

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

Edward Bulpitt
UWTSD Swansea
2317250@student.uwtsd.ac.uk

Abstract— Traffic signal controllers regulate the flow of vehicles and pedestrians at intersections, balancing safety and throughput. This project designs a fixed-time controller for a T-junction using a PIC16F877A microcontroller. The system enforces a 40 s cycle divided into four 10 s blocks, with each block granting one approach a 7 s green interval followed by a 3 s yellow phase. A pedestrian pushbutton triggers an interrupt-based routine that safely inserts a 10 s crossing phase without desynchronizing the vehicle cycle. Timing accuracy is ensured through timer interrupts and a 50 ms software debounce routine. Two multiplexed seven segment displays present real time countdown and count up of the current phase. The design was fully simulated in Proteus 8, verifying LED states and reliable push button response under rapid toggling. Key contributions include a modular firmware setup separating vehicle and pedestrian logic, and some demonstration of open drain considerations on RA4 findings.

I. INTRODUCTION

Traffic signal control facilities are an integral part of the modern transportation system, designed to control vehicle flows, reduce the chance of crashes, and maintain delays at intersections low. This project seeks to create a fixed-time, deterministic controller that is simple to implement on low-cost microcontroller, balance between guaranteed performance and simplicity. Using the PIC16F877A microcontroller, chosen for its multi-functional I/O, easy to understand instruction set, and interrupt features, the system ensures precise timing and timely processing of pedestrian calls. A 40-second repeating cycle arranges the vehicle phases into four-time blocks, a block for each approach with a normal red, amber, green sequence. Pedestrian calls are signalled by an active-low pushbutton which provides an interrupt. Allowing the system to temporarily break the cycle, make space for pedestrians, and reestablish again without stopping synchronisation. Modular software, and simulation checks to check for operation prior to hardware prototyping are the main concern for this design. The result is a straightforward but effective traffic light controller.

II. BACKGROUND RESEARCH

A. Traffic-Signal Phasing

Fixed-time controllers divide each approach into set intervals for green, amber, and red lights. In the UK, regulations mandate a minimum 3 s amber phase (± 0.25 s) and a 2 s red+amber all-stop interval (± 0.25 s) before green. [1] Some systems also include an all-red phase for simultaneous pedestrian crossings. While simple fixed time schemes (e.g., a 40 s cycle) are easy to implement, adaptive controllers use sensors (IR, video) to adjust green times dynamically to varying traffic demands/loads.[2]

B. PIC16F877A I/O Characteristics

The PIC16F877A was selected for its versatile GPIO, timer peripherals, and interrupt capabilities. It exposes five 8-bit ports (A–E), each bit individually configurable via TRIS registers, sufficient to drive three traffic lights on the three approaches plus seven-segment displays and a pedestrian button. Notably, RA4 (T0CKI) functions as an open-drain output: it can sink current but requires an external pull-up (e.g., 10 k Ω) to source a logic high. During the creation I overlooked this detail, causing some issues with a segment in the seven-segment display to not illuminate.

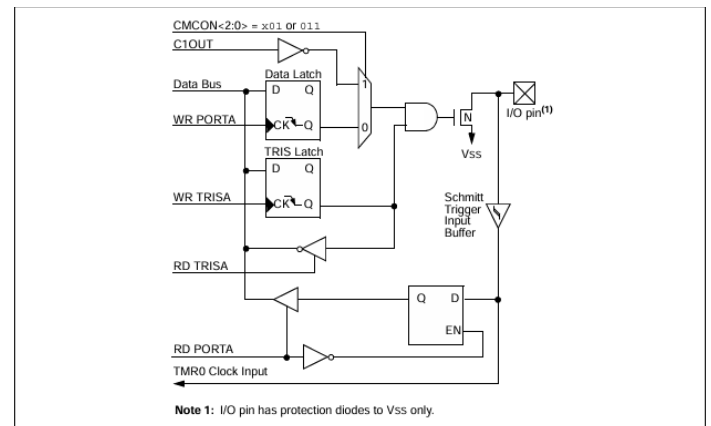


Figure 1. RA4 (T0CKI)

C. Pushbutton Debouncing

Mechanical pushbuttons inherently bounce for several milliseconds upon actuation, risking false or multiple triggers if read directly. While hardware RC filters or Schmitt triggers provide analog solutions, a software debounce is more cost-effective. In this design, the firmware checks for a low (active) button state, waits 50 ms, and re-samples. Only a sustained low confirms a valid press. This routine reliably rejects random transitions even under rapid tapping or prolonged holds, without adding components.

