

Chris James Electronics 1

by Chris James

Submission date: 14-Apr-2024 08:38AM (UTC+0100)

Submission ID: 229276161

File name: 50113_Chris_James_Chris_James_Electronics_1_568563_1730091626.docx (4.91M)

Word count: 2820

Character count: 14061

Electronics 1 – Traffic lights

Introduction

The aim of my project was to create a traffic light system that would safely allow cars to pass through without the risk of collision. While any of the lights are green to allow cars to pass, the pedestrian crossing light will remain red. When all lights are red, the pedestrian crossing will turn green for a period of time.

The circuit would take 50 seconds to complete overall, with the final 10 seconds reserved for the pedestrian crossing. The whole crossing system will be timed using a 7-segment display that will count up from 0 – 50 seconds before resetting to zero.

Software

To replicate the traffic light system, I created a code using mikroC Pro. The circuit was created using proteus eight.

There were many benefits to using software instead of creating a real prototype **(components, 2018)**:

- Cost – Using software is usually much cheaper than creating a prototype. It does not require spending money on materials or manufacturing.
- Time – I am able to make changes to software much quicker than if I was creating a real prototype. This also allows me to quickly test and make changes to my circuit.
- Testing – I can test my design and find all faults and errors before creating a final design. This will allow me to quickly see how my coding performs without wasting time and equipment.
- Accessibility – I am able to easily carry my software to various locations without damaging it. I can also share it to multiple devices without any requirement to build multiple prototypes.

When creating the code for the seven-segment display, I chose to use an up counter instead of a down counter. This is because a down counter puts limits on the amount of time I can place in my design. Using an up counter, if I wish to add more time then I can simply add it on to the end of my code however, If I were to use a down counter, then I would have to re-write the code every time I wished to expand/add more time to my design. An up-counter will allow an infinite number of expansions to the design.

I decided to use a PIC16F877A. This microprocessor was suitable for my requirements for multiple reasons. This chip is relatively cheap, costing approximately £5 in most retailers **(RS, 2023)**. The PIC168f77a has a suitable operating temperature range (-40° to +85°) and has a suitable operating voltage for

my circuit (4-5.5V) (Raj, 2018). The microcontroller also has more than enough pins for my design, with more to upgrade in future if required.

Background Research

To decide on how the sequence that I would use for my traffic light design, I looked at some real-life scenarios. This would help to keep my circuit as realistic as possible. Many different countries use different sequences however I decided to use the official traffic light sequence used in the UK (BSM, 2017) this is as follows:

Traffic light 1		Traffic light 2	
Crossing			
• Red and amber	Red	Red	
• Green	Red	Red	
• Amber	Red	Red	
• Red	Red and amber	Red	
• Red	Green	Red	
• Red	Amber	Red	
• Red	Red	Green	

This shows that when one set of lights are either amber or green, the other will stay red. This will prevent any collisions. The Crossing will stay red until both sets of lights are red, when it will then change to green to allow pedestrians to cross.

To ensure a full understanding of my design, it is important to realise the function of each light:

- Red – When the red light is displayed, cars in this direction should all be at a stop. No cars or pedestrians are permitted to move/cross.
- Amber – This means that the lights are about to change to either red or green. You must pass through an amber light only, when necessary, otherwise you should be at a stop, aware of the forthcoming change of signal.
- Green – You may advance as long as the way in front is clear. Other lights that may pose an obstruction should be on red.
- Pedestrian crossing – These lights should stay red unless all other lights are also red, when safe to do so, they will then change to green so acknowledge that it is safe for pedestrians to cross.

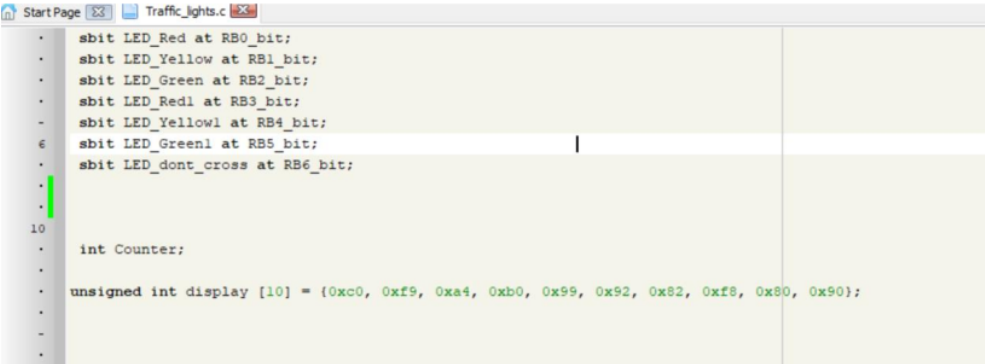
Circuit design

Using Mikroc Pro, I created a code that would allow me to create the traffic light sequence that would safely allow cars to flow without collision. It would also ensure that pedestrians do not cross until ALL the traffic lights have turned red. Once the coding was complete, I was then able to upload the complete file to a microprocessor in proteus that would then allow me to create my project.

My design would change lights in specific time intervals. The intervals that I chose were:

Time (s)	Traffic light 1	Traffic light 2	Pedestrian crossing
0-4	Red	Red/Amber	Red
5-9	Red	Green	Red
10-14	Red	Green	Red
15-19	Red	Amber	Red
20-24	Red/Amber	Red	Red
25-29	Green	Red	Red
30-34	Green	Red	Red
35-39	Amber	Red	Red
40-49	Red	Red	Green
50	Red	Red	Green

Figure 1

A screenshot of a code editor window titled 'Traffic_lights.c'. The code defines several bit variables for LEDs and a counter. The variables are: LED_Red at RB0_bit, LED_Yellow at RB1_bit, LED_Green at RB2_bit, LED_Red1 at RB3_bit, LED_Yellow1 at RB4_bit, LED_Green1 at RB5_bit, and LED_dont_cross at RB6_bit. A counter variable 'int Counter;' is also defined. At the bottom, an array 'display' is initialized with hexadecimal values: {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};. The code is written in a light-colored font on a dark background.

```
• sbit LED_Red at RB0_bit;
• sbit LED_Yellow at RB1_bit;
• sbit LED_Green at RB2_bit;
• sbit LED_Red1 at RB3_bit;
• sbit LED_Yellow1 at RB4_bit;
6 sbit LED_Green1 at RB5_bit;
• sbit LED_dont_cross at RB6_bit;
•
•
10
• int Counter;
•
• unsigned int display [10] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};
•
•
```

Figure 1 shows the beginning of my coding. Before writing the code that would create my traffic light system, I renamed the pins of the microprocessor. This would make the code more readable.

e.g. – instead of writing RBO_bit every time I referenced pin B0, the above code would allow me to type ‘LED_Red’. This would prevent a large amount of confusion

for people who are reading the code for the first time. It would also allow the code easier to create as I would not have to keep checking what light/function each pin carried out.

