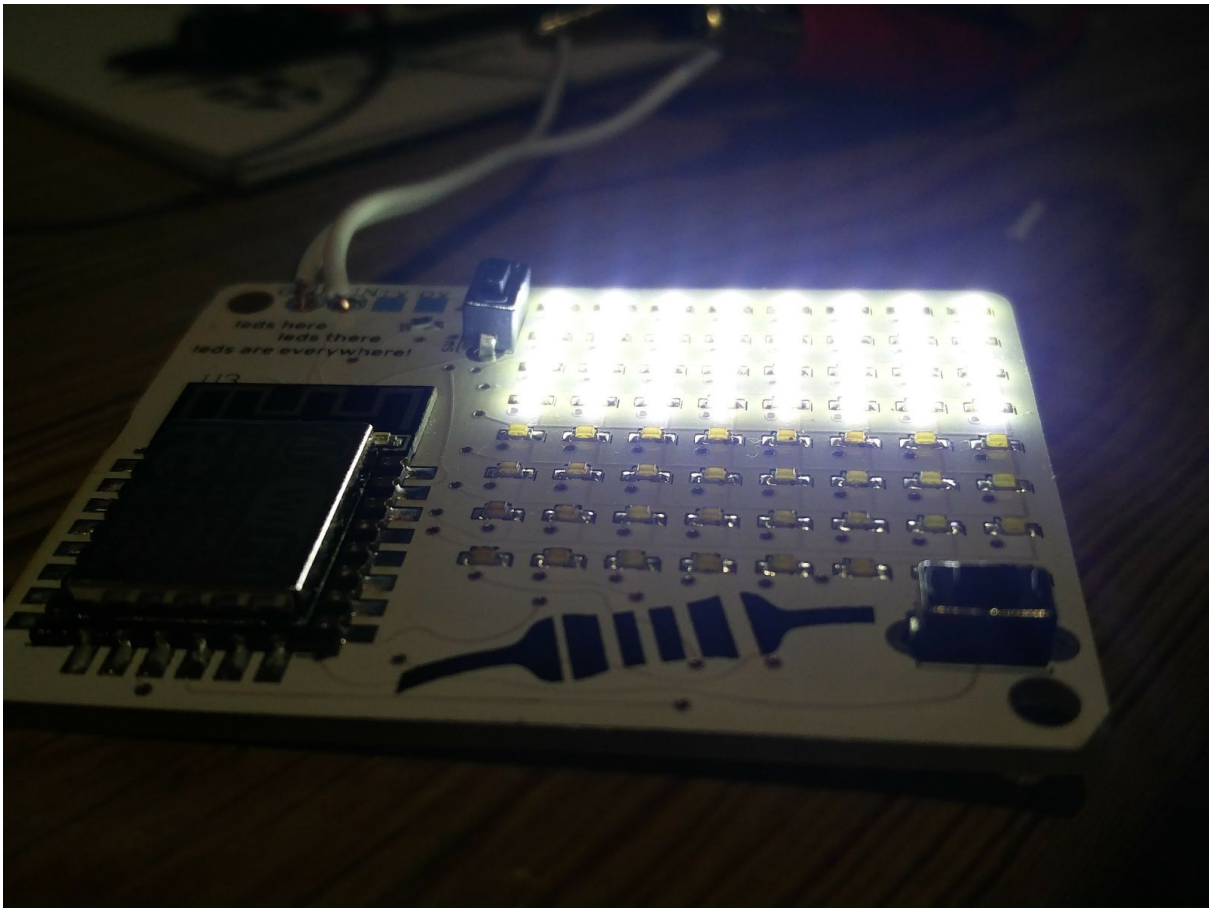
# LED Matrix Project




**Archie Roques**

**City of Norwich School**

**Silver CREST Award**

**July 2017**

# Acknowledgements

I would like to acknowledge the following people and groups for their guidance and assistance throughout this project without which it could not have happened:

- Paul Thornton, Ann Lussignea, and Barny Relph for their mentoring and guidance on programming
- Dan Herring for his assistance with and provision of manufacturing tools and components
- Tim Parnell for his general assistance
- And finally Norwich Hackspace for providing an excellent environment to work on this project.

# Table of Contents

# Research

**Why are LED matrices useful?**

LED matrices are responsible for the big screens used in football stadiums, shopping centres, art installations and transit signs - LED displays can be produced in massive sizes, up to 7.11m diameter (Anon 2011), and are also favoured over other solutions due to the lower cost (than comparable LCD or plasma displays) and durability over time. Whilst the atively simple (Brain 2011), in practice the electronics can be a lot more complicated (Anon 2003) and involve a large amount of circuitry and code.

**Why is multiplexing necessary?**

The process of getting an image (a file, or a video output from a computer) to display on a screen is relatively complicated (Nielsen 2013; 'chr' n.d.). Because each LED can measure fractions of a millimetre, and because three LEDs are needed for each pixel (one red, one green, one blue) (Brain 2011), there can be millions of LEDs per display. Driving each one of these individually from a microcontroller would be impossible, the sheer level of processing power required would be incredible. So, an alternative method is used.

**How does multiplexing work?**

By connecting all the anodes of the LEDs in rows and the cathodes in columns, it is possible to connect a grid whereby differing voltages can be applied to differing rows and columns to light up specific pixels in the grid. This is known as multiplexing.  (Nielsen 2013)

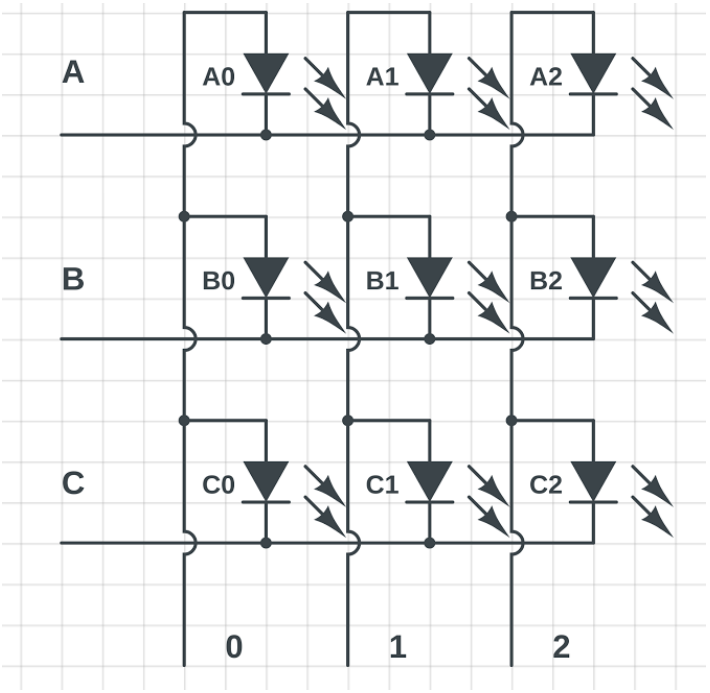



*Fig. I: A simple multiplexing layout*

For example, in the diagram above, to light LED A0, a positive voltage would be applied to column 0, and 0V would be connected to row A. To turn off the LED, either the positive voltage could be changed to 0V, or the 0V changed to the positive voltages as both would create a potential difference of zero.

