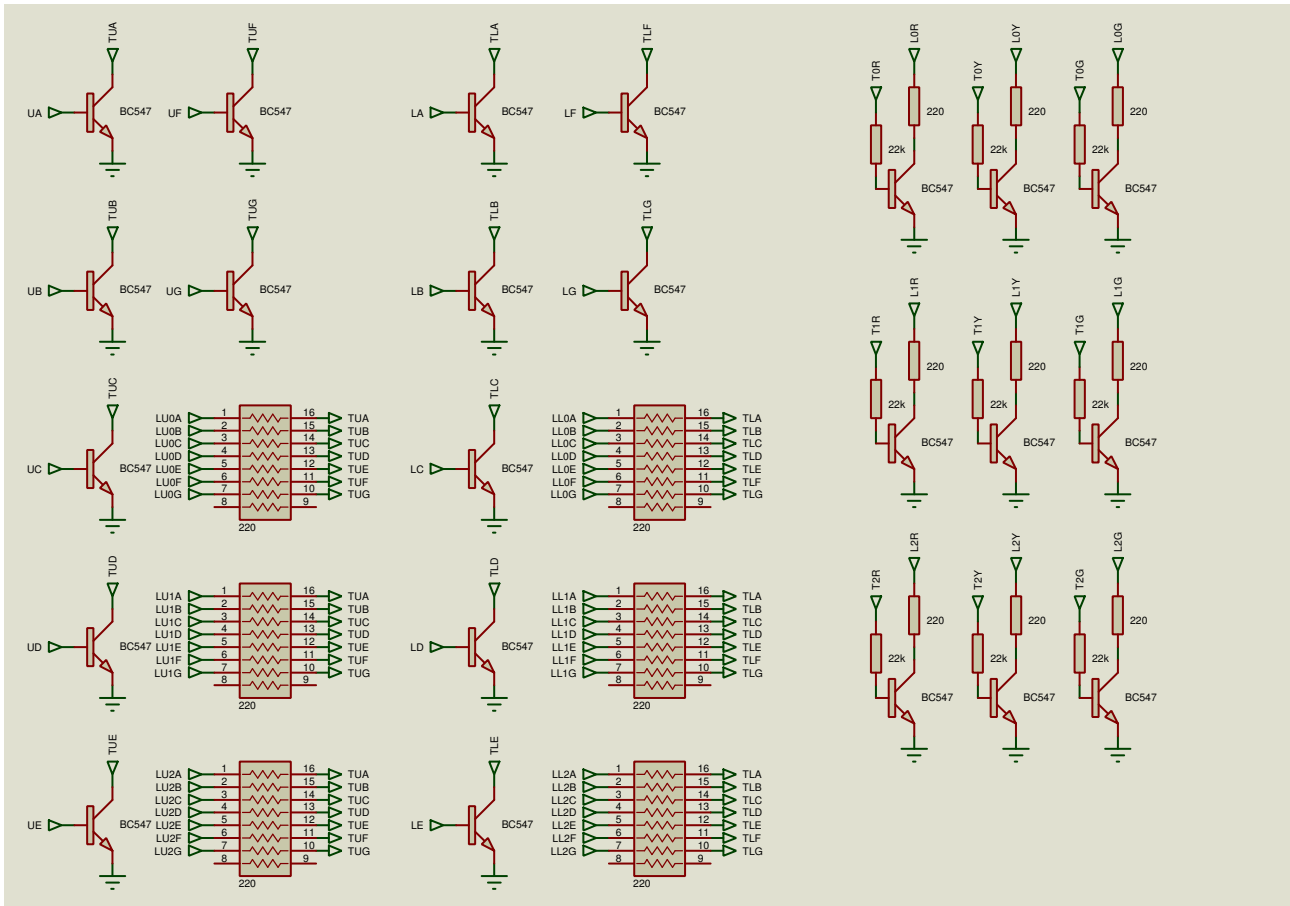


Figure 2: Sheet 1 of the schematic

3.1 Traffic Lights

The simplest configuration to attach the traffic lights to the micro-controller, requires parallel wires from each light to a corresponding pin. For the three traffic lights setup required by the specification, this would require nine pins. Using this configuration does not scale effectively, with the total number of pins used being $3n$, where n is the number of traffic lights. Additionally, as the traffic lights are not required to be updated rapidly - about 1 Hz - the opportunity presents itself for the use of a protocol.

Serial interfaces like I2C [2] and UART [3] are needlessly complicated, requiring external decoders and even additional micro-controllers to work, but SPI stores the received data into a single 8-bit register, which was ideal [4, p. 3]. The traffic light could then read bits 0 to 2 and use them as values to power the lights. Chip Select lines are used to decide which traffic light receives the value. This would reduce the total pin count to $3 + n$.

This is still too complicated. SPI is a bi-directional interface, meaning it shares data both ways when, for this use case, only uni-directional transmission - from micro-controller to traffic light - is required. This does not take into account the fact that every SPI transmission moves 8-bits of data, whilst only three are actually used to control the traffic light.

SPI uses three wires to transmit the data, which is the same number of bits that need to be transmitted to the the traffic lights. If instead of using the SPI protocol, the red, amber, and green values are simply shared on these wires, then the Chip Select could be used to “copy” the values to the corresponding traffic light.

This setup conveniently uses D-type flip flops as shown in Figure 3. Not only does it use the same pin count $3 + n$, but it is also considerably cheaper and easier to implement in both software and hardware.

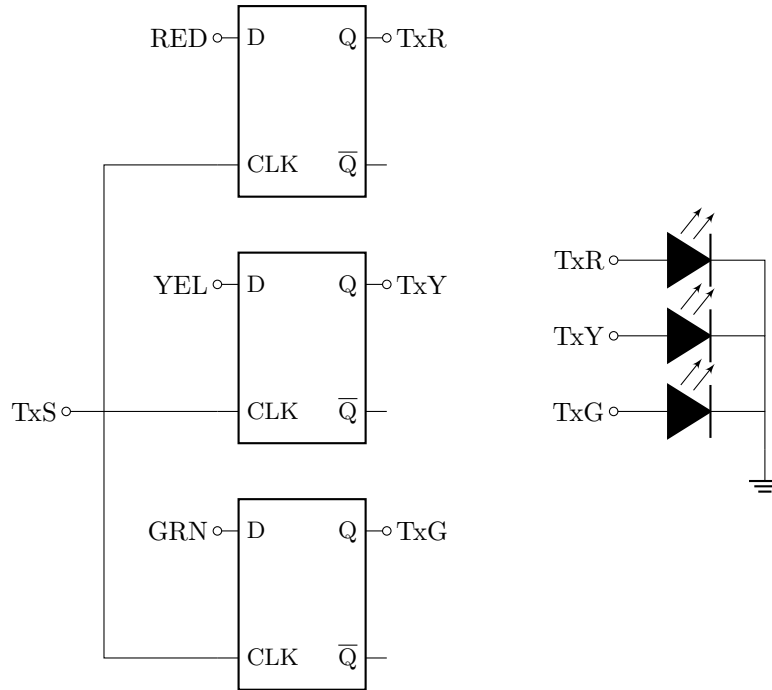


Figure 3: Circuit diagram of a Traffic Light

3.1.1 Hardware

A suitable IC was needed to replace the D-type flip flops. The 74LS175 [5] combines four into a single package, making it convenient to use one per traffic light set.

ICs of the 74LS variety typically have a low source current I_{OH} of $-400\mu A$. This is not sufficient to drive a LED with a typical current draw of 10 mA. A transistor, is used to increase the current.

A NPN transistor, in this case the BC547 [6], was chosen so that the current could be sunk, not sourced by the transistor. The target Collector Current I_C was 15 mA, which, with a minimum DC Current Gain h_{FE} of 110, meant a Base Current I_B of 0.136 mA was required. Taking into account the typical Base-Emitter On Voltage value being 0.66 V, the ideal resistor to achieve the I_C would have been $31.8k\Omega$, but the $22k\Omega$ was chosen instead to account for tolerances and to use a standard resistor.

Finally, the LEDs each need a 220Ω resistor to cap the current to 13.2 mA. The final hardware diagram for a set of traffic lights is shown in Figure 4.

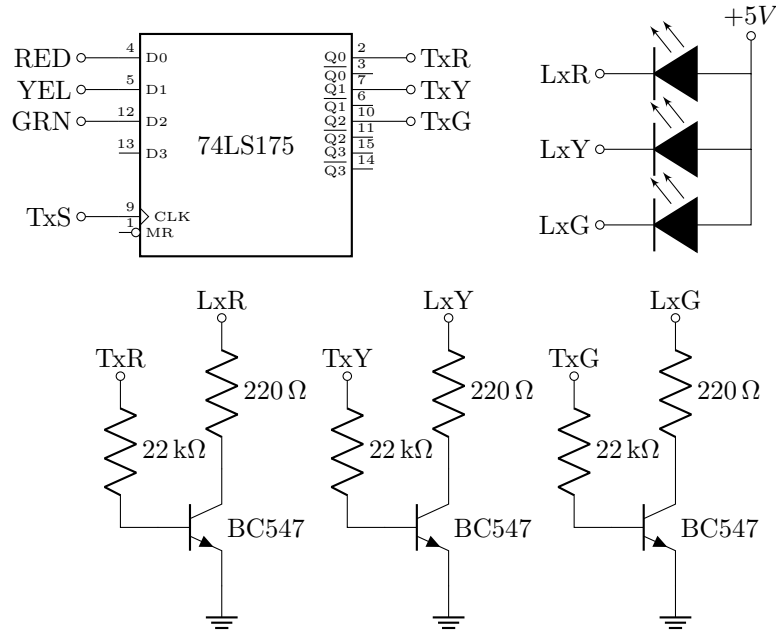


Figure 4: Hardware Diagram of a Traffic Light

3.1.2 Software Interface

The three traffic lights use six wires. Red, amber, and green are connected to pins 0 to 2 of PORTB, with the traffic light select wires connected to pins 3 to 5.

To change the value of a traffic light, first the red, amber, and green pins need to be outputting the correct value before selecting the traffic light. Then the traffic light is deselected before the red, amber, and green pins can be modified again. Algorithm 1 shows this interface.

Algorithm 1 Traffic Lights Modification Procedure

```

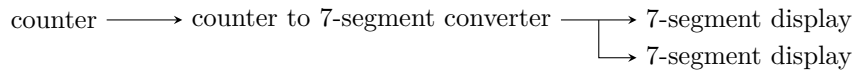
procedure SET_TRFFC_LTS(lts, val)
    PORTB0:2 ← val
    PORTB3:5 ← lts
    PORTB3:5 ← 0
end procedure

```

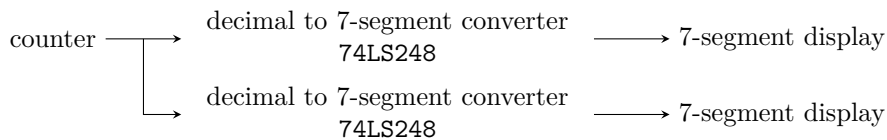
3.2 Counter

A similar problem to the traffic lights presents itself when implementing the counters. Though all the 7-segment displays can be connected in parallel to the same pins, 14 are still required. As these counters are only displaying seconds, a simpler interface is required. Since the only two actions for the counters is to increment and reset when at 40, the interface should reflect this.