Figure 2

```
unsigned int display [10] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};
```

Figure 2 shows a hexadecimal code. It is used for the seven segment displays that would time the changes of the lighting sequence. Each hexadecimal value that I used for the seven segment displays was found using the seven-segment display editor toward the top of the screen (as shown in figure 3). Each value would represent a different value on the display. Timers in the coding would ensure that these whanged in one second increments.

When each hexadecimal code is written into the program, it will display a different value onto the displays. These are as follows:

0- 0xc0	5 - 0x92
1- 0xf9	6 - 0x82
2- 0xa4	7 – 0xf8
3- 0xb0	8 – 0x80
4- 0x99	9 – 0x90

The 'unsigned int' determines the data types of each element in the array. It shows that the values will be integer (numeric).

'Display' Is the name that I have assigned to the integers.

[10] shows that there will be ten elements within the array (these are the hexadecimal values show).

Figure 3

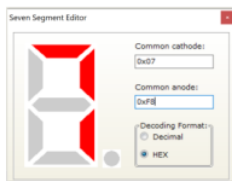


Figure 4

```
void main()
{
    TrisB=0x00;
    PortB=0x00;
    trisc =0x00;
    portc =0x00;
    trisd =0x00;
    portd =0x00;

    delay_ms(100);
}
```

Within the main program, I assigned the parameters for the ports that I will be using (shown in figure 4).

For example, it sets values such as TrisB = 0x00 and PortB = 0x00.

This determines the functionality of these pins at the beginning of the program. PORT(Pin) determines whether it is either an input or output while PORT(Pin) determines whether it will begin switched high or low. Since no inputs are used in this program, each port used was set as an output (0x00), Each port was switched to low to begin the program.

Figure 5

```
do
{
    // Traffic lights 0-5 seconds

    for (Counter=0;Counter<5;Counter++)
    {
        Portc = 0xc0; //0 seconds
        Portd = display [Counter];

        // Traffic light 1
        LED_Red = 1;
        LED_Yellow = 0;
        LED_Green = 0;

        // Traffic light 2
        LED_Red1 = 1;
        LED_Yellow1 = 1;
        LED_Green1 = 0;

        //Crossing lights
        LED_dont_cross = 1;

        Delay_ms(1000);
    }
}
```

The '/' in the code indicated that the following text is just a description and has no impact in the code itself, it is just used to help to make the coding easier to understand.

Figure 5 shows the first sequence of the traffic light system. As the first set of lights stay at red, the second set of lights have begun their sequence of changing to Red/Amber. The '1' value in the code indicated that the pin has been changed to a high state. This allows an output on the microprocessor.

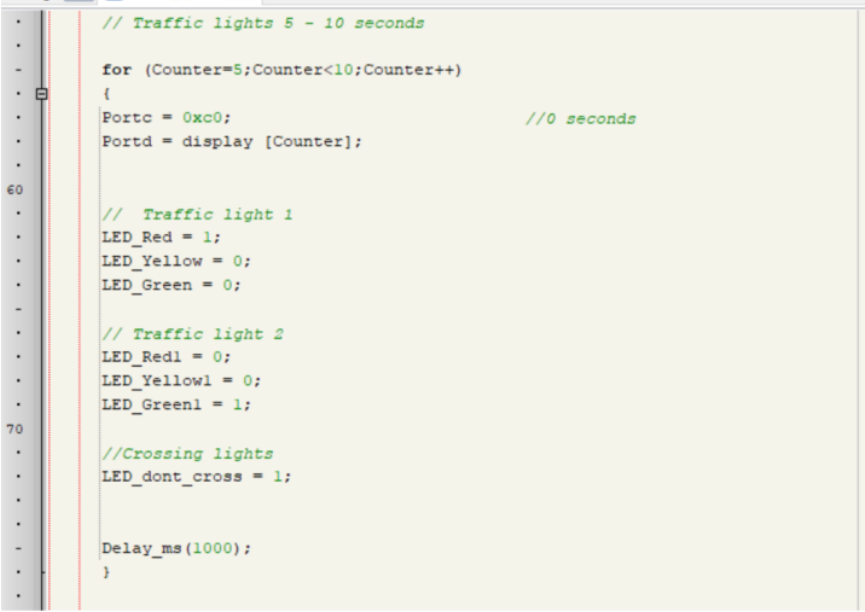
During this time, the Led_dont_cross remains high, preventing pedestrians from crossing.

The delay of 1000ms (1 second) indicated that this program will remain in this state for one second before moving to the next section.

Figure 5 also shows that while the counter is below 5, it will remain in this state, re-running this part of the program, incrementing the value on the seven-segment display by 1 each time (this acts as the counter for the traffic light sequence). Once the counter reaches ≥ 5 , it will then move onto the next part of the sequence.

In this sequence, port c represents the 'tens' counter – currently set to display zero, while port d represents the 'units' counter, incrementing by one every second.

