TRAFFIC LIGHT JUNCTION PROJECT

Electronics 1

Jacob C
2420842@student.uwtsd.ac.uk

Contents

1.0 Introduction 2
2.0 Background Research 2
3.0 System Requirements and Specifications 3
3.1 Functional Requirements 3
3.2 Hardware Requirements 4
4.0 System Design 5
4.1 Overall System Architecture 5
4.2 Circuit Design 5
4.2.1 Microcontroller Unit 6
4.2.2 Traffic Light Control Subsystem 6
4.2.3 Countdown Display Subsystem 6
4.2.4 I/O Expansion using I2C 7
4.3 Timing and Control Logic 7
5.0 Algorithm / System Operation 7
5.1 Flowchart 8
5.2 Pseudocode 9
6.0 Simulation and Implementation 11
6.1 Simulation Operation 11
7.0 Prototyping and Testing 13
8.0 Conclusion 14
9.0 References 15
10.0 Appendices 16

1.0 Introduction

The following report details the steps and considerations taken in the design and development of a traffic light system, designed to mimic a 3-way intersection. These are built to ensure smooth and safe management of traffic, providing ample time for each direction of traffic to flow without difficulty, whilst also keeping all road users safe. Without these systems people run the risk of accidental collision with other road users, especially on busier streets where more traffic can be found.

This system will feature an automated system wherein the lights will change from one phase to the next based on an internal countdown timer, displayed on 7-segment displays, and will conclude with a final phase where all lights will be red and a separate pedestrian signal with go green. This was designed to work with a PIC16F877A microprocessor, using additional components and modules where necessary to ensure smooth and organized operation.

2.0 Background Research

Traffic light systems are an essential component of the modern transportation infrastructure, designed to regulate the movement of vehicles and improve road safety. At junctions, especially T-junctions, these systems help minimise congestion, reduce the likelihood of accidents, and ensure a consistent and orderly flow of traffic. This is done by assigning right of way, through the use of signals operating on set timings, these signals provide a structured method of managing the demands imposed by traffic [1].

There are several types of traffic control systems used in practice today, with the most common being fixed time-systems and adaptive systems. Fixed-time systems operate based on the predetermined timings, wherein each phase of traffic is allotted a set period, regardless of traffic conditions, these systems are relatively simple to design and implement, making them much more cost and time effective, making them ideal for areas with predictable traffic patterns. In contrast, adaptive systems use sensors and real time data to adjust the signal timings based on the current flow of traffic, improving efficiency but increasing cost and system complexity [2] [3].

Microcontrollers play a central role in modern traffic light systems, particularly in embedded system designs. A microcontroller acts as the central control unit of the system, executing programmed instructions to control signal timings, coordinating outputs, and controlling peripheral devices, such as displays. In this project a PIC series microprocessor is selected due to its suitability for real-time control applications, ease of programming using the C programming language and due to its availability and compatibility with simulation software such as proteus. Microcontroller-based systems offer a range of flexibility and precision, allowing for accurate timing sequences and the ability to integrate additional features, such as pedestrian crossing phases. [4] [5].

Traditional traffic light systems operate using a sequence of red, amber, and green signals, each conveying a specific instruction to road users. These signals follow standardised meanings to ensure consistency and safety. For example, a red signal would indicate vehicles to stop, whilst a green signal would indicate for traffic to go. The amber signal provides a transition warning between stop and go phases. These conventions are defined within legal frameworks such as the 'Traffic Signs Regulations and General Directions' and are supported by guidance in the UK Highway Code, ensuring all road users can interpret the signals consistently and correctly [6] [7].

In addition to controlling vehicle traffic, modern systems often incorporate pedestrian crossing phases to improve safety for non-motorised users. During this phase, all vehicle signals are set to red and pedestrian indicators change to allow for safe crossing. This feature is particularly important in urban environments and aligns with broader road safety regulations [6].

Despite their advantages, fixed-time traffic control systems have limitations. Because they do not adapt to real-time traffic conditions, they may result in inefficiencies, particularly during busier periods, potentially causing congestion in certain directions. However, for this project, a fixed-time system is ideal due to its simplicity, ease of implementation, and suitability for simulation-based design. It also allows for a clear demonstration of embedded systems and timing control techniques [2] [3].

Overall, designing a traffic light control system, requires an understanding of embedded systems, engineering principles and regulatory standards. By combining microcontroller-based design with standard traffic signal practices, it is possible to develop a system that effectively manages traffic flow while maintaining safety. This background forms the foundation for the design and implementation of the proposed T-junction traffic light controller.

3.0 System Requirements and Specifications

The design of the traffic light controller system is based on a set of clearly defined functional and technical requirements derived from the assignment brief, with modifications introduced to better reflect the operation of a real-world T-junction. These adjustments improve the realism of the system while maintaining compliance with safety principles and timing constraints.

3.1 Functional Requirements

The primary function of the system is to control traffic flow at a T-junction using a fixed-time sequence. Unlike a simple equal-phase system, this design reflects a more realistic scenario in which the main road carries traffic in two directions and therefore requires a longer signal duration compared to the intersecting road.

The system managed three sets of traffic signals, each consisting of red, amber, and green LEDs, operating according to standard traffic signal conventions [6] [7]. The complete signal cycle lasts 40 seconds and is divided into three main phases:

Phase 1 (20 Seconds): Main Road Traffic Flow
Traffic on the main road is allowed to move in both directions simultaneously, while the intersecting road is held at a red signal. The extended duration reflects the higher expected volume of traffic typically associated with main roads.

