

These articles from HackSpace magazine and The MagPi Magazine along with this compilation are licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0).



Figure 1: CC-BY-NC-SA 3.0

# Getting started with KiCad, schematics

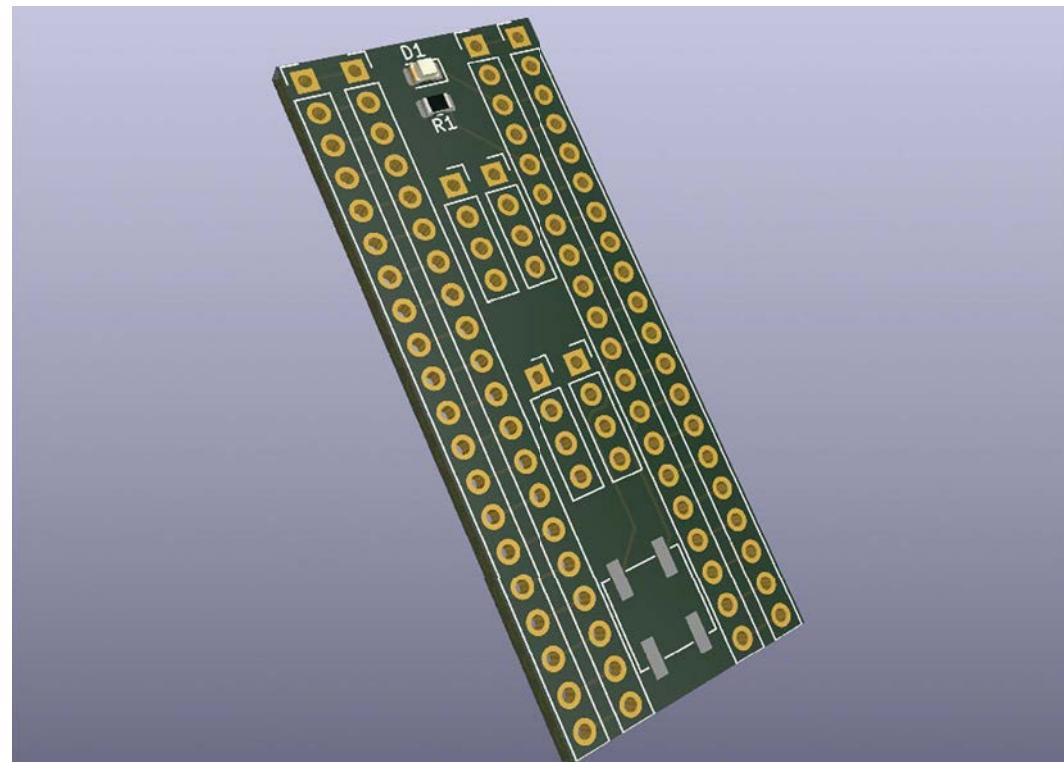
In this first of a series of articles around PCB creation with KiCad, let's dive into laying out a simple schematic



**Jo Hinchliffe**

@concreted0g

Jo Hinchliffe is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high-power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and CNC kit!



**K**iCad is an amazing piece of free and open-source software that allows anyone, with some time and effort, to make high-quality PCB designs.

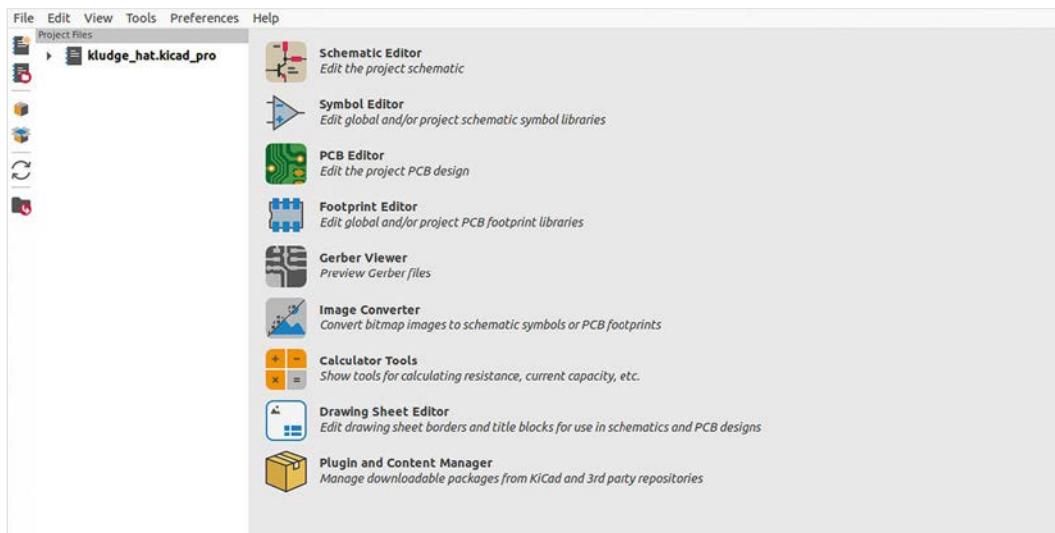
Couple this amazing software with numerous PCB fabrication companies and even PCBA services – companies that will make and assemble your PCB designs – and there's never been a better time to get into this aspect of making.

PCB design can be a steep learning curve. With that in mind, we wanted to start this series of tutorials with a hack – this is HackSpace magazine after all! In the first two parts, we are going to create

a PCB design to the point of getting it manufactured. However, to make this part accessible, we are going to cut some corners. Don't worry, we'll explain what the corner cuts are, and we'll do things more correctly in subsequent articles, once we have the basics down.

So, first thing, download and install KiCad – the current stable version is version 7, which was released in early February 2023. KiCad is available across a wide range of operating systems – Windows, macOS, and lots of Linux distributions. We're running it on Ubuntu, but it should be comparable across all the platforms.

**Above** The final simple add-on board for a Raspberry Pi Pico



**Figure 1** The opening page of KiCad with the different applications that make up the KiCad suite listed

With it installed, click on the main blue 'Ki' KiCad icon to open the main application. You should see a screen that looks like **Figure 1**. You'll see that, really, KiCad isn't just an application, it's more a suite of applications that work together. Whilst you can jump into any application from here, the most common workflow for KiCad is to first work in the Schematic Editor and then, after creating a schematic, move to the PCB Editor to lay out the PCB physically. Our first action should be to set up a new project. Click File > New to create a new project. It's worth putting projects into their own folder as each KiCad project generates around five project files and a folder in which it automatically generates some project backup files.

Once our new project is created, let's make a start. We're going to make an add-on board for the Raspberry Pi Pico. It's going to be a prototyping or 'kludge' board with not too many features on it for simplicity. It's going to have all the Pico pins broken out to through-hole pads, it will have a power-indicating LED, and it will have a reset button. After this, any spare area on the board will have simple through-hole pads on it to allow us to connect experiments to the board. As we said earlier, it's definitely hacked together, but will supply us with ample opportunity to learn some KiCad basics. To begin, open the Schematic Editor application by clicking the top icon.

You should now see a blank page ready for our schematic to be drawn (**Figure 2**). In the lower right-hand corner, you will see a small collection of text boxes which include various fields for the name of

the sheet, the revision number of the schematic, and more. If you left-click somewhere on this section and then press the E key, you should launch a window called 'Page Settings'. You can edit this to add any text details or comments you want to add to this section, but you can also change the page dimensions – it defaults to A4 – and the orientation and more. Experiment inserting text into the Page Settings window to see where the text appears on the schematic.

With your page set up, we can now begin to add schematic symbols to the schematic. The most correct way to make a Pico add-on board would be to place a Pico in the schematic and connect everything to it, but we are going to use a workaround to do this in a much simpler way. The main idea of our target add-on board is for us to be able to solder in rows of header sockets so that we can connect it onto the pins of a Pico. In turn, we also still want to be able to solder to those pins, so we need each side row of

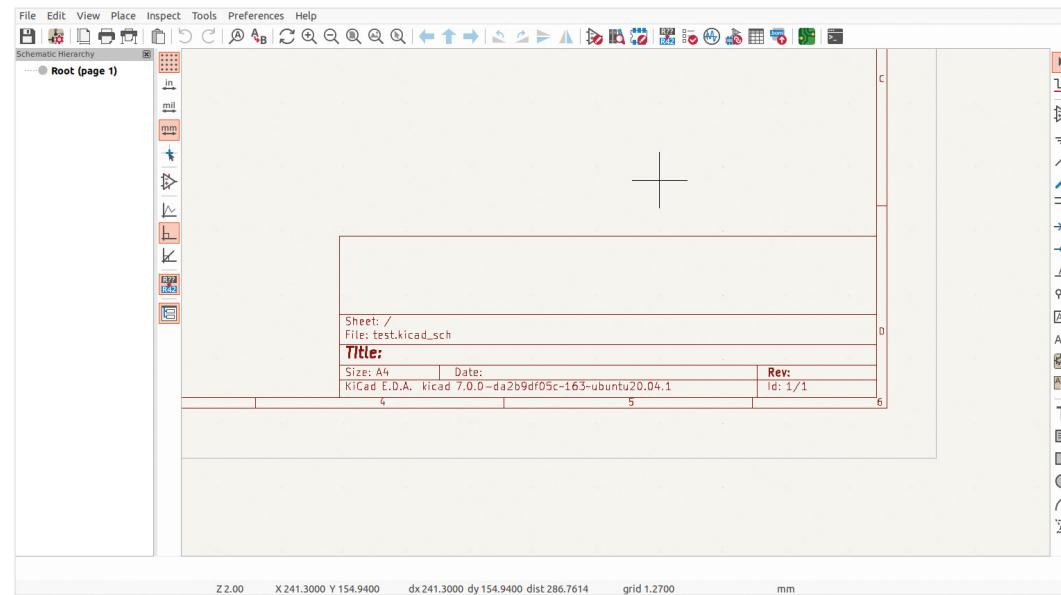
pins to be broken out to another collection of pads. To do this we'll add two 20-pin connectors, one on each side of our schematic. To add the first one, we are going to click the 'Add a Symbol' icon – the third icon down on the right-hand column. The first time you click this icon, it may take a few seconds for the libraries of schematic symbols to load. Once loaded, you should see a window called 'Choose Symbol'. On the left-hand side of this window, you should see a list of items, each of which is a library of a group of schematic symbols. These are grouped in related items; so, for example, you might find, →

**It's definitely hacked together, but will supply us with ample opportunity to learn some KiCad basics**

**QUICK TIP**  
If you hover over any tool icon in KiCad, you get a description of the tool. We'll use these tooltips to describe tool icons throughout these articles.

# Getting started With KiCad, schematics

## SCHOOL OF MAKING



**Figure 2** A blank Schematic sheet with text boxes for various schematic labelling and version numbering

### KEYBOARD WARRIOR

Whilst you can do everything in KiCad with a mouse and pointer, it's really worth learning a few quick keyboard shortcuts that can help make life easier. Brilliantly, many keyboard keystrokes offer the same functions in both the Schematic Editor, which we are using in this tutorial, but also in the PCB Editor, where you will spend a lot of your time in the next part of the series. The first useful ones are the **F1** and **F2** keys for zooming. The **F1** key, when pressed, will zoom in, and **F2** will zoom out. Note that both of these zoom actions are centred around where your pointer is. With a little practice, it becomes second nature to move the pointer and zoom in and out to get the view you need. Next up are the **M** and **R** keys. If you select an object in the Schematic Editor or the PCB Editor, you can press **M** and then that object will move until you left-click to place the object again. Note that in the PCB Editor, you can select smaller parts than the complete footprint of a component. You can therefore select and move silkscreen labels and more. If you want to select the whole component, selecting a component pad will usually then select the entire footprint. With the **R** key, you can rotate a selected item in either the Schematic Editor or the PCB Editor, again single left-clicking to place the part when rotated correctly. Note that you can rotate the item repeatedly to get to the orientation you require. Finally, another useful shortcut is the **E** key. With an item selected, pressing the **E** key allows you to edit that item's properties. This can be any number of editable parameters, from labels to pad sizes to hole dimensions and more.

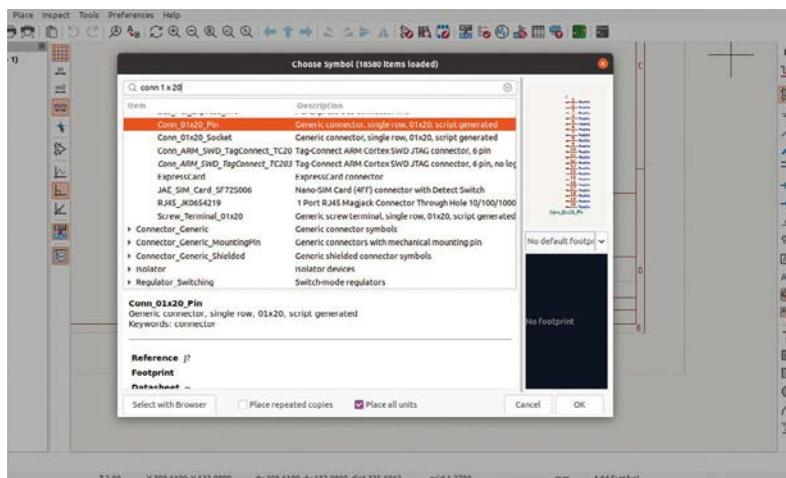
#### QUICK TIP

Whenever a tooltip has a letter in brackets, that letter is a hotkey to switch to that tool.

at the top of the list, '4xxx' which, if you click on the drop-down menu button next to the name will reveal a library full of the CMOS 4000 series of logic chips. You can of course manually scroll through the libraries and look through the items they contain, but you can also search the libraries to find what you need. We are going to add a connector symbol that represents the 20-pin header we eventually want to be able to fit. To do this, type 'conn' into the search bar and, as you type, you should notice the list items are now all the items that start 'conn'. Scroll down the list and select the item that reads 'Conn\_01x20\_Socket', and then click the OK button in the lower right-hand corner of the window (**Figure 3**). The Choose Symbol window will close and you will be back in the Schematic Editor, and you should have a symbol for our 20-pin socket attached to your pointer. Move the pointer to where you want to place the connector and left-click to place it.

#### A SACKLOAD OF SOCKETS

We are going to place three more of these into the schematic. We can again click the Add a Symbol tool icon but notice that, as the library list area populates, we now have a 'Recently Used' area at the top with our 20-pin connector listed underneath. Click this listing and then OK, and we can place the next connector in the schematic. Repeat this until you have four of the connectors placed. Next, use the **M** and **R** hotkeys to move and rotate the connectors so that we have two pairs of connectors in line with each other, with the small circle ends facing inwards



**Figure 3** The Choose Symbol dialog that allows you to search for schematic symbols

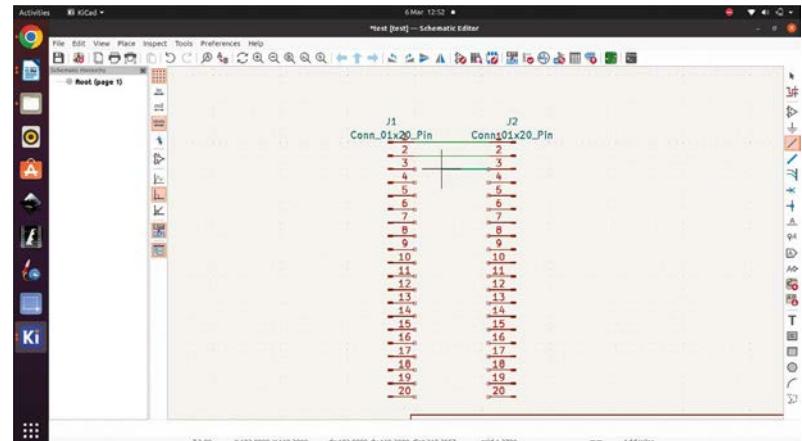
towards each other. You can also right-click on a selected symbol and use the Mirror Horizontally tool. The small circles are the part of the symbol to which we will create connections.

Using the Add a Wire (W) tool, you can left-click on one of the connector pin circles and then draw a connector wire line across to the opposite connector pin (**Figure 4**). If you make a mistake, you can use **CTRL+Z** to undo the last action – or, to cancel the wire mid-draw, you can right-click and select Cancel from the drop-down menu.

Continue to wire between each of the opposite pins on each pair of our connectors until they are all connected together. Notice that each of the connector symbols have a couple of text references connected to them. It can be good practice to edit these so that they are useful and help you keep track of what things are. Select the whole of one of our connector symbols (use the selection tool to draw a box over the entire symbol) and then press **E**. You should now see a Symbol Properties dialog box (**Figure 5**). You can now edit the ‘reference’ (if required) and the ‘value’ text entries. We edited ours and gave each connector a descriptive ‘value’ such as ‘Pico\_Pins\_1-20\_Left’ etc. This is useful when we lay out our PCB in the PCB Editor, as it ensures we can identify the multiple connectors correctly.

To finish our schematic, we are going to add more components and make more

connections. To begin, let’s add a resistor and an LED and connect them to pins 38 (GND) and 36 (3V3 (OUT)) on the Pico. These pins are the third pin and fifth pin down on our right-hand connector. Use the same techniques we used earlier to choose a symbol, searching ‘R’ for a resistor and ‘LED’ for an LED



symbol. Add wires to connect the circuit together, as shown in **Figure 6**. Next, let’s add a switch symbol which we will wire in as a reset button for the Pico.

Adding a reset button is a good example of a way that KiCad works that some other electronic design automations (EDA) don’t. In some EDAs, the symbol you choose at the schematic level defines exactly the hardware package of the electronic component

that will be on the PCB. For example, in other packages you wouldn’t place a generic resistor symbol, you would place a resistor symbol linked to a specific resistor, say a 6mm long 1/4 watt carbon resistor placed horizontally. This would

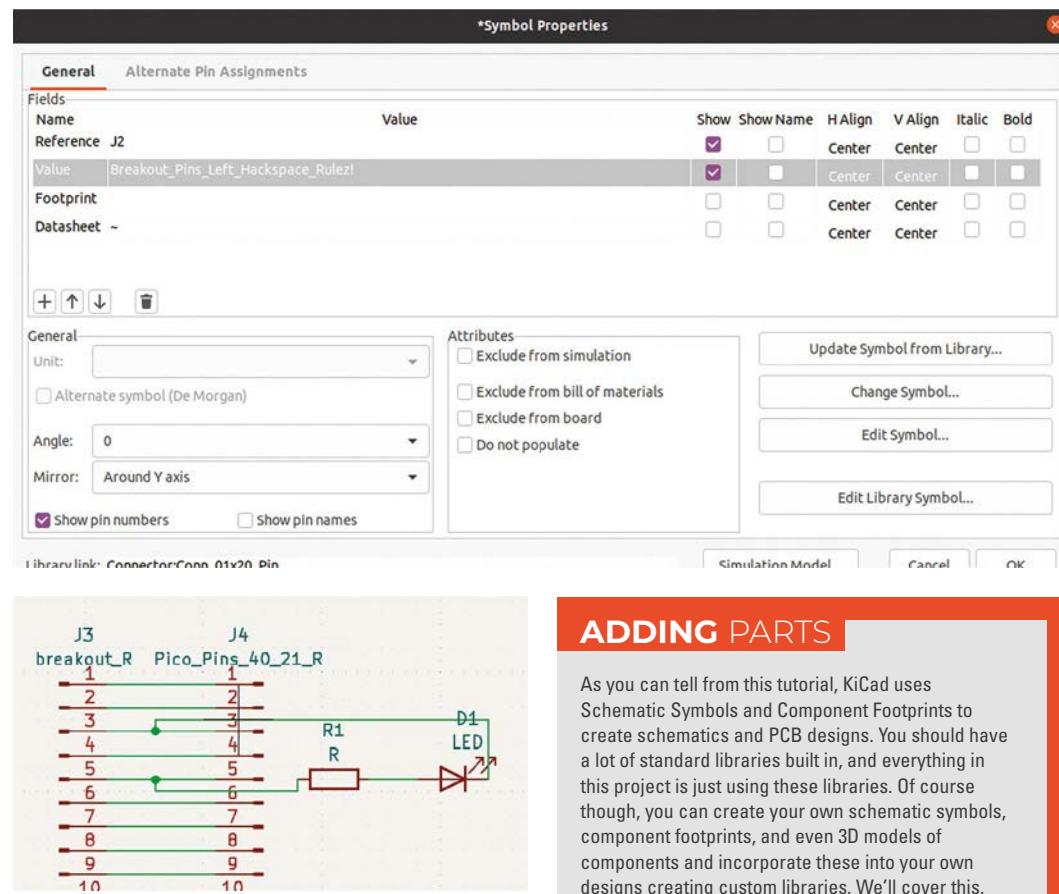
mean that, if we wanted to change the component on the PCB, we have to change the schematic. In KiCad, the schematic symbol has to be assigned a footprint – this is why we are going to place a symbol for a single-pole single-throw switch (SPST), but the actual component can be any SPST switch or any →

To finish our schematic, we are going to add more components and make more connections

**Figure 4** Adding our wire connections between the pins of the schematic symbols

# Getting started With KiCad, schematics

## SCHOOL OF MAKING



button package. In our case, we'll choose some type of momentary push-button for the reset. This way of working means that if you find you need to replace a component with a different type (commonly as you can't find a component in stock), you simply change the footprint associated with the symbol and don't have to edit your correct schematic. It also makes the schematics concise and readable, which is important if you want to share your design.