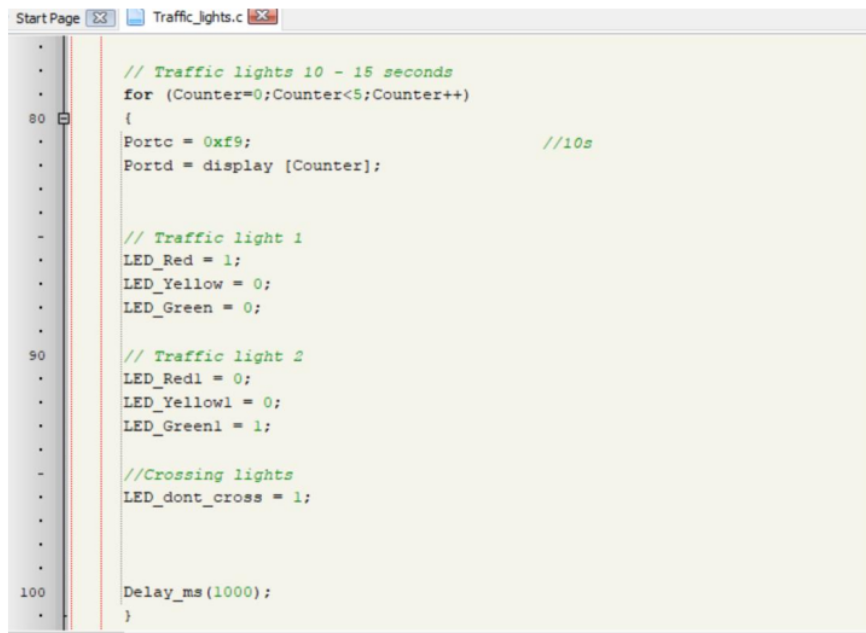
Figure 6



```
• // Traffic lights 5 - 10 seconds
•
• for (Counter=5;Counter<10;Counter++)
• {
•   Portc = 0xc0; //0 seconds
•   Portd = display [Counter];
•
•   // Traffic light 1
•   LED_Red = 1;
•   LED_Yellow = 0;
•   LED_Green = 0;
•
•   // Traffic light 2
•   LED_Red1 = 0;
•   LED_Yellow1 = 0;
•   LED_Green1 = 1;
•
•   //Crossing lights
•   LED_dont_cross = 1;
•
•   Delay_ms(1000);
• }
```

Figure 6 shows the values at 5-10 seconds. PortC is still representing 0 seconds while port d is incrementing, this time through hexadecimal values of 5 – 9. During this time, the pins are now showing that traffic light 1 is still red high while traffic light 2 has changed to green, allowing cars to pass through.

Figure 7



```
Start Page Traffic_lights.c
// Traffic lights 10 - 15 seconds
for (Counter=0;Counter<5;Counter++)
{
    Portc = 0xf9; //10s
    Portd = display [Counter];

    // Traffic light 1
    LED_Red = 1;
    LED_Yellow = 0;
    LED_Green = 0;

    // Traffic light 2
    LED_Red1 = 0;
    LED_Yellow1 = 0;
    LED_Green1 = 1;

    //Crossing lights
    LED_dont_cross = 1;

    Delay_ms(1000);
}
```

Once the counter in figure 6 is now >9, the coding moves on to figure 7:

PortC (tens) – showing value 1 (high)

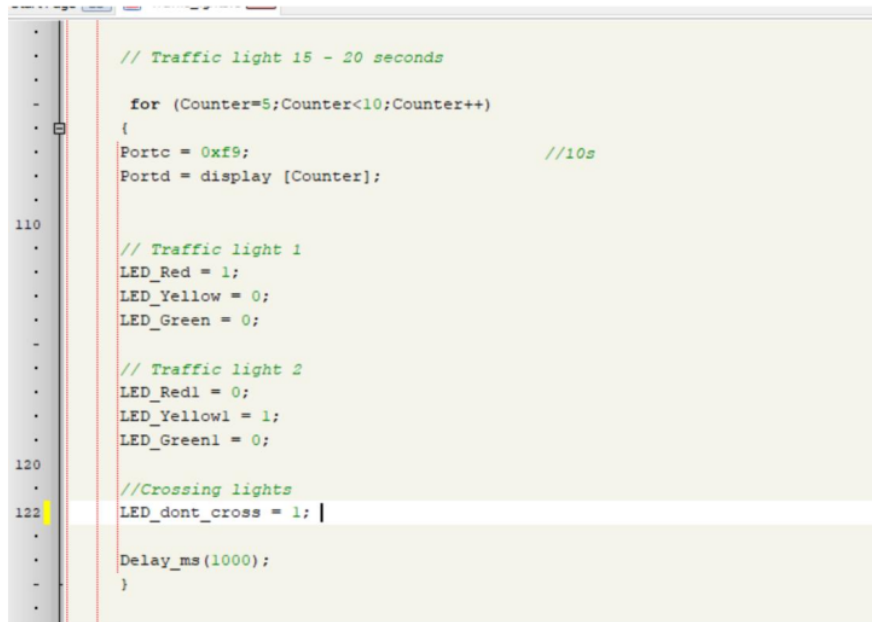
PortD (Units) – incrementing 0-4.

Traffic light 1 – red

Traffic light 2 – green

Pedestrian crossing – red

Figure 8

A screenshot of a code editor window showing C code. The code is organized into sections with comments. The first section, starting at line 110, is a loop for 'Traffic light 15 - 20 seconds' that runs from Counter=5 to Counter<10. Inside the loop, Portc is set to 0xf9 (commented as //10s) and Portd is set to display [Counter]. The second section, starting at line 120, is for 'Traffic light 1', setting LED_Red to 1, LED_Yellow to 0, and LED_Green to 0. The third section, starting at line 122, is for 'Traffic light 2', setting LED_Red1 to 0, LED_Yellow1 to 1, and LED_Green1 to 0. The final section, starting at line 122, is for 'Crossing lights', setting LED_dont_cross to 1, followed by a Delay_ms(1000) and a closing brace. Line numbers 110, 120, and 122 are visible on the left margin.

```
110 // Traffic light 15 - 20 seconds
111 for (Counter=5;Counter<10;Counter++)
112 {
113     Portc = 0xf9; //10s
114     Portd = display [Counter];
115 }
116
117 // Traffic light 1
118 LED_Red = 1;
119 LED_Yellow = 0;
120 LED_Green = 0;
121
122 // Traffic light 2
123 LED_Red1 = 0;
124 LED_Yellow1 = 1;
125 LED_Green1 = 0;
126
127 //Crossing lights
128 LED_dont_cross = 1; |
129
130 Delay_ms(1000);
131 }
```

Once the counter in figure 7 is now ≥ 5 , the coding moves on to figure 8:

PortC (tens) – showing value 1 (high)

PortD (Units) – incrementing 5-9.

Traffic light 1 – red

Traffic light 2 – amber

Pedestrian crossing – red

Delay of 1000ms to ensure a one second processing time (display counter).

