

Designing an Automatic Traffic Light
Controller for a T- Junction Road
Electronics I
ACDA5002
BEng Embedded Electronic Systems
Zakir Basit

Table of Contents

Introduction	3
Background Research	3
T-Junctions.....	3
Traffic Signals	4
Pedestrian Safety.....	4
System Design	5
Implementation	7
Testing and Results	10
Test Plan	10
Testing	10
Debugging.....	10
Simulation Results	11
Future Work.....	14
Traffic Light Sensors.....	14
Pedestrian Crossing Button	15
Conclusion.....	15
References	16
Appendix.....	17

Introduction

This report will detail the design process and provide simulation evidence for a T-junction traffic light control system. It aims to reflect the application of microcontroller programming and circuit simulation skills to solve a practical traffic management problem. The report includes background research informing the design choices, the hardware and software design methodology, the implementation steps within the Proteus simulation environment, and the results obtained from functional testing. The goal was to develop a reliable and efficient controller using a PIC16 microcontroller, featuring timed signal changes, a countdown display, and a dedicated pedestrian safety phase for a T-junction scenario.

Background Research

Prior to commencing the design phase of the T-junction traffic light controller, establishing a solid foundational understanding of pertinent principles of traffic management and relevant signalling standards. Research was therefore conducted on the inherent nature of traffic flow, particularly at T-junctions, the regulatory standards that govern traffic signal operation and the overarching consideration of pedestrian safety precautions.

T-Junctions

T-junctions, intersections where a side road terminates perpendicularly onto a continuing main road, present traffic control problems distinct from those of conventional four-way intersections. The primary problem is how to balance the safe and efficient merging of traffic turning from the side road (named road C in this system design) onto the main road with traffic turning off the main road onto the side road while minimising the disruption to the continuous flow along the main road (represented by opposing flows A and B) [1]. Simple, non-adaptive timer systems are often inadequate in such cases as they cannot respond to the potentially asymmetrical and changing traffic volumes characteristic of T-junctions, with the potential consequence of undue queuing on one approach when the other is under light traffic. The employment of a microcontroller as the main control element offers an excellent advantage by enabling precise, versatile, and programmable timing cycles. This programmability allows the signal phasing and durations to be set carefully to suit the specific geometric and flow conditions of the junction. Microcontrollers also allow for much easier incorporation of essential safety features, such as the 'All-Red' period, where signals on all vehicular approaches display red simultaneously briefly between potentially conflicting green phases. This clearance time is a vital safety element, particularly critical at T-junctions where merging and turning manoeuvres are inherent conflict points, clearing the junction before the next traffic stream is permitted to proceed [1].

Traffic Signals

Drivers must receive clear instructions from traffic lights. Effective traffic management relies heavily on providing clear and unambiguous directions to drivers in the form of traffic light indications. The sequence used in the United Kingdom mandated by The Highway Code is Red, Red and Amber together, Green, Amber, and Red again [2]. The initial red indication gives a warning for an obligatory halt. The next Red/Amber is a warning signal, advising drivers that the Green is imminent and they need to prepare to move forward. The Green light grants permission to proceed if the way is clear. The Amber warning, which occurs after Green, is a warning that the signal is changing to Red and motorists must pull up safely behind the stop line except if they have already crossed it or are too close to pull safely. The subsequent red warning finishes the sequence for that access. Central to this cycle, especially between phases giving way to opposing traffic flows, is the aforementioned 'All-Red' interval. This time, for safety purposes, gives vehicles which may have been entering the intersection at the end of the green or amber phase sufficient time to clear the intersection completely before opposing traffic from a different direction is released [1].

Pedestrian Safety

Pedestrian safety at signalised intersections is a high priority in traffic system design, especially during periods of heavy vehicular and pedestrian traffic. Conflicts between vehicles and pedestrians when pedestrians are attempting to cross the road are a frequent cause of accidents with often severe injury. A standard method for enhancing pedestrian safety is to provide a separate crossing phase in the traffic light cycle [3]. During this phase, all vehicle traffic lights are programmed to display Red, halting traffic on all approaches and creating a safe gap for pedestrians to cross the road free from any conflicting vehicle movements. Modern UK standards for pedestrian crossings also put a high emphasis on accessibility, with features beyond the normal visual signals often being mandated. These can include tactile pavement surfaces to guide visually impaired pedestrians to the crossing point, audible announcements (beeps or voice messages) that repeat during the pedestrian green phase, and turning cones located under the push-button unit that turn during the crossing phase to provide a tactile announcement for individuals with both visual and hearing disabilities [3].

System Design

The hardware for the system consists of the setup around the PIC16F877A microcontroller, which provides the smarts for the system.