However, if more than one LEDs are required to be lit (which is necessary for displays), there is an issue. For example, if one wanted to light LEDs A0 and C2, they could supply 0V to rows A and C, and positive voltages to columns 0 and 2. This would then result in A0 and C2 lighting. But, it would also result in two other LEDs lighting - C0 and A2, because they would both have a voltage across them. This creates a problem as it is seemingly impossible to light multiple LEDs at once. (Hsiung 2004), (Anthony Edward Green 1999)

**Persistence of Vision**

However, it is possible to pulse through the rows and columns of LEDs, such that only one LED is lit at one time, rather than the two as above, but the eye is tricked into thinking that multiple LEDs are illuminated at once. This is due to the scientific phenomenon of persistence-of-vision, or POV as it is commonly known. Depending on brightness, images can be seen on the retina for between 66 and 100ms (Thornton 2014). This converts to a 'framerate' of between 15 and 10 fps[1]. Modern television and film broadcasts at 24fps, and most modern displays have a 'refresh rate' of 50 or 60 Hz, ie 50 or 60 refreshes per second (M Armstrong, D Flynn, M Hammond, S Jolly, R Salmon 2008). Therefore, despite modern screens effectively showing a series of static images at great speed, it appears to be one continuous moving picture. The same principle is behind zoetropes and flip-books and enables 'dimmable' LEDs to be produced through pulse-width modulation.

It is possible to use this in an LED matrix - but to do so would ordinarily require a large amount of microcontroller pins - one for each row and for each column of the display. These pins have to also be capable of alternating their state at a rate of over 15 times per second. Without having highly complex code and a microcontroller with a significantly high number of high-speed pins, it would seem that creating a matrix using POV is impossible.

**Making multiplexing easier**

There are however integrated circuits designed to do this very thing - scan through rows and turn on or off certain ones in a given sequence. Shift registers convert between *parallel* and *serial* data. (Dee 2013) Serial data is transferred one bit at a time (ie down a single wire), whereas parallel data is transferred multiple bits at a time - faster, but uses more cores. (Jim 2012). What it is possible to do with the shift register therefore, is convert data transmitted

[1] fps = Frames Per Second

from the microcontroller over a single wire, to the series of high and low inputs and outputs needed to power the rows and columns LED display.

# Methodology

## Component Choice

### Shift Register

One of the most commonly available shift registers is the SN74HC595. It is a SIPO[2] shift register, meaning it will convert the serial data from the microcontroller to the parallel output for the LEDs rather than the other way around. It also operates at voltages between 2 and 6 volts which makes it suitable for the forward voltage of most LEDs, which lie in that range. It is also available in several surface mount formats which will save space on the design. (Texas Instruments 2017)

### Microcontroller

The ESP8266 is a popular and low-cost microcontroller, with wireless networking capability. It has a logic level of 3.3V, consistent with the forward voltages of white, green and blue LEDs, and also with the shift register's voltage range. The chip is available in a surface-mount format, suitable for hot air soldering to a circuit board which will help reduce time, and has a large amount of flash memory available for programming. It is also possible to program the IC in a number of popular programming languages, such as Arduino/C++, Node.JS and Python. (Espressif Systems IOT Team 2015)

Although it is possible to use the ESP8266 SoC[3] on the PCB alone, there are some other components (flash memory, for example) that are necessary in order for it to run code (in its bare state the chip can be communicated to from another IC to perform certain functions,

[2] SIPO = Serial In, Parallel Out
[3] SoC = System-on-a-Chip, a microcontroller needing no other components for barebones operation (although some others may be needed for desired operation)

but this project will use its full functionality as a microcontroller. Luckily there are many modules available that include both the chip and any other necessary circuitry. One of these is the ESP-12F, which will be used in this design because it is compact yet contains enough I/O[4] pins and all other electronics needed for operation of the chip as a microcontroller.

**LEDs**

In order for the image to be visible as it is intended, and the shapes properly identified, the LEDs need to be fairly small. The smallest commercially available LED is in an *0402/1005* package, meaning it measures $0.04 \times 0.02\ inches,\ or\ 1 \times 0.5mm$. This is largely impractical to solder by hand, and even to small for many commercial manufacturers to work with. The next available size, and the most common package for SMD components, is the *0603/1608* package, measuring $1.6 \times 0.8mm$. It is relatively easy to solder by hand, even without a reflow facility, and the diodes are relatively low in cost. A variety of colours are available, such that many variations of a design could be produced with different coloured LEDs. (’zerodamage' 2012)

**Other components**

The circuit will require a voltage regulator - for this I am using the XC6206P332MR as it is available in a compact format which will make circuit design easier. It is a *low dropout* regulator, meaning minimal power above the output voltage is required for the regulator to function - the dropout voltage is 250mV. It can supply 200mA which should be fine for the number of LEDs (in reality they could draw a lot more, but the display would become uncomfortable to look at). (Autodesk 2017; Ltd 2016)

[4] I/O - Input/Output

The circuit will require a variety of passive components, such as resistors and capacitors. 'Variety packs' of components can be purchased which include a range of popular parts. There are 0603/1608 versions available, which will be used for the reasons above. (Beech 2015)

## Tools and processes

### EDA Software

For the EDA (electronic design automation) software, I plan to use Autodesk EAGLE. Although not free, it has a free educational license, capable of producing boards with 4 schematic layers and a 160cm$^2$ board area. (Autodesk 2017)

### Board manufacture

For the manufacture of the circuit board, the SeeedFusionPCB has a low turnaround time and very low cost (US$5 for 10 pcs of $100 \times 100mm$). Although the boards will be manufactured in China, expedited shipping is available to the UK to reduce the turnaround time. This manufacturer has been proven in the past to be reliable, hence it is a good choice for this project. (Seeed Limited 2017). I will design the circuit and layout of the PCB, and will supply the files to the manufacturer (see appendix I).

### Application of solder

Before production, a laser-cut solder stencil will be produced using a laser cutter and Mylar film - a thin, strong and flexible polymer. Then, solder paste will be applied via the stencil and pressed onto the board, making it ready for component placing. Other methods are available, such as selective-solder and wave-soldering but these are more suited to through-hole components as opposed to the surface mount parts used in this design.

**Component placing**

In an ideal world, a pick-and-place machine would be used to place the components on the circuit board. This CAD machine uses the PCB design files, and is loaded with reels of components which are automatically picked up and placed on the circuit board, held in place with the solder paste.

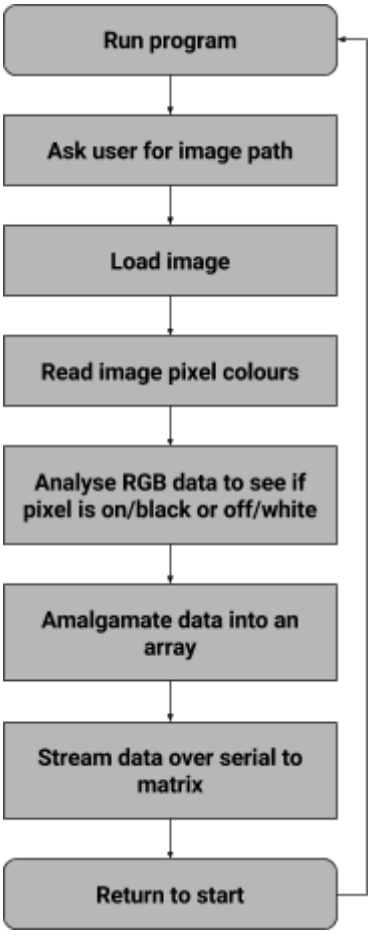
**Reflow**

In industry, a reflow oven is used, with a conveyor belt to move the circuit board through differing levels of heat, usually from an infra red lamp. The solder then melts and the components move into place on the pads. Unfortunately, without a professional oven this is somewhat unreliable, hence the board will be soldered using a hot air gun. Usually used for reworking failed boards, this has a controllable stream of air in a handheld format, with temperatures up to $500^{O}C$.