D. Seven-Segment Display Operation

A seven-segment LED display is made up of seven individual segments (plus an optional decimal point) arranged to render digits 0–9. While a firmware lookup table can map numeric values to segment patterns, a firmware lookup table can map numeric values to segment patterns.[3]

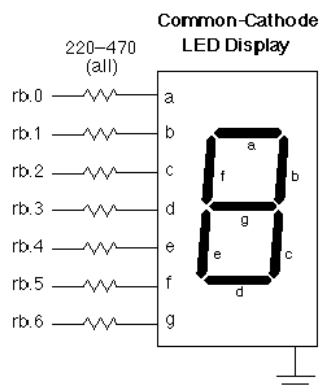


Figure 2. common cathode seven segment display diagram

For a cathode seven segment display each segment is controlled by setting its corresponding I/O pin high.

Current-limiting resistors (220Ω) prevent damage to the LEDs.

In this setup, segment pins connect to PORTC and PORTD of the microcontroller shown as numbered “ti” and ”t” outputs.

Binary-coded values are mapped to segment combinations using lookup tables.

III. CIRCUIT DESIGN & RELEVANT DIAGRAMS

The overall circuit comprises three 3 lamp traffic modules, a pedestrian signal module, 6 seven-segment timer displays, the pushbutton input, and power conditioning.

Layout:

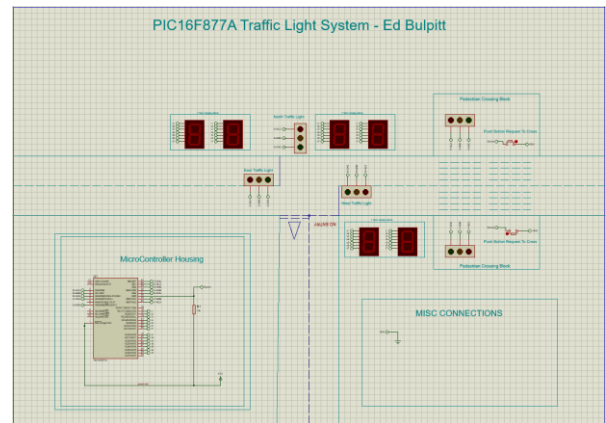


Figure 3. proteus schematic sheet

Microcontroller Core: The PIC16F877A microcontroller handles all timing, I/O control, and interrupt servicing. It interfaces directly with LEDs via current-limiting resistors(not shown here), MOSFET drivers for higher voltage applications, and the seven-segment displays.

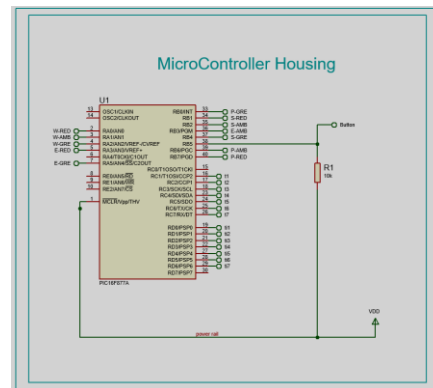


Figure 4. proteus schematic & connections of PIC microcontroller

Traffic Light Blocks: three traffic-light modules (North, East, West) each contain red, yellow, and green LEDs. These modules connect to dedicated PORT pins via current-limiting resistors. (shown in proteus file)

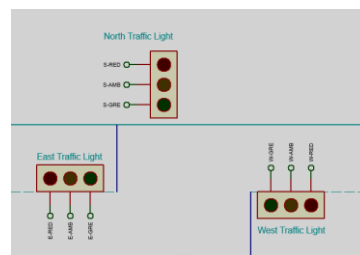


Figure 5. schematic placement of 3 traffic light modules

IV. PSEUDOCODE (BEST SEEN IN PROTEUS)

Seven-Segment Timers: Two-digit displays per approach show remaining phase time. Segments each on individual port lines are driven separately to multiplex each display pair.

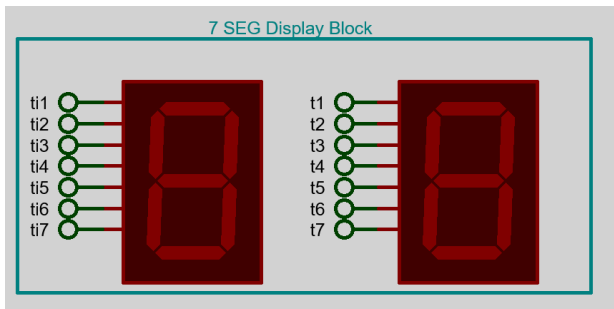


Figure 6. seven segment display block diagram

Pedestrian Crossing Units: Active-low pushbuttons trigger interrupts on the microcontroller. Debounce is handled in firmware; when pressed, the controller inserts a pedestrian phase into the cycle.

