Computer Engineering design 2 – ENEL3CB

UNIVERSITY OF
KWAZULU-NATAL

PHASE 3 REPORT

Traffic light system with Emergency vehicle detection and control

Shane Dewar – 214502730
11/09/2018

Abstract

Traffic control is becoming an increasing challenge all over the world, especially in densely populated cities where traffic congestion is an everyday problem. It is therefore essential that smarter and more efficient traffic control systems are developed. Another consequence of traffic congestion is the increase in accident rates. EThekwini(Durban) Municipality recorded an increase of 19% in road fatalities per annum [1]. Emergency vehicles such as Ambulances, Police and Firetrucks have difficulty navigating the congested roads to get to the incidents as timely and safely as possible.

This report details the design and implementation of a traffic light system. The system models the green, yellow and red lights of a 4-way road with the following time sequence: Green = 10s , Yellow = 5s , Red = 15s. The current timing sequence of the traffic light is also displayed on seven segment displays.

Added functionality of emergency vehicle(EMV) detection using Infrared is added to the system, which changes the traffic light to green for an approaching EMV to allow safe and fast way through intersections.

The problem specification, design, implementation and testing of this system follow in this report.

Table of contents

1. Problem Overview

The Aim of this project was to design a traffic light system which will model the timing sequence of a traffic light with 3 LED`s Red, Yellow and Green. The current timing sequence of the traffic light must also be displayed on a set of seven segment displays SSD`s.

The project description indicated a timing sequence of 10s for red, 5s for yellow and 15s for green. This, however was changed to a sequence of 15s for red, 5s for yellow and 10s for green. This sequence was chosen for this particular system as it allows equal timing for both adjacent roads which proved more efficient for demonstration purposes.

The system is implemented on a pic16f690 micro-controller using PIC assembly language.

1.1. Added Functionality

This system has an added smart control feature which detects any emergency vehicles approaching the traffic lights. If an emergency vehicle(EMV) is detected, the traffic light will change its sequence to allow the light to be green for the EMV by the time it arrives at the traffic light.

1.2. Constraints

The project required the use of a pic16f690 micro-controller this introduced the following constraints:

1) The Number of Input/output Pins available for use (3 ports – 18 pins in total)
2) The Pic16f690 only has one available interrupt routine
3) Available memory and speed limitations of the micro-controller

The limited number of output and input pins were of particular importance as it limited the solution to only having 2 traffic lights (instead of 4), and using only one emergency vehicle detector (instead of 4).

1.3. System overview

The traffic light system was modelled for a 4-way intersection (shown in figure 1 below). 2 sets of traffic lights are used. The first for the East-West road and the other to indicate the North-South road. The value shown on the Seven segment displays indicate the current timing sequence for the East-West road.