## Programming

### Computer-side software

There are two parts to the software needed for the matrix to work and display pre-designed images correctly. This assumes that the images are already designed, probably in common PC software such as Microsoft Paint or alternative. The images consist of $8 \times 8$ pixels, one for each LED in the matrix. The image must also only contain black and white, with black pixels representing those LEDs turned on and white representing the LEDs turned off.

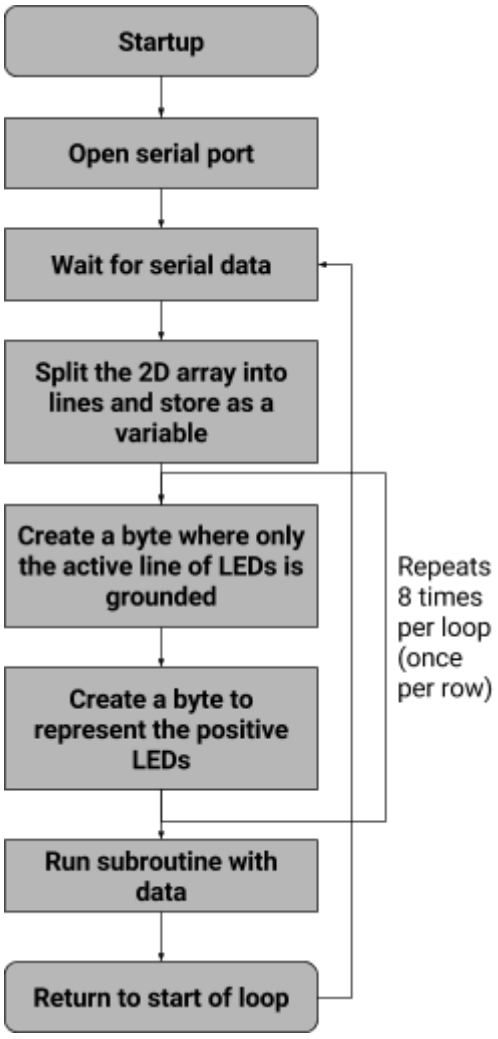



The first part is written in python, and runs on the computer. It is in this case written for Linux systems but could be modified to work on other systems (Windows and MacOSX) with little alteration. It asks the user for a path to the image file, opens it, converts it into an array, presents this to the user, and then sends it over USB Serial. The flow diagram for this software is as in *Fig. II (right)*.

### Matrix-side software

Once the data has been sent, the data will be converted via a USB-To-Serial converter PCB (purchased online), such that it is true serial data and not under a USB protocol. Software on the microcontroller must receive this over the RX/TX[5] lines, and convert the array to raw data, stream the data out to the shift registers which turn the correct LEDs on. The flow diagram for this is below, *Fig. III*.

[5] RX = Receive Serial (connected to the converter's TX), TX = Transmit Serial (connected to the converter's RX)




Matrix-Side Code Flow Diagram

Additionally, there are two subroutines in the board-side code, one for updating the vertical columns with their new value, and one for updating the horizontal rows. Both are almost identical in functionality with only a few pin numbers changed between them. The basic structure is shown in *Fig. IV* (right):

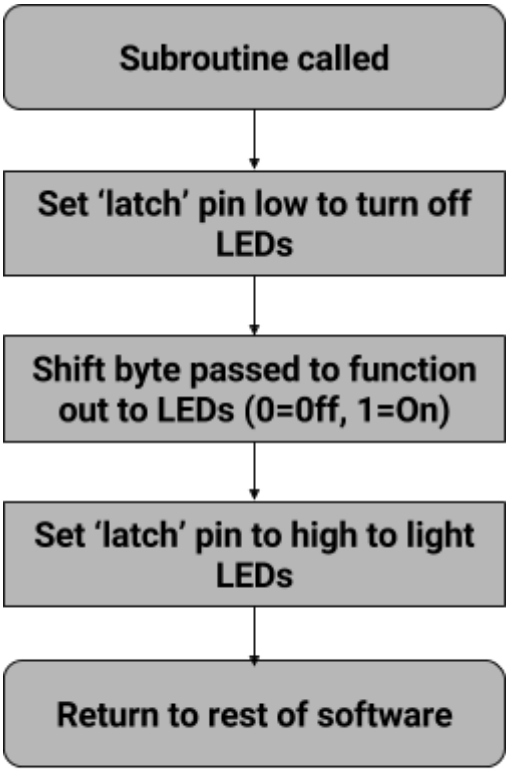



Fig. III:

*Fig. IV*: Shift Register Update Subroutine Flow Diagram

# Risk Assessment

| Process | Details of Risk | Likelihood | Severity | Level of Risk | Details of Mitigation | Assessed Likelihood | Assessed Severity | Assessed Risk |
|---|---|---|---|---|---|---|---|---|
| Laser cutting PCB stencil | Burning or blinding from laser | Medium | High | Serious Risk | Use safety guard and interlock | Very Low | High | Low Risk |
| Application of Solder Paste | Poisoning or Ingestion | Very Low | Medium | Low Risk | Take care, do not ingest paste, wear gloves | Low | Low | Low Risk |
| Melting of solder paste with Hot Air Station | Burning from hot air | Medium | Medium | Moderate Risk | Take care, have first aid nearby | Low | Low | Low Risk |
| Design and programming of circuit | RSI or eye strain | Very Low | Medium | Low Risk | Take regular breaks | Negligible | Medium | No Risk |

# Evaluation

The LED matrix was manufactured successfully, and the matrix work. It is possible to import a PNG image into the python driver code, on a computer, and connect the matrix over USB. The data can be streamed and the matrix updated accordingly as serial data is sent to the matrix. Although the matrix does not have a function in line with that of a modern display, it does work in the same way on a smaller scale.

## Changes during production

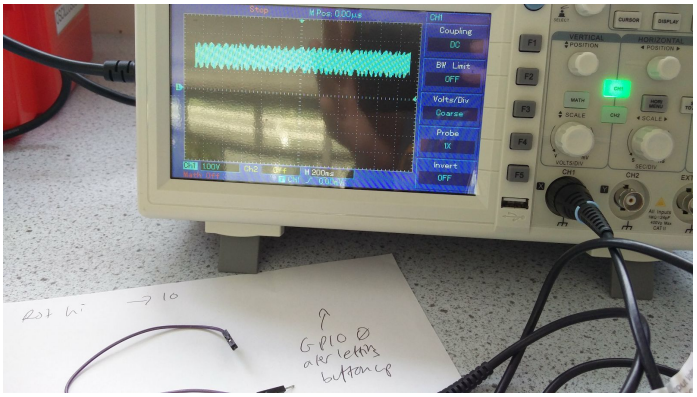
Many of the changes were due to issues in the schematic and circuit board layout. In industry it is common to have many revisions of a design before the final PCBs are produced, and prior to a PCB being ordered many small prototypes are manufactured. Unfortunately due to the time constraints of the project such a rigorous process was not possible and a couple of errors were in the production PCB.