### SWITCHED ON

To add the switch, search the Choose a Symbol dialog with 'sw\_spst' to take you directly to the single-pole single-throw switch and again connect wires, as shown in **Figure 7**. Finally, add four connectors to act as our prototyping areas on the board; search for 'conn\_01x04' to take you directly to this symbol.

Let's now set up the associated footprints for each schematic symbol. Click the Run Footprint Assignment Tool icon. Similar to the Choose Symbol dialog, a window will appear with a list of footprint libraries down the left-hand side, a centre section with the schematic symbols of the project listed, and a right-hand column of filtered footprint results. You can see a collection of three icons at the top of

## ADDING PARTS

As you can tell from this tutorial, KiCad uses Schematic Symbols and Component Footprints to create schematics and PCB designs. You should have a lot of standard libraries built in, and everything in this project is just using these libraries. Of course though, you can create your own schematic symbols, component footprints, and even 3D models of components and incorporate these into your own designs creating custom libraries. We'll cover this, and lots more in future sections.

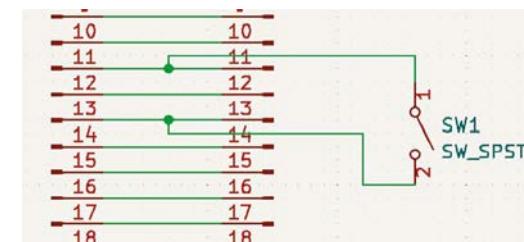
the window called 'Footprint Filters' and, in the first instance, make sure all these are selected. As you get used to component filtering, you might find you prefer to use different combinations of these filtering tools (**Figure 8**). Highlight in the centre section one of our 20-pin connector components. We need to find a footprint that has the same number of pins and has a footprint of a through-hole pad for each pin. As the Raspberry Pi Pico pins are spaced at the common 2.54 mm pitch between each pin, we also need to make sure our footprint is spaced similarly. You should

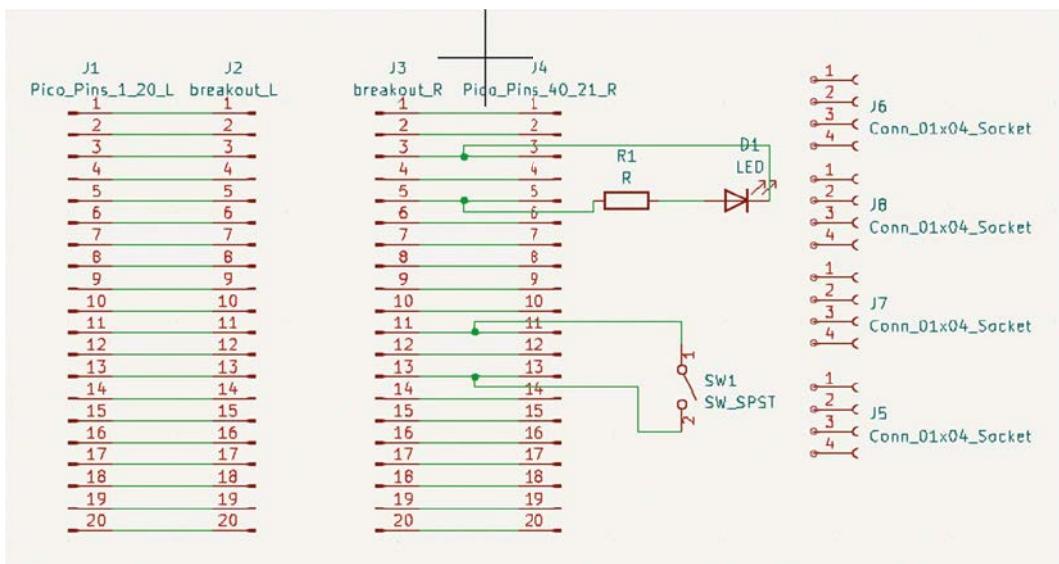
### QUICK TIP

Obviously, but remember to press the Save button now and again to save your work!

**Figure 7** ♦

Adding a single-pole single-throw switch to the schematic to work as a reset button for the Pico

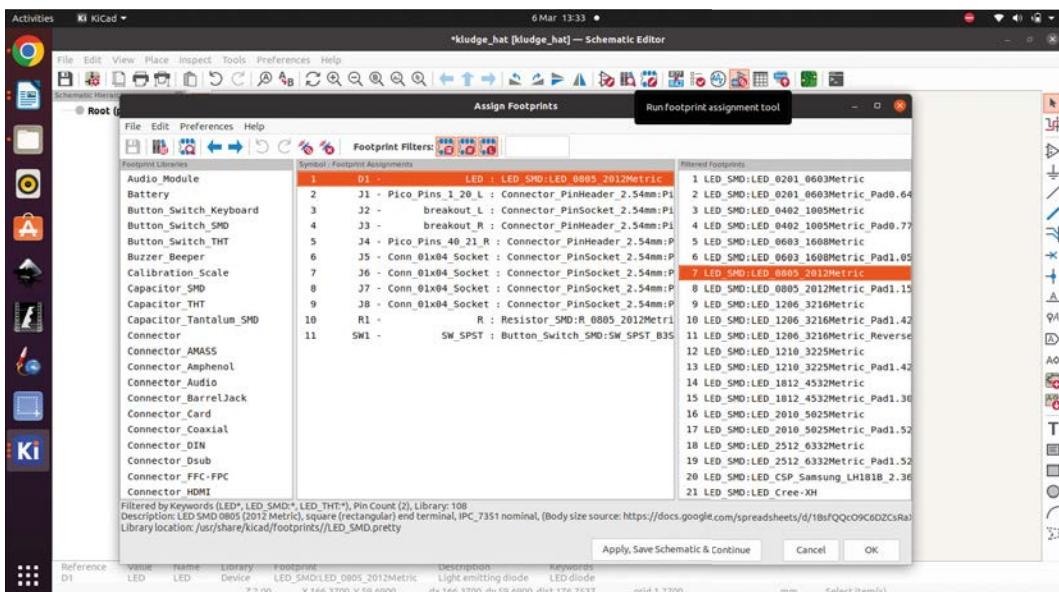




have a filtered list on the right-hand side – it should be filtered to only contain items that would attach to the 20-pin connector symbol. We chose item number 33 from the connector footprint library, which reads ‘Connector\_PinSocket\_2.54mm:Pinsocket\_1x20\_P2.54mm\_vertical’. A single left-click on the item highlights it in the list, and you can use the ‘View the selected footprint in the footprint viewer’ tool icon to open a window showing the PCB footprint layout design to check if it looks correct. Double-clicking the highlighted footprint should then add that footprint name opposite the schematic symbol in the central list. Continue and add the same footprint to all four connector symbol listings, although we’ll discuss later why that isn’t totally correct, and then click the ‘Apply, save schematic and continue’ button. We’ve decided

to add surface-mount components for the LED, resistor, and the button, although we have gone for SMD package sizes and footprints that would enable us to hand-solder these. For the LED, we chose the LED\_SMD:LED\_0805\_2012Metric footprint; for the resistor, the ‘Resistor\_SMD:R\_0805\_2012Metric’; and for the switch/button, a ‘Button\_Switch\_SMD:SW\_SPST\_B3SL-1002p’ component. For the detached 4-pin connectors, we simply selected ‘Connector\_PinSocket\_2.54mm:PinSocket\_1x04\_P2.54\_Vertical’ for each connector. With all those footprints assigned, click ‘Apply, save schematic and continue’.

In the next instalment, we will import the list of component footprints into the PCB Editor and physically lay out our board design ready for fabrication! □



**Above** Our complete circuit schematic

**Figure 8** The Assign Footprints dialog

# Getting started with KiCad, PCB layout

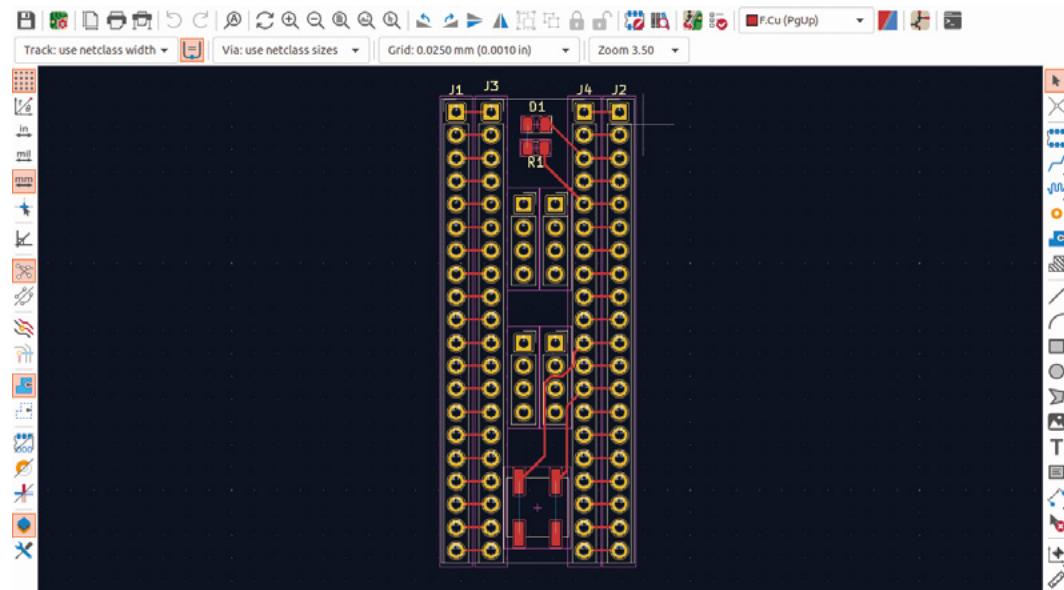
In this second part of the series looking at PCB creation with KiCad, we lay out our PCB ready for fabrication



**Jo Hinchliffe**

@concreted0g

Jo Hinchliffe is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high-power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and CNC kit!



**I**n the last issue, we got our schematic for our Pico add-on board created, and we assigned each schematic symbol a footprint which represents the individual component we are adding for each part of the design.

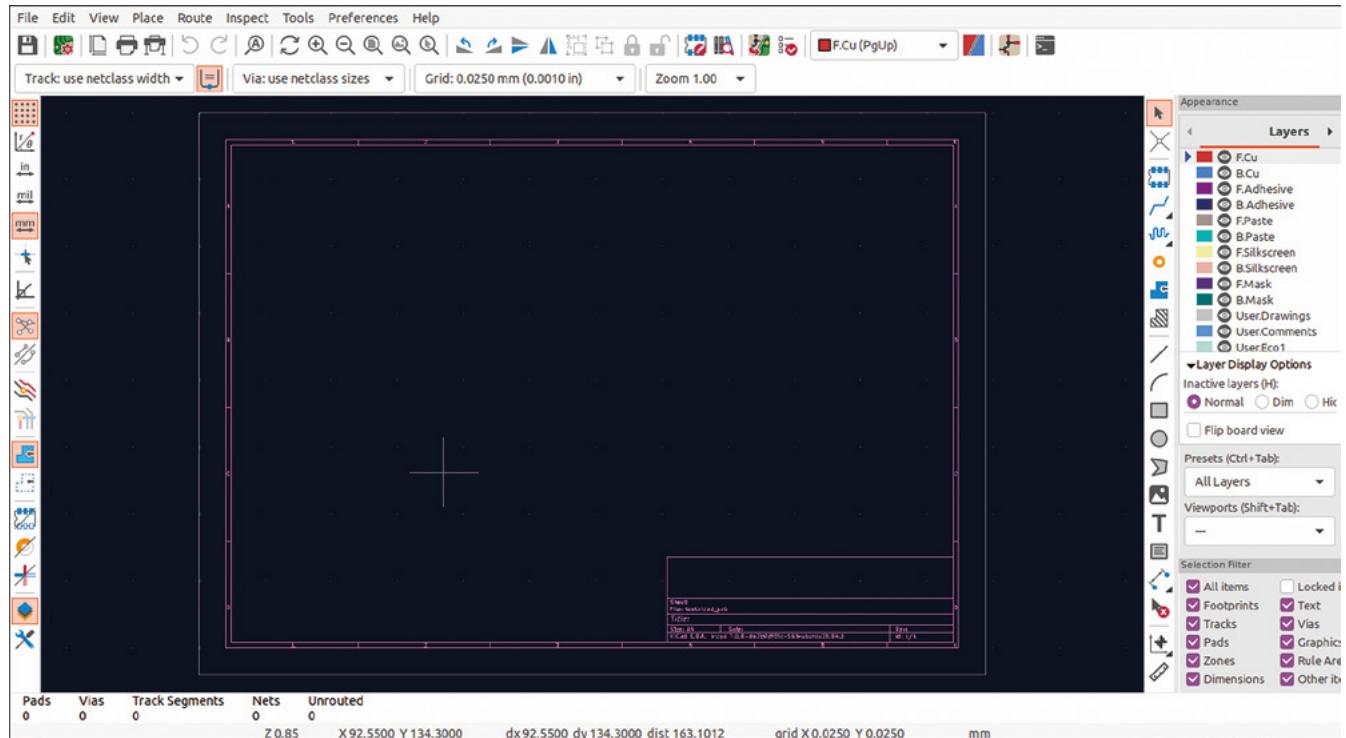
In this issue, we will finish this starter project by laying out the PCB design ready for manufacture. If you haven't opened KiCad, do so now. Open the previous project and select the PCB Editor icon from the available applications (**Figure 1**). If you have opened the project and the Schematic Editor, you can jump to the PCB Editor by clicking its icon in the Schematic Editor toolbar.

Having opened the PCB Editor, you should see a similar page to the Schematic Editor, but in black (**Figure 2**). You'll notice it is blank, so the first action is to pull in our footprints. To do this, we can click

the 'Update PCB from Schematic' icon. This opens a window, and we can simply click the Update PCB button and then click the Close button (**Figure 3**). Once back in the PCB Editor, we now should have our design as a collection of footprints attached to our pointer. Left-click to place the component bundle somewhere in the middle of the page (**Figure 4**).

The PCB editor has a grid feature which snaps footprints at useful points. In our current case, the pin socket footprints will snap with the centres of the pad holes on the grid. As the Pico conforms to 2.54 mm pin spacings, it's useful at this point to set the board editor grid to '2.54 mm (0.1000 in)'. You can select this from the centre drop-down menu on the upper toolbar. We can now align our two pairs of pin socket footprints so that they are in line with each other horizontally.

**Figure 1**  
Our completed  
routed PCB design

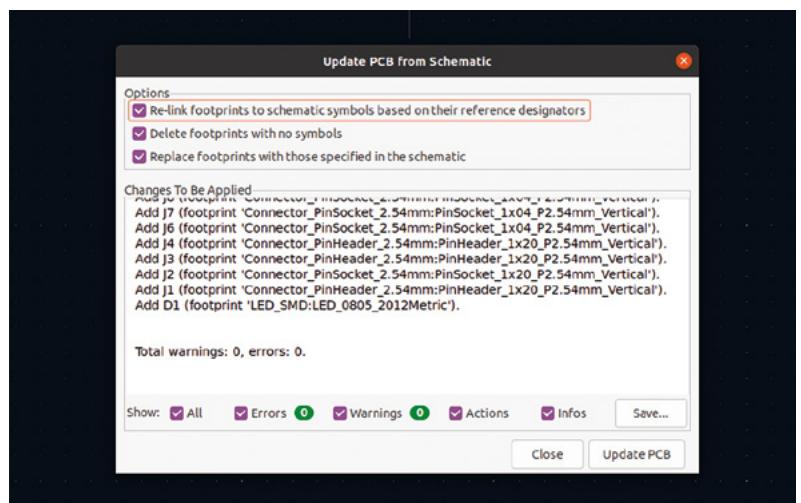


**Figure 2** The blank PCB Editor ready to import our footprints

Next, move one element of one pair so that it lies two grid dots away from its connected pair. You do this by clicking on the centre of a pad on the footprint and then pressing **M**, similar to in the Schematic Editor. If you only move it across one grid dot, they will overlap. You should see a collection of small white lines connecting the pads between the two connected footprints – this is called the ‘rat’s nest’. We will remove these white lines when we lay the trace connections. It’s important that our board matches up to a Pico, so we used the dimensional diagram (**Figure 5**, overleaf) taken from the Pico data sheet documentation, available here: [hsmag.cc/PicoDatasheet](http://hsmag.cc/PicoDatasheet). You’ll notice that the distance between the vertical rows of pins is 17.78mm – this equates to  $7 \times 2.54\text{ mm}$ . This is why your Pico fits perfectly into a prototyping breadboard. Move the remaining outer pin socket footprints so that it is on top of one of the pairs you have placed correctly, and then move it seven grid spots to the right. You can now arrange the last footprint to be two grid spots inside. The outside pair of footprints should now match a Pico’s pin →

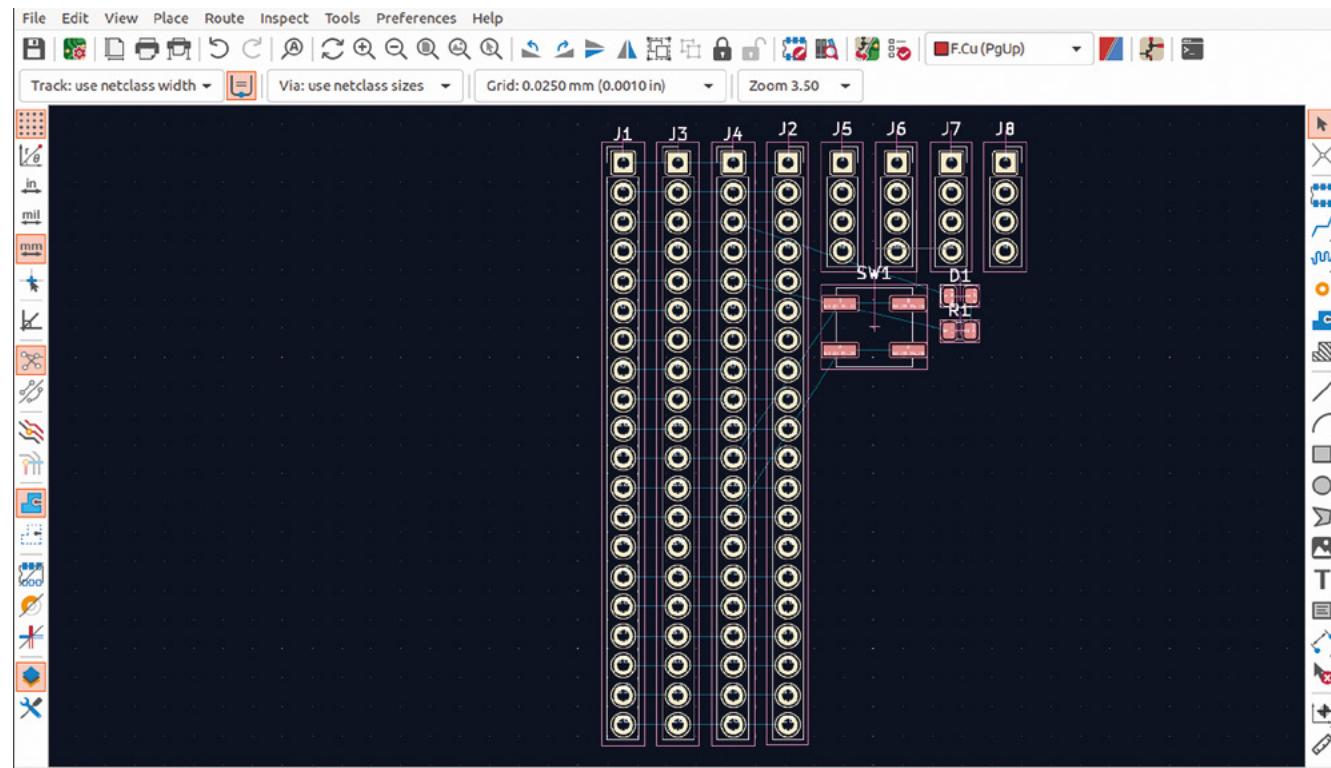
The PCB editor has a grid feature which snaps footprints at useful points

**Figure 3** The Update PCB from Schematic window primarily brings in the component footprints and connectivity, or can be used to apply later changes to a schematic design



# Getting started with KiCad, PCB layout

## SCHOOL OF MAKING



**Figure 4**

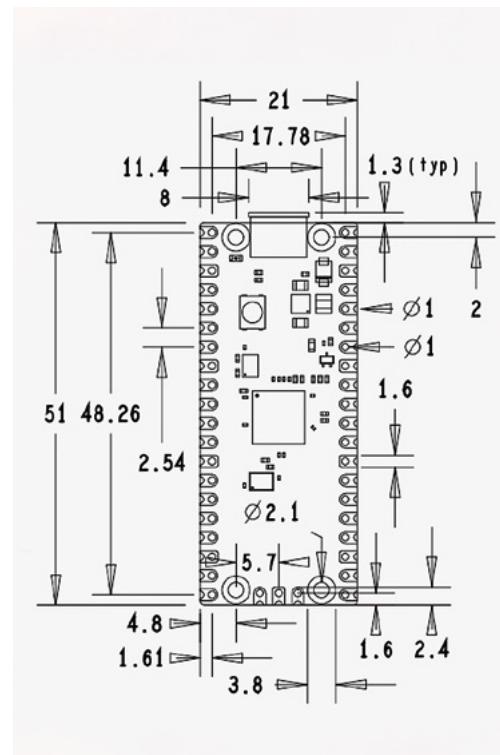
Our component footprints imported, with the rat's nest a representation of the connections between components and pads made of small lines

**Figure 5**

A technical drawing of the Pico taken from the data sheet gives us the dimensional information we need

### QUICK TIP

Remember, a lot of your keyboard shortcuts are universal. For example, F1 and F2 zooming works in both the Schematic Editor and the PCB Editor.

