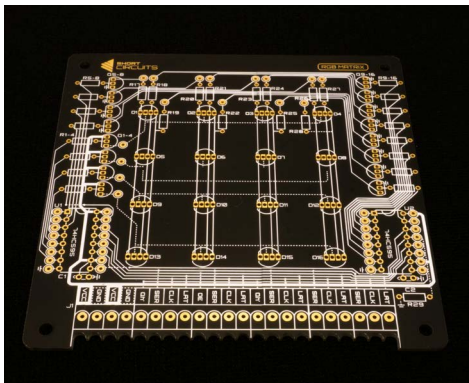




SHORT  
CIRCUITS



# RGB MATRIX

## INSTRUCTION MANUAL

# RGB MATRIX

---

The RGB MATRIX represents a very basic screen. A normal screen is made up of many pixels, all capable of emitting red, green and blue light. A combination of millions of these makes the picture we see on a TV screen or monitor. These screens usually have 1920x1080 (2,073,600) pixels or more. The RGB Matrix has a 4x4 grid of RGB LEDs to represent these pixels. Obviously we won't be streaming Youtube through this. But we can play around with some basic patterns, colour changing, colour mixing and maybe even some scrolling effects.

The RGB Matrix uses the same method of reducing microcontroller pin usage as the DIGITISER. With 2 x 595 Shift Registers, the RGB MATRIX has 16 shift register outputs to play with. As there are 4 x 4 LEDs, each with 3 colours, we need a total of 48 outputs. To solve this problem, we have arranged the LEDs in a matrix. This matrix has 4 rows and 12 columns.

## Contents

---

Circuit

4

Assembly Instructions

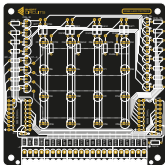
12

Coding Basics

20

Projects

24



1 x Printed Circuit Board

- 16 x RGB LEDs (CA)
- 2 x 595 Shift Registers
- 2 x DIP-16 Sockets
- 2 x 100nF Capacitors
- 4 x PNP Transistors
- 12 x NPN Transistors
- 8 x 150 $\Omega$  Resistors
- 4 x 200 $\Omega$  Resistor
- 4 x 620 $\Omega$  Resistor
- 12 x 820 $\Omega$  Resistor
- 1 x 10K Resistor
- 11 x Screw Terminals
- 4 x 6mm Hex Screws



## Tools Needed

Soldering Iron



Screwdriver

2mm



Solder

0.3 - 0.5mm



Alan Key

2mm



Side Cutters






























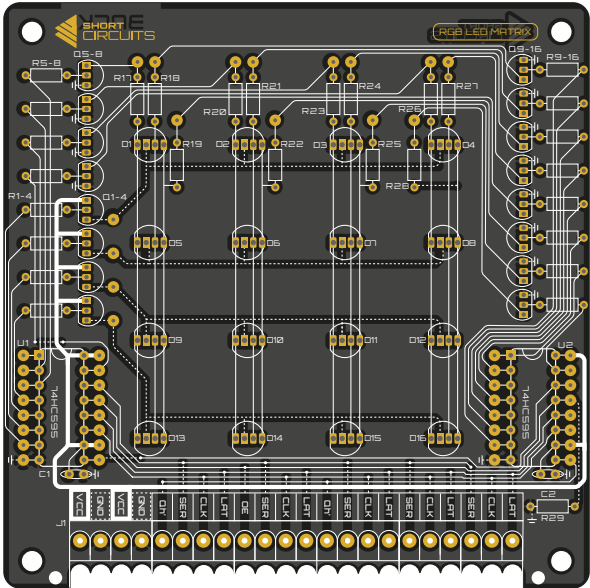
Flexible Wire

20AWG



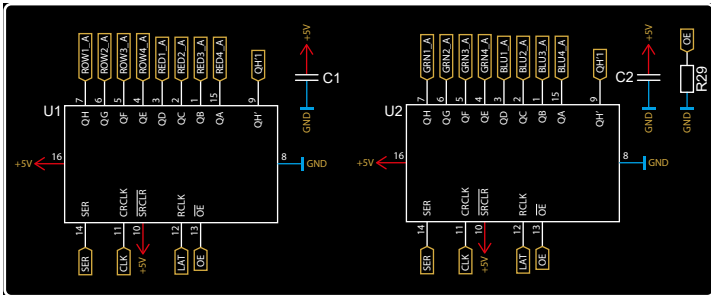
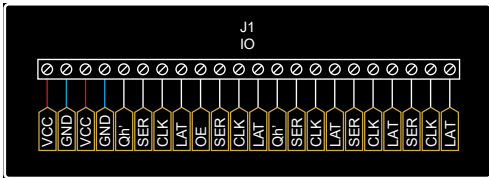
# Circuit - Symbols and Designations

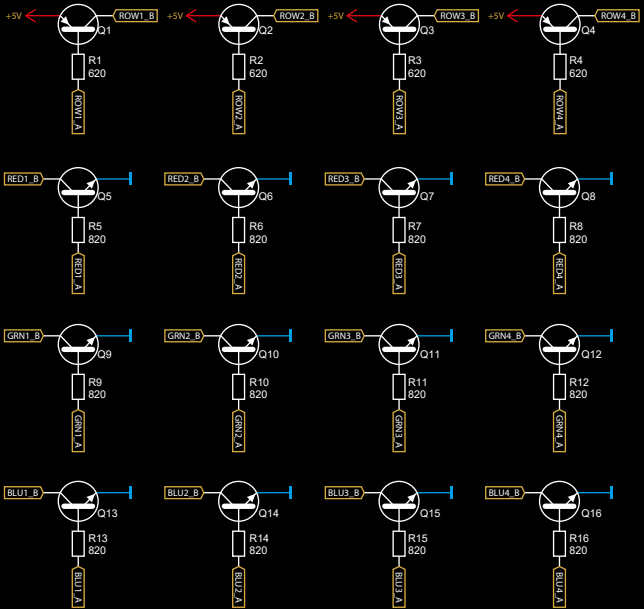
	PCB	SCHEMATIC	DESIGNATION
Copper power trace			
Copper signal trace			
Copper trace on back of board			
Through hole solder pads			
Surface mount solder pads			
Resistor			R
Ceramic capacitor			C
Transistor PNP			Q
Transistor NPN			Q
74HC595 Shift Register			U
RGB LED (common anode)			Q
Screw terminal			J
Connected to VCC			
Connected to GND plane			
Mechanical hole			



