

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

Electronics 1

ACDB5002

**BEng Embedded
Electronic Systems**

Samuel Hunt
2315404

Contents

1	Introduction	3
2	Background research	4
2.1	Government regulations	4
2.2	Case study	4
2.3	External specifications	5
3	Circuit & software design	6
3.1	Initial circuit design	6
3.2	Initial software design	6
3.3	Minimum viable product	7
4	Design alterations	9
4.1	Enable pin changes	9
4.2	External interrupts	10
4.3	Additional timer	11
4.4	Buzzer indicator	12
4.5	Light control optimisation	13
4.6	Pedestrian lights	14
4.7	High current drivers	14
5	Final design, testing & results	15
5.1	Test plan	15
5.1.1	Seven segment driver	15
5.1.2	Traffic light driver	16
5.1.3	Additional timer	16
5.1.4	Pedestrian lights & input	17
5.1.5	Software	17
5.2	Final design	18
6	Conclusion	19
A	Initial code	20
B	Final code	25

List of Figures

1	Light cycle [1]	4
2	Initial flowchart	6
3	Initial circuit design	7
4	Initial circuit design working	8
5	Altered enable gate	9
6	Astable 555 timer	11
7	Buzzer circuit	12
8	Line driver circuit	13
9	Pedestrian lights	14
10	High current relays	14
11	Final design	18

1 Introduction

In this paper, the process of designing an automatic traffic light controller will be detailed. This will include the specification, design and testing of the lights to ensure that they fulfil any requirements as set out by government or industry bodies. As such, research will take place to inform the design process and aid in the ideation of the circuits involved. Case studies of existing products will be used too, to inform the decisions about how the product could compete with other companies. Next, a minimum viable product will be created for use in testing and a baseline for further modules to be added. Then, the additional modules can be developed and testing can start.

2 Background research

2.1 Government regulations

The initial sections of research was dedicated to government regulations which would allow for the basis of the traffic light algorithms to be created.

Chapter 6 of the traffic signs manual eludes to the terminology which will be used henceforth. Stages will be the overarching events (i.e. control of the main through fair), within which there are phases, which will denote the individual events which may occur (i.e. a light cycling) [1]. Furthermore, cycles will form a constant loop of stages. As per the manual, to adhere to the Traffic Signs Regulations and General Directions 2016 the red and amber lights must been shown for at least 2 seconds $\pm$ 0.25 seconds[2]. Also, generally should have 3 seconds of amber, 2 seconds of red then the next phase / stage beginning.

Figure 1 shows the cycle which should be followed for the lights in the initial stages of the design, which could then be followed by more advanced mechanisms if needed.

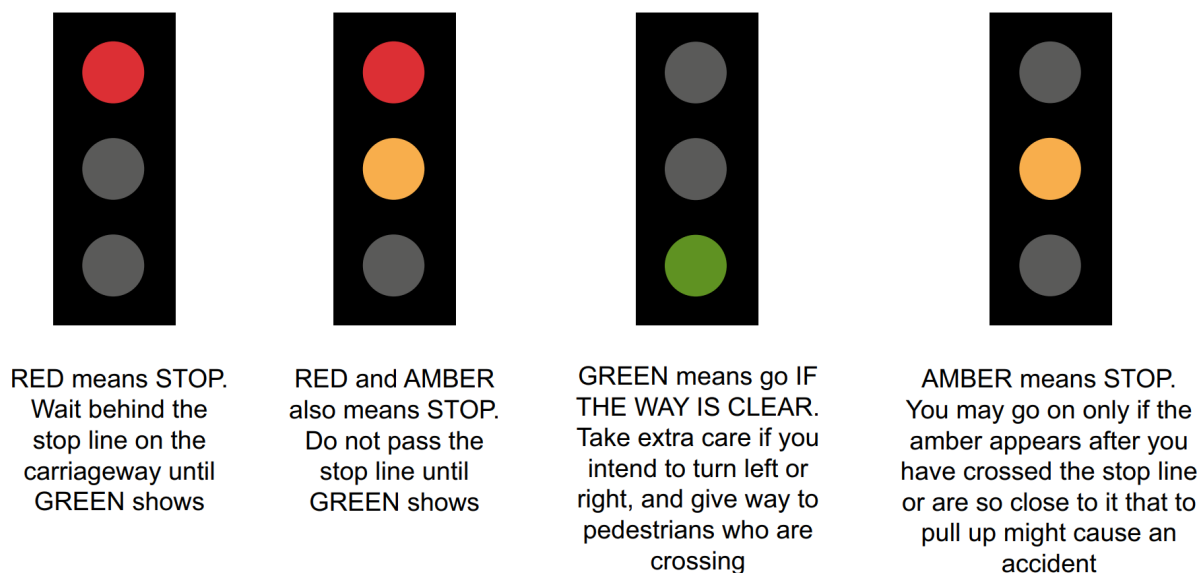


Figure 1: Light cycle [1]

2.2 Case study

Upon observing existing products, it's common to have one or more system, which can connect in a modular fashion. This allows for greater flexibility over competitors and increases the amount of applications the lights can be used for. For example, Swarco's systems, which often use rack mountable units of cabinets to contain their electronics. This allows for better integration with other systems and hot swapping components [3]. The United States (US) Department of Transport Federal Highway Administration (DoT-FHWA) states guidelines for the timing of lights dependent of the flow of traffic and light intervals. As this is not relevant to law in the United Kingdom (UK), this isn't binding but will give a good space to work within for timing the light sequence. From this, an estimated 26 seconds in a 50 second cycle could be allocated for one lane at 600 vehicles per hour per lane [4].

2.3 External specifications

According to the Traffic Open Products And Specifications (TOPAS) specifications for audible equipment for use at pedestrian crossings, a constant tone of between $2.0kHz$ and $3.5kHz \pm 2.5\%$ pulsed at $4Hz \pm 25\%$ should be used when audibly signalling a crossing is active. It should output the signal at between $47dB(A)$ and $83dB(A)$, allowing for adjustments depending on the location of the crossing. Furthermore, it should have the ability to adjust the sound level depending on the time, preventing disruption to the local population.

The software should be built according to the Rapid Application Development (RAD) model, this will involve creating multiple subsections of code, which will be combined to make a cohesive product [5].

3 Circuit & software design

3.1 Initial circuit design

At first, a general specification was drawn up to fulfil the given brief, which would allow a set of 3 traffic lights and associated counters to work. Due to the volume of pins required, hardware should be put in place to offload the amount of processing required by the microcontroller, as such potentially taking steps to reduce the amount of updates required to be issued by the microcontroller would be beneficial. As a result, additional chips for storage and updating the lights and seven segment displays would be ideal.

On initial inspection, there are dual seven segment displays with a select channel and the appropriate select pins were found. This would help to reduce the amount of pins required and offload the computing required.

3.2 Initial software design

Due to the accuracy required, it is likely that standard delay statements would be advantageous to use the inbuilt timer interrupts, which would be harder to deviate from the set timeframes, and will give a consistent base to toggle the lights from.

Additionally, a rough plan would be ideal, to flesh out the requirements that will be included. As shown in Figure 2.