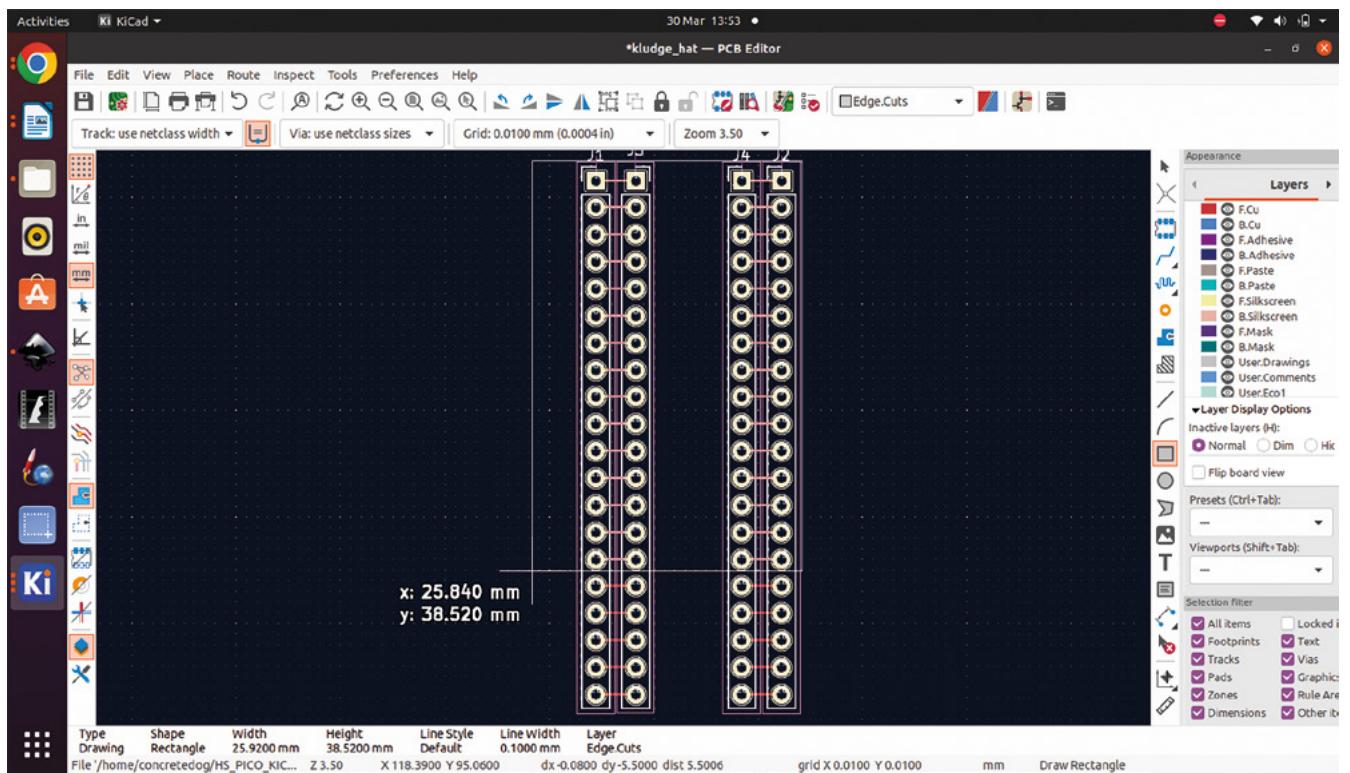


positions, and the inner footprints should be close to them, and parallel. So far, we should have only been working on the F.Cu (PgUp) layer, which is the top copper layer on our PCB board. Before we route the tracks between our footprints, double-check that you are on this layer by checking the drop-down menu on the top toolbar.

### LEAVE ONLY FOOTPRINTS

Next, we can wire the pads on the opposite footprints together. To do this, select the Route Tracks (X) icon, then click the centre of a pad and drag the track over to the centre of the opposite pad. The track should be red, indicating that it's on the top copper layer. If we, for example, were to move to the B.Cu (PgDn) layer, the default colour for tracks on the lower copper layer is blue. Continue and connect all pads together, noticing that the rat's nest lines disappear as you do so.

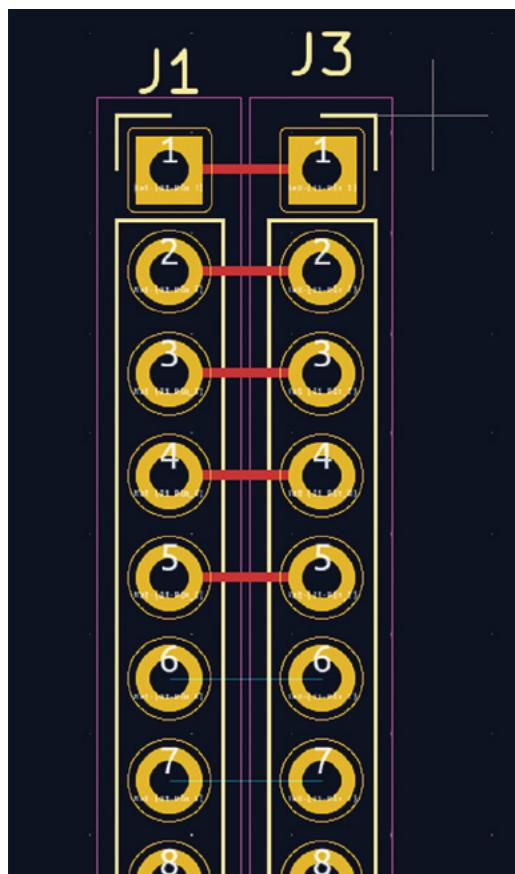
Referring to our Pico technical drawing, it's a good time to define the shape and edges of the board. To do this, we can use the uppermost drop-down menu to move from the F.Cu (PgUp) layer to the Edge.Cuts layer. On this layer, we can use the Draw a Rectangle tool to create a cutout shape for our board. Before we select this tool, let's switch



the grid to a more useful spacing. Select the 1 mm grid spacing, then left-click the rectangle tool. Left-click anywhere in the PCB Editor page and drag a rectangle (**Figure 6**). The rectangle should be snapping to the grid – we can drag it out until the labels tell us we are drawing a 21 mm width by 51 mm height rectangle. Left-click one more time to create the rectangle. Moving back to the Select Items (S) tool, we can now select the rectangle and press the **M** key to move it into position. Notice

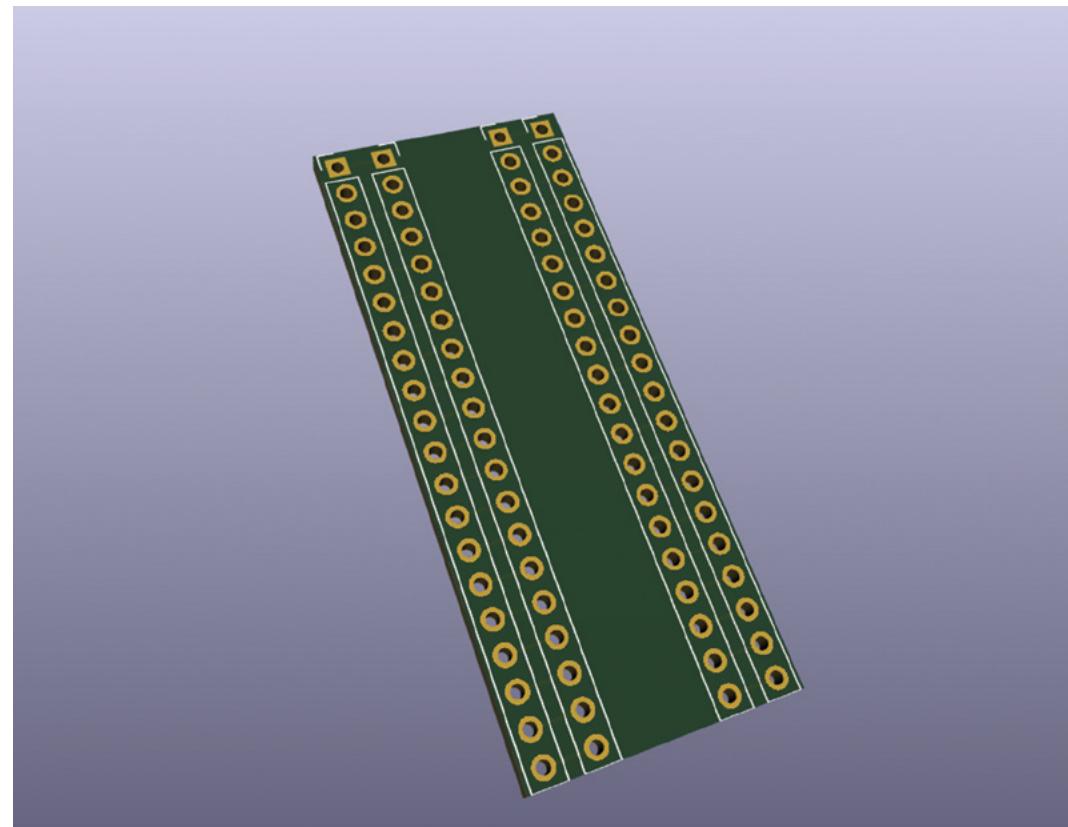
“  
Connect all pads together,  
noticing that the  
rat’s nest lines disappear  
as you do so”

from the Pico technical drawing that the outside edge of the Pico sits 1.61 mm from the centre of the pin pad position. To position this accurately, we reduced the grid spacing to the smallest listed and used a technique to measure distances on the page. If you press the **SPACE** bar at any time when in the PCB Editor, you might notice that values labelled ‘dx’, ‘dy’, and ‘dist’ are set to zero. This is very useful as you can place your pointer at a point, say the centre of the top rightmost pad, press →



**Figure 6**  
Drawing a rectangle that defines the cut edge of the PCB board

**Left**  
Routing tracks to connect the pads together



**Figure 7** A first glimpse of our rendered PCB board

### ASSIGNED AND SEALED

One point we mentioned earlier in this project is the idea that KiCad schematic symbols are generic – we assign the hardware footprint to them using the Assign Footprint tools. As an experiment, we could now show the advantage of this. Say we have our PCB layout completed and ready for manufacture, but after checking, we realise that our B3SL-1002P button package is not available or in stock anywhere. We can simply go back to the Schematic Editor and click the Assign Footprint tool icon. We can then edit the SW\_SPST symbol to have a different, and hopefully, in stock, component. For example, we selected the B3SL-1022P footprint package, double-clicking it to ensure it is added to the centre console list. You can then click Apply > Save Schematic > Continue, and then click OK to close the dialog. Moving back to the PCB Editor, you can then once again click the Update PCB with changes made to the schematic (F8) icon, or press F8 on the keyboard and the board will update with the replacement footprint added. If, as in this example, you replace the part with another part that is virtually the same footprint, you might not need to rewire the traces, but, of course, you should check and adjust connections and positioning as required.

#### QUICK TIP

There are lots of drawing tools to create more complex edge cuts in KiCad. Also, in future articles, we'll look at importing graphical elements from other drawing software, such as Inkscape.

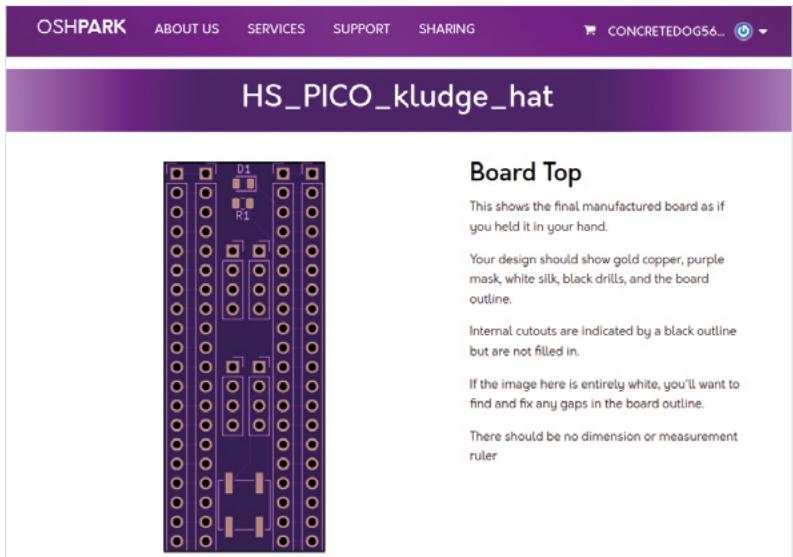
the **SPACE** bar to create a zero or datum point, and then move the pointer to, in this case, the edge cut rectangle. We can then use the dx and dy values to help us position this, or anything we need to, accurately.

Once we have positioned the board edge rectangle correctly, we can get a first glimpse of our board in the 3D Viewer. You can actually look at the board in the 3D Viewer before adding an edge cut, and it will render to a rectangle size that is the smallest that can accommodate all the footprints currently in the PCB. To view the PCB, click View > 3D Viewer. You'll see the board, but you'll notice that all the 3D models of the header sockets are all placed into the rows of holes. In reality, we would only want headers on the outer rows installed on the back of the board, with the inner rows left unpopulated, or possibly populated with header pins. We could edit the board and the component footprints to reflect this, but for this 'getting started' example, we can click the Toggle 3D Models for Through Hole type Components (T) icon to turn off the models (**Figure 7**). In future articles, we'll explore not only using correct 3D models but we'll

look at how we can add custom 3D models to our libraries.

### FLOOD ZONE

We can now continue to arrange and wire our remaining components. For the LED and the switch/button, we are wiring traces directly to the pin connections to the Pico – this is perfectly OK, but in future parts of the series, we will look at using copper flood zones. A copper flood zone is where an area on a layer of a PCB is flooded with copper that is connected to a certain ‘net’, with the net value being chosen by the user. This means, for example, that we could make a PCB where everything on the back layer is a GND-connected copper flood, then any pads that connect to ground simply join the



flood with small traces. This can dramatically reduce the amount of traces in more complex designs, making them easier to route.

Finally, how do we get this board made? Well, there are lots of options – we will look at many of them throughout these articles. Globally, there are lots of services available from companies to manufacture and even assemble your boards. These services may need different approaches in terms of what information and files you need to upload to get the job done. Often we need to export Gerber files for each layer of the PCB, and also drill files which show the position and size

**Above** We'll use numerous PCB fabricators in this series, but a great place to start, where you can directly upload KiCad PCB files, is OSHPark

**Below** Our complete PCB with all components placed and routed

In a few weeks – often less – you'll have  
**'Perfect Purple PCBs'**  
through your door!

of holes. Some companies might have limitations on what size holes they can produce and what tolerance they can produce the board too. We'll explore this in future articles, but the simplest way, if you wanted to get this board fabricated to an excellent standard, is you can upload the file that ends '.kicad\_pcb' to the OSHPark website ([oshpark.com](http://oshpark.com)). The website and service are brilliant. In-browser, it creates numerous renders of your board, which you can then inspect and check to see if they are correct before adding the PCB to your cart to be manufactured. In a few weeks – often less – you'll have 'Perfect Purple PCBs' through your door! ☐

# KiCad libraries, symbols, and footprints

In this third part of a series of articles around PCB creation with KiCad, let's look at adding or importing custom libraries, schematic symbols, and component footprints



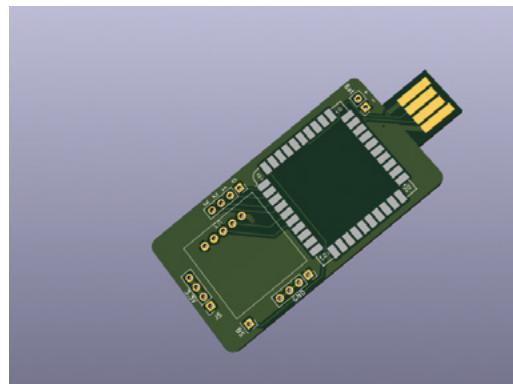
**Jo Hinchliffe**

@concreted0g

Jo Hinchliffe is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high-power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and CNC kit!

**In the first two parts of this KiCad series, we've covered enough to make a basic PCB suitable for simple circuits, or maybe a breakout board.** In this part, we are going to look at some next steps that open up the capabilities of what you can make in KiCad. We'll explore both creating libraries and contents from scratch, but also importing and using component footprints and schematic symbols from other sources. We'll also improve the quality of the boards by using flooded areas for common connections, such as all the circuit points that are connected to ground.

We've used a relatively simple design to show these techniques – making a PCB that essentially has two modules on board. Quite often, when making, we work with electronics modules on breadboards and a custom PCB can be the perfect way to take a breadboard project to a permanent home that's more rugged and usable. We aren't

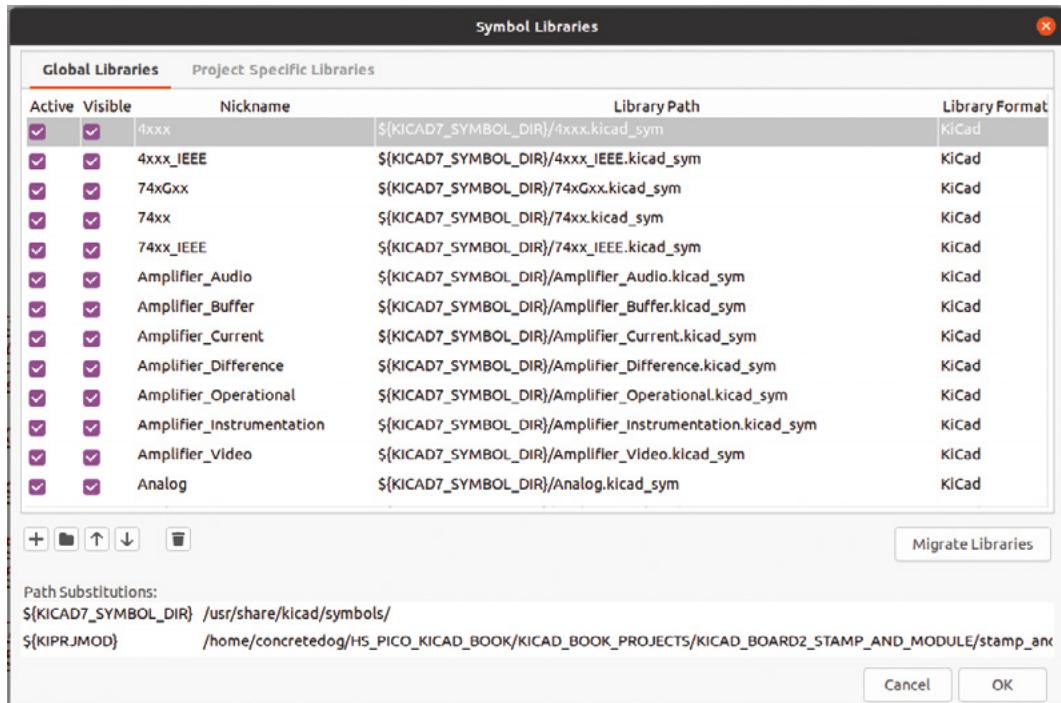


**Right** Our completed board, ready to receive a Solder Party Stamp and a Pimoroni BMP280 module

going to go step by step through the making of this board, but the project files are available at [hsmag.cc/issue68](https://hsmag.cc/issue68), and the knowledge and techniques we used in the first two parts of this series, in combination with this section, should give you enough knowledge to recreate this project.

We've chosen to use a Solder Party Stamp and a Pimoroni BMP280 module connected together so that the resulting board can be used to measure and log temperature and barometric pressure. The Solder Party Stamp is an excellent board that has an RP2040 at its heart, and is operationally similar to a Raspberry Pi Pico. The RP2040 is fully broken out to header pins which are castellated, so you can also solder the Stamp onto a recipient PCB's pads without having to use header pins. The Stamp also has the USB connection broken out as well as on-board LiPo charging. This means if we add a USB connection, we can also add a LiPo cell and make the project stand-alone. The Solder Party Stamp is really well-documented and is open-source. Solder Party has also published KiCad schematic symbols and a PCB footprint for the Stamp, therefore we can use it to learn how to add libraries and import these useful items into KiCad.

