A-Level Electronics Project

Task 1 - Microcontroller System

Traffic Light Control System

Thomas Boxall

march 2022

BEXHILL COLLEGE: 56635
CANDIDATE NO: 5377

Contents

Chapter 1

Introduction

1.1 Introduction to the Project

Traffic lights are fundamental to the smooth running of the road networks. Without them vehicle drivers would be placed in potentially dangerous situations multiple times per journey, where they approach junctions with multiple roads; not to mention the risk which pedestrians would take every time they cross the road. Traffic lights (amongst other electronically controlled traffic management systems) provide some order to what would otherwise be uncontrolled chaos.
I chose those this project with the aim of understanding how a traffic light control system works as well as understanding how microcontrollers work, interfacing with their inputs and outputs. Whilst the unknown of how microcontrollers work added to the complexity, the comfort of using traffic lights daily makes this feel like its a manageable task to complete in the time allocated.

1.2 Definition of Traffic Lights

Everyone knows what traffic lights do, but for this project I feel it is deeply important to have an understanding of the definition of a traffic lights.
Collins English Dictionary[1] defines 'Traffic Lights' as:

> Traffic lights are sets of red, amber, and green lights at the places where roads meet. They control the traffic by signalling when vehicles have to stop and when they can go. Traffic lights can also be referred to as a traffic light.

[1] https://www.collinsdictionary.com/dictionary/english/traffic-light

Chapter 2

Research

Before I start to design my traffic light control system, I need to research existing designs, to understand both how they work and how they are controlled.

2.1 How Traffic Lights Work

Traffic lights are generally divided into two categories, they will either be based off of a timing circuit or they will have some form of vehicle sensor which will trigger the changing of the lights. Each of these have their own pros and cons. Sometimes a combination of the two is preferred.

2.1.1 Systems with Vehicle Sensors

Traffic light control systems which are based off of a vehicle sensor or sensors alone are generally inadequate for for junctions in areas with a higher population density. Using the diagram below (fig. 2.1 as an example; the blue cars are able to cross the bridge, but as there are blue cars constantly arriving at traffic light set A, the system doesn't see that there are cars waiting at traffic light set B therefore it doesn't change. This means that there is a pile up of cars at traffic lights B.

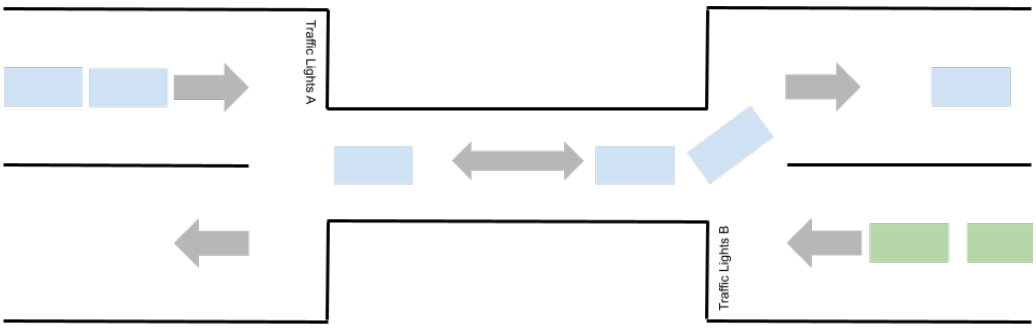

Figure 2.1: Diagram of traffic flow over a bridge

2.1.2 Systems with Timers

Traffic light control systems which are based off of a timer alone are generally more suited for areas which receive higher levels of traffic. This is because they allow one set of traffic to move for a set length of time then allow the other set to move for a set length of time, then repeats.

2.1.3 Systems with Timers and Vehicle Sensors

Traffic light control systems which are based off of a timer and vehicle sensor(s) are the most efficient, however with the added efficiency comes additional complexity. This is the best type of system as

(using the diagram in Fig. 2.1) the system would be able to tell that the blue cars have had a chance to go and there are green cars waiting therefore it can let the green cars go.

2.2 Traffic Light Controlled Junctions

There are a number of different types of traffic light and pedestrian crossing systems[1], each has a different priority pattern and each uses a different combination of vehicle/pedestrian sensors and timing systems.

2.2.1 EXAMPLE 1: Two-Way Traffic Lights

Figure 2.2: Image of the two-way junction

Data from Google Maps (2022). Available from: https://www.google.com/maps/@50.7865833,0.3056209,74m/data=!3m1!1e3 [Accessed 13 03 2022]

This junction has two identical sections to it. Each has a set of traffic lights to stop a single carriageway of traffic and a pedestrian crossing to allow pedestrians to cross the dual carriage which this is situated on. Due to the fact that this is on a dual carriageway, it can safely be assumed that this traffic light system is controlled by the pedestrian crossing alone, without any vehicle sensors or timing control systems. This assumption would work by the pedestrian pressing the button which indicates to the system that they are ready to cross then the system waiting a set length of time, allow the pedestrians to cross for a set length of time then return to a green light for the traffic.

[1] https://www.highwaycodeuk.co.uk/rules-for-pedestrians-crossings.html

2.2.2 EXAMPLE 2: Complex Junction

Figure 2.3: Image of the complex junction

Data from Google Maps (2022). Available from: https://www.google.com/maps/@50.7690679,0.2817194,121m/data=!3m1!1e3 [Accessed 13 03 2022]

This is a much more complicated junction where there are multiple lanes of traffic which converge into single lanes as well as pedestrians which have to cross. I assume this is controlled using a mixture of timings and pedestrian triggers. The big difference between this junction and the junction above is that this junction can allow pedestrians to cross simultaneously while vehicles also move, making it more efficient.

Chapter 3

Design

3.1 Specification

1. The traffic light control system should be easy to use.

2. The traffic light control system should be as efficient as possible, reducing heat dissipated to the environment.

3. The traffic light control system should recieve an input from a pedestrian and take 40 seconds (±20s) to allow them to cross.

4. The traffic light control system should not favour any particular road user over an other as well as giving equal chances to all direction of traffic.

5. The traffic light system should ensure that when a direction of traffic is not able to go, its red stop LED is illuminated.

6. The traffic light control system should take a 0V and +5V (within ±0.5V) input.

7. The traffic light control system should alert 'pedestrians' in at least two different ways that it is their turn to cross.

8. The traffic light control system should be developed in a way such that, the code is clear to read and understand, to ensure future developments can be carried out easily.

3.2 Junction design

Now I have my specification, I am able to design the junction which I will design the traffic light control system for. I will be using a junction in my home town (Eastbourne) as inspiration.

3.2.1 Real Life Junction

Figure 3.1: Image of the inspiration junction

Data from Google Maps (2022). Available from: https://www.google.com/maps/@50.7701274,0.2786944,118m/data=!3m1!1e3 [Accessed 13 03 2022]

This is a three-way traffic light controlled junction with pedestrian crossing for two of the three ways.