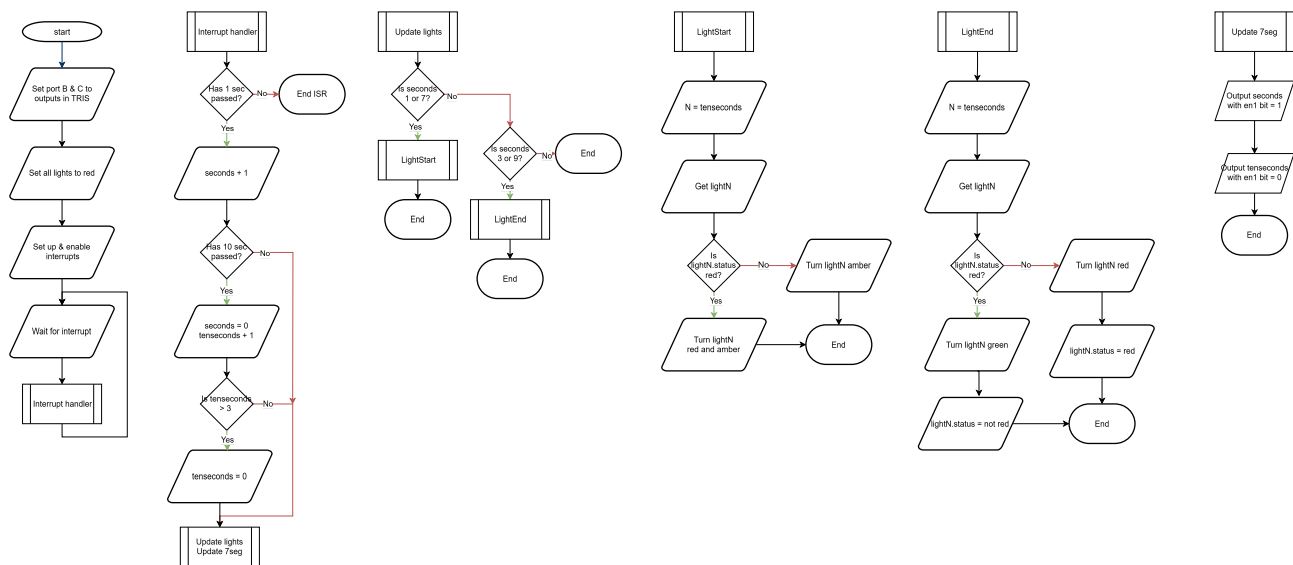


Figure 2: Initial flowchart

This is a vector graphic, you can zoom in to see it.

3.3 Minimum viable product

Figure 3 shows the circuit that was settled upon to fulfil the brief. It involves the aforementioned seven segment displays and the associated circuits. The circuits are required due to the display not latching, which means that the enable pins and input data have to be oscillated quickly to ensure that the display is readable. U7 and U3 are octa D-type flip flops, which store the data from the microcontroller through capturing the seven segment display digit when the clock pin is rising. This is beneficial as they can be clocked separately, allowing for the segments to be addressed separately.

U6 and U5 are used to output the digits to the displays. This is done so that the output can be enabled and disabled in time with the display being enabled. The displays are enabled through a singular buffer and not gate, which will take a pulsed signal and invert if needed. This is ideal as it can control both the output of the buffers and the enable pin of the displays. However, due to the enable pins on the display being not gated internally, the pulsed signal needs to be flipped. This was achieved through swapping the inputs from the buffers around.

It should be noted, in the future further reduction in the pins used could be achieved through using a not gate to invert En1, which would mean that En2 could be eliminated. Hence, further separating the ports that are used for separate functions.

The circuits used to control the traffic lights use a D-type flip flops (U8, 9 & 10), which can be addressed individually. Hence, allowing the microcontroller to broadcast to multiple lights at once or individually by pushing the required pattern to the D-type ports then clocking them. This was done to reduce the amount of pins required to control the lights to one port per overarching function. In addition, this allows for a more modular approach, which aligns the design to existing products and created a more versatile end product.

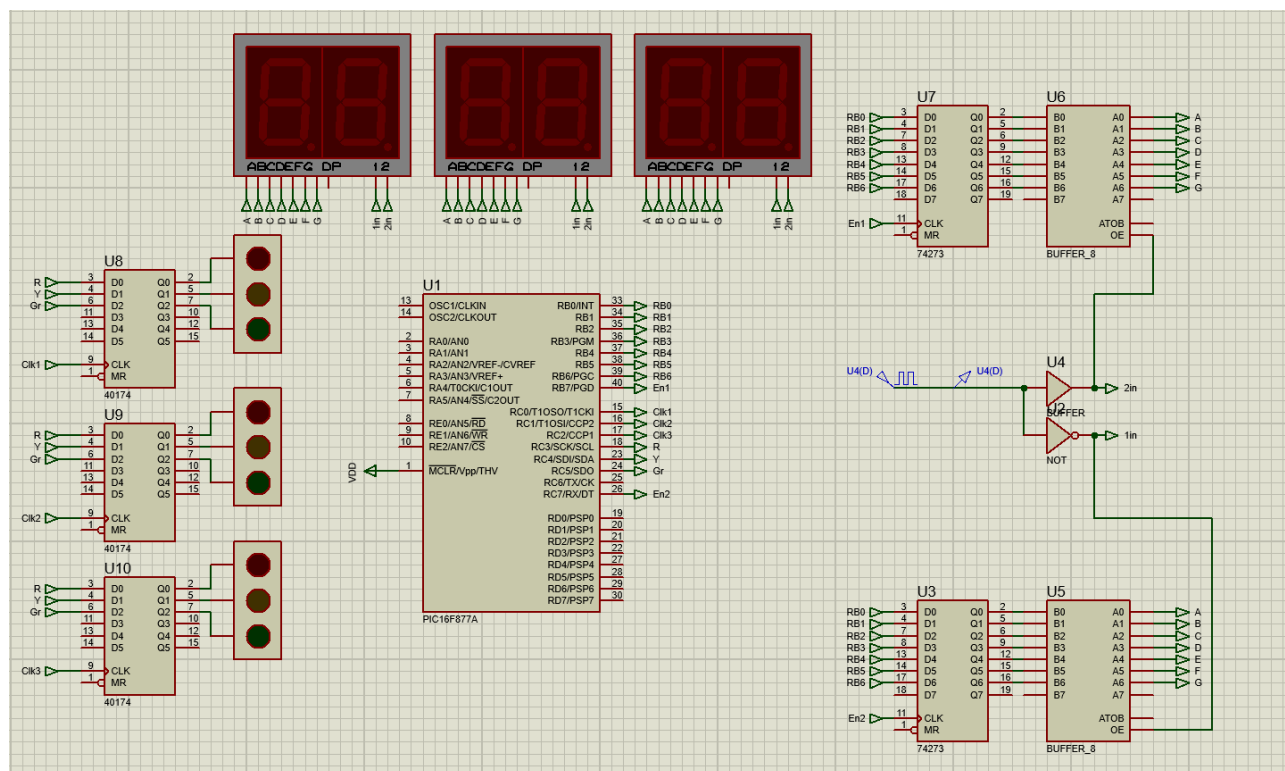


Figure 3: Initial circuit design

The full code for the microcontroller can be found in appendix A. The code will send updates to the screen and traffic lights every seconds, using a timer interrupt and counter. The interrupt takes ten milliseconds and the counter goes to one hundred, creating a one second interval, which will trigger the update functions. One the variable is greater than nine a second variable will be incremented. This will represent the first and second digit for the displays.