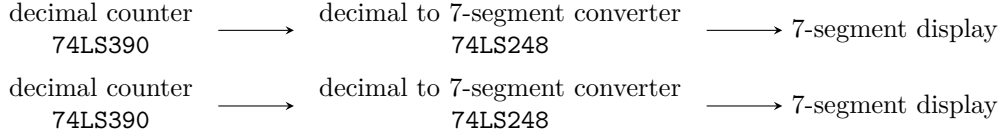
In the end, an 2-wire interface was devised; one wire for incrementing and the other for resetting. This would require a hardware counter and a means to display the counter value on 7 segment displays. The initial flow diagram looked something like this:



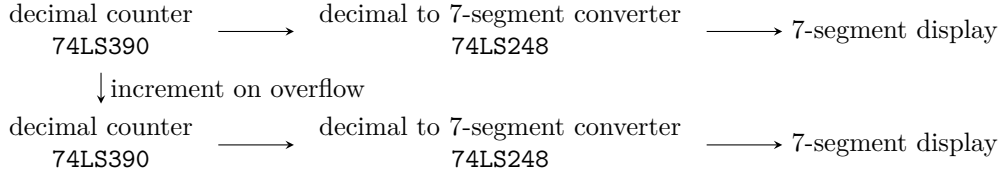
The 74LS248 [7] was identified as a suitable decimal to 7-segment converter. The 74LS248 can only convert for one 7-segment display, so two are needed:



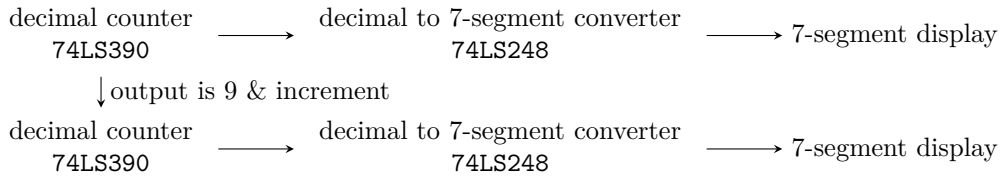
As the 7-segment converters require decimal input, a decimal counter is required. However, no decimal counter was found that outputs both ones and tens. The closest to the desired chip was the 74LS390 [8], which has 2 decimal counters build in. One will be used for the ones and the other for the tens:



In this configuration, a 3-wire interface would be needed, and the micro-controller would need to track the counter to correctly update the tens. The ideal configuration would detect the overflow of the ones counter from 9 to 0 and increment the tens:



This has been achieved by incrementing the tens counter when the ones counter is 9 and the next increment command is received:



The final circuit design is shown in Figure 5.

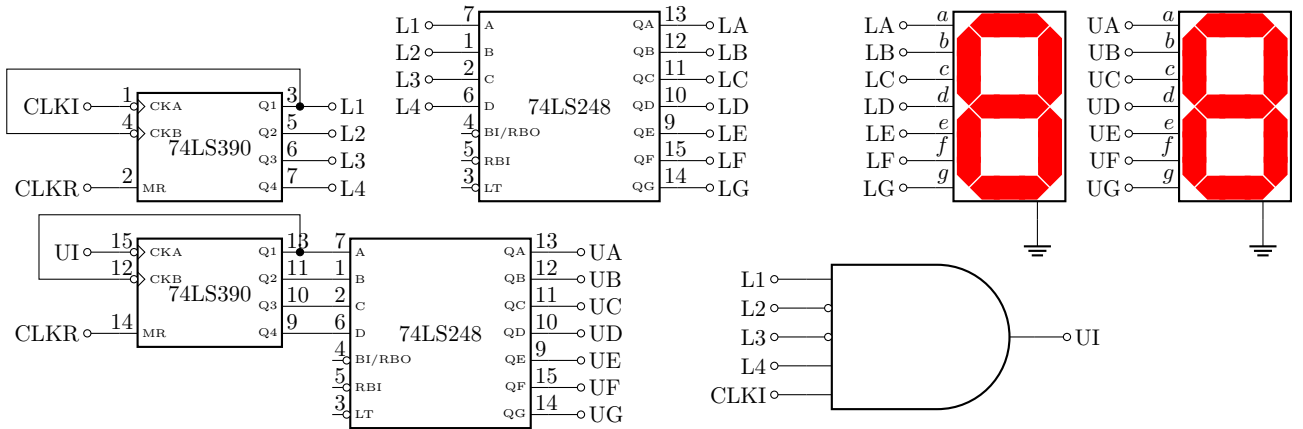


Figure 5: Circuit diagram of a single Counter

3.2.1 Hardware

The same type of ICs, the 74LS variety, were used as in the traffic lights. This time however, three counters are connected to each output.

The HDSP-550G was chosen as the red anode 7-segment display [9, pp. 4, 8], with a typical 2.1 V Forward Voltage V_F for each segment.

The same NPN transistor, the BC547, was use. The target Collector Current I_C was 45 mA, which, with a minimum DC Current Gain h_{FE} of 110, meant a Base Current I_B of 0.410 mA was required. Taking into account the typical Base-Emitter On Voltage value being 0.66 V, the ideal resistor achieve the I_C would have been 10.6 k Ω , but the 6.8 k Ω was chosen instead to account for tolerances and to use a standard resistor value.

Finally, each segment of each 7-segment display need a 220 Ω resistor to cap the current to 13.2 mA.

The final hardware diagram for the counters is shown in Figure 4.

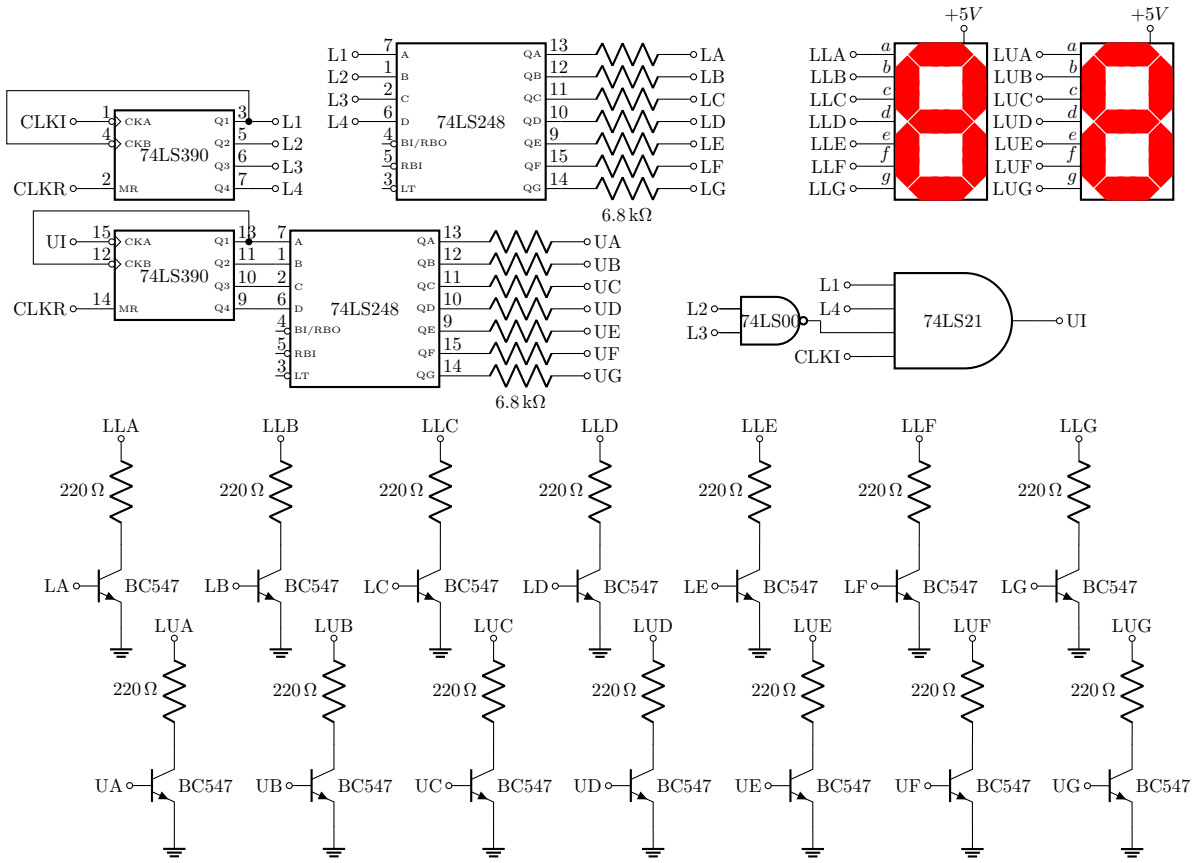


Figure 6: Hardware Diagram of the Counters

3.2.2 Software Interface

The increment and reset wires are connected to pins 6 and 7 of PORTB respectively. Both actions require the respective pin to be pulsed. Algorithm 2 and 3 show this interface.

Algorithm 2 Counter Incrementing Procedure

$PORTB_6 \leftarrow 1$
 $PORTB_6 \leftarrow 0$

Algorithm 3 Counter Resetting Procedure

$PORTB_7 \leftarrow 1$
 $PORTB_7 \leftarrow 0$

3.3 Micro-controller

The micro-controller that was chosen for this circuit was the PIC16F84A [10]. It was chosen for its minimal specifications with 1024 B of ROM, 68 B of SRAM, and only two ports. Not only that, it has 64 B of EEPROM which was important for the Software implementation. Figure 7 shows the connections of the micro-controller for this circuit.

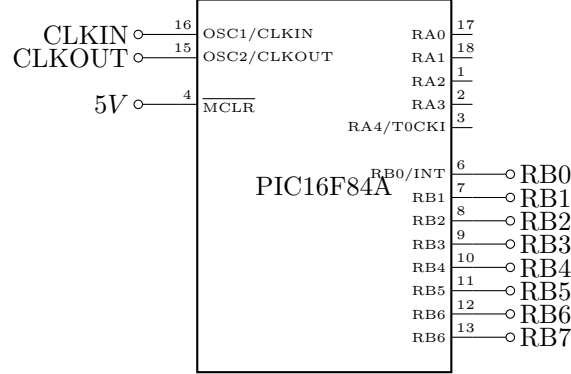


Figure 7: Hardware Diagram of the Micro-controller

3.4 Voltage Regulator

For this circuit, the input is typically a 240 V, 60 Hz mains connection. However, to simplify the prototype, the AC to DC conversion was omitted and a 7.5 V–35 V DC power source assumed.

A voltage regulator is required to convert the input voltage (V_{cc}) to 5 V. The 7805 is a voltage regulator that takes a 7.5 V–35 V input voltage and outputs 4.9 V–5.1 V [11, pp. 4, 5], safely inside the ratings required for the micro-controller to operate [10, p. 51]. Figure 8 shows the implementation of the voltage regulator for this circuit.

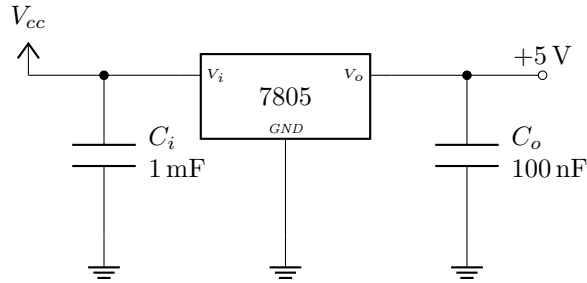


Figure 8: Hardware diagram of the voltage regulator

3.4.1 Testing

To effectively test if the voltage regulator is adequate, the maximum current draw of each component is needed, as shown in Figure 9.