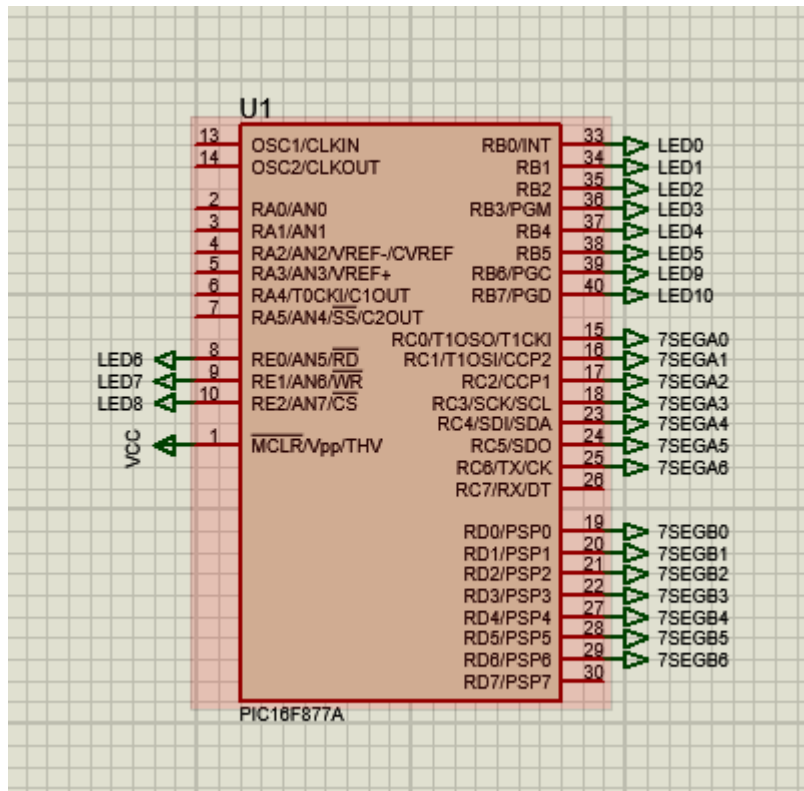


Figure 1: The PIC16F877A used for the project.

Peripheral components are connected to its I/O pins to provide the necessary signalling and display function. The key output devices include the traffic light indicators, where LEDs are used in simulation. Nine LEDs are resolute for traffic control of vehicles: Red, Amber, and Green LED modules are resolute for Road A (one lane on the main road), Road B (the other direction on the main road), and Road C (terminating side road). A second set of LEDs, one Red and one Green, are used to mimic the pedestrian signals, indicating whether it is safe for pedestrians to cross. To protect these LEDs from high current and prevent both the LEDs and the output ports of the microcontroller from being damaged, a series current-limiting resistor is utilised for each LED [4] (in this case, the simulator does this without needing to add the resistors in the schematic). The resistance value, usually selected as 220Ω for standard LEDs operated by a 5V supply, is calculated from the LED's forward voltage and the desired operating current, based on Ohm's law [5].

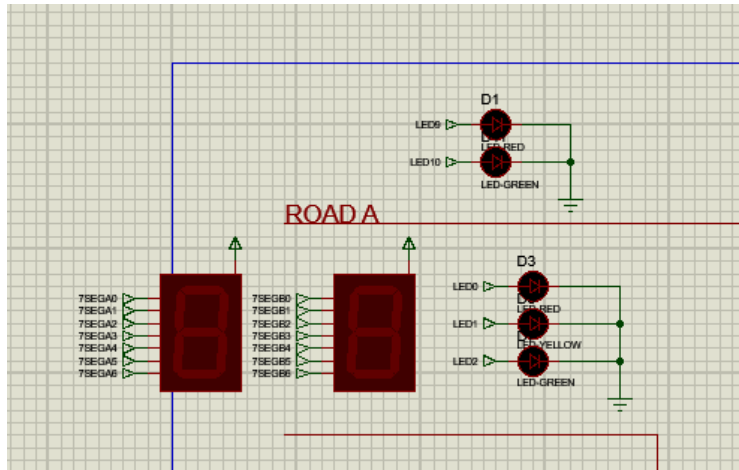


Figure 2: PORTB pins RB0, RB1, and RB2 are utilised for Red, Amber, and Green LEDs of Road A, respectively.

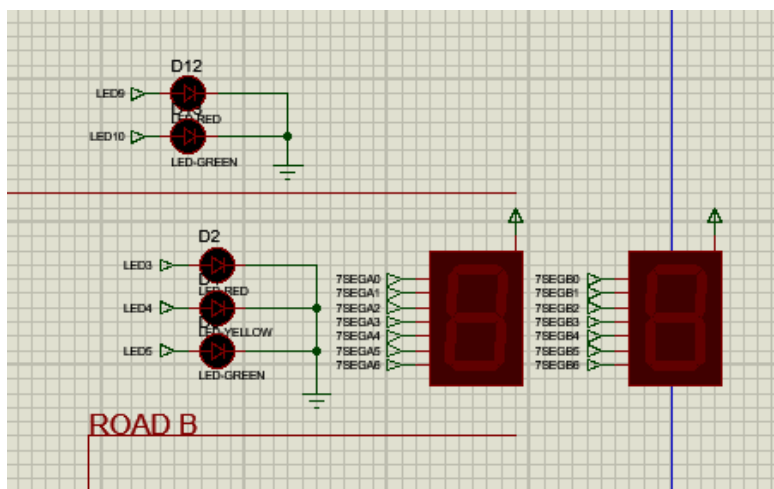


Figure 3: PORTB pins RB3, RB4, and RB5 of the same port control Red, Amber, and Green LEDs of Road B.

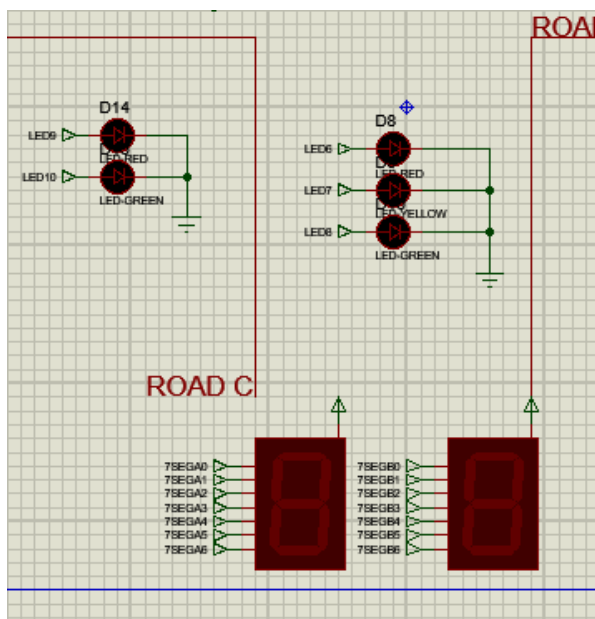


Figure 4: Road C (Red, Amber, Green) signals are assigned to pins RE0, RE1, and RE2 of PORTE, utilising another available port.

