

Designing an Automatic Traffic Light Controller for a T-Junction Road – Electronics 1 – ACDB5002

Electrical & Electronic Engineering Degree Apprenticeship

Joshua H – Student Number: 2420839

Contents

Introduction	3
2. Background Research	3
4. System Design	4
4.1 Block Diagram	4
4.2 Component Selection and Justification.....	4
5. Circuit Design	7
6. System Operation.....	9
6.1 Flowchart	9
6.2 Pseudocode	10
7. Simulation	11
8. Testing.....	15
8.2 Breadboard.....	17
9. Evaluation.....	17
10. Conclusion	18
References	18
12. Appendix	20

Introduction

Traffic light systems are essential components in modern transportation infrastructure. They are designed to control and manage vehicle movement, reduce congestion whilst enhancing road safety. When multiple roads intersect, effective signal control becomes critical to prevent accidents and ensure the smooth movement of traffic, especially where there is an imbalance in the amount of traffic proceeding in a certain direction. This project focuses on the design of a fixed time control traffic light for a T-junction utilising a PIC1 microcontroller. The system is made up of three sets of traffic lights and a set of pedestrian lights. These cycle every ten seconds with a 7-segment display visualising the count, a sounder is also included to sound when the pedestrian lights are green. Outlined below is the research, design and simulation a fixed time control traffic light in Proteus software.

Background Research

Traffic signal control systems are fundamental to the management of traffic on the road, helping to regulate vehicle movement, reduce congestion, and improve road safety. One of the most widely used approaches is fixed-time control, where traffic signals operate according to predetermined timing cycles based on historical traffic data [1].

Fixed-time systems are commonly used due to their simplicity, reliability, and low cost of implementation. They provide consistent and predictable operation, making them suitable for junctions with relatively stable traffic patterns [1]. However, a major limitation of these systems is their inability to respond to real-time traffic conditions. As a result, they can lead to increased delays and inefficient traffic flow when actual traffic demand differs from expected conditions [2], [3].

To improve traffic efficiency, adaptive traffic signal control systems have been developed. These systems use real-time traffic data from sensors such as inductive loops, cameras, or infrared detectors to adjust signal timings dynamically. By responding to current traffic conditions, adaptive systems can reduce congestion and improve overall traffic flow [1], [3]. Despite these advantages, adaptive systems are generally more complex and require additional infrastructure compared to fixed-time systems. This can increase both implementation cost and system maintenance requirements [1].

T-junctions present specific challenges for traffic management due to uneven traffic flow and conflicting vehicle movements. One road carries a higher volume of traffic than the connecting road, which needs managing to ensure efficient movement of traffic. For relatively simple junction layouts, fixed-time control systems can still provide effective operation, particularly where traffic patterns are predictable. Their consistent timing behaviour makes them suitable for applications where simplicity and reliability are important [1].

Safety is also a significant concern at T-junctions, particularly due to vehicle turning movements and pedestrian crossings. Effective signal control must ensure that conflicting traffic streams are separated and that pedestrians are provided with safe crossing opportunities [4]. As well as, this the systems must be designed to fail safely and enter a set state when power is removed [4].

Modern systems may also incorporate traffic detection sensors to monitor vehicle presence and adjust signal timings where required. This allows for improved efficiency compared to purely fixed-time systems [1]. While more advanced systems can improve traffic performance, they also increase system complexity and cost. As a result, fixed-time systems remain widely used in situations where cost-effectiveness, simplicity, and reliability are prioritised [1].

System Design

The task was to design a prototype fixed time control traffic light for a T-junction. The implementation needs to include:

- Three traffic lights
- A set of seven segment displays on each traffic light to count down from 40 to 0
- Every 10 seconds the traffic signals should change
- In the final 10 seconds there should be a cycle for pedestrians

Block Diagram

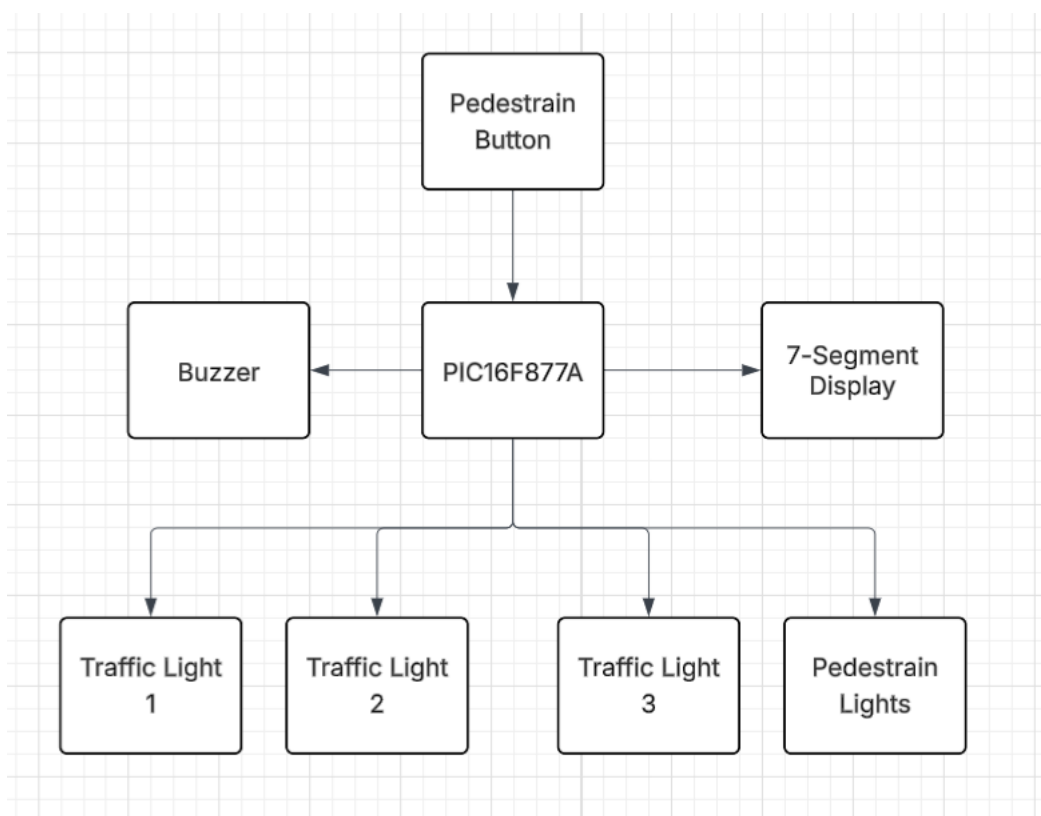


Figure 1: System Diagram

Component Selection and Justification

Components were selected for the traffic light controller based upon engineering considerations including their individual functionality, reliability, cost, scalability as well as their potential implementation in real-world systems.

Microcontroller

A PIC16F877A was selected as the main controller due to its versatility and suitability for embedded control systems [5]. The device offers 33 I/O pins, which is sufficient to manage multiple traffic lights, pedestrians LEDs, seven segment displays and other peripherals without the need for external expanders [6]. This allows the design to stay simple, reducing the overall cost of the system.