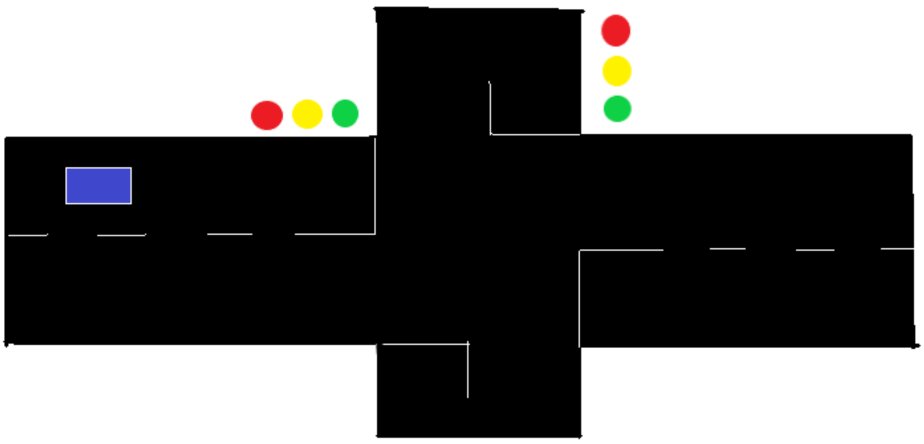

Figure 1: Model of the System implementation

1.3.1. System block diagram

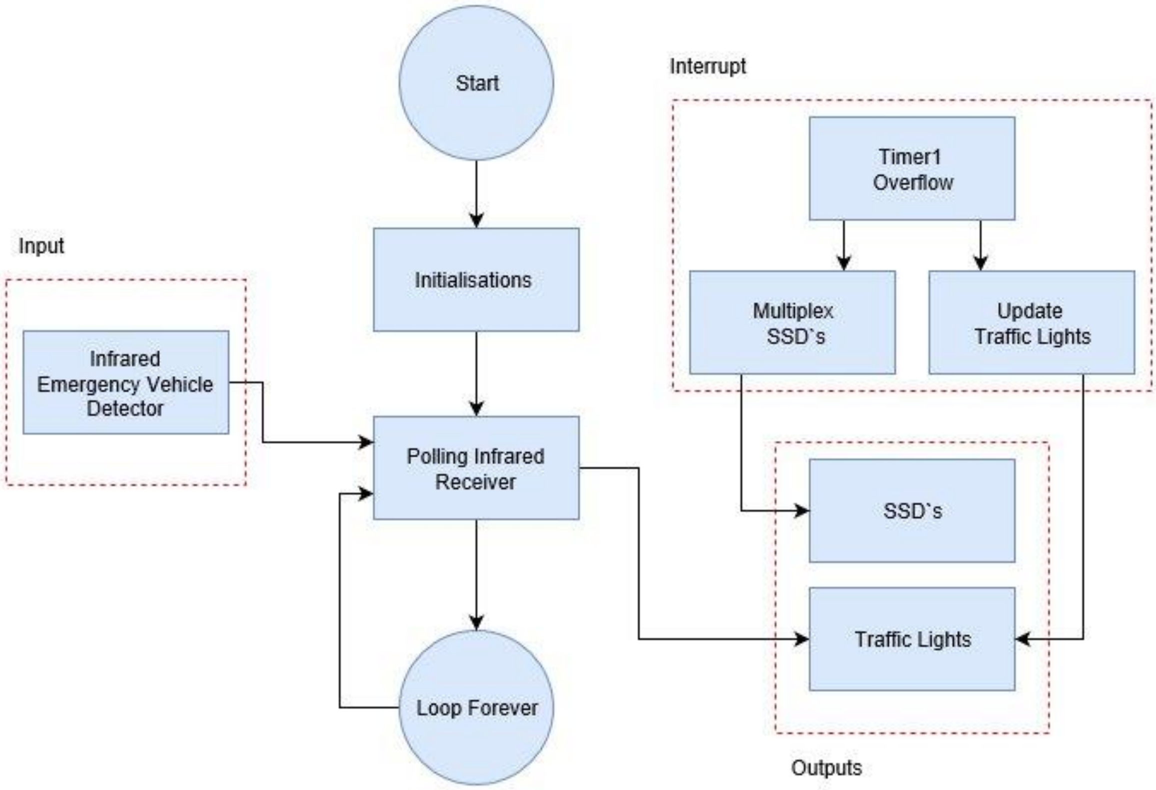

Figure 2: Traffic light system block diagram (designed using www.draw.io)

The system contains:

- 1 Input: The emergency vehicle detector (IR receiver)
- 2 outputs: The traffic light LED`s and the seven segment displays
- 1 interrupt routine: Timer 1 overflow

2. Problem Solution

The system was designed using a top down approach, with a modular implementation of the algorithms and software. Each task was broken up into a sub-routine, designed, tested and integrated into the solution. This section describes the implementation of each task`s algorithm.

Code-Reuse and expansion: Many of the subroutines required in this project were already designed in the previous projects 1-4 of this course. Each module was designed with re-use in mind. Subroutines such as Delay, Bin_To_BCD, and the Multiplexing was already implemented in the previous projects and simply altered to meet the needs of this system.

Each task in this system was executed in separate subroutines to allow for easy expansion or integration with a larger system if required.

2.1. Traffic light timing sequence

The most challenging part of the project was designing the timing sequences for the LED`s. It is only required to update the display every 1s period, and due to the high clock frequency of the micro- controller(4MHz), an extremely large number of clock cycles must be counted before a single second has elapsed.

An interrupt was chosen to control the traffic lights; this allows the micro-controller to focus on other tasks during the very long 1 second delay.

The interrupt routine is executed and controlled by timer 1. It is a 16-bit read/writeable timer that can be configured as a counter. When the counter reaches maximum value, it sets the Timer1 overflow flag which in turn executes the interrupt routine.

An interrupt period of 10mS is implemented as the timer overflow period. This was chosen as every 100 interrupts corresponds with 1S. In addition to this, the 10ms correlates with the multiplexing period. This is explained in section 2.2.

The value loaded into TMR0 was calculated with the following formula:

$$Tout = (65536 - TMR1H{:}TMR1L) * Prescalar * Tclk$$

A pre-scalar of 4 is used, with Tclk = 1µs, therefore TMR1 was set to 63036. This value is

11110110 00111100(binary). The upper 8 bits were loaded into TMR1H, and the lower 8 bits into TMR1L.

The registers associated with the Timer1 interrupt are:

- TMR1H, TMR1L: These set the interrupt frequency

- T1CON: Sets the timer pre-scalar and turns the timer on/off
- PIE1, INTCON: These registers control and turn on the interrupt

2.2. Multiplexing the SSD`s

For this project, multiplexing is achieved in software by connecting both displays to the same 7 data lines, continuously switching each display off and on and displaying the corresponding value. Common Anode SSD's were used for Displays, Thus PNP BJT`S were required to source the current and drive each display. The BJT`S were connected as an inverter pair shown in the diagram below. This allowed the entire display to be controlled on PORTC of the PIC (7 data lines: RC0-RC6, BJT gate: RC7).

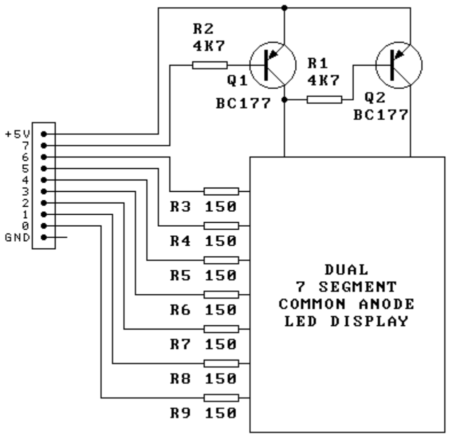

Figure 3: Hardware connections for seven segment display

The Multiplexing was achieved using the same interrupt as the traffic lights; Timer 1. It allows the process to occur in the background while the main Loop deals with polling for emergency vehicle detections.

As shown in section 2.1. above, Timer 1 overflows every 10mS. This equates to an overflow frequency of 100Hz. Research showed that multiplexing the LED`s at a rate of 60Hz or higher is undetectable to the human eye and appears as a clean digit without flicker [2].

Every time the timer1 overflow occurs, the power output pin(RC7) is toggled and the corresponding Tens or units value is outputted on pins(RC0-6).

2.3. Binary to BCD Conversion

The division by 10 conversion method was chosen when outputting the SSD values, as operation speed is a critical component of micro-controller functioning and this method performs the conversion in fewer as well as a constant number of cycles. The Method to perform this operation is as follows:

1) Divide input by 10 using formula:

$$(X + \frac{X}{2} + \frac{X}{8} - \frac{X}{64})/16 = X/10$$

rrf (rotate right) functions – divide by 2.

swapf function - divides by 16.