#### Reset line and button failure

Due to an error in the schematic, the reset button appeared connected to the reset line of the microcontroller, but in practice this was not so. This meant that while debugging, it seemed as if the chip was not resetting when intended, when in fact this could have been partially due at least to the button not being connected. To fix this, a small wire was soldered from the switch terminal to the microcontroller pin - this can be seen in *Fig. V* (below). This replaced the missing track and fixed the issue. In future editions a trace would be made on the PCB to avoid the issue.

**Boot mode resistor failure**

Unfortunately one of the resistors on the PCB, designed to pull GPIO15 to a low state via a 10k resistor, had failed. Additionally, it seemed that the resistor to pull GPIO0 to a high state had also failed, and after testing with an oscilloscope (see *Fig. VI*, a photo of the oscilloscope reading, right) it became evident that the pin was 'floating'[6]. This needs to happen in order for the chip to boot in the correct mode, and read the information from the flash memory storage on the module rather than be used via UART commands or other forms of communication. The error could have been due to a faulty component (unlikely), or due to an excess of heat being used when reflowing the solder and damaging the component, which is the most likely cause of failure. To fix this, the resistor was simply circumvented and the pin connected directly to ground, as shown in *Fig. VII* (below), which was incidentally the pin next to GPIO15. This fixed the issue. In future, using a reflow oven would avoid damage to components, and using a component tester (ohmmeter in this case) prior to fitting components would fix this issue prior to or during assembly. In the second construction, low temperature solder paste was used to try and avoid the problem and indeed the issue was averted.




[6] Floating = not in any state (high or low) - in practice, this means that the pin isn't connected to either a voltage source or ground and can lead to erroneous sensor readings or the wrong state being detected.

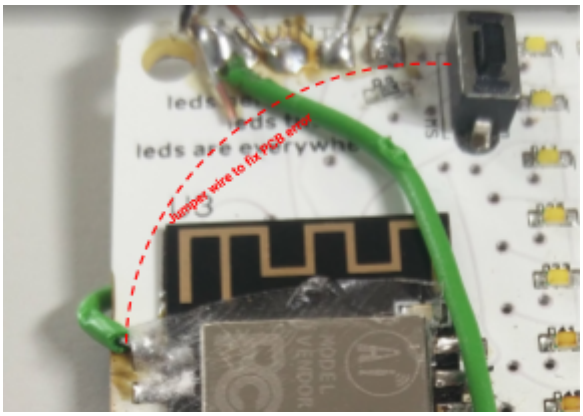



*Fig. V: The connection for the reset line*

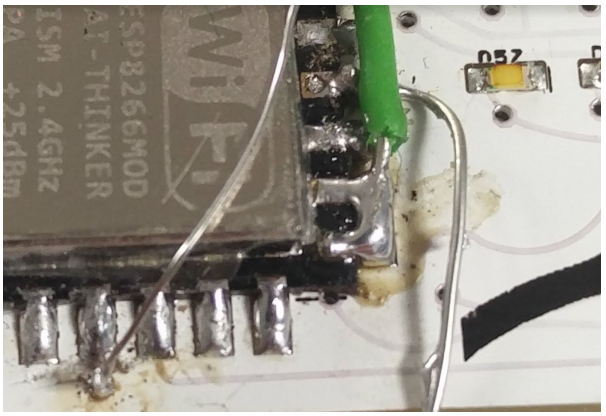


*Fig. VII: Some of the corrections on the PCB*

**Subsequent ground trace issues**

Unfortunately, while removing the faulty component, the track between the ground input to the board and the microcontroller's ground pin was destroyed - hence another wire reconnecting the microcontroller to the circuit ground pin. If the above fault had not occurred, this fix would not have been needed.

**Failure of GPIO pin 9**

On the datasheet for the ESP-12F module, there is a small asterisk by GPIO pin 9, demonstrating that it must only be used as an input, as it is used for communication with the flash memory chip on the module. Unfortunately this was not noticed when designing the PCB, so the latch pin for the columns was connected to GPIO9. This caused a soft reset when the pin was assigned as an output. This took a while to debug - the serial console did report a reason for the reset, code `'3,6'`, but this could have been due to a number of things. In order to fix the error, the code was all commented out, and gradually introduced line-by-line until the failure was found. This made it easy to deduce the fault, and correct it in hardware. To do this the trace that connected the pin of the shift register to GPIO 9 was cut. Then, a wire was soldered from a via on that trace to an alternative, unused GPIO, GPIO5.

In future, careful reading of the datasheet and a consequent schematic alteration would save this issue from occurring.

**Voltage Regulator**

During testing, it was noted that the regulator was heating up to an abnormally high temperature. To ascertain if this was a problem, a thermal imaging camera was used to examine the regulator, and check if it was an issue. This is shown in Fig. VIII and Fig. IX (below). Whilst the regulator was a little warmer than desirable, it was not warm enough to cause significant issues and demonstrate potential for a failure. In future editions, this would ideally be rectified by replacing the regulator for a component capable of handling a higher current.

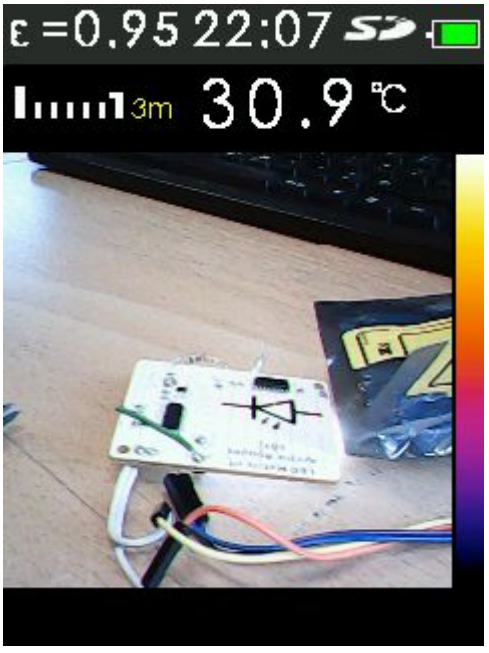



Fig. VII: The PCB in testing

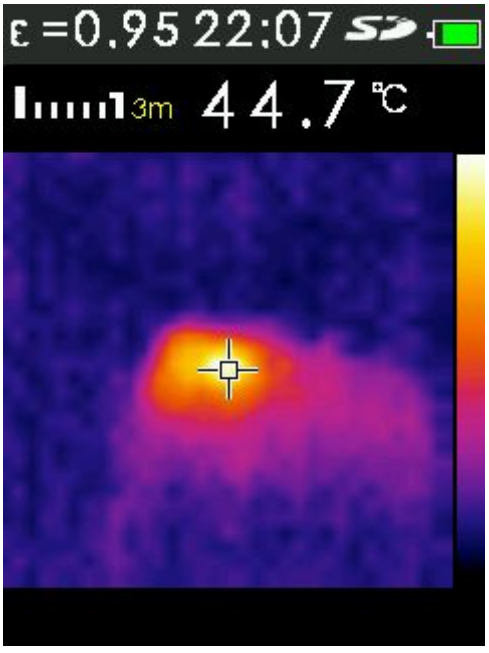



Fig. IX: Thermal image of PCB showing regulator

**Circuit Redesign**

After this project, the circuit board may be redesigned to accommodate the required changes. This would make any future construction easier.

# Analysis

## Prototyping

This project would have been easier to undertake, if an exact prototype had been constructed before the PCB design begun. Although a simple prototype was constructed to demonstrate the concept of being able to light individual LEDs whilst arranged in a matrix, the prototype did not use shift registers as the final design did, and a pre-manufactured development board was used such that issues that occurred with the specifics of the components surrounding the microcontroller could not be spotted prior to the manufacture of the PCB. This was due to the time constraints which caused the PCBs needing to be ordered sooner to account for shipping time. In an industrial implementation of this project there would be more time for prototyping.