Component	Quantity	I_{CC}
74LS00	1	4.4 mA
74LS21	1	4.4 mA
74LS175	3	18 mA
74LS248	2	13 mA
74LS390	1	26 mA
PIC16F84A	1	1 mA
LED	51	13.2 mA
Total		789 mA

Figure 9: Maximum current draw for each component.

The maximum current draw of 789 mA is safely below the maximum of 1.5 A [11, p. 10]. The other important factor to test is the transient response when all the LEDs simultaneously turn on. Figure 10 approximates the current draw change for this scenario.

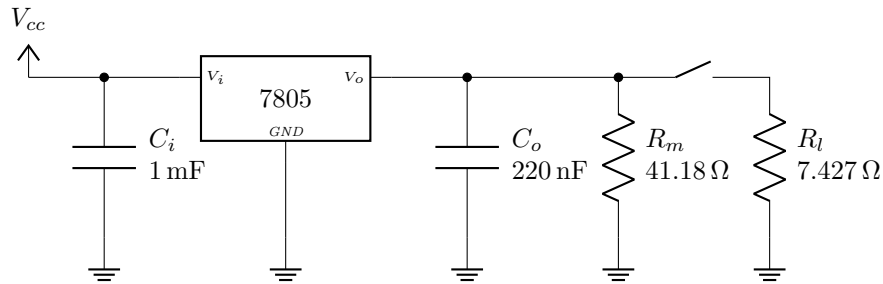


Figure 10: Hardware diagram of the voltage regulator connected to PIC16F84A with optional load

R_m represents the typical 116 mA draw from all the ICs during standard operation, and R_l represents a 673 mA load that occurs when all the LEDs turn on. Figure 11 shows the output of simulating this circuit in ngspice [12]. The switch, representing all the LEDs turning on, is closed at 10 μ s.

With no capacitor, the voltage drops to 3.7 V. This is below the minimum rating of the supply voltage, when the micro-controller is configured to work with a high speed oscillator, of 4.5 V [10, p. 51]. This would most likely cause the micro-controller to reset, creating an infinite loop.

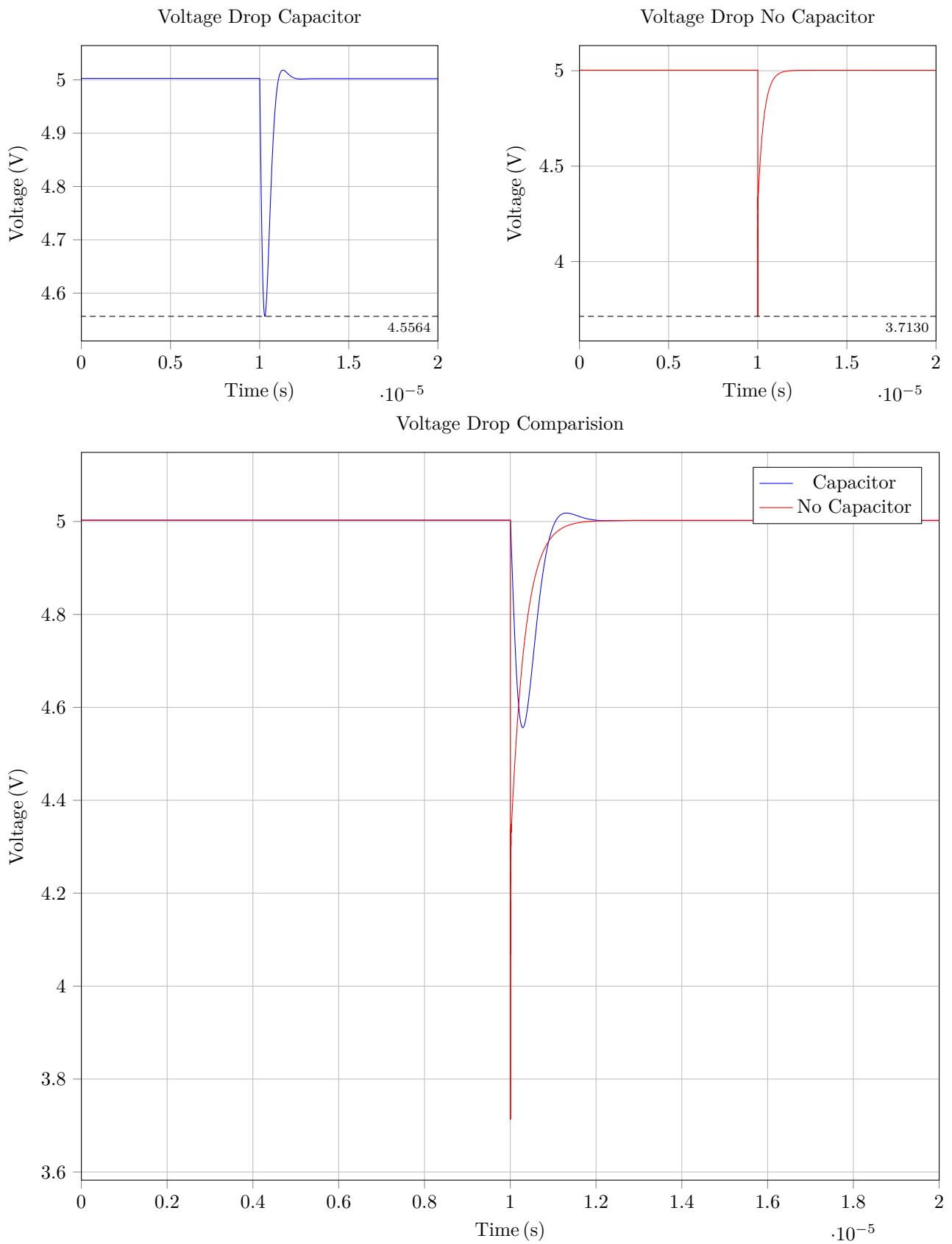


Figure 11: Simulating the voltage regulator in `ngspice` with and without C_o

3.5 Clock

The circuit uses a 1 MHz external quartz crystal oscillator to provide a clock for the micro-controller. Figure 12 shows the implementation of the crystal oscillator.

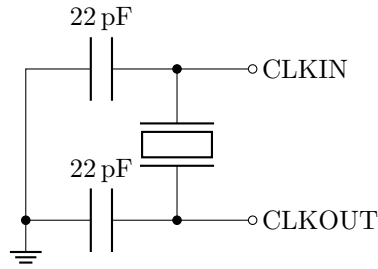


Figure 12: Hardware diagram of the crystal oscillator

4 Software

To ensure the safe operation of the T-junction, the software needed to be robust, simple and testable.

The realisation that had to be made to effectively complete this goal was that this system has no feedback and is in fact a closed system. It does output values, but the only source of input is the clock, which always runs at 1 MHz. This realisation allows the software to simply playback a pre-recorded sequence of light combinations much like a CD for music.

4.1 Pseudocode

The sequence that will be played back is a list of 40 light combinations, one for each second of the counter. Figure 13 shows the sequence for each of the three traffic lights. The decision was also made to turn all the lights red during the pedestrian phase, 30 s–39 s, as being explicit is safer for pedestrians.

0 s–9 s	10 s–19 s	20 s–29 s	30 s–39 s	0 s–9 s	10 s–19 s	20 s–29 s	30 s–39 s
Yellow	Red	Red	Red	Red	Yellow	Red	Red
Green	Red	Red	Red	Red	Green	Red	Red
Green	Red	Red	Red	Red	Green	Red	Red
Green	Red	Red	Red	Red	Green	Red	Red
Green	Red	Red	Red	Red	Green	Red	Red
Green	Red	Red	Red	Red	Green	Red	Red
Green	Red	Red	Red	Red	Green	Red	Red
Green	Red	Red	Red	Red	Green	Red	Red
Green	Red	Red	Red	Red	Green	Red	Red
Yellow	Red	Red	Red	Red	Yellow	Red	Red

0 s–9 s	10 s–19 s	20 s–29 s	30 s–39 s
Red	Red	Yellow	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Yellow	Red

Figure 13: Sequence of values for each traffic light.

The simplification becomes apparent from Figure 13 is that all the traffic lights follow the same sequence, but each at different offsets. Taking the last sequence and labelling the columns 1 through 4, as is done in Figure 14, it is clear that the original sequences for traffic lights start at columns 3, 2, and 1 respectively.

1	2	3	4
Red	Red	Yellow	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Green	Red
Red	Red	Yellow	Red

Figure 14: Combined sequence of values for the traffic lights.

This table is converted to a linear array in the Pseudocode, and each traffic light starts at its designated offset: 20, 10, and 0. When a traffic light reaches the end of the array - offset of 40 - it cycles back to the start of the array.

Each execution of this sequence also requires the counter is incremented and eventually reset when it gets to 40. The pseudocode that shows this is in Algorithm 4.

4.2 Program

The program is written in the C programming language using the CCSC compiler [13]. C was chosen for its conciseness with features specific to the CCSC compiler used to further robust the program.

The main feature used was the `rom` pre-processor directive, that allowed the traffic light sequence to be stored in read-only memory.

The source code can be found in Listing 1.

Algorithm 4 Prototype Program

```
trffc_lts_state ← [  
  'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED',  
  'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED',  
  'YELLOW', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'YELLOW',  
  'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED', 'RED'  
]  
procedure MAIN  
  configure  $PORTB_{0:7}$  ← as outputs  
   $t0 \leftarrow 0$   
   $t1 \leftarrow 10$   
   $t2 \leftarrow 20$   
  loop  
    SET_TRFFC_LTS( $S0$ , trffc_lts_state[ $t0$ ])  
    SET_TRFFC_LTS( $S1$ , trffc_lts_state[ $t1$ ])  
    SET_TRFFC_LTS( $S2$ , trffc_lts_state[ $t2$ ])  
    delay_ms(1000)  
     $t0 \leftarrow t0 + 1$   
     $t1 \leftarrow t1 + 1$   
     $t2 \leftarrow t2 + 1$   
    if  $t0 = 40$  then  
       $t0 \leftarrow 0$   
       $PORTB_7 \leftarrow 1$   
       $PORTB_7 \leftarrow 0$   
    else  
       $PORTB_6 \leftarrow 1$   
       $PORTB_6 \leftarrow 0$   
    end if  
    if  $t1 = 40$  then  
       $t1 \leftarrow 0$   
    end if  
    if  $t2 = 40$  then  
       $t2 \leftarrow 0$   
    end if  
  end loop  
end procedure
```