2) Multiply quotient by 10 using rlf functions(x2) and the formula:

$$2 * (4 * X + X) = 10 * X$$

3) Subtract the result from input number to get Units Value.
4) Finally, the Tens and Units values are concatenated using XOR function.

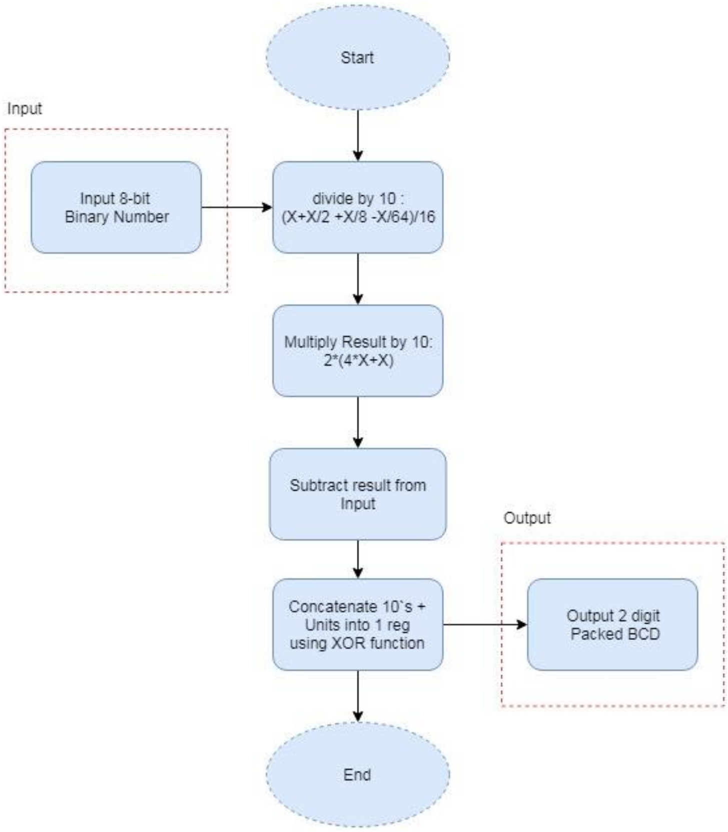

Figure 4: Block diagram of division by 10 method

2.4. BCD to SSD conversion

In order to display the BCD value on the seven segment display, it needs to be converted to an equivalent binary pattern. The following table shows the equivalent patterns for a Common Anode display (1= off, 0 = on)

Decimal	Binary	SSD equivalent
0	00000000	0111111
1	00000001	0000110
2	00000010	1011011
3	00000011	1001111
4	00000100	1100110
5	00000101	1101101
6	00000110	1111101
7	00000111	0000111
8	00001000	1111111
9	00001001	1101111

Table 1: Seven Segment display equivalent values

This was implemented as a lookup table in the code. The Program Counter (PC) is incremented by the corresponding value and the SSD equivalent value is loaded into the W register after which it is outputted on PORTC.

2.5. Emergency Vehicle(EMV) Detection

The EMV detection was implemented using an infrared (IR) transmitter and receiver. The IR receiver is embedded in the road leading to the traffic light intersection, with the transmitter on the emergency vehicle. When the EMV approaches the traffic light, the IR receiver will detect it`s presence and send a signal to the micro-controller.

The micro-controller will then change the traffic light sequence to allow passage for the EMV. This occurs as follows:

- If the traffic light is currently green for the EMV, then the traffic light will remain green but the timer will reset to 10s to allow enough time for the EMV to get through the intersection.
- If the traffic light is currently red for the EMV, then the traffic light for the adjacent road will immediately change to yellow, and 5 seconds later, the light will change green for the EMV.

If the IR receiver is placed at a far enough distance from the intersection (50 Meters or 100 Meters for example), then the traffic light would always be green for the EMV by the time it reaches the intersection.

A 38KHz transmission signal was chosen for this project as it provides the best immunity against environmental IR noise. The receiver used is a TSOP4838 IC (shown in figure 5 below). It contains a IR photodiode and a 38KHz band pass filter to attenuate unwanted frequencies.

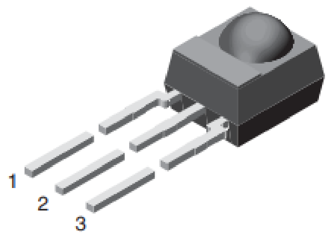

Figure 5: TSOP4838 IC (from datasheet [3])

The output of the receiver is active low, i.e. outputs 5V with no IR detection and 0V when IR is detected.

The transmitter is designed using a 555 timer in astable mode, with a frequency of 38kHz and duty cycle 50%. The transmitter was implemented using the following circuit:

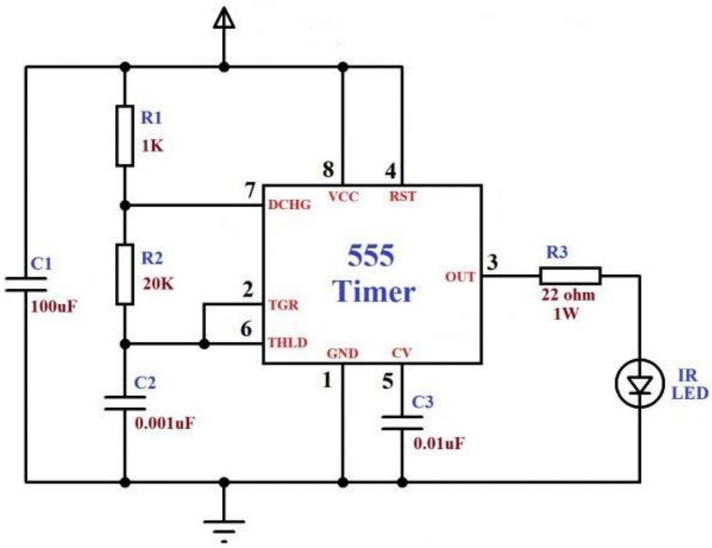

Figure 6: 38KHz frequency generator

The values for R1, R2 and C were calculated using the following formula:

$$f = \frac{1}{T} = \frac{1.44}{(R1 + 2R2) * C}$$

A variable Resistor for R2 was used to adjust the frequency using an oscilloscope to fine tune an exact 38KHZ output.