Phase 2 (10 Seconds): Intersecting Road Flow
Traffic from the side road is allowed to proceed, while the traffic from the main road is stopped, this shorter duration is appropriate for lower traffic demand,

Phase 3 (10 Seconds): Pedestrian Crossing Phase
All vehicle signals are set to red, and pedestrian crossing indicators are activated to allow safe crossing, ensuring compliance with road safety practices and prioritises pedestrian movement [6].

This phased approach ensure traffic flow is prioritised effectively while maintaining safety for all road users. The allocation of different time durations demonstrates an understanding of real-world traffic behaviour and improves the practical relevance of the design [2].

In addition, the system includes 7-Segment displays on each road, which provide a countdown from 40-0 seconds, providing an accurate representation of timings whilst also enhancing user awareness by highlighting time for each phase, helping all users to anticipate signal changes.

3.2 Hardware Requirements

The system is implemented using a PIC series microcontroller, which acts as a central control unit. The microcontroller is responsible for executing the programmed timing sequence, controlling signal outputs and updating the countdown displays. PIC microcontrollers are widely used in embedded systems due to their reliability, low cost, and ease of programming due to C programming environment.

The required hardware components include:

- Traffic signal LEDs (red, amber, green) for each road
- Pedestrian Signal LEDs
- 7-segment displays for countdown functionality
- Current-limiting resistors for all LEDs
- Microcontroller clock or internal timer module
- Power supply (5V DC)

All components are interfaced with the microcontroller through digital input/output pins, and the design must ensure correct signal operation, stable power delivery and accurate display control.

4.0 System Design

The traffic light control system is designed using a combination of a PIC microcontroller, display drivers, and external interface components to achieve accurate signal timing and efficient output control. The system integrates traffic signals, countdown displays and pedestrian indicators into a single coordinated architecture.

4.1 Overall System Architecture

The system is centred around the PIC16F877A microcontroller, which acts as the main control unit. It is responsible for executing the programmed timing sequence, controlling the traffic sequence and managing communication with the peripheral components.

The design can be separated into three main subsystems:

- Traffic light control subsystem
- Countdown display subsystem
- Pedestrian and I/O expansion system

The microcontroller coordinates all subsystems to ensure synchronization across the entire junction.

4.2 Circuit Design

Various choices were made within the circuit design process, the most important being the choice of microcontroller, whilst suggested in the brief the PIC16F877A was an ideal choice due to its internal modules enabling for the use of an interrupt driven system, whilst providing several I/O ports, and featuring built-in communication interfaces.

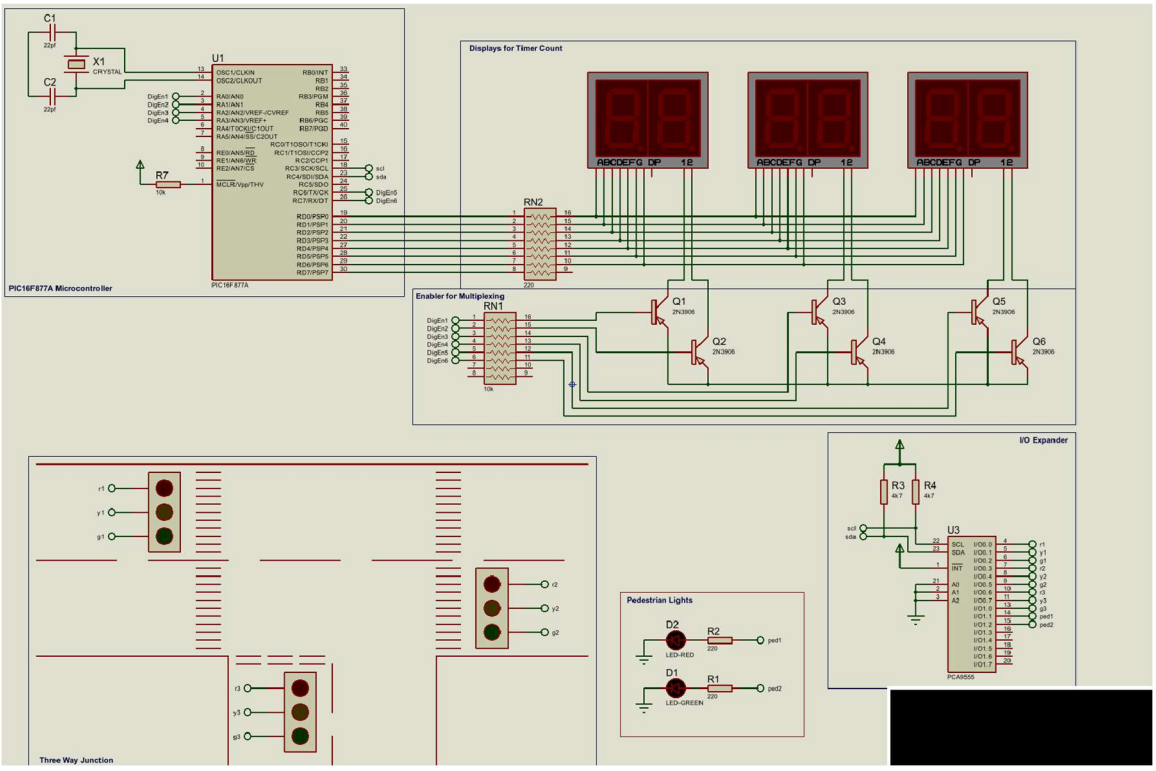

Figure 1: The overall design of the circuit

4.2.1 Microcontroller Unit

As mentioned above the PIC16F877A is an ideal choice for this implementation due to its suitability for embedded control applications, the number of I/O ports and a wide range of internal modules.