Additionally, it would have been helpful to construct a prototype PCB locally using equipment prior to sending the board design files for professional manufacture. Unfortunately equipment for this (namely a CNC mill, or laser cutter/photosensitive paper and etchant) was unavailable, but this would have resulted in a near-perfect PCB being manufactured as all problems could have been fixed before the PCBs were manufactured.

## Modular design

An additional step which would have made the troubleshooting process easier would be if the design was modular. There are a number of LED matrices already available on the market, which would have eliminated a degree of complexity in assembly, although the solution would not have been as neat and customizable as the final product created here. Additionally, the design could have been produced on several PCBs to be joined together

instead of one large unit, allowing for better flexibility, and making it easier to replace single elements of the design (changing the LED colour would be trivial for example), but at a cost of increased PCB area.

## PCB size

Altogether, it would have been easier to troubleshoot and fix the PCB if it were larger. Had the components been placed with more space between them it would have been easier to fix the errors as more space to work with would have reduced the potential of short circuits. However, the current solution is very neat and small, but would have been more suited to a finalised solution with no issues to fix.

## Increased test pads

Finally, fixing and diagnosing the errors would have been significantly easier if there were more test pads available - this way adding jumper wires to replace missing or damaged or erroneous traces would have been significantly simpler. Again, this would have come at a cost of increased PCB size.

# Further Research

There are a number of opportunities for this project to be extended. Firstly, the matrix could be extended in size to produce a larger display more suited to displaying scrolling messages or playing simple games. This would require more shift registers to drive the LEDs, and, a microcontroller module with more output pins. The ESP8266 microcontroller used has more IO pins available, but the chosen breakout, the ESP-12F, has only a few of these available to use. This was used for simplicity in the design, but the bare chip could potentially be used, along with the other circuitry required.

Additionally, a different microcontroller could be used to add functionality. For example, the ESP32 chipset is the next generation of the ESP8266, and has more functionality including bluetooth, greater IO pins, capacitive touch for added control, and a hall effect sensor. This could lead to more methods of control, including potential for sensors to be used and the results displayed on the matrix.

The matrix could also be extended to three dimensions - this would require significantly more LEDs, and would likely have to be made using through-hole components rather than the surface mount LEDs used here, such that the LEDs could be assembled into three dimensions. This also introduces another challenge of driving the third dimension and mapping and designing images to be displayed.

Another option would be to use higher-power LED chips to make a brighter and bigger matrix, which would have more power and be viewable from further away. This could be used as a clock, or larger scrolling-message display, for example a scoreboard. This would

involve some form of power management, potentially a relay or MOSFET transistor, or perhaps a specific LED driver chip. Alternatively, traditional incandescent / filament bulbs could be used. All this would require is for relays to switch the bulbs on and off to be connected in place of the LEDs. Whilst the current design for the circuit board makes this difficult, redesigning it to accommodate relays would be trivial, although it would result in a larger PCB. There may also be issues with the response rate of incandescent bulbs.

Finally, another extension would be to replace the single colour LEDs used with three-colour RGB LEDs. This allows for almost every colour to be created, and for much larger images to be displayed. However, driving these is much more of a challenge, and in reality specialist ICs are used. Commonly in situations where these are desired, the driver chips are integrated into the LEDs themselves, and data is fed over a predetermined protocol to instruct certain LEDs to light up in a certain manner - examples of such chips include the WS2812/SK6812 line of LEDs, and the APA102 family, which uses the SPI protocol to program the colours. It is, however, possible to hard-wire these to shift registers.