An advantage of the PIC16F877A is its internal timers. Timers 0,1, and 2 allow for accurate timer control for the 10 second intervals [5]. Further, the timers can be used as a method of interrupt, allowing for pin polling separate from the main code execution. This ensures that the countdown timer can stay accurate whilst maintaining the required functionality. Whilst alternatives such as an Arduino board may have been used, the PIC was preferred due to its compatibility with CCS C compiler and relatively low cost. As the PIC16F877A's cost is low and the chip itself widely available, it makes a very suitable microcontroller for both prototyping and large-scale deployment. It should be noted that more advanced microcontrollers could offer better processing speed, more efficient power usage, and better features such as Wi-Fi modules, using these would introduce unnecessary complexity for a simple fixed-time system.

Other Components:

An external crystal oscillator was used to provide a stable clock signal for the microcontroller. In a safety critical system such as this accurate timing is a must, and traffic light intervals need to be consistent and predictable.

Common anode seven-segment displays were selected to implement the countdown timer. These displays are cost-effective and interface with the microcontroller well, as the PIC's ports can be individually utilised as one large output. This makes displaying numbers on the displays from the controller simple. Using two displays allows the representation of values 99 to 00. An alternative approach could have been to use a multiplexer to reduce the usage of the I/O pins, this would greatly increase the complexity of the programming as well as introduce possible timing issues. This approach would only be necessary if requirements changed, such as another traffic light being installed.

Transistors have been used to drive the sounders used in the system. They require too much current for the microcontroller to reliably source. Using a transistor with an appropriate HFE and a correctly match current-limiting resistor the PIC will be able to biased the transistor on and off at will allowing it to control the output of the sounder [7].

Resistors are used throughout the circuit primarily for current limiting. Each LED for the traffic lights and each LED segment for the seven-segment displays require a series resistor to prevent excessive current; these values were chosen based on calculations made with Ohm's Law.

The formula used is:

$$R = \frac{V_{ss} - V_f}{I} \quad [8]$$

Where:

- R = resistance (Ω)
- V_{supply} = supply voltage (V_{ss})
- V_f = forward voltage of the LED (V_f)
- I = desired current (A)

Red LED Calculation

- $V_{supply} = 5V$
- $V_f = 2.5V$ [9]
- $R = 220\Omega$ (chosen standard value)

$$I = \frac{5V - 2.5V}{220\Omega} = \frac{2.5}{220} = 0.01136A = 11.36mA$$

Green LED Calculation

- $V_f = 1.7V$ [10]

$$I = \frac{5V - 1.7V}{220\Omega} = \frac{3.3}{220} = 0.015A = 15mA$$

Yellow LED Calculation

- $V_f = 2.2V$ [11]

$$I = \frac{5V - 2.2V}{220\Omega} = \frac{2.8}{220} = 0.0127A = 12.7mA$$

11 mA is within the safe operating range of a typical red LED, ensuring sufficient brightness without exceeding current limits. The green LED draws a slightly higher current (~15 mA), which is still safe and provides good brightness for visibility. The yellow LED operates at approximately 12.7 mA, again within a safe and effective brightness range. Although resistor values could be calculated exactly, 220 Ω is a standard, widely available value that keeps current safely below the typical 20 mA LED limit and simplifies the design by using the same resistor value throughout

A pull-up resistor has been used to ensure stable logic levels for correct function of the pedestrian button. Due to problems with the simulation software this is currently held at 10 Ohms whereas the actual prototype's value sits at 220 Ohms.

Circuit Design

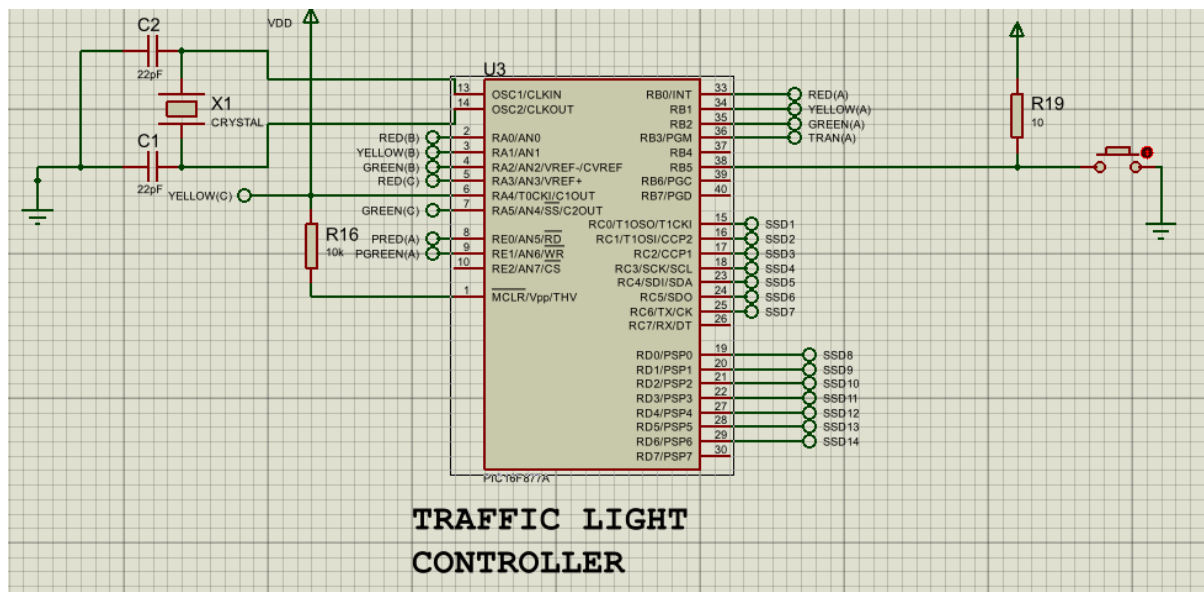


Figure 2: Simulation Diagram

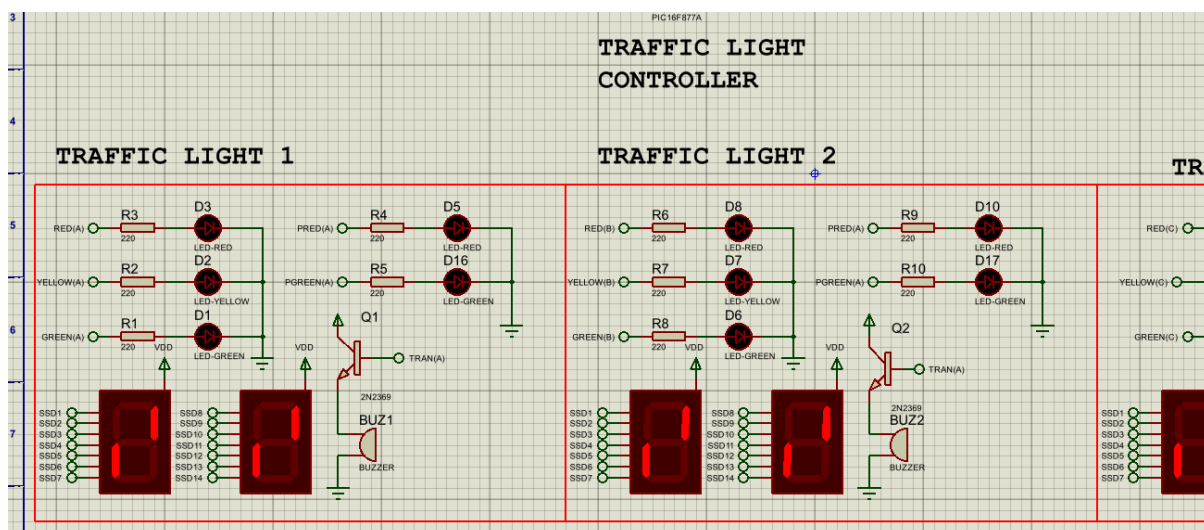


Figure 3: Simulation Diagram

Figure X above shows the microcontroller's pin configuration, Pins B0 to 3 are wired to the first traffic light with port A wired to traffic light 2 and 3 Pins B4 to B7 have been made into interrupt pins, in this configuration the pin will trigger an interrupt when a change in voltage occurs on one of those four pins. Pin B5 is set high through a pull-up resistor, such that when the pedestrian