Key features utilised in this design include:

- The internal timer0 module enabling for an interrupt driven timer module for precise delay generation
- An I2C communication interface used for external I/O expansion

Shown in figure 1, an external crystal oscillator(X1) with two capacitors(C1, C2) are used to provide a stable 20MHz clock signal, ensuring accurate timing within the system. A pull-up resistor(R7) is connector to the reset pin (MCLR) to maintain stable operation of the system.

4.2.2 Traffic Light Control Subsystem

The traffic lights are implemented using sets of red, amber, and green LEDs representing each road of the T-junction. Each LED is controlled via the microcontroller and are individually addressed through the I/O expander, allowing the system to show the correct sequencing.

The system follows the defined timing structure:

- 20 Seconds: Main Road Green, Side Road Red, Pedestrians Red
- 10 Seconds: Side Road Green, Main Road Red, Pedestrians Red
- 10 Seconds: Main Road Red, Side Road Red, Pedestrians Green

This means that at any given time traffic can only flow in one direction (Two when the main road is active), preventing conflicts between traffic flow and ensuring safe operation for all users.

4.2.3 Countdown Display Subsystem

Three dual 7-segment displays show the countdown timer.

To reduce the number of required microcontroller pins, the displays are controlled using a multiplexing technique, where:

- Segments A-G are shared across all displays
- The individual digits are enabled one at a time using switching transistors (2N3906 Transistors)
- Rapid switching between the two digits creates the illusion of simultaneous display

This approach reduces hardware complexity while maintaining display functionality. The microcontroller continuously updates each digit in sequence to maintain a stable visual output.

4.2.4 I/O Expansion using I2C

An I/O expander (PCF8574) is used to extend the number of available output pins. This is necessary due to the large number of LEDs required for traffic and pedestrian signals.

The microcontroller communicates with the I/O expander using the I2C protocol, so the microcontroller uses the SDA and SCL pins to communicate and can individually address each output pin on the chip, gaining an additional 16 output pins. Allowing for greater scalability.

The I/O expander controls the traffic signal LEDs and the Pedestrian indicator LEDs.

4.3 Timing and Control Logic

The timing control is implemented in software using internal timer0 module within the microcontroller. Enabling for an interrupt driven internal timer, creating a precise timer for the execution of the traffic light sequencing.

The countdown on the 7-segment displays are synchronized with the phases of traffic, decrementing from 40-0 seconds across the full cycle.

Accurate timing is critical and the use of a crystal oscillator ensure stable and consistent operation.

5.0 Algorithm / System Operation

The system operates using a combination of:

- Interrupt-driven timing (Timer0 ISR)
- Main loop execution (once per second)
- Lookup table for traffic light states

The countdown timer begins at 40 seconds and decrements every second. At each step, the system updates:

- Traffic light outputs via the I^2C expander
- 7-segment display values via multiplexing

Once the countdown reaches zero, it resets to 40 seconds and repeats the cycle.