# Appendix I

## Schematic

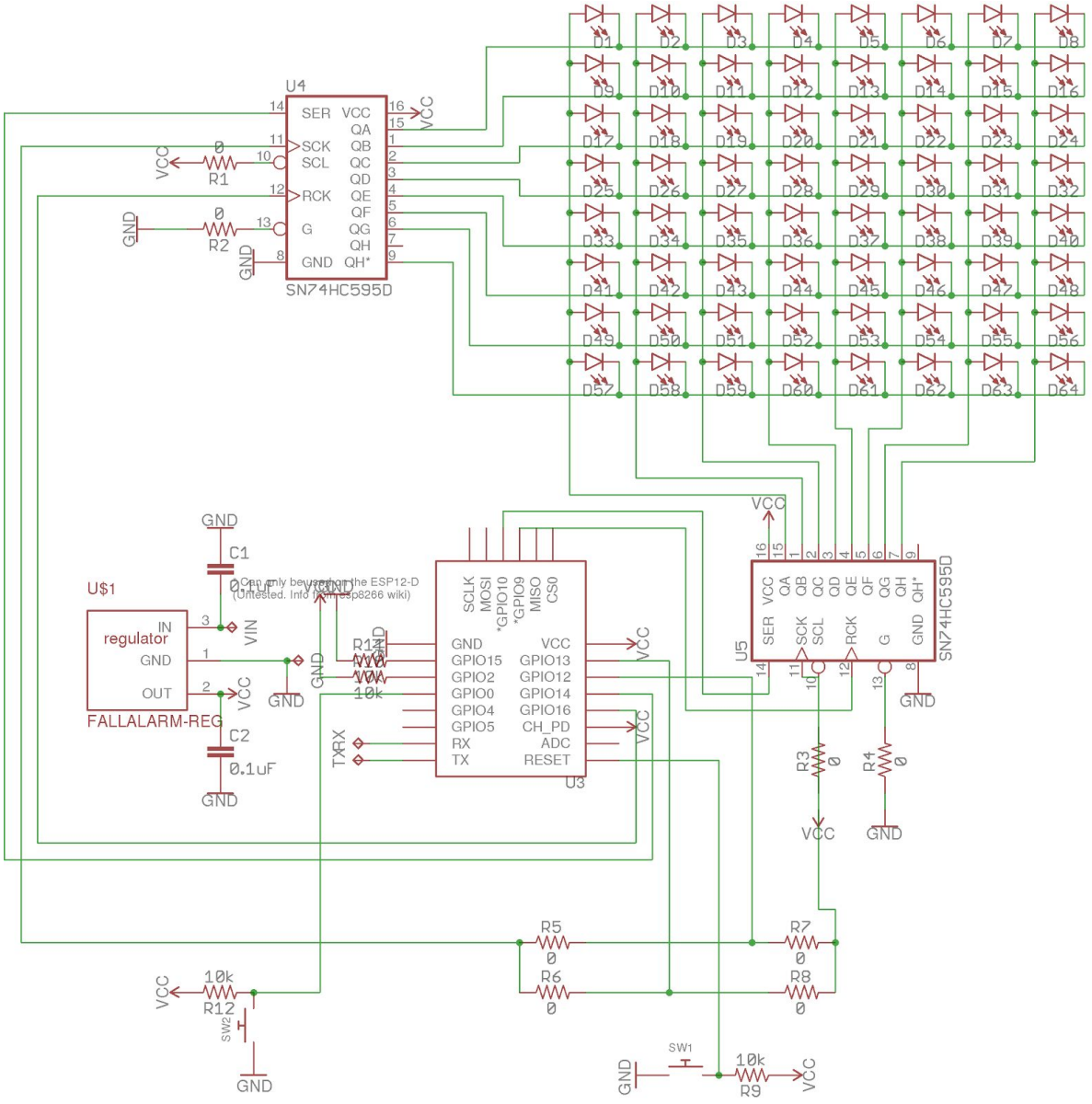



*Fig. X - Schematic File for the project*

## Circuit board layout

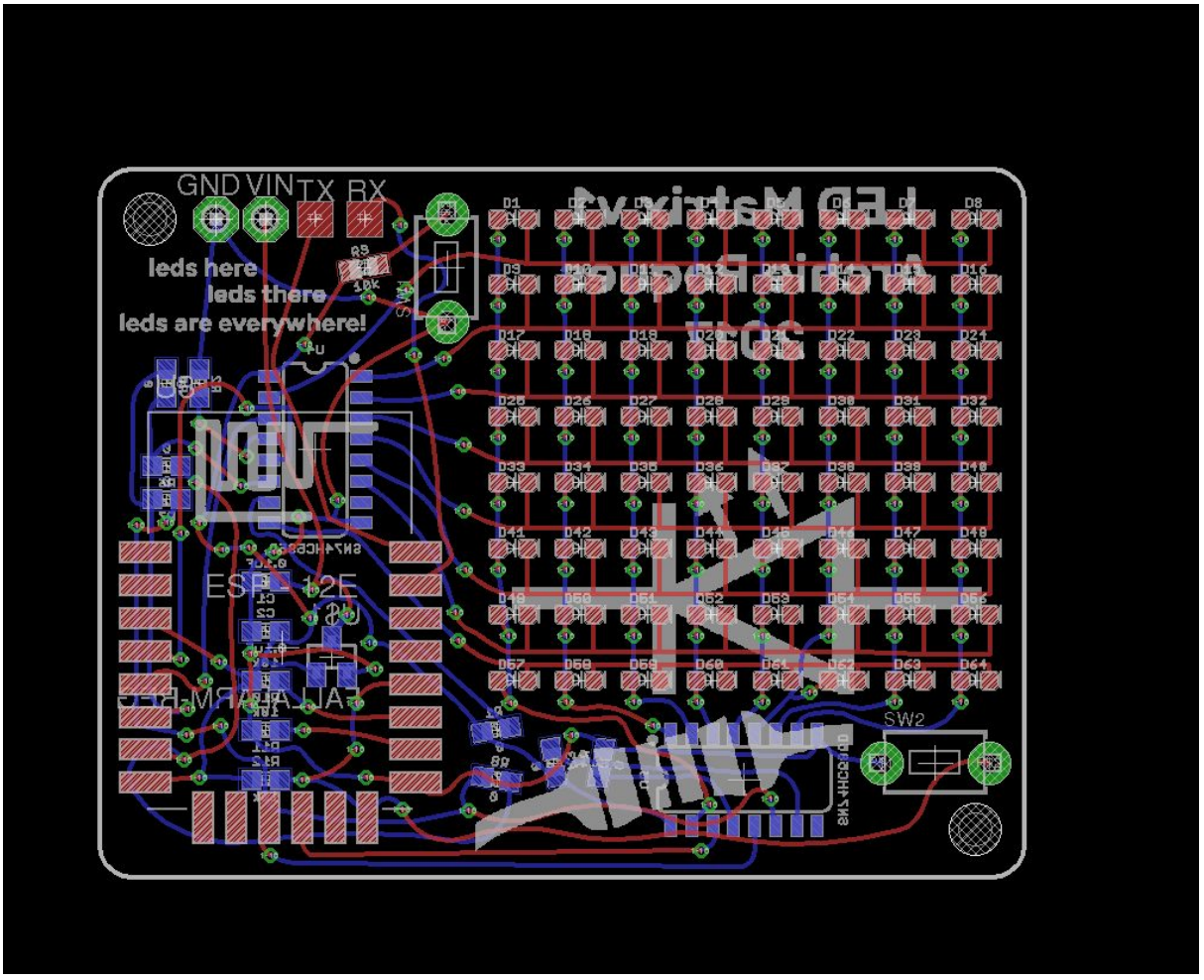



*Fig. XI - PCB layout. Red = top layer copper, blue - bottom layer copper, green = both layer copper, white = outline, grey = silkscreen*

# Appendix II

## Computer-Side Code

Written for Python 2.7

```
1.  #import the necessary libraries: Image (for loading the pic), NumPy (to store the
    image data) and serial (for talking to the matrix)
2.  import Image, numpy, serial
3.  #import the data type which will store the final number as a byte
4.  from ctypes import c_ubyte
5.
6.  #try and open the serial port:
7.  try:
8.      tothematrix = serial.Serial('/dev/ttyUSB0', 9600)
9.      #if that fails, try the other serial port (for some reason it sometimes uses
    this)
10. except serial.SerialException:
11.     tothematrix = serial.Serial('/dev/ttyUSB1', 9600)
12.
13. #define a function to convert the boolean array into integers
14. def ba2int(bool_array):
15.     #create an empty C byte
16.     x = c_ubyte()
17.     #set the value of this to the converted bool array
18.     x.value = (sum(2**i for i, v in enumerate(reversed(bool_array)) if v))
19.     #return this to be used in the code
20.     return x
21.
22. #set up a main function
23. def main():
24.     #ask the user where the image is
25.     imgpath = input('What is the path of the image?')
26.     #TESTING ONLY
27.     imgpath = "/home/archieroques/smiley.png"
28.     #open that image with the Image library
29.     img = Image.open(imgpath)
30.     #converts the image to black and white in case it isn't already
31.     img.convert("1")
32.     #makes a NumPy array from the image, as this is something the NumPy library
    will do
33.     array = numpy.array(img)
34.     #converts that to a standard python array
35.     array.tolist()
36.
37.     #create a eight rows, with eight bools per row (so one per LED)
38.     #default value is 'X', a string string which is an unacceptable/null value
    anyway
39.     row0 = ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
40.     row1 = ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
41.     row2 = ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
```

```
    row3 = ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
    row4 = ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
    row5 = ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
    row6 = ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']
    row7 = ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X']

    #create empty unsigned byte variables to hold the data to be sent
    send0 = c_ubyte(8)
    send1 = c_ubyte(8)
    send2 = c_ubyte(8)
    send3 = c_ubyte(8)
    send4 = c_ubyte(8)
    send5 = c_ubyte(8)
    send6 = c_ubyte(8)
    send7 = c_ubyte(8)


    #create a 2D array to temporarily store the boolean image data in
    bool_array = [[0 for x in range(8)] for y in range(8)]

    #for each pixel in each row:
    for row in range(8):
        for pixel in range(8):
            #if pixel is white: set that LED to off
            if str(array[row][pixel]) == '[255 255 255]':
                bool_array[row][pixel] = False
                print('off')
            #if pixel is not white: set that LED to on
            else:
                bool_array[row][pixel] = True
                print('on')

    #for each pixel in each row:
    for row in range(8):
        for pixel in range(8):
            #store the relevant pixel in a holding variable
            val = bool_array[row][pixel]
            #if the last item in the row still contains a zero (ie the row hasn't
been filled yet)
            if row0[7] == 'X':
                #set the pixel in the 1D array to the value of that pixel's
equivalent in the 2D one
                row0[pixel] = val

            #repeat this for the next 7 rows too so all pixels are accounted for:

            elif row1[7] == 'X':
                row1[pixel] = val

            elif row2[7] == 'X':
                row2[pixel] = val

            elif row3[7] == 'X':
                row3[pixel] = val
```

```
94.
95.             elif row4[7] == 'X':
96.                 row4[pixel] = val
97.
98.             elif row5[7] == 'X':
99.                 row5[pixel] = val
100.
101.                elif row6[7] == 'X':
102.                    row6[pixel] = val
103.
104.                elif row7[7] == 'X':
105.                    row7[pixel] = val
106.
107.        #print all the rows for manual checking
108.        print(row0)
109.        print(row1)
110.        print(row2)
111.        print(row3)
112.        print(row4)
113.        print(row5)
114.        print(row6)
115.        print(row7)
116.
117.        #ask user to confirm data
118.        response = input('\n\n\nData good? y/n')
119.        #if they say yes:
120.        if response == 'y' or response == 'Y':
121.            #send each array to be converted to an integer and then a c_ubyte
122.            send0 = ba2int(row0)
123.            send1 = ba2int(row1)
124.            send2 = ba2int(row2)
125.            send3 = ba2int(row3)
126.            send4 = ba2int(row4)
127.            send5 = ba2int(row5)
128.            send6 = ba2int(row6)
129.            send7 = ba2int(row7)
130.
131.            #create an empty array of bytes
132.            barray = bytearray(8)
133.            #assign each value to a place in the bytearray
134.            barray[0] = send0.value
135.            barray[1] = send1.value
136.            barray[2] = send2.value
137.            barray[3] = send3.value
138.            barray[4] = send4.value
139.            barray[5] = send5.value
140.            barray[6] = send6.value
141.            barray[7] = send7.value
142.            #write that to the matrix
143.            tothematrix.write(barray)
144.
145.        #if they say no, go back to the start
146.        else:
147.            main()
```

```
148.
149.    #when the code is run, call the main function
150.    if __name__ == '__main__':
151.        main()
```