button is pressed the pin will be brought to ground triggering the interrupt. The same could be achieved using Pin B0 however during the development of the system this could not be achieved reliably as the software was temperamental. As all pedestrian lights will function in the same way at the same time they are all controlled by two pins on port E. Because of the use of ports A and E some functionality of the microcontroller has been lost namely the analogue input pins. This however poses no real problem with the functionality of the system. Similarly to the pedestrian pins pin B3 will trigger all at once, this allows one pin to control all three sounders. As the microcontroller can only source 35mA which is not enough to drive the sounder, instead the pin biases a transistor on and off, which allows current to flow from VDD [6].

To display the count-down timer two seven-segment displays have been used. A seven-segment display is made up of seven LEDs controlled through seven data lines. Each data line is controlled by the microcontroller allowing the programmer to select each segment separately. The segments are generally labelled from A-G.

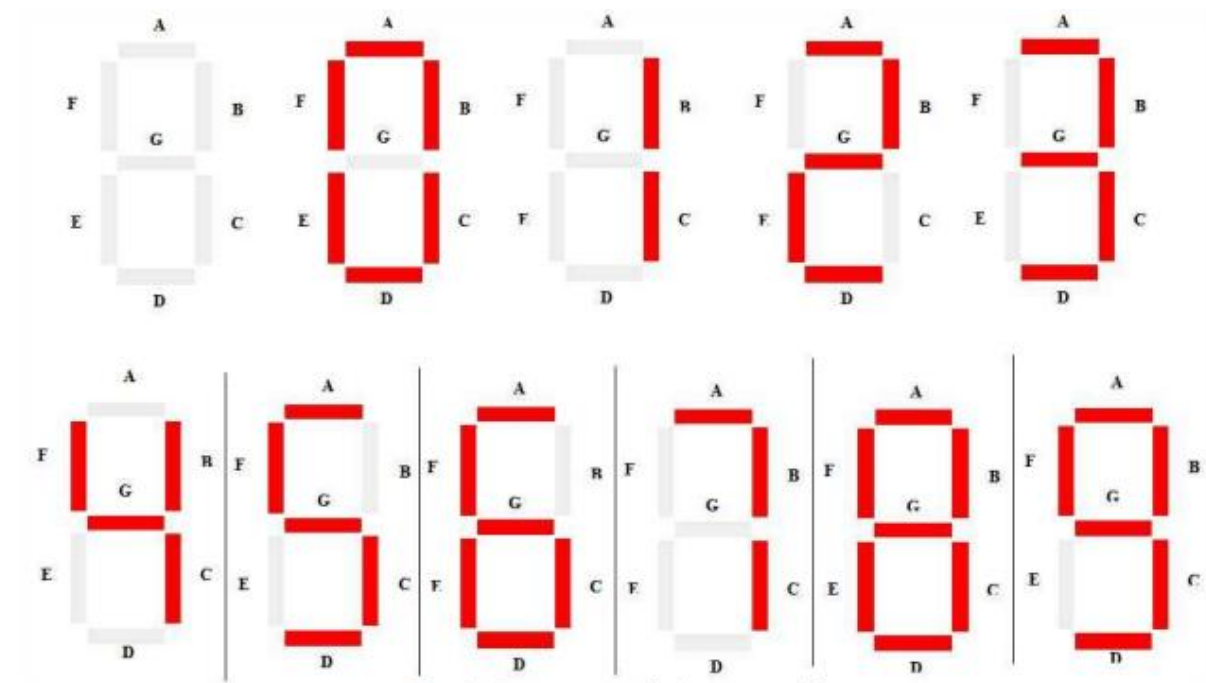


Figure 4: Seven-segment display annotated [12]

As our circuit uses a common anode, if no data pins are high all segments will be lit. By applying a voltage to a specific pin, the voltage across the LED falls enough to bias the LED off. In this circuit all ports C and D are connected to each display allowing us to effectively drive the display through outputting a binary number on the port.

