



TRAFFIC LIGHT CONTROL SYSTEMS

Electronics 1 - ACDB5002

Abstract

This project investigates the design, simulation, and theoretical underpinning of a traffic light control system for a single-lane T-junction using both traffic flow theory and embedded system development. Drawing from macroscopic traffic models, key variables such as flow rate, density, and saturation were used to evaluate the efficiency and suitability of a fixed 40-second signal cycle. The system was found to be well within operational capacity for typical urban traffic volumes, ensuring minimal delay and no long-term queue formation. A practical implementation was developed in Proteus using the PIC18F87K90 microcontroller. The design controlled three traffic light heads and a two-digit 7-segment countdown display, governed by a finite state machine and interrupt-driven timing mechanism.

Jamie Hamilton-Swingewood

2308342@student.uwtsd.ac.uk

Contents

Traffic Theory and Control Measures	2
Core Traffic Variables	3
Space-Time Averaged Relationships:	3
Application of Theory: Queues, Timing and Signal Optimisation	4
Behind the Signals	6
Traffic Signal Controller	6
Proteus Simulation – Modelling a Traffic Control Unit.....	7
Circuit Design.....	7
Design Flow.....	9
Summary of code	10
Simulation running.....	11
Conclusion	17
Bibliography.....	18
Appendix	20
Photos of circuit	20
Code in Full.....	22

Traffic Theory and Control Measures

Traffic science and flow theory underpin the conscious designs and naturally arising interactions that pedestrians and vehicles have on roadways big and small. Like many areas of science, the resolution or level of detail being observed will change the representation of variables at the given level. For traffic theory this is broadly broken down into two categories macroscopic and microscopic models.

Macroscopic models takes the 'lower resolution' view of traffic with it being treated as a continuous flow of mass very much analogous to fluid dynamics, aggregating variables like traffic density, traffic flow rate, and average speed as a whole rather than an individual vehicle or pedestrian. The macroscopic view allows phenomena like bottlenecking and Traffic shockwaves to be modelled. These are commonly used in high traffic areas such as motorways with variable speed limits and or extra lanes as in areas with lots of traffic or sudden traffic disturbances causing shockwave like effects, can be eased by slowing traffic as a whole or increasing the average flow rate of traffic by opening a new lane. It's also less computationally demanding to take flow rates or densities of traffic then trying to individually map each and every pedestrian or vehicle that is out and about in real-time. ^[1,2,4]

Network level models, an offshoot of macroscopic models that breaks up a given area into relevant nodes that makeup a network, so that areas of roughly equivalent behaviour can be appropriately mapped. One can take a busy urban area to be one node on a network with surrounding suburban areas or carriageway as different nodes on the same network. The accumulation of vehicles can be more easily mapped as different nodes will have varying traffic impingement and flow rates throughout the network. ^[5]

Microscopic models are very much the inverse of macroscopic models, focusing on the individual interactions between obstacles, vehicles and pedestrians. These models are often deployed at specific areas of high traffic for example a troublesome intersection all roads with high collision rates to describe the phenomena underpinning individual behaviours like queuing at a specific junction in detail. Here individual spacing acceleration are typically measured and naturally are computationally intensive to map over large areas. ^[1-4]

Submicroscopic models are where the system state of a vehicle is outlined by many subfactors affecting the vehicle as a whole. A few examples are, throttle position, grip of tyres, engine response, braking response, reaction time of a person, or a person's fitness to drive. Though important these are too granular for the scope of this exercise and so are not discussed in detail. ^[5]

In practical application a mixture of macro and microscopic models are required for effective city planning and traffic management with these blended approaches being known as

Mesoscopic models more generally. With areas or clusters of vehicles being grouped statistically over individually tracked or infinitely free flowing entities. Regardless of scale the three key variables underpinning traffic flow theory are, **flow rate**, **speed**, **density**, **headway**, and **spacing**.^[1-4]

Core Traffic Variables

Regardless of the modelling approach, five fundamental variables underpin traffic flow theory:

- **Flow rate** (q): number of vehicles passing a point per hour
- **Speed** (u): average travel speed of vehicles
- **Density** (k): number of vehicles per kilometre
- **Headway** (h): time between successive vehicles
- **Spacing** (s): distance between successive vehicles

These variables are interconnected through both macroscopic and microscopic relations.

$$\text{Macroscopic: Traffic Flow } \left(\frac{\text{vehicles}}{\text{hour}} \right) = \text{Average Speed } \left(\frac{\text{km}}{\text{hour}} \right) \times \text{Density } \left(\frac{\text{vehicles}}{\text{km}} \right), \quad \{1.0\}$$

$$q = ku, \quad \{1.1\}$$

$$\text{Microscopic headway (s)} = \text{Spacing (m)} \times \text{Speed (ms}^{-1}\text{)}, \quad \{1.2\}$$

$$h = sv, \quad \{1.3\}$$

Space-Time Averaged Relationships:

These variables can also be defined using integral expressions, which account for variations over time and space:

- Flow rate (vehicles/hour):

$$q = \frac{3600}{\bar{h}}, \quad \text{where } \bar{h} = \frac{1}{T} \int_0^T h(t) dt, \quad \{2.0\}$$

- Vehicle density (vehicles/km)

$$k = \frac{1000}{\bar{s}}, \quad \text{where } \bar{s} = \frac{1}{L} \int_0^L s(x) dx, \quad \{2.1\}$$

- Average speed (km/h)

$$u = 3.6\bar{v}, \quad \text{where } \bar{v} = \frac{1}{LT} \int_0^T \int_0^L v(x, t) dx dt, \quad \{2.2\}$$

Here:

- T is the duration of observation
- L is the length of the road segment

- $h(t)$, $s(x)$, and $v(x,t)$, represent instantaneous headway, spacing, and speed respectively
- Constants 3600 and 1000 are used for unit conversion (seconds to hours, meters to kilometres)
- 3.6 converts average speed from m/s to km/h

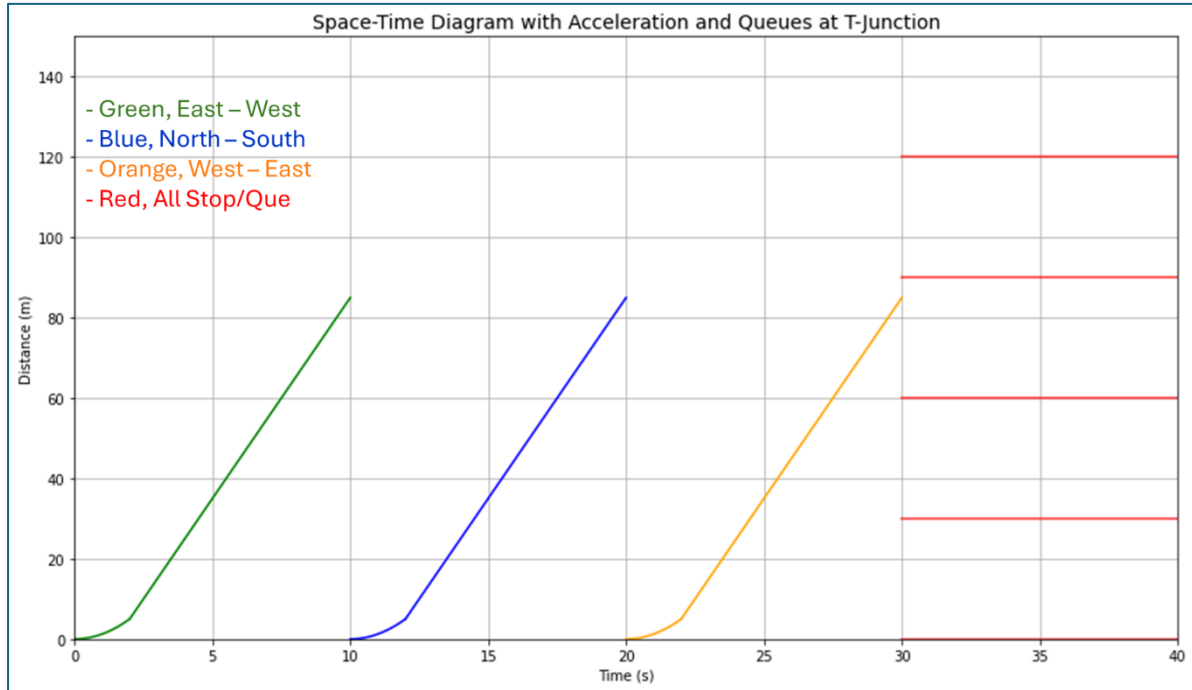


Figure 1: A simple Space-Time diagram showing the idealised movement of 3 cars at the 3 different points of the T-junction. With each car gradually accelerating from 0 to 10m/s with the red bars more for visual acuity that there are no cars in the junction and all have stopped. The traffic cycle follows 40s for each entry green light to the junction to have a 10s green cycle, with the final 10s being all red. This type of space-time diagram for characteristics of a traffic stream is typical in traffic flow theory as seen in [4] Figure 2.1.