3. System Implementation

The full circuit was designed using Proteas simulator [4]. The hardware implementation was relatively simple, due to the micro-controller performing all the processing and essentially being the "brains" of the system. Standard design Techniques were followed, such as adding Current limiting resistors to all LED outputs and well as driving the SSD`s with transistors to prevent excess current being sourced from the micro-controller.

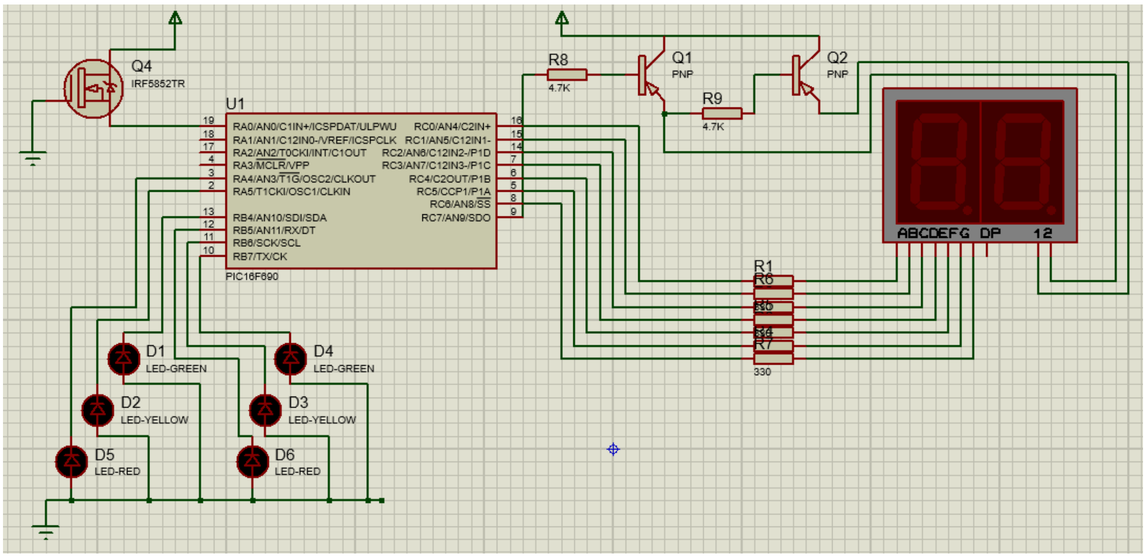

Figure 7: full System hardware design

The System was initially built on breadboard to functionality (shown in figure 8). After testing determined the system was functioning correctly, it was then implemented and soldered onto Vero board (shown in figure 9). Due to time constraint, a PCB version was unable to be implemented.