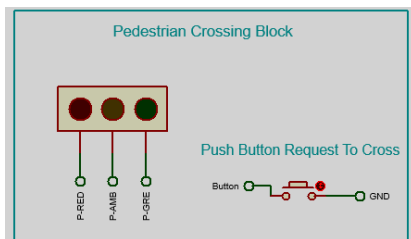


Figure 7. pedestrian block diagram

Power and Ground: The microcontroller and its logic run on 5 V. Traffic LEDs in this simulation are also powered from this source however I have created a separate circuit for a further developed real life build with a 10V supply for larger Traffic light LED's.(shown in misc box) Each LED is switched to ground by a logic-level N-channel MOSFET (2N7000): the drain ties to the LED return, the source to system ground. The MOSFET gate is driven by the PIC output (e.g., S-RED in this example as a demo of its functionality) and held low by a 10 k Ω pull-down resistor (R2). A flyback diode (D2) allows any excess current to drain through the led, while a 1 nF capacitor (C1) suppresses switching overvoltage (transients). All grounds are common, Pull-up resistors on RB4 for the button to function.

```
// === Initialization ===
init_io();
// Configure I/O pins for lights, display, button
// Set all vehicle lights to RED; pedestrian light to RED

// === Main Program ===
main():
while true:
    timer = 0
    ped_called = false

    // Run vehicle cycle; check if pedestrian button was pressed
    ped_called = vehicle_cycle()

    // If button was pressed, handle pedestrian crossing
    if ped_called:
        service_pedestrian()

// === Vehicle Cycle ===
vehicle_cycle():
    total_time = VEH_BLOCKS * BLOCK_DURATION

    while timer < total_time:
        // Determine current phase based on block
        block = timer / BLOCK_DURATION
        phase_time = timer % BLOCK_DURATION
        if phase_time < GREEN_TIME:
            state = GO
        else:
            state = YELLOW

        if block % 2 == 0:
            set main roads to state
            set side roads to STOP
        else:
            set side roads to state
            set main roads to STOP

        // Keep pedestrian light RED
        set pedestrian RED

        // Update 7-segment display
        show_timer(timer)

        // Check if pedestrian button was pressed
        if button_pressed():
            return true

        wait 1 second
        timer = timer + 1

    return false

// === Pedestrian Service ===
service_pedestrian():
    // Phase 1: Vehicles → YELLOW
    set all vehicles YELLOW
    set pedestrian RED
    countdown(YELLOW_TIME)

    // Phase 2: Vehicles → RED (safety buffer)
    set all vehicles RED
    set pedestrian RED
    countdown(SAFETY_RED_TIME)

    // Phase 3: Pedestrian → GREEN
    set all vehicles RED
    set pedestrian GREEN
    countdown(PED_TIME)

    // Phase 4: End crossing
    set pedestrian RED

// === Helpers ===
button_pressed():
    if BUTTON is LOW:
        wait 50 ms
        if BUTTON is still LOW:
            return true
        return false

show_timer(t):
    if DISPLAY_COUNTDOWN:
        display (total_time - t)
    else:
        display t

countdown(n):
    while n > 0:
        show_timer(n)
        wait 1 second
        n = n - 1
```