Application of Theory: Queues, Timing and Signal Optimisation

In line with the briefing for this research the road setup and designated signal timings are for a T-junction with 10 second intervals and a 10 second all stop period. Before designing a system, it is prudent to make sure that in average traffic flow, they're not unreasonable number of queues and cars can clear the junction within the given cycle. According to research by R.M. Kimber et al [8] the UK saturation flowrate S_{FR} for a single lane road is around 2080 veh/hr (0.58 veh/s). Given that for a typical road the actual traffic flow is approximately 90 veh/hr taken from the annual average daily flow (AADF) for minor roads in urban areas which see an average of 2.1 thousand vehicles per day, being $87.5 \approx 90$ veh/hour. ^[9]

Primary flow considerations, calculating the maximum throughput per 40s cycle.

Arrival flow:

$$q = \frac{90 \text{ veh}}{3600 \text{ sec}} = 0.025 \text{ veh/sec}$$

Capacity per phase, C :

$$C = S_{FR} \cdot \frac{g}{T}, \quad \{3.0\}$$

Where:

- S_{FR} is saturation flow rate $\approx 0.58\text{veh/s}$
- g is green time = 10s
- T is total cycle time = 40s

$$0.58 \times \frac{30}{40} = 0.435 \frac{\text{veh}}{\text{s}}, \quad \{3.1\}$$

Although this is not correct as only 1 lane is moving at a time therefore, a single lane will experience 1/3 that capacity thus the per lane capacity is:

$$C = 0.145\text{vehs}^{-1}, \{3.2\}$$

From this and using the deterministic equation for queuing is the first factor from Websters formulae^[10] for delay and is equation {3.3} which describes the aver

$$\bar{d} = \frac{T(1 - \frac{g}{T})^2}{2(1 - \frac{q}{S_{FR}})}, \{3.3\}$$

Where:

$\bar{d}$ = Average delay

$$\bar{d} = \frac{40(1 - \frac{10}{40})^2}{2(1 - \frac{0.025}{0.58})} = 11.76 \text{ seconds}$$

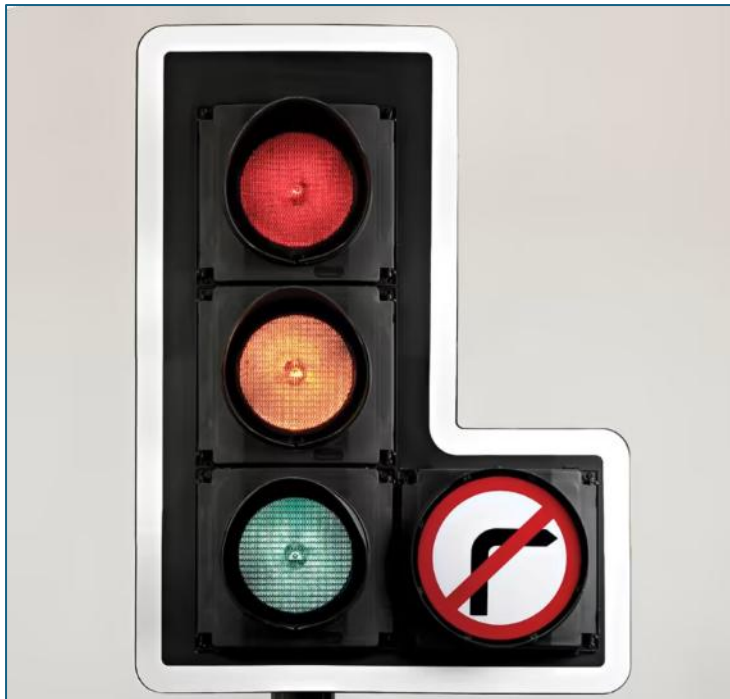
This would seem to pose a problem as our green movement cycle is only 10s per lane, however the delays are only temporary and cyclical as the arrival flow rate is much lower then the calculated saturation flow rate.

$$q < C, \quad 0.025\text{veh/s} < 0.145\text{veh/s}$$

From these calculations it can be said that the signal timings of a 10/10/10/10 split are appropriate for the type of junction and expected traffic levels with the ability to cope with a nearly sixfold increase in arrival flow. Only when the arrival flow exceeds 0.145veh/sec will the capacity of the signals be overrun and backups further down the traffic stream be caused. It is very unlikely that a minor single lane Junction will see an extra 10000 cars over a day, with that level of traffic the specifics of optimising programming and optimisation of signals will likely be on the lower end of priorities for clearing highway disruptions.

In modern systems a dynamic signalling process would be implemented with sensors

Behind the Signals



Behind the familiar sights of the coloured lights that motorists see, is a wide array of hardware and software working in unison.

Figure 2: The British traffic light, found in abundance across the UK. Brought to life by renowned industrial designer David Mellor in the 1960s and services the country's signalling needs to this day. [11]

Traffic Signal Controller

The traffic signal controller is the “brain” of the intersection. It is an embedded system that executes the signal timing plans and controls the sequence of lights, modern controllers are usually programmable and can be adjusted remotely. The data and instruction commands from a control unit are fed onto a larger network, this is monitored in specialist control rooms by stakeholders like National Highways in the case of larger motorways/A-roads and local authorities or specialist event management teams for large events such as the Olympics. The specific functioning of a single intersection is unlikely to be managed by an operator until there is need, with the majority of the monitoring of a road system network is handled by software such as the aptly named Network Monitor by causeway one network.^[12]

The signal controller is made up of two primary parts being the controller unit and controller cabinet, respectively. The controller unit in a modern traffic light are typically microprocessor-based devices (much like the PIC microcontroller in the proteus simulation to follow). The controllers will follow standardised designs for interoperability and ease of servicing, the specific design guidelines are outlined in TD 101 Traffic signalling systems (design)^[13]. Particularly section 2.1, 2.2, 3.7 and 3.8. The controller cabinet contains the controller unit itself alongside auxiliary equipment that enables the all-weather operation of the signals. In the roadside controller cabinet, there will be the typical electrical systems such as a load switches and a power supply to actually power on the system converting the mains high voltage mains AC



Figure 3: A traffic control cabinet coming in the north American flavour.[14] The specific makes and models of the equipment can and does differ from country to country, but the general layout is common.

to lower voltage DC. In the cabinet also is a conflict monitor or malfunction monitoring unit (MMU), that monitors the output signals of the of the control unit to ensure the correct signals are given and will override to reset the system if need be. MMUs are critical pieces of equipment ensuring redundancy that is critical for all essential infrastructure. Becoming more commonplace in a signal control cabinet are communication devises like a network switch or cellular based system that enable a control unit to join a traffic monitoring network either

nationally or locally allowing for active monitoring and coordination of signals across a wider area.

Proteus Simulation – Modelling a Traffic Control Unit

Circuit Design

The first step in designing a functional Traffic Control Unit within Proteus was selecting an appropriate microcontroller (MCU) capable of handling the required inputs and outputs. The circuit needed to control three full traffic light heads—each consisting of green, yellow, and red LEDs, totalling 9 digital output pins. Additionally, a two-digit 7-segment display was incorporated to indicate countdown timers for traffic phases, requiring 14 digital I/O pins (7 segments per digit, common cathode or anode depending on configuration, with either shared or independent control).

This brings the absolute minimum number of required I/O pins to 23. To ensure the design would remain scalable and flexible for future enhancements (e.g., pedestrian crossing buttons, vehicle sensors, communication modules), a microcontroller with significantly more pins is desirable.

The PIC18F87K90 microcontroller was selected as the core controller. It provides^[15]:

- 80 I/O pins (including analog-capable pins),
- Built-in multiple timers (useful for traffic phase timing),
- LCD driver capability (though not used here),
- And low power modes, which are beneficial in embedded systems.

In Proteus, the microcontroller and 7 segment LEDs were wired as seen in figure below:

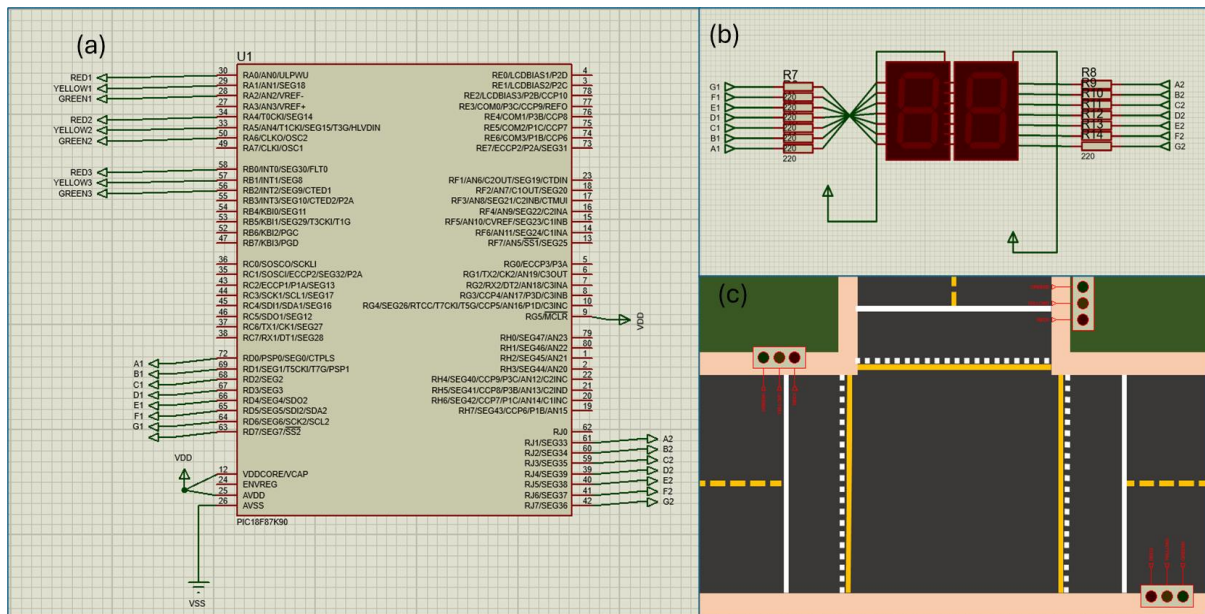
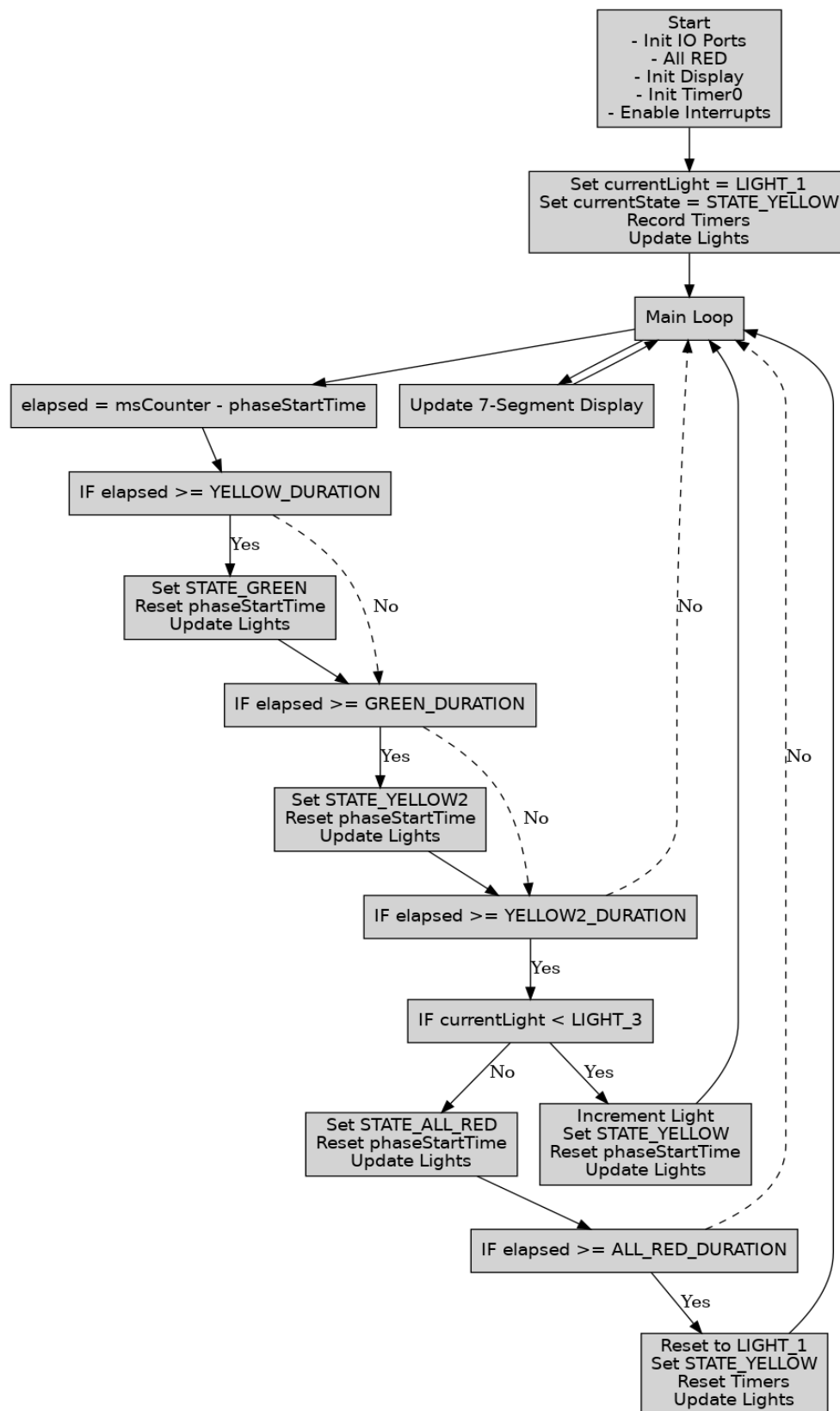


Figure 4: (a) Is the microcontroller itself. (b) is the two 7-segment displays and (c) the traffic light LEDs on the road mock up. Higher fidelity images can be found in the appendix.

Using the PIC18F87K90 allowed not only coverage of the immediate functional requirements but also offered headroom to incorporate additional features such as real-time clock integration for time-of-day signal phasing, UART/SPI/I2C communication for linking with central traffic control systems. Interrupt-based event handling for responsiveness to dynamic inputs (e.g., emergency vehicle priority), and potential for integrating sensors (e.g., vehicle detection or pedestrian presence).

Design Flow



Summary of code

Full code used is available in the appendix, the compiler used was microchips own MPLabs XC compiler.

- I/O ports are configured to output mode.
- All traffic signals are initially set to red by default (setAllRed()).
- A millisecond timer (Timer0) is set up using the ISR (__interrupt()), generating 1 ms ticks using an 8-bit timer and a 1:32 prescaler.
- Two global counters are initiated: +
 - msCounter tracks time in milliseconds.
 - displayCycleStart is used to track the progression of the current 40-second cycle.

Traffic Light State Machine

Rotates through the following sequence every 40 seconds.

Light Phase	Duration	Function
Yellow	1s	Prepares for green transition
Green	8s	Active passage
Yellow2	1s	Transition back to red
All Red	10s	Safety buffer after all 3 lights cycle

Light 1, 2, and 3 take turns being active. Each goes through Yellow → Green → Yellow2 → Red. STATE_ALL_RED marks the end of each complete cycle, ensuring a safe pause before restarting. Using the msCounter, the system measures elapsed time in each phase and switches states once the designated durations are reached (e.g., 1000 ms for yellow).

Simulation running

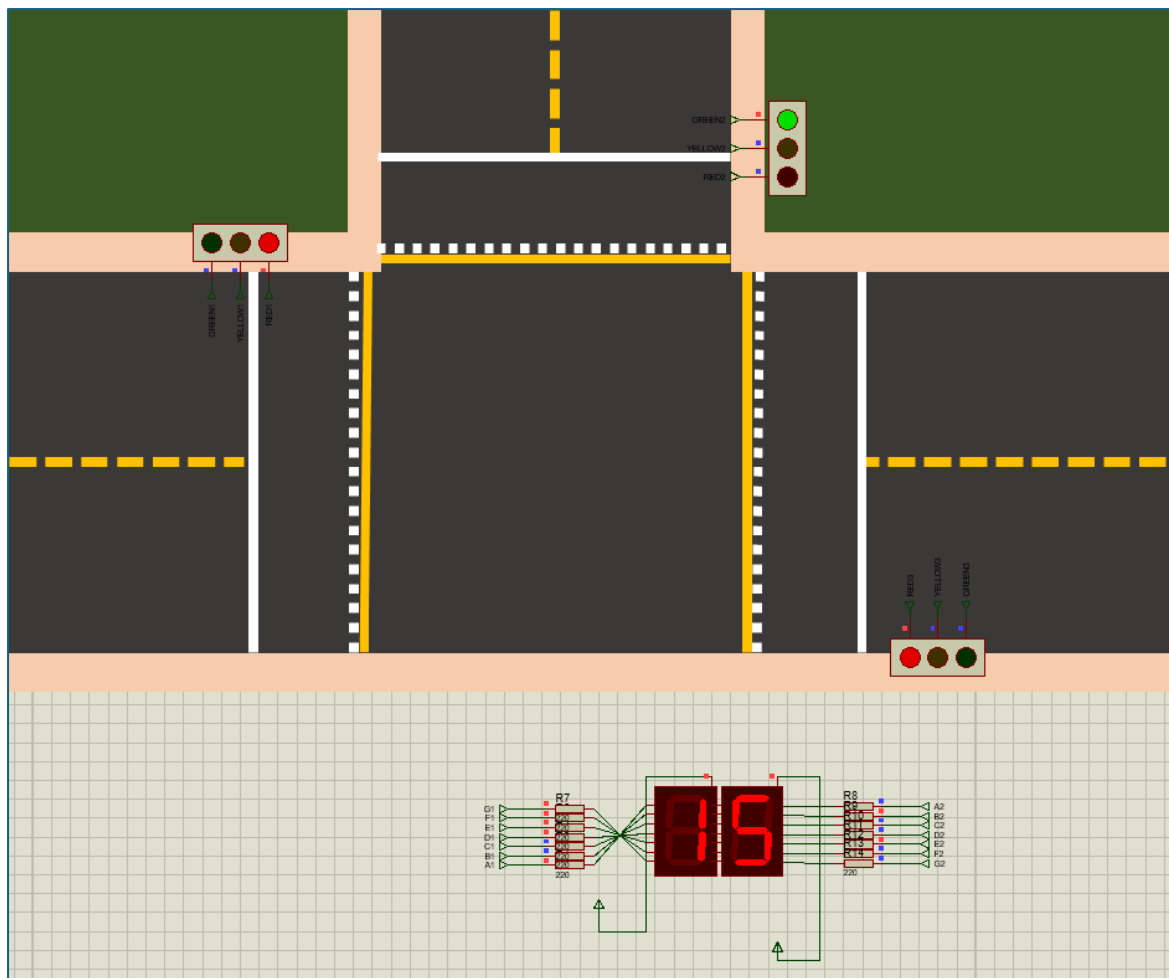


Figure 5: Mid-phase at the 15s mark, correct green signal at traffic light 2.

