Thank you for purchasing the KM1097 kit from Jaycar.

These build instructions have been extracted from Issue 34 (May 2020) of the DIYODE Magazine. Please review the instructions end to end before starting construction on your LED cube.

The code and 3D print files for the optional enclosure can be found at: https://diyode.io/034wydx

# QUADLED$^3$

## PROGRAMMABLE 4 × 4 × 4 LED CUBE

**Watching 64 LEDs in a cube dance to different patterns can be extremely satisfying. We show you how to build one for yourself.**

**BUILD TIME: 8 HOURS**
**SKILL LEVEL: INTERMEDIATE**

**JOHANN WYSS**
Staff Technical Writer

*Disclaimer: Colour may vary to the one shown. Content can change without prior notice.*
*Please visit the website page for the most up-to-date information.*

**This is the first of a few LED cube projects we're developing in conjunction with our friends at Jaycar. While all parts are still freely available as with all DIYODE projects, you should be able to find kits for these projects in Jaycar stores very soon.**

LED cubes, also known as LED Matrix, can be an amazing feature in a bedroom, study or Maker space. The lighting patterns they produce are captivating, and provide a really interesting ambiance to the room.

Building one can also be a constant reminder of your accomplished soldering skills because soldering 64 LEDs into straight lines is no easy feat. Not only that, your cube will also be a great way to show off your Maker prowess to your friends and family.

We'll show you how to make your own LED cube. We've kept the build as easy as possible, however, you will need to be competent with soldering and be familiar with programming an Arduino Uno.

## THE BROAD OVERVIEW

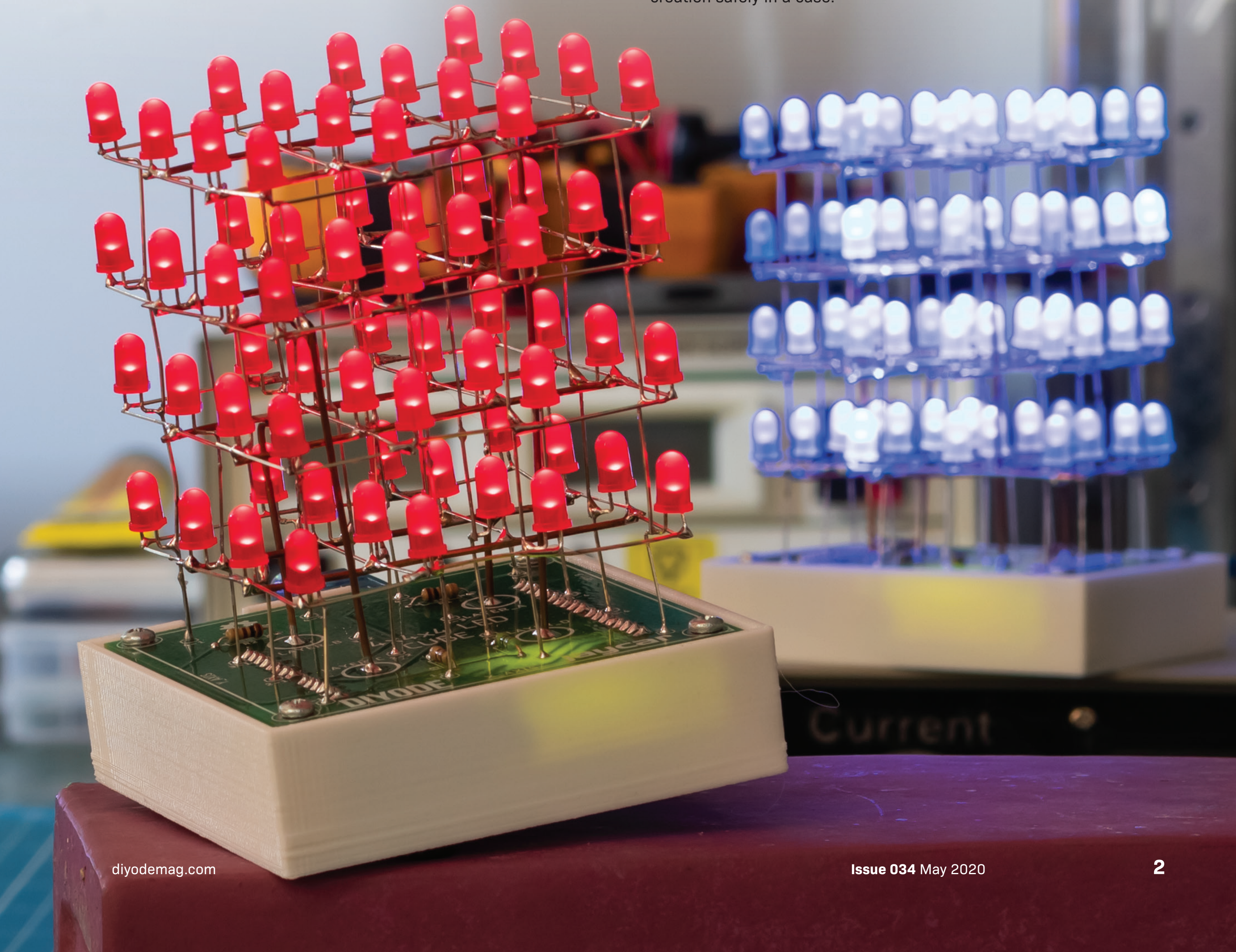
This project turns sixty-four LEDs, a handful of components, and an Arduino Uno into an animated 4 x 4 x 4 LED cube.

We guide you step-by-step on how to construct the four layers of sixteen LEDs, and provide techniques to keep the LEDs straight, evenly spaced, and soldered correctly in place.

An Arduino Uno microcontroller forms the brains of the operation, which you can program with our prepared code that provides fifteen animations. We will also show you how you can code your own LED animations.

We have provided an option for using either blue or red LEDs depending on your budget and colour preference, and also go into details about limitations if you plan to use different LEDs.

A custom PCB makes the project easy to build and avoids a lot of messy wiring that perf-board type builds involve, and we round out the project with an optional 3D printed base if you want to keep your creation safely in a case.



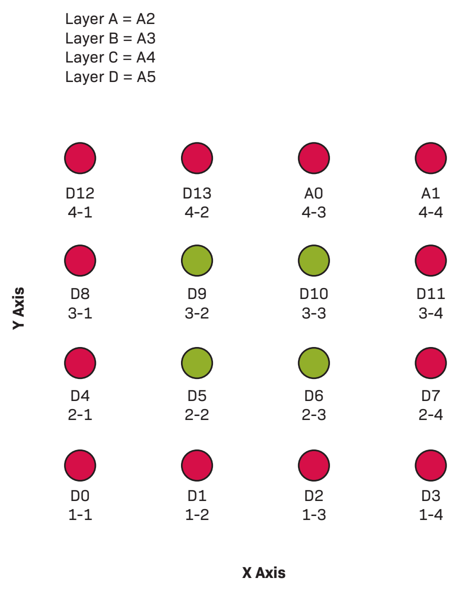
## WHAT YOU WILL NEED

You can find a list of the electronic parts required in the build section, which includes options for red or blue LEDs.

You will need all the usual Maker essentials, including a soldering iron, solder, side cutters, long nose pliers, large tweezers, and a lot of patience. An LED tester will also come in handy to test the LEDs before you solder them into the matrix.

Make sure you have a clean work area with plenty of room. The kitchen table may not be the best place because this build may take several hours over a few days to put together.

To help with construction, we have included 3D print file for a LED jig to hold the LEDs while you solder them. If you don't have access to a 3D printer you can use the supplied template to drill 5mm holes in some wood. These files can be downloaded from our website, and are described in more detail in our build section.



## HOW IT WORKS

It is important to understand how the project works before you heat up the soldering iron and start making.

### THE CONTROLLER

The controller used in this project is an Arduino Uno compatible microcontroller development board. It utilises the Atmel ATmega328P microcontroller and has a total of 20 x General Purpose Input and Output pins (GPIO). These pins consist of 14 x digital and 6 x analog pins, all of which can be (and will be) used in this project as digital outputs.

### THE ARRAY

The 4 x 4 x 4 LED cube uses a total of 64 LEDs. These LEDs are arranged in a 4x4 grid per layer with 4-layers in total. Each of the LEDs in a layer is configured in the common anode configuration, meaning all their positive pins are tied to the same potential. You're no doubt asking yourself how we can control 64 LEDs using only 20 GPIO pins on the microcontroller. The good news is this isn't very complex of a task.

We need 16 pins to control the 16 LEDs on each layer. These layers are all tied together via their positive pins. If we apply 5V (via a resistor) to the positive pin, and pull the cathode of any of the 16 LEDs to ground, current can flow through the LED to illuminate it. The remaining 4 pins can be used to provide the 5V supply to the layers. We can then program the device around this limitation.

The tricky part comes when you want different LEDs illuminated on different layers. Using this pin map, let's say we wanted the 12 outer LEDs illuminated on one layer and the four inner LEDs illuminated on the second.

To illuminate the outer 12 LEDs, you need to pull the corresponding pins (shown in red) low for the following coordinates, using the key:

| | |
|---|---|
| XX | = Arduino pin |
| (Y-X) | = (LED coordinate) |
| Pins on y = 1: | (1-1), (1-2), (1-3), (1-4) are D0, D1, D2, D3. |
| Pins on y = 2: | (2-1), (2-4) are D4 and D7. |
| Pins on y = 3: | (3-1), (3-4) are D8 and D11. |
| Pins on y = 4: | (4-1), (4-2), (4-3), (4-4) are D12, D13, A0, A1 |

With those pins pulled Low (ground potential), if we pulled the layer pin A High (Analog pin 2) current can flow out of the microcontroller pin, through the current limiting resistor, and through any of the LEDs that have their anode pulled low, making them illuminate. To illuminate the middle square, we need to pull D5, D6, D9, and D10 Low, while pulling Layer B (A3) High.