The Red and Green pedestrian are sourced from the remaining available pins on PORTB, i.e., RB6 and RB7. PORTC outputs (RC0-RC7) feed the tens digit display segments and PORTD outputs (RD0-RD7) feed the ones digit display segments. This organised distribution ensures that all the required signals are managed using the I/O facilities of the PIC16F877A provided.

Implementation

The core of the software is an infinite while(TRUE) loop, causing the traffic light system to operate continuously once initialised.

```

1  // --- Main Loop ---
2  while (1) {
3
4      // 1. Display the current count
5      display_digits(counter);
6
7      // 2. Determine and set light states based on the counter
8      if (counter >= 30) {
9          // --- Pedestrian Phase (Seconds 30-39) ---
10         // All vehicle traffic lights RED, Pedestrian GREEN
11         all_traffic_red_ped_green();
12     } else {
13         // --- Vehicle Phase (Seconds 0-29) ---
14         // Pedestrian RED is ON, Pedestrian GREEN is OFF for this entire phase
15         output_high(LED_RED_P);
16         output_low(LED_GREEN_P);
17
18         // Now, set only the vehicle traffic light states (A, B, C)
19         if (counter >= 27) { // State 9: All Traffic Red (Transition before Ped) (27-29)
20             output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A);
21             output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B);
22             output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C);
23         } else if (counter >= 25) { // State 7: A/B Red, C Amber (25-26)
24             output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A); // A Red
25             output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B); // B Red
26             output_low(LED_RED_C); output_high(LED_YELLOW_C); output_low(LED_GREEN_C); // C Amber
27         } else if (counter >= 16) { // State 6: A/B Red, C Green (16-24)
28             output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A); // A Red
29             output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B); // B Red
30             output_low(LED_RED_C); output_low(LED_YELLOW_C); output_high(LED_GREEN_C); // C Green
31         } else if (counter == 15) { // State 5: A/B Red, C Red & Amber (15)
32             output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A); // A Red
33             output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B); // B Red
34             output_high(LED_RED_C); output_high(LED_YELLOW_C); output_low(LED_GREEN_C); // C Red+Amber
35         } else if (counter >= 13) { // State 4: All Traffic Red (Transition) (13-14)
36             output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A);
37             output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B);
38             output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C);
39         } else if (counter >= 11) { // State 3: A/B Amber, C Red (11-12)
40             output_low(LED_RED_A); output_high(LED_YELLOW_A); output_low(LED_GREEN_A); // A Amber
41             output_low(LED_RED_B); output_high(LED_YELLOW_B); output_low(LED_GREEN_B); // B Amber
42             output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C); // C Red
43         } else if (counter >= 2) { // State 2: A/B Green, C Red (2-10)
44             output_low(LED_RED_A); output_low(LED_YELLOW_A); output_high(LED_GREEN_A); // A Green
45             output_low(LED_RED_B); output_low(LED_YELLOW_B); output_high(LED_GREEN_B); // B Green
46             output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C); // C Red
47         } else if (counter == 1) { // State 1: A/B Red & Amber, C Red (1)
48             output_high(LED_RED_A); output_high(LED_YELLOW_A); output_low(LED_GREEN_A); // A Red+Amber
49             output_high(LED_RED_B); output_high(LED_YELLOW_B); output_low(LED_GREEN_B); // B Red+Amber
50             output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C); // C Red
51         } else { // State 0: All Traffic Red (Transition) (0)
52             output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A);
53             output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B);
54             output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C);
55         }
56     }
57     // End of Vehicle Phase logic
58
59     // 3. Wait for 1 second
60     delay_ms(1000);
61
62     // 4. Increment counter and reset if it reaches 40
63     counter++;
64     if (counter >= 40) {
65         counter = 0; // Reset the counter for the next cycle
66     }
67 } // End while

```

Figure 5: Main loop of the program

Inside the loop, the code does a collection of operations on each run. It updates the 7-segment displays with the number for the current value of the main counter variable, counting down seconds already passed in the 40-second cycle. This invokes a special procedure, `display_digits()`, that calculates the ones and tens digits from the counter value and investigates a preloaded array named 'display' to obtain the corresponding segment patterns. This array holds the very byte patterns (e.g., 0xC0 for '0', 0xF9 for '1',

as based on the specific display arrangement used) required to illuminate the appropriate segments on the common cathode displays via PORTC and PORTD.

Following the display update, the code goes to the main state logic block. There, a nested set of if and else if statements verify the present value of the counter. Based on the value of the counter in specified ranges, the code determines the appropriate state for all traffic and pedestrian lights. For instance, if the counter value is in between 30 to 39 (inclusive), then the system switches to the pedestrian phase, calling an assistant function `all_traffic_red_ped_green()` that sets all car signals (A, B, C) Red and the pedestrian signal Green.

```
void all_traffic_red_ped_green() {  
    output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A);  
    output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B);  
    output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C);  
    output_low(LED_RED_P); output_high(LED_GREEN_P); // Ped Green ON, Ped Red OFF
```

Figure 6: Helper function to set all the traffic lights to red and pedestrian to green