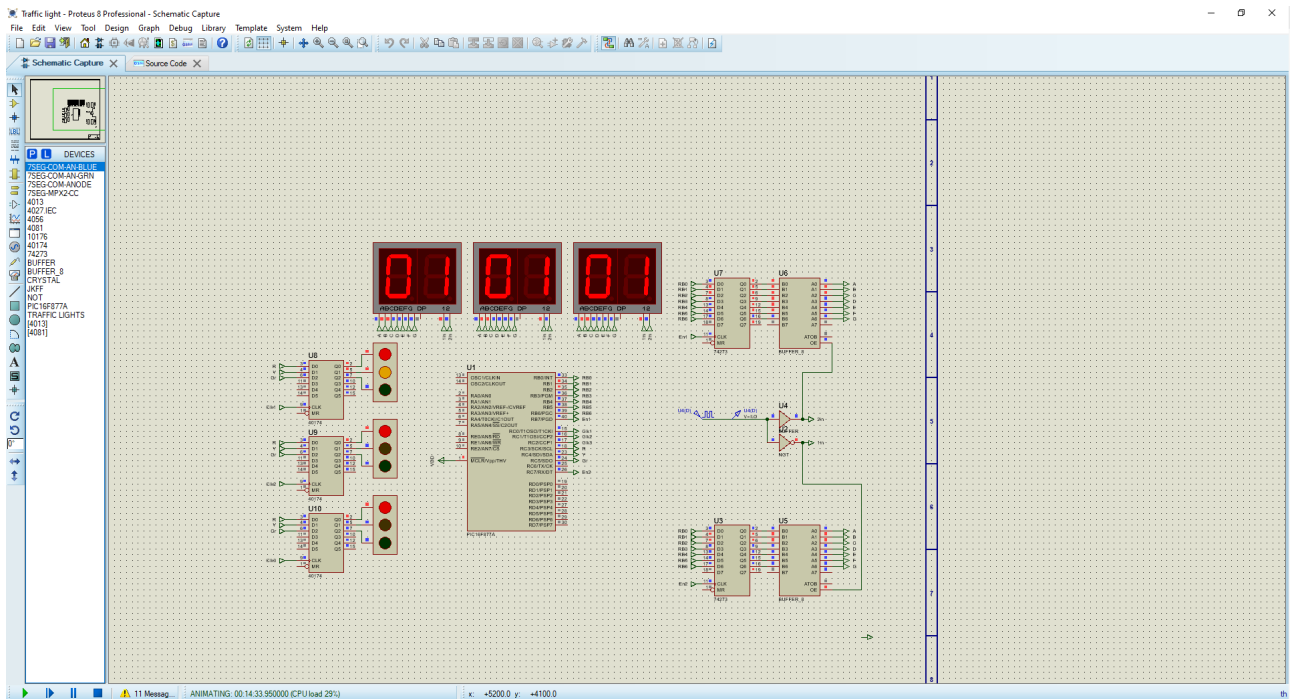


Figure 4: Initial circuit design working

4 Design alterations

4.1 Enable pin changes

As mentioned in Section 3.3, it would be ideal to further reduce the pins required to drive the seven segment displays a not gate has been installed to allow for control of both displays from one pin. This has freed up the final pin on port C, which could be used for further expansion. This was achieved through replacing the En2 on port C with a not gate on En1, port B. This will alternate the flip flop that is being addressed without the need for two ports to be written to.

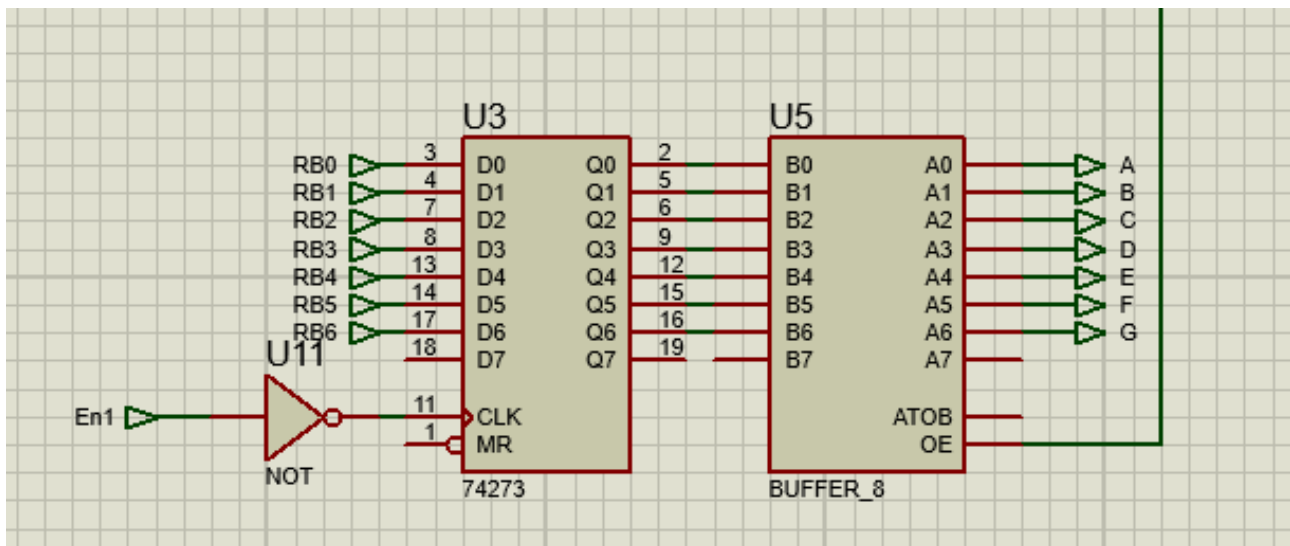


Figure 5: Altered enable gate

4.2 External interrupts

An external interrupt service routine would be ideal for a pedestrian crossing, however, due to the lack of support between the Timer2 interrupt currently used, a flag had to be used. This would still be polled at approximately one hundred times per second, which should be acceptable for the use case. To be able to create a pedestrian button, there needs to be a function which will handle turning the lights red and the ability to continue the cycle as normal once the routine has finished.

This could be achieved through pausing the counter and starting it again once it has completed. A snippet of the code used is shown below. The full code of which is accessible in appendix B.

```
1  /*****
2  * Function pex xing button
3  * Function which control routine outside of timing loop, used for pedestrian crossings
4  * Inputs: Current light which may be active / green
5  * Outputs: None
6  * Corrupts: W, selection, C ports, B ports
7  *****/
8  void pedXingButton (unsigned tens)
9  {
10     disable_interrupts(INT_TIMER2);          // Turns off timer interrupts
11     light_out* selection = findLight(tens);
12     if(!selection->isRed || selection->state == 2){
13         trafficLightDriverStart(selection);    // If light is amber or green, turn red
14         delay_ms(1000);
15         trafficLightDriverEnd(selection);
16     }
17     for (int secs = 10; secs > 0; secs--){
18         sevenSegDisplayDriver(secs/10,secs%10); // Cheap way of updating displays, using div and mod
19         delay_ms(1000);
20     }
21     enable_interrupts(INT_TIMER2);
22 }
```