To begin, go to the following link where you will find the Solder Party Stamp library components: [hsmag.cc/StampFootprints](https://hsmag.cc/StampFootprints). Click the drop-down menu on the green Code button and then select the Download ZIP option to download the libraries. Unzip the files somewhere on your machine. In the collection of folders you just unzipped, cut and paste the entire **KiCad** folder (N.B. not the **KiCad 5** folder) to wherever you want to store your additional

**QUICK TIP**

In the project schematic, we have sometimes used net labels to create connections between symbols rather than direct drawn wires. We'll cover this in some detail in an upcoming article.

**Figure 1** The Symbol Libraries dialog where we can add or remove schematic symbol libraries

external KiCad libraries. We have a folder set up in our home directory for this.

Open KiCad 7 and, in the main page, click the Preferences drop-down menu and then select the Manage Symbol Libraries option. This should open a window with two tabs: the Global Libraries tab and the Project Specific tab, (Figure 1). Ensuring you are on the Global Libraries tab, find and click the small folder icon. Navigate to the folder we downloaded, extracted, and copy-pasted and open it to find a folder called **KiCad\_stamp\_lib**. Open this folder and select **RP2040\_Stamp.kicad.sym** and then click Open. You should now see a new library listed at the bottom of the Global Libraries tab called RP2040\_Stamp. If you create a new project and open the Schematic Editor, you can now use the 'Add a Symbol' tool to place a Solder Party Stamp symbol into the schematic. You can do this by searching for RP2040 and making sure you select the symbol from the RP2040\_Stamp library (rather than the stock RP2040 symbol) or by scrolling down the list of libraries, selecting RP2040\_Stamp, and then selecting the RP2040\_Stamp symbol.

It's a similar experience to add a footprint library. Again in the KiCad landing page, click Preferences and then Manage Footprint Libraries. Again, on the Global Libraries tab, click the small

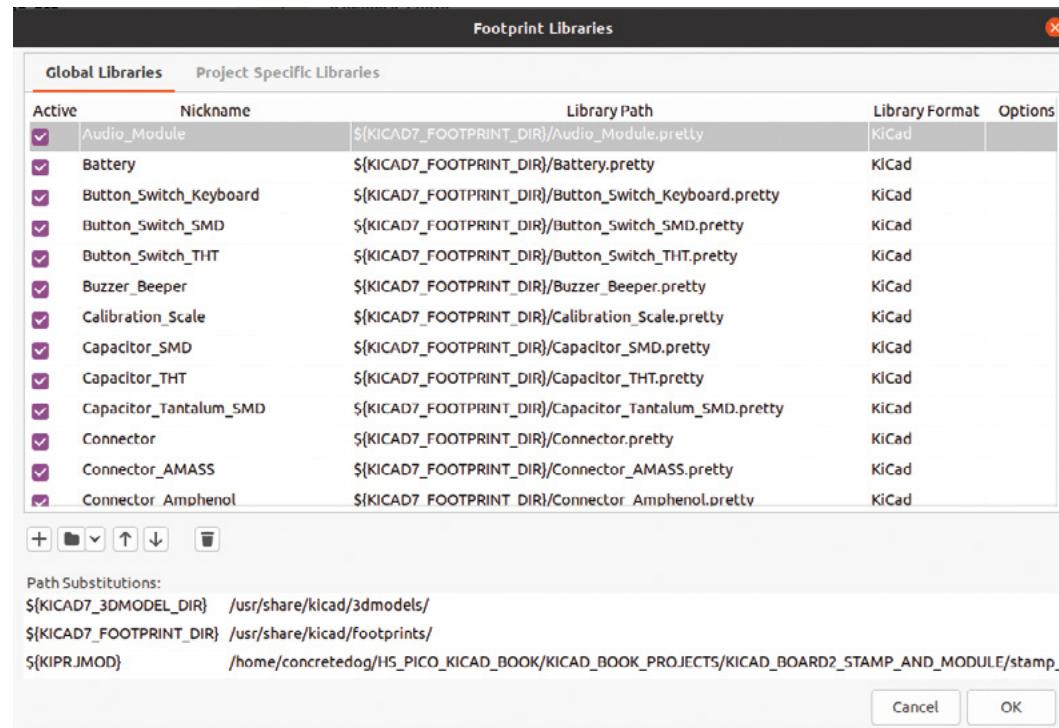
“ If we add a USB connection, we can also add a LiPo cell and make the project stand-alone ”

folder icon (Figure 2). Navigate to the folder we downloaded and find **KiCad\_stamp\_lib** – open that folder once more but, this time, select the **RP2040\_Stamp.pretty** folder and click Open. You should see three files inside, but you don't need to select any particular one – just click Open again. Now, back on the Global Libraries tab, you should be able to scroll down and see an RP2040\_Stamp library entry. You can check that this has all worked by associating the correct stamp footprint to the RP2040\_Stamp symbol we placed in the Schematic Editor, and then you can open up and import the part into the PCB Editor. If you need a reminder on how to do those tasks, check out the first and second part of this series in HackSpace #67 ([hsmag.cc/issue67](https://hsmag.cc/issue67)).

Using the BMP280 module gives us an opportunity to learn how to make both a custom schematic symbol and a custom footprint for the →

# KiCad libraries, symbols, and footprints

## SCHOOL OF MAKING



**Figure 2** The Footprint Libraries dialog where we can add or remove PCB footprint module libraries

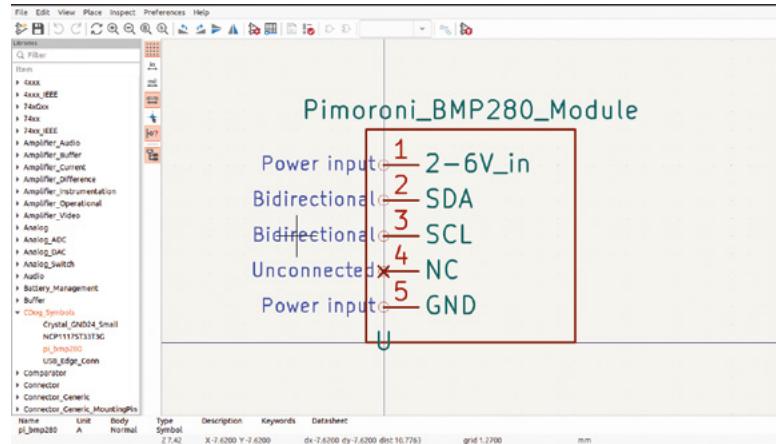
module to use in our board. To do this we will create our own custom libraries to contain these parts, and others in the future. Let's begin with a custom schematic symbol. In the Schematic Editor, click the 'Create, delete and edit symbols' tool button. This opens the schematic Symbol Editor.

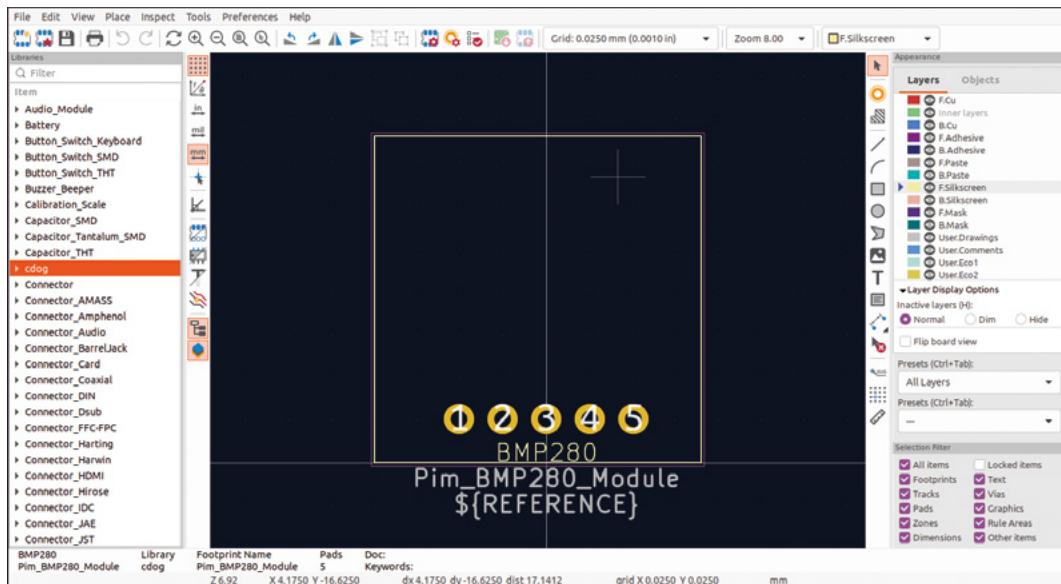
In this new window, click File and then select New Library from the drop-down menu. You should now see a small dialog box called Add To Library Table. In this box, you can select to add your new library to the Global table. This means any project in KiCad can access this library; alternatively, selecting Project means only this KiCad project can access that library. As the BMP280 is something we might use in other projects, let's make sure 'Global' is highlighted and

then click OK. You'll now be asked to give your library a name – it can be anything you want, so name it, making sure to leave the '.kicad\_sym' part of the file name intact, and click Save. You should now see your new symbol library name highlighted on the left-hand side of the Symbol Editor window. This means that this is the active library, so that when we select to create a new symbol, it will be stored in this library automatically. Click File and select New Symbol... from the drop-down menu. You should see a New Symbol dialog appear. Give your symbol a name that will be used in the library list – make it a useful name that reflects the part. We went for 'pi\_bmp280'. Clicking OK, you should now see that a 'U' has appeared in the Symbol Editor window and that the name of the symbol now appears in the active library in the list on the left-hand side of the screen. The name will have a '\*' next to it, indicating that the symbol has not been saved. In the Symbol Editor there are some familiar controls, such as the F1/F2 to zoom in and out. Zoom out a little to give yourself some room and let's get started by adding some pins.

Click the 'Add a pin' tool icon. In the dialog, we can name the pin, give the pin a number, and set the 'Electrical type' and change other settings if needed. For our first pin, let's name it '2-6V\_in', assign it pin number '1', and set the electrical type to 'Power input'. Continue to add pins 2, 3, 4, and 5, labelling them as you can see in **Figure 3**. As you place pins, notice that you can use the generic hot keys M for

**Figure 3** The Symbol Editor window can be used to create or edit schematic symbols





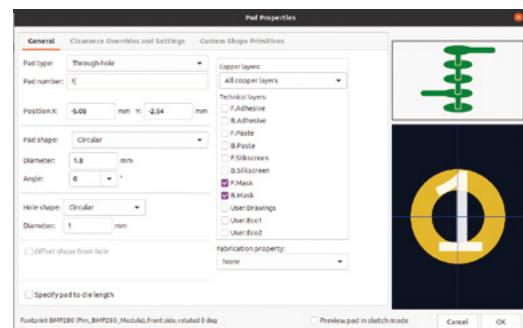
**Figure 4** ◆  
The Footprint Editor window can be used to create or edit component footprints

move and **R** for rotate, similar to the Schematic or PCB Editors. Once you have all your pins created, you can add a text label using the 'Add a text item' tool. This is useful so that you can identify the symbol quickly when looking at a schematic. Finally, let's draw a bounding box around our schematic symbol so that everything is neatly grouped together. Click the 'Add a rectangle' tool, and draw a rectangle over your design. Click the Save icon

“ Give your symbol a name that will be used in the library list – make it a useful name that reflects the part ”

in the top-left corner of the screen, and then close the Symbol Editor window. You can now go into the Schematic Editor and use the 'Add a symbol' tool to find and add your first custom symbol.

Next, we need to make a new footprint library and footprint for the Pimoroni BMP280 module. To begin, we first need to open the Footprint Editor (**Figure 4**). This is available either from the main KiCad project window, or indeed can be launched from the 'Create, delete and edit footprints' tool icon in the PCB Editor. Similar to the Symbol Editor, once you have the Footprint Editor open, the first thing to



**Figure 5** ◆  
Editing the pad size and shape using the Pad Properties menu

do is to click File and then select New Library from the drop-down menu. Again, you can select to add a new library to the Global or Project table – select Global and name your library. Once the new library appears in the list, highlight it and then click File and select New Footprint. A dialog appears and we can name our new footprint. We called ours 'Pim\_BMP280\_Module'.

We also need to select from the drop-down menu whether this is an 'SMD' or 'Through Hole' component. Selecting 'Through Hole', we are now ready to add the pads and other parts of our footprint. Similar to the PCB Editor, we can set the grid resolution and, as the pins on the BMP280 module are spaced at a standard 2.54 mm pitch, it is worth setting the grid to this initially to allow us to easily place the pins. Next, click the 'Add a pad' tool icon. We want to end up with five pads →

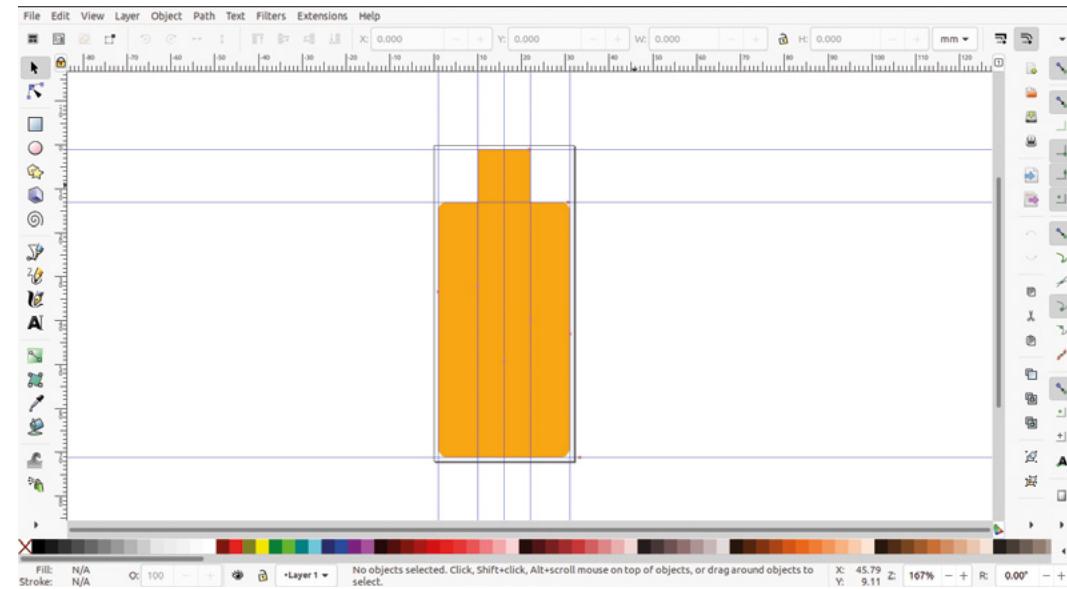
## QUICK TIP

Note that if you use a schematic symbol or a footprint module from an external library, once it is saved in your KiCad project file, the symbol/ footprint is stored in that project. If you moved to another machine with KiCad that didn't have the custom libraries added, you would be able to open the project as usual.

## SCHOOL OF MAKING

### QUICK TIP

If you aren't in the Schematic Editor, you can open the Symbol Editor from the main KiCad project window.



**Figure 6** Using Inkscape to create an accurate graphic for the PCB edge-cut geometries

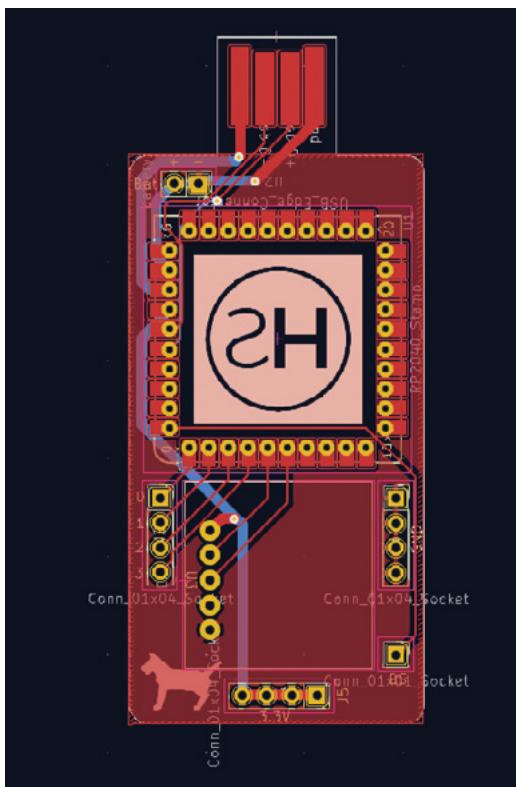
Many components will have physical package dimensions listed in their datasheet

## USB ON BOARD

We've now nearly got everything we need to make the Stamp and BMP280 board! You might have noticed in the main image that we have added a USB edge connector directly onto the PCB. This is a cheap and cheerful approach to adding USB without adding any extra components, although we will need to add around 1mm of material to the back of the 1.6mm thick PCB in this area to actually make the USB connector fit. The reason we added this is to highlight another way of using libraries and components from other, suitably licensed, KiCad projects. In this example we found a project, the USB Armory, which, in an early Mk1 version, used the USB edge connector on PCB approach. The USB Armory is an open-source project and we can download the project repository here: [hsmag.cc/USBArmory](http://hsmag.cc/USBArmory). Once downloaded, unzip the folder and use KiCad to navigate to the hardware folder, then mark-one, and then open the file `armory.kicad_pro`. Once open, move to the PCB Editor and select the USB edge connector footprint and press **E**. In the Footprint Properties window, click the Edit Footprint button. This should open the footprint in the Footprint Editor. You might get a warning that the footprint was made with an earlier version of KiCad, but saving the footprint in the editor should clear this warning. You should see that the footprints pads and the silkscreen line (which doubles as a guide for the edge of the PCB board cut out) are now opened in the Footprint Editor. Across the top of the editor you should see a warning that you are currently only editing the footprint within the current project. You can use File then Save As to rename and save this footprint into your custom library that we created earlier. As that library is created on the Global table, this USB edge connector footprint is now available to use in any project. We edited our version a little, labelling the pads more clearly, and saved it to our library. Making a note of each pad's connectivity, we then repeated the earlier approach to create a custom symbol in our library to represent the USB connector in the schematic.

labelled 1 to 5 moving from left to right. The easiest way to do this is to count two grid spacings out from the centre datum line and then click to place pad 1. Notice that the 'Add a pad' tool then increments the pad number so the next previewed pad is labelled '2'. Move one pad to the right of the pad you just made and click again. Continue to do this until you have a neat row of five pads.

Reverting to the general 'Select items' tool, hover over pad 1 and press the **E** key to open the 'Pad Properties' dialog (**Figure 5**). In this window you can change the geometry of the pad, the size of the hole, and many more options. We've found that increasing the pad size slightly from the default, and increasing the hole size, works well for soldering header pins between modules. You may have your own preferences, but we edited each pad to be a 1.8mm circle with a 1mm hole. With the pads created, we next need to add a silkscreen item that represents the physical area the module will occupy. Many components will have physical package dimensions listed in their datasheet, but that's not always the case with modules that are really designed for prototyping and breadboard use. In these cases, some investigation with a pair of callipers is a good approach to get some dimensions of the package. Measuring the Pimoroni BMP280, we realised a 19mm square area with the pin pads 2.54mm from the edge gave us a slightly oversized and therefore safe margin for the module. Select the 'F.Silkscreen' layer on the right-hand side of the screen and then use the 'Draw a rectangle' tool to draw this and position it correctly. Finally, let's add



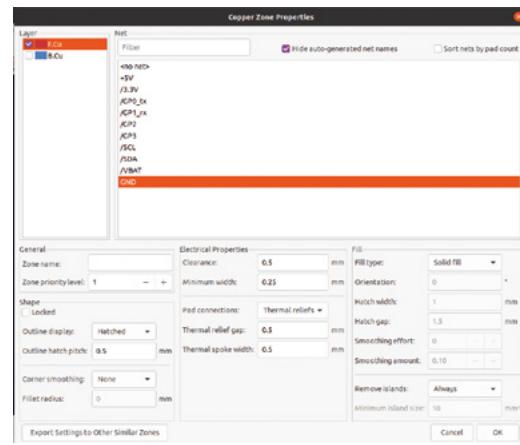
**Figure 7** Flooding the board to connect all the GND connected pads

another square on the ‘F.Courtyard’ layer that sits just slightly outside the silkscreen layer square we just created. You can achieve this by setting the grid to a very small spacing value such as 0.01 mm and then drawing a rectangle away from the silkscreen rectangle to avoid it snapping. Make the new square 19.4 mm and then you can use the **M** key to move the square into position. This new square in the front courtyard layer provides a service called the ‘DRC’ or ‘Design Rules Checker’ with a boundary that shouldn’t be overlapped. This means that later in the process, if a component is overlapping, this boundary on the PCB when we run the DRC will be highlighted as an issue. We’ll look at using the DRC in the next part of this series.