If the counter is below 30, the system is in vehicle mode; the pedestrian indication is Red, and further if/else if conditions propel the vehicle light state from increasingly narrower ranges of counters to the different phases of the vehicle cycle (e.g., Main Road Green, Main Road Amber, All Red phase transition, Side Road Green, etc.). The `output_high()` and `output_low()` functions are called often here to turn on the appropriate LEDs or turn them off by setting the corresponding microcontroller pins high (5V) or low (0V).

```

// --- Determine and Set Light States based on counter ---
IF counter >= 30 THEN
    // Pedestrian Phase Logic
    CALL all_traffic_red_ped_green() // Sets A=R, B=R, C=R, Ped=G
ELSE
    // Vehicle Phase Logic
    SET Pedestrian_Light = RED // Pedestrian Red always ON during vehicle phases

    // Determine specific vehicle light states
    IF counter >= 27 THEN // All Red (before Ped)
        SET Lights_A = RED, Lights_B = RED, Lights_C = RED
    ELSE IF counter >= 25 THEN // C Amber
        SET Lights_A = RED, Lights_B = RED, Lights_C = AMBER
    ELSE IF counter >= 16 THEN // C Green
        SET Lights_A = RED, Lights_B = RED, Lights_C = GREEN
    ELSE IF counter == 15 THEN // C Red+Amber
        SET Lights_A = RED, Lights_B = RED, Lights_C = RED_AMBER
    ELSE IF counter >= 13 THEN // All Red (Transition)
        SET Lights_A = RED, Lights_B = RED, Lights_C = RED
    ELSE IF counter >= 11 THEN // A/B Amber
        SET Lights_A = AMBER, Lights_B = AMBER, Lights_C = RED
    ELSE IF counter >= 2 THEN // A/B Green
        SET Lights_A = GREEN, Lights_B = GREEN, Lights_C = RED
    ELSE IF counter == 1 THEN // A/B Red+Amber
        SET Lights_A = RED_AMBER, Lights_B = RED_AMBER, Lights_C = RED
    ELSE // counter == 0 // All Red (Start/Transition)
        SET Lights_A = RED, Lights_B = RED, Lights_C = RED
    END IF // End of vehicle state determination
END IF // End of Pedestrian/Vehicle phase check

```

Figure 7: Pseudocode entailing the main control loop logic for the traffic light controller

After setting up the light states, the program uses a time delay using the `delay_ms(1000)` call. This pauses for 1000 milliseconds, or one second. This delay really sets the system timer resolution; the counter is updated, and thus the state potentially changes, every second. Finally, at the end of each loop iteration, the counter variable is updated. There is a check for whether the counter has hit 40. In case it has, it is a sign of the completion of the entire 40-second sequence, and the counter is reset to 0 so that the entire process may proceed uninterrupted.

```

// --- Wait for 1 second ---
DELAY 1000ms

// --- Update Counter for next cycle ---
INCREMENT counter

// --- Check for cycle completion and Reset ---
IF counter >= 40 THEN
    SET counter = 0 // Reset counter to start new cycle
END IF

// End of Main Control Loop
END WHILE

```

Figure 8: Pseudocode illustrating the counter progression and reset

Testing and Results

Test Plan

A test plan was developed to extensively evaluate the implemented traffic light controller in the Proteus simulation software. It verifies against all the functional requirements, i.e., correct sequencing of signals, correct phase timing, correct operation of the countdown display, and correct functioning of the pedestrian safety phase and the transition intervals. This involved executing the simulation and compare observed behaviour with pre-defined expected behaviour for certain situations.

Testing

The first test was for the overall signal sequence for the main road (A and B). The simulation was initiated, and what was expected was the default UK sequence: Red, Red-Amber, Green, Amber, and then Red, accurately phased between A and B. The simulation started with the sequence cycling through properly from Red, Red-Amber, and Green. But the next Amber phase for A and B roads was missed unexpectedly, and they went straight from Green to All-Red. This was the first bug that was found.

A second test verified the duration of the dedicated pedestrian phase. The simulation was allowed to run until the counter reached 30. The expected outcome was that the pedestrian green signal would be on for the full 10 seconds, which are counter values 30 through 39, with all the car lights set to Red. The actual result in early testing was that the pedestrian phase terminated prematurely after 9 seconds (at counter 38), with the system resetting the counter to 0 when it should have transitioned to state 39. This revealed a timing or logic error associated with the pedestrian phase boundary condition.

A third test examined the 7-segment display's operation through one complete cycle. The expected behaviour under normal conditions would be a defined counting from 0 up to 39 in one-per-second increments. An initial finding, as discovered by initial testing, was that the counting being performed well for most of the cycle would stop short, instead of displaying '39' and counter-counting back, instead resetting prematurely from '38', as if in the malfunction encountered with the second test.

Debugging

For the skipped Amber phase along the main road, the 'if/else if' software code governing states after the main green was investigated. It was found that the condition test for the Amber phase (expected around counter 11-12) was logically overridden by the subsequent 'All-Red' condition and therefore skipped. The conditional statements were re-arranged and boundaries re-set (else if (counter >= 11 && counter <= 12)) to enable the Amber state to execute as expected prior to the 'All-Red' state. The main loop

counter logic was verified to correct the premature end of the pedestrian phase and consequent premature reset of the counter. The error was discovered in the final check: if (counter >= 39) was inadvertently used rather than if (counter >= 40) as the reset condition. Modifying this condition to if (counter >= 40) { counter = 0; } permitted the counter to reach 39, providing the pedestrian phase its full 10-second duration and the display time to show '39' before resetting.

Simulation Results

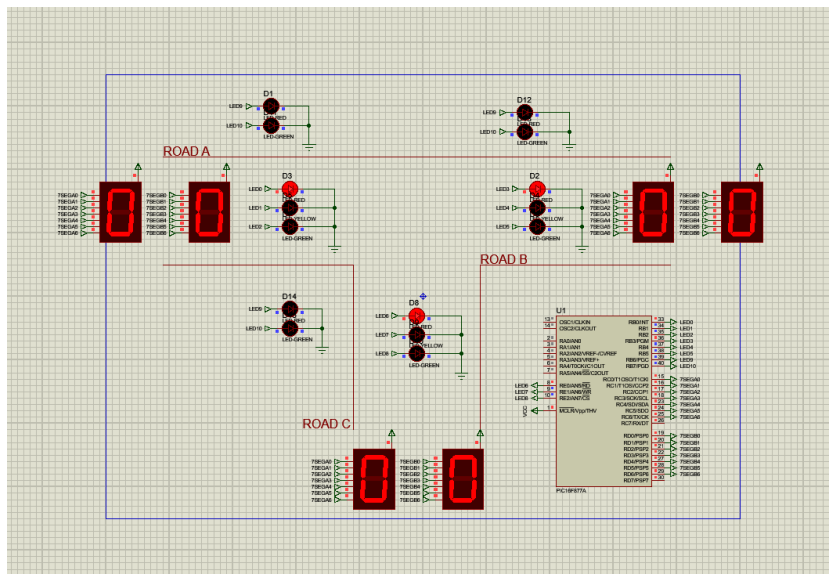


Figure 9: Initial all-red state

Roads A, B and C display red signals.