Ground (GND) Copper Area On Back of PCB

# Circuit - Schematic






















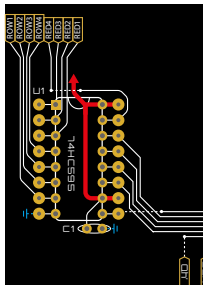
# Bill of Materials (BOM)

Designation	Value	Name	Footprint / Pitch	Datasheet
C1, C2	100nF	Ceramic Capacitor	2.54mm	
D1 - D16	RGB, C. Anode	RGB LED	Ø5mm, 1.3mm	
J1	11 x 2 pos	Screw Terminal	3.5mm	
Q1 - Q4	2N3906	PNP Transistor	TO-92	
Q5 - Q16	2N3904	NPN Transistor	TO-92	
R1 - R4	620Ω, 1/4W, ±1%	Resistor	7mm	
R5 - R16	820Ω, 1/4W, ±1%	Resistor	7mm	
R17, R20, R23, R26	200Ω, 1/4W, ±1%	Resistor	7mm	
R18, R21, R24, R27	150Ω, 1/4W, ±1%	Resistor	7mm	
R19, R22, R25, R28	150Ω, 1/4W, ±1%	Resistor	7mm	
R29	10K, 1/4W, ±1%	Resistor	7mm	
U1, U2	DIP16	IC Socket	DIP16	
U1, U2	74HC595	Shift Register	DIP16	

# Circuit - Shift Registers

## 74HC595 Shift Register

The 74HC595 shift register is a Serial In, Parallel Out (SIPO) shift register. Data in electronics is stored as 1's and 0's which represents a switch in an on or off state. One of these switches is a bit, and 8 bits make a byte. In a SIPO Shift Register data is sent 1 bit at a time into the 8 bit register, these can then be output all at the same time in parallel.



Bits of data are sent from the microcontroller using the shift register's Serial, Clock and Latch pins. First, the Latch pin is turned on ("sent high"), this tells the shift register that data is going to be sent via the Serial pin. The bits are sent to the Serial pin one at a time. Whenever the Clock pin is sent high, the current state of the Serial pin (on or off, 1 or 0) is pushed through the shift register. After all 8 bits are shifted, the Latch pin should go low again, this sends the bits to the outputs. See the component index for a diagram.

## Bypass Capacitors C1, C2

These are decoupling capacitors. Their function is to smooth out any noise (spikes or drops in voltage) due to other components affecting the power supply to the Integrated Circuit. It "decouples" the chip from the noise created by other parts of the circuit. It does this by holding an amount of charge and releasing it when the voltage drops, to compensate. It is placed between power and ground, as close to the IC's (chip's) VCC pin as possible.

## Output pins

The outputs can comfortably output 6mA (milliamp). If they exceed this, Vcc (power supply voltage) will likely drop considerably. To keep current output low, we've used transistor switches for each row and each column of the matrix.

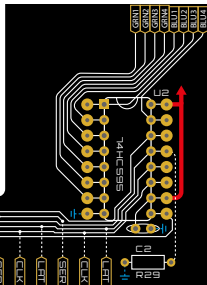
## Output Enable

Output Enable (OE) is used to instantly turn

all outputs off, or on. To enable the outputs, the OE pin must be pulled low. To turn them all off, OE must be pulled high. There is a pull-down resistor connected to OE, which means it is on, until it is pulled high by the microcontroller. If connected to a PWM enabled microcontroller pin, OE can be used to dim the LEDs. Use `analogWrite(0)`; to turn the display on, `analogWrite(255)`; to turn them off. You can use any number in between to dim the LEDs.

## Qh' Serial Output

Several 595 Shift Registers can be chained together using the same 3 or 4 data pins of a microcontroller. Bits are sent from the microcontroller into the board's SER input, then to the SER pin of the first shift register, out the Qh' pin of the first, into the SER pin of the second. The data can then be chained to another shift register (on the Digitiser for example) by connecting the output Qh' terminal (from the second register) to the SER terminal of the next board.



## 5V VCC

The 74HC595 can run on 2 to 6 Volts. So our 5V VCC works fine. The problem with using the 595s is their recommended output of 6mA per pin. This isn't enough to drive several LEDs, so we use the output of the 595 to activate a transistor switch, which in turn lets more current flow safely outside the chip. There are several better shift registers out there, but these teach us some valuable lessons about the limits of components and ways to get around them.

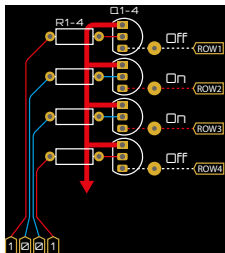
# Circuit - Transistor Switches

## Transistor Switches

A Transistor can be used as a switch to apply higher currents to a different part of the circuit with only a small amount of current to trigger it. This is perfect for our scenario, as the 595 Shift Register can't source enough current to power 12 LEDs on one output pin, or 36 LEDs across 4 pins (our worst case scenario).