4.3 Additional timer

To complete the realism of the circuit, it would be ideal to add an internal source for oscillating the display input buffers, which could be achieved through using a 555 timer. As shown in Figure 6. To create this an equation can be used to calculate the resistor values needed. Given as:

$$f = \frac{1.44}{(R_1 + 2R_2)C}$$

In this case, the frequency require is 10 and the capacitance can be set to an arbitrary number, like 1×10^{-6} . And grouping the resistors. From this the resistors required can be calculated:

$$10 = \frac{1.44}{3R \times 1 \times 10^{-6}}$$

$$R = \frac{1.44}{3 \times 10 \times 1 \times 10^{-6}}$$
$$= 48,000\Omega$$

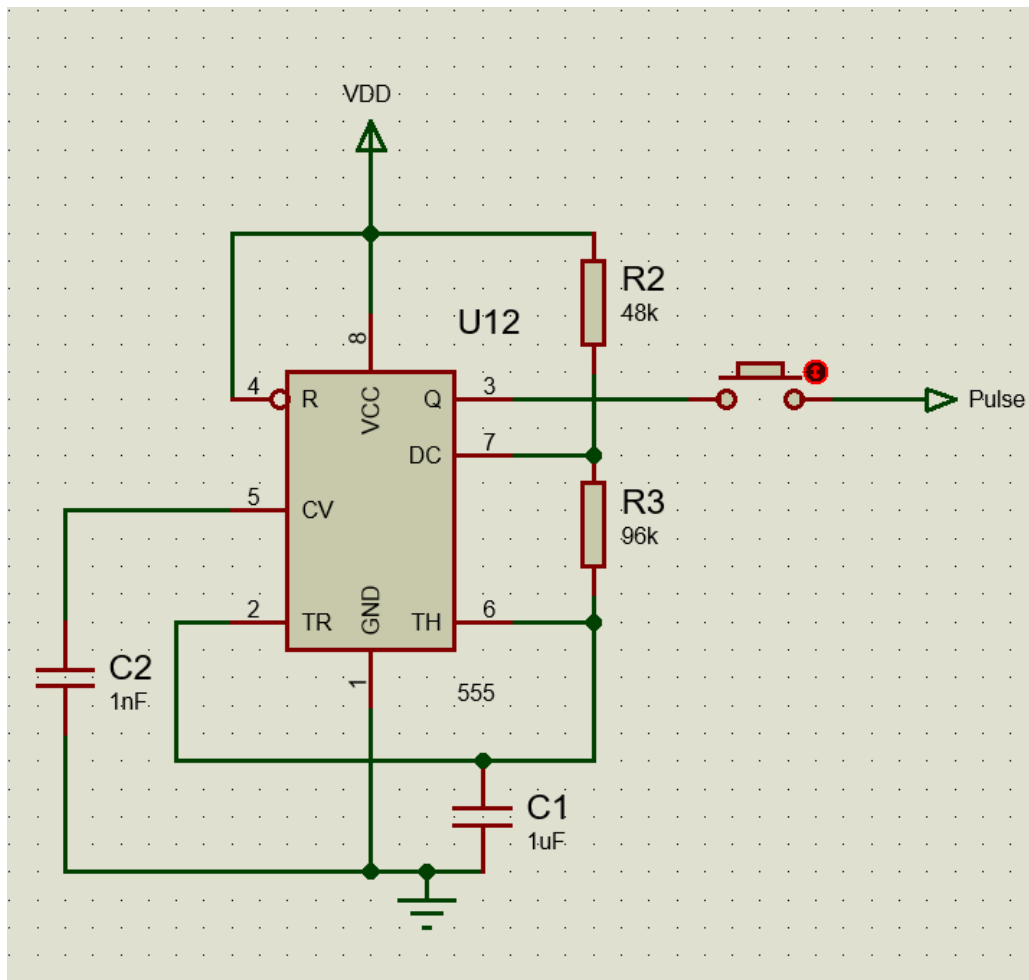


Figure 6: Astable 555 timer

Note, due to the simulator used, a button is required to prevent crashing. This can be omitted in a physical circuit.

4.4 Buzzer indicator

A buzzer to indicate the status of the lights would be ideal, for those who have limited or no sight to be able to see any other indicators, as such another medium would be ideal. In accordance with the research set out above, the buzzer will be set to output $2500Hz$ pulsing at $4Hz$. This will be integrated in to the for loop above, allowing for the output of sound.

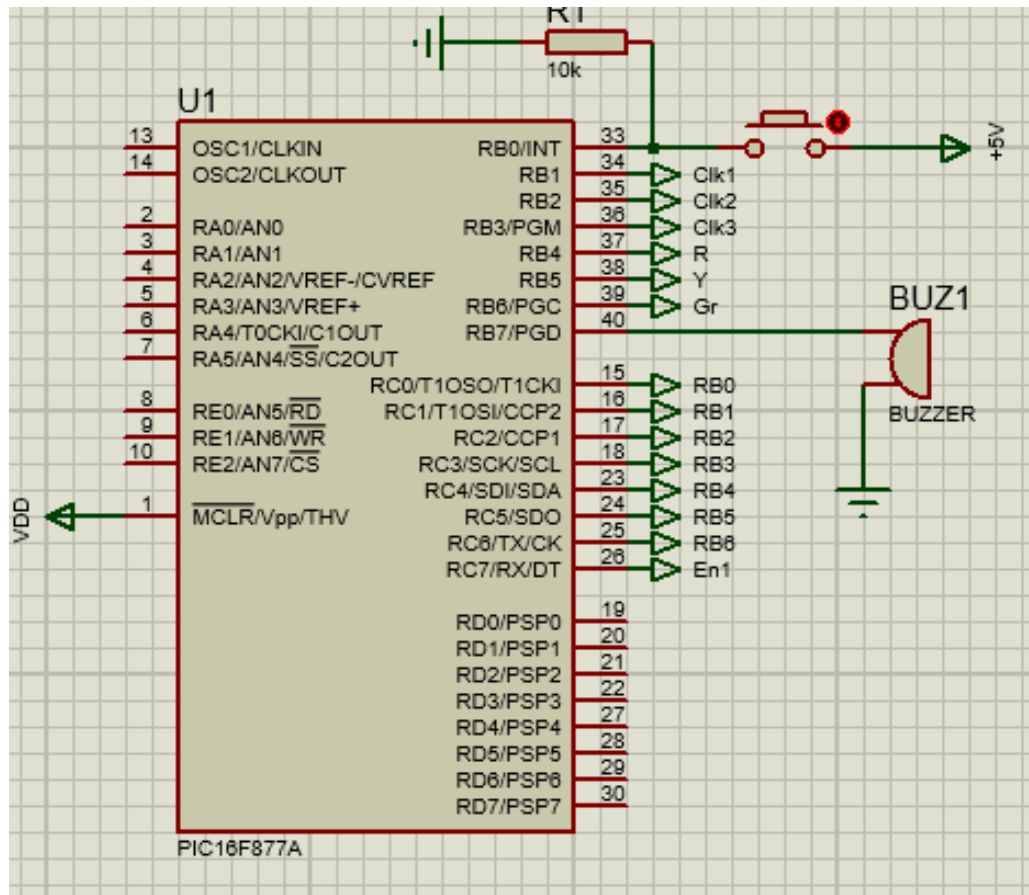


Figure 7: Buzzer circuit

4.5 Light control optimisation

To allow for the optimisation of pin usage, the clock (Clk) pins on port B could be condensed using a set of gates to ensure their functionality. This was achieved through using a two to four line decoder, which converts the binary input to a singular output, as such, expanding the range of ports that can be outputted to. This could be done with more pins if required. As shown it 8 the new implementation saves space for extra features. This would also be achievable with the lights if the red-amber pattern was omitted. The program was edited to only use the two pins required herein.