3.2.2 Abstracted Junction

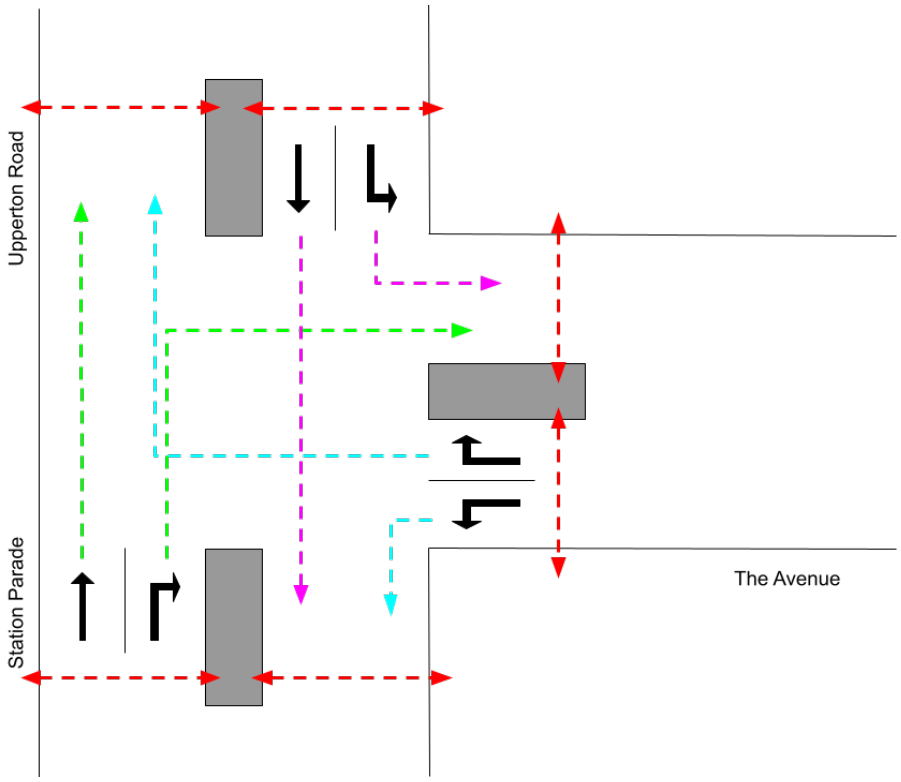

Figure 3.2: Diagram of the junction used in this project

The diagram above shows an abstracted version of the junction shows in figure 3.1. This abstracted version has four colours dashed lines drawn on it. These show the routes which can be taken by either the pedestrians or cars. The red route indicates a pedestrian crossing. The blue, pink and green lines indicate routes which can be taken by vehicles. The grey boxes indicate pedestrian islands. Additional pedestrian crossing routes have been included in my abstracted diagram. This is because in the real junction, the areas which do not have pedestrian crossings are extremely dangerous as cars travel at high speeds.

3.3 Traffic Light Sequence

After working out the type of junction I will design an traffic light control system for, I am now able to work out the sequence which the traffic lights will cycle through. At this point, I have also decided that the traffic light sequence will be contained within a subroutine; hence the references about 'Main Program' in the diagram below.

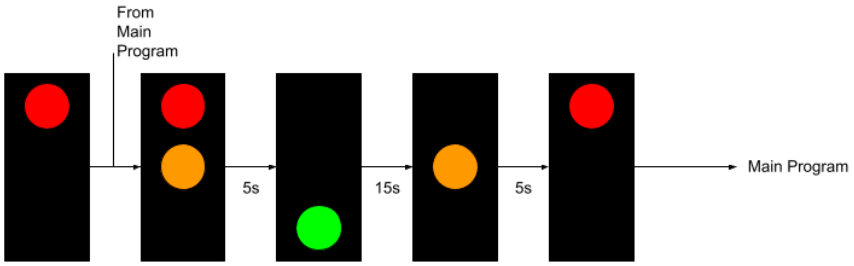

Figure 3.3: Sequence of lights in the traffic lights

3.4 Inputs and Outputs

The PIC16F88 microcontroller has 16 input output bits, divided into two 'ports' of 8 (PORTA and PORTB). In my system, all inputs and outputs will be active high (logic 1).

3.4.1 PORTA

0 Output, red LED

1 Output, amber LED

2 Output, green LED

3 Output, red LED

4 Output, amber LED

5

6 Output, red LED

7 Output, amber LED

3.4.2 PORTB

0 Output, green LED

1 Input, button

2 Output, white LED

3 Output, red LED

4 Output, green LED

5 Output, buzzer

6

7 Output, green LED

3.5 Component of the control system

The control system will be based off of a PIC16F88 microcontroller, this will be the heart of the system. Alongside the microcontroller, I will also need some LEDs (in red, amber and green), a buzzer and a button.

3.5.1 Components list

- 4 red LEDs
- 4 green LEDs
- 3 amber LEDs
- 1 white LED
- 2 eight 220KΩ resistor packs
- 3 220kΩ resistors
- 1 1KΩ resistor
- 1 Push-To-Make button
- 1 PIC16F88 Microcontroller

In reality, my circuit will only need 12 220KΩ resistors. However, for convenience, and minimising the number of components on the breadboards, it is easier to use 2 resistor packs and a number of loose resistors.

Chapter 4

Algorithm Design

After planning my traffic light control system, I can now begin to design the algorithms. Looking at the exam board template provided to us, it is clear that the microcontroller can run subroutines as well as a procedural main program code section.
Using subroutines is the most efficient way to program as it allows you to use the same lines of code a number of times, not only does this reduce the total number of lines of code you have to program but it also reduces the number of lines of code which are flashed onto the microcontroller. Using this knowledge, I designed my main program to call a number of subroutines.

Shown below are the flowcharts and initial iteration of the algorithms. Full code listings are available in Appendix A. Testing and debugging of the code is available in Chapter 5.

4.1 Main Program

This code gets run automatically once the microcontroller has initialised.
This algorithm works by initialising the stop LEDs by turning them all on. It then sequentially cycles through all of the sets of traffic lights, allowing them to cycle each time. After each set of traffic lights, it runs the `checkPedstrian` algorithm which will check if there are pedestrians waiting to cross. This cycle repeats until the system is powered down.

Flowchart

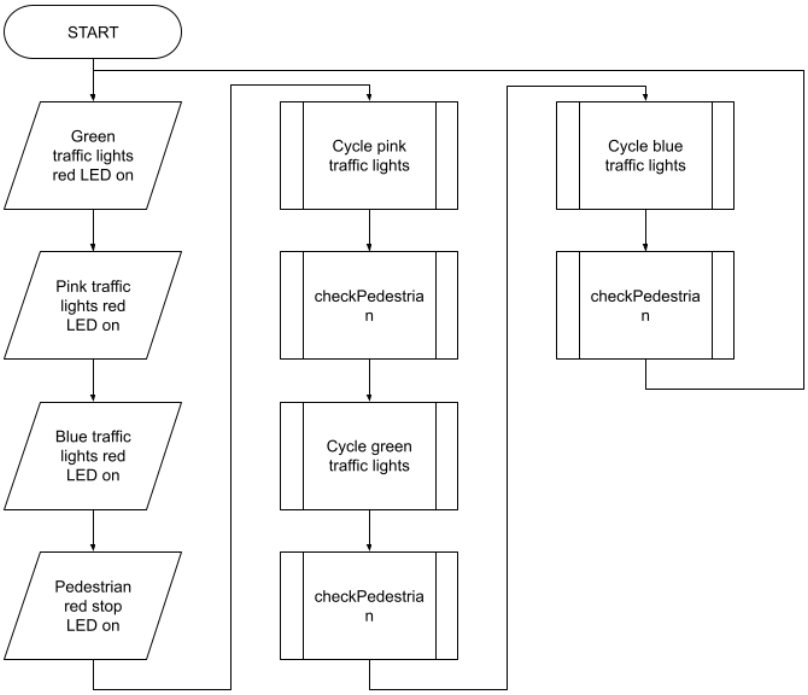

Figure 4.1: Main program flowchart

Code

```
;first turn on all stop leds
  bsf PORTA, 0 ;turn on pink set stop eld
  bsf PORTA, 3 ;turn on green set stop led
  bsf PORTA, 6 ;turn on blue set stop led
  bsf PORTB, 3 ;turn on pedestrian stop led

mtop call pinkTrafficLightSequence
  call checkPedestrian
  call greenTrafficLightSequence
  call checkPedestrian
  call blueTrafficLightSequence
  call checkPedestrian
  goto mTop
```

Listing 4.1: Main sequence code

4.2 checkButton Subroutine

This algorithm is used to check if the pedestrian button has been pressed and if it has, turn on the red wait LED which would be found at traffic lights with pedestrian crossings.

Flowchart

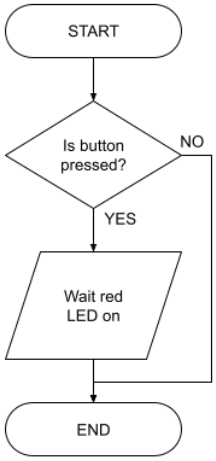

Figure 4.2: Check button subroutine flowchart

Code

```
;function to check the button
checkButton
  btfss PORTB, 1 ;skip next line of code if button pressed (active high)
  return ;can go back to where called from
  bsf PORTB, 2 ;turn red wait LED on
  return ;go back to main code
;end of function
```

Listing 4.2: checkButton subroutine

4.3 checkPedestrian Subroutine

This algorithms is used to read the current state of the pedestrian wait LED. Depending on this state, it will either return straight to the main code or run the `cyclePedestrian` subroutine.