To create the board outline, we decided to not use the graphical tools in the PCB Editor but, rather, we imported an outline we drew in the free and open-source Inkscape application (**Figure 6**). Whilst the included KiCad tools are excellent, Inkscape can offer some advantages when designing graphic components. We drew a simple outline object for the board in Inkscape, saved it as an SVG file, and then imported it into KiCad. To do this, click File and then select Import > Graphics... from the drop-down menus. In the Import Vector Graphics

File dialog, you can navigate to the file and select the working layer to import to. Setting the graphic layer as ‘Edge.Cuts’, notice that we can also set an Import Scale value. In this instance we designed the board outline to be the correct size in Inkscape, so we leave the import scale at 1.00 so that it imports at its original size. This function is useful. However, if we have an oversized graphic or logo to import to a silkscreen layer, like the HackSpace logo, we’ve reversed in Inkscape and then imported it onto the back silkscreen layer. We’ll look at creating PCB artwork and components in Inkscape in more depth later in this series.

Finally, another technique that we’ve used on this board is to create flooded copper zones attached to a net label. This is an excellent way of automatically connecting groups of common connections (see **Figure 7**). You can create complex systems with lots of different flooded areas with differing connectivity, but in this project we have just used one flood on the front copper surface that connects all pads attached to ground. To do this, once the board is laid out, we used the ‘Add a filled zone’ tool. When you select this tool, you then left-click to start to draw a fill area over your board design, ensuring you are on the correct layer. A dialog appears and you can select the net connection from the list, so we selected ‘GND’ (**Figure 8**). Leaving all the other settings at the default values, we drew a rectangle over our board. Don’t worry too much about accuracy as the flood will only appear inside the edge-cut geometry. Drawing three points of your rectangle you can then right-click and select ‘Close outline’ from the drop-down menu. The rectangle (or other shape you drew) should now flood, and you should see that all the previously disconnected GND pads are now connected together – neat! □

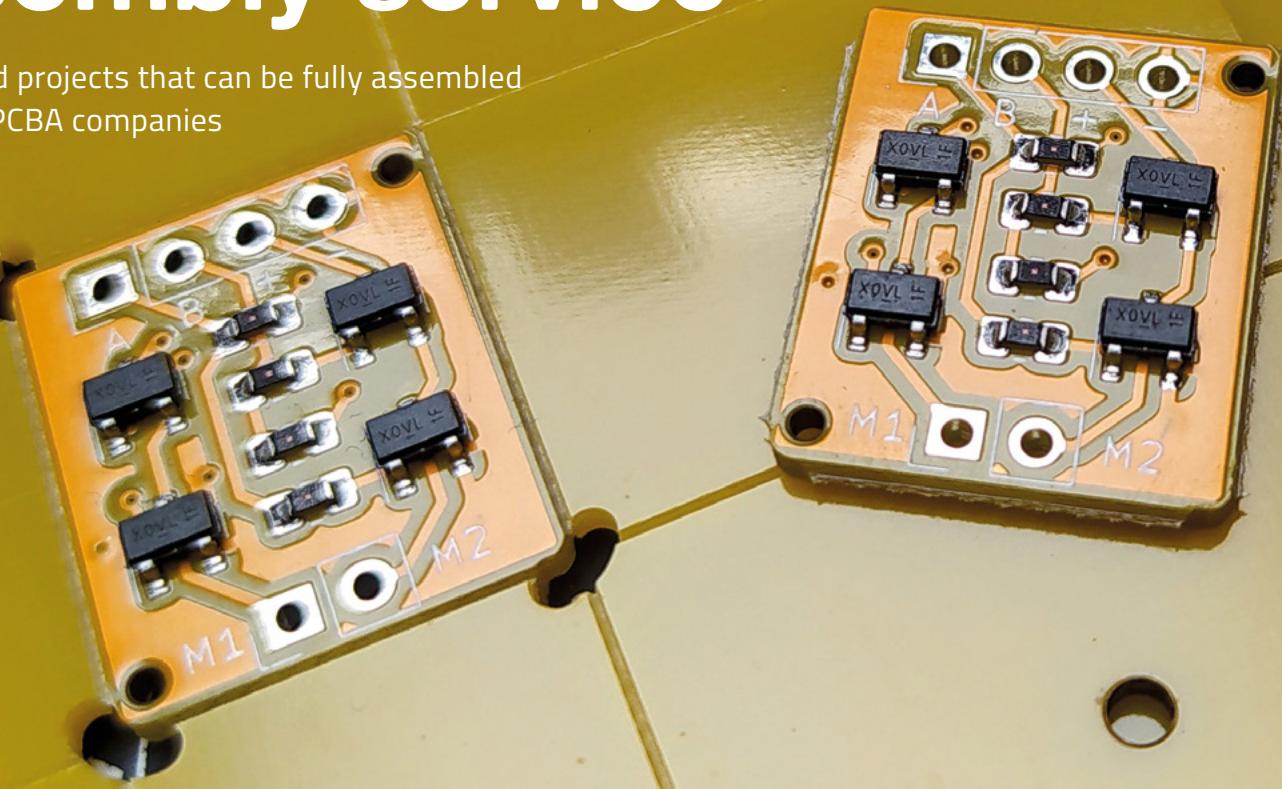


## QUICK TIP

Note that the schematic symbol pin numbers directly link to the assigned footprint pad numbers, so you must make custom symbols and footprints match.

# KiCad: using a PCB assembly service

Create KiCad projects that can be fully assembled by popular PCBA companies



**Jo Hinchliffe**

@concretedOg

Jo Hinchliffe is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high-power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and CNC kit!

## W

**e've covered a lot in the previous three parts of this series. If you've worked through them all, you should be at a point where you can create simple board designs**

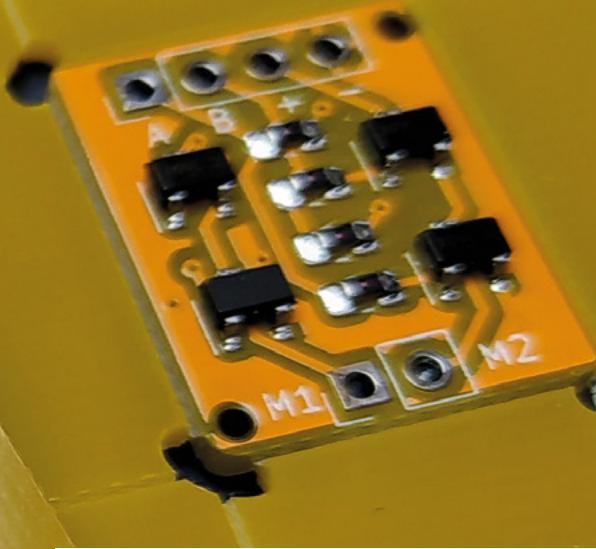
**pretty well.** In the next part of this series, we are going to look at a more complex board, building a minimal RP2040 board example. The RP2040 chip itself comes in a QFN-56 package, and whilst that can be soldered at home using reflow or hotplate soldering techniques, for many, that will be a challenge too far. To avoid this, we are going to use a PCB assembly (PCBA) service.

There is a lot to look at in making an RP2040-based board, so in this article, we will prepare a simpler design for manufacture to help us learn the PCBA approach. Designing for PCBA adds complexity in that we have to create numerous files, not only defining

the PCB design, but also choosing and placing known components on the board. These components have to be available to the PCB assembly house, which again adds a little complexity. It's fair to say the first few times you do this process, it will seem like a lot of work compared to simply uploading your project file to OSH Park for it to create and send you a PCB!

So, for this exploration of PCBA services, we've laid out a small design in KiCad for a little H-bridge circuit prototype using four N-channel MOSFETs. We aren't going to step through the board design, particularly as we've covered the approach in earlier parts of the series. You can check out the project files here: [hsmag.cc/issue69](http://hsmag.cc/issue69).

We're going to use the popular and reasonably affordable JLCPCB assembly service to manufacture and assemble our boards. This means that we need to consider what parts we are going to use on our board, as each part needs to be available in the JLCPCB



parts library. The first thing to do is to head over to JLCPCB ([jlpcb.com](http://jlpcb.com)) and register for an account. You can explore the JLCPCB parts library using the search function and filters – you will see parts' cost and availability. In fact, you can also advance purchase components so that they are held in your own virtual warehouse ready to be used on your board designs in the future. One thing of note is that available components fall into two distinct groupings: 'basic parts' and 'extended parts'. Basic parts will be added

**These components have to  
be available to the  
PCB assembly house, which  
again adds a little complexity**

to your board at the price listed. So, if you are adding five basic part resistors at 0.07 dollars each, then they will cost 0.35 per board. However, if that part was listed as an extended part, then that part will still cost the same unit price, but there will be a one-off \$3 setup cost of including that part in your project, as the part will have to be manually retrieved from storage and loaded into the pick-and-place machines.

As you peruse the JLCPCB parts library, make sure that if you spot a component that you are likely to use, you make a note of the part number – these usually start with the letter 'C' and are listed on the main component landing page. For our small H-bridge board example, we are only interested in having →

**Figure 1** Our PCB design successfully assembled by the PCBA service

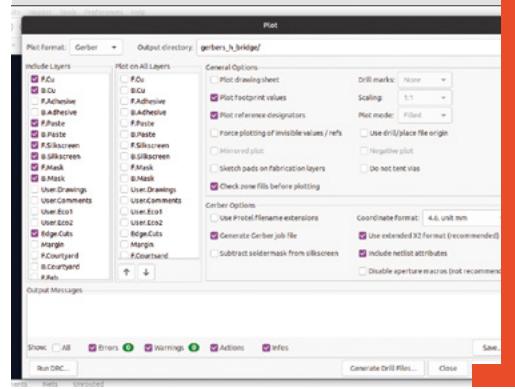
## PCB ASSEMBLY

In earlier parts of this series, we used OSH Park to manufacture our PCBs – they make it super-easy by accepting KiCad PCB file upload directly. If you want to use other PCB fabrication services, or indeed as we are in this article, a PCBA service, it's much more likely you will need to plot Gerber files for your PCB and also files containing drilling information.

Gerber and drill files have some variables, and your fabrication service should give you some information regarding what they need in terms of Gerber files and drill files. For example, JLCPCB has a page ([hsmag.cc/gerberdrillkiCad](http://hsmag.cc/gerberdrillkiCad)) which outlines the settings required by the Gerber plotter in KiCad that their service needs. It's a little out of date in that the screenshots are from KiCad 5.19, but you can find all the same options on the KiCad 'Plot' dialog. To access the latter, you click File > Fabrication Outputs > Gerbers from the PCB Editor.

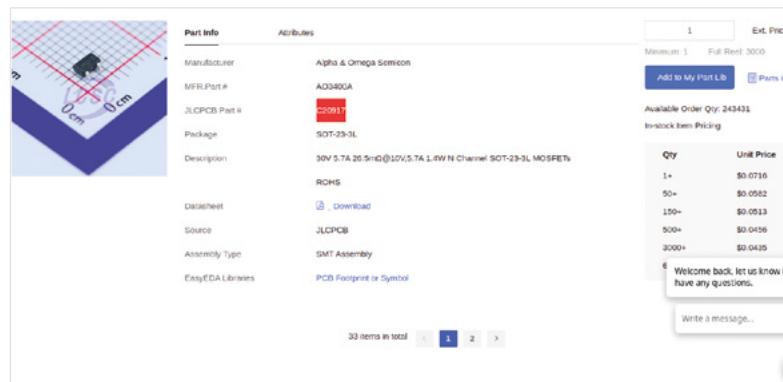
One thing of note is that plotting Gerbers creates a bunch of files for different layers of your PCB design, so make sure that at the top of the 'Plot' dialog, you create a folder for your Gerber files or else they all end up mixed into the main root folder of your project. You can also create the drill file from the 'Plot' dialog after you have plotted your Gerber files. Clicking the 'Generate drill files' button will launch a drill file dialog. For JLCPCB, place your drill file into the same folder as you did your Gerber files and, finally, compress this folder into a zip file for upload to JLCPCB.

Again, most PCB fabrication houses will have guidance on what format they require – JLCPCB list the settings needed on the link above. Don't worry too much if you upload something that doesn't work: it will show as an error and you can always ask the online chat service for help or guidance as to what settings to change.



# KiCad: using a PCB assembly service

## SCHOOL OF MAKING



**Figure 2**  
Searching for parts in the JLCPCB part library

**Figure 3**  
Adding an extra LCSC field to Symbol Properties enables a correct BOM to be created

JLCPCB add the SMD components, so we chose some 10K resistors (part number C49122), and the four MOSFETs are going to be AO3400 chips (part number C20917) (**Figure 2**). We need to add these details to an extra field added to the Symbol Properties so that later, when we generate a bill of materials (BOM), these specific components will be identified. In order for JLCPCB to ignore the through-hole components in our design, we've simply not added this extra LCSC field to their schematic symbols, and therefore they won't be included in the assembly process later on.

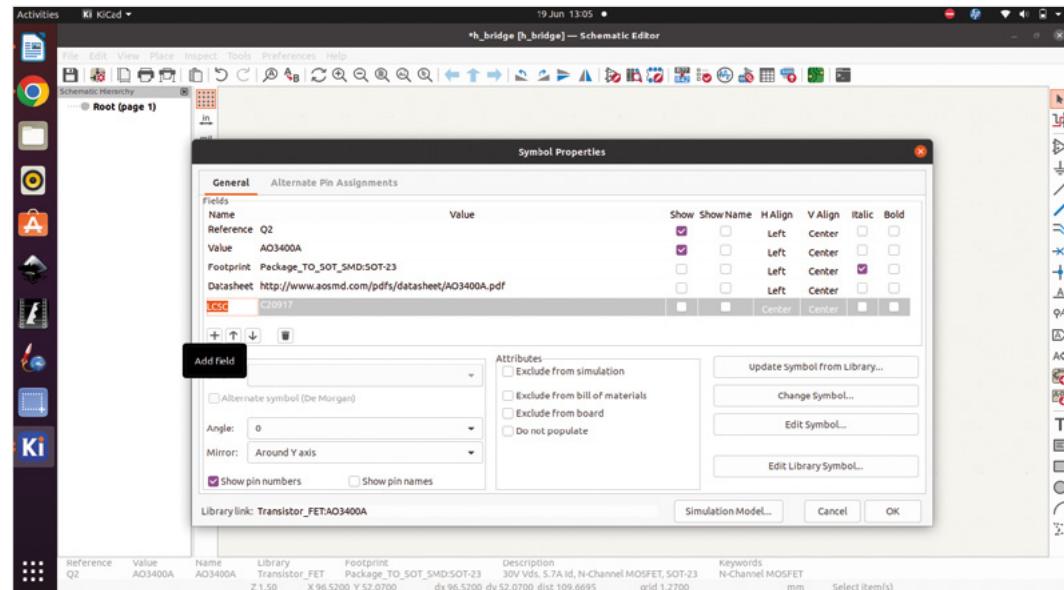
To add these details, in the Schematic Editor, highlight a component and press the **E** key to open the Symbol Properties dialog. Click the **+** button, which is labelled 'Add field' when you hover over it (**Figure 3**). You should see a new field line appear. In the 'Name'

You need to do this for every component you expect JLCPCB to add to your board

column, we need to label this field 'LCSC', and then in the 'Value' column, we need to add the CXXXXX number we researched for that component. For small projects, such as this example, you can do this manually for each schematic symbol. You need to do this for every component you expect JLCPCB to add to your board; you can't just add this field to one 10K resistor and expect it to work out to add the same part for the others. An alternate approach for larger projects is that you can edit the symbol at the library level, or copy the symbol to a custom library with the LCSC field and number populated at the symbol level. This means that whenever you place that custom symbol in the schematic, you have the correct LCSC part number ready for the BOM. Before creating the BOM, you still have to assign the footprints to the symbols as you would for any PCB design.

KiCad has a tool to generate a BOM – the tool icon is accessible from the Schematic Editor and shows 'Generate a bill of materials from the current schematic' when

you hover over it. If you click this tool, it will open the Bill of Materials dialog. On the left-hand side of the dialog, you'll see an area titled 'BOM generator scripts'. There are some scripts already installed, but there is an excellent script written by arturo182 which specifically creates a JLCPCB-formatted BOM. To use



### QUICK TIP

We covered creating and editing custom schematic symbols and libraries in the third part of this series.

```

<!--XSL style sheet to convert EESchema XML Partlist Format to grouped CSV BOM Format
Copyright (C) 2014, Wolf Walter.
Copyright (C) 2013, Stefan Helmert.
Copyright (C) 2018, Kicad developers.
Copyright (C) 2019, arturo182.

GPL v2.

Functionality:
    Generation of JLPCB PCBA compatible BOM

How to use this is explained in eeschema.pdf chapter 14. You enter a command line into the netlist exporter using a new (custom) tab in the netlist export dialog.
The command line is
    xlproc -o "%O.csv" "fullPathOfFile/bom2grouped_csv_jlpcb.xls" "%I"
-->
<!--
@package
Generates a JLPCB PCBA service compatible BOM

Functionality:
    * Generate a comma separated value BOM list (csv file type).
    * Components are sorted by ref and grouped by same value+footprint
    One value per line
Fields are
Comment,Designator,Footprint,LCSC

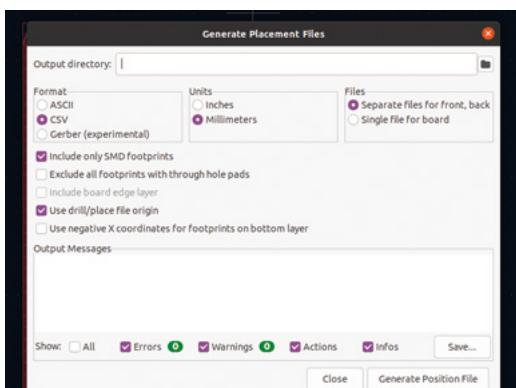
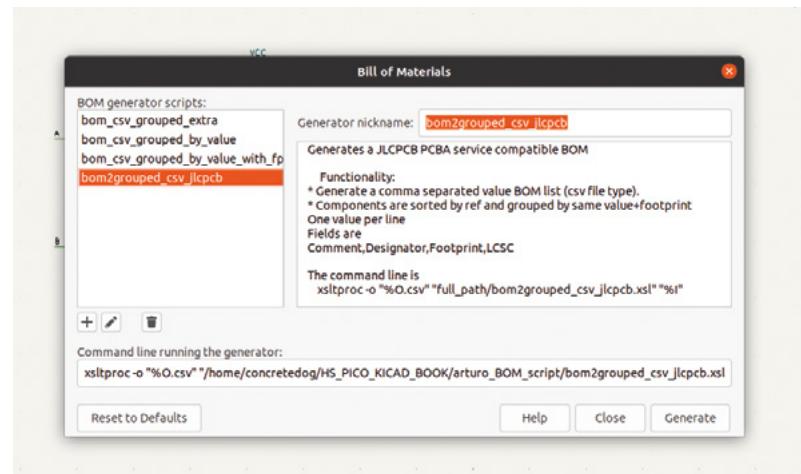
```

this script, go to the GitHub repository, then in the upper right-hand corner, you will see a Download ZIP button. Click it to download, then extract the zip file (**Figure 4**). In the Bill of Materials dialog, click the + button and navigate into the folder you just unzipped and select the **bom2grouped\_csv\_jlpcb.xls** file. Clicking the OK button will add this handy script to the list.

## BOM SQUAD

Once your board is complete and you have all the LCSC numbers attached to the schematic symbols, you can reopen the BOM dialog. Select the 'bom2grouped\_csv\_jlpcb' script, then click Generate to create a BOM that the JLPCB service will be able to use (**Figure 5**).

The final piece of the PCBA puzzle is to generate a footprint position file, also referred to as a centroid file. Similar to the BOM file, this is essentially a spreadsheet which contains details of each placed component, showing both coordinates and the rotational angle of the part. To generate this, in the PCB Editor, we need to select File > Fabrication



Outputs > Component Placement. Make sure that your settings match the dialog box shown in **Figure 6**. Note that if your project contains through-hole components that you want JLPCB to include, you need to uncheck the 'Include only SMD footprints' option and include the LCSC numbers for those through-hole parts in the BOM. Note that for this component footprint POS file and the BOM file we generated earlier, we don't want those files inside the zip file of Gerbers, as they are uploaded separately. Once you are ready, click the Generate Position File button. Notice that this will generate two POS files: one for the upper layer and one for the lower layer. As our design is single-sided, we are only interested in – and later, need to upload – the upper layer file. With the upper layer POS file generated, we need to make a few alterations to the spreadsheet column header titles for it to work properly for JLPCB. Open the file you have generated in your spreadsheet program. We use LibreOffice Calc, but MS Excel or Google Docs →

**Figure 4** ◇  
Downloading the custom JLPCB BOM generator script

**Figure 5** ◇  
The BOM dialog with the custom JLPCB generator script added and selected

**Figure 6** ◇  
The Generate Placement Files dialog

# KiCad: using a PCB assembly service

## SCHOOL OF MAKING

**Figure 7** Using LibreCalc to edit the generated positional file column titles

The screenshot shows a LibreOffice Calc spreadsheet with the following data:

	A	B	C	D	E	F	G	H
1	Designator	Val	Package	Mid X	Mid Y	Rotation	Layer	
2	Q1	AO3400A	SOT-23	96.078	-50.1165	90	top	
3	Q2	AO3400A	SOT-23	104.4	-50.3705	90	top	
4	Q3	AO3400A	SOT-23	96.078	-55.5475	90	top	
5	Q4	AO3400A	SOT-23	104.328	-55.5475	90	top	
6	R1	10k	R_0805_2012Metric	100.4335	-49.022	0	top	
7	R2	10k	R_0805_2012Metric	100.4335	-53.848	0	top	
8	R3	10k	R_0805_2012Metric	100.4335	-56.134	180	top	
9	R4	10k	R_0805_2012Metric	100.4335	-51.308	180	top	
10								
11								

should work. We need to make the following changes: the title of the first column, Ref, should be changed to Designator; PosX and PosY should be changed to Mid X and Mid Y; Rot should be changed to Rotation, and finally, Side should be changed to Layer (**Figure 7**). Save the POS file with these alterations – we now have everything we need to upload to the JLCPCB service.