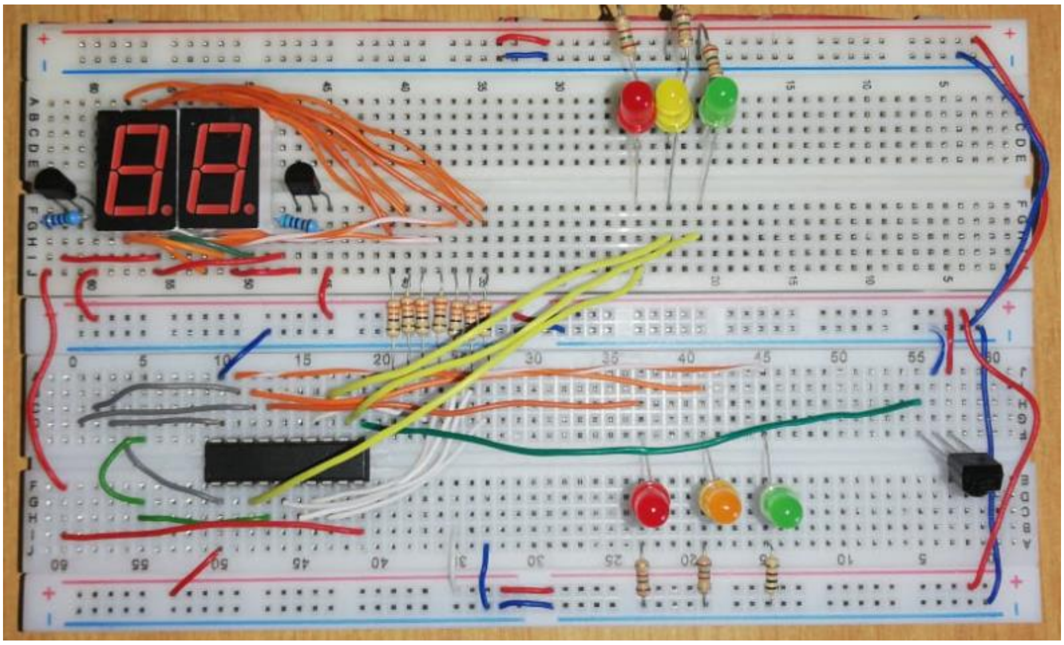

Figure 8: Breadboard implementation

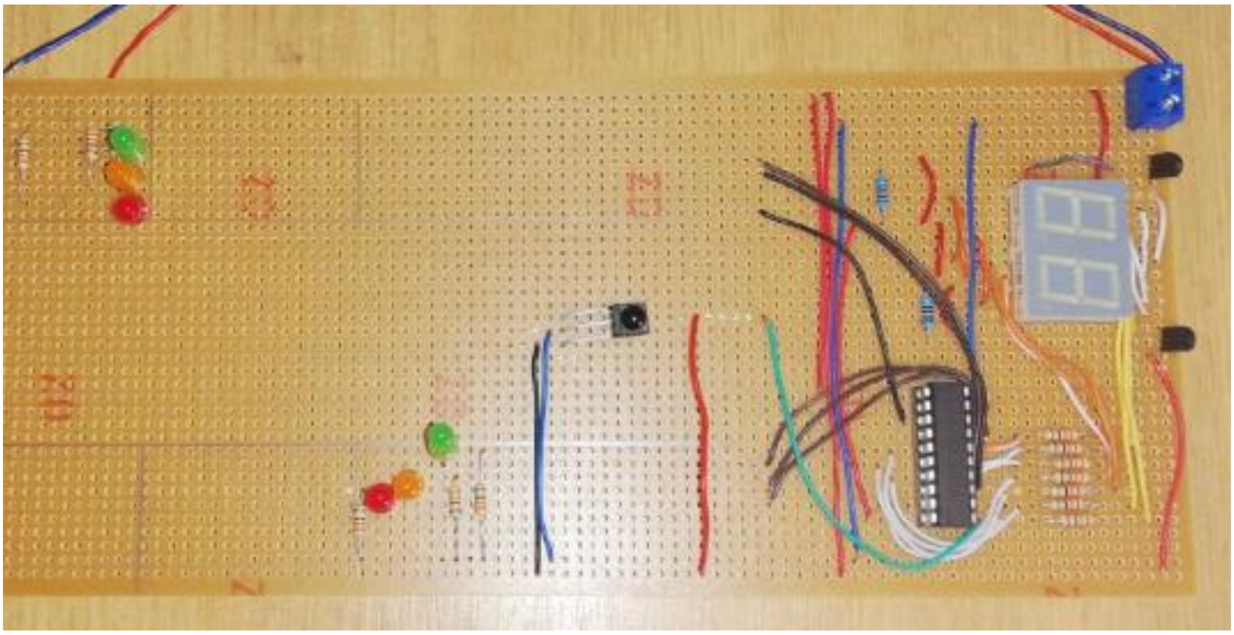

Figure 9: Vero board Final system implementation

4. Testing and Performance

The System was tested and simulated under all possible conditions to ensure functionality. The first stage of testing was Modular Testing. Each Subroutine was individually tested with a variety of inputs and under different conditions to ensure correct output. All the modules performed correctly with no bugs. Following this, integration testing was performed to ensure all modules function together and finally system testing was performed to ensure the entire functions correctly as well as with efficient performance.

The following were outputs of the testing results obtained:

Timing sequence simulations

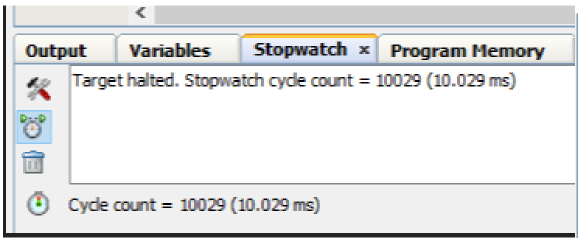

Figure 10: 10ms Interrupt timing

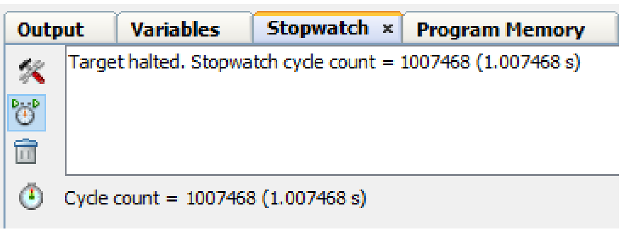

Figure 11: 1S timing Simulation

Figures 10 and 11 above show the Simulated timing sequences. Both of the values (10.029ms and 1.0075s) were within 1% error of the required times. This proved more than satisfactory for the system requirements.

Infrared transmitter and receiver testing:

Figure 12 shows the output square wave of the 555 timer 38kHz frequency transmitter discussed in section 2.5.

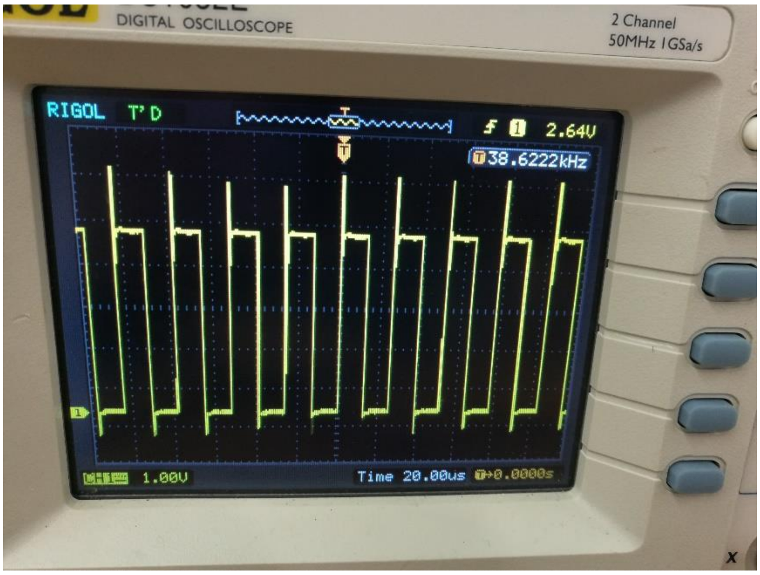

Figure 12: 38KHz 555 timer output

Figures 13 and 14 show the output of the IR receiver when there is no detected IR signals, and when IR signals are detected respectively.