Flowchart

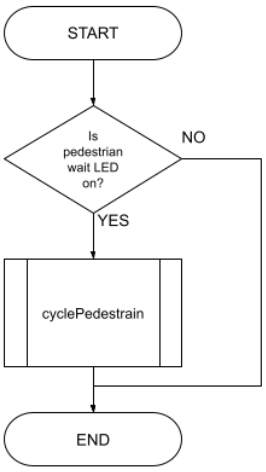

Figure 4.3: checkPedestrain subroutine flowchart

Code

```
checkPedestrian
  ;function to check if the wait led is on
    ;if yes - cycle pedestrian crossing
    ;if no - return to main
  btfsc PORTB, 2 ;skip next line if wait led NOT on therefore if on, run
   ↪ next line
  call cycleCrossing
  return
;end function
```

Listing 4.3: checkPedestrian subroutine

4.4 cycleCrossing Subroutine

This algorithm will control the pedestrian crossing to allow the pedestrians to cross.

Flowchart

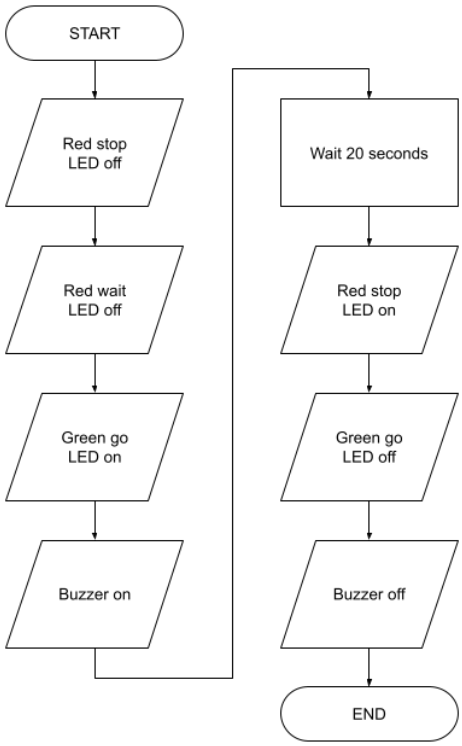

Figure 4.4: cycleCrossing subroutine flowchart

Code

```
; function to cycle the pedestrian crossing
cycleCrossing
  bcf PORTB, 2 ;turn off red wait LED
  bcf PORTB, 3 ;turn off red stop LED
  bsf PORTB, 4 ;turn on green go LED
  bsf PORTB, 5 ;turn on buzzer
```

```
7   call wait1000ms ;wait 1second
8   call wait1000ms ;wait 1second
9   call wait1000ms ;wait 1second
10  call wait1000ms ;wait 1second
11  call wait1000ms ;wait 1second
12  call wait1000ms ;wait 1second
13  call wait1000ms ;wait 1second
14  call wait1000ms ;wait 1second
15  call wait1000ms ;wait 1second
16  call wait1000ms ;wait 1second
17  call wait1000ms ;wait 1second
18  call wait1000ms ;wait 1second
19  call wait1000ms ;wait 1second
20  call wait1000ms ;wait 1second
21  call wait1000ms ;wait 1second
22  call wait1000ms ;wait 1second
23  call wait1000ms ;wait 1second
24  call wait1000ms ;wait 1second
25  call wait1000ms ;wait 1second
26  call wait1000ms ;wait 1second
27  ;now have waited 20seconds so invert things and return to main sequence
28  bcf PORTB, 4 ;turn off green go LED
29  bsf PORTB, 3 ;turn on red stop LED
30  bcf PORTB, 5 ;turn off buzzer
31  return ;return to the main program
32 ;end of function
```

Listing 4.4: cycleCrossing subroutine

4.5 wait5SecondsCheckButton Subroutine

This algorithm is much more complicated that I originally anticipated it to be. What I wanted it to do was to wait 10ms then run the `checkButton` subroutine. To do this, I would have a loop which counted to 500. This is not possible using this PIC microcontroller as it has a maximum number size of 256. To get around this, I am using two loops, X and Y. This will allow me to wait for five seconds and every 10ms of that, check if the button is pressed or not.

Flowchart

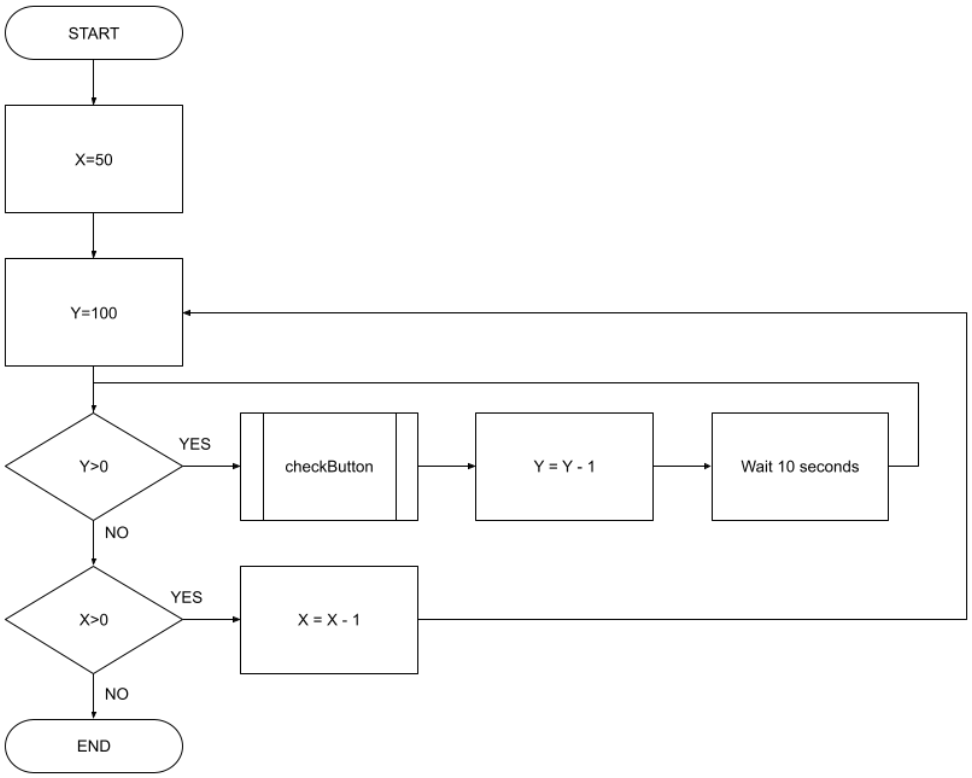

Figure 4.5: wait5SecondsCheckButton subroutine flowchart

Code

```
waitFiveSecondsCheckButton
  ;first, stary loop x
  movlw d'51' ;set val of 51 into w register
  movf loopX ; move the val in the working register (51) into loopX

topX decfsz loopX, 1 ;decrement loopX and place the new value back into
    ↪ loopX
  goto startY ;line skipped if result is 0
  return

startY movlw d'101'; ;set value of 101 into w register
  movf loopY ;move value in working register (101) into loopY

topY decfsz loopY, 1 ;decrement loopY and place new value back into loopY
  goto finalPart ;goto final part of the function
  goto startX ;run if loopY = 0

finalPart call checkButton
  call wait10ms ;wait for 10ms function
  goto startY ;go back up to startY and loop around again
;end of function
```

Listing 4.5: wait5SecondsCheckButton subroutine

4.6 Traffic lights

The traffic light control algorithm is fundamentally the same for the three sets of traffic lights. This is shown in the flowchart, as there is only one for the three subroutines. The only difference between the three sets is which IO bits are changed.