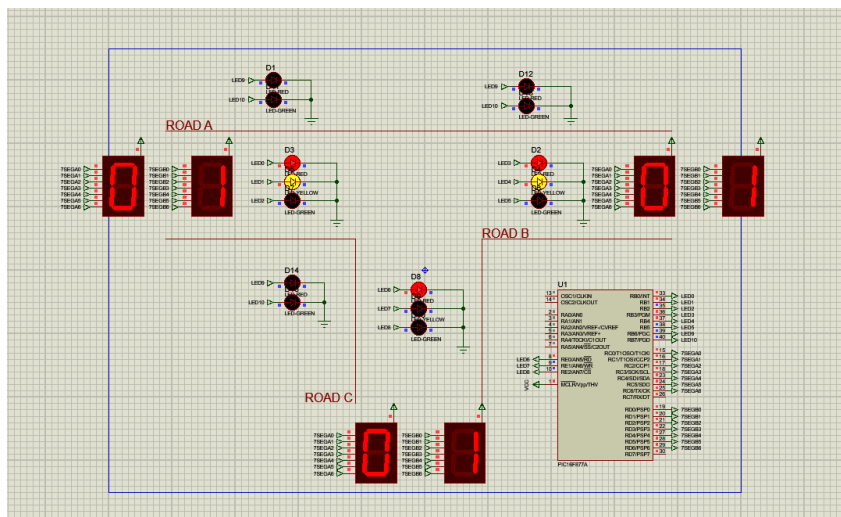


Figure 10: Main Road prepares for green state

Roads A and B display red and amber signals simultaneously whilst road C remains red.

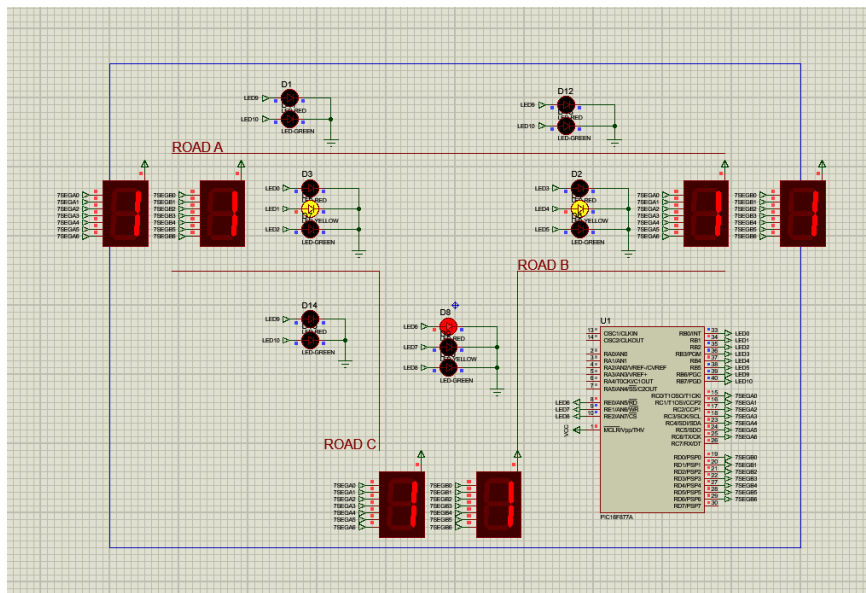


Figure 11: Main Road amber state

Roads A and B display amber signals.

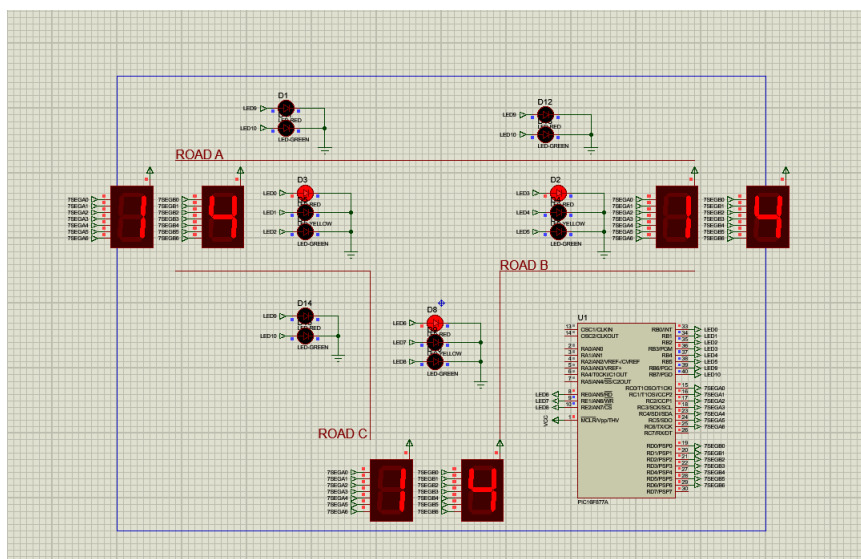


Figure 12: All-red transition state

All roads display red signal for a brief period.