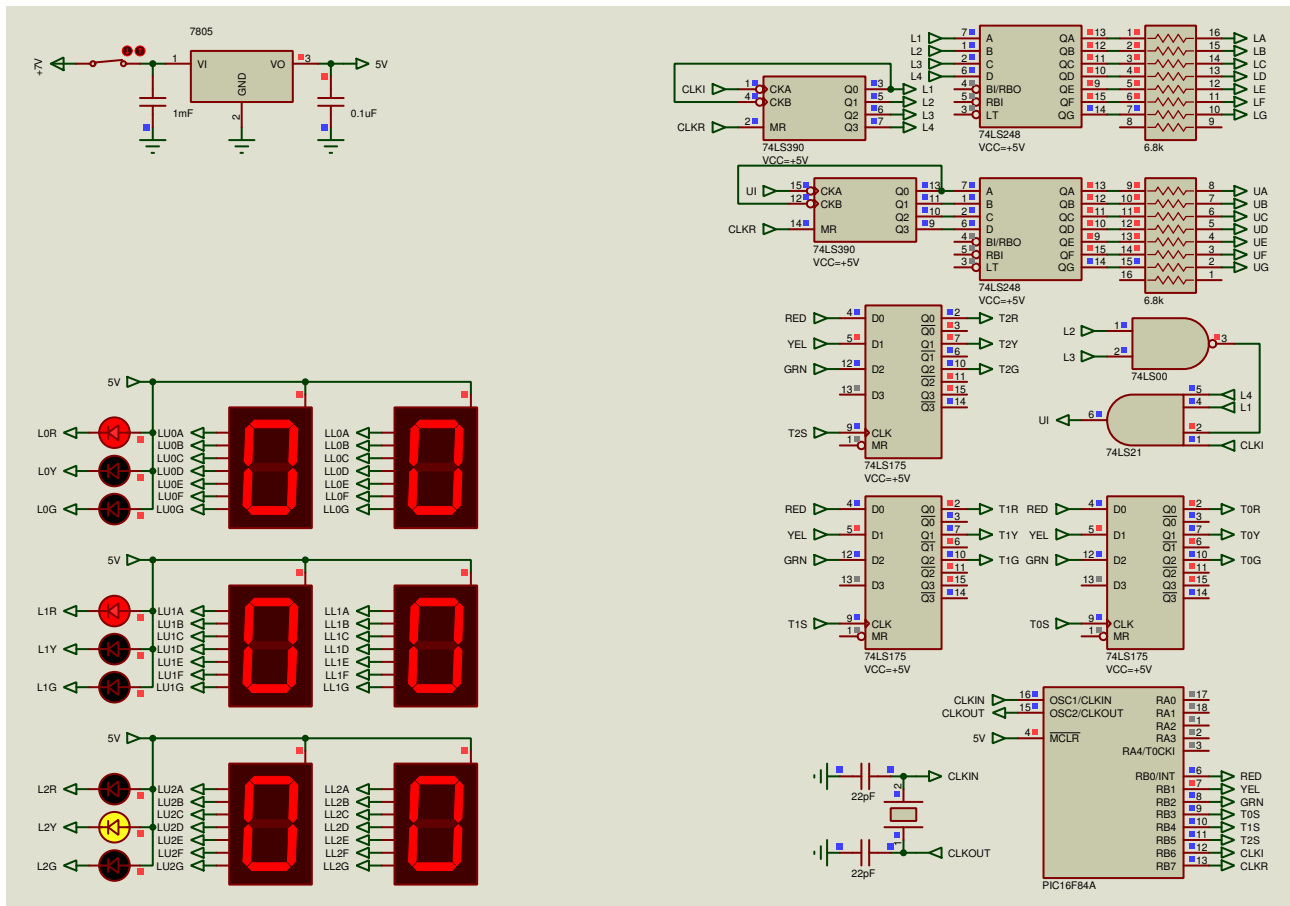


Figure 16: Circuit after initialisation and the LEDs updated

5.2 Lights Change

Throughout a period of 1 s, the counter and traffic lights display the same values. After 1 s the counter is incremented, and the traffic lights are updated to their next state: t_0 red, t_1 red, and t_2 green, as shown in Figure 17.

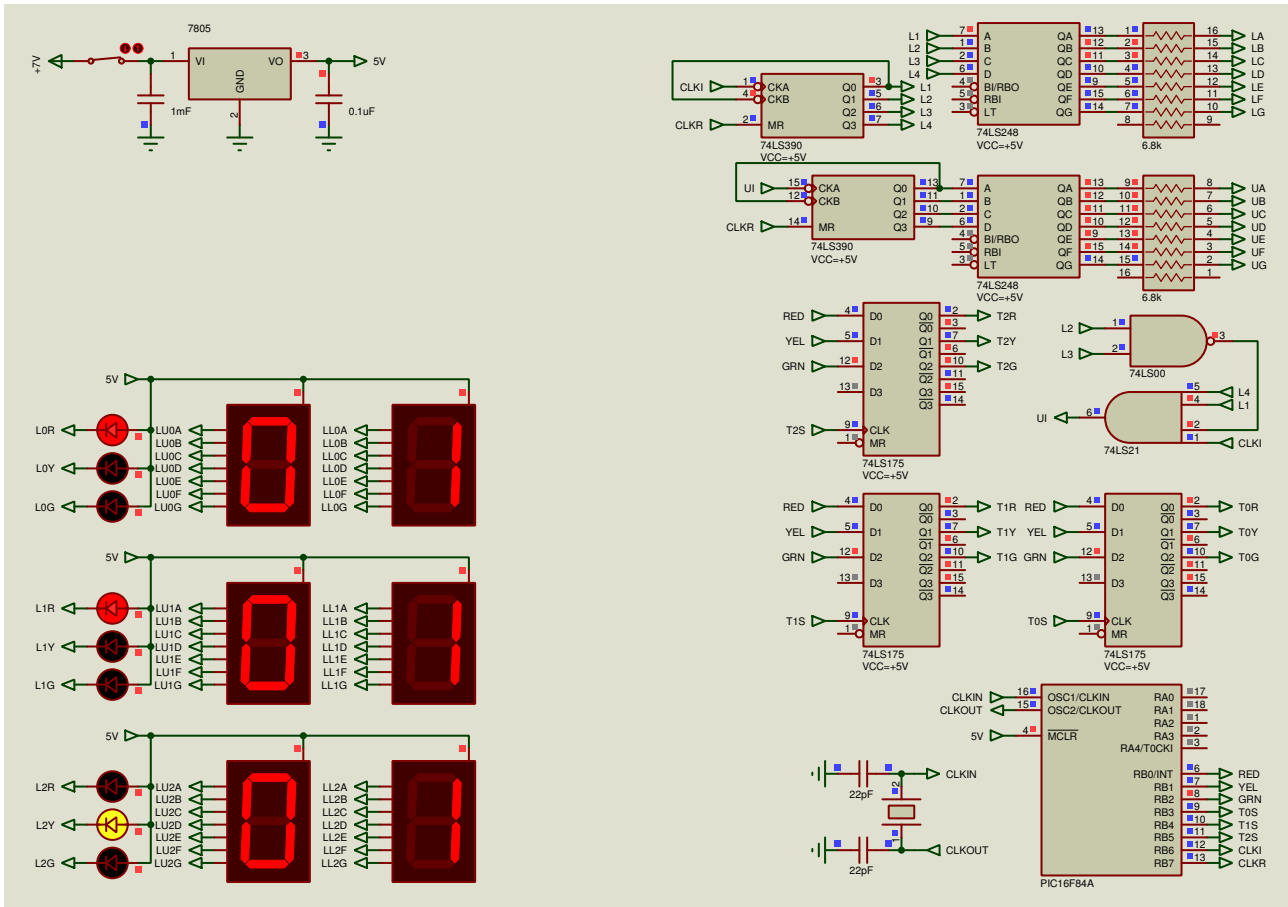


Figure 17: Circuit after change

A further 25 ms is required for the LEDs to adjust to their new values, as shown in Figure 18.

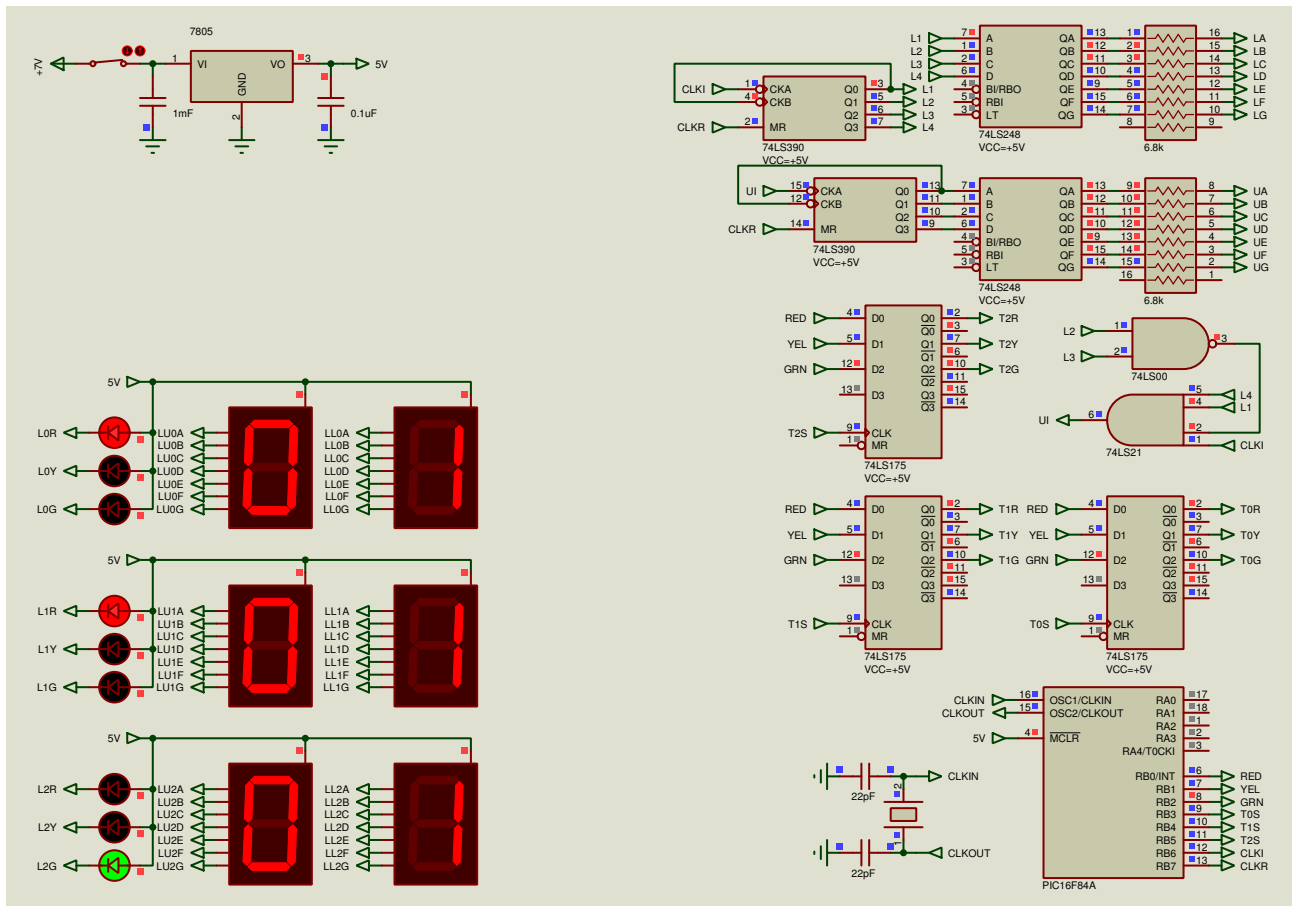


Figure 18: Circuit after change and the LEDs updated

5.3 T2 Cycle Reset

After 20 s has elapsed, traffic light **t2** will have reached the end of the sequence. When this occurs, the sequence needs to restart from the beginning. This reset is shown in Figure 19 and Figure 20.