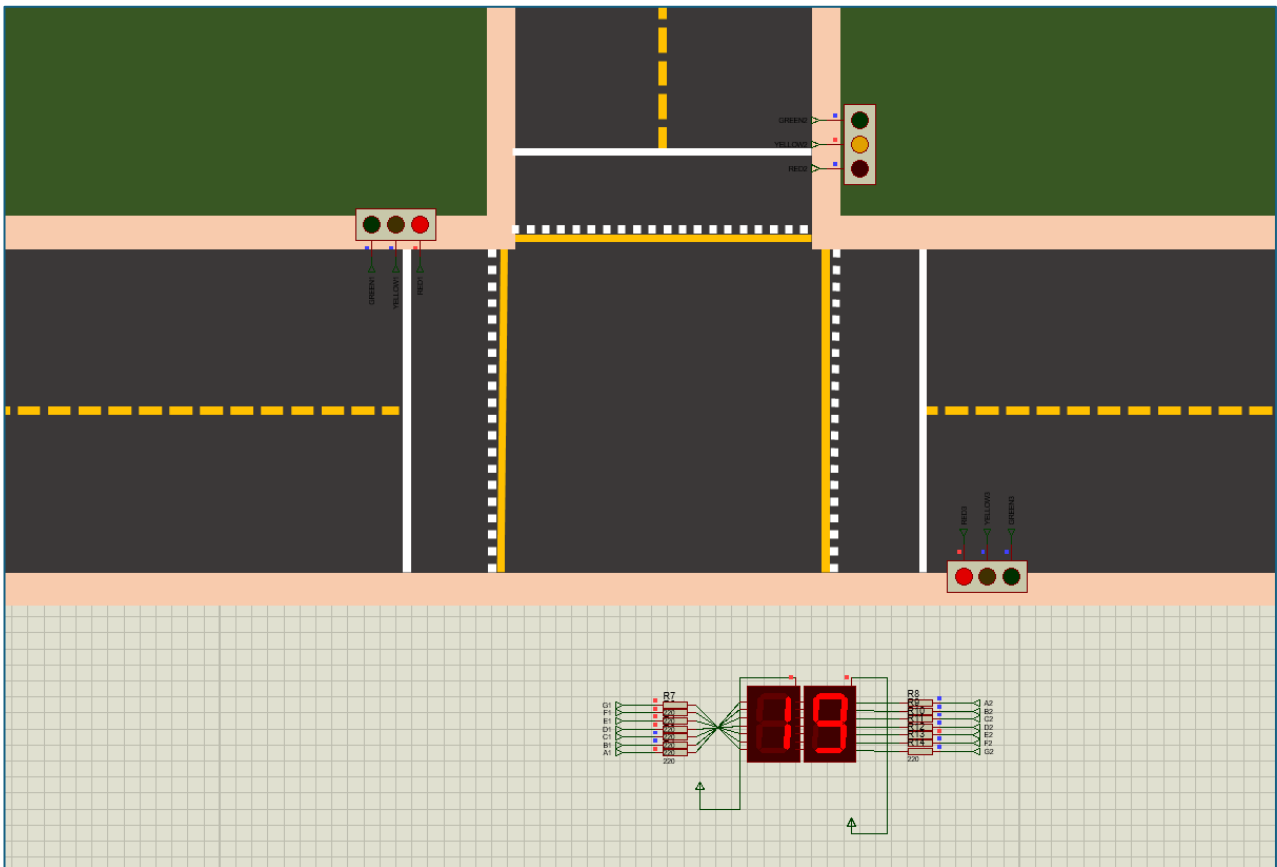


Figure 6: The light transitioning out of green into the 2nd yellow state for the switch to the next traffic light.

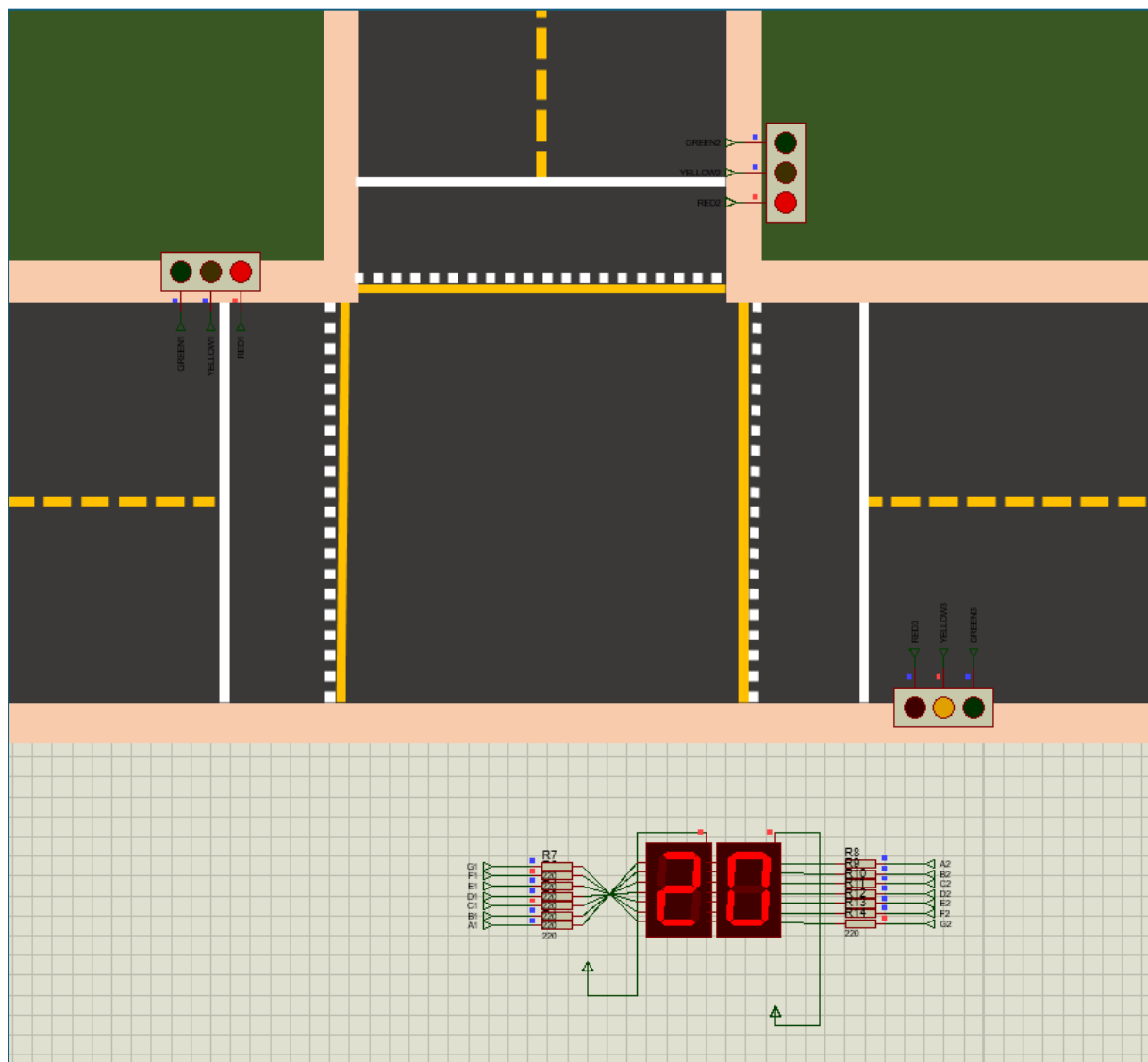


Figure 7: The next traffic light correctly transitioning to yellow.