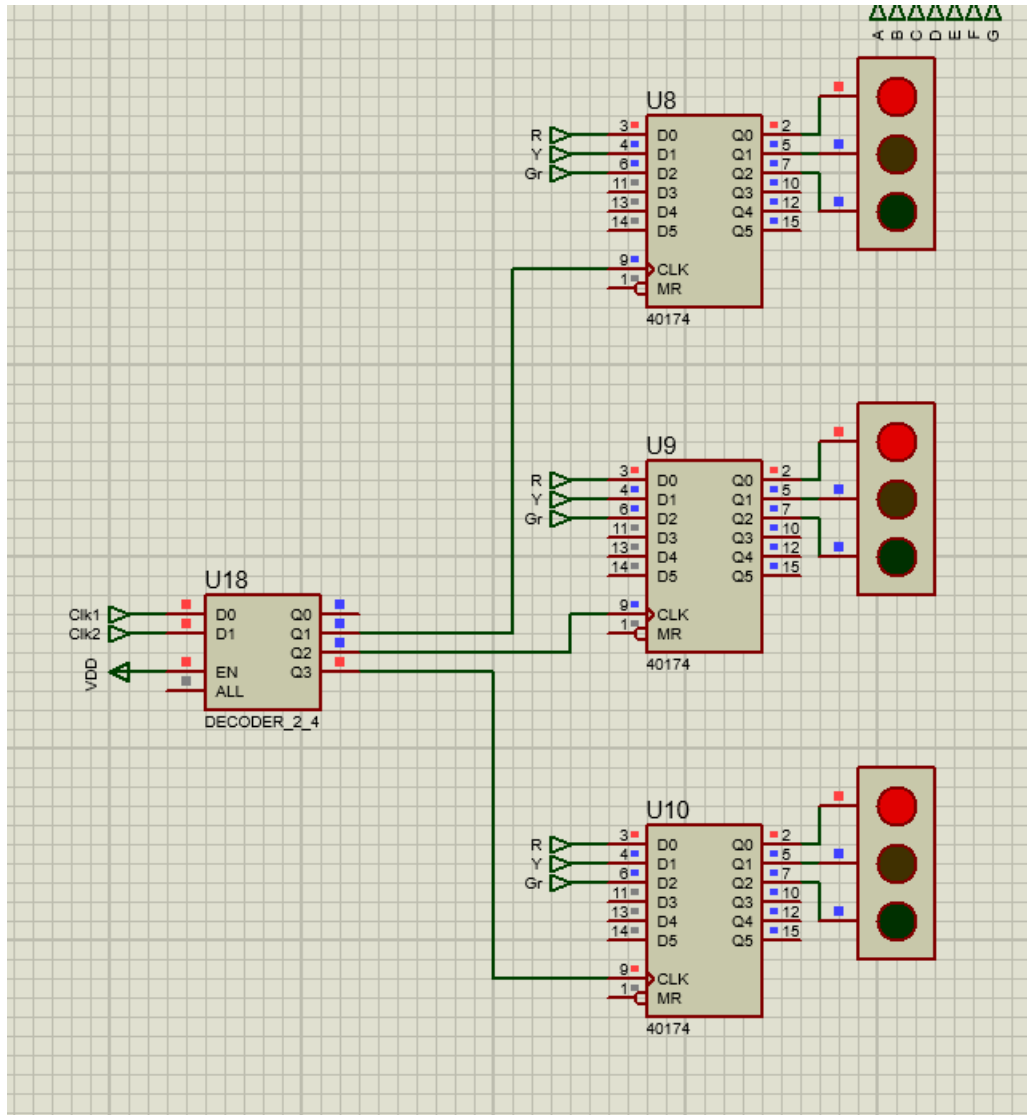


Figure 8: Line driver circuit

4.6 Pedestrian lights

In conjunction with the buzzer, lights would be ideal to improve the usability of the pedestrian crossing along with allowing people with impaired hearing to tell when they can cross. As such, the pin freed up by the line driver will be used for the light. This was achieved through using a not gate, which will control an alternative light. In addition to this, the code was edited to output a signal to the light when in use.

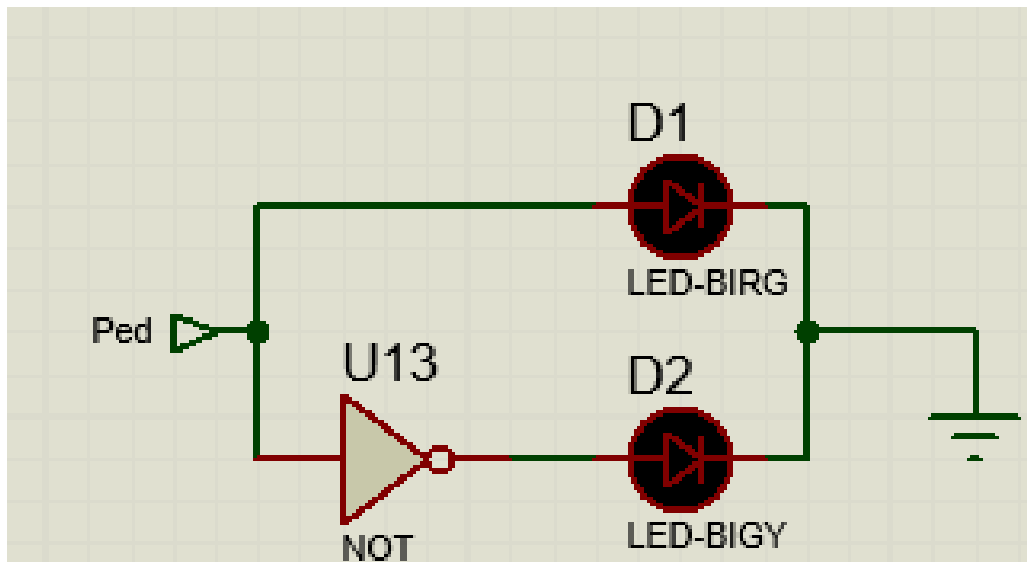


Figure 9: Pedestrian lights

4.7 High current drivers

In previous sections, the fact that the lights are likely to run off of higher current and voltage sources than currently used was omitted. As such, the improvement to make the design more realistic is to add the components required to driver a higher current load.

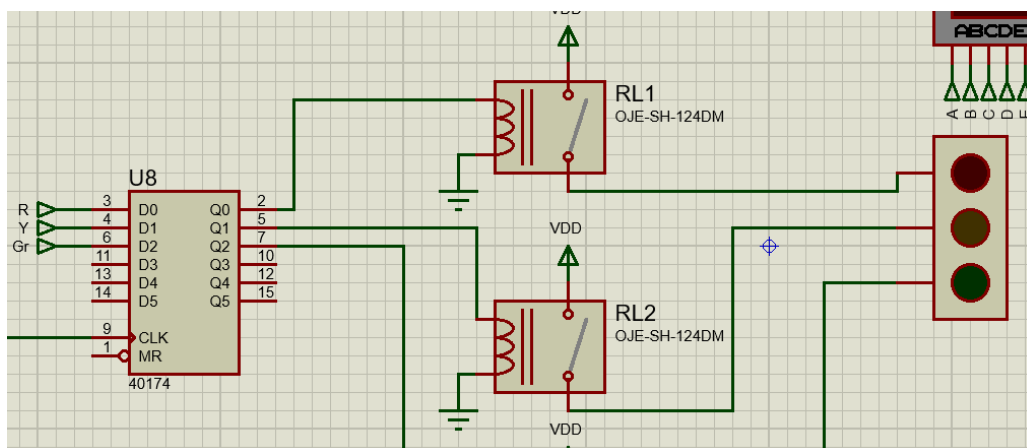


Figure 10: High current relays

5 Final design, testing & results

5.1 Test plan

As mentioned in section 2.3, a RAD methodology is being used. As such subsections are tested individually as they're being built. However, the results have been consolidated into this section for clarity. Each section hereafter will be used for the subsections in the circuit and its code.