However, if we were to have Layer A and Layer B High, while all 16 of the LED pins pulled LOW, then every LED on layer A AND layer B will be illuminated. To overcome this, you need to rapidly set the pins required Low, pull the layer High and wait a small amount of time before pulling the layer Low, changing the pins to suit the next layer and pulling that layer High and waiting a small amount of time (tens of milliseconds). By repeating this very quickly, it will trick the eye into thinking that different LEDs on different layers are illuminated at the same time when, in fact, only one layer at any given time is.

To see a demonstration of this principle, you can check out our twinkle() function. In this function, we assign a random value between 0 or 1 to four 16 element integer arrays.

These arrays represent each of the four layers. Using a For loop, we use the random 1's or 0's stored in the arrays to set a digital pin High or Low. We then pull the layer pin High, before moving on to the next layer.

We use a For loop to repeat the entire process 500 times, and the result is a random pattern of illuminated LEDs on each layer.

Of course, it's much simpler to just choose the animation that matches the limitation. For the most part, the cube is animated so that we only need to manipulate one variable at a time, either the X-Y domain (i.e. the LED pins) or the Z axis (i.e. the layers). This is much easier to program and produces a more vibrant looking animation, as the LEDs are not pulsing on and off reducing their overall brightness.

We will discuss how we programmed the LED cube in more detail in the code section, which will give you a good idea of how you can create your own animations.

## CURRENT LIMITATIONS

Our red and blue LED cube builds have been successfully running without any issues, but it is worth pointing out that if you plan to use LEDs that draw more current, then you may damage the Uno board.

According to the datasheet from Atmel, the ATmega328P microcontroller used on the Arduino UNO has an absolute maximum current source of 40mA for any individual pin. Exceeding 40mA may damage the microcontroller.

Best practice would be to include transistors on every layer and have individual resistors to every LED, but a design like this would be unnecessarily complex for the task.

Potentially, with all four layers on, the same pixel on each layer and only those four LEDs illuminated, it's possible to exceed the maximum current of the digital pin. However, we tested this in a practical application using 220Ω resistors, and the current did not exceed 40mA. The current we measured was a constant 34.84mA.

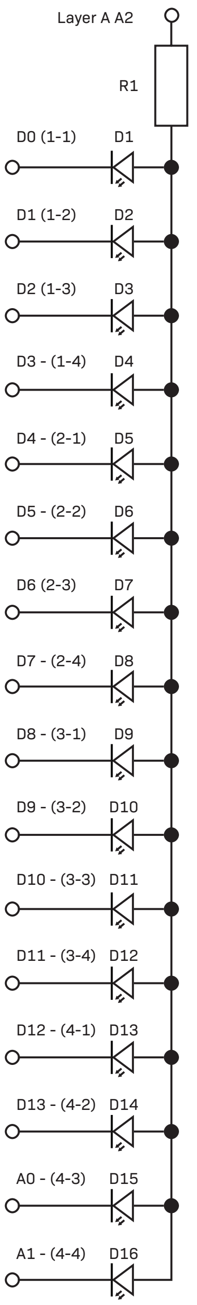
Provided that the LEDs you use have a minimum forward voltage of 2V, you only have a slim chance of damaging the LEDs. However, if you're in any doubt, or want to use LEDs outside of this range, increasing the size of the resistors on the analog pins will reduce the current further.

Another way to avoid any risk of damage would be to program the cube so that no more than three layers can be on at one time if the same LED, and only that LED, is illuminated on each layer.

## PICKING THE LEDS

The LED layers in our project are configured like you see here with a current limiting resistor connected to 16 LEDs in parallel.

Using a single resistor in series with a parallel string of LEDs is, generally speaking, is not best practice. Each LED may have slightly different characteristics, such as forward voltage (Vf). Therefore, with LEDs in parallel, the LED with the lowest Vf will sink more current than the others. This could shorten that LED's lifespan. However, to keep the circuit simple and affordable, provided the current limiting resistor on the layer pin limits the current to less than the LED's absolute maximum forward current, the risk of damage to the LEDs is minimal.



### DIFFERENCES BETWEEN LED COLOURS

We tried this project with two different LED colours. The first LEDs were a standard red LED from Jaycar, which provides a great low-cost option, and blue LEDs, which are more expensive, but produce a much more vibrant output.

### RED LEDS

The red LEDs we used had a forward voltage (Vf) of 2V and a forward current (If) of 20mA, producing 8 millicandelas (mcd) intensity. Red LEDs are the cheapest of all the colours available, however, they are not very bright when set up as a parallel string of 16 LEDs with a 220Ω resistor (powered with 5V).

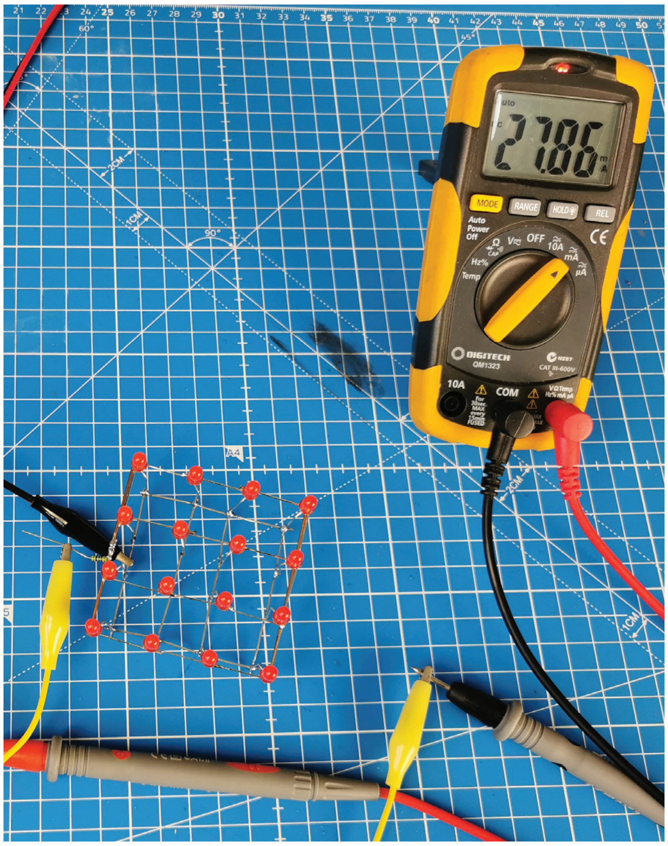
$$I = \left(\frac{\frac{V}{R}}{\text{\# of LEDs}}\right) = \frac{\frac{(V_{in} - V_f)}{R}}{16} = = \left(\frac{\frac{(5_v - 2_v)}{220}}{16}\right) = 850\mu A$$

Thus, when all 16 LEDs on a layer are illuminated, each LED is only getting about 850µA, less than 5% of its forward current. Therefore, it's likely producing less than 5% of its very modest 8 millicandela intensity. Whereas, if only one LED in the string was illuminated it would get the full current.

$$I = \frac{V}{R} = \frac{5_v - 2_v}{220} = 13.6mA$$

This is less than the absolute maximum of that LED, meaning it won't be destroyed and will be reasonably close to its absolute brightness. With that said, in practice, the difference in apparent brightness between a few LEDs per layer and all 16 LEDs per layer illuminated, is not all that apparent when the project is running.

We confirmed this calculation by building a test layer and measuring the current into the circuit.



We left this test running on 5V for the duration of this project, and we saw no obvious reduction in output brightness (At the time of writing, it has been over 14 days or roughly 330 hours).

When we measured the voltage across the LEDs in this configuration, we measured 1.88V, showing the forward voltage of the LEDs was not being exceeded, and no LED was seen to have an obvious difference in its level of intensity.

*Note: This experiment was done using red LEDs from Jaycar using our QM1323 digital multimeter (DMM).*

**BLUE LEDS**

For a more vibrant display, we tried the project using blue LEDs. These LEDs we purchased from Jaycar had a forward voltage of 3.3V, meaning that in the very worst-case scenario, each LED would only get about half a milliamp.