System Operation

Flowchart

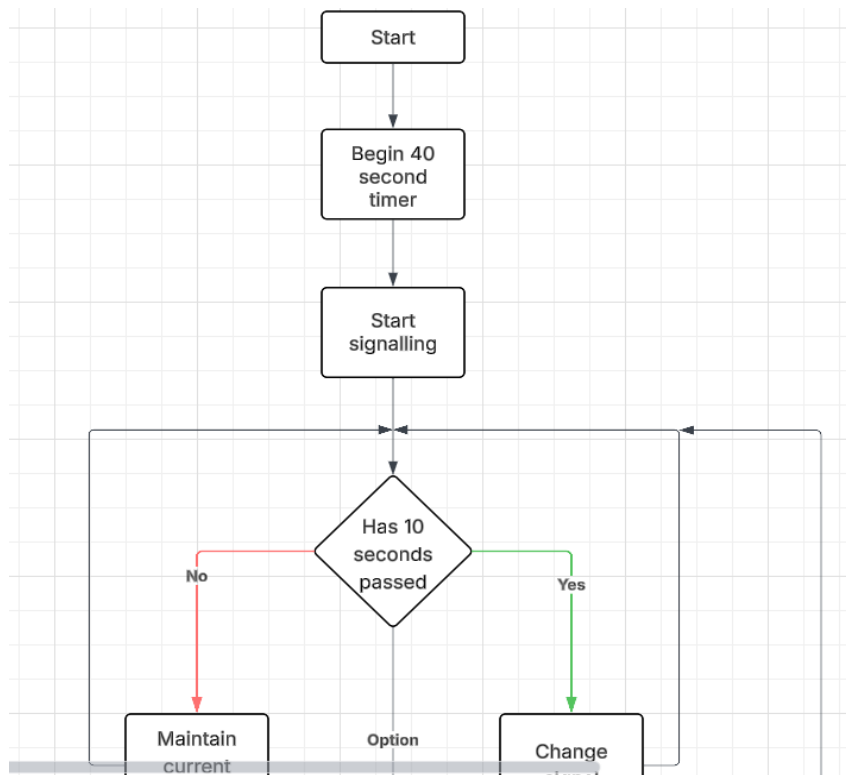


Figure 5

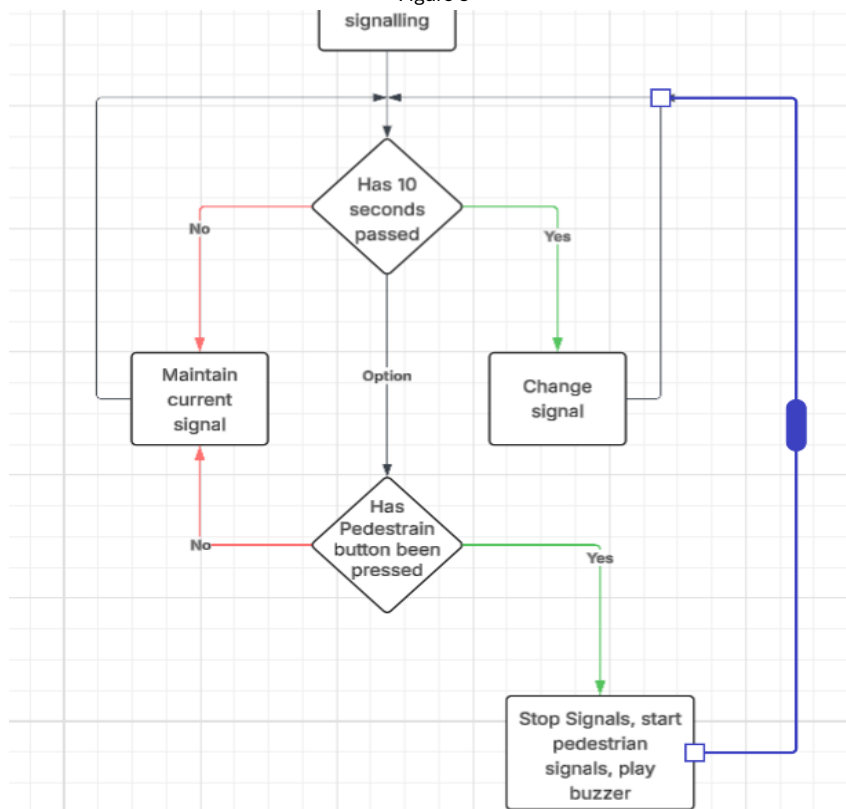


Figure 6

Pseudocode

```
1  instantiate(PORTS);
2  set_low(PORTS);
3
4  traffic_light(){
5
6      set_low(RED_LIGHT);
7      set_high(YELLOW_LIGHT);
8      wait_ms(200);
9      set_high(GREEN_LIGHT);
10
11 }
12
13 pedestrian(){
14
15     set_high(ALL_PED_GREEN_LIGHTS);
16     set_low(ALL_PED_RED_LIGHTS);
17     set_high(ALL_RED_LIGHTS);
18     set_high(BUZZER);
19
20 }
21
22 interrupt(){
23
24     if(BUTTON == PRESSED)
25         pedestrian();
26
27 }
28
29 main(){
30
31
32     for i in range(0,3)
33         first_seven_seg.update();
34
35     for i in range(0,9);
36         second_seven_seg.update();
37
38     switch (stage)
39
40         case 1:
41             traffic_light(RED1,YELLOW1,GREEN1);
42             stage + 1;
43
44         case 2:
45             traffic_light(RED2,YELLOW2,GREEN2);
46             stage + 1;
47
48         case 3:
49             traffic_light(RED3,YELLOW3,GREEN3);
50             stage + 1;
51
52         case 4:
53             pedestrian();
54             stage = 1;
55
56
57
58 }
```

Figure 7

The operation of the system follows a structured and modular approach to control the sequencing, countdown display and pedestrian functionality. The program is designed around a set timing model to ensure consistent and predictable behaviour

Using two for loops the operational cycle is implemented, divided into four discrete 10-second intervals. Each case corresponds to a predefined traffic state in the actual code this is achieved through the use of pins as parameters in functions. The use of cases and a function that calls for parameters appending extra traffic lights is simple.

The countdown mechanism is controlled through direct port manipulation, hexadecimal values are translated to segment control signals allowing for all display signals to be controlled through a look-up table. This is abstracted in the above pseudocode however can be seen in the appendix.

The pedestrian function is incorporated using interrupt driven logic. A change in the designated button pin initiates an interrupt service routine allowing the system to respond independently of the main loop. This ensures pedestrian's requests are reliably and safely processed. All vehicle signals turn red in order to maintain safety.