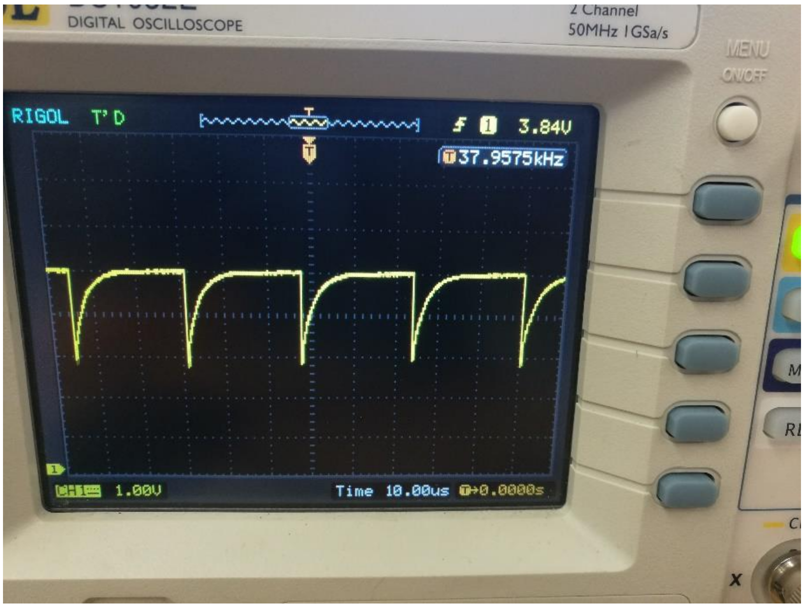

Figure 13: IR receiver output – No Signals detected

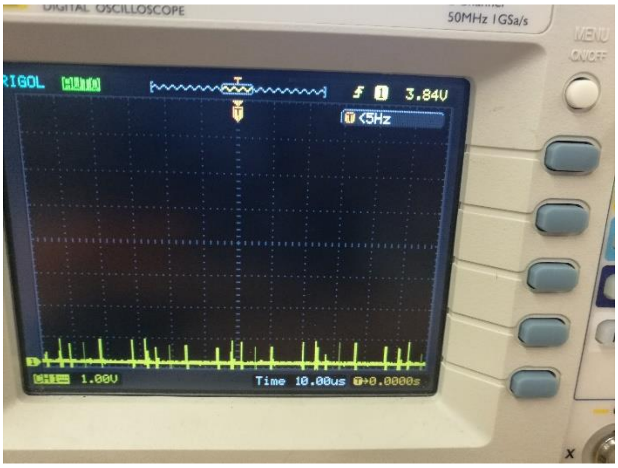

Figure 14: IR receiver output – IR Signals detected

Binary to BCD conversion

A key performance factor of the System is the Binary to BCD conversion. It is crucial to implement a fast algorithm as the function is called every time a display value is updated This algorithm was designed to execute as efficiently as possible. Results are shown below:

Function:	Bin_To_BCD (Div by 10)
Cycles	33
Memory	33
Execution Time: (4MHz)	33µs

Table 2: Binary to BCD sub routine performance

System performance

Program memory: The program contains 350 Lines of Code.

Execution Time: The execution time of the code is dependent on the user as well as interrupts. Therefore, this cannot be accurately measured.

5. Project time analysis

Activity	Time Taken
Research(Timer1, Infrared, interrupts etc.)	4 hours
Software implementation	9 hours
Testing	2 Hours
Hardware Implementation	4 Hours
Report writing	4 hours
Total	**23 hours**

6. Conclusion

The aim of this project was to implement a system that models the timing sequence of a traffic light system on LED`s as well as display the current timing sequence of the LED`s and a pair of multiplexed SSD`s. The system was successfully implemented, performed effectively and had good performance characteristics as shown in this report. The PIC allowed for easy hardware implementation as most of the functions could be performed in software. For this System smooth multiplexing was achieved with no flicker on the SSD`s and the timing functions of the traffics lights were extremely accurate with an error of less than 1%.

The added system functionality of the EMV detection also functioned efficiently and could, with some improvements, be implemented in real world use.

The system could have been improved by using Radio frequency identification(RFID) instead of Infrared for the EMV detection. This would improve security and range of the system. Another possible improvement would be to use a larger micro-controller with more input/ output pins. This would have allowed the entire traffic system to be modelled with 4 sets of LED`s and 4 Infrared detectors.

7. References

[1] EThekwini Municipality "Road incident statistics and road traffic volumes", 2016 [Available online at] http://www.durban.gov.za/City_Services/ethekwini_transport_authority/reports/Documents/ROAD%20ACCIDENT%20STATISTICS%20%20ROAD%20TRAFFIC%20VOLUMES%202014-2015.pdf [Accessed] 17/10/2018

[2] Joel Ghelin "Drive Time multiplexed LED arrays at high current" April 8,2013 [Available online at] https://www.ledsmagazine.com/articles/print/volume-10/issue-3/features/drive-time- multiplexed-led-arrays-at-high-current-magazine.html [Accessed] 17/10/2018

[3] Vishay Electronics (datasheet) "TSOP4838 IR receiver modules" 2017 [Found online at] https://www.vishay.com/docs/82459/tsop48.pdf [Accessed] 18/10/2018

[4] Proteus PCB Design & Simulation software - Labcenter Electronics. 2016. *Proteus PCB Design & Simulation software - Labcenter Electronics*. [ONLINE] Available at: https://www.labcenter.com/. [Accessed] 18/10/2018