5.1 Flowchart

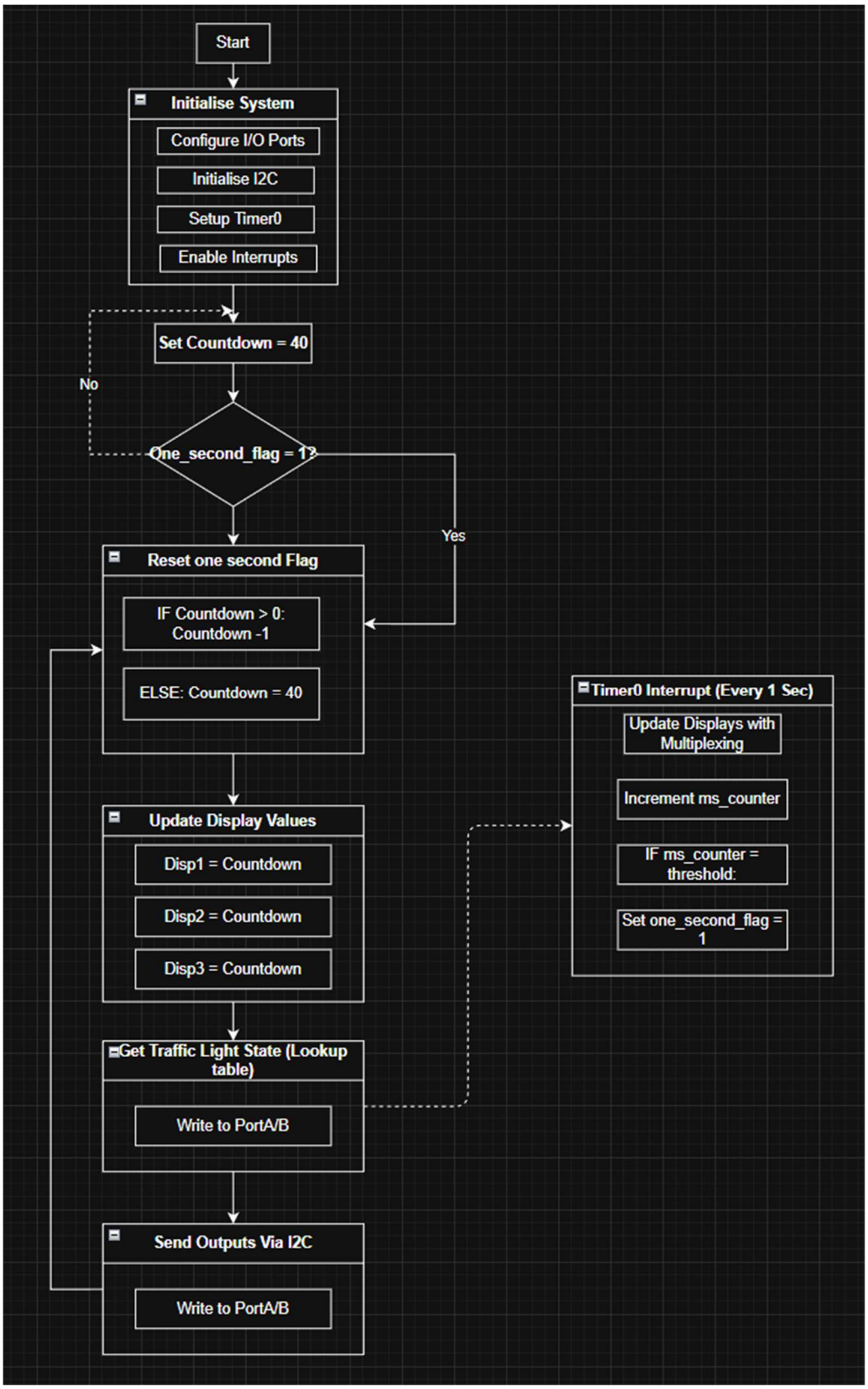

Figure 2: A flowchart depicting the operating logic of the system

5.2 Pseudocode

```
START
Initialise system:
        Configure all ports as digital
        Set display pins as output
        Setup Timer0
        Initialise I2C expander
Enable interrupts
Set countdown = 40
LOOP forever:
        IF one_second_flag == TRUE THEN
                Reset one_second_flag
        IF countdown > 0 THEN
                countdown = countdown - 1
        ELSE
                countdown = 40
        ENDIF
        Update displays:
                disp1 = countdown
                disp2 = countdown
                disp3 = countdown
        Determine traffic light state:
                (portA, portB) = lookup_table[countdown]
        Send outputs via I2C:
        Write portA and portB to expander
        ENDIF
END LOOP
```

Figure 3: Main Program Pseudocode

```
ON TIMER0 INTERRUPT:
Disable all display digits
Clear segment outputs
SELECT current digit based on mux_index:
        CASE 0: display units of disp1
        CASE 1: display tens of disp1
        CASE 2: display units of disp2
        CASE 3: display tens of disp2
        CASE 4: display units of disp3
        CASE 5: display tens of disp3
Enable selected digit
Increment mux_index (wrap 0–5)
Reload timer
Increment ms_counter
IF ms_counter reaches threshold THEN
        Reset ms_counter
        Set one_second_flag = TRUE
ENDIF
RETURN
```

Figure 4: Timer Interrupt Routine (ISR) Pseudocode

```
FUNCTION set_lights(time):
  IF time > 40 → time = 40
  IF time < 0 → time = 0
  RETURN state_table[time]
END FUNCTION
```

Figure 5: Traffic Light Control Logic (Lookup Table)

6.0 Simulation and Implementation

The traffic light control system was designed and tested using the proteus design suite, which provides a realistic platform for modelling embedded systems. The simulation included the full circuity design consisting of a PIC16F877A driven by an external crystal oscillator, traffic signal LEDs, pedestrian LEDs, multiplexed 7-segment displays and an I2C based I/O expander.

The microcontroller was programmed in C using a CCS compiler, where the compiled hex file was loaded into the simulation.

6.1 Simulation Operation

The simulation allowed for observation of the system under normal operating conditions. Upon starting the simulation, the system initialised correctly, however the countdown could not be verified to be correctly functioning. This was due to a limitation within the proteus visual engine, where the code was operating beyond a speed that could be rendered within proteus. The countdown however could be verified by using the in-built timer within proteus, present to show the duration of the simulation.

The system was observed to follow the intended three phase cycle where:

- Phase 1 (0 – 20 Seconds): Main Road Traffic Active
- Phase 2 (20 – 30 Seconds): Side Road Traffic Active
- Phase 3 (30 – 40 Seconds): Pedestrian Crossing Active

During each phase each of the corresponding traffic lights were activated while all conflicting routes held a red signal, ensuring safe operation.