Figure 9



```
Start Page Traffic_lights.c
// Traffic light 20 - 25 seconds
for (Counter=0;Counter<5;Counter++)
{
130 Portc = 0xa4; //20s
    Portd = display [Counter];

    // Traffic light 1
    LED_Red = 1;
    LED_Yellow = 1;
    LED_Green = 0;

140 // Traffic light 2
    LED_Red1 = 1;
    LED_Yellow1 = 0;
    LED_Green1 = 0;

    //Crossing lights
    LED_dont_cross = 1;

148 Delay_ms(1000);
150 }
```

Code for incrementing 20-25 seconds (Figure 9)

PortC (tens) – showing value 2 (high)

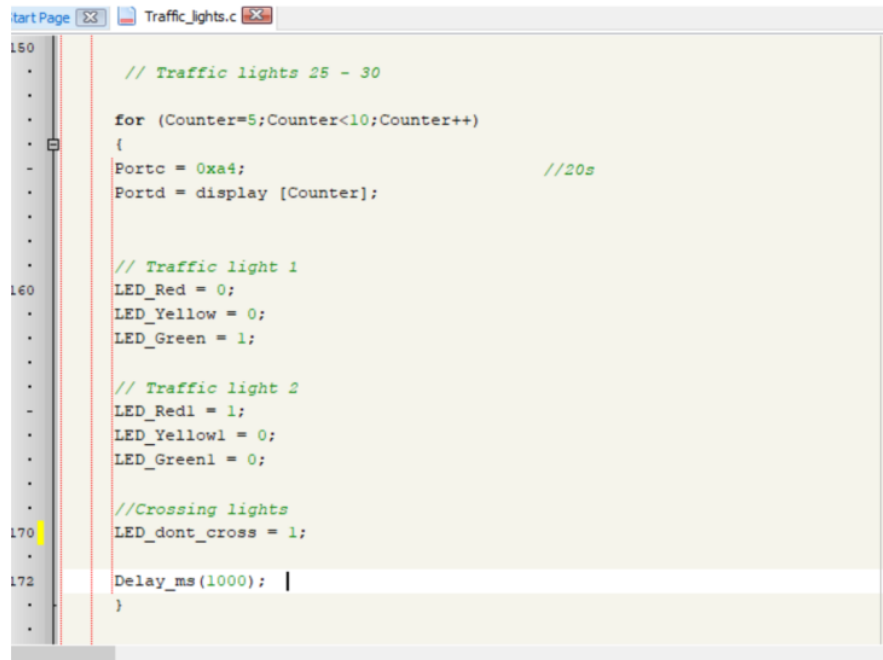
PortD (Units) – incrementing 0-4.

Traffic light 1 – red/amber

Traffic light 2 – red

Pedestrian crossing – red

Figure 10



```
150      // Traffic lights 25 - 30
      for (Counter=5;Counter<10;Counter++)
      {
160      Portc = 0xa4;           //20s
      Portd = display [Counter];

      // Traffic light 1
      LED_Red = 0;
      LED_Yellow = 0;
      LED_Green = 1;

      // Traffic light 2
      LED_Red1 = 1;
      LED_Yellow1 = 0;
      LED_Green1 = 0;

      //Crossing lights
170      LED_dont_cross = 1;

172      Delay_ms(1000); |
      }
```

Code for incrementing 25-30 seconds (Figure 10)

PortC (tens) – showing value 2 (0xa4)

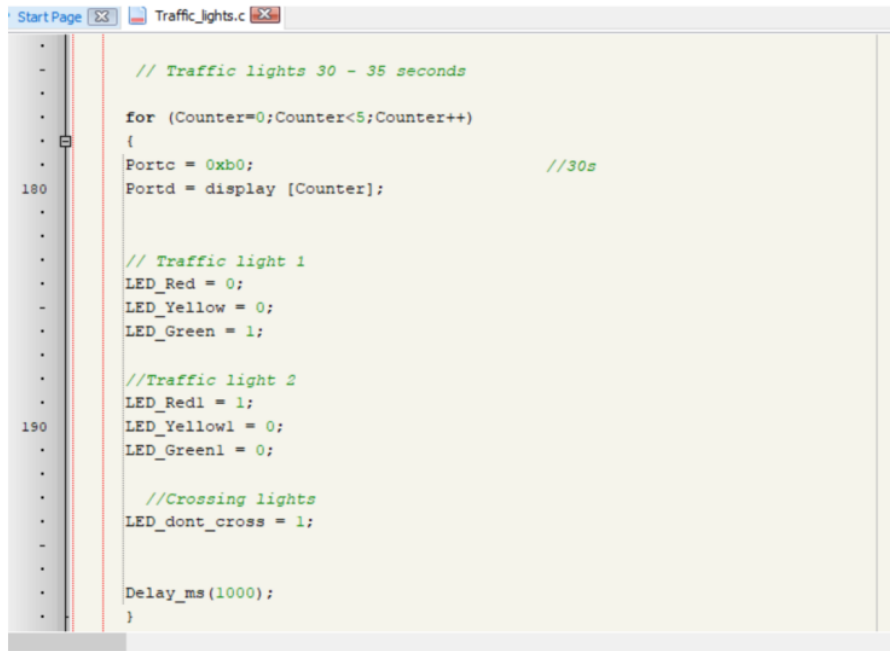
PortD (Units) – incrementing 5-9.

Traffic light 1 – Green

Traffic light 2 – red

Pedestrian crossing – red

Figure 11



```
Start Page Traffic_lights.c
// Traffic lights 30 - 35 seconds
for (Counter=0;Counter<5;Counter++)
{
    Portc = 0xb0; //30s
    Portd = display [Counter];

    // Traffic light 1
    LED_Red = 0;
    LED_Yellow = 0;
    LED_Green = 1;

    //Traffic light 2
    LED_Red1 = 1;
    LED_Yellow1 = 0;
    LED_Green1 = 0;

    //Crossing lights
    LED_dont_cross = 1;

    Delay_ms(1000);
}
```

Code for incrementing 30-35 seconds (Figure 11)

PortC (display 1 - tens) – showing value 3 (0xb0)

PortD (display 2 - Units) – incrementing 0-4.