Flowchart

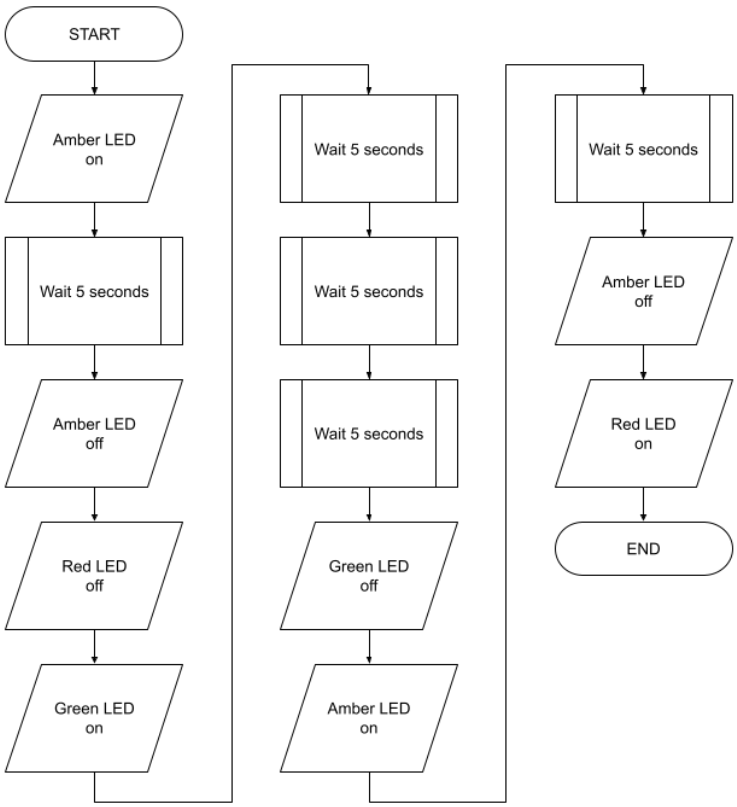

Figure 4.6: trafficLights subroutine flowchart

Code

Pink set

```
pinkTrafficLightSequence
  bsf PORTA, 1 ;turn on amber led
  call waitFiveSecondsCheckButton ;wait 5s
  bcf PORTA, 1 ;turn off amber led
  bcf PORTA, 0 ;turn off red led
  bsf PORTA, 2 ; turn on greeen led
  call waitFiveSecondsCheckButton
  call waitFiveSecondsCheckButton
  call waitFiveSecondsCheckButton
  bcf PORTA, 2 ;turn off green led
  bsf PORTA, 1 ;turn on amber led
  call waitFiveSecondsCheckButton
  bsf PORTA, 0 ;turn on red led
  bcf PORTA, 1 ;turn off amber led
  return ;go back to main code
;end of function
```

Listing 4.6: Pink Traffic Light sequence

Green set

```
greenTrafficLightSequence
  bsf PORTA, 4 ;turn on amber led
  call waitFiveSecondsCheckButton ;wait 5s
  bcf PORTA, 4 ;turn off amber led
  bcf PORTA, 3 ;turn off red led
  bsf PORTA, 5 ; turn on greeen led
  call waitFiveSecondsCheckButton
  call waitFiveSecondsCheckButton
  call waitFiveSecondsCheckButton
  bcf PORTA, 5 ;turn off green led
  bsf PORTA, 4 ;turn on amber led
  call waitFiveSecondsCheckButton
  bsf PORTA, 3 ;turn on red led
  bcf PORTA, 4 ;turn off amber led
  return ;go back to main code
;end of function
```

Listing 4.7: Green Traffic Light sequence

Blue set

```
blueTrafficLightSequence
  bsf PORTA, 7 ;turn on amber led
  call waitFiveSecondsCheckButton ;wait 5s
  bcf PORTA, 7 ;turn off amber led
  bcf PORTA, 6 ;turn off red led
  bsf PORTB, 0 ; turn on greeen led
  call waitFiveSecondsCheckButton
  call waitFiveSecondsCheckButton
  call waitFiveSecondsCheckButton
  bcf PORTB, 0 ;turn off green led
  bsf PORTA, 7 ;turn on amber led
  call waitFiveSecondsCheckButton
  bsf PORTA, 6 ;turn on red led
  bcf PORTA, 7 ;turn off amber led
  return ;go back to main code
;end of function
```

Listing 4.8: Blue Traffic Light sequence

Chapter 5

Testing

Now I have designed the algorithms and developed the code. I can flash the microcontroller chip with the program. To do this, I will first need to compile the assembly language program into `.hex` format which can be programmed onto it.

```
BUILD SUCCESSFUL (total time: 106ms)
Loading code from C:/Users/james kimber/MPLABXProjects/Thomas Boxall.X/dist/default/production/Thomas_Boxall.X.production.hex...
Program loaded with pack,PIC16Fxxx_DFP,1.2.33,Microchip
Loading completed
```

Figure 5.1: Screenshot of MPLABX showing my code successfully compiling

After constructing the circuit as specified in my circuit design (*Appendix B)*, I connected the breadboard to power and let the program run
This didn't work at first. After some troubleshooting, I realised that the problem was within the `waitFiveSecondsCheckButton` subroutine. To fix this, I replaced it with a new subroutine, which only waited for one second, this mitigated the need for a second loop as I was only looping to 100, rather than 100 and 5 in two separate loops. This does increase the size of my program as I have to call the same function five times to achieve the same wait time as I would for the five second subroutine. This worked first time, and the microcontroller cycled through the traffic light sequence perfect first time.
Now that the traffic lights are working, I am able to check if the pedestrian crossing is working and triggering correctly. To test this, I can press the button while the traffic lights are in a random state and observe the outcome. This worked first time, with the white wait LED turning on as soon as the button was pressed then when the current set of traffic lights reached the end of their sequence, the pedestrian crossing cycled as laid out in the design. The timings felt extremely out of proportion, so I reduced the crossing time to 3 seconds, this made the pedestrian crossing time feel better.

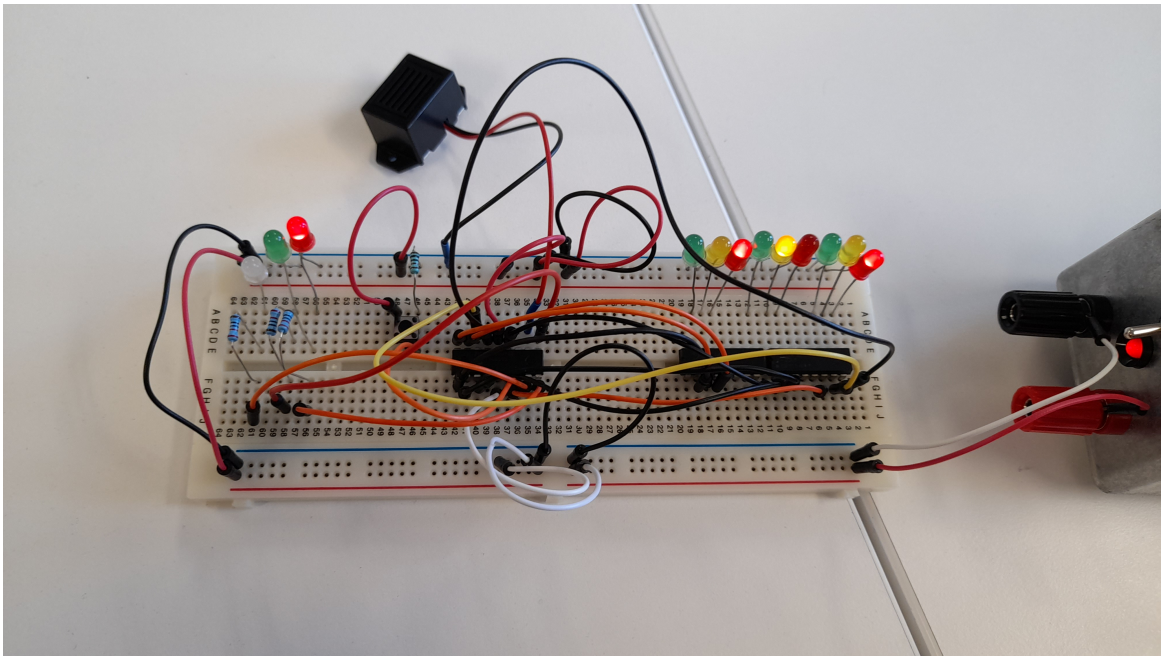

Figure 5.2: Circuit connected with temporary jumper wires

This means that I have now completed construction of the system. I will now need to neaten the circuit.

5.1 Sequence testing

There are two different sets of sequences which I will need to check. One being the traffic light sequence and the other being the pedestrian crossing sequence.