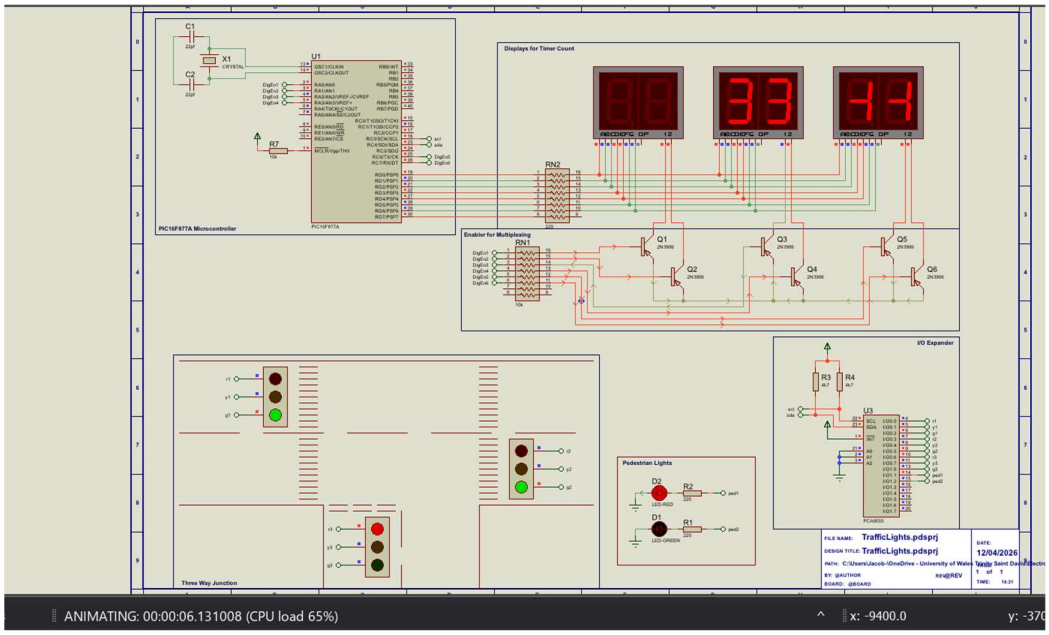

Figure 6: showing the active state of the system at 6 Seconds

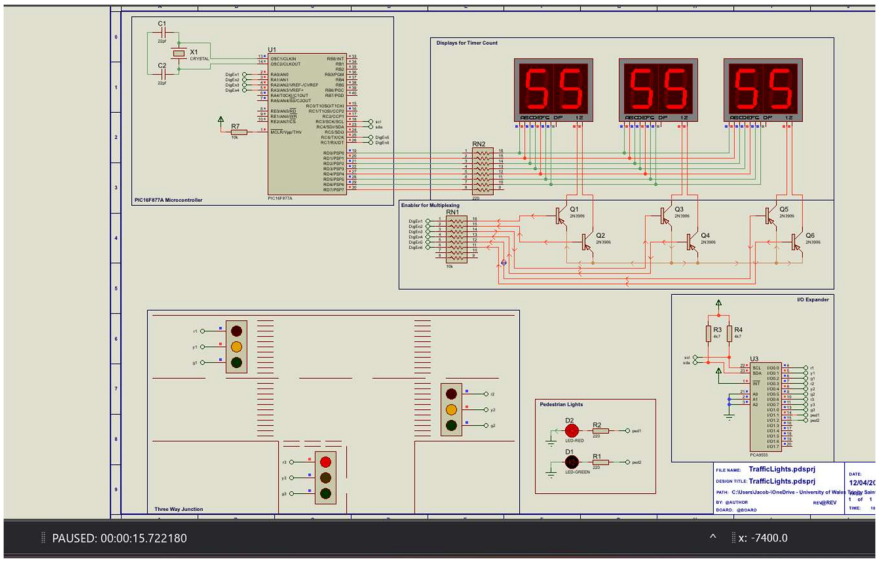

Figure 7: Showing the change in active state at 15 seconds

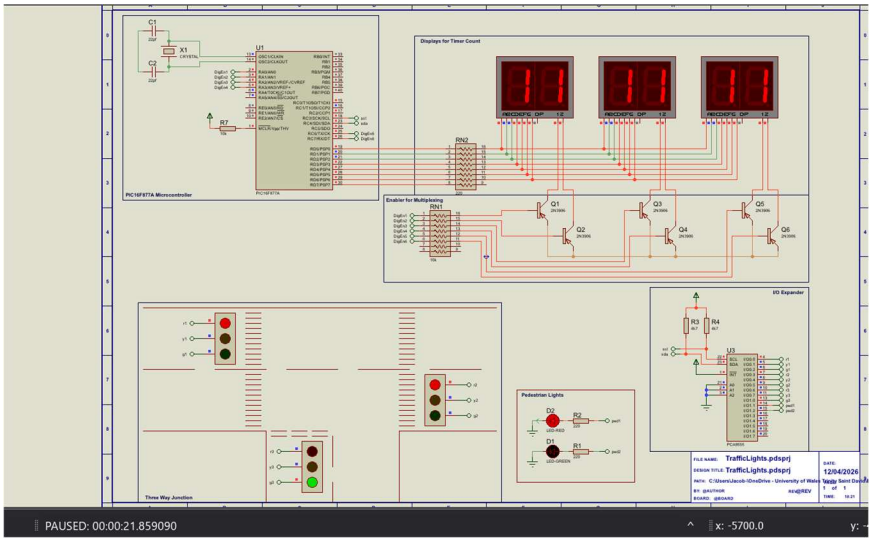

Figure 8: Showing the change of active state to allow for the traffic to pass through the interesting road at 21 seconds