Traffic light 1 – Green

Traffic light 2 – red

Pedestrian crossing – red

Figure 12



```
200 // Traffic lights 35 - 40 seconds
.
.
. for (Counter=5;Counter<10;Counter++)
. {
. Portc = 0xb0; //30s
. Portd = display [Counter];
.
. // Traffic light 1
. LED_Red = 0;
. LED_Yellow = 1;
210 LED_Green = 0;
.
. //Traffic light 2
. LED_Red1 = 1;
. LED_Yellow1 = 0;
. LED_Green1 = 0;
.
. //Crossing lights
. LED_dont_cross = 1;
.
220
221 Delay_ms(1000);
.
. }
```

Code for incrementing 35-40 seconds (Figure 12)

PortC (display 1 - tens) – showing value 3 (0xb0)

PortD (display 2 - Units) – incrementing 5-9.

Traffic light 1 – amber

Traffic light 2 – red

Pedestrian crossing – red

Figure 13



```
Start Page Traffic_lights.c
// Traffic lights 40 - 49 seconds
for (Counter=0;Counter<10;Counter++)
{
    Portc = 0x99; //40s
    Portd = display [Counter];

    // Traffic light 1
    LED_Red = 1;
    LED_Yellow = 0;
    LED_Green = 0;

    // Traffic light 2
    LED_Red1 = 1;
    LED_Yellow1 = 0;
    LED_Green1 = 0;

    //Crossing lights
    LED_dont_cross = 0;

    Delay_ms(1000);
}
```

Now that the code shows that both traffic lights are red, the pedestrian crossing has now changed to green. This is achieved by changing the LED_dont_cross to low (0). I have connected the same pin in proteus using a 'NOT' gate (Seen in figure 14), this will switch the signal from a red light to a green light.

Code for incrementing 40-49 seconds (Figure 13)

PortC (display 1 - tens) – showing value 4 (0x99)

PortD (display 2 - Units) – incrementing 0-9.

Traffic light 1 – red

Traffic light 2 – red

Pedestrian crossing – Green

Figure 14

```
250      // Traffic lights 50 seconds
      for (Counter=0;Counter<1;Counter++)
      {
        Portc = 0x92;           //50s
        Portd = display [Counter];

        // Traffic light 1
        LED_Red = 1;
        LED_Yellow = 0;
        LED_Green = 0;

        // Traffic light 2
        LED_Red1 = 1;
        LED_Yellow1 = 0;
        LED_Green1 = 0;

        //Crossing lights
        LED_dont_cross = 0;

        Delay_ms(1000);
      }

      while(1);
278 }
```

This final part of the code allows the counter to display the final '50 second'. Once this has been displayed, it will then return to the beginning of the code and repeat the sequence until stopped.

Code for incrementing 50 seconds (Figure 14)

PortC (display 1 - tens) – showing value 4 (0x99)

PortD (display 2 - Units) – Only incrementing from zero to less than 1.

Traffic light 1 – red

Traffic light 2 – red

Pedestrian crossing – Green

Proteus design

Figure 15 - 0-5 seconds.

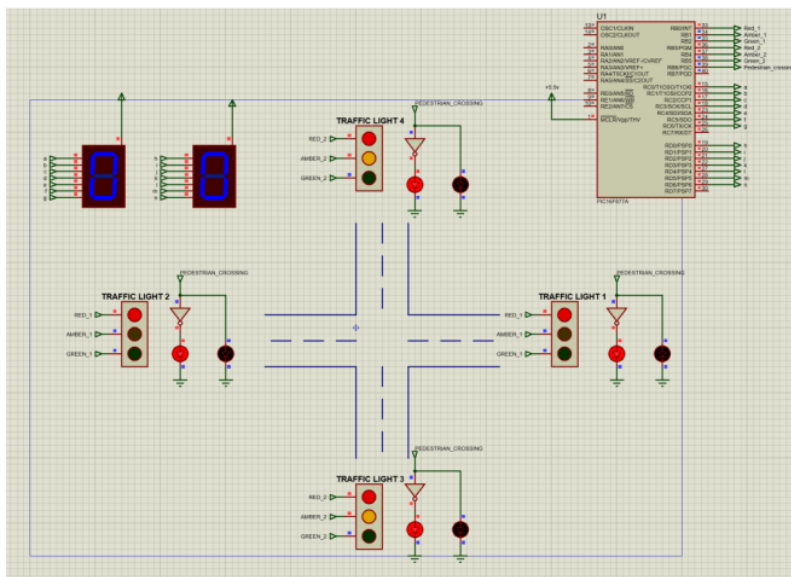


Figure 20 - 25-30 seconds.

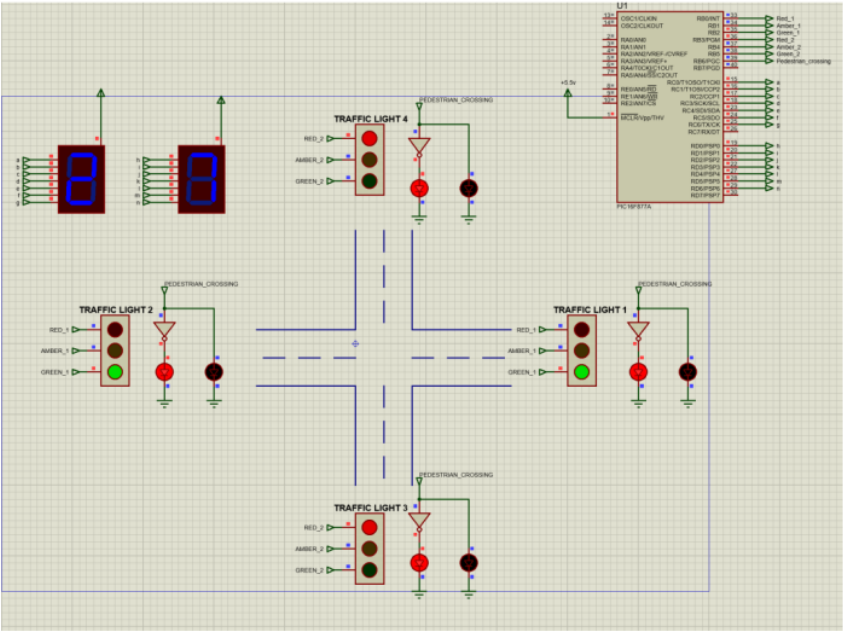


Figure 21 - 30-35 seconds.