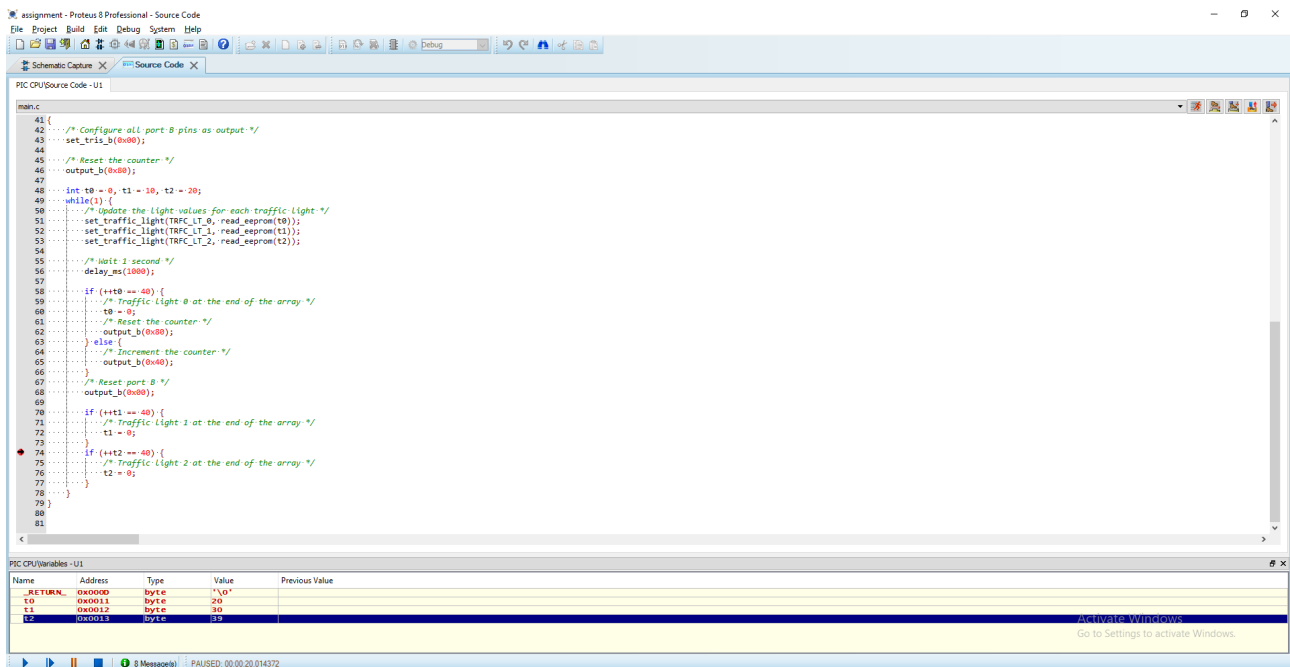


Figure 19: T2 before cycle reset

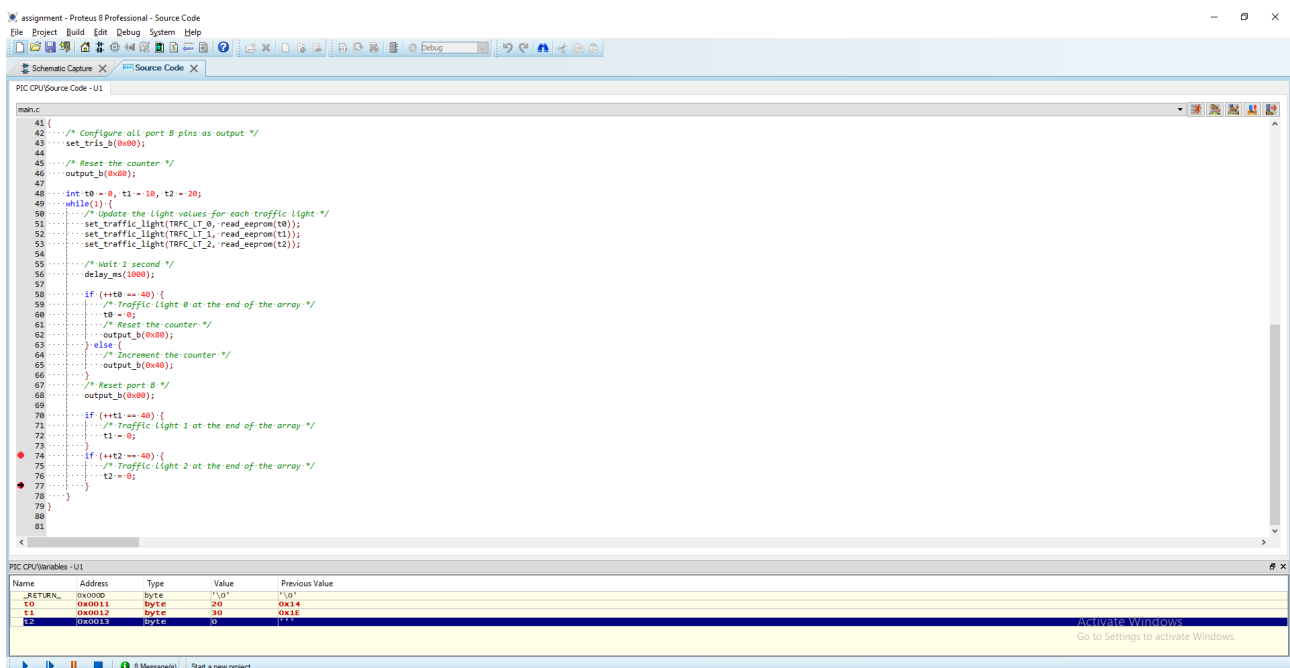


Figure 20: T2 after cycle reset

5.4 T1 Cycle Reset

After 30 s has elapsed, traffic light **t1** will have reached the end of the sequence. When this occurs, the sequence needs to restart from the beginning. This reset is shown in Figure 21 and Figure 22.

```
main.c
41 {
42     /* Configure all port B pins as output */
43     set_tris_b(0x00);
44
45     /* Reset the counter */
46     output_b(0x00);
47
48     int t0 = 0, t1 = 10, t2 = 20;
49     while(1) {
50         /* Update the light values for each traffic light */
51         set_traffic_light(TRFC_L1_0, read_eeprom(t0));
52         set_traffic_light(TRFC_L1_1, read_eeprom(t1));
53         set_traffic_light(TRFC_L1_2, read_eeprom(t2));
54
55         /* Wait 2 second */
56         delay_ms(1000);
57
58         if (++t0 == 40) {
59             /* Traffic light 0 at the end of the array */
60             t0 = 0;
61             /* Reset the counter */
62             output_b(0x00);
63         } else {
64             /* Increment the counter */
65             output_b(0x00);
66         }
67         /* Reset port B */
68         output_b(0x00);
69
70         if (++t1 == 40) {
71             /* Traffic light 1 at the end of the array */
72             t1 = 0;
73         }
74         if (++t2 == 40) {
75             /* Traffic light 2 at the end of the array */
76             t2 = 0;
77         }
78     }
79 }
80
81
```

Name	Address	Type	Value	Previous Value
RETURN	0x0000	byte	0x01	0x01
t0	0x0011	byte	30	0x0011
t1	0x0012	byte	0	0x0012
t2	0x0013	byte	9	0x0013

Figure 21: T1 before cycle reset

```
main.c
41 {
42     /* Configure all port B pins as output */
43     set_tris_b(0x00);
44
45     /* Reset the counter */
46     output_b(0x00);
47
48     int t0 = 0, t1 = 10, t2 = 20;
49     while(1) {
50         /* Update the light values for each traffic light */
51         set_traffic_light(TRFC_L1_0, read_eeprom(t0));
52         set_traffic_light(TRFC_L1_1, read_eeprom(t1));
53         set_traffic_light(TRFC_L1_2, read_eeprom(t2));
54
55         /* Wait 2 second */
56         delay_ms(1000);
57
58         if (++t0 == 40) {
59             /* Traffic light 0 at the end of the array */
60             t0 = 0;
61             /* Reset the counter */
62             output_b(0x00);
63         } else {
64             /* Increment the counter */
65             output_b(0x00);
66         }
67         /* Reset port B */
68         output_b(0x00);
69
70         if (++t1 == 40) {
71             /* Traffic light 1 at the end of the array */
72             t1 = 0;
73         }
74         if (++t2 == 40) {
75             /* Traffic light 2 at the end of the array */
76             t2 = 0;
77         }
78     }
79 }
80
81
```

Name	Address	Type	Value	Previous Value
RETURN	0x0000	byte	0x01	0x01
t0	0x0011	byte	30	0x0011
t1	0x0012	byte	0	0x0012
t2	0x0013	byte	9	0x0013

Figure 22: T1 after cycle reset

5.5 T0 Cycle Reset

After 40 s has elapsed, traffic light t0 will have reached the end of the sequence. When this occurs, the sequence needs to restart from the beginning. At the same time, the counter needs to be reset. This reset is shown in Figure 23, 24, 25, and 26.