Simulation

Below are screenshots of the simulation running

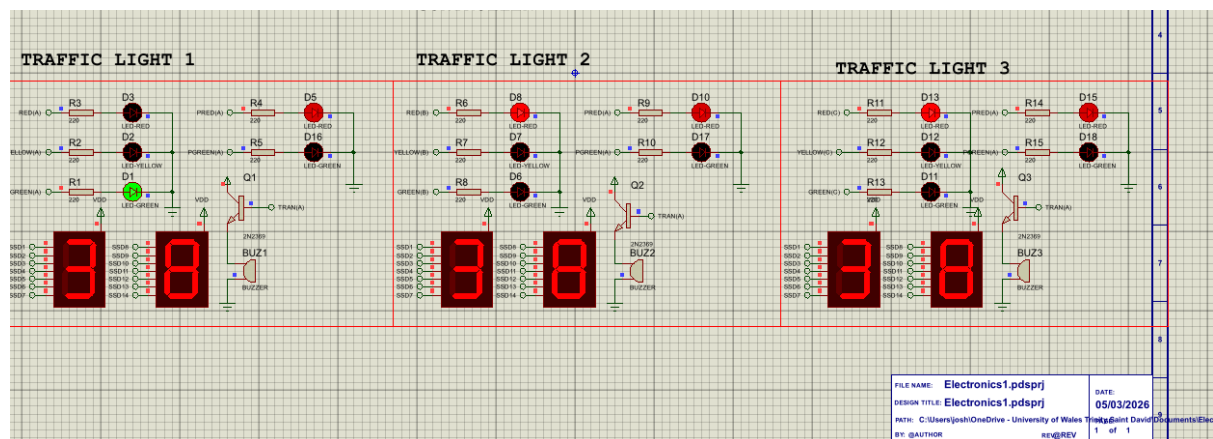


Figure 8: Simulation at Start

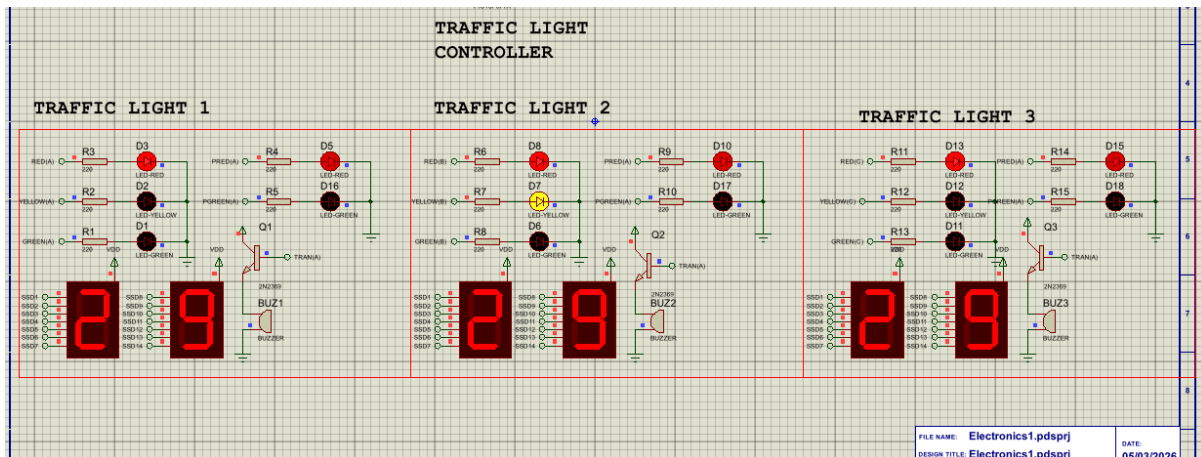


Figure 9: Simulation during a RED/YELLOW cycle

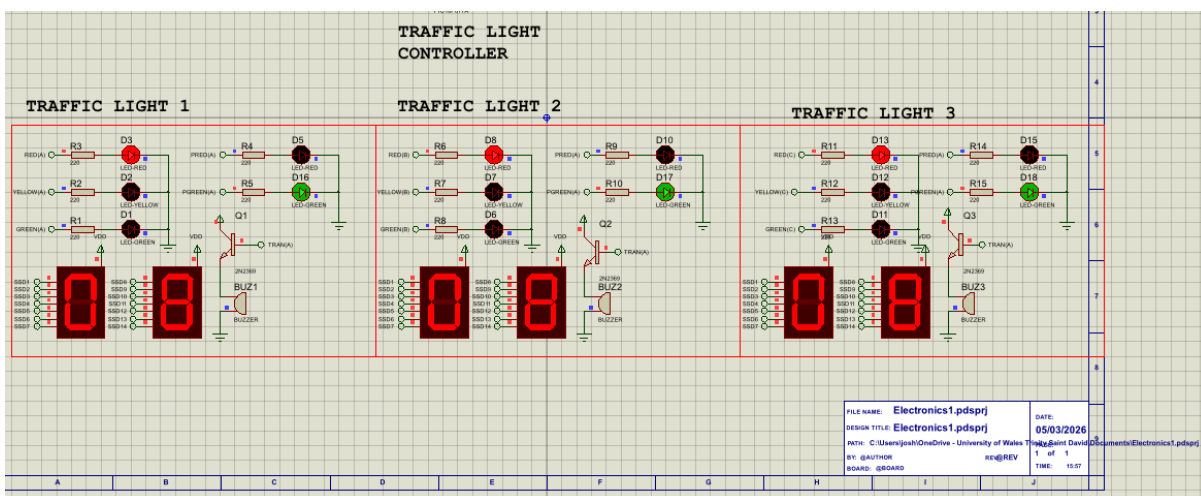


Figure 10: Simulation during a pedestrian crossing

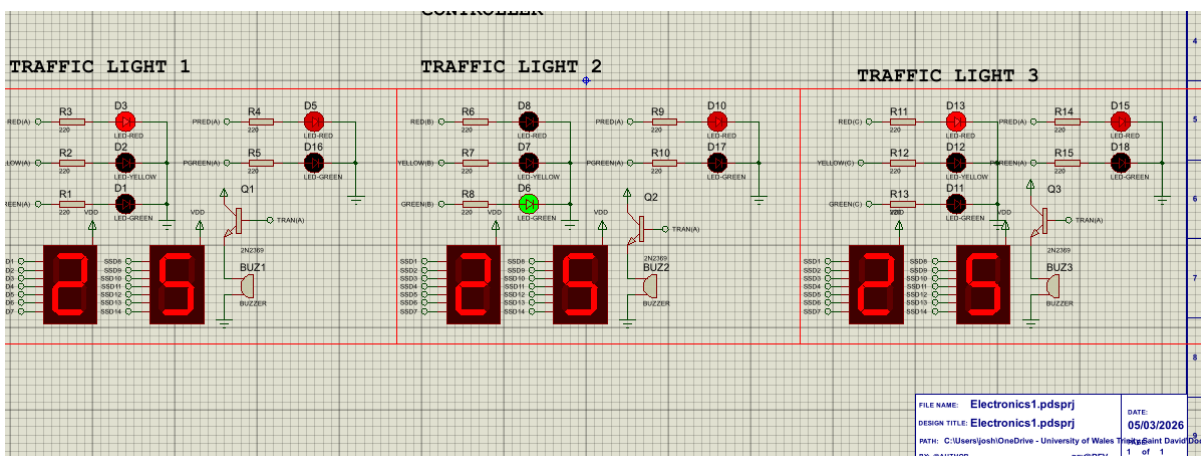


Figure 11: Simulation during a GREEN cycle