5.1.1 Traffic Lights sequence

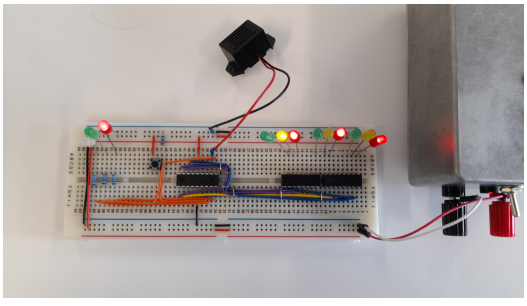

Figure 5.3: State 1 - pink traffic lights in red and amber phase

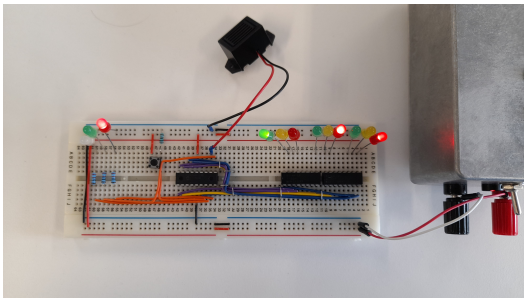

Figure 5.4: State 2 - pink traffic lights in green phase

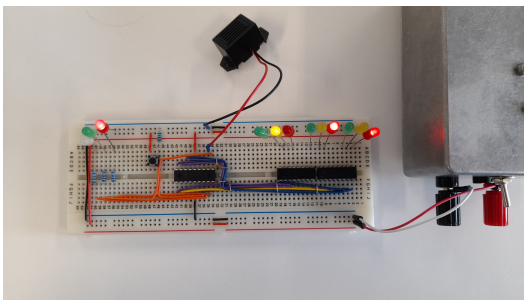

Figure 5.5: State 3 - pink traffic lights in amber phase

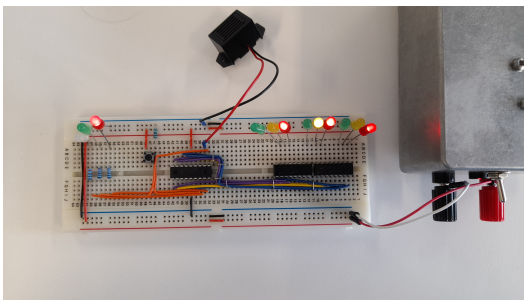

Figure 5.6: State 4 - green traffic lights in red and amber phase

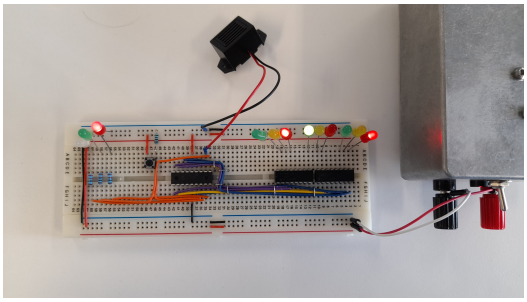

Figure 5.7: State 5 - green traffic lights in green phase

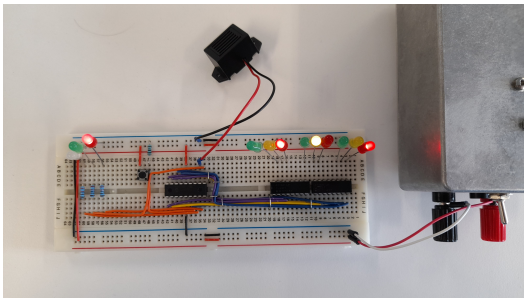

Figure 5.8: State 6 - green traffic lights in amber phase

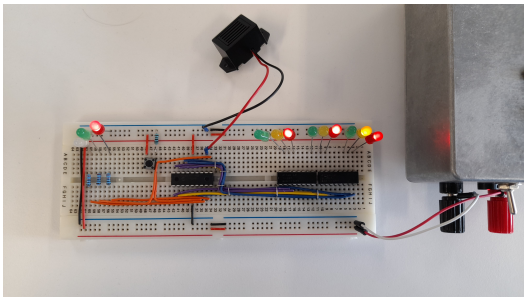

Figure 5.9: State 7 - blue traffic lights in red and amber phase

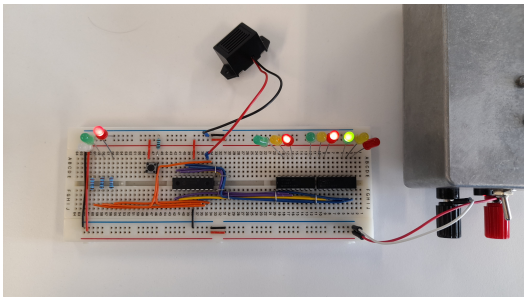

Figure 5.10: State 8 - blue traffic lights in green phase

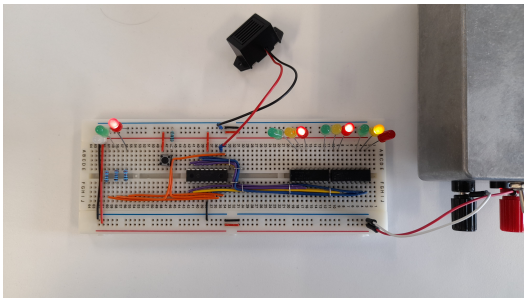

Figure 5.11: State 9 - blue traffic lights in amber phase

The system then loops back to state 1.

5.1.2 Crossing Sequence

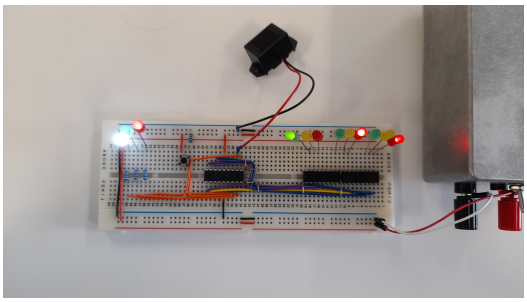

Figure 5.12: Crossing state 1 - wait button on phase

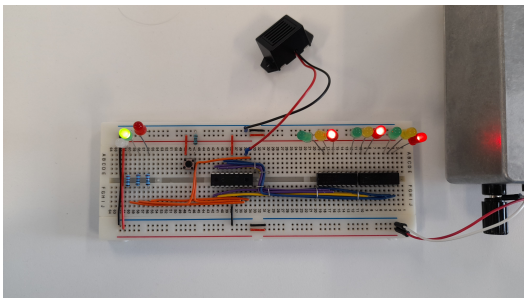

Figure 5.13: Crossing state 2 - safe to cross phase

Chapter 6

Evaluation

To evaluate my system, I will use my specification to evaluate if I have designed a satisfactory system.

6.1 Specification Evaluation

1. **The traffic light control system should be easy to use**
   The traffic light control system is operated with only a single button. In a real world installation, this button would be clearly labelled with its function. This means that this point can be marked as achieved.

2. **The traffic light control system should be as efficient as possible, reducing heat dissipated to the environment**
   As my circuit is using a low voltage and current input, there isn't much energy to be dissipated. This low energy input requirement is partially due to the fact that I chose to use CMOS chips rather than TTL. TTL require a much higher operating voltage and current therefore they dissipate a much higher amount of energy. Therefore this point can be marked as achieved.

3. **The traffic light control system should recieve an input from a pedestrian and take 40 seconds (±20s) to allow them to cross.**
   With the timings currently programmed, the longest wait a pedestrian could have is 20 seconds. Therefore, this point can be can be marked as achieved.

4. **The traffic light control system should not favour any particular road user over an other as well as giving equal chances to all direction of traffic.**
   The traffic light control system is programmed to sequentially cycle through the different routes which cars can take and after each, check if there are pedestrians waiting to cross. If there are, they will be allowed to cross. Therefore this point can be marked as achieved.

5. **The traffic light system should ensure that when a direction of traffic is not able to go, its red stop LED is illuminated**
   From testing my system, I have observed that when a direction of travel is not able to go, its red stop LED is illuminated. Therefore this point can be marked as achieved.

6. **The traffic light control system should take a 0V and +5V (within ±0.5V) input**
   The traffic light control system takes a 0V and +5V input. Therefore, this point can be marked as achieved.

7. **The traffic light control system should alert 'pedestrians' in at least two different ways that it is their turn to cross.**
   The traffic light control system alerts the pedestrians that it is their turn to cross by illuminating a green LED and by sounding a buzzer. Therefore, this point can be marked as achieved.