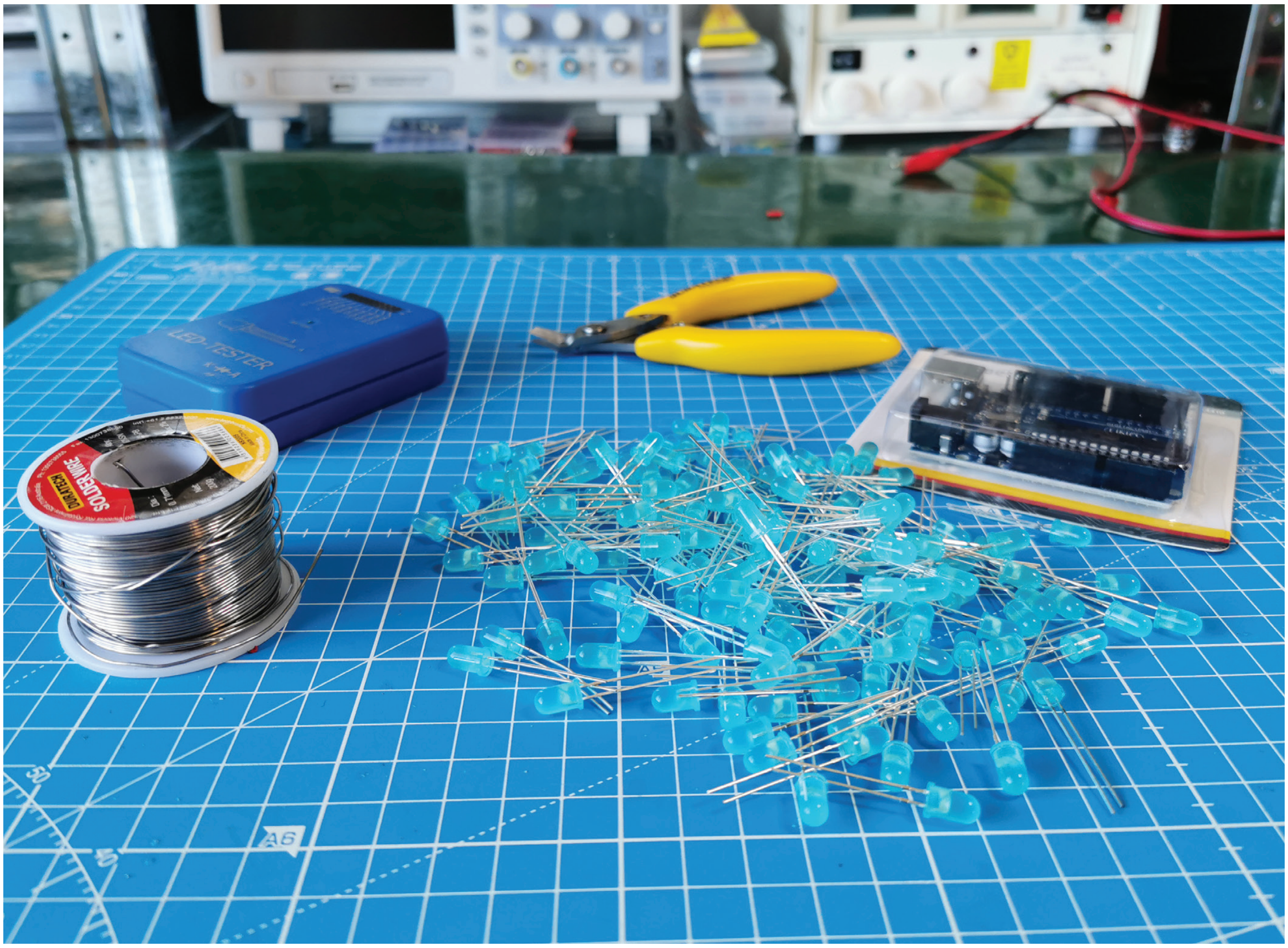
$$I = \frac{V}{R} = \frac{(V_{in} - V_f)}{\frac{R}{16}} = = \left(\frac{(5_v - 3.3_v)}{\frac{220}{16}}\right) = 483\mu A$$

This is negated, however, by the fact that the blue LEDs have a significantly higher intensity, rated at 350 millicandela, which is about 43 times higher than the red LEDs. The difference in apparent brightness is amazing, the blue LEDs appear many times brighter despite the fact they are supplied half the current. With that said, it is entirely up to you which you choose to go with. If you want a night light, for example, we recommend using the red LEDs as it's much more subtle. However, if you want a conversation starter on your brightly lit desk, then nothing is more eye-catching than the blue LEDs.

*Note: The relationship between the luminous intensity and forward current isn't linear, and so, it's not quite as simple as we suggest here. We just want to show the thought process behind the LED selection and what you may need to consider if modifying the project.*

In short, you will need to consider the luminous intensity of the LEDs you decide to use, along with the forward current and forward voltages. You will need to ensure that you don't exceed the maximum forward current / peak forward current of the LEDs by choosing an appropriate resistor, and overall, ensure that no pin will exceed the 40mA maximum current of the microcontroller.

We also recommend you use diffused LEDs, as the non-diffused variants put very little light out of the sides of the LED. Instead, they direct their light through the top, which will significantly reduce the visible light when viewed from the sides, and thus, not look anywhere near as good.



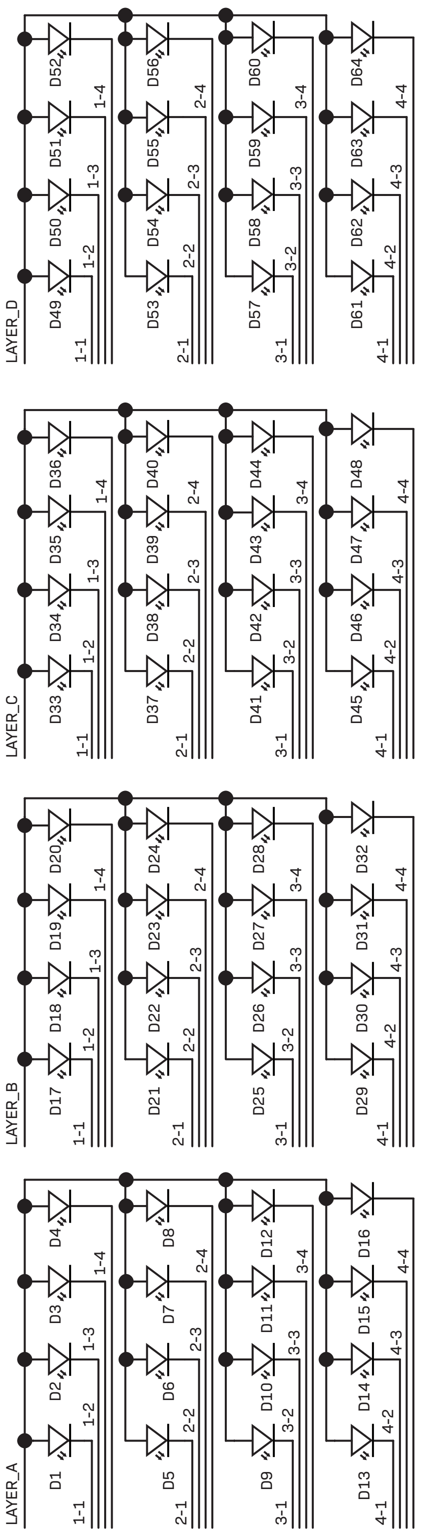
## The Build:

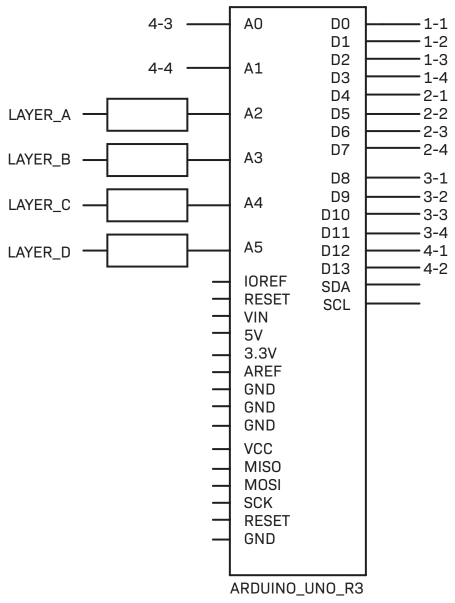
We're now going to cover the build from the PCB that is required to powering it all up. Before you heat up the soldering iron, we'll discuss how you'll first need to prepare and test the LEDs. and what you will need to help ensure the LEDs solder together perfectly.

You may notice a mix of red and blue construction photos in our build. The build is the same regardless of the LED colour you choose.

| PARTS REQUIRED: | PART NUMBER |
|---|---|
| 1 x Arduino Uno or Compatible Board | XC4410 |
| 2 x Pin Header Strips (32 pins needed in total) | HM3211 |
| 4 x 220Ω Resistors* | RR0556 |
| **LED COLOUR OPTIONS:** | |
| 64 x 5mm Red Diffused LEDs (8mcd or brighter) | ZD0150 or ZD1690 |
| 64 x 5mm Blue Diffused LEDs (350mcd) | ZD0185 |
| **OPTIONAL:** | |
| 1m x Twin & Earth 10A Solid 1mm2 Electrical Wiring (For LED Structure Supports) | WB1565 |
| 4 x #4 9mm Screws* (For Mounting the Arduino Uno into a 3D Printed Case) | HP0565 |
| 1 x LED Tester | AA0274 |
| 1 x Pack of Alligator Clip Leads | WC6010 |
| 1 x Roll of Electrical Tape | NM2800 |

* Quantity shown, may be sold in packs. You'll also need a breadboard and prototyping hardware.

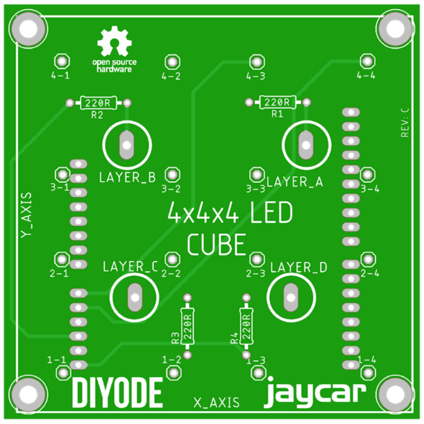


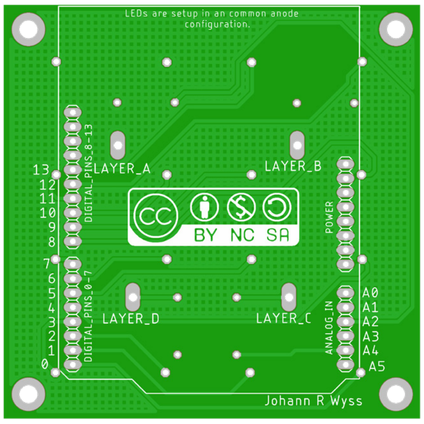


## THE PCB

To make the project as easy as possible to build, we have designed the circuit on a PCB. This will allow you to easily build the project without needing to go to the trouble of using stripboard with lots of tricky links or messy wiring. If your favourite electronics retailer hasn't kitted up for this project, you will need to get a PCB made. We have provided the files on our website for you to send off to your preferred PCB manufacturer. The ones we used were from Lintek.

The PCB files can be found on our website. The double-sided board measures 73mm x 73mm.