The Digitiser cycles through its digits one at a time to show numbers. This essentially quarters the amount of current needed. You may want to turn all the LEDs on at once on the RGB Matrix, so the current would be much higher. Because of this, we have used a transistor on all 16 of the shift register's outputs, instead of just 4 on the Digitiser. We use 4 PNP transistors to source the current for the 4 rows. We then use 12 NPN transistors to help sink the current from each column; 4 red, 4 green and 4 blue.

## PNP Transistors

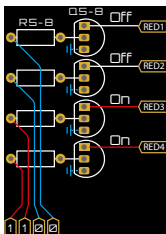


The PNP Transistors source current for each row of LEDs. When a small amount of current flows from the Emitter to the Base of the tran-

sistor, a much greater current will flow from the Emitter to the Collector (see the Component Index for more info).

To enable current to flow from the Emitter to the Base, the shift register's output needs to be LOW (0V). This creates a potential difference between the Base (0V) and the Emitter (5V).

## NPN Transistors



The NPN transistors, when turned on, complete the LEDs circuit to ground. To turn them on, a small amount of current needs to flow from the Base to the Collector. This allows current to flow from the Emitter to the Collector, to ground.

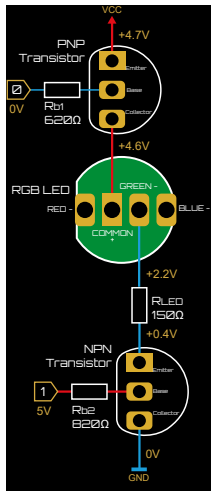
To allow current to flow from the Base to the Collector, a positive voltage needs to be applied. The shift register's output needs to be HIGH (5V). This creates a potential difference between the Base (5V) and the Collector (0V).

## How the circuit works

If we wanted to turn the top left LED green, we would need to do the following:

Set the first shift register's QH output LOW (0V). This is connected to the first row's anodes.

Set the second shift register's QH output HIGH (5V). This is connected to the first column's green cathodes.



When these two things happen, a small amount of current flows from the PNP transistor's Emitter to its Base, and a small amount of current flows from the NPN transistor's Base to its Collector. These currents are limited by

the Base Resistors (Rb1 and Rb2 in the diagram). When these small currents flow, both transistors are in an ON state and larger currents can flow from their Emitters to their Collectors, and through the LED and its current limiting resistor.

### Base Resistor Values

To use a Transistor as a switch, we need to make sure the Transistor is in Saturation Mode. For this we need to apply more than enough current to the Base to ensure the Transistor switches on fully.

To calculate the Base Resistor value, we need to know a few things. Firstly, how much current will flow through the Transistors and the LEDs. We also need to know the Current Gain ( $h_{FE}$ )

We will be using up to 11mA of current per LED, which is controlled by each LED column's current limiting resistor. The worst case scenario for the PNP transistors would be if all columns were on, showing 4 red, 4 green and 4 blue LEDs (which makes white) on a single row. The worst case scenario for the PNP transistors would be if the whole row of 4 LEDs were on.

Worst case for PNP:  $(3 \times 11) + (6 \times 8) = 81\text{mA}$   
 Worst case for NPN:  $4 \times 11\text{mA} = 44\text{mA}$

As the Transistor is an amplifier, the current gain ( $h_{FE}$ ) represents how many times the Base current is amplified (multiplied) by, to equal the Collector current. According to the PNP transistor's datasheet, at -50mA, this would be 60 times. So if we apply a 60th of the Collector Current to the Base, it will turn the transistor on (when the Collector Current is around -50mA).

This would be the minimum amount of current

needed. Any fluctuations in temperature etc would make this unreliable as a switch.

To make sure we turn the transistor on, we can use an  $h_{FE}$  of 10. This means a tenth of the collector current will be applied to the base, ensuring it turns on.

So we know the Collector Current to be around -120mA and we will assume the Current Gain to be 10. We also know that  $V_{CC}$  is 5V. With these values we can calculate the value of our PNP Base Resistors (R1-4).

First we calculate the current needed at the Base.

$$I_B = \frac{I_C}{H_{FE}} \quad I_B = \frac{0.081}{10} \quad I_B = 0.0081\text{A}$$

Then we use Ohm's law to calculate the Resistance needed.

$$R = \frac{V}{I} \quad R = \frac{5}{0.0081} \quad R = 617\Omega$$

So we now know that a 620Ω resistor would result in the Transistor operating in saturation mode, allowing enough current at the Collector for our display, with minimal resistance.

### NPN Transistor

First we calculate the current needed at the Base.

$$I_B = \frac{I_C}{H_{FE}} \quad I_B = \frac{0.044}{10} \quad I_B = 0.0044\text{A}$$

Then we use Ohm's law to calculate the Resistance needed.

$$R = \frac{V}{I} \quad R = \frac{5}{0.0044} \quad R = 1136\Omega$$

From this we can see that a 1100Ω resistor would result in the transistor operating in saturation mode. For some reason we used an 820Ω resistor for these. 820Ω works on an equivalent  $h_{FE}$  of around 13, so we are well under the 60 that is suggested by the NPN transistor's datasheet.

The 595 shift registers have a max output of 30mA per output pin, and a total output of 70mA. Our max of 8.1mA is well within the single output limit. To work out if our shift registers will survive the max current draw across all pins, we can add up the max expected draw on each pin and compare this total with the shift registers max value.

Shift Register 1 (U1) will experience a maximum current of:

$8.1 \times 4 = 32.4\text{mA}$  - PNP outputs  
 $4.4 \times 4 = 17.6\text{mA}$  - NPN outputs

Total = 50mA

Shift Register 2 (U2) will experience a maximum current of:

$4.4 \times 8 = 35.2\text{mA}$  (all NPN outputs)

Total = 35.2mA

Both of these are within our max total current per shift register of 70mA.

# Circuit - LED Matrix

## RGB LED

RGB LEDs have 3 LEDs in a single package. 1 red, 1 green and 1 blue. They come in two varieties, common anode and common cathode. See the [\\_](#) for more info. We are using common anode LEDs.