8. **The traffic light control system should be developed in a way such that, the code is clear to read and understand, to ensure future developments can be carried out**

easily.
The assembly language code has been developed making use of sensibly named variables and subroutine names. This means that it is clear to understand the code and trace through the program. Therefore, this point can be marked as achieved.

As I have successfully completed all 8 points of my specification, this traffic light control system can be concluded as a success.

6.2 Further evaluation

Overall, I'm extremely pleased with the outcome of this project, especially with the timescale within which it was achieved in. Completing this project has taught me a lot about developing systems for microcontrollers and working with limited memory bit sizes.

6.3 Improvements

There are a number of improvements I would like to include in a future iteration of this project. I would like to have separated the pedestrian crossings out so rather than all the crossings going at the same time, the crossings would go when cars are unable to drive down the road which they cross. It would also be nice to lay out the project across a number of breadboards which would mimicked the physical layout of the crossing, as well as increase the number of sets of traffic lights for the same junction mimicking the fact that there might be more than one set of traffic lights for the same junction.

Appendix A

Full Code Listing

```
;*****************************************************************
;                               TEMPLATE PROVIDED BY CENTRE

;          TITLE:  #### TRAFFIC LIGHTS CONTROL SYSTEM #####
;          AUTHOR: ######## THOMAS BOXALL #########
;          DATE:   ####### 17-03-2022 ######
;

;*****************************************************************
; PROGRAM DESCRIPTION:
;
; ####  State here what the program does  #######
;
;*****************************************************************
;     DEFINITIONS
;*****************************************************************
    list    p=16F88               ; tells the assembler which PIC chip to
    ↪ program for
    radix dec                     ; set default number radix to decimal
    ;radix  hex                    uncomment this to  set  radix to hex
    __config h'2007', 0x3F50  ; internal oscillator, RA5 as i/o, wdt off
    __config h'2008', 0x3FFF
    errorlevel -302               ; hide page warnings

W             EQU h'00' ; pointer to Working register
F             EQU h'01' ; pointer to file

;****** REGISTER USAGE ******

;For PIC16F88, user RAM starts at h'20'. The following definitions
;will be found useful in many programs.

; Register page 1
TRISA EQU h'85' ; data direction registers
TRISB EQU h'86'
OSCCON  EQU h'8F' ; internal oscillator speed
ANSEL EQU h'9B' ; ADC port enable bits

; Register page 0
STATUS  EQU h'03'   ; status
PORTA EQU h'05' ; input / output ports
PORTB EQU h'06'
INTCON  EQU h'0B' ; interrupt control
ADRESH  EQU h'1E' ; ADC result
ADCON0  EQU h'1F' ; ADC control
```

```
45
46 B0   EQU h'20' ; general use byte registers B0 to B27
47 B1   EQU h'21'
48 B2   EQU h'22'
49 B3   EQU h'23'
50 B4   EQU h'24'
51 B5   EQU h'25'
52 B6   EQU h'26'
53 B7   EQU h'27'
54 B8   EQU h'28'
55 B9   EQU h'29'
56 B10  EQU h'2A'
57 B11  EQU h'2B'
58 B12  EQU h'2C'
59 B13  EQU h'2D'
60 B14  EQU h'2E'
61 B15  EQU h'2F'
62 B16  EQU h'30'
63 B17  EQU h'31'
64 B18  EQU h'32'
65 B19  EQU h'33'
66 B20  EQU h'34' ; used in interrupt routine
67 B21  EQU h'35' ; used in interrupt routine
68 B22  EQU h'36'
69 B23  EQU h'37'
70 B24  EQU h'38'
71 B25  EQU h'39'
72 B26  EQU h'3A'
73 B27  EQU h'3B'
74
75 WAIT1 EQU h'3C' ; counters used in wait delays
76 WAIT10  EQU h'3D'
77 WAIT100 EQU h'3E'
78 WAIT1000  EQU h'3F'
79 ADCTEMP EQU h'40' ; adc loop counter
80
81 ;my vars below
82 loopX EQU h'41' ;loop counter variable x
83 loopY EQU h'42' ;loop counter variable y
84
85 ;****** REGISTER BITS ******
86
87 C           EQU h'00'    ; carry flag
88 Z             EQU h'02' ; zero flag
89 RP0           EQU h'05' ; register page bit
90 INT0IF        EQU h'01' ; interrupt 0 flag
91 INT0IE       EQU h'04' ; interrupt 0 enable
92 GIE           EQU h'07' ; global interrupt enable
93
94 ;*****************************************************************
95 ;     VECTORS
96 ;*****************************************************************
97
98 ;The PIC16F88 reset vectors
99
100    ORG     h'00'          ; reset vector address
101  goto  start    ; goes to first instruction on reset/power-up
102    ORG     h'04'         ; interrupt vector address
103  goto  interrupt
104 ;
105 ;*****************************************************************
```

```
;     SUBROUTINES
;*****************************************************************
; Predefined wait subroutines - wait1ms, wait10ms, wait100ms, wait1000ms

wait1ms    ; (199 x 5) + 5 instructions = 1000us = 1ms @ 4MHz resonator
  movlw d'199'     ; 1
  movwf WAIT1    ; 1
loop5ns
  clrwdt           ; 1 this loop 1+1+1+2 = 5 instructions
  nop              ; 1
  decfsz WAIT1,F   ; 1
  goto loop5ns   ; 2
  nop              ; 1
  return             ; 2
wait10ms
  movlw d'10'        ; 10 x 1ms = 10ms
  movwf WAIT10
loop10ms
  call  wait1ms
  decfsz  WAIT10,F
  goto  loop10ms
  return

wait100ms
  movlw d'100'   ; 100 x 1ms = 100ms
  movwf WAIT100
loop100ms
  call  wait1ms
  decfsz  WAIT100,F
  goto  loop100ms
  return

wait1000ms
  movlw   d'10'      ; 10 x 100ms = 1000ms
  movwf   WAIT1000
loop1000ms
  call  wait100ms
  decfsz  WAIT1000,F
  goto  loop1000ms
  return

; MY SUBROUTINES BELOW

;wait five seconds and check the pedestian crossing while looping
;wait5SecondsButtonCheck
; ;setup times etc for loops
; movlw d'50' ;set val of 50 into working register
; movf loopX ;move 50 into loopX
; movlw d'100' ;set val of 100 into working regsiter
; movf loopY ;move 100 into loopY
;
; movf loopY, 0  ; move loopY into W register
; sublw d'100' ;subtract W register (loopY) from 100
; btfss STATUS, Z ;skip next line if the zero bit is set (finished the loop)
; goto contY ;goto the continue working on loopY


;contY  movf loopY, 0 ;moove loopY into W register


waitFiveSecondsCheckButton
```

```
167   ;first, stary loop x
168   movlw d'5' ;set val of 5 into w register
169   ;movf loopX ; move the val in the working register (5) into loopX
170
171 topX decfsz loopX, 1 ;decrement loopX and place the new value back into
      ↪ loopX
172   goto startY ;line skipped if result is 0
173   return ;if result of loopX - 1 is 0, run this line
174
175 startY movlw d'101'; ;set value of 101 into w register
176   ;movf loopY ;move value in working register (101) into loopY
177
178 topY decfsz loopY, 1 ;decrement loopY and place new value back into loopY
179   goto finalPart ;goto final part of the function
180   goto topX ;run if loopY = 0
181
182 finalPart call checkButton
183   call wait10ms ;wait for 10ms function
184   goto topY ;go back up to topY and loop around again
185 ;end of function
186
187
188 waitOneSecondCheckButton
189   movlw d'101' ;set value of 101 into w register
190   movf loopY ;move val from working register into loopY
191 topZ decfsz loopY, 1 ;decrement loopY and place new value back into loopY
192   goto finalPart2 ;goto final part of the function (if loopyY>0)
193   return ; go back to where called from if loopY=0
194 finalPart2 call checkButton
195   call wait10ms ;wait for 10ms function
196   goto topZ ;go back up to the decrement of loopY
197 ; end of function
198
199
200 ;function to check the button
201 checkButton
202   btfss PORTB, 1 ;skip next line of code if button pressed (active high)
203   return ;can go back to where called from
204   bsf PORTB, 2 ;turn white wait LED on
205   return ;go back to main code
206 ;end of function
207
208
209 ; function to cycle the pedestrian crossing
210 cycleCrossing
211   bcf PORTB, 2 ;turn off red wait LED
212   bcf PORTB, 3 ;turn off red stop LED
213   bsf PORTB, 4 ;turn on green go LED
214   bsf PORTB, 5 ;turn on buzzer
215   call wait1000ms ;wait 1second
216   call wait1000ms ;wait 1second
217   call wait1000ms ;wait 1second
218   ;now have waited 3seconds so invert things and return to main sequence
219   bcf PORTB, 4 ;turn off green go LED
220   bsf PORTB, 3 ;turn on red stop LED
221   bcf PORTB, 5 ;turn off buzzer
222   return ;return to the main program
223 ;end of function
224
225 greenTrafficLightSequence
226   bsf PORTA, 4 ;turn on amber led
```

```
227   call waitOneSecondCheckButton ;wait 5s
228   call waitOneSecondCheckButton
229   call waitOneSecondCheckButton
230   call waitOneSecondCheckButton
231   call waitOneSecondCheckButton