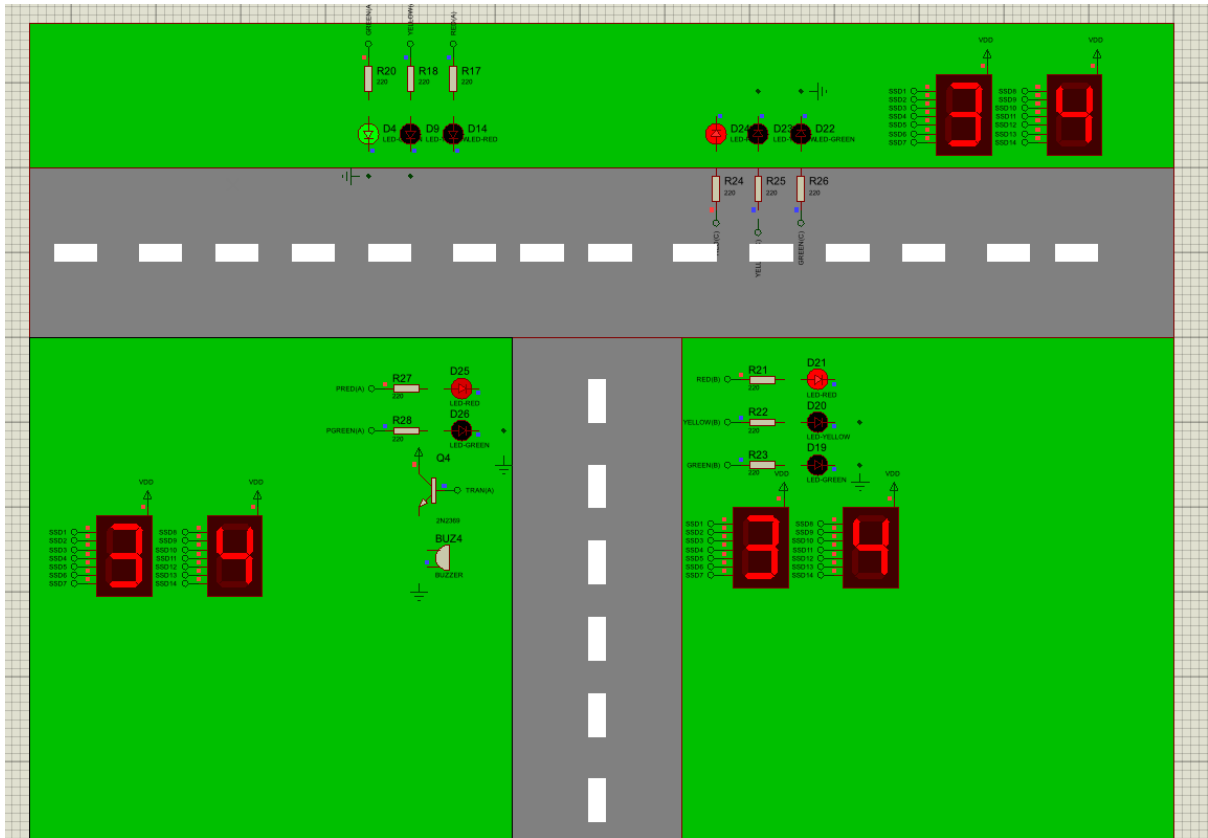


Figure 11: During a GREEN cycle

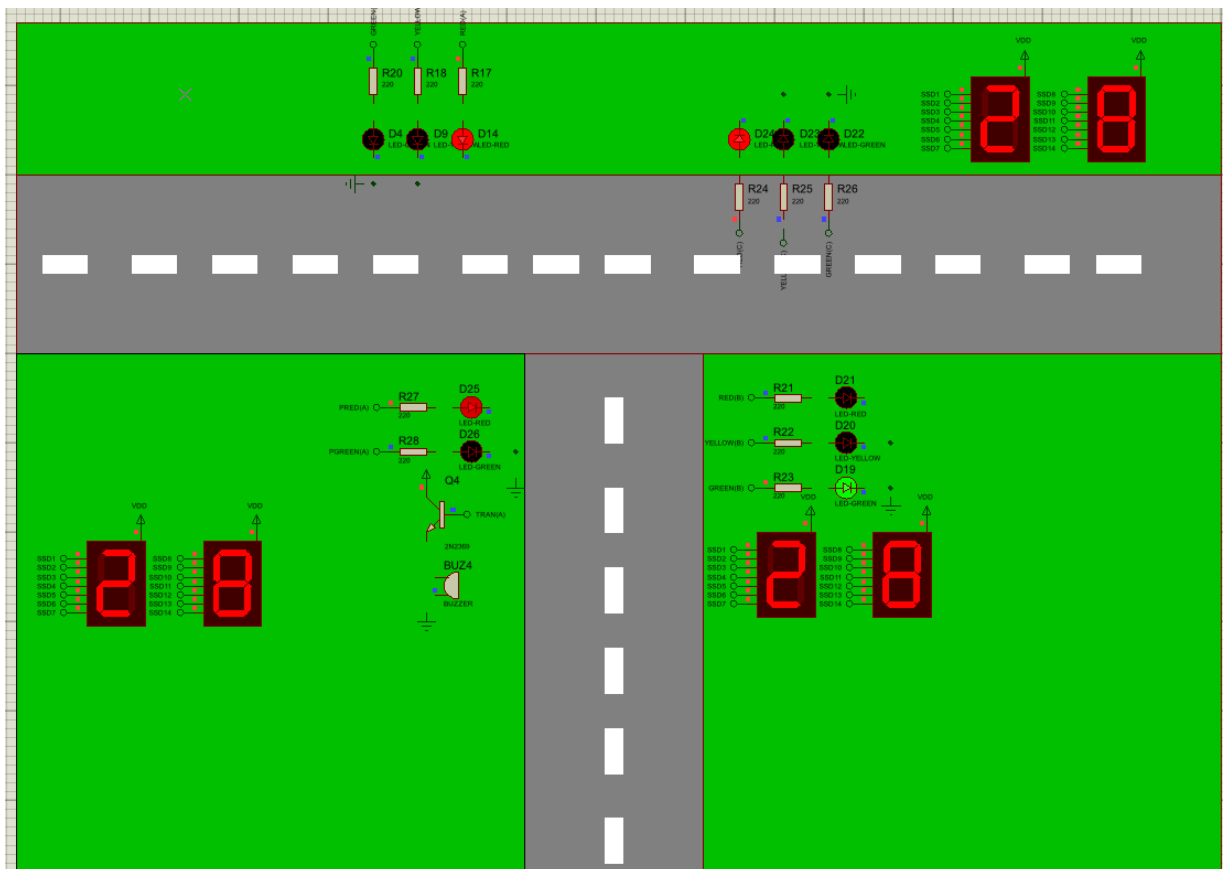


Figure 12: During a GREEN cycle

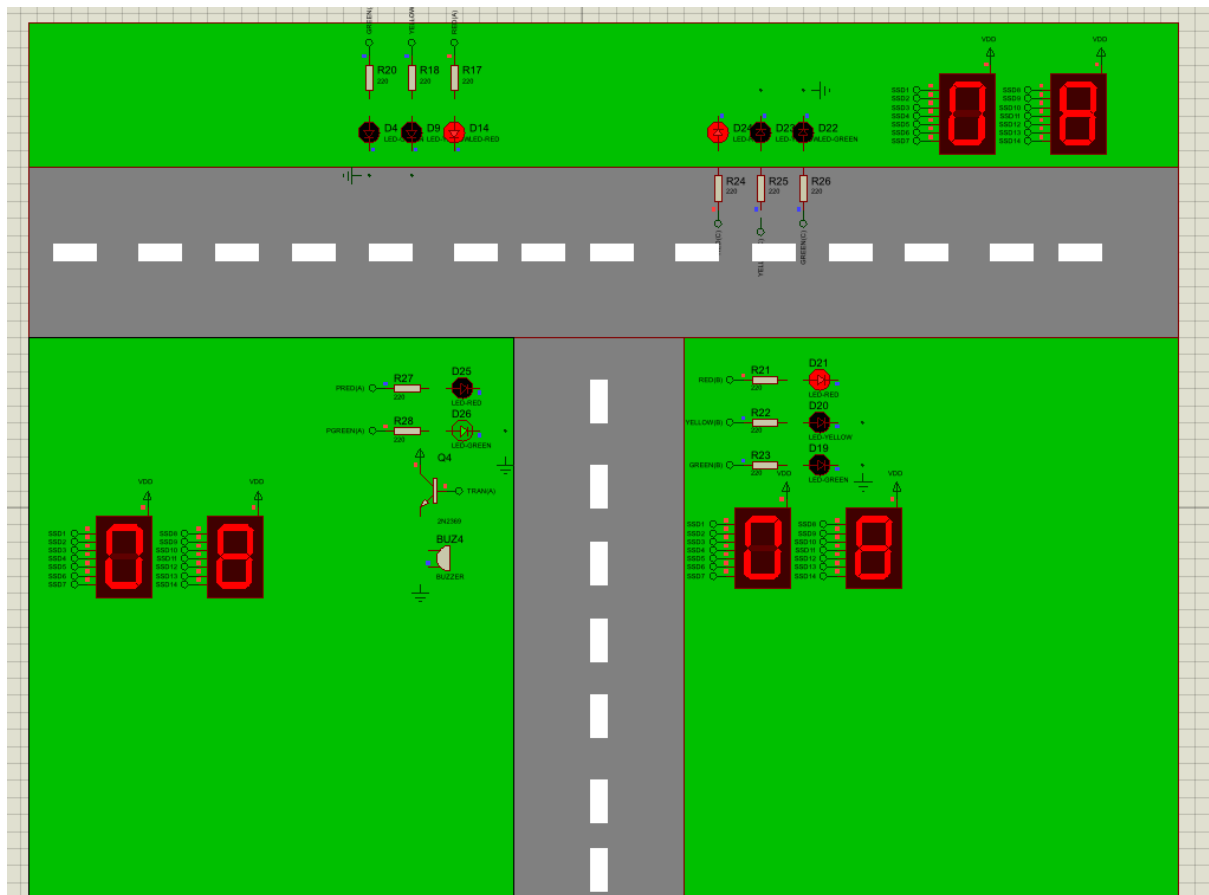


Figure 13: During a pedestrian cycle

Testing

Testing will be made up of 5 parts, to test the functionality of the circuit. Testing outlined below follows the chronological order of the program. Traffic lights will be referred to as TL.