## LED Matrix

In electronics, a matrix is a grid of connections separated into rows and columns. To turn on a specific node in the matrix, both the row and the column the node belongs to must be turned on. This is useful when we have a limited number of IO pins available on the microcontroller.

There are downsides to using an LED matrix. As an example: If you tried turning on the red LED in D1 (row 1, col 1), and the green LED in D5 (row 2, col 1), you would find that both the red and green LEDs in both D1 and D5 turn on. This would mix the red and green light to make yellow. To achieve different colours in the same column on different rows, we need to refresh the "display" one row at a time. If we do this quickly enough, the human eye won't be able to notice.

## Current Limiting Resistors

As with all LEDs, we need to use a current limiting resistor to protect the LED from burning out. Ideally we would use a resistor for each individual LED. This is because using one resistor for several LEDs can result in different amounts of current flowing through each of the LEDs, causing some to be brighter than others. In practice, using one for each column is more practical and in this application, doesn't have much noticeable negative visual effects.

With the RGB Matrix, we want to have access to maximum brightness without damaging the LED or the transistors. If we use 10mA we can bring the max possible current for each row to 120mA, which is well within the transistor's 200mA max. This is also under the LEDs max current of 20mA, found in the [datasheet](#). The LED's forward voltage is also in the datasheet.

The source voltage is affected by the other components connected in series with the LEDs. This would be the PNP and NPN transistors. These have a voltage drop of 0.3V and 0.4V, which should be taken into account in the calculation.

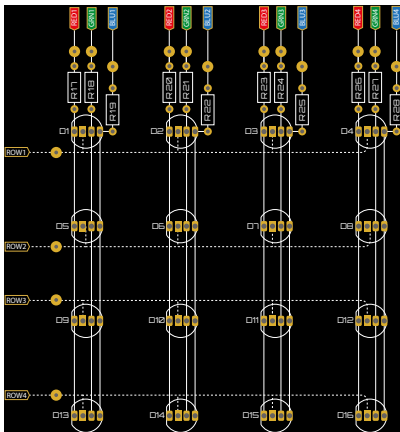
$$R = \frac{V_{\text{source}} - V_{\text{FNP}} - V_{\text{FNN}} - V_{\text{FLED}}}{I\text{F}}$$

$$R = \frac{4.3 - 2.1}{0.01} \quad R = 220\Omega$$

$$R = \frac{4.3 - 3.1}{0.01} \quad R = 120\Omega$$

$$R = \frac{4.3 - 3.1}{0.01} \quad R = 120\Omega$$

We can use the closest common values of 200Ω, 150Ω and 150Ω respectively. This would make the current 11mA for red, 8mA for green and 8mA for blue.



# Assembly Instructions

## General Soldering tips

### 1. ALWAYS KEEP YOUR TIP CLEAN

To ensure the soldering iron can transfer enough heat from it to your solder/component leg you must keep the tip clean and shiny. A dull tip means the outside layer of metal has oxidised. This oxidised layer is a poor transferer of heat. Because of this, you will have to hold the tip against the component for a longer period of time, which can result in the component failing. It's also very frustrating.

To keep your soldering iron tip clean, wipe it on a wet sponge or wire ball, then apply some solder to coat it, then wipe it again. Ideally, you should do this after every component. At the very least, do it after every 4 components.

### 2. CONTACT

When soldering, make sure the tip of your iron is making contact with both the leg of the component and the pad on the PCB. Apply heat to the area, then, within a second or two, apply the solder to the point of contact.

### 3. HEAT

It is better to be too hot than too cold. As mentioned earlier, when the iron tip isn't hot enough, you have to hold it on longer. This allows heat to transfer into the component and could cause a failure. It is better to set your soldering iron a little hot so the solder melts instantly and flows around the leg of the component with ease. You can start at around 350°C and adjust from there.

### 4. SOLDER

Leaded solder is much easier to work with, which makes it easier to learn with. It can be hard to find in some countries, but can often be ordered from China. There are potential health risks, but these are very low. Make sure you have a fan pointing away from your work area to blow the fumes away. Work in a well ventilated area.

Thinner is better. Working with a thick wire of solder can get messy. Use 0.4-0.5mm solder for more control. You will have to feed more into the solder joint, but you have more control when there are other pins close by that you want to avoid. This is essential when soldering surface mount components and Integrated Circuits.

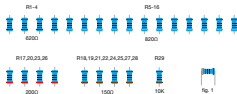
### 5. SAFETY

350°C is obviously very hot. Stuff catches fire at this temperature. Skin fries. Please be careful. Remember to turn it off when you are finished. This will prevent a potential house fire and also save your iron tip from continued oxidation.

### 6. ANGLE OF ATTACK

When soldering, make sure you position the board so that it is easy to access the area you are working on. It is easy to make a mistake when you are trying to maneuver your iron into position around some obstacle. You may have to spin the board 180 degrees, or make sure you have snipped the legs off the previous component. It's easy to be impatient so try to plan ahead.

## Resistors



Bend each leg of the resistor by 90 degrees, as close to the resistor as possible (fig.1). Then insert the legs into the correct holes by finding the correct reference (R1, R2 etc) on the board, or by using the picture on the next page as a guide. Polarity is not important with resistors, so either way is fine.

Hold the resistor in place with sticky tack or tape, then flip the board over, ready for soldering. Solder each leg by holding the soldering iron tip to the leg and the pad, then apply the solder to the joint. Let it flow around the leg and create a shiny cone. Use flush cutters to cut the legs off, then briefly hold the soldering iron to the joint to re-flow the solder.