## Matrix-Side Code

Written in Arduino, A C++-like language

```
1.  /* LED Matrix Code
2.   *  By Archie Roques, July 2017
3.   *  Released under CC-BY-SA-NC Creative Commons 3.0 License
4.   */
5.
6.  //define the pins for one shift register, the active high one
7.  int latchPin = 14;
8.  int clockPin = 12;
9.  int dataPin = 16;
10. //define pins for the other register, the active low one
11. //(the clock pin is shared so only defined once)
12. int neglatchPin = 5;
13. int negdataPin = 4;
14. //define a pin for the onboard LED used for indication
15. int ledPin = 2;
16.
17. //define a counter variable for when data is recieved
18. int counter = 0;
19. //define a rownumber variable, starts at 0 by default
20. int rowNumber = 0;
21.
22. //define an integer i for use in counting
23. int i;
24.
25. //set up eight empty bytes to keep the data in, one for each row
26. byte row0 = B11111111;
27. byte row1 = B00000000;
28. byte row2 = B00000000;
29. byte row3 = B00000000;
30. byte row4 = B00000000;
31. byte row5 = B00000000;
32. byte row6 = B00000000;
33. byte row7 = B00000000;
34.
35.
36.
37. void setup() {
38.   //set all the pin modes as output (there are no input pins for the matrix)
39.   pinMode(latchPin, OUTPUT);
40.   pinMode(dataPin, OUTPUT);
41.   pinMode(clockPin, OUTPUT);
42.   pinMode(ledPin, OUTPUT);
43.   pinMode(neglatchPin, OUTPUT);
44.   pinMode(negdataPin, OUTPUT);
```

```
45.
46.   //begin a serial port at a baudrate of 9600b/s
47.   Serial.begin(9600);
48.
49.   //turn the LED pin to high, to turn it off (the ESP module uses an active-LOW
    configuration)
50.   digitalWrite(ledPin, HIGH);
51.
52.
53.
54. }
55.
56. void loop() {
57.   //call serialEvent because it sometimes won't work automatically on the ESP8266
58.   serialEvent();
59.
60.   delay(100);
61.   //this will basically just continually update the registers in turn with the byte
    data
62.
63.   //for each column:
64.   for (int col = 0; col < 8; col++) {
65.     //set the values to default (LEDs off)
66.
67.     byte colshifter = B11111111;
68.     byte rowshifter = B00000000;
69.     //set the current column LOW to activate the LEDs there
70.     bitClear(colshifter, col);
71.   //evaluate if the relevant LED in each row should be on, and set the row byte as
    such
72.   if ((bitRead(row0, col)) == 1) {
73.     bitSet(rowshifter, 0);
74.   }
75.   if ((bitRead(row1, col)) == 1) {
76.     bitSet(rowshifter, 1);
77.   }
78.   if ((bitRead(row2, col)) == 1) {
79.     bitSet(rowshifter, 2);
80.   }
81.   if ((bitRead(row3, col)) == 1) {
82.     bitSet(rowshifter, 3);
83.   }
84.   if ((bitRead(row4, col)) == 1) {
85.     bitSet(rowshifter, 4);
86.   }
87.   if ((bitRead(row5, col)) == 1) {
88.     bitSet(rowshifter, 5);
89.   }
90.   if ((bitRead(row6, col)) == 1) {
91.     bitSet(rowshifter, 6);
92.   }
93.   if ((bitRead(row7, col)) == 1) {
94.     bitSet(rowshifter, 7);
95.   }
```

```
96.   //send the data to the registers
97.    updateShifts(rowshifter, colshifter);
98.   }
99.
100.
101.    }
102.
103.
104.
105.   //create a function to update the registers with data as a parameter
106.   void updateShifts(byte data, byte negdata) {
107.     //turn outputs off
108.     digitalWrite(latchPin, LOW);
109.     digitalWrite(neglatchPin, LOW);
110.     //shift the data out
111.     shiftOut(dataPin, clockPin, LSBFIRST, data);
112.     shiftOut(negdataPin, clockPin, LSBFIRST, data);
113.     //turn outputs back on
114.     digitalWrite(latchPin, HIGH);
115.     digitalWrite(neglatchPin, HIGH);
116.   }
117.
118.
119.   //create a special function, which will run every time new data is recieved
    automatically
120.   void serialEvent() {
121.     //while there is a serial port open
122.     while (Serial.available()) {
123.       //store the latest byte in a variable
124.       byte inByte = Serial.read();
125.       //print this over serial to check
126.       Serial.print(inByte, BIN);
127.       // create a switch/case to check what byte it is we've recieved (basically a
    fancy if statement)
128.       //if it's true, assign the byte to the relevant row
129.       switch(counter) {
130.         case 0:
131.           row0 = inByte;
132.           break;
133.         case 1:
134.           row1 = inByte;
135.           break;
136.         case 2:
137.           row2 = inByte;
138.           break;
139.         case 3:
140.           row3 = inByte;
141.           break;
142.         case 4:
143.           row4 = inByte;
144.           break;
145.         case 5:
146.           row5 = inByte;
147.           break;
```

```
148.          case 6:
149.            row6 = inByte;
150.            break;
151.          case 7:
152.            row7 = inByte;
153.            break;
154.          default:
155.          //the variable should never exceed 8 if we're only sending 8 bytes,
    especially as counter is reset to 0
156.            Serial.print("This should never happen..");
157.            break;
158.        }
159.        //increment the counter by one
160.        counter++;
161.        //reset the counter to 0 if it ever gets to 8 (though in principle it never
    should)
162.        if(counter == 8){
163.          counter = 0;
164.      }
165.    }
166.    }
```

# References


Anon, 2003. Charlieplexing - Reduced Pin-Count LED Display Multiplexing. *Maxim Integrated*. Available at: https://www.maximintegrated.com/en/app-notes/index.mvp/id/1880 [Accessed June 9, 2017].