## LED JIG

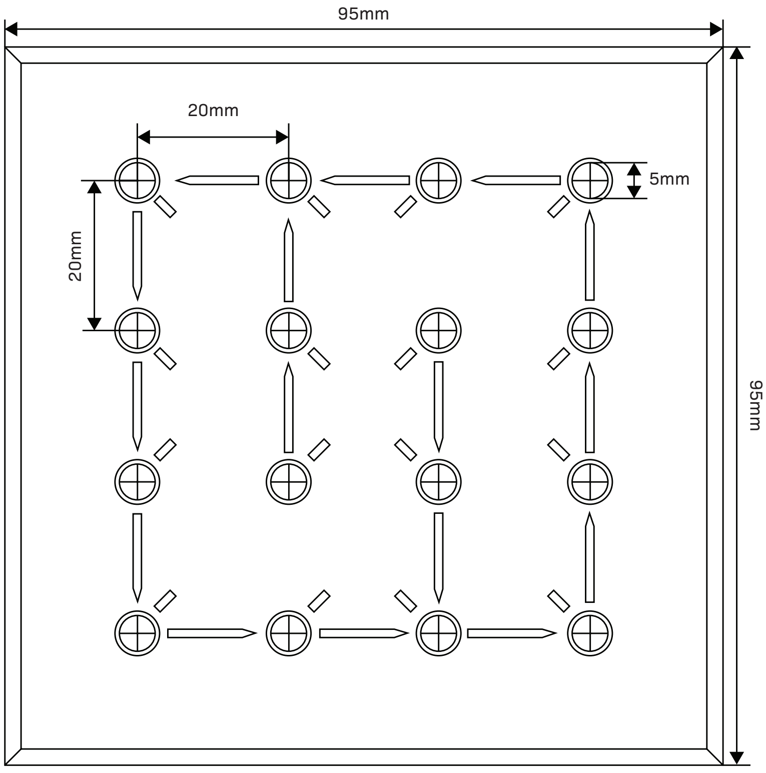
Considering that you are about to solder 64 LEDs, neatly spaced apart, and all pointing in the same direction, we recommend you make yourself a jig to hold the LEDs in place while you solder them. Using a jig will turn a good-looking cube into an awesome looking one.

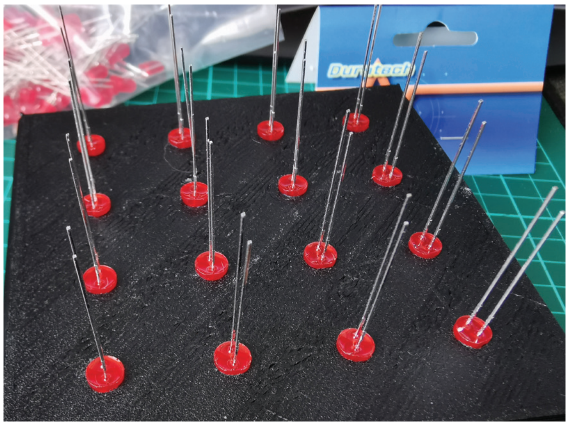
We have provided a 3D print file that we used, or alternatively, you can use the template here to make something similar from a piece of timber and a 5mm drill.

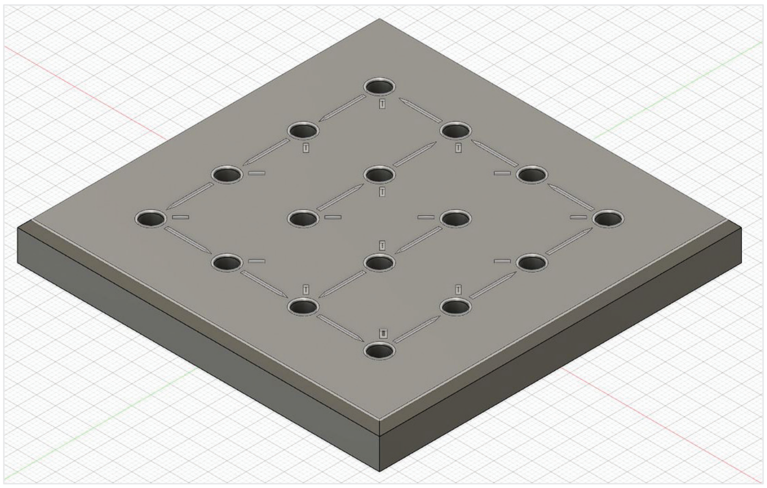
If you use our 3D printed file, you may find that you need to drill out the LED holes to 5mm depending on the tolerances of your printer. Don't make the holes too big though. You need the LEDs to fit firmly enough that they don't move or fall out when you are soldering.

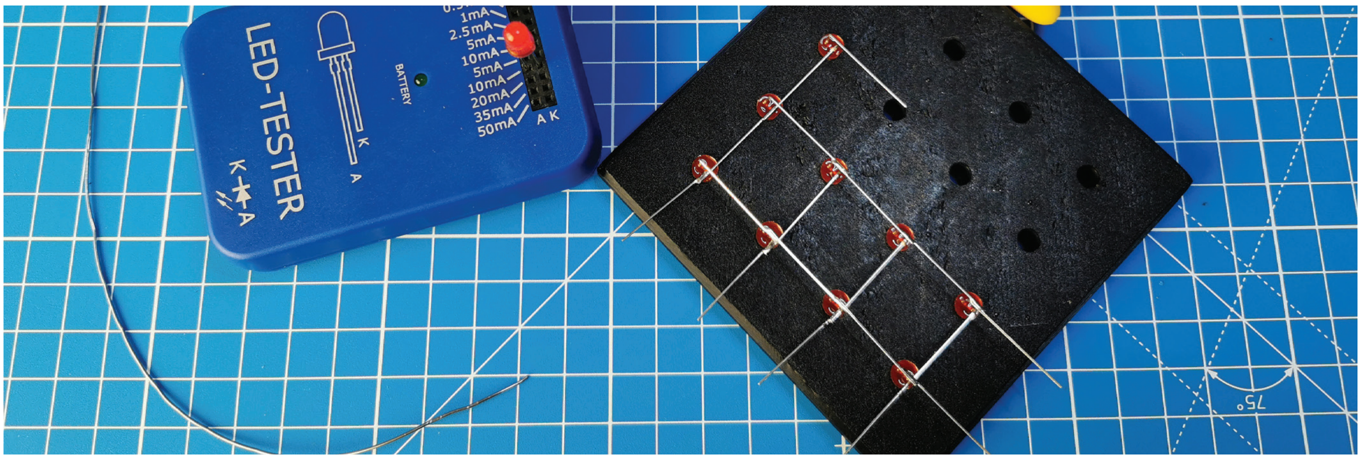
This template shows you the bends required for each LED to create four identical layers and holds the LEDs in the desired position.

*Note: If you are 3D printing the LED Jig, there is also a second part to print you should print, which is explained in further detail later on. This second part will help remove the LEDs from the Jig.*



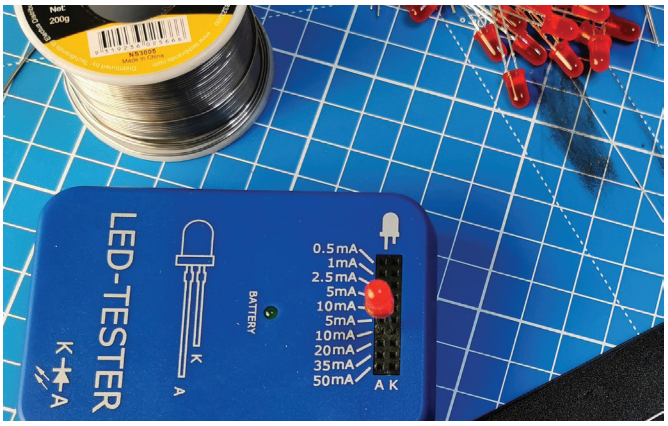






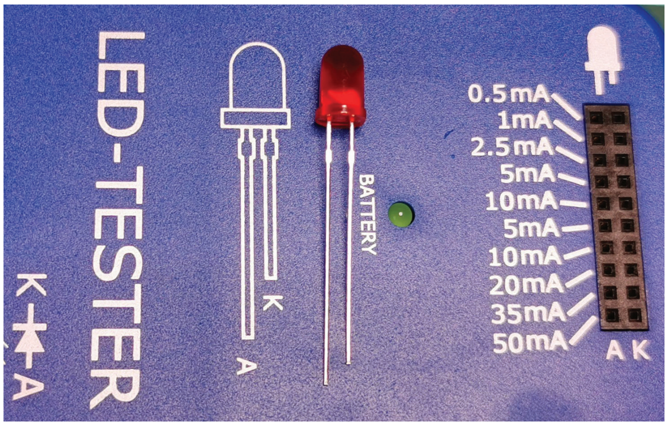
## PREPARING AND TESTING THE LEDS

There would be nothing worse than soldering your 64 LEDs together, only to find out that one of them doesn't work or has a different brightness to the others.

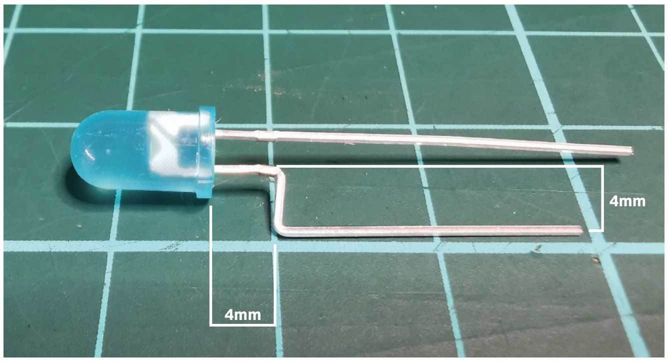


Before you start soldering the LEDs, we recommend you test them first. You can do this with a 3V lithium coin battery, but with 64 LEDs to test, an LED tester is ideal. Test the LED on the maximum average and minimum current range you expect the LED to operate at. For the standard LEDs in our build, we tested them at 10mA, 5mA, and 0.5mA. This ensured that the difference in brightness between each step was not too different. More importantly, it ensured the LED would illuminate at the minimum expected current of 0.5mA.