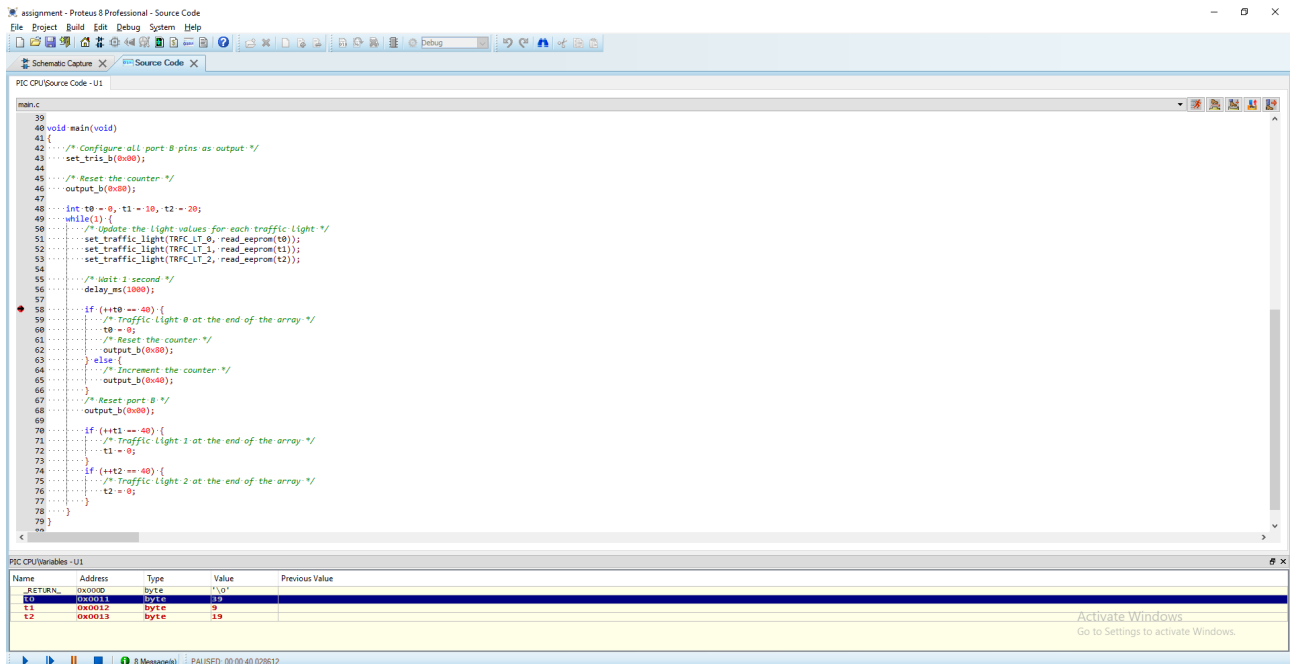


Figure 23: T0 before cycle reset

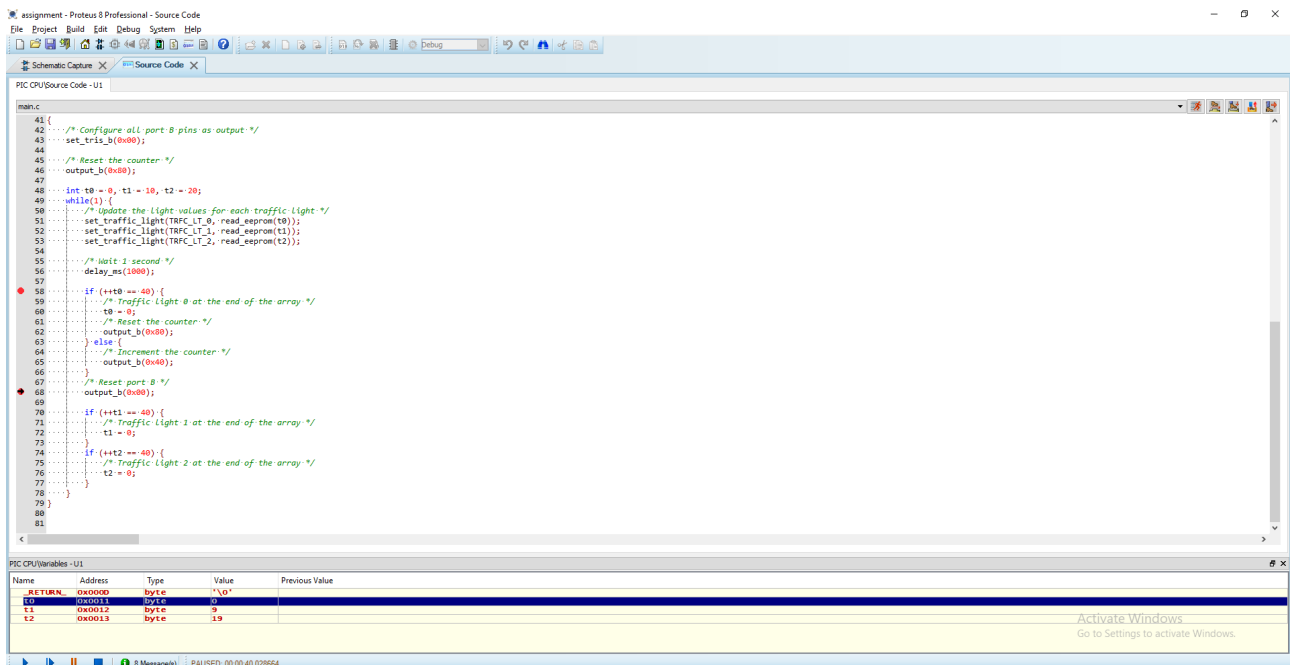


Figure 24: T0 after cycle reset

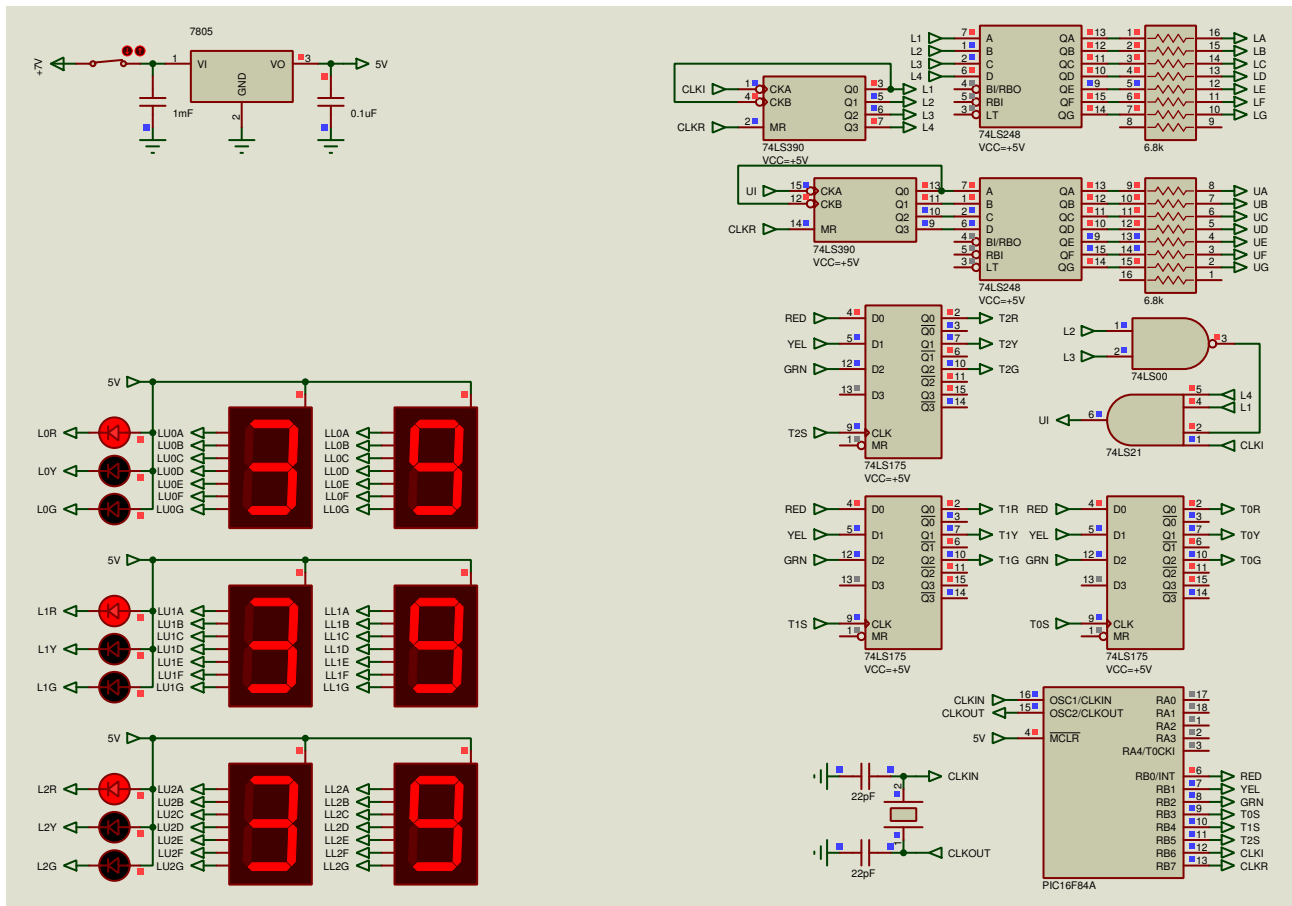


Figure 25: Circuit before T0 cycle reset

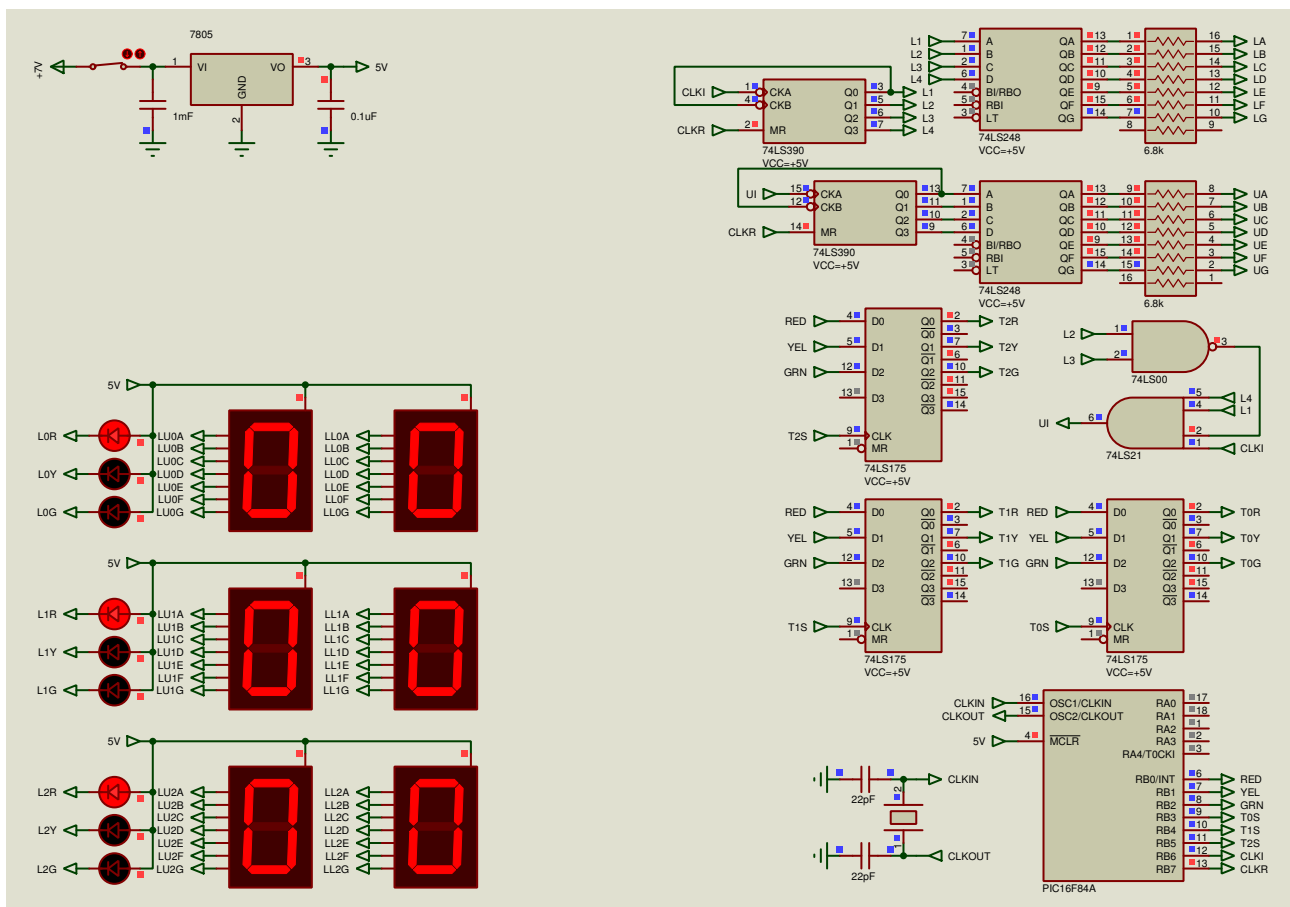


Figure 26: Circuit after T0 cycle reset

6 Improvements

Though the simulation meets the requirements of the tasking, it does not follow any of the guidance set out by the Department of Transport [14].