## Ceramic Capacitors

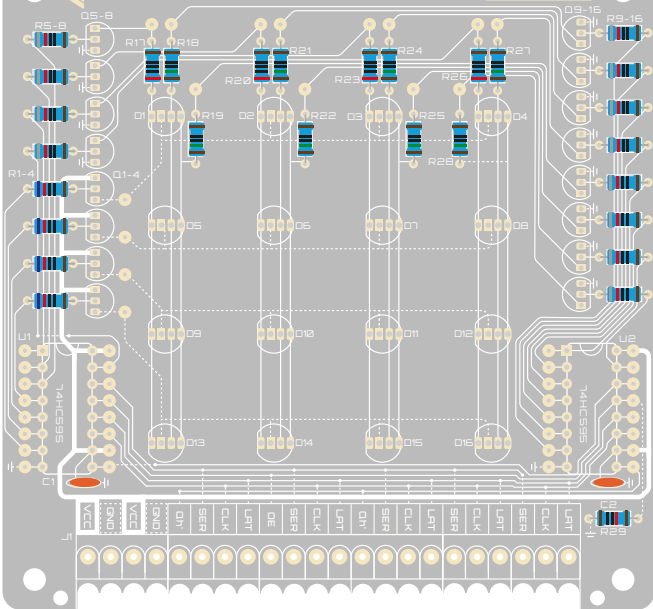


Install the ceramic capacitors into the corresponding holes (C1, C2). They will easily fall out, so add some sticky tack or tape to secure them. Polarity does not matter. Flip the board and solder the pins to the pads, then trim and re-flow.



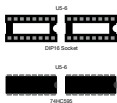
SHORT  
CIRCUITS

RGB LED MATRIX



# Assembly Instructions

## DIP16 Sockets



We are going to solder a socket to the PCB rather than the chip itself. We do this to avoid heating up the IC and to enable us to swap it out if it stops working.

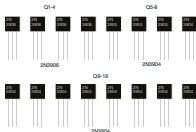
Insert the sockets into the areas marked U1 and U2. Pay attention to the semi-circle indicator and match the one on the socket with the one on the board.

Now, stick it in place with sticky tack and solder 1 corner pin, then the diagonally opposite corner. Now check alignment and whether its flush with the board. If one corner is too high, add pressure to it while heating the pin underneath.

Now you can solder all the other pins knowing that the socket will stay exactly where it needs to be. Remove the sticky tack before soldering the rest of the pins. As with the Transistors, if you accidentally bridge two of the pins, wipe your clean iron tip through the center of them until they are separated.

**DO NOT INSERT THE CHIPS UNTIL YOU HAVE COMPLETED THE TESTING FOR FAULTS SECTION!!!!**

## Transistors (PNP & NPN)



The transistors can be tricky to solder because the legs are so close together. We have spaced the pads out a little to make it easier. Notice the semi-circular shape of the transistor, looking at the top. Match this with the footprint on the PCB.

The outer legs of the transistor will spread out a little as you insert it into the holes. Don't push it too far as the legs will spread too much and crack the black casing of the transistor. Stop when there's a 2-3mm gap. Add sticky tack to hold in place, flip, solder, snip and re-flow.

If you accidentally bridge two pads together, clean your iron, re-tin the tip, then drag the tip through the gap between the pads. This should separate them. If it doesn't, repeat the process. If your tip isn't clean, the solder won't stick to the tip when you drag it through, so make sure that tip is clean and re-tinned.

## Terminal Blocks



The terminal blocks are interlocking, so slide each of them together to make a chain of 11. To make this line of terminal blocks fit

in their holes, you may need to squeeze the blocks together a bit.

If you are using the Rack case to display your boards, I would suggest clipping the legs off. These can interfere with the wires and cause a short when they are sandwiched between the case and the board.

## RGB LEDs

The RGB LEDs are similar to the transistors. Their pins are close together, so we spread the pads out to make it easier to avoid solder bridges. Just like the transistors, be careful when inserting them. They will push all the way to the board, but the casing will crack in the process. This shouldn't stop them from working, but a 2-3mm gap between the board and the LED casing is preferable.

I would solder 1 row or column at a time, and make sure they are all at the same height. Use tack or tape to keep them in place. Tack is better here as the tack will help keep the LED from listing to one side if wrapped around the whole LED. You could use strips of tack placed either side of the row or column to help keep them in line.

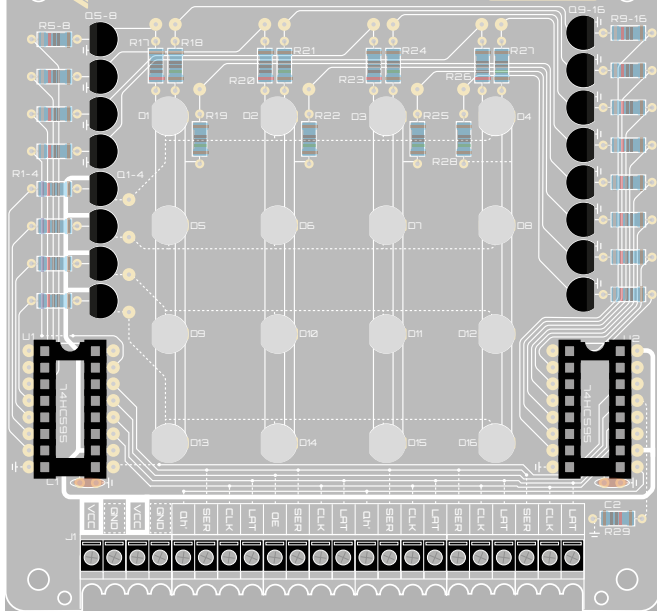
Flip the board when the LEDs are secure and solder like you did the transistors. Snip the leads and reflow the solder. Check for any bridges and deal with them like before.