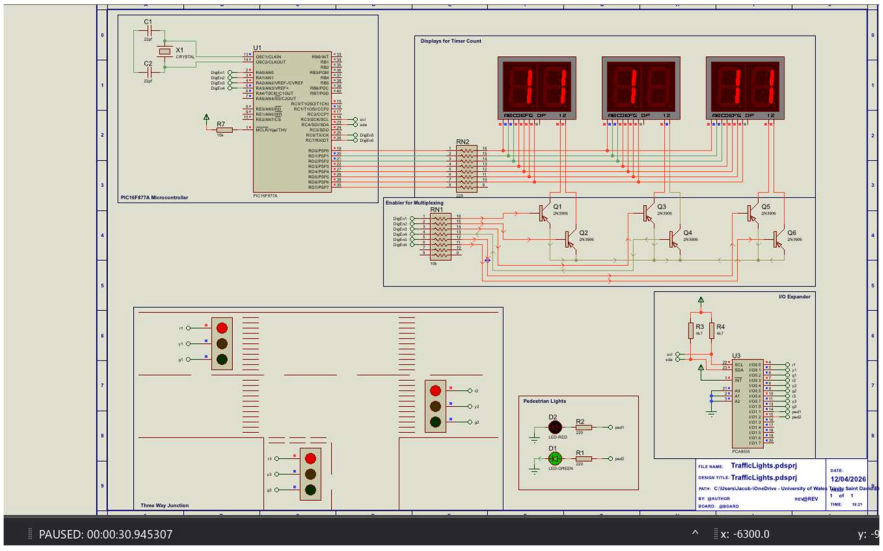

Figure 9: Showing the pedestrian phase within the 30-40 second window

Although limitations within the Proteus simulation environment affected the visual verification of the countdown display, the correct operation of the system was confirmed through analysis of the internal timing behaviour and supporting project files. The simulation demonstrated accurate phase sequencing in accordance with the programmed timing logic, with each traffic state transitioning at the expected intervals. The countdown functionality, while not clearly rendered due to simulation speed constraints, was validated through the timer configuration and interrupt-driven design, ensuring precise one-second decrements.

7.0 Prototyping and Testing

The proposed circuit was built on a breadboard, where all connections were built in alignment with the schematic outlined within the proteus simulation. The microprocessor was programmed using a PICKIT3 programming board using the MPLAB IPE.

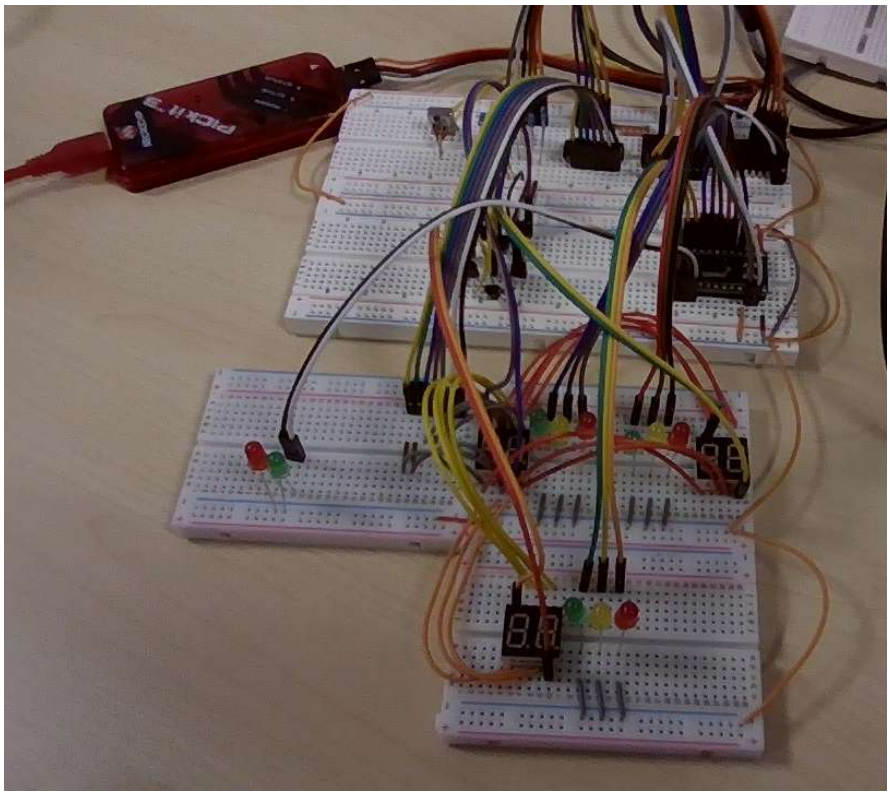

Figure 10: The physical assembly of the traffic light system, with 3 LEDs and a seven segment display for each signal

Unfortunately, the testing of the board was unsuccessful due to the PICKIT3 platform being unable to supply enough current to power the circuit. Due to the combination of many LEDs, transistors, and the I/O expander. This conclusion was drawn from an error code flashing on the PICKIT where it holds a green light on the status LED whilst steadily flashing red, indicating the PICKIT is unable to supply enough power to the target circuit.

8.0 Conclusion

The aim of this project was to design and implement a traffic light control system for a T-junction using a PIC16F877A microcontroller, with a focus on safety, timing accuracy, and practical system design. This objective was achieved through the development of a fully functional system that integrates traffic signal control, countdown displays and a dedicated pedestrian crossing phase.

The system operates on a fixed-timing structure, with a total cycle time of 40 seconds divided into 3-phases, 20 seconds for the main road, 10 seconds for the intersecting road and a further 10 seconds for the pedestrian phase. This modified timing structure reflects real world conditions, by prioritising the main road traffic and pedestrians. The inclusion of 7-segment displays further enhances the usability of the system providing insight on the timings to the various road users.

From a technical perspective, the system demonstrates an effective use of embedded system principles. The implementation of an interrupt-driven timing mechanism ensures accurate and consistent operation, while the multiplexing techniques provides an effective and efficient method of controlling multiple displays with minimal use of hardware resources. Additionally, the inclusion of an I2C I/O expander enables the system to use several digital outputs without exceeding the limits of the microcontroller.

The simulation results confirmed that the system performs as expected, with correct signal sequencing and stable operation across all phases. Although physical prototyping was attempted, limitations in delivery prevented full hardware validation. Despite this the simulation provides strong evidence that the design is fully functional, correct and suitable for real world use with appropriate power considerations.

In conclusion, the project successfully demonstrates the design and implementation of a practical traffic light system using a microcontroller-based approach. The system meets the functional requirements outlined in the assignment brief whilst also incorporating realistic design considerations, highlighting both technical competence and an understanding of real-world traffic conditions.