Without 1732 2047

Test Number	Test Spec	Expected	Actual	Adjustments
1.1	On start all should lights should be RED	All light are RED	All lights were red but TL3's yellow LED remained on	Due to proteus issues the pin connected to that light remained high, pin assignment changed
1.2	Yellow and red light for TL1 should be on	Yellow light should switch on a red remain on	Correct lights lit	None required
1.3	Green light for TL1	Red and Yellow should turn off and Green should turn on	Correct lights lit	None required
1.4	Briefly all lights should be red	Green should turn off and all red lights should be on	Correct lights lit	None required
2.1	Yellow and red light for TL2 should be on	Yellow light should switch on a red remain on	Correct lights lit	None required
2.2	Green light for TL2	Red and Yellow should turn off and Green should turn on	Correct lights lit	None required
2.3	Briefly all lights should be red	Green should turn off and all red lights should be on	Correct lights lit	None required
3.1	Yellow and red light for TL3 should be on	Yellow light should switch on a red remain on	Red Light lights but yellow flashes	Error found in the switch statements, the variable 'once' isn't properly reassigned
3.2	Green light for TL3	Red and Yellow should turn off and Green should turn on	Correct lights lit	None required

3.3	Briefly all lights should be red	Green should turn off and all red lights should be on	Correct lights lit	None required
4.1	When pedestrian button is pressed after one cycle all lights should go red and pedestrian lights should go green	All TL lights should go red. And pedestrian lights should turn green	Nothing happens	RB0/Interrupt Pin would not function in proteus, the interrupt type and pin was changed to the RB5 pin
4.2	When pedestrian button is pressed after one cycle all lights should go red and pedestrian lights should go green	All TL lights should go red. And pedestrian lights should turn green	Correct lights lit	None required
4.3	When pedestrian button is pressed the buzzer should sound	The buzzer should sound during the pedestrian cycle	Buzzer sounds	None required
4.4	After TL3 finishes its cycle pedestrian cycle should start in the same way as if a person pressed a button	All TL lights go red, pedestrian lights go green, buzzer sounds	Correct lights lit, buzzer sounds	None required
5.1	Timer counts down correctly	Timer decrements in time with the simulation timer	Timer decreases correctly	None required
5.2	Timer loops back from 00 to 40	Timer loops around correctly	Timer loops around correctly	None required

8.2 Breadboard

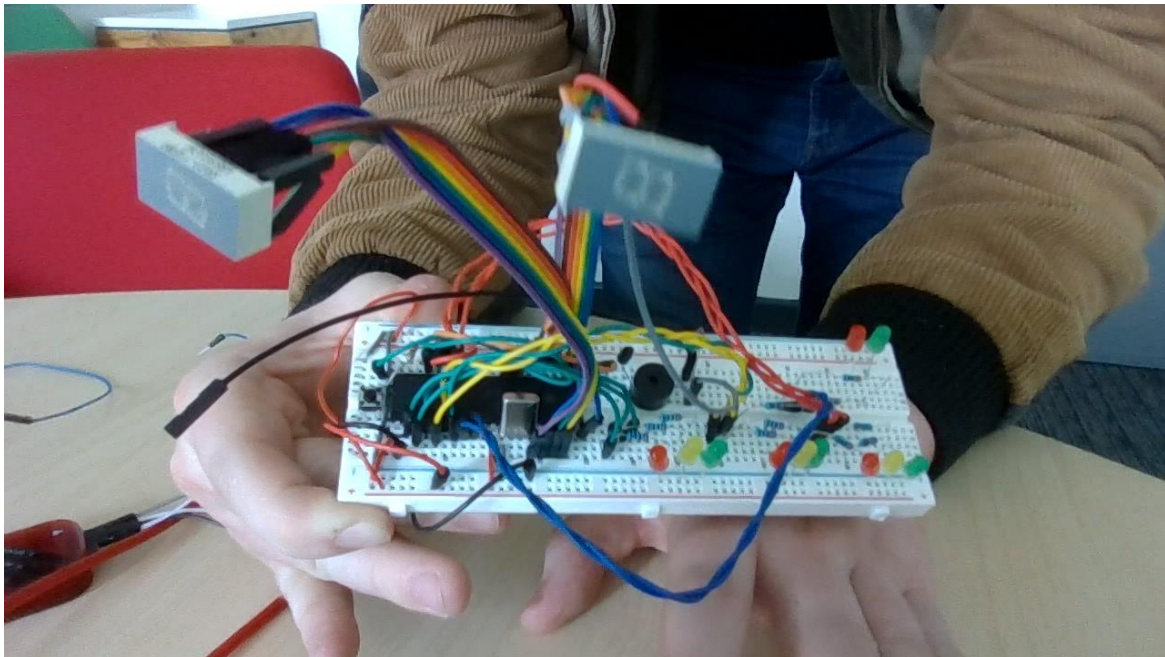


Figure 14: Breadboard Prototype

The breadboard prototype shown in the figure above is a recreation of the system as seen in proteus. Whilst there are less components the system is functionally the same. As proteus is simulation software some components are different on the board differ to the simulated design. Such as the pull-up resistor for the button, in the simulation this is a 10 Ohm resistor however in the breadboard prototype it is a 220 Ohm this is due to limitations on the stock available at the time. Another difference is the lack of resistors into the seven-segment displays, for similar reasons of limitations of stock.

9. Evaluation

The traffic light system was evaluated through a series of tests to assess its functionality, and overall performance. Testing confirmed that the traffic light sequencing operates correctly across all three junction approaches. Each phase was activated in the intended order, with transitions occurring at consistent 10-second intervals. This indicates that the timing logic implemented using the PIC16F877A microcontroller is stable and reliable. The lack of timing drift across multiple cycles supports the logic is correct. The countdown timer was also seen to function accurately, looping back when finished and matching the system state. This demonstrates the code loops effectively and the state-to-state logic is correct. Furthermore, no latency was observed from the port manipulation and look-up table.

The pedestrian button functionality was found to be responsive and reliable. Upon activation the system correctly finished its current state and transitioned to the safe state for pedestrians. The inclusion of a buzzer further enhances safety providing additional indication of the crossing state.

This aspect of the design reflects the real-world considerations and safety requirements for a traffic light system.