With everything ready, we can now head to the JLCPCB website. Click on the Standard PCB/PCBA tab to upload our Gerber zip file. After a short upload, you should see a render of the upper and lower sides of your board in the preview window (**Figure 8**). You can make changes to the board type, material, thickness, and more on this initial page. However, apart from changing the colour of the board to yellow, we left everything at the default setting.

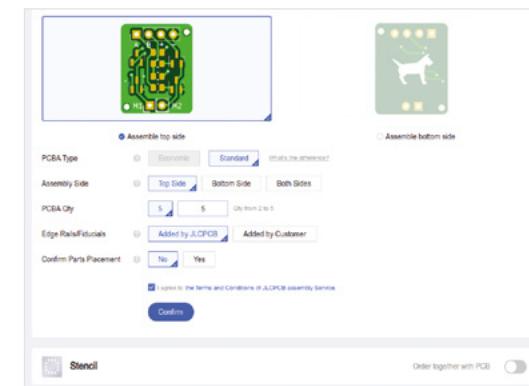
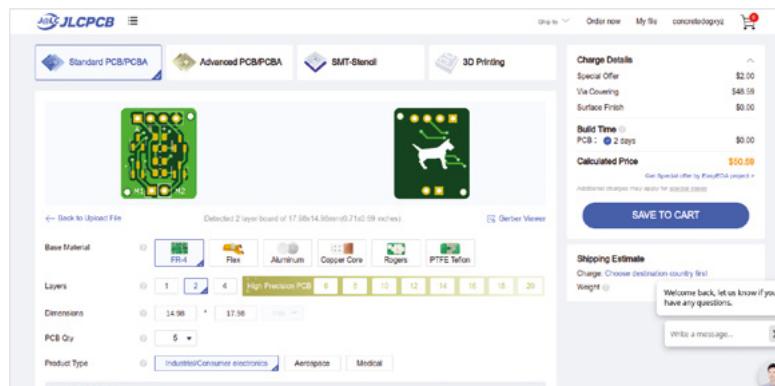
**Figure 9** The preliminary choices for the PCBA service

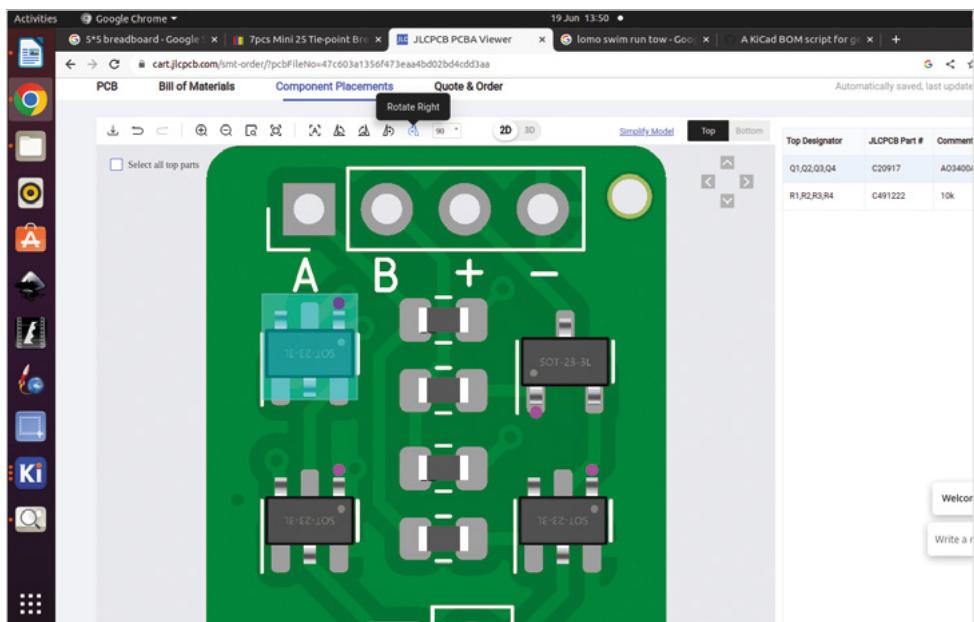
**Figure 8** If the Gerber zip file uploads correctly, you'll be rewarded with the first of many preview images of your board

You should see a render of the upper and lower sides of your board in the preview window

At the bottom of the page, you can click a button to add/expand the PCBA services. In **Figure 9**, you can see that we have opted to have the top side assembled only (as we only have components on this side). We've also ensured that the Edge Rails/Fiducials entry has the Added by JLCPCB option highlighted. This indicates that for our very small boards, JLCPCB services will create any needed panel layouts. With all that selected, click Next.

You'll get another larger preview of the PCB layout generated from the Gerber uploads. Check it carefully and then click Next to move to the next tab. This will look like **Figure 10**. It's reasonably self-explanatory. Click the Add BOM File button and upload the BOM file we created earlier, then click the Add CPL File button and upload the top layer CSV positional file we made and edited earlier. Clicking



**Figure 11**

Correcting footprint rotational position can be done in-browser as part of the JLCPCB order process, or you can edit your positional file offline to create correct values

Next, these will be uploaded and processed, which may take a few minutes, and you should see a render with the board and the components placed.

### SPIN ME ROUND

Often, at this point, you will find that components are not rotated correctly on the footprints. There are two ways to correct this, if your components are at standard angles, you can left-click to highlight a component in the render image and, when highlighted, use the rotation tools above the PCB render in the JLCPCB web page. You can continue to do this until your design looks correct. Clicking Next will save these orientations and take you to the Add to Basket ordering page. Whilst this is a fine approach, another approach is to open the positional file we created and uploaded offline and edit the rotational value of the components that have appeared incorrectly in the render. In our experience, either way is fine, and the JLCPCB engineers will question if a component isn't sitting on a footprint correctly.

All that's left to do is to add the order to your shopping basket and pay! Once the order is confirmed and paid, you get regular updates on the order listing and, if it's a complex design, it's worth checking back into your account four to six working hours after placing the order to check the DFM analysis regarding component placing in the order history details.

Once everything is ordered, all you have to do is wait! However, not for that long! We started this simple H-bridge motor driver design in KiCad and had the assembled PCBs (**Figure 1**) in our hand eight days later. ☐

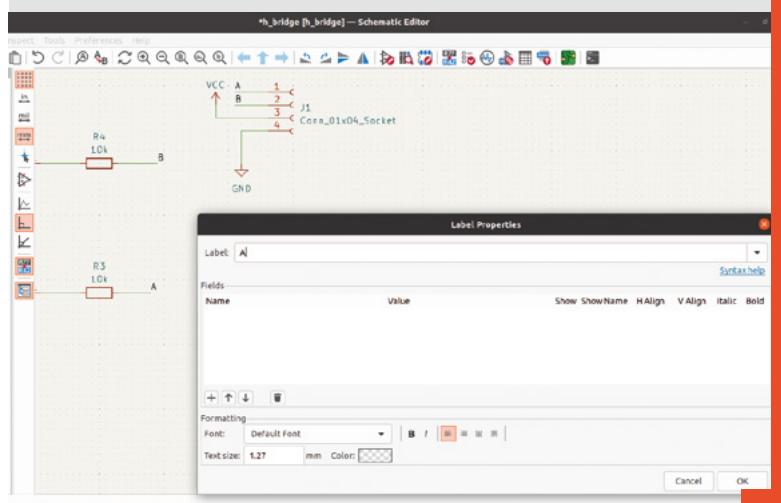
**Figure 10**

Adding the BOM files and the positional file we made earlier



## CASTING A NET

In the schematic for our H-bridge circuit, you'll notice that we haven't directly wired everything together and that the connections between the gate pins on each MOSFET are labelled 'A' or 'B'. In turn, there are also two pins on the connector marked 'J1' labelled 'A' and 'B'. These are 'net labels' and, to make one, you simply click the 'Add a net label (L)' tool icon. You then left-click in the schematic and the 'Label Properties' dialog will appear. Into this you simply type the name of your net label, so, for example, type the letter 'A'. You can change the font and size, but when you are happy, click the OK button and you can now place your 'A' label into the schematic. Notice that it has a small square connector. You can then connect a wire to the 'A' label as you would to any other component using the 'Add a wire' tool. You can recreate another 'A' label using the same method to attach to the other end of your wireless connection, or you can copy and paste the original net label. If you have more complex descriptive net label names, it can be a good idea to use copy and paste as if you create a net label with a spelling mistake, it will not connect and may take you a while to discover the error. In this simple H-bridge example, we didn't really need to use this technique but, in the next article with a more complex design, using net labels can really help to keep a schematic cleaner and more readable.



# Designing an RP2040 board using KiCad

Build a custom microcontroller board and get it assembled



**Jo Hinchliffe**

@concreted0g

Jo Hinchliffe is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high-power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and CNC kit!

In the previous parts of this series, we've worked through the basics of making PCB boards from schematic to board layout, and we've learned a variety of skills and approaches along the way. We built on this by exploring how to use KiCad and a PCB assembly (PCBA) service to not only have the PCB manufactured, but to also be populated with components and supplied to us fully assembled (Figure 1). If you've worked through the earlier parts, we now have enough skill and knowledge to tackle a more extensive project like creating an RP2040-based board.

RP2040 is a great target for PCB projects that will be assembled using industrial approaches. The bolder amongst us might successfully be able to solder up the QFN (Quad Flat No-leads) 56 package at home, but it's at the edge of where PCBA starts to make a lot of sense. Many commercially available boards that use the RP2040 are manufactured using PCBs with four or more layers. Of course, it's possible to do this in KiCad, but for many of us, four-layer boards can be difficult to debug or correct if something goes wrong.

RPI2040 A microcontroller by Raspberry Pi  
**Hardware design with RP2040**  
 Using RP2040 microcontrollers  
 to build boards and products

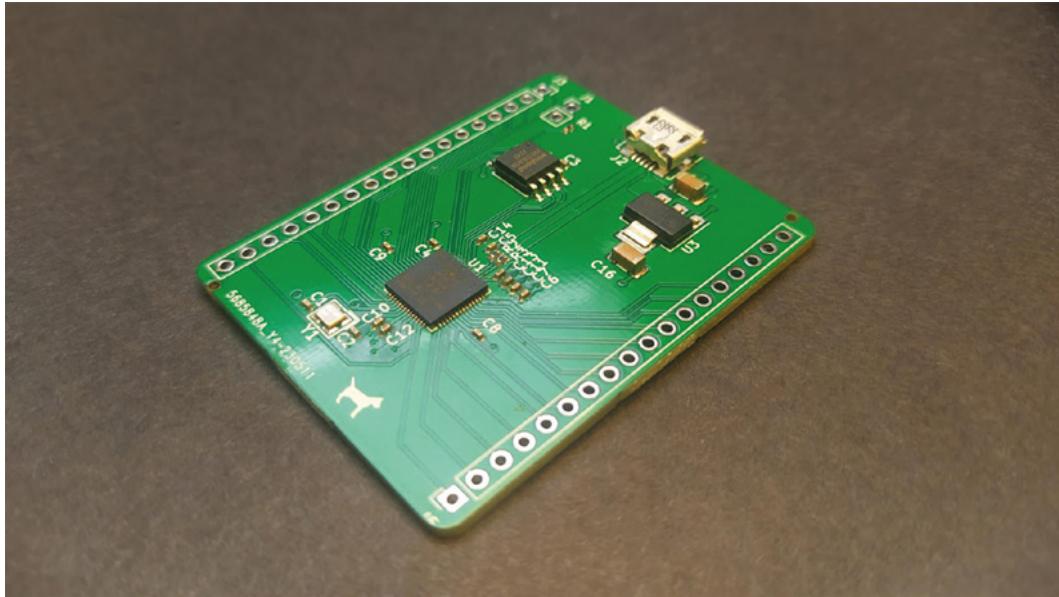
Fortunately, for simpler boards, two layers is enough to get our microcontroller running and break out the features we need.

When the RP2040 was released, so too was a stack of excellent documentation, including the very readable 'Hardware design with RP2040' (Figure 2). The PDF is available to download from [hsmag.cc/hardware\\_design\\_rp2040](http://hsmag.cc/hardware_design_rp2040).

This document shows an example of a minimal RP2040-based board, a two-layer design, and then describes various aspects of the design and considerations. There is even a KiCad project file for the design. It's a great idea to download the project file and take a look around it (Figure 3). In this article, we are going to replicate this board design, but we will start from a blank project rather than use the one supplied.

Starting from scratch means that it's easier to adapt this project to your own needs. We'll also tweak the project because currently, JLCPCB doesn't supply some of the exact components used on the Raspberry Pi example. The Raspberry Pi example was built in a previous version of KiCad and uses in-house schematic symbols for the RP2040. Since then, the RP2040 has become a standard library component within KiCad, but some of the connections on the RP2040 schematic symbol are already made at a symbol level (like common power pin connections), so it makes sense to use the built-in KiCad symbols and footprints. With that all said, we aren't going to step through every stage of designing this board as we've learnt the approach in the previous parts of the series, so this article

**Figure 2** The excellent Hardware design with RP2040 documentation from Raspberry Pi



**Figure 1** A fully assembled and functional RP2040-based board

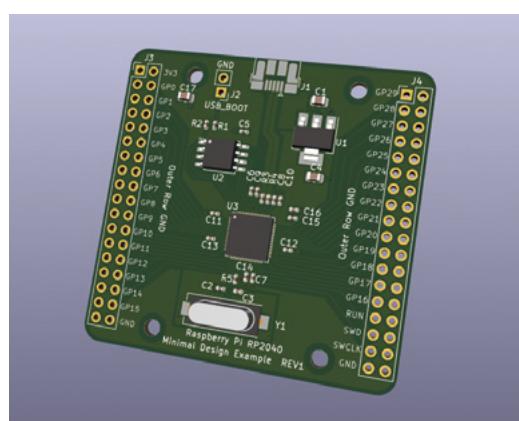
looks at things we consider specific to creating an RP2040 board.

We've tended to emulate both the schematic layout and the PCB layout of the minimal design example and our project. Partly this is because the PCB layout has neatly solved a lot of the layout complexity, but also it allowed us to compare the project as we worked through building our own. With the USB symbol (and footprint) sourced, we added the RP2040 component from the KiCad

**Starting from scratch  
means that it's easier  
to adapt this project to your  
own needs**

library. We added resistors and a pair of labels to connect the D+ and D- to the correct pins on the RP2040. The documentation states that we need to create the traces for these connections with accurate dimensions and clearances. We'll achieve this by assigning a custom net class to the connection, or 'net', so that when we draw these traces in the PCB Editor, they should be created correctly.

In the Schematic Setup dialog in the Schematic Editor, click the + button in the uppermost Net Classes window, add a new net class, and give it a name. Note that local net class names within a project should start with a '/' – we went with /USB\_lines. Select a wire segment that connects the RP2040 D+ or D- pin out to the USB\_D+ or USB\_D- label that we added, and then right-click. Select Assign Netclass from the drop-down menu. In the Add Netclass Assignment dialog box, you should see the selected label USB\_D+ and, to the right of it, a drop-down menu to select the net class – this is currently 'Default', but if you click down, you →



**QUICK TIP**  
Reading through the Hardware design with RP2040 PDF a couple of times is beneficial to laying out an RP2040-based project!

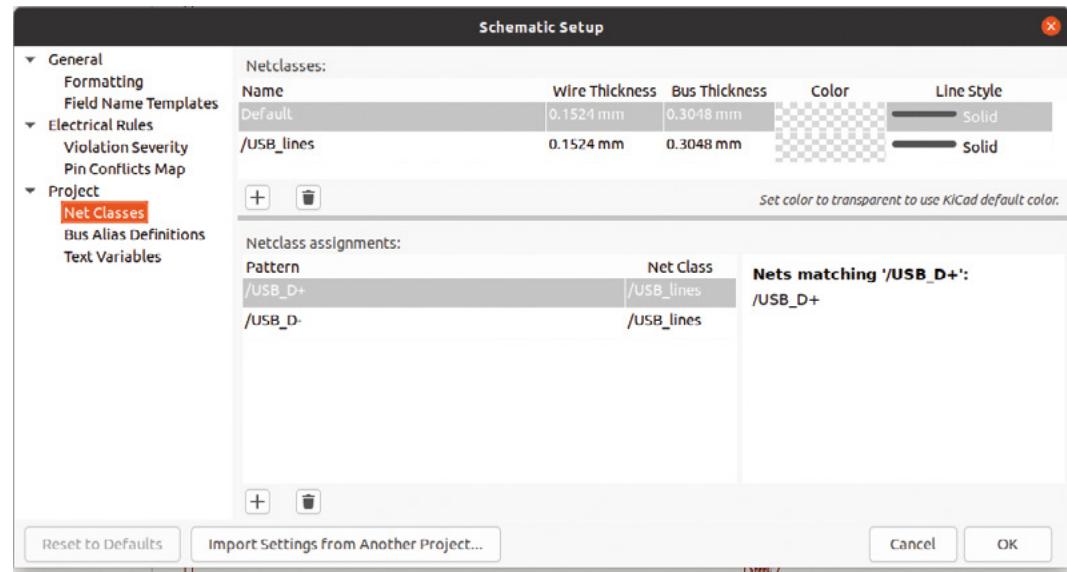
**Figure 3** The Hardware design with RP2040 documentation also includes a KiCad example project

# Designing an RP2040 board using KiCad

## SCHOOL OF MAKING

### QUICK TIP

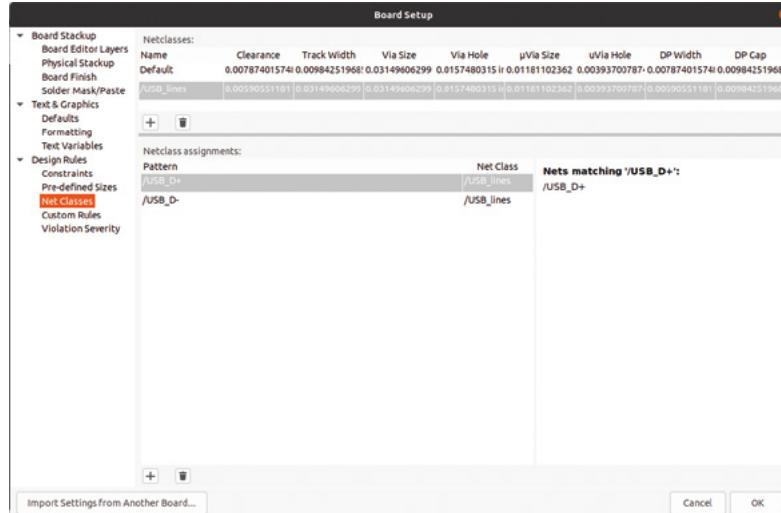
Do take care when creating custom footprints to ensure that the component pin number is compatible with your schematic symbol and vice versa.



**Figure 4** Setting up and assigning a net class

should be able to see the 'USB\_lines' net class that we added earlier. Select this, then close the dialog box. Repeat this for the USB\_D- label (**Figure 4**).

When you get to opening the PCB Editor, you can use the Board Setup tool (in the same positions as the Schematic Setup tool) to adjust the net class variables. This includes the track width, track clearance, via size, and more (**Figure 5**). We are predominantly interested in the track width and the track clearance to create the track lines that we need for the USB lines – this information on track geometry was taken directly from the Hardware Design with RP2040 documentation (page 10).

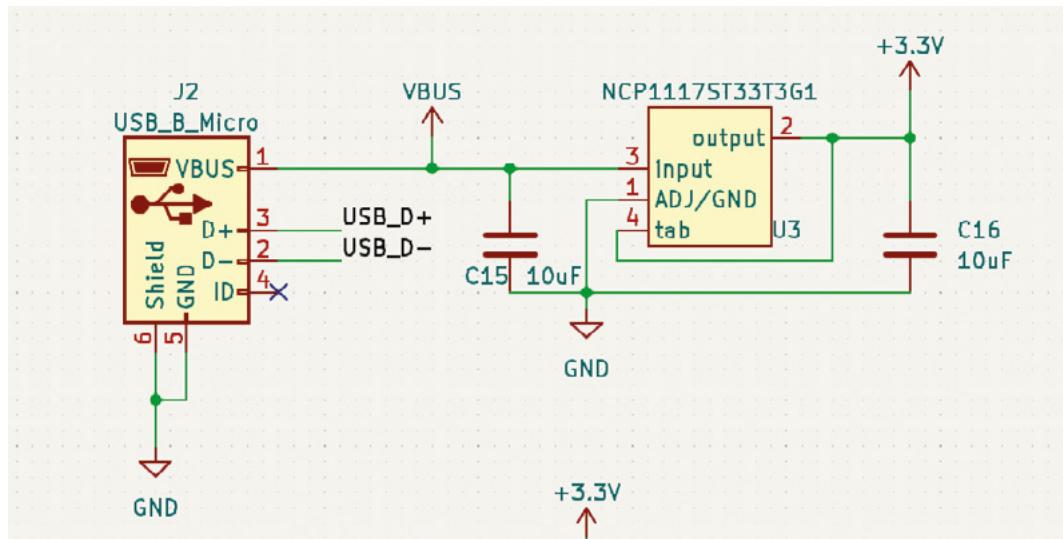


**Figure 5** Setting the net class variables for our USB lines in the Board Setup dialog in the PCB Editor

We laid out the power regulator and found a similar device to the minimal design example; however, the JLC component, C26537, had an extra pin and was in the SOT-223-3 package. We have a matching KiCad library footprint, but it was easier to just make a quick custom schematic symbol to ensure that the pin numbering was correct and matching (**Figure 6**).