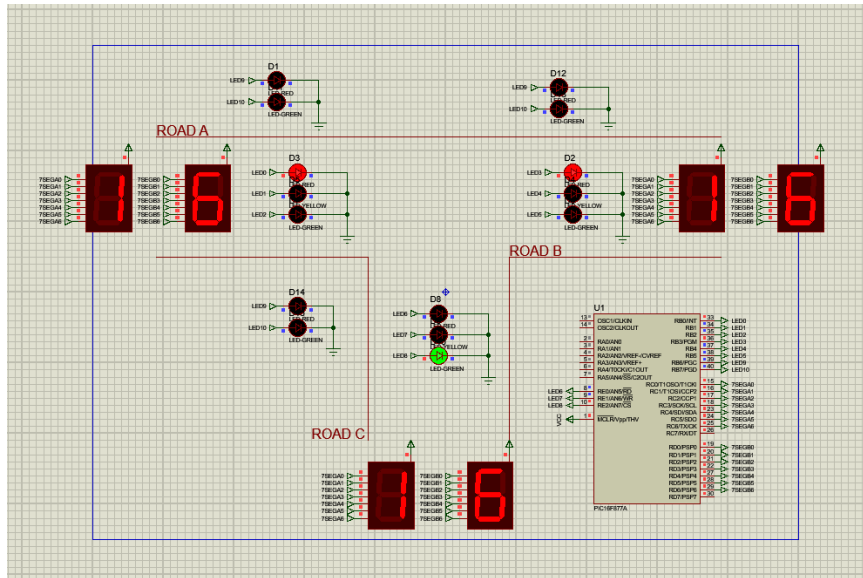


Figure 13: Side Road turns green

Roads A and B now display red signals, whilst road C displays green.

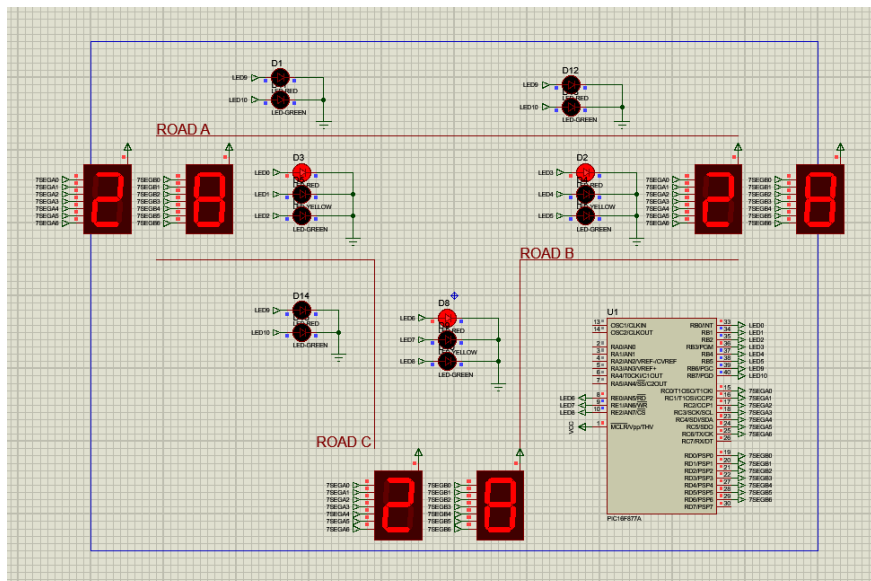


Figure 14: All red transition state before pedestrian crossing phase

All roads display red signals prior to pedestrian crossing phase to clear the road.

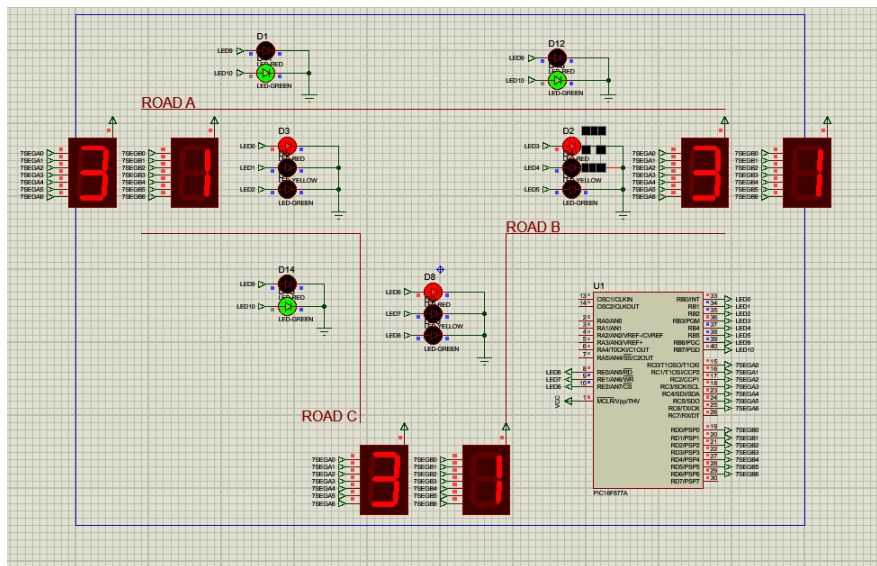


Figure 15: Pedestrian crossing phase

All roads still display red while the pedestrian signal displays green, indicating it is safe to cross.

Future Work

Traffic Light Sensors

One of the most key areas for development is to shift from the current fixed-time control system to an actuated or adaptive control system. This would involve the installation of sensors, such as inductive loops embedded in the road surface, overhead infrared sensors, or even video cameras and image processing algorithms, to detect the actual number and volume of incoming vehicles on each approach.