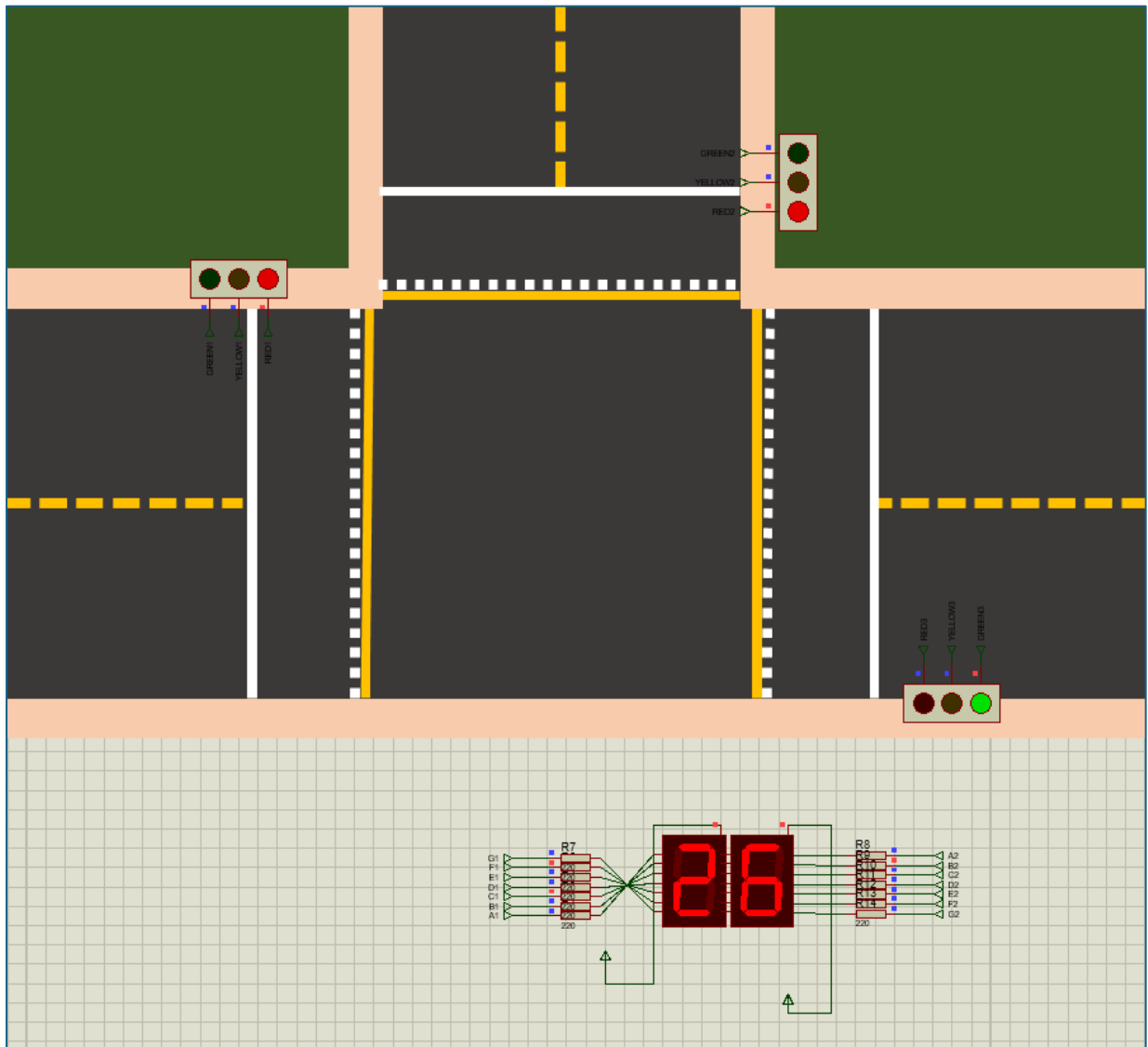


Figure 8: Correct mid cycle green.

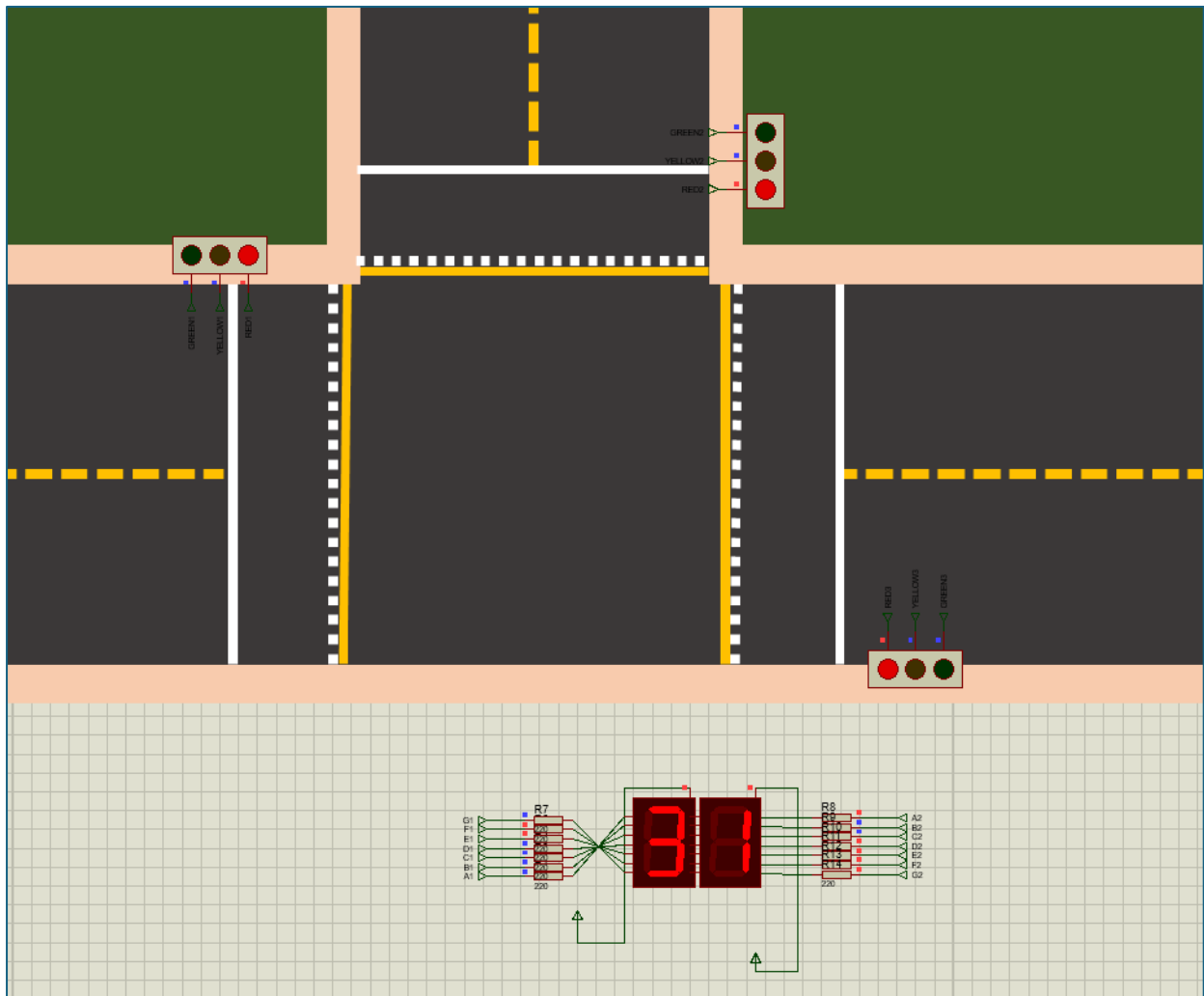


Figure 9: All red signal correct.

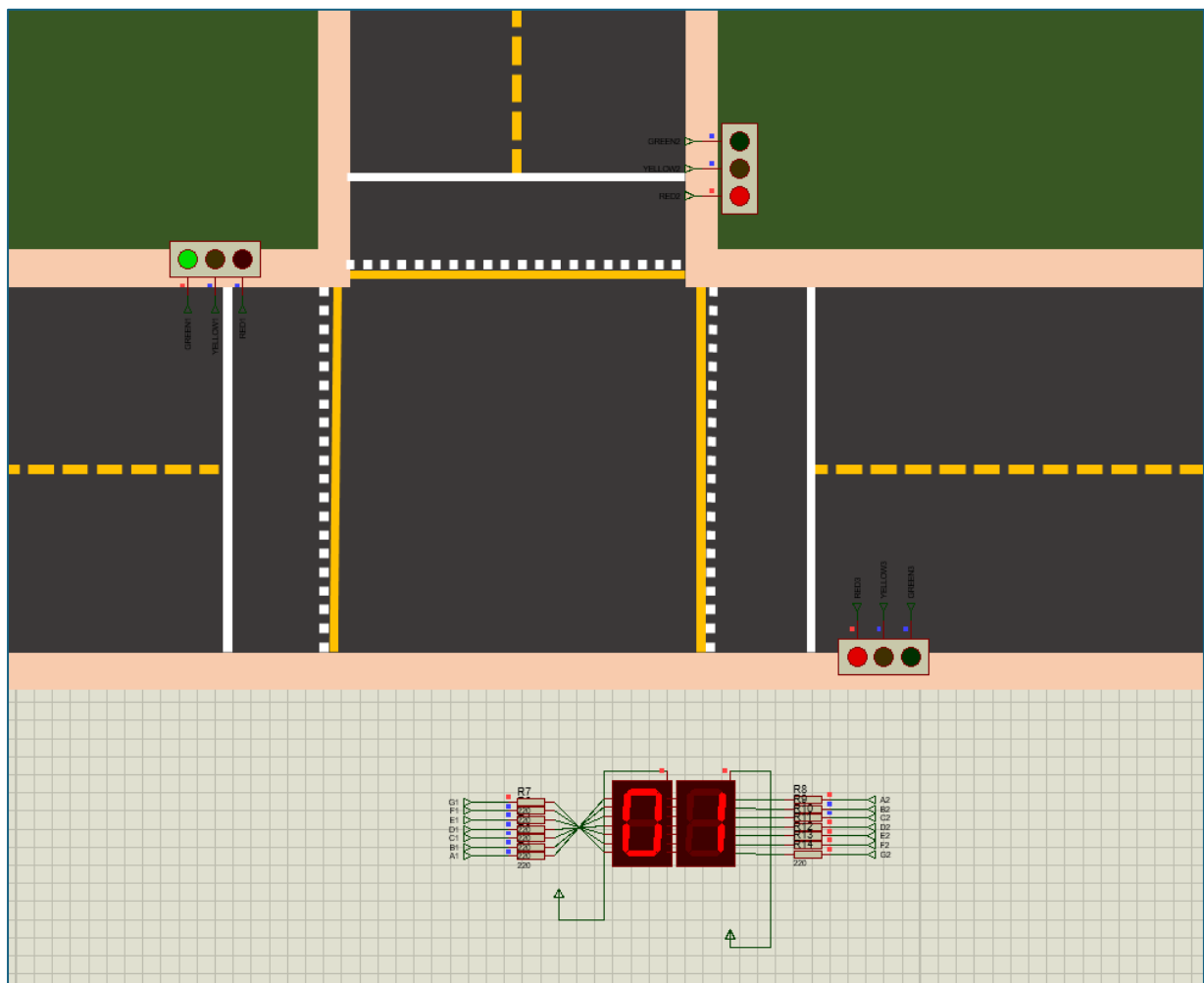


Figure 10: The system has correctly reset and has now gone back to the first traffic light at simulation time of 41.6s.