Out of the 230 LEDs that we tested for this project (we made three different LED cubes with 64 LEDs each, plus another two layers for testing), we had zero failures. However, two LEDs had a significantly notiaceable difference in brightness between the average 5mA and minimum 0.5mA current. We excluded these from our build, of course.



Once you have confirmed that the LED is working correctly, you can then carefully bend the LED's cathode leg, which is the shorter leg (also the side with the flat edge on the LED's rim). You will also notice a printed diagram of the LED and its anode (A) and cathode (K) on the LED tester.



You need to bend this cathode leg so that it has two 90° bends, each about 4mm apart. To make these bends, we used a pair of long nose pliers to hold the leg where we wanted the bend, and then gently bent the leg at that point.

Keep in mind that these bent cathode legs become the vertical connections. The longer anode legs will bend to form the horizontal connections.

*Note: You can either test and prepare all of the LEDs before you start, or one-by-one when you solder them into the matrix.*

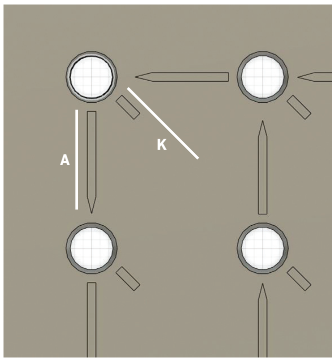
## SOLDERING THE LEDS TOGETHER

Now the fun begins! It's time to put your soldering skills to the test and assemble your project. Make sure you have all of the necessary components, tools, and LED jig at the ready.

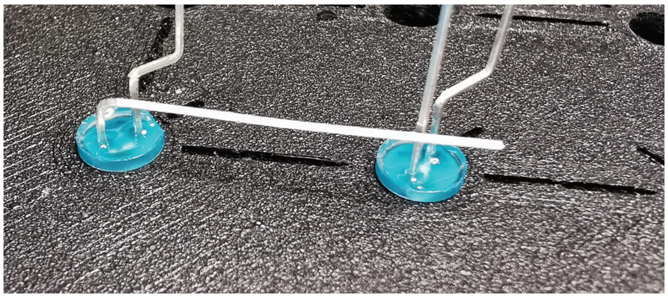
Before creating and soldering the array of LEDs to the PCB, we recommend you first solder the resistors and pin headers to the PCB. This will be much easier to do now before the array of LEDs are soldered to the board.

Once the resistors are in, you can start the process of assembling the four layers of LEDs.

Start by inserting an LED into the 3D printed or drilled out template to hold it firmly in place. It should be inserted into the template so that the newly bent lead aligns with the short indentation on the template.

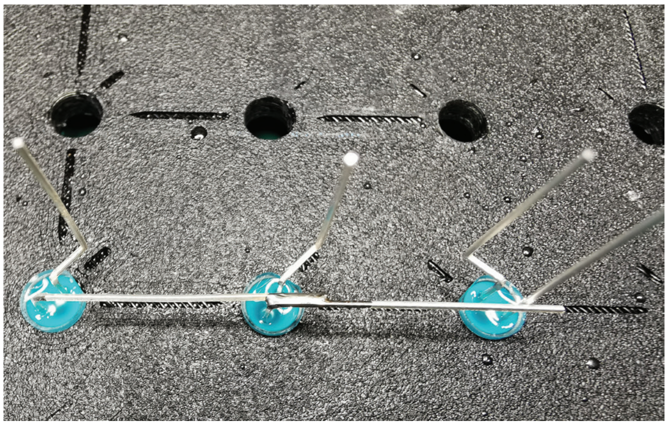


With the first LED in place, repeat the process with a second LED and insert it next in line. Bend the anode of the previous LED along the indicator line to meet the next LED, as shown here. This bend is made about 3mm from the base of the LED.



Once the first three LEDs are in place, start soldering the anodes together. This will help avoid the LEDs being accidentally misaligned if you bump them, etc.

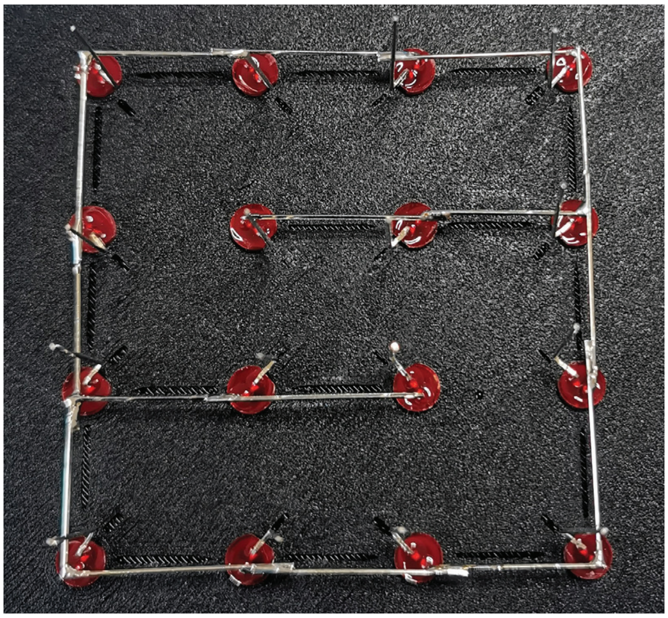
*Note: It is important that the leads are touching without any tension before you solder them together. i.e. you shouldn't need to push down on the leads to make them hold together when you're soldering. This avoids the layer of LEDs trying to spring apart and twist when you remove them from the jig. It may take some patience, and trial and error at first, but by the end, you'll have it mastered.*



Follow the exact same procedure by following the template, until your layer is complete. It's a process of:

1. Test the LED
2. Bend the cathode 90° in the two places shown
3. Insert the LED into the template with the cathode aligned with the small marker.
4. Bend the previous LEDs anode in the direction of the arrow on the template until it is aligned with the LED you just placed
5. Gently bend the anode lead of the previous LED until it is sitting comfortably, and solder it in place

Once all 16 LEDs for the layer are completed, your template should look like this (with your choice of LED colour).

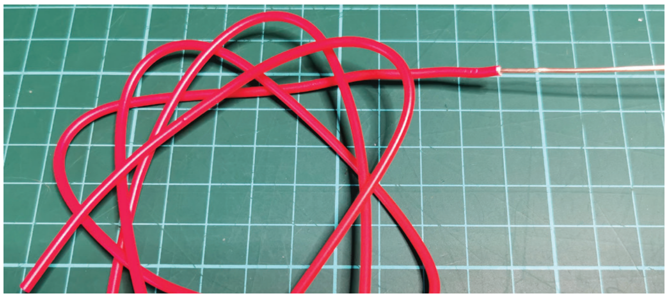


## LAYER REINFORCEMENT

Before you remove the LEDs from the jig, it needs to be reinforced in the centre to give the layer rigidity, and to help prevent any twisting when it is removed. The 400-micron thin legs of the LEDs are not enough to hold its structure so we'll add some solid core wire with a diameter of around 1mm.

We used the 1mm solid core wires from the active and neutral wires in a length of mains cable. We simply stripped off the outer insulation, then stripped off the insulation from the active and neutral wires to expose the single core copper strand.

**IMPORTANT:** *This copper will oxidise rapidly once you remove the insulation, which makes it difficult to solder to. Only prepare the wire just before you need to solder it.*

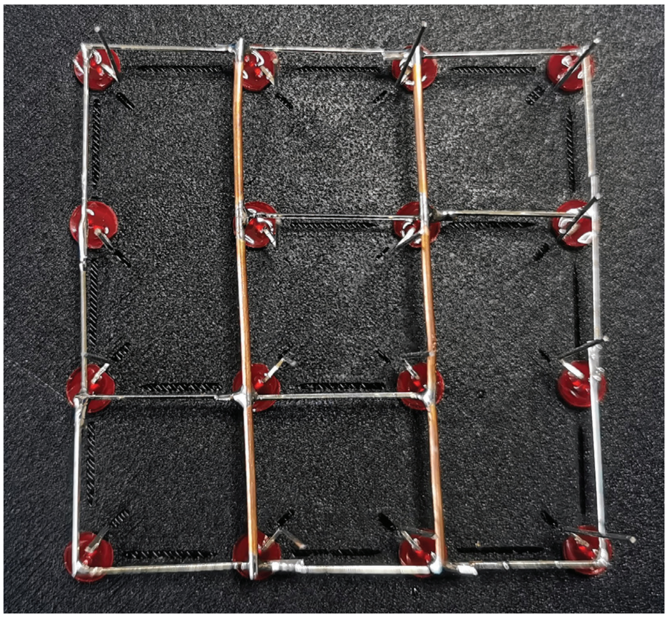


You will need two lengths approximately 70mm long to bind each layer, and an additional four lengths to secure each layer to the PCB.

Use two pairs of pliers to straighten the wire. You need to put the pliers on each end and firmly jerk the wire hard to make it straight. You could also straighten the wire by rolling it between two thick pieces of MDF or similar. The straighter you make them, the nicer the LED cube will look, so take your time.

Similar to soldering the LED legs together without tension, the same method needs to be applied to soldering this solid wire. Make sure the length of wire sits flat against the lead you're about to solder to before soldering. Any stress on the joints may make the structure twist when you remove it from the jig.

Solder the lengths of solid wire across both of the middle columns as shown here.

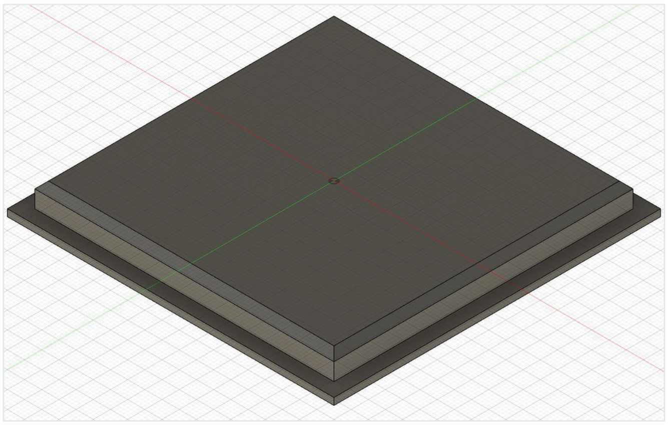


Pay attention to make sure that you are not shorting an LED's vertical anode to this wire. The wire may come close in some spots. Make sure you avoid any short circuits because these will cause a layer to not function correctly.