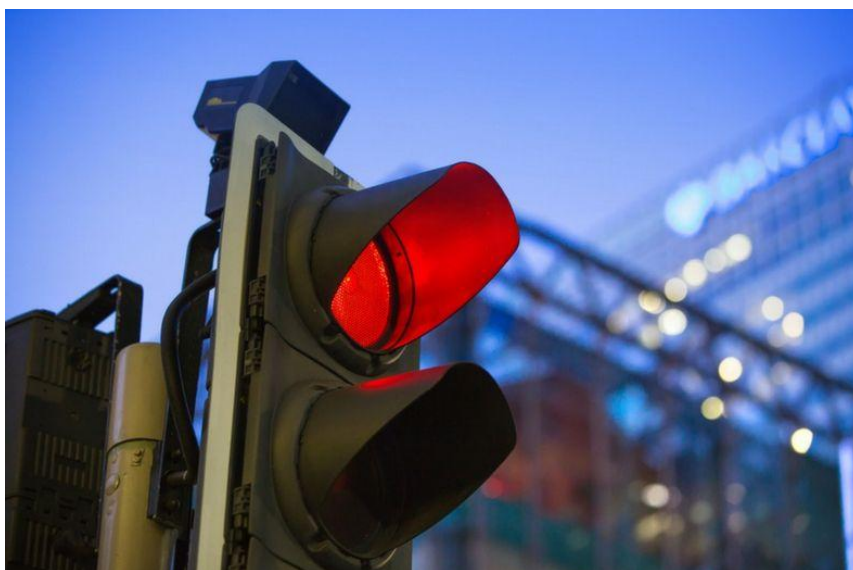


Figure 16: Image of a typical traffic light found in the UK. Sitting on top of it is a camera which reads the traffic [6].

Readings from such sensors can be offered as inputs to the PIC microcontroller. Software logic would also need to be significantly enhanced to modify the green light period durations dynamically based on such real-time traffic demand. For example, green time for the main road can be extended during heavy traffic, or side road phase skipped entirely if no traffic is waiting on road C. This type of adaptive control can cause substantive reductions in average vehicle delay and fuel consumption compared to fixed-time systems, especially with fluctuating traffic conditions.

Pedestrian Crossing Button

Another useful enhancement would be the inclusion of a pedestrian demand button. In the current implementation, the 10-second pedestrian phase repeats automatically in every 40-second cycle regardless of whether there are any pedestrians waiting to cross. This can cause unnecessary delays for vehicular traffic in periods of low pedestrian traffic.



Figure 17: Pedestrian crossing button found in the UK [7]

By adding a push-button connected to an input pin of the microcontroller, the pedestrian phase can be made demand actuated. The software would be reprogrammed to call the pedestrian crossing sequence (stopping all vehicular traffic and displaying the pedestrian green signal) only after the button is pressed.

Conclusion

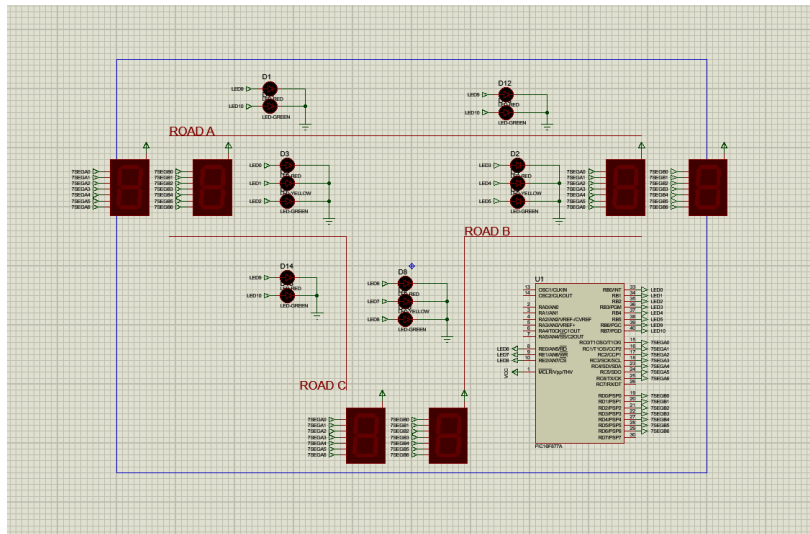
This report has presented the design, development, and iterative testing of a T-junction traffic light controller modelled on Proteus and a PIC16F877A microcontroller. The project employed embedded systems principles to create an effective solution to a common traffic situation. While the initial design had tried to meet specifications, the much-needed simulation test procedure revealed working flaws, such as a missed Amber signal changeover and errant timing on pedestrian phase length, directly attributed to errors in conditional logic and boundary checks within the software.

Through systematic debugging within the simulation setting, such specific issues were identified and fixed by adjusting the control flow logic and counter reset conditions in the firmware. The final test simulation confirmed that the controller could sequence the signals correctly, adhere to precise timings, safely control the pedestrian phase, and display the countdown correctly, validating the improved design.

References

- [1] R. J. Salter and N. B. Hounsell, HIGHWAY TRAFFIC ANALYSIS AND DESIGN. THIRD EDITION, Basingstoke, UK: Palgrave macmillan, 1996.
- [2] Department for Transport, *The Highway Code: Light Signals Controlling Traffic*, London, UK, 2015.
- [3] D. o. Transport, *Traffic Advisory Leaflet*, London, 2012.
- [4] Microchip, *PIC16F87XA Data Sheet*, 2013.
- [5] P. Horowitz and W. Hill, *The Art of Electronics Third Edition*, Cambridge University Press, 2015.
- [6] RAC, "Traffic light cameras: what you need to know," RAC, 8 May 2024. [Online]. Available: <https://www.rac.co.uk/drive/advice/cameras/traffic-light-cameras/>. [Accessed 23 April 2025].
- [7] A. Wong, "Do you press the buttons?," Medium, 18 October 2018. [Online]. Available: <https://medium.com/@anyaw/do-you-press-the-buttons-73a80270280b>. [Accessed 23 April 2025].