V. SOFTWARE LISTING

```
//////////////////////////////////////////////////////////////////
// Project: T-Junction Traffic Light
// Ed Bulpitt - 2317250
//////////////////////////////////////////////////////////////////

#include <16F877A.h>

// === CONFIG ===
#fuses XT, NOWDT, NOPROTECT, NOLVP
#use delay(clock=4000000)

// === PIN ASSIGNMENTS ===
// Main West
#define MW_RED    PIN_A0
#define MW_YEL    PIN_A1
#define MW_GRN    PIN_A2
// Main East
#define ME_RED    PIN_A3
#define ME_YEL    PIN_B3
#define ME_GRN    PIN_A5
// Narrow South
#define NS_RED    PIN_B1
#define NS_YEL    PIN_B2
#define NS_GRN    PIN_B4
// Pedestrian
#define PD_RED    PIN_B7
#define PD_YEL    PIN_B6
#define PD_GRN    PIN_B0
#define BUTTON    PIN_B5 // active-low call button

// === 7-SEGMENT MAP ===
const unsigned char SEG_MAP[10] = {
    0b01111111,0b0000110,0b1011011,0b10011111,
    0b1100110,0b1101101,0b1111101,0b0000111,
    0b1111111,0b1101111
};
const int8 TENS_PINS[7] = {PIN_D0,PIN_D1,PIN_D2,PIN_D3,PIN_D4,PIN_D5,PIN_D6};
const int8 UNIT_PINS[7]= {PIN_C1,PIN_C2,PIN_C3,PIN_C4,PIN_C5,PIN_C6,PIN_C7};

// === TIMING ===
#define VEH_BLOCKS    4 // 4×10 seconds for 40 second vehicle cycle
#define BLOCK_DURATION 10
#define GREEN_TIME    7 //leaves 3 seconds for yellow in the 10s blocks
#define YELLOW_TIME    3
#define SAFETY_RED_TIME 3
#define PED_TIME      10

// pick exactly one display mode:
#define DISPLAY_COUNTUP

// light states
#define LS_STOP  0
#define LS_GO    1
#define LS_SLOW  2

// #####
//
void init_io(void);
void set_light(int r,int y,int g,int st);
void delay_s(unsigned int s);
void show_2digit(unsigned char v);
void service_ped(void);

// #####
// === MAIN ===
void main() {
    int timer;
    int block;
    int sec_in_blk;
    int state;
    int ped_called;

    init_io();

    while(TRUE) {
        timer = 0;
        ped_called = 0;

        // VEHICLE CYCLE with possible interrupt
        while(timer < VEH_BLOCKS * BLOCK_DURATION) {
            block = timer / BLOCK_DURATION;
            sec_in_blk = timer % BLOCK_DURATION;
            state = (sec_in_blk < GREEN_TIME)? LS_GO : LS_SLOW;

            // activate road
            if((block & 1) == 0) {
                // Main roads
                set_light(MW_RED,MW_YEL,MW_GRN,state);
                set_light(ME_RED,ME_YEL,ME_GRN,state);
                // Side STOP
                set_light(NS_RED,NS_YEL,NS_GRN,LS_STOP);
            }
        }
    }
}
```

```
} else {
    // Side roads
    set_light(MW_RED,MW_YEL,MW_GRN,LS_STOP);
    set_light(ME_RED,ME_YEL,ME_GRN,LS_STOP);
    set_light(NS_RED,NS_YEL,NS_GRN,state);
}

// ped always red here
output_low(PD_GRN); output_low(PD_YEL); output_high(PD_RED);

// show global timer
#ifdef DISPLAY_COUNTDOWN
    show_2digit(VEH_BLOCKS*BLOCK_DURATION - timer);
#else
    show_2digit(timer);
#endif

// check call-button (debounced)
if(!input(BUTTON)) {
    delay_ms(50);
    if(!input(BUTTON)) {
        ped_called = 1;
        break;
    }
}

delay_s(1);
timer++;
}

if(ped_called) {
    service_ped();
    // resume exactly at "timer" second
}
}

// #####
// === I/O INIT ===
void init_io(void) {
    set_tris_a(0xC1);
    set_tris_b(0x20); // RB5 input (button)
    set_tris_c(0x01);
    set_tris_d(0x80);
    set_tris_e(0x00);

    // start all STOP + ped RED
    set_light(MW_RED,MW_YEL,MW_GRN,LS_STOP);
    set_light(ME_RED,ME_YEL,ME_GRN,LS_STOP);
    set_light(NS_RED,NS_YEL,NS_GRN,LS_STOP);
    output_high(PD_RED); output_low(PD_YEL); output_low(PD_GRN);
}

// #####
// === SET LIGHT ===
void set_light(int r,int y,int g,int st) {
    switch(st) {
        case LS_STOP: output_high(r); output_low(y); output_low(g); break;
        case LS_GO:  output_low(r); output_low(y); output_high(g);break;
        case LS_SLOW: output_low(r); output_high(y); output_low(g); break;
    }
}

// #####
// === PEDESTRIAN SERVICE SEQUENCE ===
void service_ped(void) {
    int t;

    // 1) All vehicles ? ORANGE (3 s)
    set_light(MW_RED,MW_YEL,MW_GRN,LS_SLOW);
    set_light(ME_RED,ME_YEL,ME_GRN,LS_SLOW);
    set_light(NS_RED,NS_YEL,NS_GRN,LS_SLOW);
    output_low(PD_GRN); output_low(PD_YEL); output_high(PD_RED);
    for(t = YELLOW_TIME; t > 0; t--) {
        show_2digit(t);
        delay_s(1);
    }

    // 2) All vehicles ? RED (3 s safety buffer)
    set_light(MW_RED,MW_YEL,MW_GRN,LS_STOP);
    set_light(ME_RED,ME_YEL,ME_GRN,LS_STOP);
    set_light(NS_RED,NS_YEL,NS_GRN,LS_STOP);
    output_low(PD_GRN); output_low(PD_YEL); output_high(PD_RED);
    for(t = SAFETY_RED_TIME; t > 0; t--) {
        show_2digit(t);
        delay_s(1);
    }

    // 3) Pedestrian ? GREEN & Countdown (10 s)
    set_light(MW_RED,MW_YEL,MW_GRN,LS_STOP);
    set_light(ME_RED,ME_YEL,ME_GRN,LS_STOP);
    set_light(NS_RED,NS_YEL,NS_GRN,LS_STOP);
    output_low(PD_RED); output_low(PD_YEL); output_high(PD_GRN);
}
```

```

for(t = PED_TIME; t > 0; t--) {
    show_2digit(t);
    delay_s(1);
}

// 4) End pedestrian back to RED
output_low(PD_GRN);
output_high(PD_RED);
}

#####
// == 2-DIGIT 7-SEG DISPLAY ==
void show_2digit(unsigned char v) {
    unsigned char hi = v/10, lo = v%10;
    unsigned char mhi = SEG_MAP[hi], mlo = SEG_MAP[lo];
    int i;
    for(i = 0; i < 7; i++) {
        output_bit(TENS_PINS[i], (mhi >> i) & 1);
        output_bit(UNIT_PINS[i], (mlo >> i) & 1);
    }
}

#####
// == SIMPLE SEC DELAY ==
void delay_s(unsigned int s) {
    while(s--) delay_ms(1000);
}