Conclusion

This report has explored the theoretical foundations and practical implementation of a traffic light control system tailored for a signalised T-junction. By applying principles of traffic flow theory, such as flow rate, density, and signal cycle timing, we established that the chosen 10/10/10/10 second phase split is sufficient for managing minor road volumes typically seen in urban settings. Calculations confirmed that under average traffic loads (≈ 90 veh/hr), the system operates well below saturation, ensuring that delays remain short, and queues do not build up over successive cycles in regular traffic.

The system was then brought to life through a Proteus-based simulation using the PIC18F87K90 microcontroller. The circuit was designed to drive three independent traffic lights and a two-digit 7-segment countdown display. The software implemented a state-driven approach, leveraging a finite state machine and Timer0-based interrupt-driven timing to ensure accurate phase transitions. Key functionality included dynamic light control, countdown display updates, and seamless reset logic between cycles — all synchronised over a 40-second total cycle.

The design proved both scalable and efficient. The use of a high-pin-count microcontroller allowed room for future enhancements such as vehicle detection sensors, pedestrian crossing integration, or adaptive signal timing. Overall, this project has demonstrated the full design loop from traffic theory analysis through to embedded system simulation and functional signal logic, offering a robust foundation for future developments in traffic control system design.

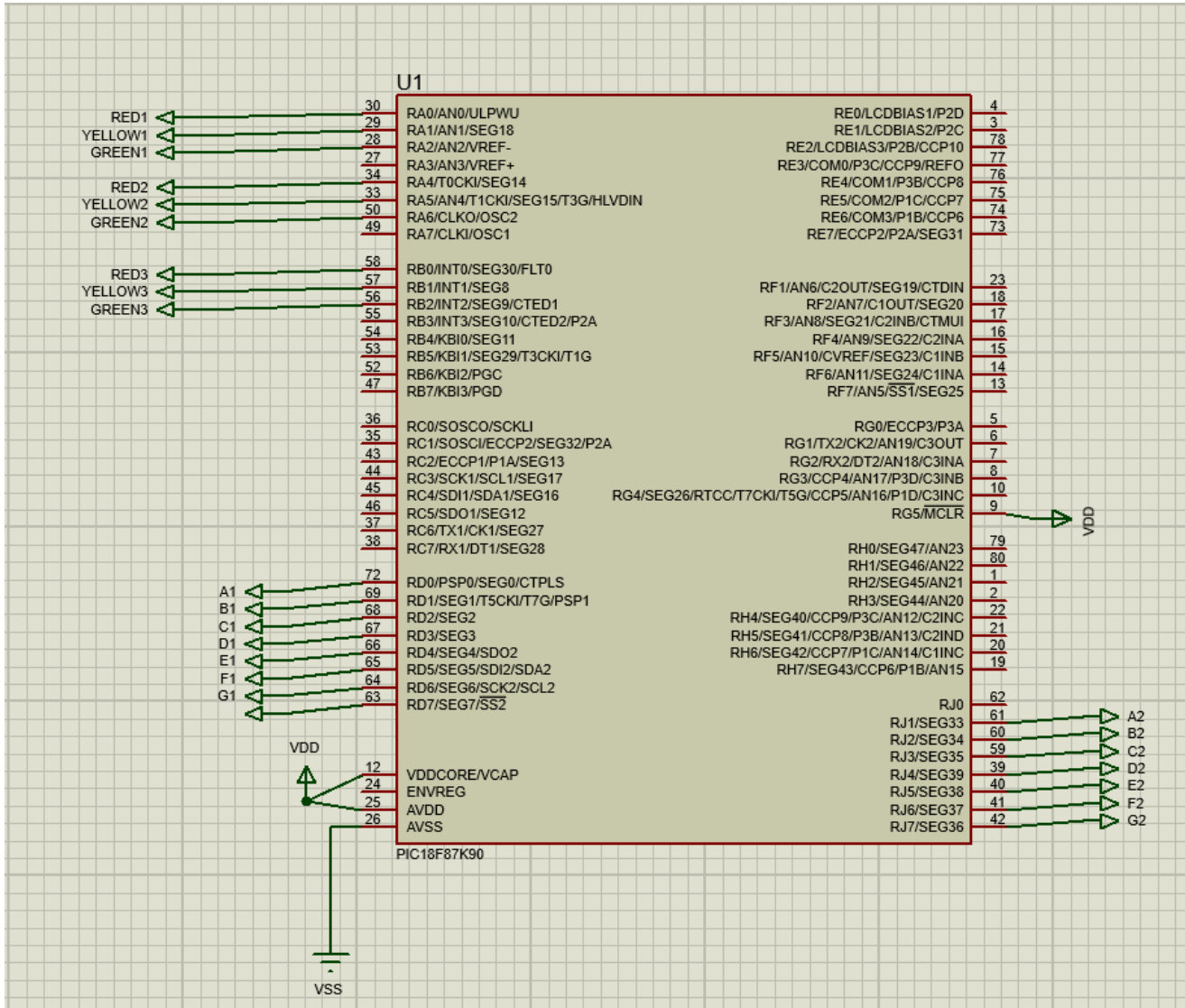
Bibliography

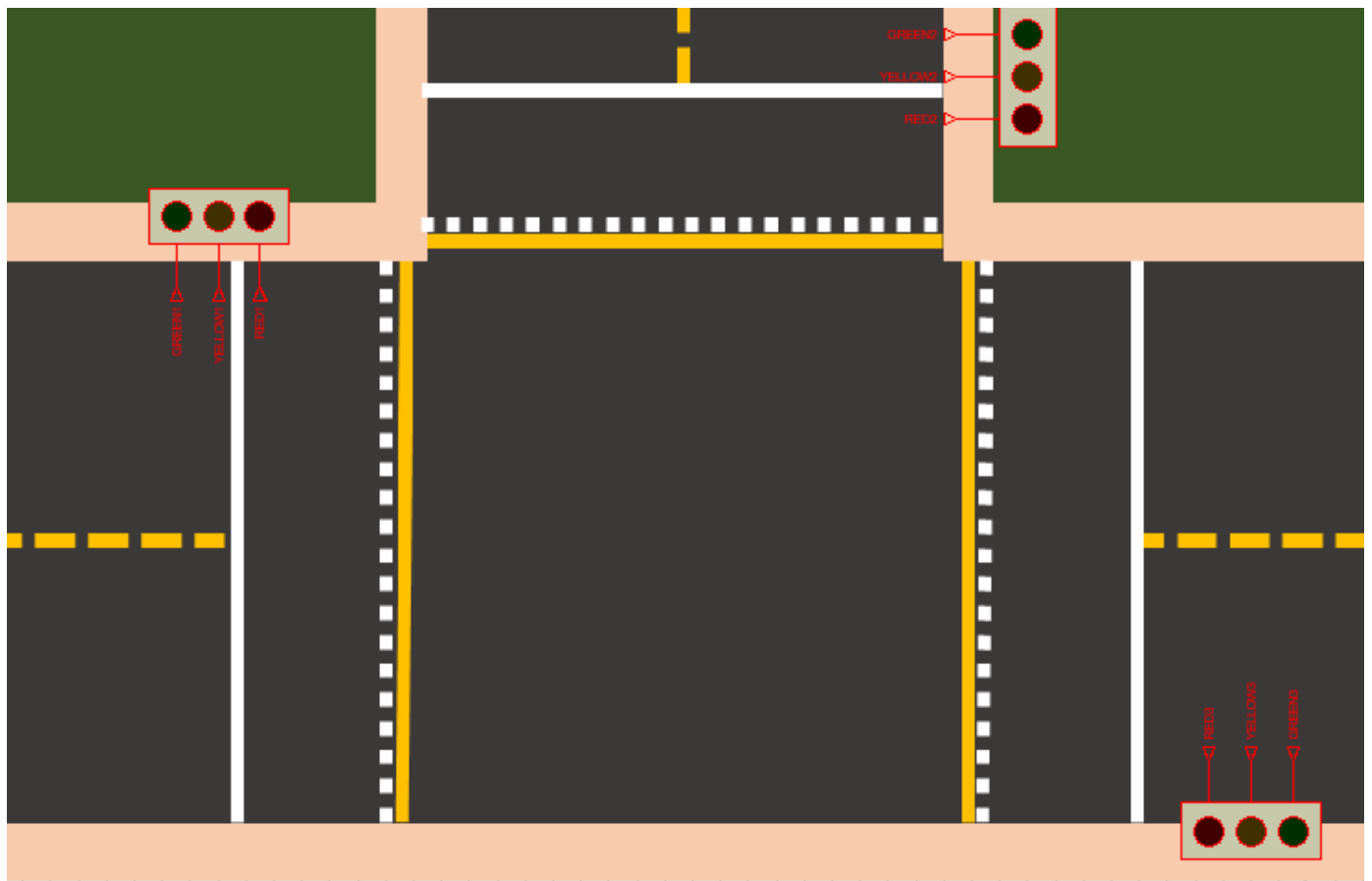
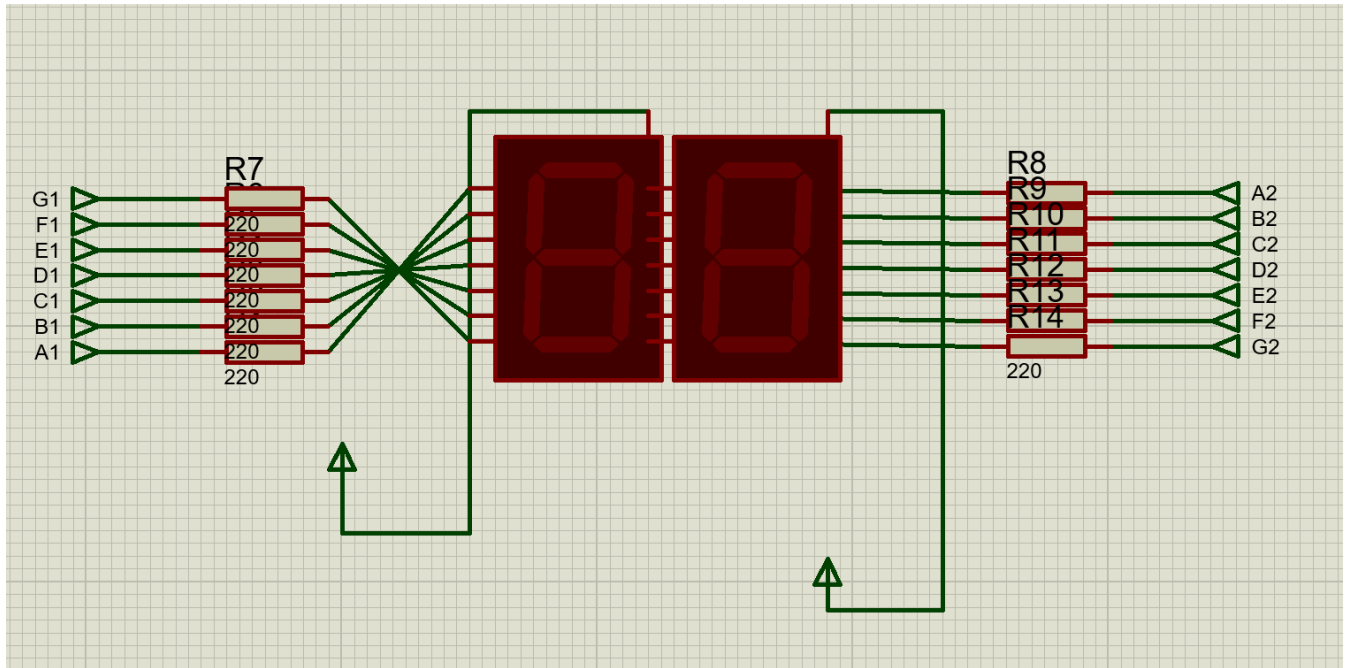
- [1] T. Bellemans, B. De Schutter, and B. De Moor, "Models for traffic control Models for traffic control," 2002. Available: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2f470510c5c31257700b1aba52641abcfb7ee3f2>
- [2] T. Fan, S. C. Wong, Z. Zhang, and J. Du, "Stochastic Lighthill-Whitham-Richards traffic flow model for nonlinear speed-density relationships," *Transportmetrica B Transport Dynamics*, vol. 12, no. 1, Oct. 2024, doi: <https://doi.org/10.1080/21680566.2024.2419402>.
- [3] M. Akanegawa, Y. Tanaka, and M. Nakagawa, "Basic study on traffic information system using LED traffic lights," *IEEE Transactions on Intelligent Transportation Systems*, vol. 2, no. 4, pp. 197–203, Jan. 2001, doi: <https://doi.org/10.1109/6979.969365>.
- [4] U.S. Department of Transport, Federal Highway Administration Research and Technology, Revised Monograph on Traffic Flow Theory. U.S. Department of Transport, 2017, p. Chapter 2. Accessed: Apr. 18, 2025. [Online]. Available: <https://www.fhwa.dot.gov/publications/research/operations/tft/>
- [5] V. L. Knoop, Introduction to Traffic Flow Theory, 2nd ed. Delft University of Technology, 2018, p. 4. Available: https://victorknoop.eu/research/book/Knoop_Intro_traffic_flow_theory_edition2.pdf
- [6] A. Awami and Z. Yusupov, "Microcontroller based traffic light control system," 2017. Available: <https://isites.info/PastConferences/ISITES2017/ISITES2017/papers/A6-ISITES2017ID187.pdf>
- [7] A. P. AKGÜNGÖR and E. KORKMAZ, "Investigating Parameter Interactions with the Factorial Design Method: Webster's Optimal Cycle Length Model," *Tehnicky vjesnik - Technical Gazette*, vol. 25, no. Supplement 2, Sep. 2018, doi: <https://doi.org/10.17559/tv-20170908185847>.
- [8] R. Kimber, M. McDonald, and N. Hounsell, "THE PREDICTION OF SATURATION FLOWS FOR ROAD JUNCTIONS CONTROLLED BY TRAFFIC SIGNALS," 1986. Available: <https://www.eastleigh.gov.uk/media/8592/cd58.pdf>

- [9] Department for Transport, "Road traffic estimates in Great Britain, 2023: Traffic in Great Britain by road type," GOV.UK, May 22, 2024. <https://www.gov.uk/government/statistics/road-traffic-estimates-in-great-britain-2023/road-traffic-estimates-in-great-britain-2023-traffic-in-great-britain-by-road-type> (accessed Apr. 23, 2025).
- [10] L. Mindur and G. Sierpiński, "Webster's delay formula revision by M+/G+/1 queuing model usage," Logistics and Transport, 2017.