5.1.1 Seven segment driver

The table below describes the tests carried out and the expected results:

Test	Method	Expected result	Simulated result
Output select voltages	Varying input voltages $5V \pm 25\%$	Operates at $5V \pm 25\%$	Operated correctly
Timer pulse input voltage	Varying the timer input voltage $5V \pm 25\%$	Operates at $5V \pm 25\%$	Operated normally
D-type flip flop input voltage	Verifying the input data voltage range	Operates at $5V \pm 25\%$	Operates at $5V \pm 10\%$
Maximum change frequency	Find the maximum frequency that data can be read at	Above $20Hz$	Operates up to $10kHz$
Maximum data output frequency	Ramp up input frequency until no proper output	Above $20Hz$	Operates up to $250Hz$

5.1.2 Traffic light driver

The table below describes the tests carried out and the expected results:

Test	Method	Expected result	Simulated result
Input 230V voltage response	Varying input response and noting changes to output	Operates at $230V \pm 5\%$	Operates at correct voltage range
Output select voltages	Varying input voltages $5V \pm 25\%$	Operates at $5V \pm 25\%$	Operated correctly
D-type flip flop input voltage (RYG input)	Verifying the input data voltage range	Operates at $5V \pm 25\%$	Operates at $5V \pm 10\%$
Maximum change frequency	Ramp up input frequency until no longer able to be read properly	Above $10Hz$	Operates up to $1MHz$ (unable to test further)
Relay voltage spike	Testing voltage spike due to relay operation	Doesn't disrupt circuit operation	Causes excessive voltage over buffer output chips; requires flyback diode

5.1.3 Additional timer

The table below describes the tests carried out and the expected results:

Test	Method	Expected result	Simulated result
Input voltage response	Varying input response and noting changes to output	Operates at $5V \pm 25\%$	Operates at $5V \pm 20\%$
Output impedance	Varying resistance of output for changes	Operates up to $10k\Omega$	Operated correctly until $12k\Omega$

5.1.4 Pedestrian lights & input

The table below describes the tests carried out and the expected results:

Test	Method	Expected result	Simulated result
Input voltage response	Varying input response and noting changes to output	Operates at $5V \pm 25\%$	Operates at $5V \pm 30\%$
Frequency response	Varying input frequency and observing normal operation	Operates above $5Hz$	Operates above $5kHz$

5.1.5 Software

The table below describes the tests carried out and the expected results:

Test	Method	Expected result	Simulated result
Interrupt routine return	Running interrupts and checking it returns to same cycle	Returns to same cycle	Returned to same cycle
Timing accuracy	Measuring the time using break points within software	Follows legislation set out in section 2.1	All timings correct, follows legislation
Interrupt routine response	Running interrupts and checking lights are handled correctly	Works at all times when activated	Produces very short cycles when pedestrian button is pressed at start of cycle
Switch bouncing prevention	Testing software response to high frequency noise	Ignores bounces and continues normal operation	Works normally

5.2 Final design

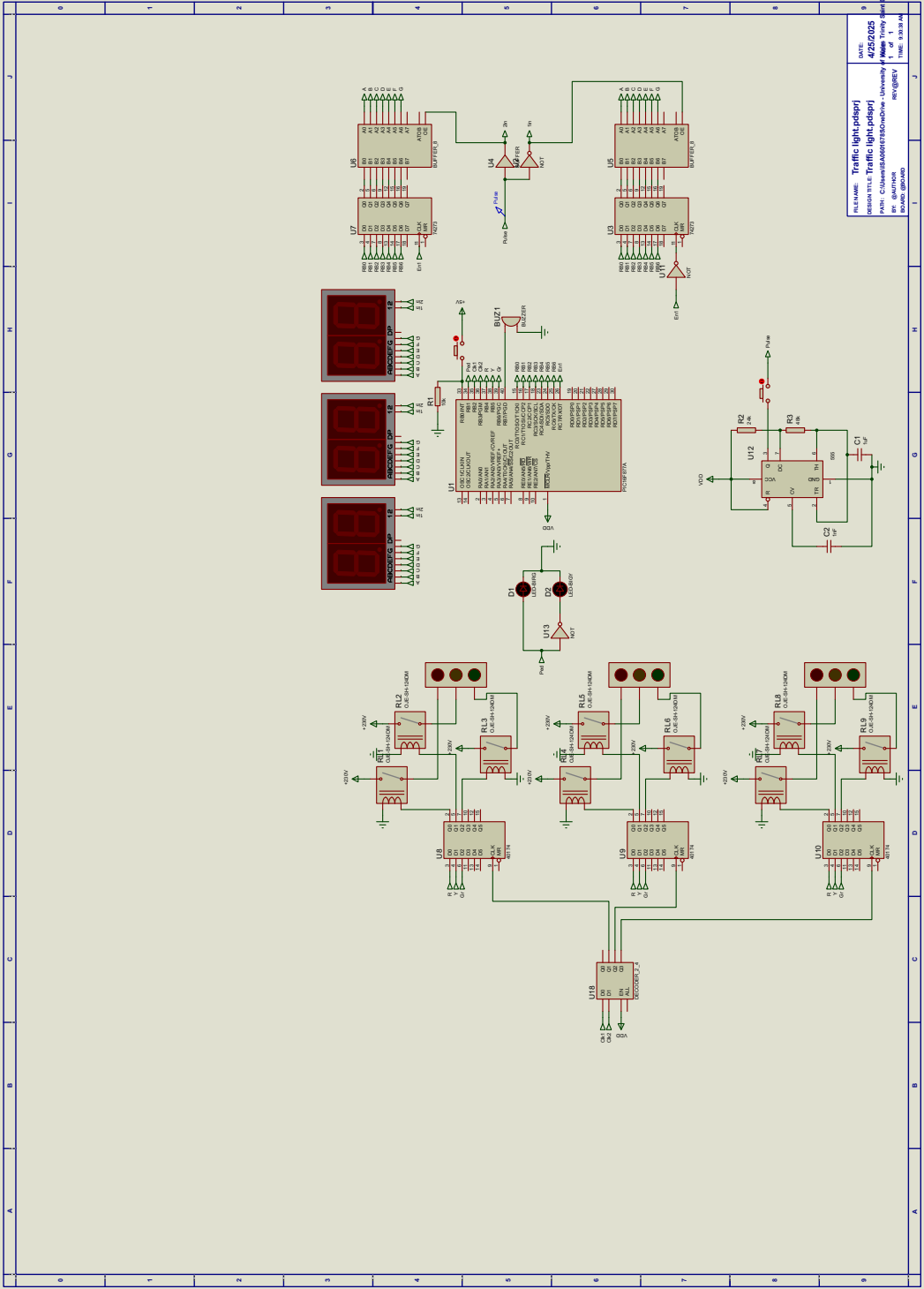


Figure 11: Final design

This is a vector graphic, zoom in to see it.