```

Every 10 seconds the lights change allowing vehicles on “narrow North” to move

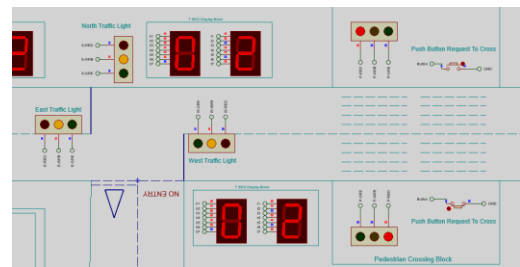


Figure 11.schematic image

When the pushbutton (pedestrian call) is pressed all vehicle lights will show amber for the standard 3 seconds

VI. CIRCUIT SIMULATION

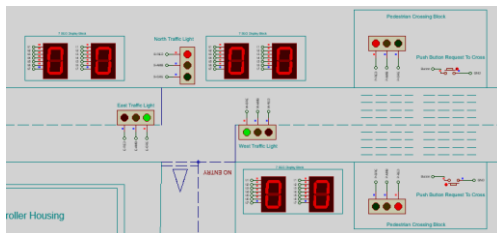


Figure 8.schematic image

The traffic for vehicles is allowed to flow EAST and WEST

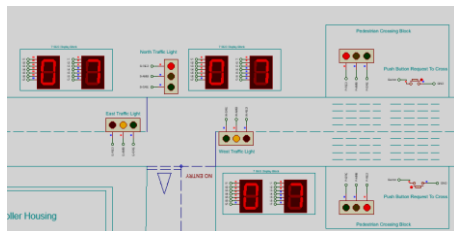


Figure 9.schematic image

At every 7 seconds the lights switch yellow allowing time for drivers to slow down.

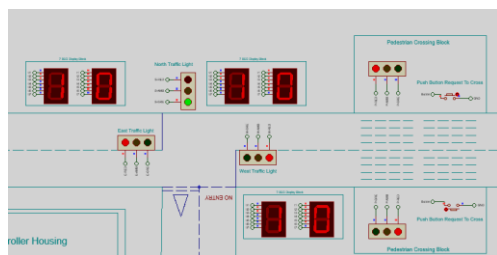


Figure 10.schematic image

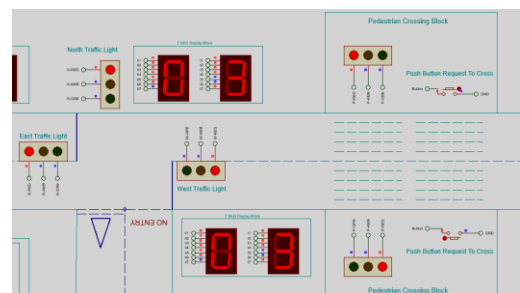


Figure 12.schematic image

A safety period for all lights pedestrian and vehicle is shown for 3 seconds

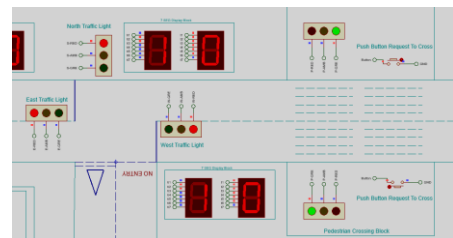


Figure 13.schematic image

Once this safety period is over the pedestrian lights will turn green allowing them to cross safely, there is a 10 second timer displayed which unlike the other timer, counts down.

VII. TESTING, SIMULATED OUTPUTS, AND RESULTS

Simulation Setup tests were performed in Proteus 8 Professional, to monitor GPIO lines probes were used.