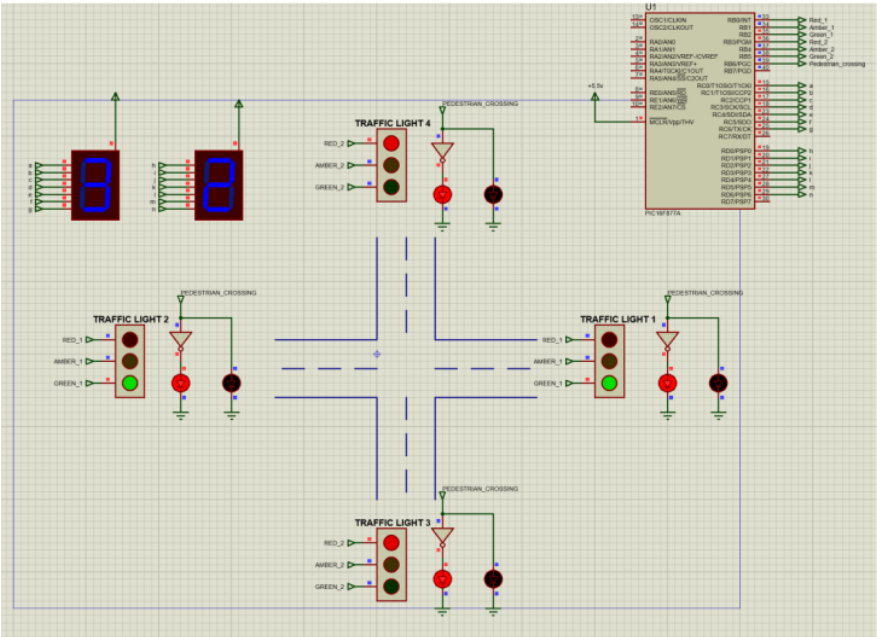


Figure 22 - 35-40 seconds.

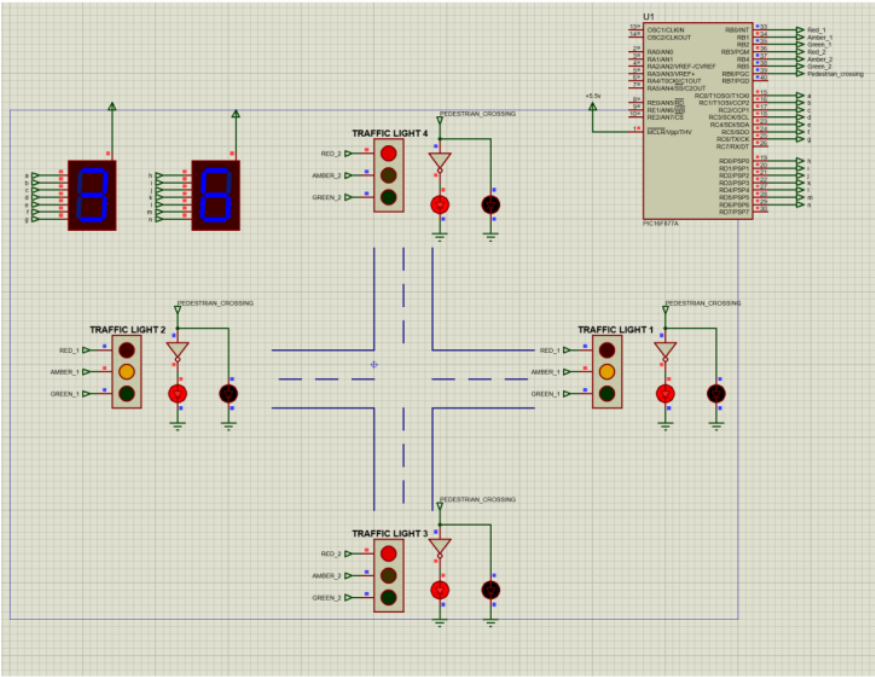


Figure 23 - 40-49 seconds.

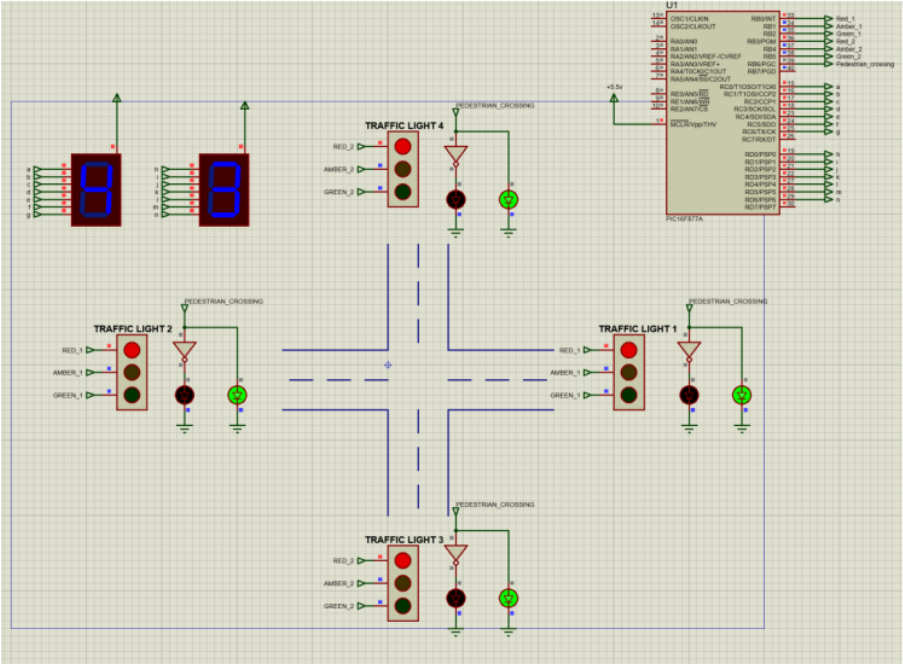


Figure 24 - 50 seconds.