One area that challenged us was, when looking up the components that the minimal design example project used for the crystal, we found no similar device available on the JLCPBCB component library. Using a consummate hardware hacker's approach, we researched what other open-source RP2040-based designs used, making sure to limit the list to projects we knew worked well. Having used Solder Party's Stamp in an earlier part of this series, we checked out its design. It looked simple and straightforward with just two 12pF capacitors, and on checking JLCPBCB, the crystal part was available as part number C521567. We again used the EasyEDA conversion website (see box, opposite) to create a custom footprint (**Figure 7**, overleaf).

Laying out the decoupling capacitors in the schematic is straightforward, but we've taken advantage of the KiCad text tool to add notes, occasionally acting as reminders for important information. In the Hardware design with RP2040 documentation, for example, it shows that the 1uF capacitors should be placed close to pins 44 and 45 on the RP2040, so it makes sense to add a schematic note as a reminder (**Figure 8**, overleaf).



**Figure 6** ◇  
Setting up the power regulator is simple once the target JLCPCB components are identified

The flash memory chip used in the minimal design example is available at JLCPCB and, as such, we went with the same design. In the Hardware design with RP2040 documentation, they have added a footprint for an optional pull-up resistor but found, with this chip, that it wasn't needed, so we omitted that part of the design. The important thing about the flash chip and the associated traces on the PCB is that it all sits over a continuous ground plane. This adds some head-scratching with the lay up of a two-layer board, but again, use our project or the Raspberry Pi project as an example of the routing.

As we aren't aiming for any particular use case with this example, we simply broke out all the pins to a pair of headers that we will place on each side →

**We simply broke out all the pins to a pair of headers that we will place on each side of the board**

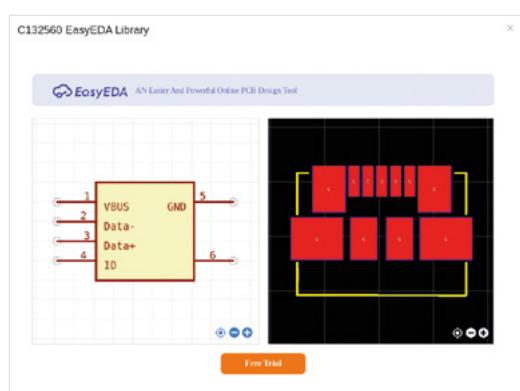
## EASILY CONVERTED

When designing for JLCPCB, even at the schematic level, you need to be thinking about what component and footprint you will be using. For our RP2040 board, we wanted to use a surface-mount micro USB-B socket where all the USB chassis and ground points are on the upper layer – this means we could keep all the assembly as SMD components. As such, apart from the pin headers/sockets, the units would be fully assembled by JLCPCB.

Looking through the LCSC component library, we opted for the C132560 component. JLCPCB often supplies schematic and footprint symbols for EasyEDA, and, in the case of the component footprint, we can use a handy online tool to convert it into a KiCad component footprint and add it to our libraries (we covered working with custom libraries and components in the third part of this series – [hsmag.cc/issue68](https://hsmag.cc/issue68)).

To make use of this converter, you'll need to set up an EasyEDA account ([easyeda.com](https://easyeda.com)). Click the link that says 'PCB Footprint or Symbol' on the component page on the JLCPCB library website, then click the Free Trial button underneath the pop-up window that shows the component symbol and footprint. After setting up a login, you should see a browser-based EasyEDA project with just the footprint of the component loaded. In the browser, go to File > Export, and then select EasyEDA ... from the list. This should then download a small JSON file. Navigate to [hsmag.cc/EasyEDA2KiCad](https://hsmag.cc/EasyEDA2KiCad), click the Load EasyEDA File button, and then upload the JSON file. The website will automatically convert the EasyEDA project and will then download a KiCad footprint file to your Downloads folder. As a side note, we also used this approach for the crystal package we used on the board design.

Sadly there is no conversion available for the schematic symbols; however, with some careful checking of pin numbers to ensure compatibility, we found a built-in KiCad USB schematic symbol that worked well with the footprint. Again, though, if you've worked through all parts of this series, you should be happy to create your own custom symbol.



### Above ◇

LCSC provide EasyEDA symbols for most of their components so you have to convert these before you can use them in KiCAD

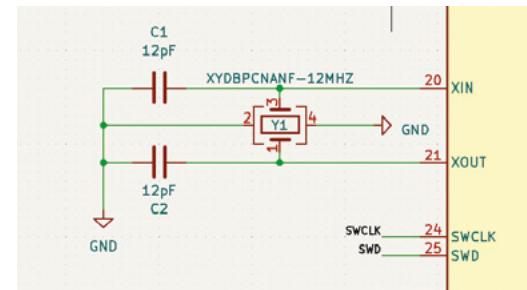
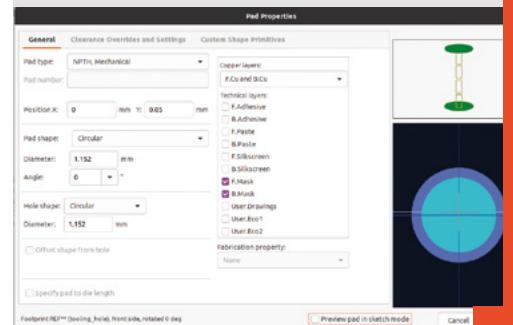
## SCHOOL OF MAKING

### TOOLING HOLES

When ordering assembled PCBs from JLCPCB services, you might need to consider tooling holes. These are small holes placed in your design layout that JLCPCB machines use to locate and hold the boards when they are being assembled. You can add these yourself, or you can omit them, knowing that JLCPCB engineers will place them for you. It's fair to say that JLCPCB engineers will place the holes sensibly and won't plonk one through a trace or in the middle of a component footprint; however, you might want to manually add them into your design so that you decide where they are finally placed.

The rules are that a minimum of two, preferably three, holes should be placed in the PCB design, and they should be placed at opposite corners – as far apart as they can practically be. The holes should be 1.152 mm diameter circular non-plated holes with a 0.148 mm solder mask expansion. The simplest thing we found to do for projects where we want to add the tooling holes is to create a custom component in our KiCad libraries that we can drag into and place in our design.

To make a tooling hole component footprint in the Footprint Editor, you need to create a new footprint and then add a single pad. Selecting the pad, click the **E** key to edit and enter the Pad Properties dialog. On the first 'General' tab, set the pad as 'NPTH', 'Mechanical' in the pad type – NPTH stands for 'non-plated through-hole'. Staying on the General tab, make sure the pad shape is set to circular and the diameter is 1.152 mm. Finally, click onto the second tab in the Pad Properties dialog called 'Clearance Overrides and Settings'. On this tab, set the solder mask expansion to 0.148 mm. Save this as a footprint and you can place them when needed into JLCPCB-oriented designs.



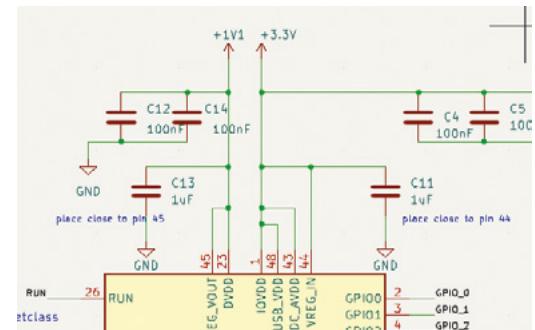
**Figure 7**

Due to JLCPCB not stocking the crystal that the RP2040 documentation recommends, we needed to rethink this part of the circuit

of the board. With the schematic largely complete, we moved over to the PCB Editor to begin the layout and routing.

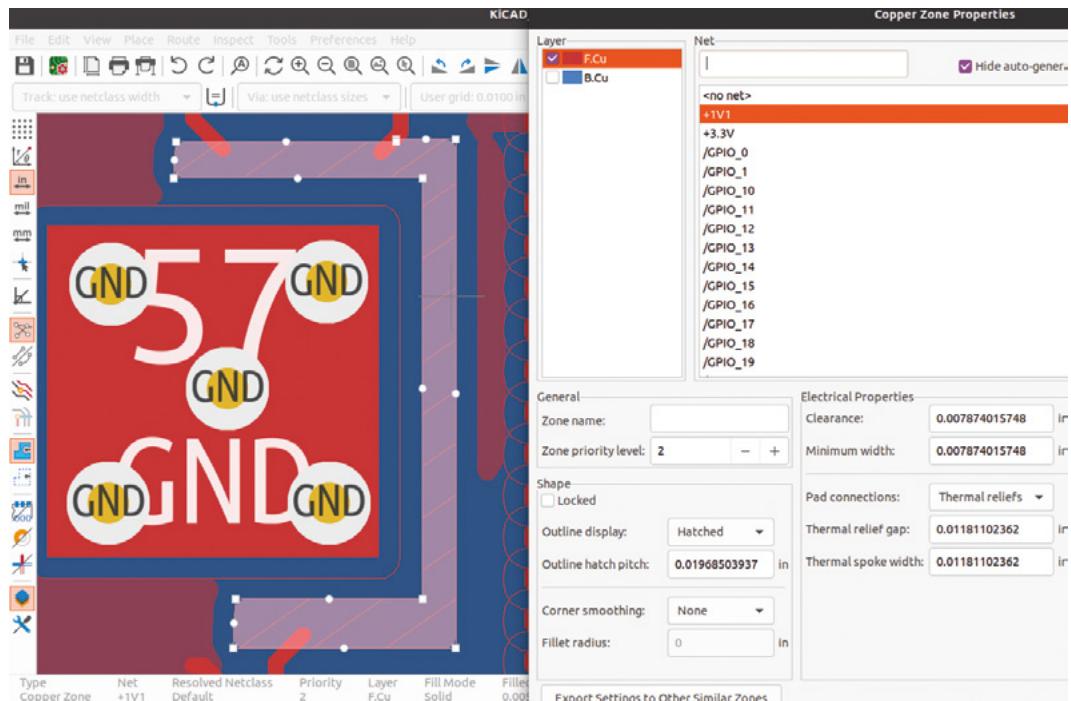
One interesting aspect of the board design, that we haven't looked at yet, is that it has numerous different voltages which each have their own copper flood zones. Making these is similar to how you make any other copper flood, as we have done in previous boards, but you need to set a priority value so that the different floods know how to flood separately. So on the top layer of our board there is a general 3V3 flood, a flood connected to the VBUS, which is the 5V input from the USB to the voltage regulator, and there is a small 1V1 zone inside the footprint of the RP2040. Notice in **Figure 9** that we have assigned the 1V1 flooded area priority 2, the VBUS area priority 1, and the general 3V3 priority 0. This essentially shows that they are separate areas.

Components-wise, we decided that for the resistors and the majority of the decoupling capacitors, we would go for 0402 packages. Again, we wouldn't make this choice if we were planning to assemble this board by hand, but with the assembly engineers and robots doing this fine work, we might as well use the tiny packages. It's less common for



**Figure 8**

Laying out the decoupling capacitors in the Schematic Editor is largely straightforward

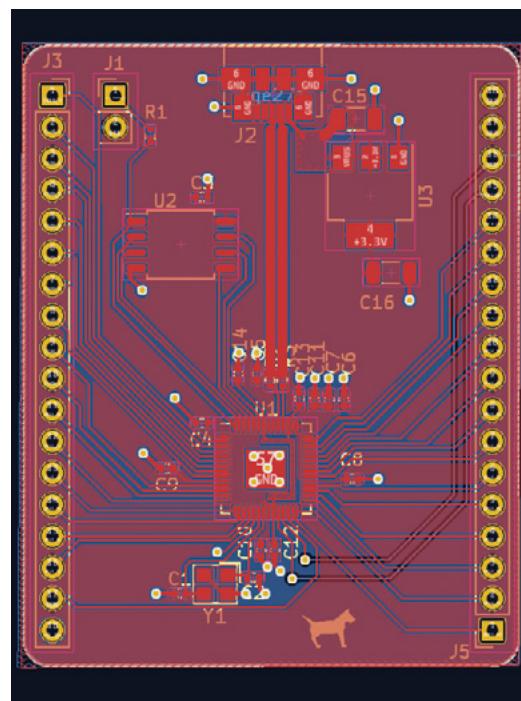


**Figure 9** Setting up different priorities to allow different zones to coexist

With the assembly engineers and robots doing this fine work, we might as well use the tiny packages

the very tiny packages of capacitor to be of accurate value as they increase in capacitance. So, although JLCPCB does offer components that claim 10uF in 0402 packages, we went with a more common larger 1206 variant.

We've largely covered everything specific to this project in this article. If you've worked through the previous parts of this series, you'll be familiar with all the other aspects of the process. It's definitely worth working through some smaller projects (like the small H-Bridge design in the last part of the series – [hsmag.cc/issue69](http://hsmag.cc/issue69)) to get used to the JLCPCB-specific processes. We'd also reiterate that reading Hardware design with RP2040 a couple of times before tinkering with an RP2040 board is time well spent. Finally, be warned, having fully assembled and hopefully functional boards delivered to your door is highly addictive! The KiCad files for this project can be found at [hsmag.cc/issue70](http://hsmag.cc/issue70). □



**Above** Our completed layout in the PCB Editor

# KiCad: schematic organisation and hierarchical sheets

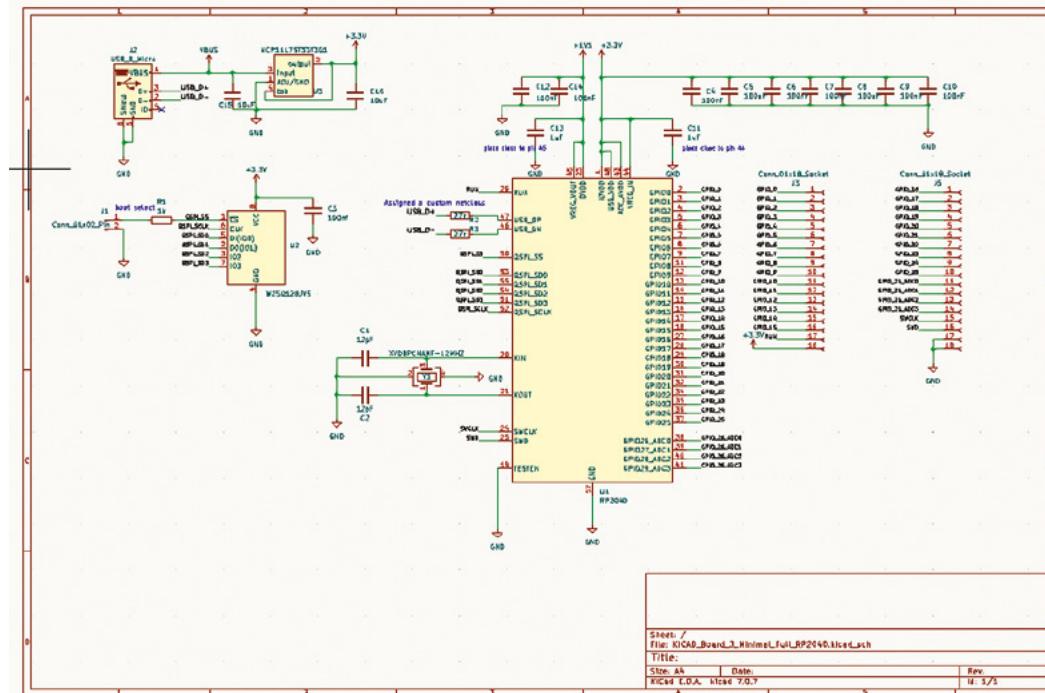
How to get your sheets in order before starting a larger project



**Jo Hinchliffe**

[@concreted0g](#)

Jo Hinchliffe is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high-power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and CNC kit!



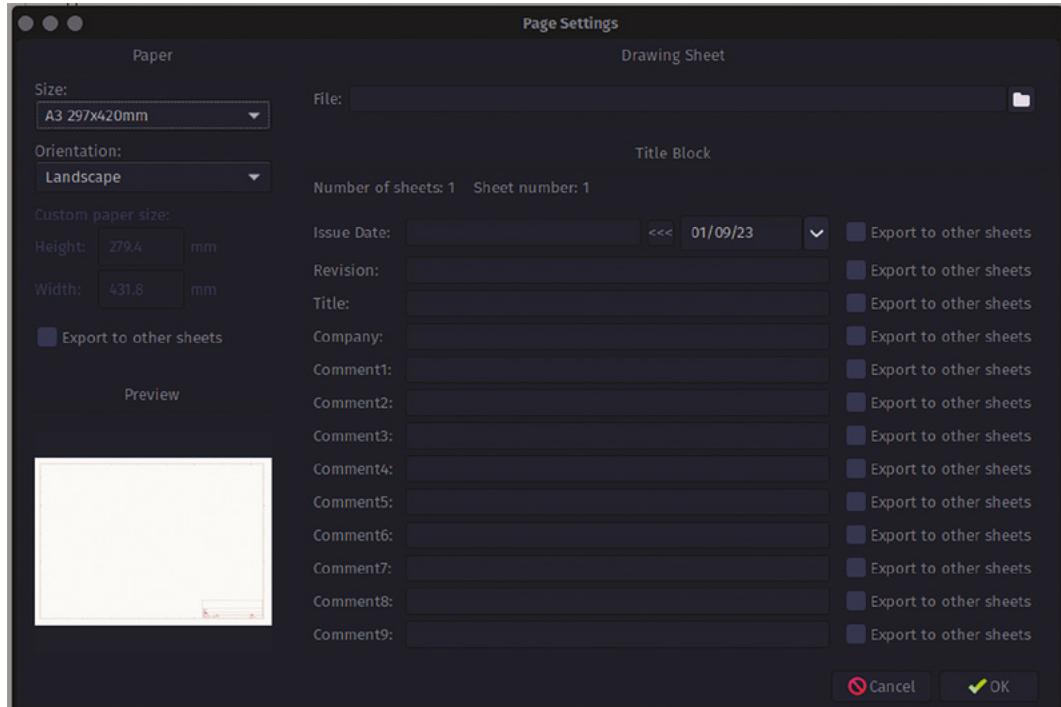
In the last issue, we laid out our largest project so far – an RP2040 minimal layout example and, in the next issue, we want to add to this design. However, the schematic is already quite full and, if we're not careful, the whole thing could become unmanageable. Let's take a look at how we can keep things clean, tidy, and easy to work with.

If you view last issue's project ([hsmag.cc/issue70](http://hsmag.cc/issue70)), and jump into the Schematic Editor, it looks quite cramped (**Figure 1**). One of the first things we can do is to simply increase the schematic size. Navigate to File > Page Settings and, in that dialog, you can

swap the page size, currently at A4. Changing this to A3 will give us plenty more room. Whilst we are in this Page Settings dialog, we can add some detail to the Title Block fields. This is the lower right-hand corner collection of text boxes that list the title, revision issue date, and more. Whilst we don't always practise what we preach, clear titles and revision labels and dates are really useful if you plan to publish your project or use the schematic as part of documentation (**Figure 2**).