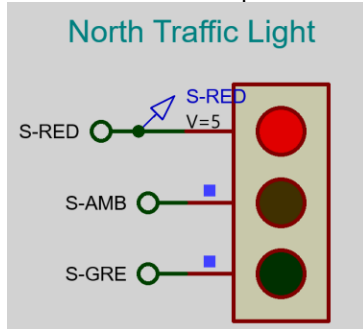


Figure 14.schematic image

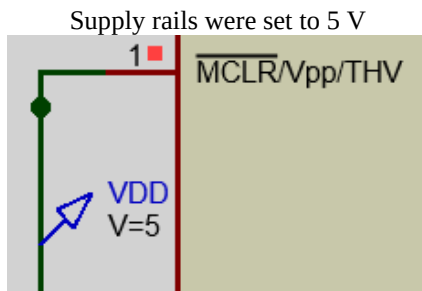


Figure 15.schematic image

the PIC16F877A firmware was loaded directly into the Proteus simulation.

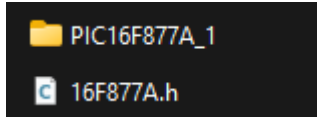


Figure 16. runtime folder location showing header file of PIC device

set_light(r,y,g,state)

- Tested each of the three states (STOP, GO, SLOW) in isolation.

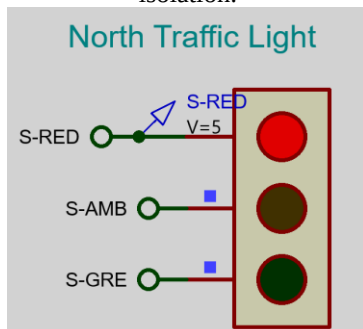


Figure 17.schematic image

Digit testing:

Tested values 00–99 in steps with different firmware
Confirmed correct segment patterns on both digits with a visual inspection of the seven-segment modules.

```
// loop through all digits to check timer is working correctly

#include <16F877A.h>

// LOOP 00 – 99
//fuses XT, NOWDT, NOPROTECT, NOLVP
//use delay(clock=4000000)
// LOOP 00 – 99

const unsigned char SEG_MAP[10] = {
    0b0111111,0b0000110,0b1011011,0b1001111,
    0b1100110,0b1101101,0b1111101,0b0000111,
    0b1111111,0b1101111
};

const int8 TENS_PINS[7] = {PIN_D0,PIN_D1,PIN_D2,PIN_D3,PIN_D4,PIN_D5,PIN_D6};
const int8 UNIT_PINS[7] = {PIN_C1,PIN_C2,PIN_C3,PIN_C4,PIN_C5,PIN_C6,PIN_C7};

void init_io(void);
void show_2digit(unsigned char v);

void main() {
    int i;
    init_io();

    while(TRUE) {
        for(i = 0; i < 100; i++) {
            show_2digit(i);
            delay_ms(50);
        }
    }

    void init_io(void) {
        set_tris_e(0x00); // All output
    }

    void show_2digit(unsigned char v) {
        unsigned char hi = v / 10;
        unsigned char lo = v % 10;
        unsigned char mhi = SEG_MAP[hi];
        unsigned char mlo = SEG_MAP[lo];
        int i;
        for(i = 0; i < 7; i++) {
            output_bit(TENS_PINS[i], (mhi >> i) & 1);
            output_bit(UNIT_PINS[i], (mlo >> i) & 1);
        }
    }
}
```

Active-Low Pushbutton and Debouncing

The pedestrian call button (PIN_B5) was wired as active-low with a 10 kΩ pull-up resistor to 5 V.

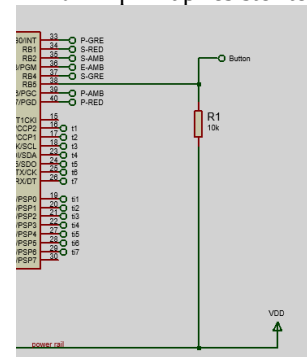


Figure 18.schematic image

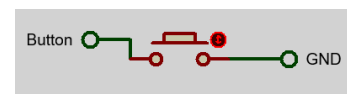


Figure 19.schematic image

Oscilloscope capture of button press:

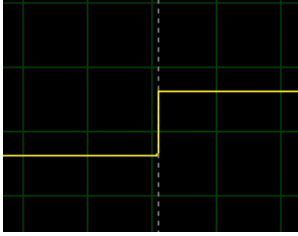


Figure 20. oscilloscope image showing no debouncing,

Verified that only one pedestrian called flag was set per valid press, even when the button was held down or tapped rapidly.

t = 0 s: Main roads green, north road red, display shows “00”.
t = 7 s: Main roads yellow, north road red, display shows “07”.
t = 10 s: Swap to north road green,

Pedestrian-Service Sequence

Upon valid button press, the system immediately:
Forces all roads to yellow for 3 seconds.
Switches all roads to red for a 3 second safety buffer.
LED states transition without overlaps or dead time.