## REMOVING THE LED LAYER FROM THE JIG

With the layer complete, you need to carefully remove it from the jig. This needs to be done using even pressure on all of the LEDs protruding through the bottom of the template to ensure that the layer doesn't bend.

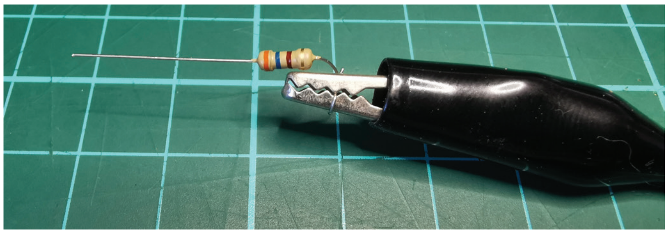
If the LEDs are in your LED jig firmly, then you will need to find a suitable method to remove the layer without bending them. In our case, we used a 3D printed part designed for this task (Print file is available on our website). This part fits into the bottom of the LED jig. You then firmly push on the top of the jig and this part applies even pressure to the LEDs o they push out evenly from the jig.



## TESTING THE LED LAYER

With the LED layer freed from the template, we recommend that you test every LED for peace of mind that they have been soldered in correctly, and without any heat damage caused by the soldering iron. If you assemble the cube and then discover you have a faulty LED, it will be extremely difficult to replace it, compared to replacing a faulty LED now.

In our case, we had access to a lab power supply with the current limited to 5mA. Using alligator probes, we connected the anode (red) to the common anodes of the LED array and touched the cathode (black) clip to each of the cathode vertical leads. If the LEDs all illuminated, we were good to go.



*Note: if your power supply does not have the ability to limit the current you can use a resistor on the cathode alligator clip like shown here. You simply touch the exposed end of the resistor to the cathodes of each LED. Note be very careful with this though. If* *the alligator clip touches the LED directly, the current will bypass the current limiting resistor, and this could damage the LED. Use the protective sheath of the alligator clip to cover as much exposed metal as possible to reduce the chances of this happening.*

To calculate the resistor you need to test each LED, you just need to calculate the resistor value using the formula:

$$R = \frac{(V_{in} - V_f)}{0.005}$$

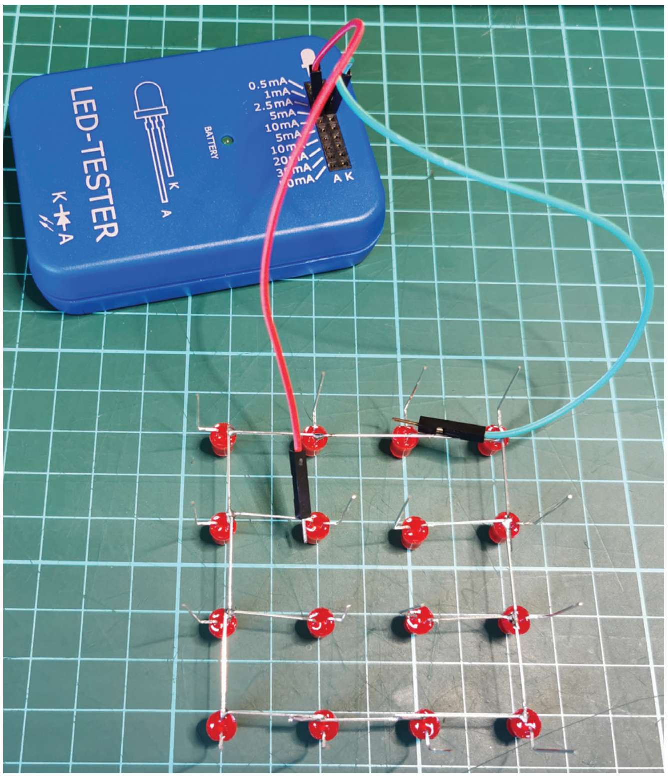
Where:

Vin is the voltage supplied by your power supply

Vf is the forward voltage of the LED you're using

0.005 is the current in Amps. i.e. 5mA.

Another method to test your LEDs is to use the LED tester shown earlier, which is designed to produce a current limited supply. Just use a set of male jumper wires to test the LEDs in the layer. Simply insert the male to male headers into the socket on the LED tester that corresponds to the 5mA output. Attach the anode (A marked on LED tester) to the common anodes of the LED array, and then touch the cathode (K on the LED tester) to each one of the LED vertical cathodes to verify each LED illuminates.



This layer was a conceptual layer and not the same design as our final design. We are simply using it here to demonstrate how you can use the LED tester to test the layer functions correctly.

*Note: Whilst the LED tester is likely designed to withstand a direct short circuit be careful to ensure that the output is not shorted to reduce the risk of damage to the tester.*

If all of the LEDs illuminated correctly then congratulations, your first layer is done!

If any of the LEDs failed to illuminate, check the solder connections and look for any short circuits across an LED. If you can't find any issues with the solder or connections it's possible the LED was damaged during soldering. In this case, you need to very carefully remove the faulty LED. The best way to do this is to hold the plastic case of the LED so the array is slightly above the desk / work surface. Using your soldering iron, heat the solder on the LED's anode. The weight of the array should mean that the LED is easily freed once the solder is molten.

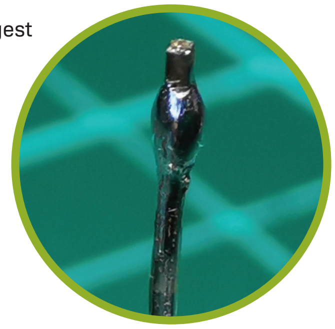
With the faulty LED removed, very carefully put the layer back into the LED jig to avoid bending, and re-insert a new LED. Once completed, test every LED again.

## RINSE AND REPEAT

With the first LED layer soldered together and tested, you can repeat the process for the remaining three layers, so you have four layers in total. Take your time. Any rushing may cause LEDs to fail or a cube that doesn't look straight once it's all assembled.

## LED CUBE ASSEMBLY

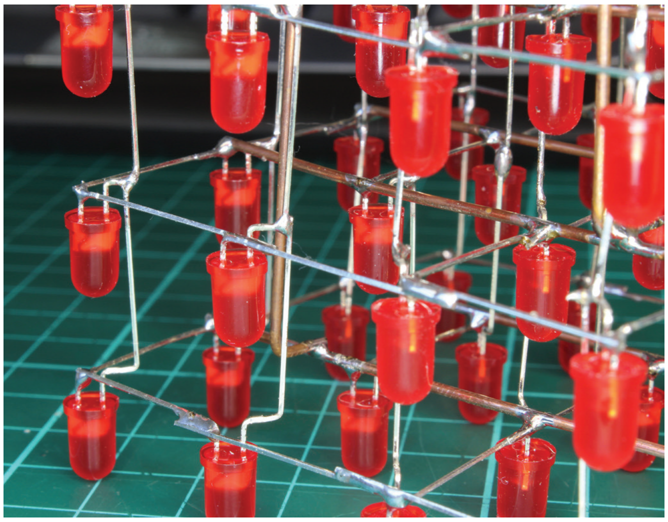
With the four layers complete, it is time to assemble the cube. The easiest way to do this is to put the first layer face down on your workbench so that the LED tops are touching the work surface.



To make assembly easier, we suggest you pre-tin (add some solder) to the LEDs cathode leads, which will make the task of joining the layer significantly easier. The tinning process will help the joint take solder much more easily, especially when access is impeded by the layer placed on top.

You should also tin the point where the cathode will join the next layer, which is the 90° bend where the lead goes from horizontal to vertical.

**IMPORTANT:** *Do not tin the 90° bends on the layer that will form the top of your LED matrix as there will be no join there.*

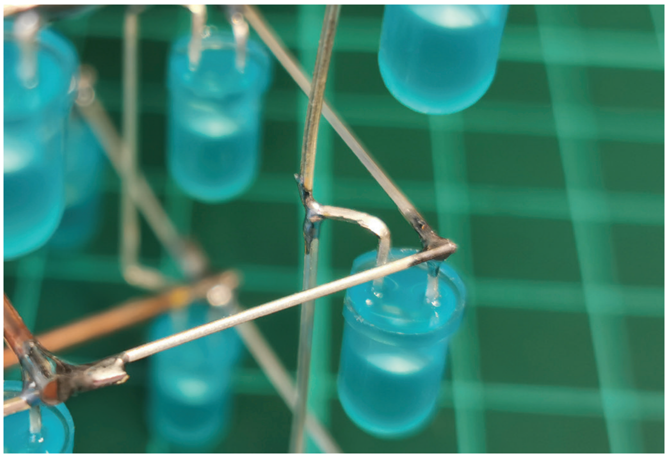


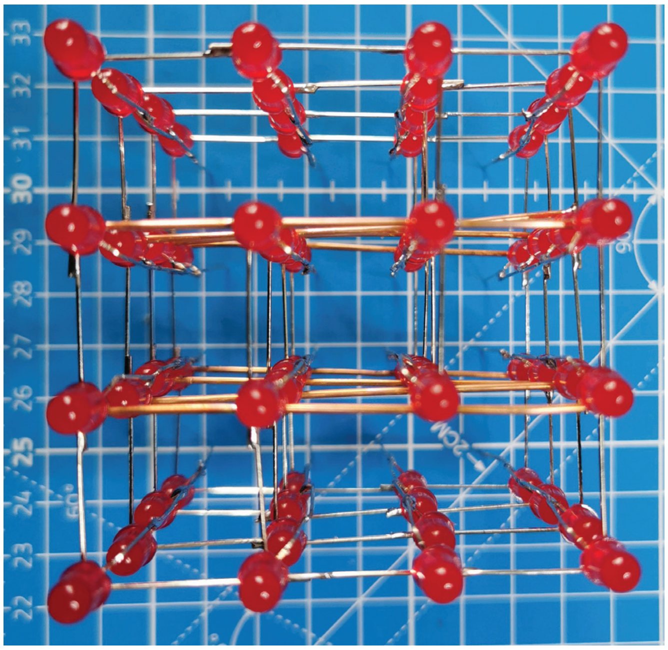
To join two layers, you first need to align the layer making sure the strengthening copper rods are all running in the same direction.