232   bcf PORTA, 4 ;turn off amber led
233   bcf PORTA, 3 ;turn off red led
234   bsf PORTB, 7 ;turn on greeen led
235   call waitOneSecondCheckButton ;wait 10s
236   call waitOneSecondCheckButton
237   call waitOneSecondCheckButton
238   call waitOneSecondCheckButton
239   call waitOneSecondCheckButton
240   call waitOneSecondCheckButton
241   call waitOneSecondCheckButton
242   call waitOneSecondCheckButton
243   call waitOneSecondCheckButton
244   call waitOneSecondCheckButton
245   bcf PORTB, 7 ;turn off green led
246   bsf PORTA, 4 ;turn on amber led
247   call waitOneSecondCheckButton ;wait 5s
248   call waitOneSecondCheckButton
249   call waitOneSecondCheckButton
250   call waitOneSecondCheckButton
251   call waitOneSecondCheckButton
252   bsf PORTA, 3 ;turn on red led
253   bcf PORTA, 4 ;turn off amber led
254   return ;go back to main code
255 ;end of function
256
257 pinkTrafficLightSequence
258   bsf PORTA, 1 ;turn on amber led
259   call waitOneSecondCheckButton ;wait 5s
260   call waitOneSecondCheckButton
261   call waitOneSecondCheckButton
262   call waitOneSecondCheckButton
263   call waitOneSecondCheckButton
264   bcf PORTA, 1 ;turn off amber led
265   bcf PORTA, 0 ;turn off red led
266   bsf PORTA, 2 ; turn on greeen led
267   call waitOneSecondCheckButton ;wait 10s
268   call waitOneSecondCheckButton
269   call waitOneSecondCheckButton
270   call waitOneSecondCheckButton
271   call waitOneSecondCheckButton
272   call waitOneSecondCheckButton
273   call waitOneSecondCheckButton
274   call waitOneSecondCheckButton
275   call waitOneSecondCheckButton
276   call waitOneSecondCheckButton
277   bcf PORTA, 2 ;turn off green led
278   bsf PORTA, 1 ;turn on amber led
279   call waitOneSecondCheckButton ;wait 5s
280   call waitOneSecondCheckButton
281   call waitOneSecondCheckButton
282   call waitOneSecondCheckButton
283   call waitOneSecondCheckButton
284   bsf PORTA, 0 ;turn on red led
285   bcf PORTA, 1 ;turn off amber led
286   return ;go back to main code
287 ;end of function
```

```
288
289 blueTrafficLightSequence
290   bsf PORTA, 7 ;turn on amber led
291   call waitOneSecondCheckButton ;wait 5s
292   call waitOneSecondCheckButton
293   call waitOneSecondCheckButton
294   call waitOneSecondCheckButton
295   call waitOneSecondCheckButton
296   bcf PORTA, 7 ;turn off amber led
297   bcf PORTA, 6 ;turn off red led
298   bsf PORTB, 0 ; turn on greeen led
299   call waitOneSecondCheckButton ;wait 10s
300   call waitOneSecondCheckButton
301   call waitOneSecondCheckButton
302   call waitOneSecondCheckButton
303   call waitOneSecondCheckButton
304   call waitOneSecondCheckButton
305   call waitOneSecondCheckButton
306   call waitOneSecondCheckButton
307   call waitOneSecondCheckButton
308   call waitOneSecondCheckButton
309   bcf PORTB, 0 ;turn off green led
310   bsf PORTA, 7 ;turn on amber led
311   call waitOneSecondCheckButton ;wait 5s
312   call waitOneSecondCheckButton
313   call waitOneSecondCheckButton
314   call waitOneSecondCheckButton
315   call waitOneSecondCheckButton
316   bsf PORTA, 6 ;turn on red led
317   bcf PORTA, 7 ;turn off amber led
318   return ;go back to main code
319 ;end of function
320
321 checkPedestrian
322   ;function to check if the wait led is on
323     ;if yes - cycle pedestrian crossing
324     ;if no - return to main
325   btfsc PORTB, 2 ;skip next line if wait led NOT on therefore if on, run
      ↪ next line
326   call cycleCrossing
327   return
328 ;end function
329
330 ; Predefined ADC subroutines - readadc0, readac1, readadc2
331
332 readadc0
333   movlw b'00000001' ; setup mask for pin A.0
334   call  readadc ; do the adc conversion
335   movwf B0          ; save result in B0
336   return
337 readadc1
338   movlw b'00000010' ; setup mask for pin A.1
339   call  readadc ; do the adc conversion
340   movwf B1            ; save result in B1
341   return
342 readadc2
343   movlw b'00000100'   ; setup mask for pin A.2
344   call  readadc ; do the adc conversion
345   movwf B2            ; save result in B2
346   return
347
```

```
348 readadc
349 ; Generic sub routine to read ADC 0, 1 or 2 (pass appropriate mask in W)
350 ; To start conversion we need mask (001, 010, 100) in ANSEL bits 0-2
351 ; but the actual channel number (0, 1, 2) in ADCON0 channel select bits
352 ; Then set the ADCON0, GO bit to start the conversion
353
354   bsf        STATUS,RP0  ; select register page 1
355   movwf ANSEL    ; move mask value 001,010,100 into ANSEL
356   bcf        STATUS,RP0  ; select register page 0
357   movwf ADCTEMP ; 00000???  get mask value
358   rlf        ADCTEMP,F ; 0000???x  rotate twice
359   rlf        ADCTEMP,W ; 000???xx
360   andlw b'00011000' ; 000??000  mask off the unwanted bits
361   iorlw b'00000001' ; 000??001  set the 'ADC on' bit
362   movwf ADCON0  ; move working into ADCON0
363       movlw     d'10'        ; 10 x 3 = 30us acquistion time
364       movwf    ADCTEMP      ; re-use ADC1 register as a counter
365 loopacq
366       decfsz     ADCTEMP,F    ; loop around to create short delay
367       goto       loopacq      ; each loop is 1+2 = 3 instructions = 3us @ 4MHz
368   bsf        ADCON0,2  ; now start the conversion
369 loopadc
370   clrwdt           ; pat the watchdog
371   btfsc ADCON0,2  ; is conversion finished?
372   goto  loopadc ; no, so wait a bit more
373   movf  ADRESH,W  ; move result into W
374   return              ; return with result in W
375
376 ; NOTE for PICAXE users:  the following four specific subroutines and two
    ↪ instructions are not supported by PICAXE compiler
377 readtemp1:
378 readtemp2:
379 readtemp3:
380 debug:
381 lcd:
382   clrw                  ; instruction not supported by this template
383   return      ; instruction not supported by this template
384 ;*****************************************************************
385 ;     MAIN PROGRAM
386 ;*****************************************************************
387
388 ;****** INITIALISATION ******
389 start
390   bsf        STATUS,RP0  ; select register page 1
391   movlw   b'01100000' ; set to 4MHz internal operation
392   movwf OSCCON
393   clrf  ANSEL    ; disable ADC (enabled at power-up)
394   bcf        STATUS,RP0  ; select register page 0
395
396 ;The data direction registers TRISA and TRISB live in the special register
    ↪ set. A '1' in
397 ;these registers sets the corresponding port line to an Input, and a
398 ;'0' makes the corresponding line an output.
399
400 Init
401   clrf       PORTA        ; make sure PORTA output latches are low
402       clrf       PORTB        ; make sure PORTB output latches are low
403   bsf        STATUS,RP0  ; select register page 1
404 ;Modify the next line to correspond with your input output reqirements
405   movlw b'00000000' ; set port A data direction (0 = output bit, 1 = input
    ↪ bit)
```

```
406   movwf TRISA    ;
407 ;Modify the next line to correspond with your input output requirements
408   movlw b'00000010' ; set port B data direction (0 = output bit, 1 = input
        ↪ bit)
409   movwf TRISB    ;
410   bcf        STATUS,RP0   ; select register page 0
411
412 ;****** MAIN PROGRAM ******
413 ;************* remove semicolons from next two lines to enable interrupt
        ↪ routine************
414     ; bsf      INTCON,INT0IE   ; set external interrupt enable
415     ; bsf      INTCON,GIE        ; enable all interrupts
416
417 main
418 ;*****************************************************************
419
420 ; YOUR PROGRAM GOES HERE ######################
421
422   ;first turn on all stop leds
423   bsf PORTA, 0 ;turn on pink set stop eld
424   bsf PORTA, 3 ;turn on green set stop led
425   bsf PORTA, 6 ;turn on blue set stop led
426   bsf PORTB, 3 ;turn on pedestrian stop led
427
428 mTop call pinkTrafficLightSequence
429   call checkPedestrian
430   call greenTrafficLightSequence
431   call checkPedestrian
432   call blueTrafficLightSequence
433   call checkPedestrian
434   goto mTop
435
436 ;*****************************************************************
437 ;      INTERRUPT SERVICE ROUTINE
438 ;*****************************************************************
439
440 W_SAVE  EQU B20          ; backup registers used in interrupts
441
442 interrupt
443   movwf    W_SAVE  ; Copy W to save register
444
445       btfss    INTCON,INT0IF  ; check correct interrupt has occurred
446       retfie                    ; no, so return and re-enable GIE
447
448 ;**********The interrupt service routine (if required) goes here*********
449
450     bcf     INTCON,INT0IF     ; clear interrupt flag
451   movf    W_SAVE,W          ; restore W
452   retfie                  ; return and re-set GIE bit
453
454       END                     ; all programs must end with this
```