High-Voltage LED Driver Circuit

MOSFET (2N7000) with pull-down (R2 - 10 k Ω), flyback diode (D2) and transient-suppressor capacitor (C1 - 1 nF)
Tested the power voltage at LED to what was expected

VIII. CONCLUSION, AND FUTURE RECOMMENDATIONS

A. D. Conclusion

This project delivered a foundation to a fixed-time traffic light controller for a T-junction, implemented on a PIC16F877A microcontroller and verified through Proteus 8 Pro simulation.

Key achievements:

timing: A 40 s cycle divided into four 10 s blocks, each providing a 7 s green and 3 s amber phase

Modular code: Separation of vehicle phase logic and pedestrian phase logic in a state machine design, good for future enhancements.

Pedestrian integration: An interrupt-driven, debounced pushbutton safely inserts a 10 s crossing phase without desynchronising the main cycle.

Two multiplexed seven-segment displays show real-time countdown/count up of the current phase.

Overall, the controller meets the core objectives of simplicity, predictability, making it well suited for small intersections.

B. D. Future Work

To develop this into an improved, adaptive and expandable system, the following are recommended:

C. Actuated Control:

an inductive loop or infrared sensor strategies.

semi and full actuated logic: extend green time where there is a detection of queuing. “Ignore” phases in case of off demand times. Etc.

D. Adaptive Algorithms:

real-time optimization methods such as MOVA (Microprocessor Optimised Vehicle Actuation) [4] dynamically learn optimal phase splits for various traffic scenarios.

E. Sensor and Classification:

video or lidar detection with vehicle classification (cars, buses, bicycles) and green time tuning by vehicle class. For example, lower times for city busses based on time schedule or longer crossing times for example people with disabilities could use an NFC tap card instead of a button press to increase walk time. Other sensors to improve detection quality under poor weather or night conditions.

F. Hardware and Network Enhancements:

Upgrade to a larger microcontroller for more advanced controls processing, larger I/O counts, and onboard networking. For example, Zigbee for remote monitoring, over-the-air firmware updates, connectivity to multiple traffic management systems. GPS unique identifiers modules for each unit, allowing a network to show locations and current states of traffic lights.

G. More Comprehensive Simulation and Field Testing:

Extend testing with lifelike simulated traffic, collecting data and adjusting based on it.

A hardware prototype for real street testing, recording real traffic cycles, failure modes, and maintenance requirements. Obtain measurements (cycle times, queue lengths, pedestrian waiting times) to refine timing parameters and safety margins.

H. Accessibility and Safety Features:

Mount audible warnings and tactile vibration units for blind pedestrians.

Set pedestrian-green times based on crossing width.