9.0 References

[1] L. Ye, “Traffic Light Control System,” *Journal of Theory and Practice of Engineering Science,* vol. 4, no. 3, pp. 111-124, 2024.

[2] C.-T. C. M. S. Firas Zahwa, “Novel Intelligent Traffic Light Controller Design,” *Machines 2024,* vol. 12, no. 7, p. 469, 2024.

[3] I. I. R. Nazar Elfadil Mohamed, “Traffic light control design approaches: a systematic literature review,” *International Journal of Electrical and Computer Engineering (IJECE),* vol. 12, no. 5, pp. 5355-5363, 2022.

[4] G. O. E. O. A. Aniru Abudu Muhammed, “Design and Implementation of a Microcontroller-Based Adaptive Four-Way Traffic Light Control System for Traffic Optimization,” *Journal of Power, Energy, and Control,* vol. 1, no. 2, pp. 92-102, 2024.

[5] M. R. Gordon, “Review on traffic light control systems based on single-chip microcomputers and smart traffic light control systems,” *Applied and Computational Engineering,* vol. 8, pp. 213-221, 2023.

[6] D. o. Transport, “The Traffic Signs Regulations and General Directions 2016,” UK Government, 2026. [Online]. Available: https://www.legislation.gov.uk/uksi/2016/362/contents/made. [Accessed 04 2026].

[7] D. o. Transport, “The Highway Code,” UK Government, 2023. [Online]. Available: https://www.gov.uk/guidance/the-highway-code. [Accessed 04 2026].