Listing A.1: Full code listing

Appendix B

Full Schematic

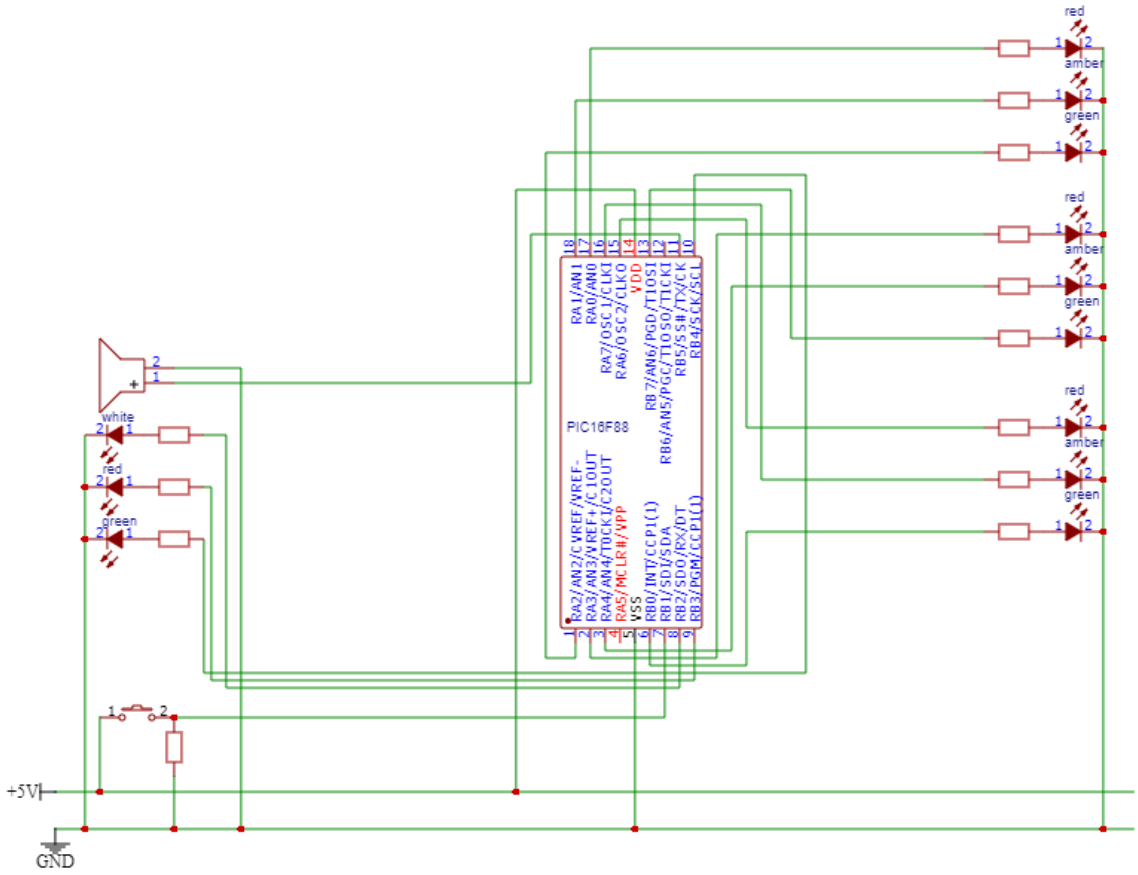

Figure B.1: Schematic of the circuit

Appendix C

Final Physical Layout

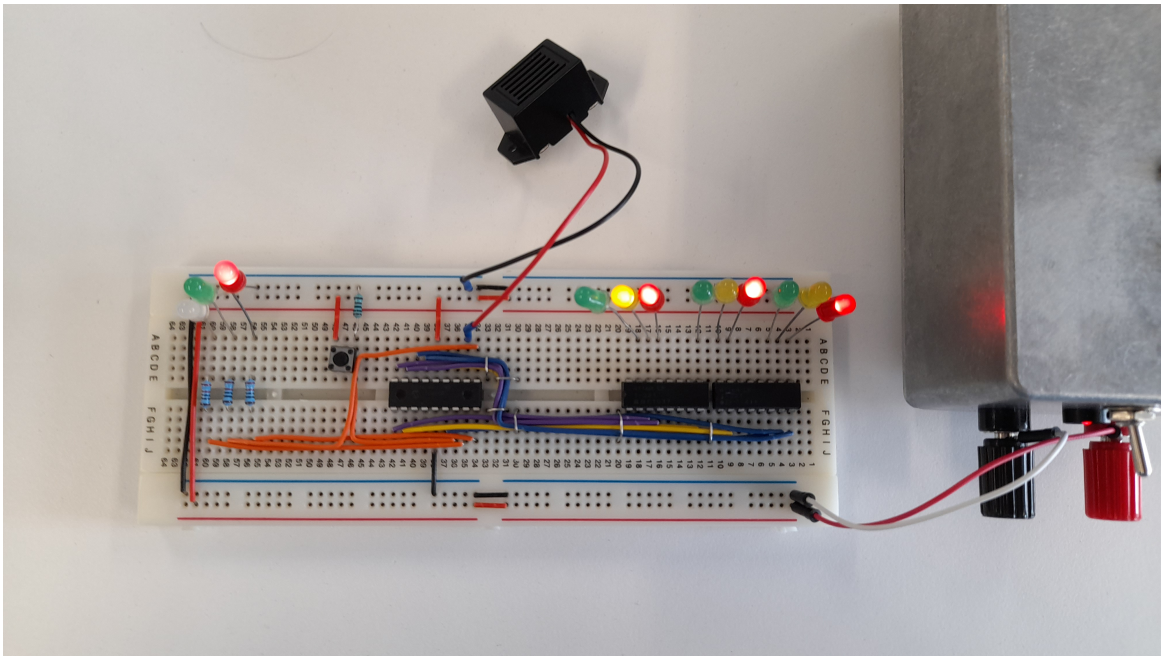

Figure C.1: Final physical layout