I. Data Logging and Analysis:

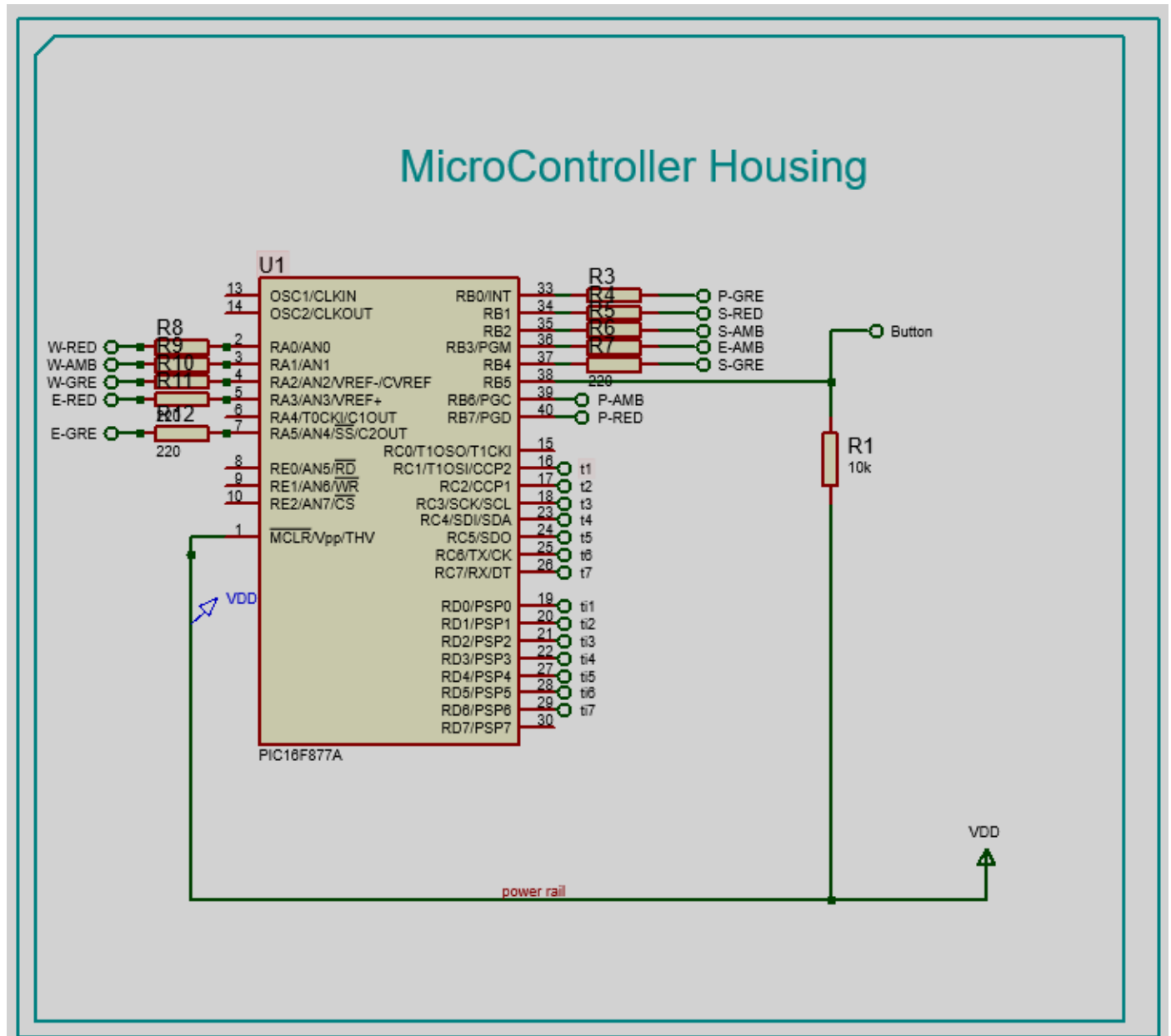
SD-card or cloud logging of events, detectors, and error conditions to aid analysis.

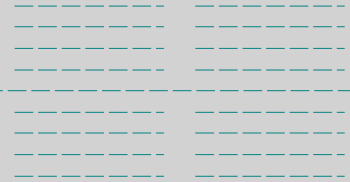
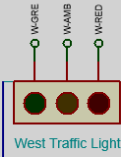
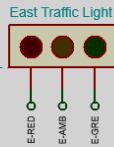
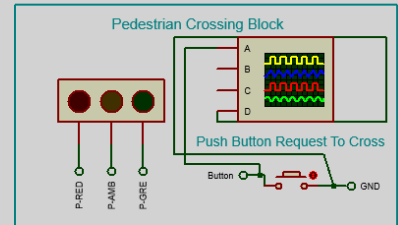
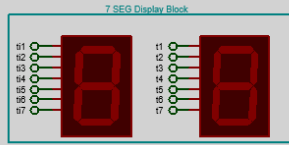
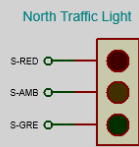
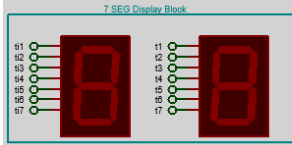
A dashboard) for real-time monitoring, trends, and predictive maintenance.

REFERENCES

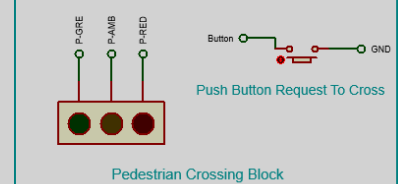
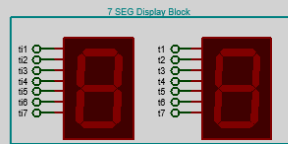
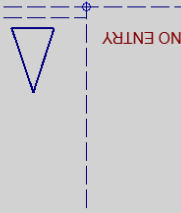
- [1] From Darren Cottingham "Traffic light rules in the UK" [Traffic light rules in the UK | Highway Code Resources](#).
- [2] Saiba P A*1, Afeefa M U#2, Aruna T S#3, Clincy Jose#4, Radhika V M#5 Density Based Traffic Signal System using PIC Microcontroller [Density Based Traffic Signal System using PIC Microcontroller](#)
- [3] From "Lookup" [Using lookup tables to convert numbers to seven segment LED display patterns](#)
- [4] From MOVA ["MOVA - TRL Software"](#)

APPENDIX



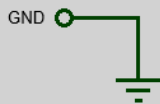


r Housing



MISC

simple GND pin



microcontroller 5V to High Voltage Traffic light