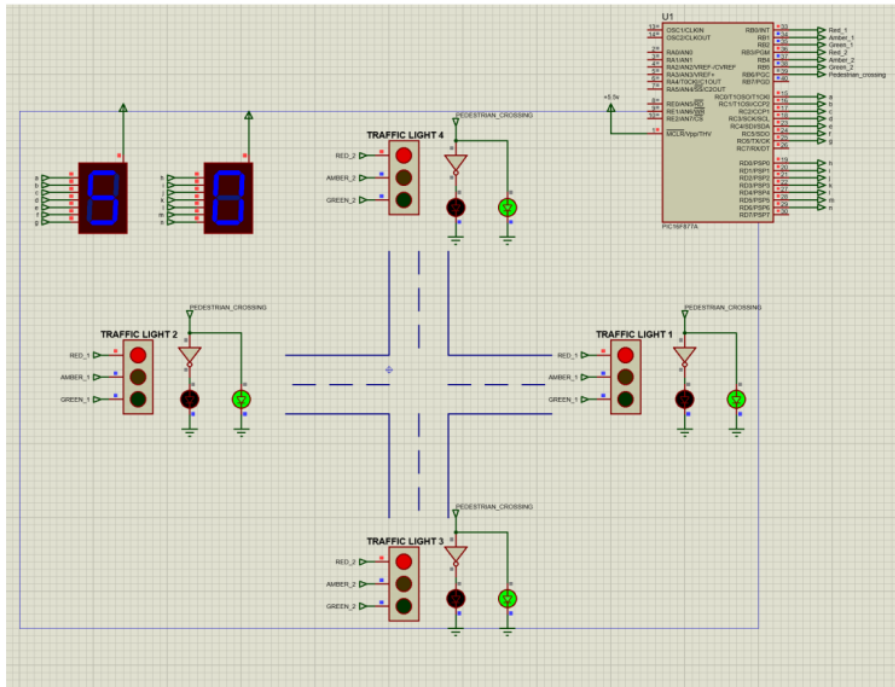


Figure 25 shows the PIC16f877a that I used in the project. The output tags from portB were identified as red_1, Amber_2 etc, to make them easier to find in the project and overall easier for a person to understand.

The seven segment displays are connected to Ports C and D. These are switched on and off as the coding calls for it. Each tag corresponds to a receiving tag that on each display, as shown in figure 26.

Figure 25 - Pic 16f788a Microcontroller.

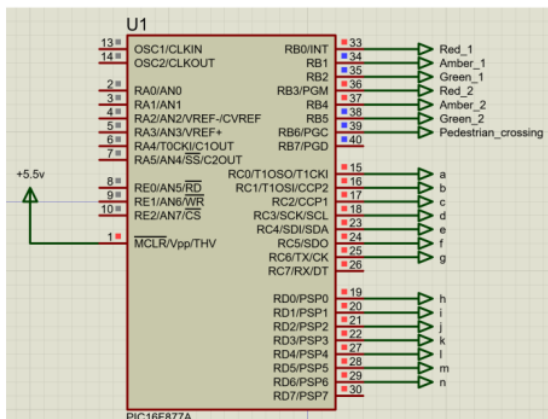


Figure 26 - Seven segment displays.

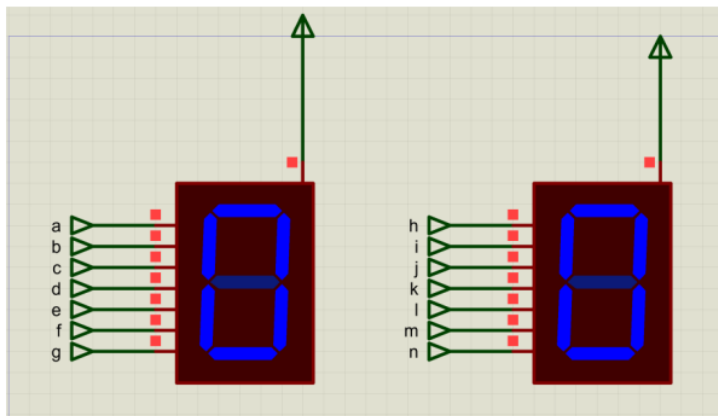
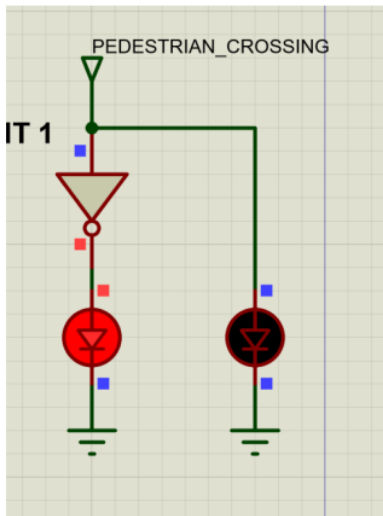


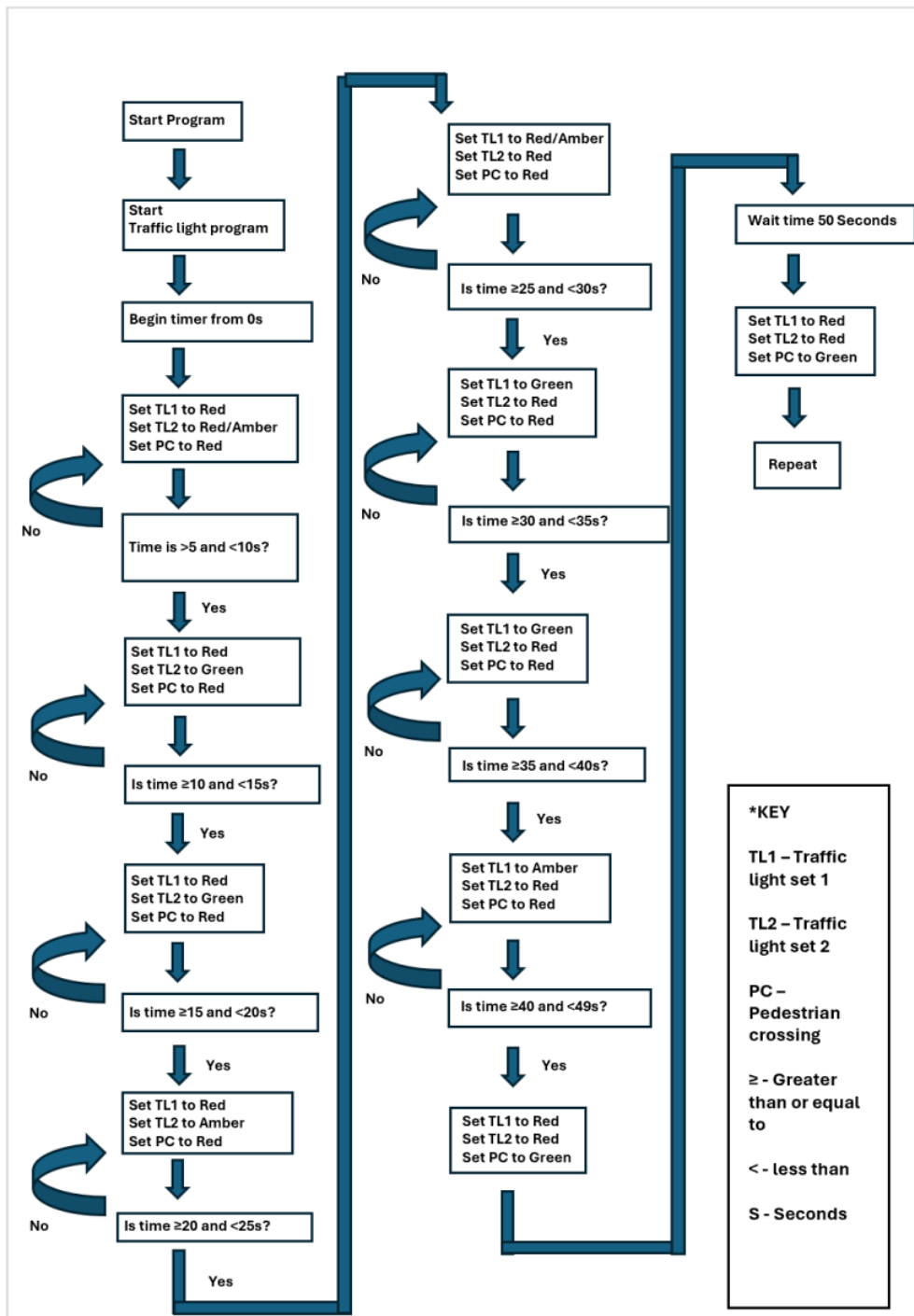
Figure 27 - Pedestrian crossing and 'Not' gate.