8. Appendix

Code:

```
;   Filename: TrafficLights (Project 5)
;   Date Created: 17/ 09/ 2018
;   Author: SHANE DEWAR -214502730
;  Description: This is the assembly code for a Traffic Light Systems
;            Port C: RC0-RC6 - SSD`s
;            Port B: RB7- SSD toggle, RB6 - IR Input ,RB4,5 = LED 2
;            Port A: RA0,1,2 (LEDS) RA4 =LED2
;*****************************************************************************
;*            MICRO CONTROLLER DEFINITIONS:                          *
;*****************************************************************************
list p=16f690           ; chip pic16f690
#include "P16F690.inc"    ; including PIC default
;*****************************************************************************
__CONFIG _CP_OFF & _CPD_OFF & _BOR_OFF & _PWRTE_ON & _WDT_OFF & _INTRC_OSC_NOCLKOUT &
_MCLRE_OFF & _FCMEN_OFF & _IESO_OFF
;*****************************************************************************
;*             VARIABLE DEFINITIONS                          *
;*****************************************************************************
 cblock 20h ;start of general purpose register addresses
   Counter     ;Holds display valuedf
   Counter2
   Seconds
   Mode
   Mode2
   ESet        ;emergency set
   InputVal  ;Binary to BCD conversion registers
   Tens
   Units
   w_temp    ;used for context saving
   s_temp
   p_temp
 endc
;*****************************************************************************
; Reset Vector
;*****************************************************************************
 RES_VECT  CODE    0x0000          ; processor reset vector
  GOTO  SETUP                 ; go to beginning of program

  INT_VECTOR CODE 0x0004      ;interupt vector address
 GOTO ISR
;*****************************************************************************
; MAIN PROGRAM
;*****************************************************************************
MAIN_PROG CODE                  ; let linker place main program

SETUP
    banksel INTCON
      bsf INTCON, GIE         ; Enabling interrupts
      bsf INTCON, PEIE

    clrw                  ; Clearing W register from previous runs
     banksel ANSEL          ; Select bank 2
```

```
    clrf ANSEL              ; Digital I/O

    banksel TRISC           ; Setting up ports: 1-Input 0-Output
     clrf TRISC            ; setting PORTC -Output
      movlw b'01000000'
     movwf PORTB            ; rb7 output(Toggle), 4,5 out(LED2) , 6 input(IR)
    clrf TRISA              ; output - leds

    banksel T1CON           ; Timer 1 setup
     bsf T1CON,5            ; Tmr1 prescalar 4
     bsf T1CON,0            ; Tmr1 on
      movlw b'11110110'      ; Setting up TMR1 for 10ms overflow
     movwf TMR1H
     movlw b'00111100'
    movwf TMR1L

    banksel PIE1
     bsf PIE1,0             ; Timer1 interupt enabled

    banksel PORTC           ; Select Bank 0
     clrf PORTC
      clrf PORTA
       movlw b'00000000'      ; Enable units
      movwf PORTB
     clrf Units             ; Clearing tens+ Units Value
    clrf Tens

    movlw 0x0A              ;10 as an initial Time(for green)
     movwf Counter
      movlw b'00000001'
      movwf Mode            ;mode = green
     movlw 0x64             ;100- to count each second
      movwf Seconds
       bsf PORTA,RA4
      bcf PORTB,RB4
     movlw b'00000100'
     movwf Mode2
     clrf ESet          ; no emergency has occcured
    GOTO LOOP
;*****************************************************************************
;*                  Interupt Service Routine                          *
;*****************************************************************************
;* Description: Timer1 Overflow Interupt:
;*          The Interupt handles the multiplexing by switching each display
;*          on/off and Outputing the Tens or Units Value to PortC
;*****************************************************************************
ISR
   BCF INTCON,GIE     ;disable interupts
    movwf w_temp      ;context saving
     movf STATUS,w
     movwf s_temp
    movf PCLATH,w
    movwf p_temp

   call Bin2BCD
    call Multiplex
```

```
decfsz Seconds      ;decrementing seconds timer
goto Timer
  movlw 0x64         ;reload 100 mSe
    movwf Seconds

btfsc Mode2,0       ;checking if Red Mode
  call LED2
decfsz Counter
     goto Timer

btfsc Mode,0
   movlw 0x05        ;orange next - 010 - 5s
   call Green
btfsc Mode,1
;movlw 0x0F        ;red next - 100 - 15s
    call Orange
  btfsc Mode,2
     ;movlw 0x0A        ;green next - 001 - 10s
call Red
;movwf Counter

clrc
   rlf Mode           ;moving to next mode
    movlw b'00001000'
       subwf Mode,w
       btfss STATUS,Z       ;if mode is out of bounds, reset
     goto Timer
   movlw 0x01
movwf Mode

Timer
bcf PIR1,TMR1IF        ;clearing overflow flag
  movlw b'11110110'      ;reloading timer - 10mss
    movwf TMR1H
      movlw b'00111100'
      movwf TMR1L
    movf Mode,w
   movwf PORTA
  btfss Mode,2        ; re-lighting red led2
 bsf PORTA,RA4

Restore
movf p_temp,w       ;context restores
movwf PCLATH
movf s_temp,w
movwf STATUS
movf w_temp,w
BSF INTCON,GIE     ;re-enabling interupts
retfie
;*****************************************************************************
;*                    Main Loop                                    *
;*****************************************************************************
LOOP
btfsc ESet,0          ;poll infrared input
GOTO LOOP
btfss PORTB,RB6
call Emergency
```

```
GOTO LOOP               ; loop forever

;******************************************************************************
 LookUpSSD
        addwf PCL,f
              retlw b'01000000' ;0
              retlw b'01111001' ;1
              retlw b'00100100' ;2
              retlw b'00110000' ;3
              retlw b'00011001' ;4
              retlw b'00010010' ;5
              retlw b'00000010' ;6
              retlw b'01111000' ;7
              retlw b'00000000' ;8
              retlw b'00010000' ;9
;******************************************************************************
Green
   movlw 0x05      ;orange next - 010 - 5s
     movwf Counter
   bcf ESet,0      ; look for new emergency vehicles
return
;******************************************************************************
Orange
  movlw 0x0F       ;red next - 100 - 15s
   movwf Counter
     movlw 0x0A       ; 10 seconds for green (LED2)