- [11] David Mellor Design Ltd, "About Us," Davidmellordesign.com, 2017. <https://www.davidmellordesign.com/about/#:~:text=David%20Mellor's%20concern%20with%20design,design%20is%20still%20in%20use.> (accessed Apr. 24, 2025).
- [12] Causeway Technologies, "Network Monitor | The power of a traffic control centre ," One.network, 2025. <https://uk.one.network/monitor/network-monitor> (accessed Apr. 24, 2025).
- [13] Standards for Highways, "TD 101 - Traffic signalling systems (design)," Standardsforhighways.co.uk, 2023. <https://www.standardsforhighways.co.uk/search/a50aa56a-ffb0-4eac-8d3b-8b380cd0d636> (accessed Apr. 23, 2025).
- [14] Imgur, "This is what the inside of a traffic signal control cabinet looks like," Imgur, 2015. <https://imgur.com/this-is-what-inside-of-traffic-signal-control-cabinet-looks-like-8fWLjzM> (accessed Apr. 24, 2025).
- [15] Microchip, "PIC18F87K90," Microchip.com, 2025. <https://www.microchip.com/en-us/product/pic18f87k90> (accessed Apr. 24, 2025).

Appendix

Photos of circuit





Code in Full

```
#include <xc.h>

#define _XTAL_FREQ 8000000UL // 8 MHz oscillator frequency used for delay calculations

#define COMMON_ANODE

#ifndef COMMON_ANODE

    // For common anode displays:

    // The display's anode is tied to +V, and each segment lights up when its pin is driven LOW.

    // Inverts the pattern bits so that a 0 in the pattern turns the segment ON.

    #define SEG_PATTERN(x) ((~(x)) & 0x7F)

#else

    // For common cathode displays, the pattern is used directly.

    #define SEG_PATTERN(x) (x)

#endif

// Traffic Light Pin Definitions

// These macros define which microcontroller output bits drive the traffic light LEDs.

#define RED1    LATAbits.LATA0    // Light 1 Red connected to RA0
#define YELLOW1 LATAbits.LATA1    // Light 1 Yellow connected to RA1
#define GREEN1  LATAbits.LATA2    // Light 1 Green connected to RA2

#define RED2    LATAbits.LATA4    // Light 2 Red on RA4
#define YELLOW2 LATAbits.LATA5    // Light 2 Yellow on RA5
#define GREEN2  LATAbits.LATA6    // Light 2 Green on RA6

#define RED3    LATBbits.LATB0    // Light 3 Red on RB0
#define YELLOW3 LATBbits.LATB1    // Light 3 Yellow on RB1
#define GREEN3  LATBbits.LATB2    // Light 3 Green on RB2

// 7-Segment Display Lookup Table
const unsigned char digitPatterns[10] = {

    0x3F, // 0: segments a, b, c, d, e, f ON; g OFF.
    0x06, // 1: segments b, c ON.
    0x5B, // 2: segments a, b, d, e, g ON.
    0x4F, // 3: segments a, b, c, d, g ON.
    0x66, // 4: segments b, c, f, g ON.
    0x6D, // 5: segments a, c, d, f, g ON.
    0x7D, // 6: segments a, c, d, e, f, g ON.
    0x07, // 7: segments a, b, c ON.
    0x7F, // 8: all segments ON.
    0x6F  // 9: segments a, b, c, d, f, g ON.
};
```

```

void updateDisplay(unsigned char count) {
    // Separate count into tens and ones.
    unsigned char tens = count / 10;
    unsigned char ones = count % 10;

    // SEG_PATTERN applies inversion for common anode if needed.
    LATD = SEG_PATTERN(digitPatterns[tens]);

    // Output the ones digit pattern to PORTJ.
    // The pattern is shifted left by 1 so that bit0 goes to RJ1, bit1 to RJ2, etc.
    LATJ = (SEG_PATTERN(digitPatterns[ones]) << 1);
}

// Enumeration for identifying which traffic light is active.
typedef enum {
    LIGHT_1 = 1,
    LIGHT_2,
    LIGHT_3
} LightNumber;

// Enumeration for the state within a light's cycle.
typedef enum {
    STATE_YELLOW,    // Yellow phase lasting 1 second
    STATE_GREEN,     // Green phase lasting 8 seconds
    STATE_YELLOW2,   // Second yellow phase lasting 1 second
    STATE_ALL_RED    // All lights red phase lasting 10 seconds (after Light 3)
} SystemState;

// Duration constants in milliseconds.
#define DURATION_YELLOW    1000UL    // 1 second yellow
#define DURATION_GREEN     8000UL    // 8 seconds green
#define DURATION_YELLOW2   1000UL    // 1 second yellow
#define DURATION_ALL_RED   10000UL   // 10 seconds all red

// Global timing and state variables.
volatile unsigned long msCounter = 0; // Global millisecond counter, incremented by Timer0 ISR
volatile LightNumber currentLight = LIGHT_1; // Current active traffic light
volatile SystemState currentState = STATE_YELLOW; // Current state in the traffic light cycle
unsigned long phaseStartTime = 0;      // Timestamp (in ms) when the current phase started