10.0 Appendices

Appendix A:

```
#include <16F877A.h>
#fuses HS,NOWDT,NOLVP,PUT
#use delay(crystal=20000000)

// Using hardware I2C on PORTC
#use i2c(master, sda=PIN_C4, scl=PIN_C3, slow)

// PCA9555 Address (A0, A1, A2 tied to GND)
#define PCA_ADDR 0x40

// PCA9555 register map (see datasheet)
#define PCA_OUTPUT0 0x02   // Output port 0 register
#define PCA_OUTPUT1 0x03   // Output port 1 register
#define PCA_CONFIG0 0x06   // Direction config port 0
#define PCA_CONFIG1 0x07   // Direction config port 1


// - TRAFFIC LIGHT BIT DEFINITIONS FOR PORTS A/B OF I/O EXPANDER -

// Port A
#define R1 0x01 // Traffic Light 1 - Red
#define Y1 0x02 // Traffic Light 1 - Yellow
#define G1 0x04 // Traffic Light 1 - Green
#define R2 0x08 // Traffic Light 2 - Red
#define Y2 0x10 // Traffic Light 2 - Yellow
#define G2 0x20 // Traffic Light 2 - Green
#define R3 0x40 // Traffic Light 3 - Red
#define Y3 0x80 // Traffic Light 3 - Yellow

// Port B
#define G3      0x01 // Traffic Light 3 - Green
#define PED_RED 0x02 // Pedestrian Light - Red
#define PED_GRN 0x04 // Pedestrian Light - Green


// - 7-SEGMENT DISPLAY CONTROL -

// Digit enable pins (3x 2 digit multiplexed display)
#define DigEn1 PIN_A0
#define DigEn2 PIN_A1
#define DigEn3 PIN_A2
#define DigEn4 PIN_A3
// DigEn5/6 moved to PORTC due to analogue behaviours on A4/A5
#define DigEn5 PIN_C6
#define DigEn6 PIN_C7

// 7-segment encoding for common anode seven segment display
const int8 SEG_MAP[10] =
{0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};


// - GLOBAL STATE VARIABLES -

// Countdown timer (seconds)
volatile int8 countdown = 40;

// Used for 1-second timing from Timer0 ISR
volatile int16 ms_counter = 0;
```

```
volatile int1 one_second_flag = 0;

// Display buffers (3 two-digit displays)
int8 disp1 = 40;
int8 disp2 = 40;
int8 disp3 = 40;

// Multiplex index (which digit is active)
int8 mux_index = 0;


// - I/O EXPANDER INITIALISATION -

void PCA_init(void) {
// Provides 50ms delay to allow system stabilisation post start-up
   delay_ms(50);

   // Set Port0 as OUTPUT (0 = output)
   i2c_start();
   i2c_write(PCA_ADDR);
   i2c_write(PCA_CONFIG0);
   i2c_write(0x00);
   i2c_stop();

   // Set Port1 as OUTPUT
   i2c_start();
   i2c_write(PCA_ADDR);
   i2c_write(PCA_CONFIG1);
   i2c_write(0x00);
   i2c_stop();
}

// Writes to both output ports in one action
void PCA_write_ports(int8 port0, int8 port1) {
   i2c_start();
   i2c_write(PCA_ADDR);
   i2c_write(PCA_OUTPUT0);
   i2c_write(port0); // Port A equivalent
   i2c_write(port1); // Port B equivalent
   i2c_stop();
}


// - TIMER0 INTERRUPT (DISPLAY + TIMING) -

#int_TIMER0
void TIMER0_isr(void) {

   // Disables all digits on start-up to prevent ghosting
   output_high(DigEn1);
   output_high(DigEn2);
   output_high(DigEn3);
   output_high(DigEn4);
   output_high(DigEn5);
   output_high(DigEn6);

   // Clear segments before writing new value
   output_d(0xFF);

   // Multiplexing: display one digit per interrupt
   switch(mux_index) {
      case 0: output_d(SEG_MAP[disp1 % 10]); output_low(DigEn1); break; // units
      case 1: output_d(SEG_MAP[disp1 / 10]); output_low(DigEn2); break; // tens
```

```
      case 2: output_d(SEG_MAP[disp2 % 10]); output_low(DigEn3); break;
      case 3: output_d(SEG_MAP[disp2 / 10]); output_low(DigEn4); break;
      case 4: output_d(SEG_MAP[disp3 % 10]); output_low(DigEn5); break;
      case 5: output_d(SEG_MAP[disp3 / 10]); output_low(DigEn6); break;
   }

   // Advance to next digit (wrap around 0-5)
   mux_index = (mux_index + 1) % 6;

   // Reload timer for consistent interrupt period
   set_timer0(100);

   // Software timing: ~1 second flag generation
   ms_counter++;
   if(ms_counter >= 250) { // depends on timer config
      ms_counter = 0;
      one_second_flag = 1; // signal main loop
   }
}


// - TRAFFIC LIGHT CONTROL -

// Each entry represents the full system state for a given countdown value
// This removes any conditional logic and replaces it with direct lookup

typedef struct {
   int8 portA; // Road lights
   int8 portB; // Road 3 + pedestrian lights
} state_t;

// Lookup table (index = countdown value)
// This defines the ENTIRE behaviour of the system
const state_t state_table[41] = {
/*0*/  {Y1 | Y2 | R3, PED_RED},
/*1*/  {Y1 | Y2 | R3, PED_RED},
/*2*/  {R1 | R2 | R3, PED_RED},
/*3*/  {R1 | R2 | R3, PED_RED},
/*4*/  {R1 | R2 | R3, PED_GRN},
/*5*/  {R1 | R2 | R3, PED_GRN},
/*6*/  {R1 | R2 | R3, PED_GRN},
/*7*/  {R1 | R2 | R3, PED_GRN},
/*8*/  {R1 | R2 | R3, PED_GRN},
/*9*/  {R1 | R2 | R3, PED_GRN},
/*10*/ {R1 | R2 | R3, PED_GRN},
/*11*/ {R1 | R2 | R3, PED_RED},
/*12*/ {R1 | R2 | Y3, PED_RED},
/*13*/ {R1 | R2 | Y3, PED_RED},
/*14*/ {R1 | R2,      PED_RED | G3}, //G3 = Port B, moved for legibility
/*15*/ {R1 | R2,      PED_RED | G3},
/*16*/ {R1 | R2,      PED_RED | G3},
/*17*/ {R1 | R2,      PED_RED | G3},
/*18*/ {R1 | R2,      PED_RED | G3},
/*19*/ {R1 | R2,      PED_RED | G3},
/*20*/ {R1 | R2 | Y3, PED_RED},
/*21*/ {R1 | R2 | R3, PED_RED},
/*22*/ {R1 | R2 | R3, PED_RED},
/*23*/ {Y1 | Y2 | R3, PED_RED},
/*24*/ {Y1 | Y2 | R3, PED_RED},
/*25*/ {Y1 | Y2 | R3, PED_RED},
/*26*/ {G1 | G2 | R3, PED_RED},
/*27*/ {G1 | G2 | R3, PED_RED},
/*28*/ {G1 | G2 | R3, PED_RED},
```

```
/*29*/ {G1 | G2 | R3, PED_RED},
/*30*/ {G1 | G2 | R3, PED_RED},
/*31*/ {G1 | G2 | R3, PED_RED},
/*32*/ {G1 | G2 | R3, PED_RED},
/*33*/ {G1 | G2 | R3, PED_RED},
/*34*/ {G1 | G2 | R3, PED_RED},
/*35*/ {G1 | G2 | R3, PED_RED},
/*36*/ {G1 | G2 | R3, PED_RED},
/*37*/ {G1 | G2 | R3, PED_RED},
/*38*/ {G1 | G2 | R3, PED_RED},
/*39*/ {G1 | G2 | R3, PED_RED},
/*40*/ {G1 | G2 | R3, PED_RED}
};

// Constant-time state selection
void set_lights(int8 t, int8 *portA, int8 *portB) {
   // Sets bounds to ensure the counter does not go ary
   if(t > 40) t = 40;
   if(t < 0)  t = 0;

   // Direct lookup tables for ports A and B of the I/O Expander
   *portA = state_table[t].portA;
   *portB = state_table[t].portB;
}


// - MAIN LOOP -

void main(void) {

   setup_adc_ports(NO_ANALOGS); // All PORTA pins as digital

   set_tris_d(0x00); // PORTD = output (7-seg segments)
   set_tris_c(0b00011000); // RC6/7 outputs for digits
   set_tris_a(0x00); // PORTA = output (digit enables)

   // Timer0 setup (used for display refresh + timing)
   setup_timer_0(RTCC_INTERNAL | RTCC_DIV_128);
   set_timer0(100);

   PCA_init();   // Initialise I/O expander

   enable_interrupts(INT_TIMER0);
   enable_interrupts(GLOBAL);

   while(TRUE) {

      // Execute once per second
      if(one_second_flag) {

         one_second_flag = 0;

         // Countdown logic (wraps back to 40)
         if(countdown > 0)
            countdown--;
         else
            countdown = 40;

         // Update all displays with the current countdown value
         disp1 = countdown;
         disp2 = countdown;
         disp3 = countdown;
```

```
        int8 portA, portB;

        // Determine traffic light state (constant time)
        set_lights(countdown, &portA, &portB);

        // Output to hardware via I2C expander
        PCA_write_ports(portA, portB);
    }
  }
}
```