      movwf Counter2
    bsf Mode2, 0      ;set green mode
    bsf PORTB,RB5
  bcf PORTA,RA4
return
;******************************************************************************
Red
  movlw 0x0A          ;green next - 001 - 10s
    movwf Counter
     bsf PORTA,RA4      set second LED`s to Red
      bcf PORTB,RB4
     bcf PORTB,RB5
   movlw b'00000100'   ; red mode
  movwf Mode2
return
;******************************************************************************
LED2
 decfsz Counter2         ;determining 5s of yellow
   return
    bcf PORTB,RB5        ; if 0, change to red
     bsf PORTB,RB4
    movlw b'00000010'
  movwf Mode2
return
;******************************************************************************
Emergency
    btfsc PORTB,RB6
     return
      bsf ESet, 0        ;emergency has occured
       btfsc Mode2,1      ;LED2 orange?
    return              ;already orange so return
```

```
    btfsc Mode2,0      ; LED2 green?
  goto green
   btfsc Mode,2                  ;Main red?
  return
   movlw b'00000001'          ;set green mode
   movwf Mode
   movlw 0x0A                    ;reset 10s Timer
   movwf Counter
  return
green
 movlw 0x05                       ;5 second counter2
  movwf Counter2
   movwf Counter
   movlw b'00000010'          ; mode2 orange
  movwf Mode2
return
;******************************************************************************
;*********************** DELAY SUB-ROUTINE ***********************************
;******************************************************************************
;delay routine with multiple delay values *
;call specified delay value
;or load w with custom value and call Delay
;max 255ms Delay
;******************************************************************************
Del0  retlw 0x00              ;returns immediatly - 0ms
Del1  movlw d'1'             ; Delay 1ms
     goto Delay
Del5  movlw d'5'              ; Delay 5ms
     goto Delay
Del10 movlw d'10'          ; Delay 10ms
     goto Delay
Del20 movlw d'20'           ; Delay 20ms
     goto Delay
Del50 movlw d'50'          ; Delay 50ms
     goto Delay
Del100 movlw d'100'      ; Delay 100ms
      goto Delay
Del250 movlw d'250'      ; Delay 250ms
Delay  movwf count1
d1   movlw 0xC7              ;Delay 1ms
    movwf counta
    movlw 0x01
    movwf countb
Delay_0
    decfsz counta, f
      goto $+2
        decfsz countb,f
          goto Delay_0
      decfsz count1,f
    goto d1
    retlw 0x00
;******************************************************************************
;*********************** Multiplex ************************************
;******************************************************************************
; This Method determines which Display to Toggle and outputs the corresponding
; Tens or Units Value to Output to PORTC
;******************************************************************************
```

```
Multiplex
    btfsc PORTB,7            ; Check if Tens or Units is currently on
   GOTO toggle10
   GOTO toggle01
toggle01
     movf Units,w
      andlw 0x0f                    ; Make sure in range of table
   call LookUpSSD
     movwf PORTC                    ; Output Units digit
      movf PORTB, w
        xorlw b'10000000'           ; Toggle RB7
      movwf PORTB
   return
toggle10
   movf Tens,w
     andlw 0x0f                     ; Make sure in range of table
      call LookUpSSD
      movwf PORTC                   ; Output Tens digit
      movf PORTB, w
      xorlw b'10000000'             ; Toggle RB7
     movwf PORTB
    btfsc Mode2,1                   ;orange?
      bsf PORTB,RB4
     btfsc Mode2,0                  ;green?
    bsf PORTB,RB5
   return
;*****************************************************************************
;* Module Name: Bin2BCD
;* Description:
;* This Function uses the Division by 10 method to convert the binary input
;* value stored in the InputVal reg, to a packed BCD value which is returned in
;* the W reg.
;*
;* This is acheived by initially dividing the the Input value by 10 and saving
;* the quotient(Tens BCD value) in bcd1 reg. the division by 10 is calculated
;* using the following formula:
;*                      x/10 = (x + x/2 + x/8 - x/64)/16
;* where each division(2,8,16,64) is achieved with a corresponding number of rrf
;* functions(each rotate right divides by 2).
;* The units value is obtained by multiplying the Tens value by 10 using the
;* form.ula: rlf( rlf(rlf(X)) + X)= 2*( 2*(2*( X)) + X) = 2*(4*X +X) = 10*X
;* This value is then subtracted from the original binary value to obtain the
;* units value
;* The Units and Tens are concatenated using XOR function and returned in the w
;* register.
;*****************************************************************************
;* Registers:   W, InputVal,bcd1,bcd2
;* Inputs:      InputVal(Binary value)
;* Outputs:     Packed BCD(in the W register)
;* Performance:  cycles : 33
;*****************************************************************************
Bin2BCD
        movf Counter,w
                sublw d'99'                 ; Test if >99
                 btfss STATUS,C
               clrf Counter
                movf Counter,w
```

```
        movwf InputVal
    movwf Tens                    ; Copying input value
    movwf Units
CalcTens:
        clrc                      ; Clearing carry flag
         rrf InputVal,f           ; x/2
        movf InputVal,w
        addwf Tens,f               ; x + x/2
         rrf InputVal,f           ; x/4
        clrc
         rrf InputVal,f           ; x/8
        movf InputVal,w
        addwf Tens,f              ; x + x/2 + x/8
         rrf InputVal,f           ; x/16
        clrc
         rrf InputVal,f           ; x/32
        clrc
         rrf InputVal,f           ; x/64
        clrc
         movf InputVal,w
         subwf Tens,f             ; x + x/2 + x/8 - x/64

        swapf Tens,w             ; (x + x/2 + x/8 - x/64)/16
         andlw 0x0f              ; Clearing fractional nibble
         movwf Tens
        movwf InputVal
CalcUnits:
        clrc
         rlf InputVal,f            ; y*2
         rlf InputVal,f            ; y*4
        addwf InputVal,f         ; (y*4 + y) = y*5
         rlf InputVal,f            ; y*10 = 10* Tens values
        movf InputVal,w
        subwf Units,f             ; Sub 10*Tens from Original value to get unit
Return
END
```