During testing, minor issues were identified. Mainly due to simulator issues. For example, whilst in separate files using the dedicated interrupt pin on RB0 the interrupt functioned when the same code and circuit design were placed into the project file the pin stopped functioning. This was resolved by changing the interrupt type to the RB port instead. This tracks changes in the pins B4-B7 which worked for the simulation and breadboard. Further, pin A4 would always remain high, this pin was changed to E2. This however, identified another problem, the LED would flash during the entire cycle. A problem with how the program checks if the traffic sequence has run was identified, and the logic error removed.

Despite the system functioning as intended it is limited by its own design. Fixed time control systems do not and cannot adapt to the varying traffic conditions found in the real-world. This leads to inefficient operation in both heavy and low traffic scenarios, during periods of low traffic, for example, vehicles may have unnecessary delays, whilst during peak traffic congestion may not be managed effectively. This results in congestion and longer wait times. The main limitation of fixed-time control systems.

In addition, the design does not currently incorporate fail-safe mechanisms, such as a default safe state in the event of system failure or power loss. In real-world applications, these features are critical to ensure the safety of road users and pedestrians and compliance with industry standards. With the reliance on simulation, real-world factors such as electrical noise, hardware tolerances, and environmental conditions are not fully represented further limiting the system.

Conclusion

Overall, the system shows a high level of reliability and accuracy within its functioning. The implementation of countdown timers and traffic and pedestrian sequencing is successful, with a further implementation of a pedestrian input and sounder. This is shown both within the simulation and breadboard prototype. However, the system still requires improvement most importantly a fail-safe function and adaptability functions which would be needed for the system to be used in a real-world scenario.

References

- [1] S. S. Qadri et al., "State-of-the-art review of traffic signal control methods," European Transport Research Review, 2020.
- [2] P. Michailidis et al., "Traffic Signal Control: A Review," MDPI, 2025.

- [3] C. Cai and M. Wei, "Adaptive urban traffic signal control," Scientific Reports, Nature, 2024.
- [4] S. Omefe, "Adaptive traffic signal control and pedestrian safety," ResearchGate, 2025.
- [5] Wikipedia Contributors (2026b). *PIC microcontrollers*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/PIC_microcontrollers
- [6] PIC16F87XA Data Sheet 28/40/44-Pin Enhanced Flash Microcontrollers. (n.d.). Available at: <https://ww1.microchip.com/downloads/en/devicedoc/39582b.pdf>.
- [7] Horowitz, P. and Hill, W. (n.d.). *The art of electronics 3rd Edition*. 3rd ed. [online] New York: Cambridge University Press, pp.71–74. Available at: https://kolegite.com/EE_library/books_and_lectures/%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0/_The%20Art%20of%20Electronics%203rd%20ed%20%5B2015%5D.pdf
- [8] Nih.gov. (2024 [online] Available at: <https://pmc.ncbi.nlm.nih.gov/articles/PMC11053092/> [Accessed 6 May 2026].
- [9] Standard LED Red Emitting Colour Chip Material Lens Colour Source Colour Part Number Specification Table Notes. (n.d.). Available at: <https://www.farnell.com/datasheets/1498852.pdf>
- [10] Anon, (n.d.). Available at: <https://www.farnell.com/datasheets/1660999.pdf>
- [11] Anon, (n.d.). Available at: <https://cdn.sparkfun.com/assets/1/3/2/0/4/COM-09592-YSL-R531K3D-D2.pdf>
- [12] Available: <https://datavisualexpert.com/seven-segment-display-diagram>

12. Appendix

```
32 // Function to be called for the pedestrian light
33 void pedestrians(){
34     output_high(RED_A); // Set traffic to RED
35     output_a(0b00001001);
36     output_e(0b010); // Set pedestrain to green
37
38 }
39
40 #INT_RB // ISR called when a change is detected on RB4-7 for the pedestrain button press
41 void portb_isr() {
42     int8 state;
43
44     state = input_b(); // Read PORTB to clear mismatch
45
46     pflag = 1; // set flag
47
48 }
49
50 #INT_TIMER0 // ISR on a timer after a set time this code will run
51 void timer0_isr(){
52     // If the bflag has been raised the sounder will sound
53     if(bflag == 1){
54         output_toggle(BEEP_A);
55     }
56     else{
57         output_low(BEEP_A);
58     }
59 }
60
61
```

```
64
65 // Main traffic light function loops through the state a traffic light is in
66 void traffic_sequence(int RED, int YELLOW, int GREEN, int prev_RED, int prev_GREEN){
67
68     output_e(0b001); // Pedestrian Light on
69
70     output_low(prev_GREEN); // Switch off the previous LEDs
71     output_high(prev_RED);
72     output_high(YELLOW); // Output yellow
73
74     delay_ms(500); // for half a seconds
75
76     //Output Green
77     output_low(RED);
78     output_low(YELLOW);
79     output_high(GREEN);
80 }
81
```

```

82 int main (void)
83 {
84
85     // Pin Config
86     enable_interrupts(INT_RB);
87     enable_interrupts(INT_TIMER0);
88     enable_interrupts(GLOBAL);
89     setup_timer_0(RTCC_INTERNAL | RTCC_DIV_64);
90     set_timer0(156);
91     port_b_pullups(TRUE);
92     setup_adc_ports(NO_ANALOGS);
93     setup_adc(ADC_OFF);
94     setup_comparator(NC_NC_NC_NC);
95
96     // Setting all pins to a known state
97     set_tris_a(0x00);
98     set_tris_b(0xF0);
99     set_tris_c(0x00);
100    set_tris_d(0x00);
101    set_tris_e(0x00);
102
103    // Counter variables will be iterated through
104    int stage = 0;
105    int once = 0;
106
107    // Setting the initail state
108    output_high(RED_A);
109    output_a(0b00001001);
110    output_e(0b001);
111    output_low(BEEP_A);
112
113    while (1){
114
115        for (int i = 6; i < 10; i++){ // every ten seconds
116
117            stage += 1; // Iterate through the stage
118            once = 0; // Set single run intger
119            output_c(display[i]); // display num
120            bflag = 0;
121            for (int y = 0; y < 10; y++){ // Every second
122
123                output_d(display[y]); // Display num
124
125                delay_ms(1000); // Wait one second
126
127                if(pflag){ // If an RB pin has been pressed
128                    bflag = 1;
129                    pedestrians();// Run pedestrain func
130                    pflag = 0;
131
132                }
133
134                switch (stage){
135
136                    case 1: // First traffic Light
137                        if(once == 0){
138                            traffic_sequence(RED_A, YELLOW_A, GREEN_A, RED_C, GREEN_C);
139                            once = 1;

```

```

137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165

```

```

switch (stage){

    case 1: // First traffic Light
        if(once == 0){
            traffic_sequence(RED_A, YELLOW_A, GREEN_A, RED_C, GREEN_C, stage);
            once = 1;
        }

        break;

    case 2: // Second Traffic Light
        if(once == 0){
            traffic_sequence(RED_B, YELLOW_B, GREEN_B, RED_A, GREEN_A, stage);
            once = 1;
        }

        break;

    case 3: // Third Traffic Light
        if(once == 0){
            traffic_sequence(RED_C, YELLOW_C, GREEN_C, RED_B, GREEN_B, stage);
            once = 1;
        }

        break;

    case 4: // Pedestrian Step
        bflag = 1;
        pedestrians(); // Run pedestrian func

```