6 Conclusion

Overall, the design of the traffic lights and associated infrastructure has been a resounding success. The designs cover the brief and the regulations set out by government and industry standards. The circuits operates as it should, using the integrated timing circuits in the microcontroller and external astable circuit to ensure that the light's timing is as accurate as possible.

References

- [1] Department for Transport. Traffic signs manual chapter 6. <https://assets.publishing.service.gov.uk/media/5df0e29fed915d15f42c4820/dft-traffic-signs-manual-chapter-6.pdf>, 2019. Accessed: 27/2/25.
- [2] United Kingdom Government. The traffic signs regulations and general directions 2016. <https://www.legislation.gov.uk/uksi/2016/362/schedule/14>, 2016. Accessed: 27/2/25.
- [3] Swarco. Traffic light controllers — swarco. <https://www.swarco.com/products/traffic-light-controllers>. Accessed: 27/2/25.
- [4] United States Department of Transportation Federal Highway Administration. Traffic signal timing manual. <https://ops.fhwa.dot.gov/publications/fhwahop08024/chapter5.htm#5.2>, 2021. Accessed: 12/4/25.
- [5] Nik Marsyahariani Nik Daud, Nor Azila Awang Abu Bakar, and Hazlifah Mohd Rusli. Implementing rapid application development (rad) methodology in developing practical training application system. In *2010 International Symposium on Information Technology*, volume 3, pages 1664–1667, 2010.

Appendix

A Initial code

```
1  #include <16F877A.h>
2  #use delay(clock=20000000)
3
4  #define      REDAMBER      3
5  #define      RED           1
6  #define      AMBER        2
7  #define      GREEN        4
8
9  // Integers, which represent counter (for counting x many 10ms intervals), number of 10 seconds passed,
10 unsigned int countState = 0, secondCountTens = 0, secondCountOnes = 0;
11 unsigned segDisp[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67};
12
13
14
15 /*****
16  * Struct seven seg output struct
17  * Used to centralise the structure outputted to the displays, uses one of two bits to select which
18  *****/
19 struct seven_seg_output_struct{
20     unsigned int8      digits      :7;
21     unsigned int8      select     :1;
22 };
```

```

23 typedef struct seven_seg_output_struct b_out;
24 /*****
25  * Struct traffic lights output struct
26  * Intended order light to control, holds state of each light.
27  *****/
28 struct traffic_lights_output_struct{
29     unsigned int8      clkSel      :3;          // MSB, C0
30     unsigned int8      state       :3;
31     unsigned int8      isRed       :1;
32     unsigned int8      nouse       :1;          // LSB, C7
33 };
34 typedef struct traffic_lights_output_struct light_out;
35
36
37 light_out light1 = 0b01001001; // LSB = C7, MSB = C0 (nouse, isRed, 3xstate, 3xclockSelect)
38 light_out light2 = 0b01001010; //01001010
39 light_out light3 = 0b01001100; //01001100
40
41
42 /*****
43  * Function traffic light driver start
44  * Used for switching state of light from green to amber or red to red/amber.
45  * Inputs: Pointer to light_out object, light that will be changed
46  * Outputs: None
47  * Corrupts: W, selection, C ports
48  *****/
49 void trafficLightDriverStart (light_out* selection)
50 {
51     if (selection->isRed==1){
52         selection->state = REDAMBER;
53     } else{
54         selection->state = AMBER;
55     }
56     output_c(*selection); // Outputting twice, so that clock pin
57     output_c(*selection);
58 }
59
60 /*****
61  * Function traffic light driver end
62  * Used for switching state of light from amber to red and red/amber to green
63  * Inputs: Pointer to light_out object, light that will be changed
64  * Outputs: None
65  * Corrupts: W, selection, C ports
66  *****/
67 void trafficLightDriverEnd (light_out* selection){
68     if (selection->isRed==1){

```

```

69     selection->state = GREEN;
70     selection->isRed = 0;
71 } else{
72     selection->state = RED;
73     selection->isRed=1;
74 }
75 output_c(*selection);
76 output_c(*selection);
77 }
78
79 /*****
80  * Function find light
81  * Used to find light object that needs to be manipulated and returns it
82  * Inputs: Integer channel, represents which light need to be used
83  * Outputs: Pointer to light_out object
84  * Corrupts: W, selection
85  *****/
86 light_out* findLight (unsigned channel){
87     light_out* refToLight;
88     switch(channel){
89         case 0:
90             refToLight = &light1;
91             break;
92         case 1:
93             refToLight = &light2;
94             break;
95         case 2:
96             refToLight = &light3;
97             break;
98     }
99     return refToLight;
100 }
101
102 /*****
103  * Function light order driver
104  * Uses first and second digits of timing loop to determine if lights need to change
105  * Inputs: first digit, second digit from timing circuit, represents which light will be used and wh
106  * Outputs: None
107  * Corrupts: W, selection, C ports
108  *****/
109 void lightOrderDriver (firstDigit, secondDigit){
110     if (firstDigit == 1 || firstDigit == 8){ // If light is in f
111         light_out* refToLight = findLight(secondDigit);
112         trafficLightDriverStart(refToLight);
113     } else if (firstDigit == 2 || firstDigit == 9){ // Second half of fir
114         light_out* refToLight = findLight(secondDigit);

```

```

115     trafficLightDriverEnd(refToLight);
116 }
117
118 }
119
120 /*****
121  * Function seven seg display driver
122  * Controls the output to the seven segment buffers
123  * Inputs: first digit, second digit from timing circuit, pushed to display as digits, representing
124  * Outputs: None
125  * Corrupts: W, selection, C ports, B ports
126  *****/
127 void sevenSegDisplayDriver (unsigned firstDigit = 0, unsigned secondDigit = 0)
128 {
129     b_out test = 0b00000000;
130
131     if (firstDigit < 10)
132     {
133         test = segDisp[firstDigit];
134         test.select = 1;
135         output_c(0b00000000);
136         output_b(test);
137     }
138     if (secondDigit < 10)
139     {
140         test= segDisp[secondDigit];
141         test.select = 0;
142         output_b(test);
143         output_c(0b10000000);
144         output_c(0b00000000);
145     }
146 }
147
148 /*****
149  * Function TIMER2 isr
150  * Used a timing circuit, which will branch into other functions.
151  * Inputs: None
152  * Outputs: None
153  * Corrupts: W
154  *****/
155 #INT_TIMER2
156 void TIMER2_isr(void)
157 {
158     countState++;
159     if (countState > 99)
160     {

```

```

161     countState = 0;
162     secondCountOnes++;
163     if (secondCountOnes > 9)
164     {
165         secondCountTens++;
166         secondCountOnes = 0;
167
168         if (secondCountTens > 3)
169         {
170             secondCountTens = 0;
171         }
172     }
173     lightOrderDriver(secondCountOnes, secondCountTens);           // Updates functions every s
174     sevenSegDisplayDriver(secondCountTens, secondCountOnes);
175 }
176 }
177
178
179 /*****
180  * Function main
181  * Initial setup for chip before interrupt takes over.
182  * Inputs: None
183  * Outputs: None
184  * Corrupts: W, selection, C ports, B ports
185  *****/
186 int main (void)
187 {
188     set_tris_b(0b00000000);
189     set_tris_c(0b00000000);
190     output_b(0b00000000);
191     output_c(light1);
192     output_c(light1);
193     output_c(light2);
194     output_c(light2);
195     output_c(light3);
196     output_c(light3);
197     setup_timer_2(T2_DIV_BY_16,195,16);           //627 us overflow,
198     enable_interrupts(INT_TIMER2);
199     enable_interrupts(GLOBAL);
200     while (1){}
201 }