J1



SHORT  
CIRCUITS

RGB LED MATRIX



50

### Visual Check

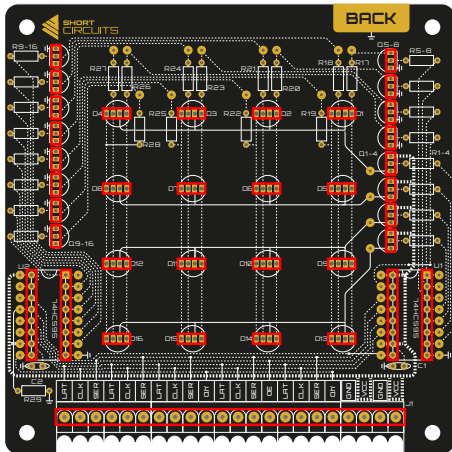
If you see any solder bridges, bring your iron up to temperature and drag it between the two pads. You may have to repeat this a few times. Make sure your iron is clean or the solder won't cling to it. Add flux to aid the process.

## Short Circuits

If you here a constant sound from your meter when the probes are in contact with Vcc and GND then you have a short. Check the back of the board again for solder bridges. Focus on the areas marked with boxes on the diagram on the next page. This is where Vcc and GND are closest.

## Polarity

Check all the components that are polarity dependent. In this circuit, that would be the Transistors, Shift Registers and LEDs. The shape of the transistor should match the silkscreen (white markings) on the board. The shift registers should have their semi-circle and pin 1 at the top. The LEDs flat edge should match the silkscreen below it. We can now insert the shift registers into their sockets. Making sure that the semi-circle is pointing towards the top of the board.



### Powering the Board

We can now connect the RGB Matrix to the Motherboard. Connect VCC, and GND to the Motherboards Power Out terminals. Check that the board is receiving 4.5-5V by switching the multimeter to DC Voltage ( ) and putting the probes on the same points as last time.

### Testing the Various Functions

Now connect SER, CLK, LAT and OE to the

Motherboard's D3, D4, D5 and D6 respectively. OE needs to be connected to a PWM pin in order to dim the display. The other connections can use any pin. Remember to change the pin declarations in the code if you use different pins. To test the function of the board, upload [this test sketch](#) (if the sketch opens in the browser, right click on the page and click save as. Delete the .txt at the end of the file name to leave .ino). When the sketch is uploaded, the display should show all red LEDs, then all green, then all blue, fading out of each colour in turn.

### Troubleshooting:

1 or 2 LEDs show a different colour:

- Check those LEDs for solder bridges.

A whole row is acting strangely:

- Check the PNP Transistor connected to that row for solder bridges, orientation or damage.
- Check the output pin of the Shift Register that controls that row for bridges.

A whole column of a specific colour is acting strangely:

- Check the resistor for damage
- Check the transistor for solder bridges, orientation and damage.
- Check the relevant Shift Register's output for bridges.

The board is powered and reading around 4.8V and nothing is happening:

- Check the Shift Registers are inserted correctly. A leg may have gotten bent, or it may not be seated properly. If they were inserted the wrong way round, you should have noticed the smell by now...
- Check the code uploaded successfully.
- Does the Motherboard function correctly? Test the Motherboard.
- Try different pins and change the code accordingly.

**Still having problems? Head over to the forums on our website for help and support. [www.shortcircuits.cc](http://www.shortcircuits.cc)**

# Coding Basics - All Rows & Fading

```
1  const int SER = 3; // Shift Register Serial Data pin
2  const int CLK = 4; // Shift Register Clock pin
3  const int LAT = 5; // Shift Register Latch pin
4  const int OE = 6; // Shift Register Output Enable pin
5
6  void setup()
7  {
8      pinMode(OE, OUTPUT);
9      pinMode(SER, OUTPUT);
10     pinMode(CLK, OUTPUT);
11     pinMode(LAT, OUTPUT);
12
13     analogWrite(OE, 0);
14 }
15
16 void loop()
17 {
18     digitalWrite(LAT, LOW);
19     shiftOut(SER, CLK, MSBFIRST, B00000000);
20     shiftOut(SER, CLK, MSBFIRST, B00001111);
21     digitalWrite(LAT, HIGH);
22
23     delay(100);
24
25     digitalWrite(LAT, LOW);
26     shiftOut(SER, CLK, MSBFIRST, B11110000);
27     shiftOut(SER, CLK, MSBFIRST, B00000000);
28     digitalWrite(LAT, HIGH);
29
30     delay(100);
31
32     digitalWrite(LAT, LOW);
33     shiftOut(SER, CLK, MSBFIRST, B00001111);
34     shiftOut(SER, CLK, MSBFIRST, B00000000);
35     digitalWrite(LAT, HIGH);
36
37     delay(100);
38
39     digitalWrite(LAT, LOW);
40     shiftOut(SER, CLK, MSBFIRST, B01000010);
41     shiftOut(SER, CLK, MSBFIRST, B00001001);
42     digitalWrite(LAT, HIGH);
43
44     delay(100);
45 }
```

These code examples and explanations will test all of the RGB Matrix's functions and teach you the basics. We will cover the following:

- Show single colour on all LEDs
- Show different coloured columns
- Fade the display

As always, you will need to open the Arduino IDE and either open a new sketch or download the sketches from the links provided.

The first sketch simply cycles through red, green and blue on all LEDs.

First we state the pins used to connect the RGB Matrix to the Motherboard. Here we use 3, 4, 5 and 6. D6 being a PWM pin which is needed to dim the display.

We then declare the 4 pins as outputs within the setup() function. We also write 0 to the OE pin to make sure the shift registers outputs are enabled.

Shifting data to the shift registers is pretty simple. First we need to set the Latch pin low so the register can accept data (line 71). We learned how to set a pin high or low in the Motherboard manual, so this should be easy.