With the topmost layer face down on the work surface, gently hold the next layer on top of it so that the cathodes of the lower layer are just overlapping the cathodes of the next layer up.

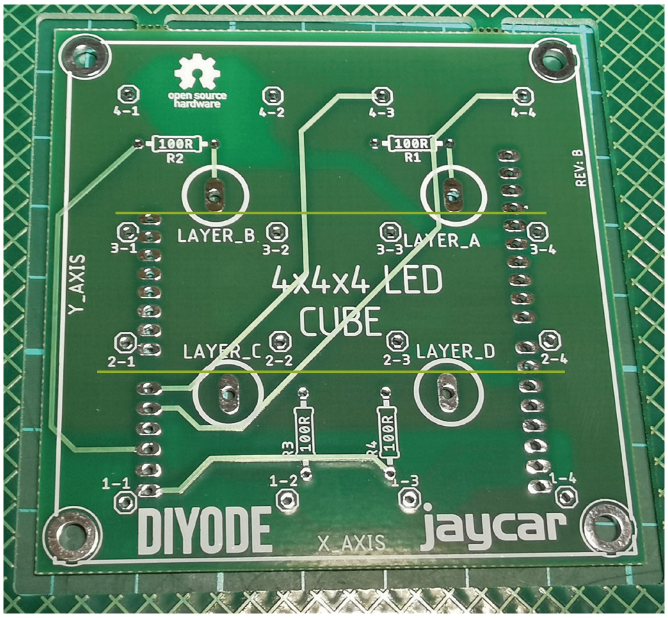
We found the best practice was to first solder the four corners by just touching the soldering iron tip (with a small amount of solder on it) to the pre-tinned joint. This is just a temporary join, so it does not need to be perfect. Its job is to simply hold the layer in place, allowing you to solder the remaining 16 joints per layer. Once all the joints are made you can then go back over the initial 4 corner joints.

**IMPORTANT:** *As with all the other soldering processes in this project, it's imperative that you solder these connections in their resting state and not under any tension. If you find yourself holding a lead in place to solder it then you need to stop and go back over each joint to remove any stored tension. Just applying the hot soldering iron tip should be enough to reflow the solder, if the lead jumps away, you need to re-bend it until it is resting in the desired position, allowing you to re-solder it. Failure to do this will result in an unattractive or oddly shaped cube.*





When you're done soldering the layers you should have a cube that resembles something like this. Note the middle two rows (horizontal) are closer together than the outer layers. These rows need to line up with the corresponding two layers on the PCB, as shown here.



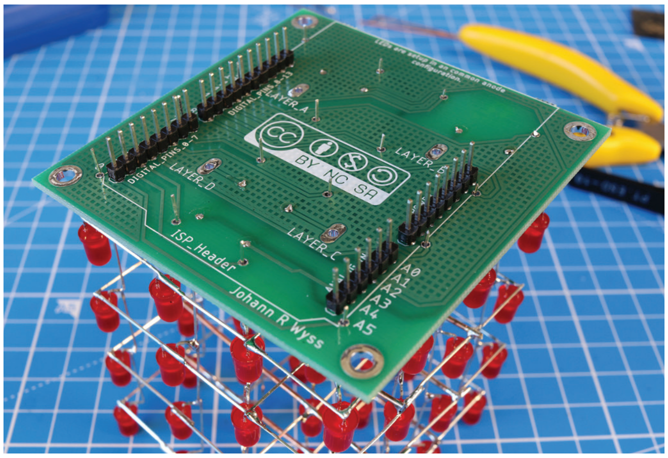
## CONNECTING LED LAYERS TO THE PCB

Aligning the PCB on the LED array can be quite a challenge, as the leads will not immediately align with the holes in the PCB. This is due to subtle differences in the bend of the leads as well as an intentional offset in the PCB design. We intentionally have the leads spaced a little further apart to cause them to fan out slightly, which will help give added strength to the design.

We found the best way to get the leads into the PCB was to work one row at a time, keeping the PCB at a slight angle, and using tweezers to push the pins to the corresponding PCB hole. Having the PCB at an angle means the completed rows won't come out while manipulating the next row.

Once the PCB has slid onto all 16 of the LED cathode pins, you need to position the PCB at an even distance from the first LED layer. Don't use the length of leads poking through the PCB as a guide because it may end up creating a slightly angled cube (we made this mistake in our second cube build). Rather, measure to the base of the LED.

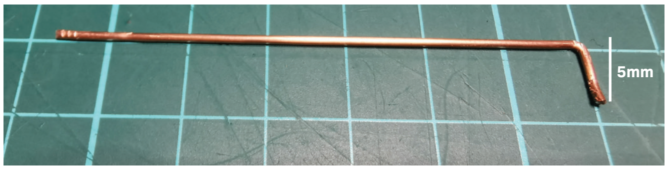
We suggest you solder just the four corners, measure them and make changes if needed to one corner at a time, until the cube is level. This way, the other three soldered corners hold the cube and PCB in place while you re-adjust the fourth corner. Once you are satisfied that the cube is level, solder the remaining 12 LED leads in place.



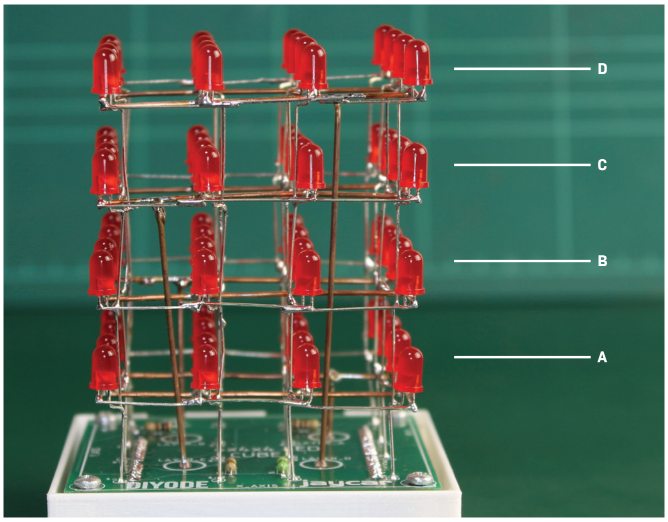
## SOLDERING THE BUSBARS

The final construction step is to create the layer busbar/cables, which will provide the voltage and current to the individual layers. This busbar is made using the same 1mm diameter solid core wire we used to reinforce the layers. This bar will also help to mechanically secure the LED array to the PCB and provide much of the cube's strength.

We stripped and straightened enough wire to feed in from the bottom of the PCB and to reach the corresponding layer. The wire should have a 5mm length with a 90° bend on the end that connects to the layer, as this will help avoid contact with the layers below it. Note, this isn't necessary for the bottom layer.



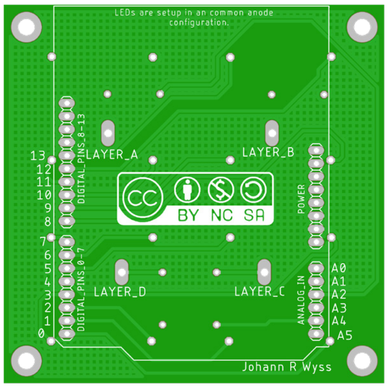
When connecting the busbars, remember that layer A is the bottom layer and layer D is the top, as shown here.

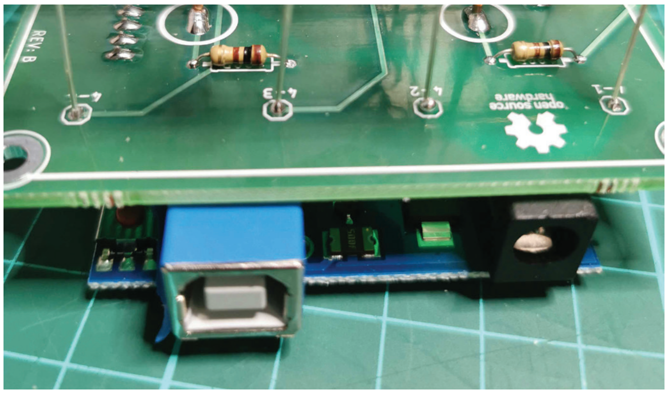


With the LED cube constructed, you can insert it into your Arduino Uno or compatible board (Hopefully you have the header pins already soldered onto the PCB).

You will notice there is a shape of an Arduino Uno silkscreened on the PCB to show you the way they need to connect, even though the cube and Uno will only connect one way.