6.1 Timings

Following the signal sequence guidance [14, p. 37], The amber signal is displayed for 3 s and the red/amber signal is displayed for 2 s. To accommodate these changes, each traffic light sequence was increased from 10 s to 20 s.

6.1.1 Software

Due to the limitation of the micro-controller that was used having only 64 B of ROM - this being too small to store the 80 long sequence - the program has to be redesigned. The new program is shown in Algorithm 5.

Algorithm 5 Timings adjusted Program

```
trffc_lts_state ← [
  'RED/YELLOW', 'RED/YELLOW', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN',
  'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'YELLOW', 'YELLOW', 'YELLOW', 'RED'
]
procedure MAIN
  configure  $PORTB_{0:7} \leftarrow$  as outputs
  SET_TRFFC_LTS( $S0, RED$ )
  SET_TRFFC_LTS( $S1, RED$ )
  SET_TRFFC_LTS( $S1, RED$ )
  loop
    for  $t \leftarrow \{S0, S1, S2\}$  do
      for  $i \leftarrow 0, \dots, 19$  do
        SET_TRFFC_LTS( $t, trffc\_lts\_state[i]$ )
        delay_ms(1000)
         $PORTB_6 \leftarrow 1$ 
         $PORTB_6 \leftarrow 0$ 
      end for
    end for
    for  $i \leftarrow 0, \dots, 19$  do
      delay_ms(1000)
       $PORTB_6 \leftarrow 1$ 
       $PORTB_6 \leftarrow 0$ 
    end for
     $PORTB_7 \leftarrow 1$ 
     $PORTB_7 \leftarrow 0$ 
  end loop
end procedure
```

The source code can be found in Listing 2.

6.2 Pedestrian Controls

A typical traffic light system does not have a fixed delay period for pedestrians. Usually, a pelican or puffin crossing is utilised to allow pedestrians to press a button to request to cross the road.

6.2.1 Hardware

Figure 27 and 28 show the updated schematic with the additional buttons and lights.

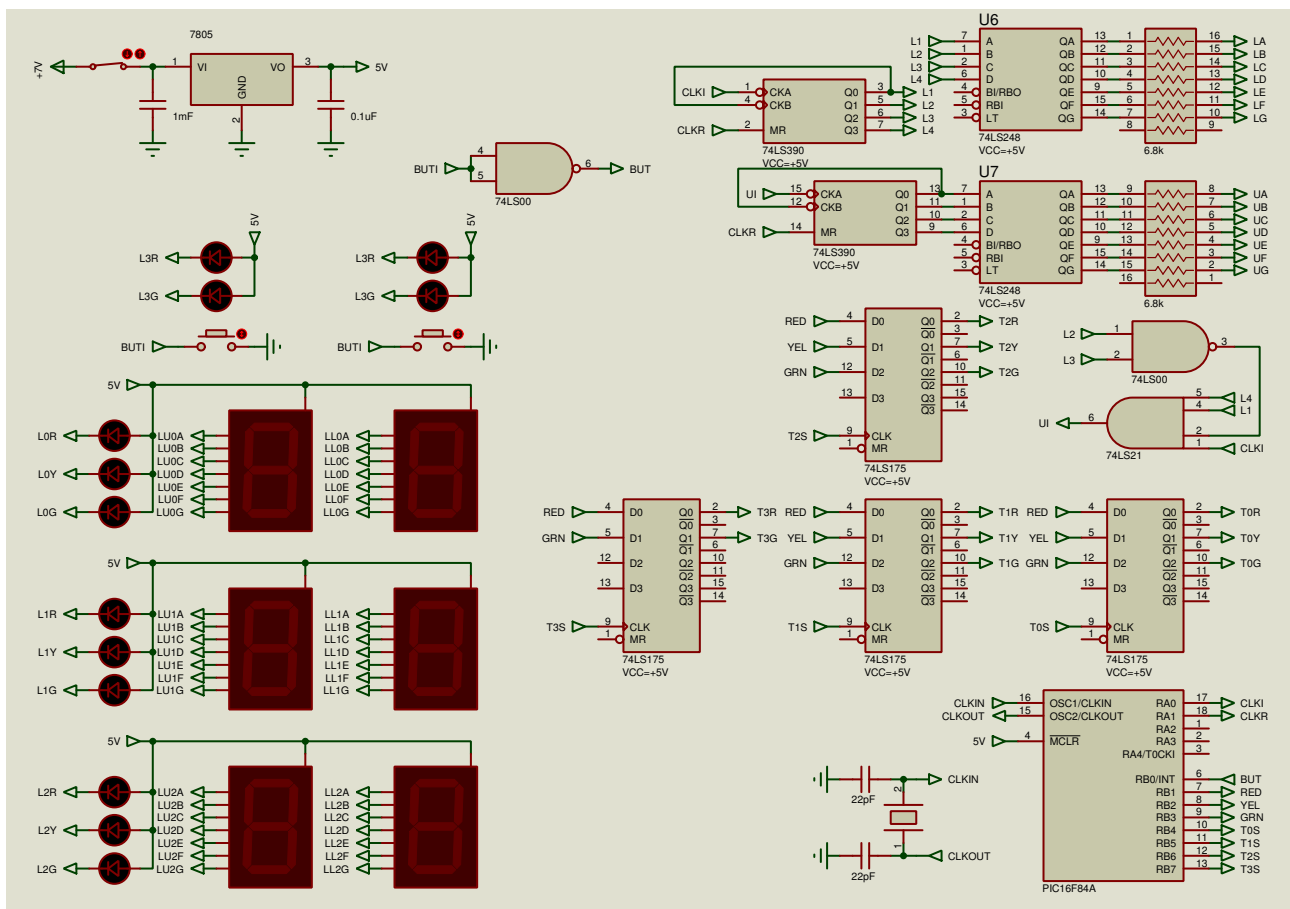


Figure 27: Sheet 0 of the schematic with the pedestrian controls

The pedestrians can press either of the two buttons, one on either side of the crossing. Figure 29 shows the implementation of the buttons.

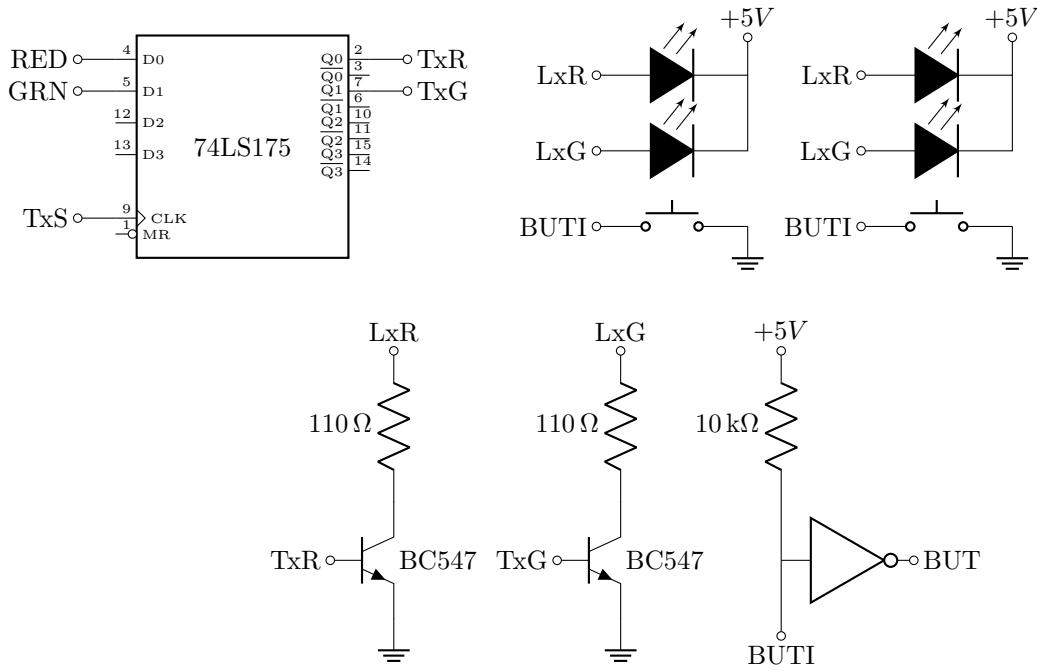


Figure 29: Circuit diagram of the buttons

Both buttons are connected to pin RB0. In this configuration, the resistor pulls the signal high when the button is not pressed, and shorts to ground when pressed. Without this setup, the input to RB0 would be floating at an unknown voltage when the buttons are not pressed. The 10 k Ω resistor limits the current shorted to ground to 0.5 mA.

6.2.2 Software

Connecting the buttons to RB0 allows the external interrupt to be used. The interrupt is triggered on the rising edge of the input. When an interrupt is triggered and the current traffic light phase finishes, all three traffic lights go red and the pedestrian lights go green. The interrupt and main program are shown in Algorithm 6 and 7 respectively.

Algorithm 6 Pedestrian Button Program External Interrupt

pressed $\leftarrow$ 1

The source code can be found in Listing 3.

Algorithm 7 Pedestrian Button Program

```
trffc_lts_state  $\leftarrow$  [  
    'RED/YELLOW', 'RED/YELLOW', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN',  
    'GREEN', 'GREEN', 'GREEN', 'GREEN', 'GREEN', 'YELLOW', 'YELLOW', 'YELLOW', 'RED'  
]  
pressed  $\leftarrow$  0  
procedure MAIN  
    configure PORTB0  $\leftarrow$  as an input  
    configure PORTB1:7  $\leftarrow$  as outputs  
    configure PORTA0:7  $\leftarrow$  as outputs  
    Enable External Interrupt  
    SET_TRFFC_LTS(S0, RED)  
    SET_TRFFC_LTS(S1, RED)  
    SET_TRFFC_LTS(S1, RED)  
    loop  
        for t  $\leftarrow$  {S0, S1, S2} do  
            for i  $\leftarrow$  0, ..., 19 do  
                SET_TRFFC_LTS(t, trffc_lts_state[i])  
                delay_ms(1000)  
                PORTA0  $\leftarrow$  1  
                PORTA0  $\leftarrow$  0  
            end for  
        end for  
        if pressed = 1 then  
            SET_TRFFC_LTS(S3, GREEN)  
            for i  $\leftarrow$  0, ..., 19 do  
                delay_ms(1000)  
                PORTA0  $\leftarrow$  1  
                PORTA0  $\leftarrow$  0  
            end for  
            SET_TRFFC_LTS(S3, RED)  
            pressed  $\leftarrow$  0  
        end if  
        PORTA1  $\leftarrow$  1  
        PORTA1  $\leftarrow$  0  
    end loop  
end procedure
```

7 Conclusion

The simulation demonstrates a minimal viable design for a prototype traffic light system. With the improvements, the simulation has started to become a more complete model of a real set of traffic lights.