We then use the function shiftOut which is a function written into the Arduino IDE to help us interface with shift registers. The code is actually pretty simple. It shifts out a byte of data, one bit at a time. Each bit is shifted into the register through SER after each clock pulse is sent through CLK.

shiftOut requires 4 things to function. It needs the Serial data pin, the clock pin, either MSBFIRST (Most Significant Bit First) or LSBFIRST (Least significant Bit First) followed by a byte of data (8 bits).

The first two are easy, we can just add the names we assigned for those pins.

MSBFIRST and its counterpart determine which order the bits are shifted into the register. The most significant bit is the one with the most value, which is the one on the left. The least significant bit is on the right. The first bit in will end up on the last output of the shift register. You can use either, but the bit order does matter.

We have 2 shift registers chained together, that's why we are using shiftOut twice. The first byte will end up in the second shift register, with the first bit on the last output. The second byte will end up in the first shift register, with the first bit in landing on the last output.

MSBFIRST:

	Green Col 1	Green Col 2	Green Col 3	Green Col 4	Blue Col 1	Blue Col 2	Blue Col 3	Blue Col 4
B	1	1	1	1	1	1	1	1

	Row 1	Row 2	Row 3	Row 4	Red Col 1	Red Col 2	Red Col 3	Red Col 4
B	0	0	0	0	1	1	1	1

The first Byte controls the Green columns followed by the Blue columns. The second Byte controls the Rows followed by the Red columns. LSBFIRST would reverse the order of the bits.

LSBFIRST:

	Blue Col 4	Blue Col 3	Blue Col 2	Blue Col 1	Green Col 4	Green Col 3	Green Col 2	Green Col 1
B	1	1	1	1	1	1	1	1

	Red Col 4	Red Col 3	Red Col 2	Red Col 1	Row 4	Row 3	Row 2	Row 1
B	1	1	1	1	0	0	0	0

Because the PNP transistors are turned on when the output is 0V, the rows are turned on with a 0. The columns are turned on with a 1. The previous tables show all the outputs in an on state.

The first shiftOut sequence turns all the red columns on and all the rows on. We then delay for 100 milliseconds. The shiftOut sequence is then repeated with the green columns then the blue columns.

The last shiftOut sequence illustrates the limitations of the matrix. When can display different colours on each row, but we cannot display different colours on the same column. Without cycling through the rows the rows one at a time that is.

To fade the display from 100% to 0% we can use a for loop that increments what we write to OE by 1 each time it loops. To fade from 0% to 100% we would change the for function's variables to: (int i = 255; i>0; i--).

The dimming will apply to the last bits shifted out before the for loop. You could nest the shiftOut sequences within the for loop if you needed to dim and cycle through the rows at the same time. You could fade out of one sequence/colour to another by using a different analogWrite() before each sequence. One incrementing and one decrementing. There are lots of ways to manipulate this. Have a go and see what you can come up with. The [forum](#) will have some different adaptations, so check it out.

```

16 void loop()
17 {
18   digitalWrite(LAT,LOW);
19   shiftOut(SER,CLK,MSBFIRST,B00000000);
20   shiftOut(SER,CLK,MSBFIRST,B00001111);
21   digitalWrite(LAT,HIGH);
22
23   for (int i=0; i<255; i++)
24   {
25     analogWrite(OE,i);
26     delay(10);
27   }
28 }

```

# Coding Basics - Cycling through the rows

If we want to be able to display patterns and animations with different colours in the same column, we need to cycle through each row in quick succession.

This sketch displays a static pattern as the loop runs through each row again and again. As you can see, the first shiftOut sequence turns the first row and any colours we want on. The second sequence turns the second row on with the colours we want in the second row and so on.

What if we wanted to show this pattern for a few seconds, then a different pattern for another few seconds. Could we just add a longer delay, then repeat the code with the new pattern? No, unfortunately it isn't that simple. The longer delay would only apply to the last row of that pattern. Try it and see.