**IMPORTANT:** *If you look closely, you may notice the USB socket on the Uno board may foul against the solder pads for LED pin (4-3). If the header pins in your board are long, the PCB and USB socket shouldn't short, but if they do, it is very important that you prevent these from shorting. Simply cover the top of the USB socket with some electrical tape.*

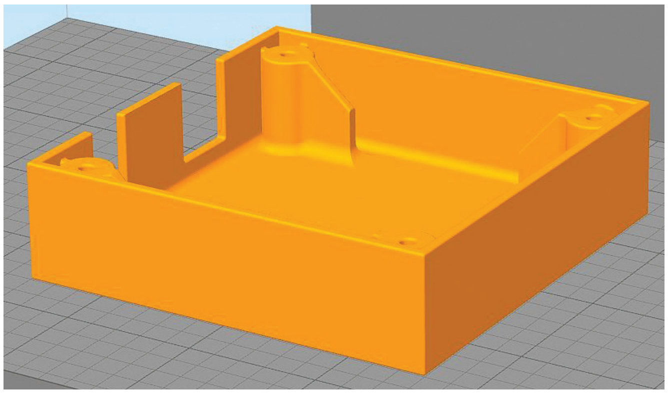




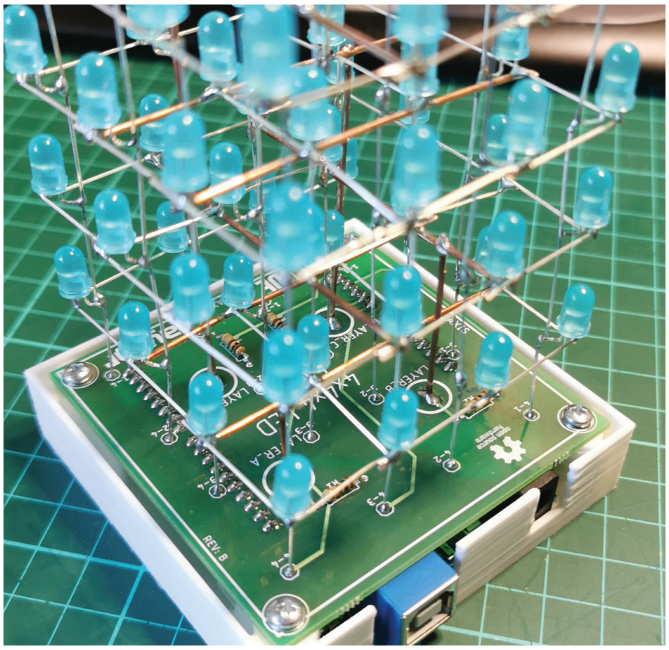
## 3D PRINTED ENCLOSURE

To protect the LED cube and Arduino Uno, we have created a very simple enclosure, which measures just 80 x 80 x 24mm. This will reduce the chance of short circuits if you were to accidentally place the exposed solder pads of the Arduino Uno on a conductive surface. We designed the enclosure in Fusion 360 and will provide the working files on the website if you wish to modify it to better suit your specific needs.

The one we show here was printed on our Flashforge Creator Pro at 200-micron layer height using Flashforge branded white PLA. It is designed to be printed without needing supports and uses around 35 grams of plastic. At these settings and orientation, it took about 2.5 hours to print.

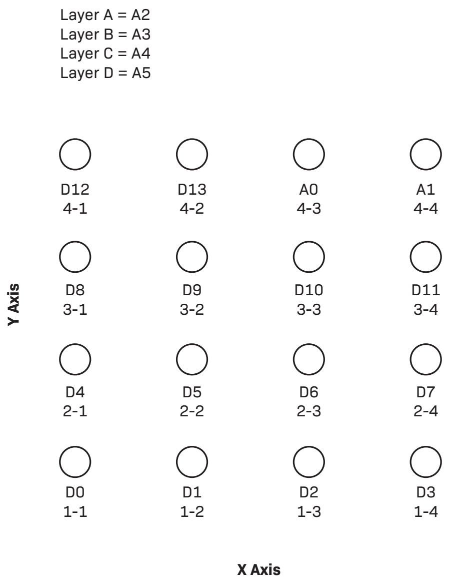


The PCB sits on top of the enclosure with the USB and DC jack protruding through the gap. This allows you to power the cube by either means. The PCB is secured to the enclosure using 4 x #4 screws at least 6mm long. Take care of screwing the screws to the PCB as the LEDs are very close and could easily be damaged.



## PROGRAMMING THE CUBE

We briefly explained the coding process earlier in the article, however, we will go into a little more detail here, showing how you can code your own animations. The illustration here will be a great handy tool that will make programming the device as simple as possible.



This shows the position of each LED, along with their corresponding pins. You can use the values silkscreened on the PCB to help orient the cube. i.e. (1-1) is the bottom left corner, while (4-4) is the top right, etc.

Controlling the cube is easy. You just create an array of integers that relate to the LEDs on each layer that you want to control. For example, if you want LEDs:

(1-1), (1-2), (1-3) and (1-4) illuminated on layer A

(2-1), (2-2), (2-3) and (2-4) illuminated on layer B

(3-1), (3-2), (3-3) and (3-4) illuminated on layer C

(4-1), (4-2), (4-3) and (4-4) illuminated on layer D

You need to create four arrays representing the specific pins, like this:

0, 1, 2, 3

4, 5, 6, 7

8, 9, 10, 11

12, 13, A0, A1

To put this in code, it would be:

```
int pins1[4] = {0, 1, 2, 3};
int pins2[4] = {4, 5, 6, 7};
int pins3[4] = {8, 9, 10, 11};
int pins4[4] = {12, 13, A0, A1};
```

This instantiates four arrays, each with four elements.

If we were to pull all of these pins Low, and then pull each layer High, we would see all 64 LEDs illuminate at once, which we don't want. To overcome this, we need to create a code that will rapidly pull the pins Low for a specific layer, pulls the layer High, and then Low again before pulling the pins Low for the next layer.

It needs to do this so quickly that the human eye believes that the LEDs are illuminated at the same time when in reality, only four, in this case, will ever be illuminated in any one moment.

To do this for the first layer, the code will look like this:

```
  allOff();
  int count = 0;
  while(count < 1000){
    for(int i = 0; i < 4; i++){
      digitalWrite(pins1[i], LOW);
    }
    digitalWrite(A2, HIGH);
count ++;
  }
```

Note the line allOff(); calls a function that turns every layer Low and every pin High, making sure no LEDs can be illuminated.

'Int count = 0' instantiates a variable called count, which we will use as a counter to keep track of the number of loops the program has done.

'while(count < 1000)' creates a loop that will repeatedly run each line between the parenthesis { } until the condition is met. In our case, it will keep running until the variable counter exceeds 1000.

'for(int i = 0; i < 4; i++)' creates a loop that runs exactly four times, doing each line between the parenthesis { }, while simultaneously incrementing a counter on variable i.

'digitalWrite(pins1[i], LOW)' pulls the pin, which is stored in the array pins1 in position [i], LOW.

*Note: Arrays in C++ start at 0, therefore the array value in [i] = 0 is the first value in the array. In our case, 0. This can be shown here:*

| ARRAY | POS 0 | POS 1 | POS 2 | POS 3 |
|---|---|---|---|---|
| pins1[4] | 0 | 1 | 2 | 3 |
| pins2[4] | 4 | 5 | 6 | 7 |
| pins3[4] | 8 | 9 | 10 | 11 |
| pins4[4] | 12 | 13 | A0 | A1 |

'digitalWrite(A2, HIGH)' pulls the layer A pin High, allowing current to flow into the LEDs pulled Low.

'count ++' increments the count variable by 1. When this value exceeds 1000, the while loop will no longer be true and the program can leave the while loop.

To make it illuminate the next layer, we can add the code:

```
allOff();
for(int i = 0; i < 4; i++){
  digitalWrite(pins2[i], LOW);
}
digitalWrite(A3, HIGH);
allOff();
```

You may notice it's the same as previous. We are just using the next array, and we added the functions to turn off the previous layer before and after this layer.

In total, we do these four times, once for each layer, and the final result is as shown in the following code.

As you can see, the code will repeat itself 1000 times before escaping the while loop and moving onto the next line. In our code, we have created a number of such blocks of code and assigned them to functions. Much like the allOff() function. This way, we can call specific blocks of code when needed, such as the allOff(); and allOn(); functions.

```
allOff();
   int count = 0;
   while (count < 1000) {
      for (int i = 0; i < 4; i++) {
         digitalWrite(pins1[i], LOW);
      }
      digitalWrite(A2, HIGH);
      allOff();
      for (int i = 0; i < 4; i++) {
         digitalWrite(pins2[i], LOW);
      }
      digitalWrite(A3, HIGH);
      allOff();
      for (int i = 0; i < 4; i++) {
         digitalWrite(pins3[i], LOW);
      }
      digitalWrite(A4, HIGH);
      allOff();
      for (int i = 0; i < 4; i++) {
         digitalWrite(pins4[i], LOW);
      }
      digitalWrite(A5, HIGH);
      allOff();
    count ++;
   }
```

What if you want the same LEDs illuminated on every layer? Well, that's even more simple. To do that, all you need to do is tell the LEDs to be pulled Low using the for statement from earlier, and then tell all layers to go High using a function we created.

In code, this would look like this:

```
      for (int i = 0; i < x; i++) {
         digitalWrite(arrayName[i], LOW);
      }
layersOn();
```

Where 'arrayName' is the name of the array you have stored the pins you want to be pulled Low.

'X' is the number of elements in the array

'layersOn' is a function we wrote to easily tell every layer pin to go high.

You will need to instantiate the array using the code:

```
Int arrayName[x] { "values you want in here"};
```

If you're new to programming with Arduino, this may initially be a bit to get your head around, but with practice, it becomes quite simple to follow.

## WHERE TO FROM HERE?

Well done on mastering your soldering and building your LED cube. We hope it is sitting proudly on a desk for you to admire and show off to your friends and family.

You could find a suitable transparent acrylic box to go over the cube to protect it, but keep in mind, this may cause some internal reflections, which may or may not enhance the display.

Challenge for beginners: For practicing you coding skills, make a test program to light each LED one at a time in turn.

Now that you have mastered this 4 x 4 x 4 LED cube, you can move onto an RGB version or an 8 x 8 x 8. Phew! A bit of soldering to look forward to with those! Keep an eye on our website for new LED cube projects. ›› ■

## WANT MORE?

**For the code, PCB, and 3D print files, or to discuss this project, visit:**
**https://diyode.io/00?xxxx**