Anon, 2011. Largest LED 3D TV. *Guinness World Records*. Available at: http://www.guinnessworldrecords.com/world-records/largest-led-3d-tv/ [Accessed June 9, 2017].

Anthony Edward Green, C.J.O., 1999. Configurable led matrix display. *US Patent*.

Autodesk, 2017. EAGLE: PCB design made easy. *Autodesk*. Available at: https://www.autodesk.com/products/eagle/overview [Accessed June 20, 2017].

Beech, P., 2015. SMD Resistor and Capacitor Book. *Pimoroni Ltd*. Available at: https://shop.pimoroni.com/products/smt-smd-0805-resistor-and-capacitor-book [Accessed June 16, 2017].

Brain, M., 2011. How Jumbo TV Screens Work. *How Stuff Works*. Available at: http://electronics.howstuffworks.com/jumbo-tv2.htm [Accessed June 9, 2017].

'chr', LED Cube 8x8x8. *Instructables*. Available at: http://www.instructables.com/id/Led-Cube-8x8x8/ [Accessed June 9, 2017].

Dee, J., 2013. Shift Registers. *SparkFun Electronics LLC*. Available at:

https://learn.sparkfun.com/tutorials/shift-registers [Accessed June 14, 2017].

Espressif Systems IOT Team, 2015. *ESP8266EX Datasheet*, Espressif Systems Inc. Available at: http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-v4.3.pdf.

Hsiung, B., 2004. LED matrix moldule. *US Patent*.

Instruments, T., 2017. *SN54HC595*, Texas Instruments. Available at: http://www.ti.com/lit/ds/symlink/sn74hc595.pdf.

Jim, 2012. Serial Communications. *SparkFun Electronics LLC*. Available at: https://learn.sparkfun.com/tutorials/serial-communication [Accessed June 14, 2017].

Limited, S.D., 2017. Seeed Fusion PCB. *Seeed Studio*. Available at: https://www.seeedstudio.com/fusion_pcb.html [Accessed June 20, 2017].

Ltd, T.S., 2016. *XC6206 Series voltage regulators*, TOREX SEMICONDUCTOR LTD. Available at: http://www.farnell.com/datasheets/2012663.pdf?_ga=2.227266334.2089807715.1497964990-1799001191.1496076387.

M Armstrong, D Flynn, M Hammond, S Jolly, R Salmon, 2008. *High Frame-Rate Television*, British Broadcasting Corporation.

Nielsen, M., 2013. Multiplexing (for beginners). *Youtube // ECProjects*. Available at: https://www.youtube.com/watch?v=rOAzENzgA8U [Accessed June 9, 2017].

Thornton, J., 2014. *Persistence of Vision Display*, University of Manchester.

'zerodamage', U., 2012. Example of component sizes. *Wikimedia Commons*. Available at: https://en.wikipedia.org/wiki/Surface-mount_technology#/media/File:SMT_sizes,_based

_on_original_by_Zureks.svg [Accessed June 16, 2017].

# Appendix III: How I met the Gold CREST Award Criteria

## Planning

**Set a clear aim and break it down into smaller steps/objectives**

The aim of this project was to 'build a fully functional LED Matrix, on which pre-designed 8 x 8 pixel bitmap images can be displayed'. In order to break this down into manageable steps, the following sub-objectives were created:

- To construct a basic prototype checking that the principle of multiplexing could work
- To construct a schematic diagram in EDA software, using chosen components
- To construct a PCB design from this schematic and order it
- To assemble the PCBs into an operational circuit
- To write test code to run on the microcontroller to blink all of the LEDs
- To design and write a program to run on a computer, to read the image file, convert it into an array of bits, and transmit it to the matrix in an appropriate
- To design and write a program to run on the matrix, to receive the data and display it on the LEDs.

**Explain the wider purpose of your project**

This project is what's behind many displays used in industry - dot matrix displays are commonly found as simple indicators - for example at buses and train stations - because they are simpler, often cheaper, and more reliable than LCD screens. Commonly a character set must be created for each display, which can be somewhat laborious and takes up a large amount of memory. If more of the code could run on the computer, the electronics required to drive each display could be significantly simpler, and it would be easier to use different typefaces as with a little modification a string of images could be passed to the display.

Additionally, LED matrices are behind many modern big screens (at sports stadiums etc). These display videos which are essentially lots of images in sequence, so if this project was made with RGB LEDs on a larger scale it could be used as a big screen with a little modification.

**Consider different ways to do your project**

There are several main methods of controlling the LEDs, which are listed below along with the reasons for choosing or not choosing to use them)

- Controlling each LED individually from a microcontroller pin (laborious and requires powerful microcontroller with a lot of pins)
- Using a dedicated LED driver chip to control each LED (chips are expensive and have certain requirements, was not of enough interest)
- Multiplexing the LEDs and driving them directly from microcontroller pins (still too many pins, and too slow - also a significant amount of current would be required from a single microcontroller pin)

- Multiplexing the LEDs and driving them with two shift registers, the output of one shifting into the input of the other (unnecessarily complex and there were more pins available)
- Multiplexing the LEDs and driving them with two shift registers controlled independently (this was the method used - not sharing pins made the circuit simpler, but 6 pins was still low enough to make it viable)
- Using an alternative method - charlieplexing - a method of multiplexing where some LEDs are connected in reverse to further reduce the input pins (unnecessarily complex for a matrix of this size, and hard to drive without specialist ICs)

**Describe your plan for how to complete your project and give reasons for the approach you chose**
See above in the ‘methodology’ and ‘research’ section, and immediately above for reasoning to choose the approach I used.

**Explain how you planned your time and organised who would do what**
I planned time by setting a deadline for the design and production of the PCBs, which was dictated by the manufacture time and time needed to assemble and program them. I then set deadlines for each section (eg the research would be complete before I began designing the PCBs, and the code would be complete before the PCBs arrived from manufacture, the assembly and troubleshooting would finish the week before the deadline) and ensured I adhered to these. I worked on the project regularly in school and out of school, with extra out-of-school time contributed on an as-needed basis to ensure I met the deadlines.

## Throughout the Project

**Say who and what materials you needed to help you complete your project**
See the ‘Acknowledgements’ and ‘Methodology’ sections above

**Summarise the background research you did to help you understand your project and where you found the information**
See the ‘Research’ section and ‘References’ above

## Finalising the Project

**Make logical conclusions and explain the implications for the wider world**
For conclusion, see the ‘Analysis’ section above. For implications, see the ‘wider purpose of your project’ part of this section.

**Describe how what you did affected the outcome of your project**
See the ‘Evaluation’ section above.

**Explain what you learned and how you would change your project if you did it again**
See the ‘Analysis’ section above.

## Project-Wide Criteria

**Show understanding of the science behind your project**
See ‘research’ and ‘methodology’ above, and presentation.

**Describe how you made sensible decisions about your project. Consider safety and risks**
See ‘Risk Assessment’ above, and ‘Methodology’ for sensible decisions related to science.

**Show creativity in the way you carried out your project**
This has been demonstrated through the design of the circuit board, which is somewhat different to typical circuit boards, and through the methods used in the code.

**Explain how you identified and overcame problems**
Demonstrated in ‘Evaluation’ above

**Explain your project clearly in writing and in conversation**
Seen above and in presentation.