```

```

// Variable for the display cycle timer; used to update the 7-segment displays every second.
unsigned long displayCycleStart = 0;

// Timer0 Interrupt Service Routine (ISR)
// This ISR is called every time Timer0 overflows
void __interrupt() ISR(void) {
    if (INTCONbits.TMR0IF) {
        // Reload Timer0 for approximately 1ms.
        // For an 8MHz clock with a 1:32 prescaler:
        // - Instruction cycle = Fosc/4 = 2MHz, so each tick = 0.5  $\mu$ s.
        // - With a prescaler of 32, each timer tick = 16  $\mu$ s.
        // - To get ~1ms, we need ~63 ticks (63 * 16 $\mu$ s = ~1ms).
        // - Preload value = 256 - 63 = 193.
        TMR0 = 193;
        INTCONbits.TMR0IF = 0; // Clear the Timer0 interrupt flag

        // Increment global millisecond counter.
        msCounter++;
    }
}

// setLight(), Sets the output levels for the red, yellow, and green LEDs for a specified traffic light.
static void setLight(LightNumber ln, unsigned char redVal, unsigned char yellowVal, unsigned char greenVal)
{
    switch (ln) {
        case LIGHT_1:
            RED1 = redVal;
            YELLOW1 = yellowVal;
            GREEN1 = greenVal;
            break;
        case LIGHT_2:
            RED2 = redVal;
            YELLOW2 = yellowVal;
            GREEN2 = greenVal;
            break;
        case LIGHT_3:
            RED3 = redVal;
            YELLOW3 = yellowVal;
            GREEN3 = greenVal;
            break;
    }
}

```

```

}

// setAllRed(), Turns all traffic lights to red.
void setAllRed(void) {
    RED1 = 1; YELLOW1 = 0; GREEN1 = 0;
    RED2 = 1; YELLOW2 = 0; GREEN2 = 0;
    RED3 = 1; YELLOW3 = 0; GREEN3 = 0;
}

// updateTrafficLights(), Updates the outputs of the traffic lights according to the active light and its
current state.
// Non-active lights are forced to red.
void updateTrafficLights(void) {
    // Force any light that is not currently active to display red.
    if (currentLight != LIGHT_1) setLight(LIGHT_1, 1, 0, 0);
    if (currentLight != LIGHT_2) setLight(LIGHT_2, 1, 0, 0);
    if (currentLight != LIGHT_3) setLight(LIGHT_3, 1, 0, 0);

    // For the active light, set the color based on the current state.
    switch (currentState) {
        case STATE_YELLOW:
        case STATE_YELLOW2:
            // Yellow is on (red off, yellow on, green off).
            setLight(currentLight, 0, 1, 0);
            break;
        case STATE_GREEN:
            // Green is on.
            setLight(currentLight, 0, 0, 1);
            break;
        case STATE_ALL_RED:
            // All lights are red.
            setAllRed();
            break;
    }
}

// Timer0 Initialisation Function: initTimer0()
//Sets up Timer0 for 8-bit mode with a 1:32 prescaler so that it overflows every ~1ms.

void initTimer0(void) {

```

```

/* Configure Timer0:
    - TMR0ON = 1: Turn Timer0 on.
    - T08BIT = 1: Use 8-bit mode.
    - T0CS = 0: Use the internal instruction cycle clock.
    - PSA = 0: Prescaler is assigned to Timer0.
    - T0PS (bits 2-0) = 100 for a 1:32 prescaler.

    These settings yield T0CON = 0xC4.
*/

T0CON = 0xC4;

TMR0 = 193;          // Preload timer to achieve ~1ms period.
INTCONbits.TMR0IF = 0;    // Clear the Timer0 overflow flag.
INTCONbits.TMR0IE = 1;    // Enable Timer0 interrupts.
}

// Main
void main(void) {
    // Configure the I/O ports

    TRISA = 0x00; // Set all PORTA pins (used for traffic lights) as outputs.
    TRISB = 0x00; // Set all PORTB pins (used for traffic lights) as outputs.
    LATA  = 0x00; // Initialize PORTA outputs to low.
    LATB  = 0x00; // Initialize PORTB outputs to low.
    TRISD = 0x00; // Set PORTD (first digit: tens) as outputs.
    LATD  = 0x00; // Initialize PORTD outputs.
    TRISJ = 0x00; // Set PORTJ (second digit: ones) as outputs.
    LATJ  = 0x00; // Initialize PORTJ outputs.

    // Initialise traffic lightsto start with all lights red.
    setAllRed();

    // Initialise the display to show "0" at the start.
    updateDisplay(0);

    // Initialise Timer0 and Enable Interrupts
    initTimer0();          // Set up Timer0 to generate a 1ms tick.
    INTCONbits.GIE = 1;    // Enable global interrupts.
    INTCONbits.PEIE = 1;   // Enable peripheral interrupts.

    // Initialise the Traffic Light State Machine and Display Timer
    currentLight = LIGHT_1;      // Start with Light 1.
    currentState = STATE_YELLOW; // Begin in the yellow phase.
    phaseStartTime = msCounter;  // Record the start time of the current phase.
    displayCycleStart = msCounter; // Start the 40-second cycle timer for the display.
}

```

```

updateTrafficLights();           // Set the initial state of the traffic lights.

// Main Loop
while (1) {

    // Calculates the time in the current phase.
    unsigned long elapsed = msCounter - phaseStartTime;

    // Traffic Light State
    if (currentState == STATE_YELLOW) {

        // If yellow phase has lasted 1 second, transition to green.
        if (elapsed >= DURATION_YELLOW) {
            currentState = STATE_GREEN;
            phaseStartTime = msCounter; // Reset phase timer.
            updateTrafficLights();
        }
    }

    else if (currentState == STATE_GREEN) {

        // If green phase has lasted 8 seconds, transition to second yellow.
        if (elapsed >= DURATION_GREEN) {
            currentState = STATE_YELLOW2;
            phaseStartTime = msCounter;
            updateTrafficLights();
        }
    }

    else if (currentState == STATE_YELLOW2) {

        // If second yellow has lasted 1 second, then complete this light's cycle.
        if (elapsed >= DURATION_YELLOW2) {
            // Turn current light red.
            setLight(currentLight, 1, 0, 0);
            if (currentLight < LIGHT_3) {
                // Move on to the next light if not on the last one.
                currentLight++;
                currentState = STATE_YELLOW;
                phaseStartTime = msCounter;
                updateTrafficLights();
            } else {
                // If this was Light 3, start the all-red phase.
                currentState = STATE_ALL_RED;
                phaseStartTime = msCounter;
                updateTrafficLights();
            }
        }
    }
}

```

```

    }

    else if (currentState == STATE_ALL_RED) {

        // All-red phase lasts 10 seconds.

        if (elapsed >= DURATION_ALL_RED) {

            // Resets the cycle after 40 seconds

            currentLight = LIGHT_1;

            currentState = STATE_YELLOW;

            phaseStartTime = msCounter;

            displayCycleStart = msCounter;

            updateTrafficLights();

        }

    }

    // Update the 7-Segment Display:

    // Calculates how many whole seconds have elapsed in the 40-second cycle.

    unsigned long elapsedDisplay = msCounter - displayCycleStart;

    unsigned char secondsDisplay = elapsedDisplay / 1000; // Convert ms to seconds.

    // holds the value to 40 seconds

    if (secondsDisplay > 40) {

        secondsDisplay = 40;

    }

    // Updates the display to show the elapsed seconds.

    updateDisplay(secondsDisplay);

}

}

```