By using a 'NOT' gate in my circuit, I am able to activate both the red and green crossing using only one output pin. This saves space on the PIC controller, allowing more room to upgrade the project if required in future. The NOT gate will invert the signal that has been output by the controller. If the code is calling for a high state (1) then the green light will switch on, allowing pedestrians to cross. If this pin is low (0), the green light will switch off while the NOT gate will invert this low signal, switching the red light on.

Flowchart

The following flowchart shows the correct sequencing of the circuit and the time conditions required before moving on to the next part of the sequence. It shows that it will not change the outputs to the traffic lights unless it is in the correct time limits.



Conclusion - Recommendations of future work

The overall project was successful in creating a traffic light system that operates using a PIC controller. Using techniques such as a NOT Gate and input/output tags ensured that the circuit design is easy to follow, minimising the number of lines in the circuit.

By coding the seven segment displays to count up rather than counting down, it allows room for future improvement and expansion to the project if required (such as adding more traffic lights or creating filter lights).

The traffic light sequence correctly represents an ideal 'real world' traffic light sequence and ensures that the correct colour lights are switched on at the correct times. Using a counter to switch the time ensures that all traffic lights are on for a specific amount of time. In a real world situation, these times can be adjusted based on the density of traffic coming from each direction. This would ensure a smooth flow of traffic.

There are other ways in which I could have created my design. The sequence that I chose will use a set time to change the lights. Another way of changing the lights would be to use sensors that would dictate the density of traffic in one direction. In a real world scenario, this may be achieved by using inductive loop sensors that will detect a build-up of traffic in one direction, causing the lights to change early to prevent a build-up of vehicles. (Crisara, 2023). Lights may also have 'trigger sensors' on top of them that will be able to detect the approach of vehicles. These allow lights to maintain green lights on the busiest intersections to allow a smoother flow of traffic (mate, 2019). My design used a timer as it was based on a junction that has a similar flow of traffic through each entry point. As each junction would have a similar density of traffic, the timer-based approach was suitable.

In future it is possible to make some changes if required to the circuit that would potentially reduce the number of pins used. Implementing more NOT gates into the traffic light system instead of just the pedestrian crossing may help to reduce the number of pins by even more.

The PIC controller used has many spare pins, if required, there are other PIC boards available that have less pins such as the PIC18f26k22. These are slightly less expensive, approximately £4 compared to the £5 of the PIC controller (Mouser, 2021) so if cost is an issue these could be used, they do however limit the potential for future improvement.

Software/Equipment used.

The following software was used to create my project:

- MikroC Pro

- Proteus 8
- Microsoft Word
- Microsoft Excel

Within Proteus, the following components were used:

- PIC16F877A
- LED Red
- LED amber
- LED green
- Traffic light module
- Seven segment display
- 'NOT' Gate
- Power – 5vdc
- Ground

References

BSM, 2017. *Traffic lights*. [Online]

Available at: <https://www.bsm.co.uk/traffic-lights>
[Accessed 24 03 2024].

components, A., 2018. *ACI*. [Online]

Available at: <https://www.alliedcomponents.com/blog/circuit-simulator-how-it-helps-you-understand-any-circuit>
[Accessed 01 04 2024].

Crisara, M., 2023. *Popular mechanics*. [Online]

Available at: <https://www.popularmechanics.com/cars/car-technology/a45015101/how-traffic-lights-work/>
[Accessed 27 03 2024].

mate, A., 2019. *Auto mate systems*. [Online]

Available at: <https://www.automatesystems.co.uk/traffic-lights-signals-work>
[Accessed 25 03 2024].

Mouser, 2021. *Mouser electronics*. [Online]

Available at: https://www.mouser.co.uk/ProductDetail/Microchip-Technology/PIC18F26K22-I-SP?qs=RXaOwepKIZ15live%2FKf%252BKg%3D%3D&mgh=1&vip=1&utm_id=20808080842&gad_source=1&qclid=Cj0KCQjwk6SwBhDPArisAJ59GwebSAsVSzz3wbQK0ZmdcYFXA09hY8V29TqblOnGKKMn1dMSeo7Fh_laAnDSEALw_w
[Accessed 31 03 2024].

Raj, 2018. *embetronicx*. [Online]

Available at:

<https://embetronicx.com/tutorials/microcontrollers/pic16f877a/pic16f877a-introduction/#:~:text=PIC16F877A%20has%2040%20pins%20by,pins%20to%20do%20the%20job.>

[Accessed 29 03 2024].

RS, 2023. *PIC16f877a*. [Online]

Available at: <https://uk.rs-online.com/web/p/microcontrollers/4671690>

[Accessed 29 03 2024].