8 Recommendations

Furthering what is described in Improvements, additional modifications could be made to improve the realism of the simulation. Some of the modifications that could be made are:

- Replace the LED traffic lights with proper bulbs
- Add a count down for pedestrians so they know how long they have left to cross
- Add sensors to detect if a vehicle is waiting at a particular side and prioritise them over a side with no vehicle

9 References

- [1] *The Official Highway Code*, 17th ed., Department of Transport, 2022.
- [2] *UM10204 I²C-bus specification and user manual*, 7th ed., NXP Semiconductors, 2021. [Online]. Available: <https://web.archive.org/web/20210813122132/https://www.nxp.com/docs/en/user-guide/UM10204.pdf>, (accessed: 22.04.2025).
- [3] *SCC2691 Universal asynchronous receiver/transmitter (UART)*, Philips Semiconductors, 2004. [Online]. Available: <https://www.nxp.com/docs/en/data-sheet/SCC2691.pdf>, (accessed: 22.04.2025).
- [4] *SPI Block Guide V03.06*, Motorola, 2003. [Online]. Available: <https://web.archive.org/web/20150413003534/http://www.ee.nmt.edu/~teare/ee3081/datasheets/S12SPIV3.pdf>, (accessed: 22.04.2025).
- [5] *SN54174, SN54175, SN54LS174, SN54LS175, SN54S174, SN54S175, SN74174, SN74175, SN74LS174, SN74LS175, SN74S174, SN74S175 HEX/QUADRUPLE D-TYPE FLIP-FLOPS WITH CLEAR*, Texas Instruments, 2001. [Online]. Available: <https://www.ti.com/lit/ds/symlink/sn74ls175.pdf>, (accessed: 15.04.2025).
- [6] *BC546 / BC547 / BC548 / BC549 / BC550 – NPN Epitaxial Silicon Transistor*, Fairchild, 2014. [Online]. Available: <https://www.mouser.com/datasheet/2/149/BC547-190204.pdf>, (accessed: 15.04.2025).
- [7] *SN54246, SN54247, SN54LS247, SN54LS248, SN74246, SN74247, SN74LS247, SN74LS248 BCD-TO-SEVEN SEGMENT DECODERS/DRIVERS*, Texas Instruments, 1988. [Online]. Available: <https://www.ti.com/lit/ds/symlink/sn74ls247.pdf>, (accessed: 15.04.2025).
- [8] *SN54390, SN54LS390, SN54393, SN54LS393, SN74390, SN74LS390, SN74393, SN74LS393, DUAL 4-BIT DECADE AND BINARY COUNTERS*, Texas Instruments, 1988. [Online]. Available: <https://www.ti.com/lit/gpn/sn74ls390>, (accessed: 15.04.2025).
- [9] *HDSP-550x, HDSP-552x, HDSP-560x, HDSP-562x, HDSP-570x, HDSP-572x, HDSP-H15x Series*, Broadcom, 2024. [Online]. Available: <https://docs.broadcom.com/doc/AV02-1107EN>, (accessed: 08.03.2025).
- [10] *PIC16F84A Data Sheet*, Microchip, 2001. [Online]. Available: <https://ww1.microchip.com/downloads/en/devicedoc/35007b.pdf>, (accessed: 08.03.2025).
- [11] *LM340, LM340A and LM7805 Family Wide V_{IN} 1.5-A Fixed Voltage Regulators*, Texas Instruments, 2000. [Online]. Available: <https://www.ti.com/lit/ds/symlink/lm7800.pdf>, (accessed: 15.04.2025).
- [12] “Ngspice, the open source Spice circuit simulator,” ngspice. (2025), [Online]. Available: <https://ngspice.sourceforge.io/>. (accessed: 02.04.2025).
- [13] “Code Optimizing C Compilers for Microchip PIC® and dsPIC® DSCs,” Custom Computer Services. (2025), [Online]. Available: <https://www.ccsinfo.com/compilers.php>. (accessed: 10.04.2025).
- [14] *Traffic Signs Manual Chapter 6 Traffic Control*, Department of Transport, 2019. [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/851465/dft-traffic-signs-manual-chapter-6.pdf, (accessed: 24.04.2025).

A C Code

A.1 Base Assignment

```
1  #include <16F84A.h>
2
3  #case
4  #fuses NOWDT
5  #use delay(crystal=1MHz)
6
7  enum traffic_light {
8      TRFC_LT_0 = 0x08,
9      TRFC_LT_1 = 0x10,
10     TRFC_LT_2 = 0x20
11 };
12
13 enum light_val {
14     LT_NONE = 0x00,
15     LT_R = 0x01,
16     LT_Y = 0x02,
17     LT_G = 0x04
18 };
19
20 #rom 0x2100={
21     LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R,
22     LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R,
23     LT_Y, LT_G, LT_G, LT_G, LT_G, LT_G, LT_G, LT_G, LT_G, LT_Y,
24     LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R, LT_R
25 }
26
27 /*
28  * Modify the light values for a set of traffic lights.
29  */
30 void set_traffic_light(enum traffic_light lt, enum light_val val)
31 {
32     /* Present the new light values */
33     output_b(val);
34     /* Update the light values for the selected traffic lights */
35     output_b(lt|val);
36     /* Lock in the values for the selected traffic lights */
37     output_b(val);
38 }
39
40 void main(void)
41 {
42     /* Configure all port B pins as output */
43     set_tris_b(0x00);
44
45     /* Reset the counter */
46     output_b(0x80);
47
48     int t0 = 0, t1 = 10, t2 = 20;
49     while(1) {
50         /* Update the light values for each traffic light */
51         set_traffic_light(TRFC_LT_0, read_eeprom(t0));
52         set_traffic_light(TRFC_LT_1, read_eeprom(t1));
53         set_traffic_light(TRFC_LT_2, read_eeprom(t2));
54
55         /* Wait 1 second */
56         delay_ms(1000);
57
58         if (++t0 == 40) {
59             /* Traffic light 0 at the end of the array */
60             t0 = 0;
61             /* Reset the counter */
62             output_b(0x80);
```

```

63     } else {
64         /* Increment the counter */
65         output_b(0x40);
66     }
67     /* Reset port B */
68     output_b(0x00);
69
70     if (++t1 == 40) {
71         /* Traffic light 1 at the end of the array */
72         t1 = 0;
73     }
74     if (++t2 == 40) {
75         /* Traffic light 2 at the end of the array */
76         t2 = 0;
77     }
78 }
79 }

```

Listing 1: C code for the base assignment

A.2 Timings

```
1  #include <16F84A.h>
2
3  #case
4  #fuses NOWDT
5  #use delay(crystal=1MHz)
6
7  enum traffic_light {
8      TRFC_LT_0 = 0x08,
9      TRFC_LT_1 = 0x10,
10     TRFC_LT_2 = 0x20
11 };
12
13 enum light_val {
14     LT_NONE = 0x00,
15     LT_R = 0x01,
16     LT_Y = 0x02,
17     LT_RY = 0x03,
18     LT_G = 0x04
19 };
20
21 #rom 0x2100={
22     LT_RY, LT_RY, LT_G, LT_G, LT_G, LT_G, LT_G, LT_G, LT_G, LT_G,
23     LT_G, LT_G, LT_G, LT_G, LT_G, LT_G, LT_Y, LT_Y, LT_Y, LT_R
24 }
25
26 /*
27  * Modify the light values for a set of traffic lights.
28  */
29 void set_traffic_light(enum traffic_light lt, enum light_val val)
30 {
31     /* Present the new light values */
32     output_b(val);
33     /* Update the light values for the selected traffic lights */
34     output_b(lt|val);
35     /* Lock in the values for the selected traffic lights */
36     output_b(val);
37 }
38
39 void main(void)
40 {
41     /* Configure all port B pins as output */
42     set_tris_b(0x00);
43
44     /* Clear the traffic lights */
45     set_traffic_light(TRFC_LT_0|TRFC_LT_1|TRFC_LT_2, LT_R);
46
47     while (1) {
48         for (traffic_light t = TRFC_LT_0; t != TRFC_LT_2<<1; t <= 1) {
49             for (int i = 0; i != 20; ++i) {
50                 /* Update the light value for the current traffic light */
51                 set_traffic_light(t, read_eeprom(i));
52
53                 /* Wait 1 second */
54                 delay_ms(1000);
55
56                 /* Increment the counter */
57                 output_b(0x40);
58                 output_b(0x00);
59             }
60         }
61
62         /* Allow pedestrians to cross */
63         for (int i = 0; i != 20; ++i) {
64             delay_ms(1000);
```

```
65  
66         output_b(0x40);  
67         output_b(0x00);  
68     }  
69  
70     /* Reset the counter */  
71     output_b(0x80);  
72     output_b(0x00);  
73 }  
74 }
```

Listing 2: C code for the correct timings

A.3 Pedestrian Buttons

```
1 #include <16F84A.h>
2
3 #case
4 #fuses NOWDT
5 #use delay(crystal=1MHz)
6 #use fast_io(ALL)
7
8 enum traffic_light {
9     TRFC_LT_0 = 0x10,
10    TRFC_LT_1 = 0x20,
11    TRFC_LT_2 = 0x40,
12    TRFC_LT_3 = 0x80
13 };
14
15 enum light_val {
16     LT_NONE = 0x00,
17     LT_R = 0x02,
18     LT_Y = 0x04,
19     LT_RY = 0x06,
20     LT_G = 0x08
21 };
22
23 #rom 0x2100={
24     LT_RY,LT_RY, LT_G, LT_G, LT_G, LT_G, LT_G, LT_G, LT_G, LT_G,
25     LT_G, LT_G, LT_G, LT_G, LT_G, LT_G, LT_Y, LT_Y, LT_Y, LT_R
26 }
27
28 int pressed = 0;
29
30 #INT_EXT
31 void EXT_isr(void)
32 {
33     /* External button was pressed */
34     pressed = 1;
35 }
36
37 /*
38  * Modify the light values for a set of traffic lights.
39  */
40 void set_traffic_light(enum traffic_light lt, enum light_val val)
41 {
42     /* Present the new light values */
43     output_b(val);
44     /* Update the light values for the selected traffic lights */
45     output_b(lt|val);
46     /* Lock in the values for the selected traffic lights */
47     output_b(val);
48 }
49
50 void main(void)
51 {
52     /* Configure pin 1 as an input and the rest of port B's pins as output */
53     set_tris_b(0x01);
54     /* Configure all port A pins as output */
55     set_tris_a(0x00);
56
57     /* Configure the external interrupt for the pedestrian button */
58     enable_interrupts(INT_EXT);
59     enable_interrupts(GLOBAL);
60
61     /* Clear the traffic lights */
62     set_traffic_light(TRFC_LT_0|TRFC_LT_1|TRFC_LT_2|TRFC_LT_3, LT_R);
63
64     while (1) {
```

```

65     for (traffic_light t = TRFC_LT_0; t != TRFC_LT_2<<1; t <= 1) {
66         for (int i = 0; i != 20; ++i) {
67             /* Update the light value for the current traffic light */
68             set_traffic_light(t, read_eeprom(i));
69
70             /* Wait 1 second */
71             delay_ms(1000);
72
73             /* Increment the counter */
74             output_a(0x01);
75             output_a(0x00);
76         }
77         if (pressed) {
78             /* The pedestrian button was pressed */
79
80             /* Allow the pedestrians to walk */
81             set_traffic_light(TRFC_LT_3, LT_G);
82             for (int i = 0; i != 20; ++i) {
83                 /* Wait 1 second */
84                 delay_ms(1000);
85
86                 /* Increment the counter */
87                 output_a(0x01);
88                 output_a(0x00);
89             }
90             /* Stop the pedestrians from walking */
91             set_traffic_light(TRFC_LT_3, LT_R);
92
93             /* Clear the flag */
94             pressed = 0;
95         }
96     }
97
98     /* Reset the counter */
99     output_a(0x02);
100    output_a(0x00);
101 }
102 }

```

Listing 3: C code for the pedestrian buttons