Appendix



Appendix 1: Schematic capture from Proteus

```
1  #include <16F877A.h>
2  #use delay(clock=1000000) // 1 MHz clock speed
3
4  // Define LED pins
5  // Traffic Lights 1 (Main Road A)
6  #define LED_RED_A   pin_b0
7  #define LED_YELLOW_A pin_b1
8  #define LED_GREEN_A pin_b2
9  // Traffic Lights 2 (Main Road B - Same as A)
10 #define LED_RED_B   pin_b3
11 #define LED_YELLOW_B pin_b4
12 #define LED_GREEN_B pin_b5
13 // Traffic Lights 3 (Perpendicular Road C)
14 #define LED_RED_C   pin_e0
15 #define LED_YELLOW_C pin_e1
16 #define LED_GREEN_C pin_e2
17 // Pedestrian Lights
18 #define LED_RED_P   pin_b6
19 #define LED_GREEN_P pin_b7
20
21 // 7-Segment Display Digits (Common Cathode - Active High for Segments)
22 unsigned int display[10] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90};
23
24
25 // Function to display two digits on separate 7-segment displays
26 void display_digits(int count) {
27     int tens = count / 10;
28     int ones = count % 10;
29
30     if (tens >= 0 && tens <= 9) {
31         output_c(display[tens]);
32     } else {
33         output_c(0x00); // Turn off display (all segments off for CC)
34     }
35     if (ones >= 0 && ones <= 9) {
36         output_d(display[ones]);
37     } else {
38         output_d(0x00); // Turn off display
39     }
40 }
41
42 // Helper function to set all traffic lights to RED and Pedestrian to GREEN
43 void all_traffic_red_ped_green() {
44     output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A);
45     output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B);
46     output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C);
47     output_low(LED_RED_P); output_high(LED_GREEN_P); // Ped Green ON, Ped Red OFF
48 }
49
50
```

Appendix 2: Part 1 of source code

```

51 void main() {
52     int counter = 0; // Initialize the main counter
53
54     // --- Initialization ---
55     set_tris_b(0x00); // All PORTB pins as output
56     set_tris_c(0x00); // All PORTC pins as output (7-segment Tens)
57     set_tris_d(0x00); // All PORTD pins as output (7-segment Ones)
58     set_tris_e(0x00); // All PORTE pins as output (Traffic C LEDs)
59
60     // Initialize outputs to a safe state (All Traffic Red, Pedestrian Red)
61     output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A);
62     output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B);
63     output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C);
64     output_high(LED_RED_P); output_low(LED_GREEN_P); // Ped Red ON, Ped Green OFF
65     output_c(0x00); // Turn off 7-seg C (assuming common cathode)
66     output_d(0x00); // Turn off 7-seg D (assuming common cathode)
67
68
69     // --- Main Loop ---
70     while (1) {
71
72         // 1. Display the current count
73         display_digits(counter);
74
75         // 2. Determine and set light states based on the counter
76         if (counter >= 30) {
77             // --- Pedestrian Phase (Seconds 30-39) ---
78             // All vehicle traffic lights RED, Pedestrian GREEN
79             all_traffic_red_ped_green();
80
81         } else {
82             // --- Vehicle Phase (Seconds 0-29) ---
83             // Pedestrian RED is ON, Pedestrian GREEN is OFF for this entire phase
84             output_high(LED_RED_P);
85             output_low(LED_GREEN_P);
86
87             // Now, set only the vehicle traffic light states (A, B, C)
88             if (counter >= 27) { // State 8: All Traffic Red (Transition before Ped) (27-29)
89                 output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A);
90                 output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B);
91                 output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C);
92             } else if (counter >= 25) { // State 7: A/B Red, C Amber (25-26)
93                 output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A); // A Red
94                 output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B); // B Red
95                 output_low(LED_RED_C); output_high(LED_YELLOW_C); output_low(LED_GREEN_C); // C Amber
96             } else if (counter >= 16) { // State 6: A/B Red, C Green (16-24)
97                 output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A); // A Red
98                 output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B); // B Red
99                 output_low(LED_RED_C); output_low(LED_YELLOW_C); output_high(LED_GREEN_C); // C Green
100             } else if (counter == 15) { // State 5: A/B Red, C Red & Amber (15)
101                 output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A); // A Red
102                 output_low(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B); // B Red
103                 output_high(LED_RED_C); output_high(LED_YELLOW_C); output_low(LED_GREEN_C); // C Red+Amber
104             } else if (counter >= 13) { // State 4: All Traffic Red (Transition) (13-14)
105                 output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A);
106                 output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B);
107                 output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C);
108             } else if (counter >= 11) { // State 3: A/B Amber, C Red (11-12)
109                 output_low(LED_RED_A); output_high(LED_YELLOW_A); output_low(LED_GREEN_A); // A Amber
110                 output_low(LED_RED_B); output_high(LED_YELLOW_B); output_low(LED_GREEN_B); // B Amber
111                 output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C); // C Red
112             } else if (counter >= 2) { // State 2: A/B Green, C Red (2-10)
113                 output_low(LED_RED_A); output_low(LED_YELLOW_A); output_high(LED_GREEN_A); // A Green
114                 output_low(LED_RED_B); output_low(LED_YELLOW_B); output_high(LED_GREEN_B); // B Green
115                 output_low(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C); // C Red
116             } else if (counter == 1) { // State 1: A/B Red & Amber, C Red (1)
117                 output_high(LED_RED_A); output_high(LED_YELLOW_A); output_low(LED_GREEN_A); // A Red+Amber
118                 output_high(LED_RED_B); output_high(LED_YELLOW_B); output_low(LED_GREEN_B); // B Red+Amber
119                 output_low(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C); // C Red
120             } else { // State 0: All Traffic Red (Transition) (0)
121                 output_high(LED_RED_A); output_low(LED_YELLOW_A); output_low(LED_GREEN_A);
122                 output_high(LED_RED_B); output_low(LED_YELLOW_B); output_low(LED_GREEN_B);
123                 output_high(LED_RED_C); output_low(LED_YELLOW_C); output_low(LED_GREEN_C);
124             }
125         } // End of Vehicle Phase logic
126
127         // 3. Wait for 1 second
128         delay_ms(1000);
129
130         // 4. Increment counter and reset if it reaches 40
131         counter++;
132         if (counter >= 40) {
133             counter = 0; // Reset the counter for the next cycle
134         }
135     } // End while
136 } // End main

```

Appendix 3: Part 2 of source code