To achieve what we want, we can use a for loop to contain each pattern, or frame. The number of times the for loop loops determines the length of time the pattern/frame is shown. You can string lots of for loops together to make an animation. The Code on the next page shows the void loop() of a simple 2 frame animation. Play around with the bits and the number of for loops to make your own animation.

```
1  const int SER = 3;  // Shift Register Serial Data pin
2  const int CLK = 4;  // Shift Register Clock pin
3  const int LAT = 5;  // Shift Register Latch pin
4  const int OE = 6;  // Shift Register Output Enable pin
5
6  void setup()
7  {
8      pinMode(OE, OUTPUT);
9      pinMode(SER, OUTPUT);
10     pinMode(CLK, OUTPUT);
11     pinMode(LAT, OUTPUT);
12
13     analogWrite(OE, 0);
14 }
15
16 void loop()
17 {
18     digitalWrite(LAT, LOW);
19     shiftOut(SER, CLK, MSBFIRST, B000001001);
20     shiftOut(SER, CLK, MSBFIRST, B01110110);
21     digitalWrite(LAT, HIGH);
22     delay(2);
23
24     digitalWrite(LAT, LOW);
25     shiftOut(SER, CLK, MSBFIRST, B00000110);
26     shiftOut(SER, CLK, MSBFIRST, B10111001);
27     digitalWrite(LAT, HIGH);
28     delay(2);
29
30     digitalWrite(LAT, LOW);
31     shiftOut(SER, CLK, MSBFIRST, B00000110);
32     shiftOut(SER, CLK, MSBFIRST, B11011001);
33     digitalWrite(LAT, HIGH);
34     delay(2);
35
36     digitalWrite(LAT, LOW);
37     shiftOut(SER, CLK, MSBFIRST, B000001001);
38     shiftOut(SER, CLK, MSBFIRST, B11100110);
39     digitalWrite(LAT, HIGH);
40     delay(2);
41 }
```

```

16 void loop()
17 {
18   for (int i = 0; i < 100; i++)
19   {
20     digitalWrite(LAT,LOW);
21     shiftOut(SER,CLK,MSBFIRST,B00001001);
22     shiftOut(SER,CLK,MSBFIRST,B01110110);
23     digitalWrite(LAT,HIGH);
24     delay(2);
25
26     digitalWrite(LAT,LOW);
27     shiftOut(SER,CLK,MSBFIRST,B00000110);
28     shiftOut(SER,CLK,MSBFIRST,B10111001);
29     digitalWrite(LAT,HIGH);
30     delay(2);
31
32     digitalWrite(LAT,LOW);
33     shiftOut(SER,CLK,MSBFIRST,B00000110);
34     shiftOut(SER,CLK,MSBFIRST,B11011001);
35     digitalWrite(LAT,HIGH);
36     delay(2);
37
38     digitalWrite(LAT,LOW);
39     shiftOut(SER,CLK,MSBFIRST,B00001001);
40     shiftOut(SER,CLK,MSBFIRST,B11100110);
41     digitalWrite(LAT,HIGH);
42     delay(2);
43   }
44 }

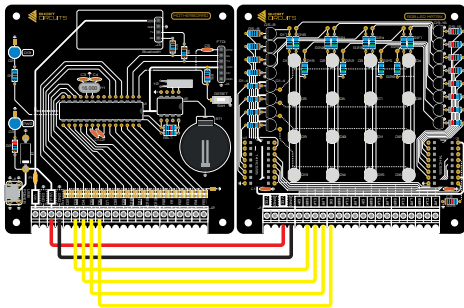
```

```

45 for (int i = 0; i < 100; i++)
46 {
47   digitalWrite(LAT,LOW);
48   shiftOut(SER,CLK,MSBFIRST,B00000110);
49   shiftOut(SER,CLK,MSBFIRST,B01110001);
50   digitalWrite(LAT,HIGH);
51
52   delay(2);
53
54   digitalWrite(LAT,LOW);
55   shiftOut(SER,CLK,MSBFIRST,B00000100);
56   shiftOut(SER,CLK,MSBFIRST,B10110110);
57   digitalWrite(LAT,HIGH);
58
59   delay(2);
60
61   digitalWrite(LAT,LOW);
62   shiftOut(SER,CLK,MSBFIRST,B00000100);
63   shiftOut(SER,CLK,MSBFIRST,B11010110);
64   digitalWrite(LAT,HIGH);
65
66   delay(2);
67
68   digitalWrite(LAT,LOW);
69   shiftOut(SER,CLK,MSBFIRST,B00000110);
70   shiftOut(SER,CLK,MSBFIRST,B11101001);
71   digitalWrite(LAT,HIGH);
72
73   delay(2);
74 }
75 }

```

# Project Ideas - Light Alarm



This can be placed on your desk ready to remind you when to stop working for the day, or go to lunch, or to stretch your legs. The project utilizes the DS1307 Real Time Clock on the Motherboard to set a reminder at certain times of the day. The RGB MATRIX flashes a certain colour to remind you to do a certain task.

Connect the RGB MATRIX up to the MOTHERBOARD as shown in the diagram. We recommend 20AWG flexible stranded wire. If you are going to stack the boards without a case, then the wires need to be as short as possible. A good way to manage this is to insert a wire into one of the

screw terminals on the lower board, then stack the boards using the provided stand-offs. Cut the wire to length while offering it up to the screw terminal directly above it. Now remove the sheathing from the tip of the wire and push it firmly into the screw terminal. Make sure none of the strands of wire are hanging out the edges. Remove any extra slack if the wire is too long. If you are happy with the length, you can use it as a guide when cutting all the other wires.

If you are going to panel the boards, or stack them in a case, then leave a decent amount of slack. Forward planning is key here. Assemble it without wires, and take

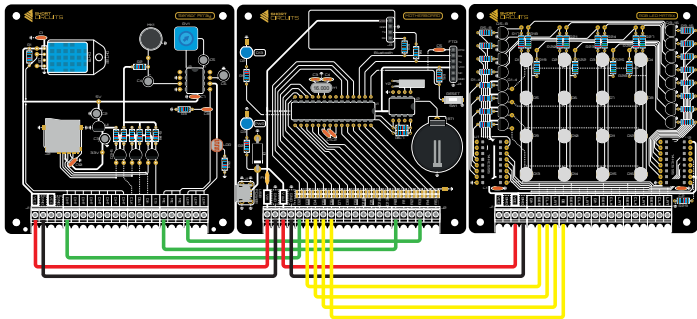
some measurements to make sure.

Once connected, head over to the [forums](#) and download the LightAlarm sketch and upload it using the Arduino IDE and the instructions in the Programming section of the Motherboard's manual.

Remember to put a battery in the holder (BT1) of the Motherboard so the DS1307 can remember the time if you unplug the power.



# Project Ideas - Sensor Bar Display



Adding the SENSOR ARRAY to the previous build allows you to show temperature, relative humidity, sound and light levels on the RGB MATRIX as bars.

If you are stacking these, you will get better results if the Sensor Array faces outwards. This will expose the DHT11 sensor to the outside world, and minimise the effect of heat from the other boards on the sensor. If the board is flipped, it may make sense to move the connections around a bit, for neatness and to use less wire. Make sure you change the pin references at the beginning of the code if you do this.

Wire the boards as shown in the diagram

above. You can switch outputs on the SENSOR ARRAY and RGB MATRIX as long as you switch to an IO port that has the same label. Switching outputs on the MOTHERBOARD is fine, just make sure the Potentiometer is connected to an analog pin, and the Output Enable pin is connected to a PWM pin. Change the code to match these changes.

Either write your own code from the hints in each kit's manual, or head over to the [forums](#) and grab the Sensor Bar Display sketch. Upload the sketch using the Arduino IDE and make sure all the functions work. If they don't, check the pin references at the beginning of the code.

Play around with the variables and code to better understand it. Then you can make changes to suit your requirements.