```

B Final code

```
1  #include <16F877A.h>
2  #use delay(clock=20000000)
3
4  #define      REDAMBER      3
5  #define      RED          1
6  #define      AMBER        2
7  #define      GREEN        4
8
9
10 // Integers, which represent counter (for counting x many 10ms intervals), number of 10 seconds passed,
11 unsigned int countState = 0, secondCountTens = 0, secondCountOnes = 0;
12 unsigned segDisp[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x67};
13
14
15 /*****
16  * Struct seven seg output struct
17  * Used to centralise the structure outputted to the displays, uses one of two bits to select which
18  *****/
19 struct seven_seg_output_struct{
20     unsigned int8      digits      :7;
21     unsigned int8      select      :1;
22 };
23
24
25 /*****
26  * Struct traffic lights output struct
27  * Intended order light to control, holds state of each light.
28  *****/
29 typedef struct seven_seg_output_struct b_out;
30 struct traffic_lights_output_struct{
31     unsigned int8      nouse       :1; // MSB, B0
32     unsigned int8      clkSel      :3;
33     unsigned int8      state       :3;
34     unsigned int8      isRed       :1; // LSB, B7
35
36 };
37
38 typedef struct traffic_lights_output_struct light_out;
39
40 light_out light1 = 0b10010010; // LSB = B7, MSB = B0 (isRed, 3xstate, 3xclockSelect,nouse)
41 light_out light2 = 0b10010100; //01001010
42 light_out light3 = 0b10011000; //01001100
43
44
```

```

45  /*****
46      * Function traffic light driver start
47      * Used for switching state of light from green to amber or red to red/amber.
48      * Inputs: Pointer to light_out object, light that will be changed
49      * Outputs: None
50      * Corrupts: W, selection, C ports
51      *****/
52  void trafficLightDriverStart (light_out* selection)
53  {
54      if (selection->isRed==1){
55          selection->state = 3;
56      } else{
57          selection->state = 2;
58      }
59      output_b(0);
60      output_b(*selection);
61  }
62
63
64  /*****
65      * Function traffic light driver end
66      * Used for switching state of light from amber to red and red/amber to green
67      * Inputs: Pointer to light_out object, light that will be changed
68      * Outputs: None
69      * Corrupts: W, selection, C ports
70      *****/
71  void trafficLightDriverEnd (light_out* selection){
72      if (selection->isRed==1){
73          selection->state = 4;
74          selection->isRed = 0;
75      } else{
76          selection->state = 1;
77          selection->isRed=1;
78      }
79      output_b(0);
80      output_b(*selection);
81  }
82
83
84  /*****
85      * Function find light
86      * Used to find light object that needs to be manipulated and returns it
87      * Inputs: Integer channel, represents which light need to be used
88      * Outputs: Pointer to light_out object
89      * Corrupts: W, selection
90      *****/

```

```

91 light_out* findLight (unsigned channel){
92     light_out* refToLight;
93     switch(channel){
94         case 0:
95             refToLight = &light1;
96             break;
97         case 1:
98             refToLight = &light2;
99             break;
100        case 2:
101            refToLight = &light3;
102            break;
103    }
104    return refToLight;
105 }
106
107
108 /*****
109  * Function light order driver
110  * Uses first and second digits of timing loop to determine if lights need to change
111  * Inputs: first digit, second digit from timing circuit, represents which light will be used and wh
112  * Outputs: None
113  * Corrupts: W, selection, C ports
114  *****/
115 void lightOrderDriver (firstDigit, secondDigit){
116     if (firstDigit == 1 || firstDigit == 7){ // If light
117         light_out* refToLight = findLight(secondDigit);
118         trafficLightDriverStart(refToLight);
119     } else if (firstDigit == 3 || firstDigit == 9){ // Second hal
120         light_out* refToLight = findLight(secondDigit);
121         trafficLightDriverEnd(refToLight);
122     }
123 }
124
125
126
127 /*****
128  * Function seven seg display driver
129  * Controls the output to the seven segment buffers
130  * Inputs: first digit, second digit from timing circuit, pushed to display as digits, representing
131  * Outputs: None
132  * Corrupts: W, selection, C ports, B ports
133  *****/
134 void sevenSegDisplayDriver (unsigned firstDigit = 0, unsigned secondDigit = 0)
135 {
136     b_out out = 0b00000000;

```

```

137
138     if (firstDigit < 10)
139     {
140         out = segDisp[firstDigit];
141         out.select = 1;
142         output_c(out);
143     }
144     if (secondDigit < 10)
145     {
146         out = segDisp[secondDigit];
147         out.select = 0;
148         output_c(out);
149     }
150 }
151
152
153 /*****
154  * Function pex_xing button
155  * Function which control routine outside of timing loop, used for pedestrian crossings
156  * Inputs: Current light which may be active / green
157  * Outputs: None
158  * Corrupts: W, selection, C ports, B ports
159  *****/
160 void pedXingButton (unsigned tens)
161 {
162     disable_interrupts(INT_TIMER2); // Turns off timer i
163     light_out* selection = findLight(tens);
164     if(!selection->isRed || selection->state == 2){
165         trafficLightDriverStart(selection); // If light is amber or green
166         delay_ms(1000);
167         trafficLightDriverEnd(selection);
168     }
169     for (int secs = 10; secs > 0; secs--){
170         sevenSegDisplayDriver(secs/10,secs%10); // Cheap way of updating display
171         delay_ms(1000);
172     }
173     enable_interrupts(INT_TIMER2);
174 }
175
176
177 /*****
178  * Function TIMER2 isr
179  * Used a timing circuit, which will branch into other functions.
180  * Inputs: None
181  * Outputs: None
182  * Corrupts: W

```

```

183  *****/
184  #INT_TIMER2
185  void TIMER2_isr(void)
186  {
187      if (input(PIN_B0)) {
188          pedXingButton(secondCountTens);
189          countState = 0;
190          secondCountOnes = 0;
191      }
192      countState++;
193      if (countState > 99) // If one second has passed (100 x 10ms)
194      {
195          countState = 0;
196          secondCountOnes++;
197          if (secondCountOnes > 9)
198          {
199              secondCountTens++;
200              secondCountOnes = 0;
201
202              if (secondCountTens > 2)
203              {
204                  secondCountTens = 0;
205              }
206          }
207          lightOrderDriver(secondCountOnes, secondCountTens); // Updates functions every s
208          sevenSegDisplayDriver(secondCountTens, secondCountOnes);
209      }
210  }
211
212
213  /***/
214      * Function main
215      * Initial setup for chip before interrupt takes over.
216      * Inputs: None
217      * Outputs: None
218      * Corrupts: W, selection, C ports, B ports
219      *****/
220  int main (void)
221  {
222      set_tris_b(0b10000001);
223      set_tris_c(0b00000000);
224      output_c(0b00000000);
225      output_b(0x00);
226      output_b(light1);
227      output_b(light1);
228      output_b(light2);

```

```
229     output_b(light2);
230     output_b(light3);
231     output_b(light3);
232     setup_timer_2(T2_DIV_BY_16,195,16);      //627 us overflow, 10.0 ms interrupt
233     enable_interrupts(INT_TIMER2);
234     enable_interrupts(GLOBAL);
235     while (1){}
236 }
```