With our schematic size increased and neatly titled, we can look at other ways of organising our schematic content. One approach that we have seen,

**Figure 1** Our schematic from the RP2040 minimal example project



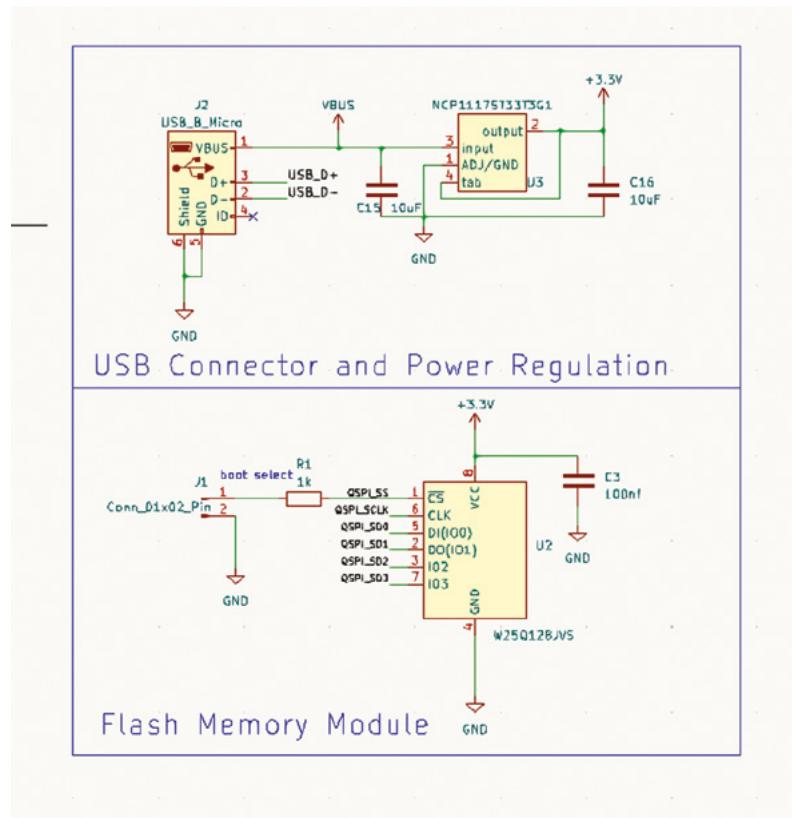
**Figure 2** Increasing the schematic page size and adding clear labelling is a good start to keeping organised

**Figure 3** Adding boxes and text labels can help organise sections of complex schematics

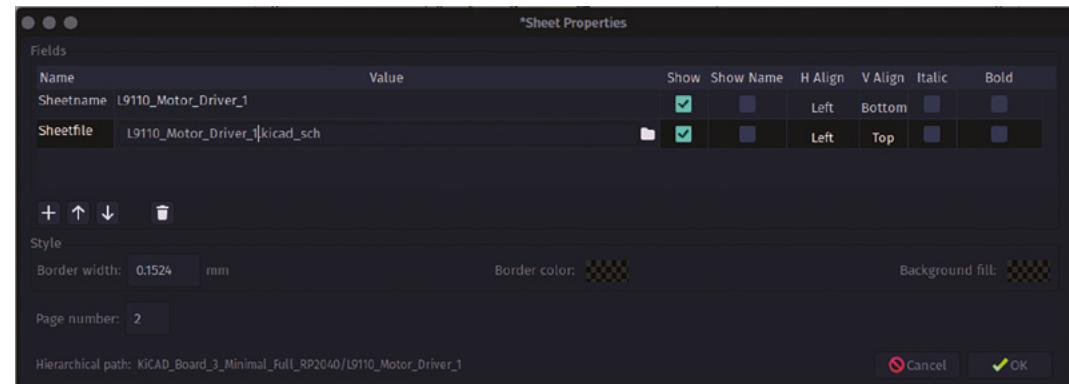
that some people like, is to create graphic boxes around different subsections of a project's circuits. For this we can use the 'Add a rectangle' tool, again in the lower right-hand side of the screen. You might need to use the selection tool to select and move parts of the schematic into a position where you can completely draw a rectangle around them.

**Clear titles and revision labels and dates are really useful if you plan to publish your project**

To draw a rectangle using the 'Add a rectangle' tool, you do a single left-click in the start position (a corner of your rectangle) and then move the pointer to the opposite corner position you require. Once the rectangle is drawn, clicking on it with the selection tool, you can either grab the anchor points to change its dimensions, or press **M** on the keyboard to move the rectangle as you would any other object in the Schematic Editor. With a section of your schematic now encapsulated in a rectangle, we can simply use the 'Add text' tool to create and then to place →



## SCHOOL OF MAKING



**Figure 4**

Adding a name for a new hierarchical sheet and a file name for the new hierarchical sheet schematic file

labels into our boxes (**Figure 3**). We did this in the original RP2040 minimal example project, in the last section of this series, when we added notes to place capacitors close to certain points etc. What we didn't make note of is that, new to KiCad 7, we can use any font for our text, allowing us to move away from the default KiCad fonts and bring a little of our own house style to our schematics.

### ROVING ROBOT

We're planning to use last month's RP2040 example project as a base for a larger project. We're aiming to make a simple robot rover design with the main chassis of the robot being a PCB with all the essential components mounted on that single board. We want to keep our original RP2040 board project, so we

need to make a copy of our project to develop into our robot rover platform. This is one way we can almost use KiCad projects in a modular fashion.

To copy a project in the original project go to File > Save As, then select or create a new folder to save the project to. Insert a new name for the new project copy in the Name dialog box, then click Save. Although not completely necessary, it's not a bad idea to then open the new project folder in a file browser and delete any unneeded files for the new project. For example, if you have a Gerber file folder, these probably won't be relevant to the new project. Any backup files can also be removed as you would probably return to the original source project, if required, rather than use a backup from this new branched project.

The basic premise for our robot is that it is going to have four wheels, all of which are driven by N20-style motors. This gives us options down the line in that it can run with normal style wheels and tyres, but we are also interested in mecanum wheels, which need to be driven individually to create the interesting sideways and diagonal motions mecanum wheels are known for. This means that a primary part of the design is to add four motor driver circuits to our RP2040 board, one for each wheel. Again, considering

### QUICK TIP

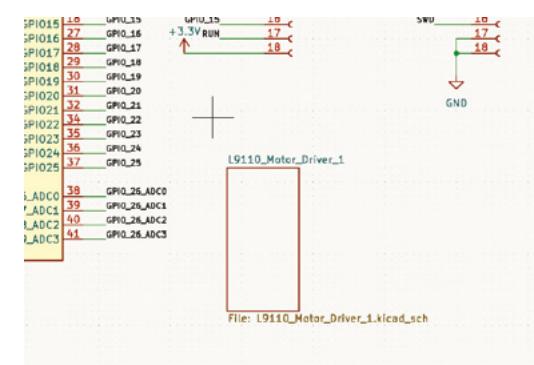
If you can't select a section of schematic completely using the selection box tool, select as much as you can, then press and hold the **CTRL** key whilst clicking any missed components or wires.

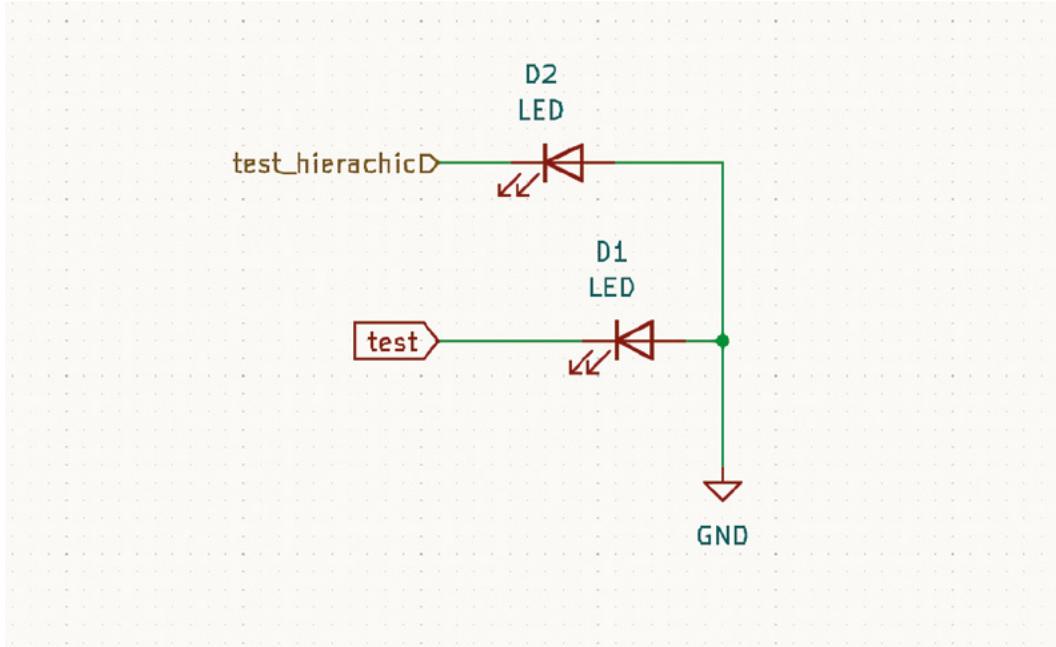
### A SPLASH OF COLOUR

You can add images to schematics and, whilst you can place images anywhere on any page, a common use case for this is to add a company logo to the Title Block area. The Schematic Editor supports a wide range of image formats, including PNG, JPG, TIFF, BMP, and many more. Usefully, on import you can rescale the imported image. This is useful as you can import a larger, higher DPI image and then scale it down to fit the box. In the image, the dog logo has been created in Inkscape and exported as a PNG file with a transparent canvas. The image as exported is sized at around 700×500 pixels and has been exported at 300 dpi. This means that the image is good enough quality when scaled down to fit into the title block, but is still a comparably small file size at around 16kB.

To add your image to the schematic, simply click the 'Add a bitmap' tool icon on the lower right-hand side of the screen, and navigate to your chosen image file. Once the object is imported, you can move it and scale it whilst maintaining its aspect ratio by dragging the corners of the image bounding box.

**Right**  
Our empty hierarchical sheet placed in the original schematic





schematic clarity, we could just place the four motor driver circuits into our A3 schematic and wire them using either labels or direct wiring. However, another way we can add content to the schematic is by using hierarchical sheets. A hierarchical sheet can be thought of as a sub-sheet that exists in the schematic below the main page, into which we can insert designs which can be connected to the main top-layer schematic page.

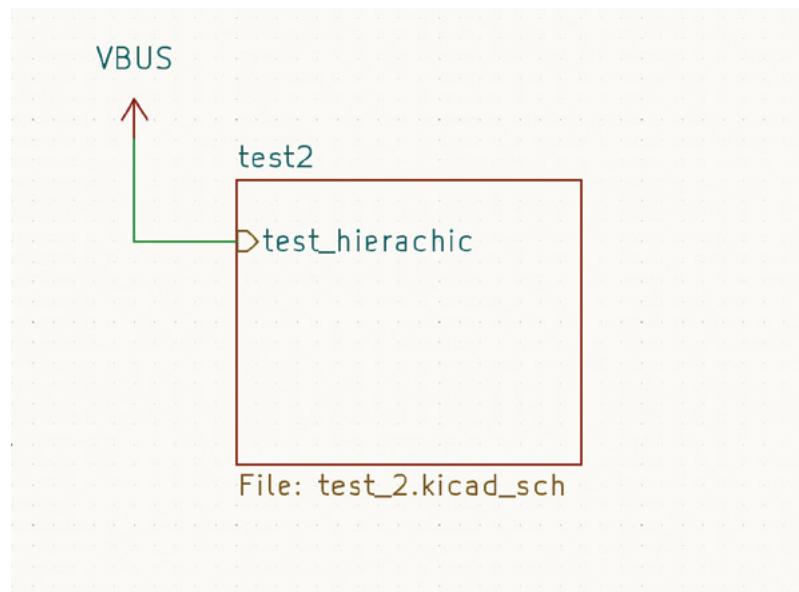
To create a hierarchical sheet, you can use the 'Add a hierarchical sheet' tool icon found in the lower right-hand area of the Schematic Editor, or you can press the **S** key on your keyboard. Either way, you then click and drag to create a rectangle in the schematic. When you left-click to finish the rectangle, a dialog window will launch called 'Sheet properties'. Into this you can put a sheet name, which will be displayed in the rectangle on the main schematic page. We'll call our first sheet 'L9110\_Motor\_Driver\_1'. Under this is an input box for a sheet file name. It will currently be populated with 'untitled.kicad\_sch'. This is the name of a separate file that will be created for this hierarchical sheet. Again, we changed this to something appropriate, such as L9110\_Motor\_Driver\_1.kicad\_sch. Notice that there are checkboxes labelled 'show' and, by default, they will be ticked (**Figure 4**). This means that, in the main schematic sheet, our hierarchical sheet rectangle will appear with both of these pieces of information listed. It is advisable to show one, or →

## Another way we can add content to the schematic

is by using  
hierarchical sheets

**Figure 5** Inside a hierarchical sheet with a design using a hierarchical label and a global label

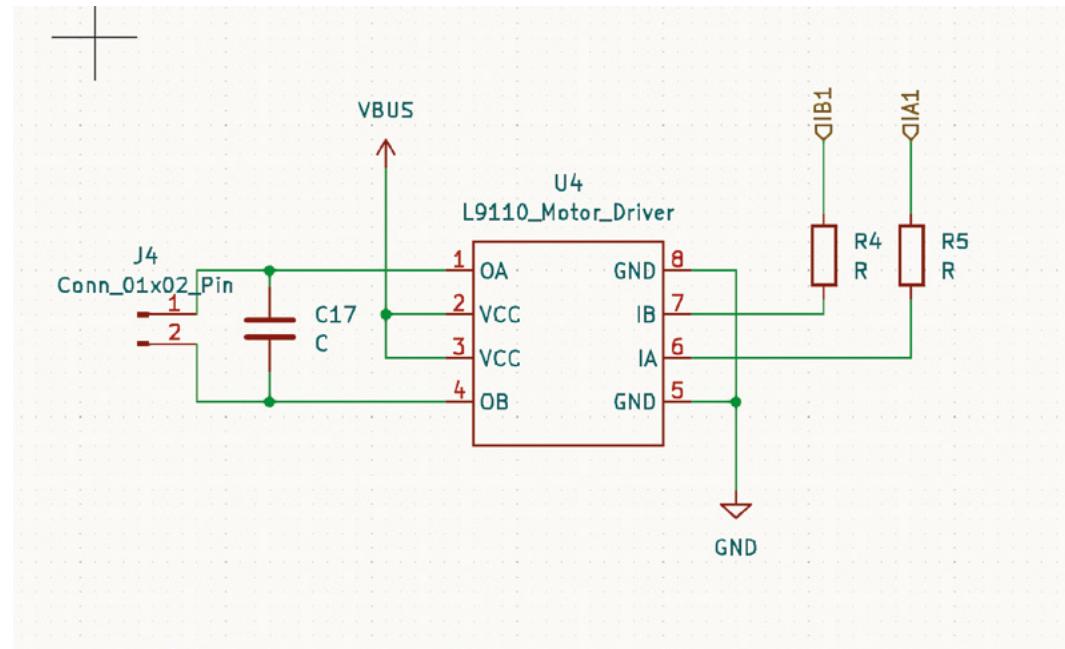
**Figure 6** The test hierarchical sheet object viewed from the main schematic. The hierarchical pin has been connected to VBUS



## SCHOOL OF MAKING

### QUICK TIP

As a hierarchical sheet is a separate schematic file, you can use 'File-page settings' to set the title, page size, and other details, as we did earlier.



both, of these pieces of information or else you have to navigate into the hierarchical sheet to see what it contains and it can be confusing if you have multiple hierarchical sheets. On a similar theme, notice that you can also play with the appearance of the hierarchical sheet rectangle, increasing or decreasing line width, background, and border colours. Whilst this might seem frivolous, if you end up working in a design with a lot of hierarchical sheets, being able to differentiate them by colour scheme can be an aid to productive working.

To enter your new hierarchical sheet from the main schematic page, you can either right-click on the rectangle and select 'Enter sheet', or you can double-

click on the hierarchical sheet rectangle. You should be met with a brand new empty schematic.

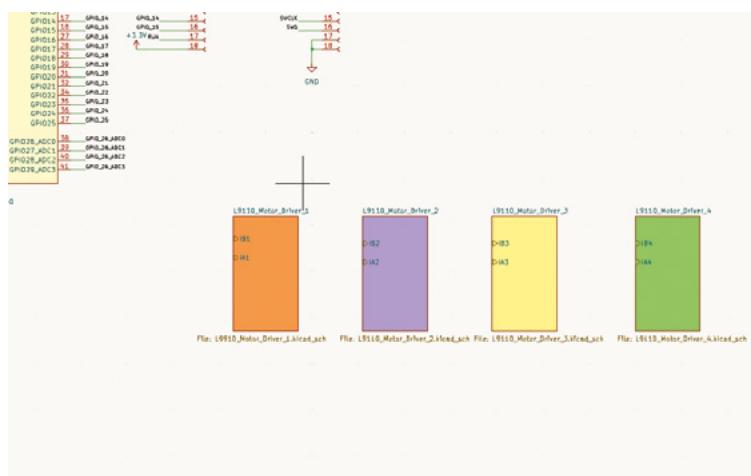
You can now begin to place components and create your circuit in the hierarchical sheet. The first thing to note is that anything connected to a global label or net will automatically be connected to those points globally. So if, for example, you add a component and connect it to a ground 'GND' symbol and to 'VCC', they will automatically be connected to those points in the top-layer schematic and will appear as GND and VCC connected when you import those components and connectivity to the PCB Editor. If you did place and connect such a component in a hierarchical sheet, when you return to the main sheet you won't see any connections coming out of the hierarchical sheet rectangle. For general ground and power connections this might be OK, but it may get confusing if you use this approach for all your connectivity. A common way of making connections into and out of hierarchical sheets is to use hierarchical labels.

### HIERARCHICAL LABELS

To place a hierarchical label, you can either click the 'Add a hierarchical label' tool icon or press **H** on your keyboard. A Hierarchical Label Properties dialog will appear. Insert a name into the Label field. You will now have a label you can place, move, and rotate and connect into your design. In **Figure 5**, we have made the hierarchical label 'test\_hierarchic' and connected it to the cathode end of an LED component symbol. We've also placed a global label test to another

**Above** The L9110 circuit for each motor. Note the hierarchical labels used on the IB and IA pins

**Below** Each of our L9110 driver circuits sits in its own hierarchical sheet, and the individual motor driver pins are broken out, ready to be wired into the RP2040



LED component. Moving out of the hierarchical sheet is simple: you can either right-click and select ‘Leave sheet’ or you can hold **ALT** and then tap the **BACKSPACE** key. Even after creating a hierarchical label inside a hierarchical sheet, you won’t see that connection until you right-click over the hierarchical sheet rectangle and click ‘Import Sheet Pin’ from the drop-down menu. You should now see any hierarchical pins appear aligned with the edge of the rectangle. You can move these pins to any position around the rectangle, and you can connect and wire to these labels as you would any other symbol. In **Figure 6** you can see the ‘test\_hierarchic’ label in the sheet rectangle, and we have wired it to VBUS. Note that you can’t see the global label ‘test’, but that point inside the hierarchical sheet will be connected to any other connections with that global label anywhere in the project schematic. Note again that inside this hierarchical sheet, there is a connection to a GND symbol and, as such, that point is connected to the project’s global GND label.

For our robot rover design, we are going to keep it simple and affordable and will use the L9110 motor driver IC as it’s adequate for the N20 motors,

**“One of the great benefits of using hierarchical sheets is that we can copy them”**

affordable, and there is a large amount of stock available on JLCPCB. The circuit around the L9110 is pretty straightforward, with a couple of pull-up resistors and a decoupling capacitor across the motor outputs. Creating a hierarchical sheet for the motor driver circuit, we’ve used hierarchical labels to create the two signal inputs. We could have opted to have the motor outputs as hierarchical labels, but it was cleaner to add the motor output connector symbols inside the hierarchical sheet, avoiding more clutter on the main schematic.

One of the great benefits of using hierarchical sheets is that we can copy them, or copy their contents, quickly to create multiples of similar modules, either within the same project or into other projects. In our robot rover project, we have simply added three more hierarchical sheets and named them sequentially as Motor\_Channel\_1, Motor\_Channel\_2, Motor\_Channel\_3, and Motor\_Channel\_4.

We then copied and pasted the contents of Motor\_Channel\_1 into each different hierarchical sheet. Whilst not wholly necessary, it seemed good practice to avoid confusion. So, each time we copied the L9110 circuit, we relabelled the hierarchical pins so that each motor channel was unique. We can then use the ‘Import Sheet Pin’ function that we used earlier on each of the motor channel hierarchical sheets, and we are ready to wire the pins to our chosen GPIO pins on the RP2040 – either directly or using labels to again keep the main schematic clean and tidy.

Hopefully you have found this guide useful. It’s great to think about your practice with KiCad and how you can be organised in a way that suits you. A final good idea is to look through some open-source hardware projects that have used KiCad. You’ll certainly find lots of different approaches to organisation and project management. □

## LARGE PROJECTS

One thing about KiCad is that you can use many different approaches, depending on your needs or your particular way of thinking. One such approach we’ve seen out there in KiCad-land is to use a flat hierarchy for your schematic layout. This essentially turns the first schematic sheet into a kind of holding sheet where all the hierarchical sheets are held. You can then, in each sheet, use general and global labels so you don’t need to draw any connectivity on the main topmost schematic. Although this might momentarily confuse anyone else opening your project, it’s actually a brilliant way to organise a particularly large project into specific sections. Of course, as each hierarchical sheet is an individual schematic file, you can simply work on each file individually as needed in development. One tip that we noticed for this approach, or for working with multiple hierarchical sheets, is that if you copy and paste a hierarchical sheet rectangle in the main schematic, it will automatically add and increment a number to the name of the sheet. So ‘sheet’ would become ‘sheet1’, then ‘sheet2’. We even noticed that if you created a first sheet called ‘Something\_1’ and copied and pasted it, the clone would increment to ‘Something\_2’.



# KiCad, mechanical accuracy, and silkscreen features

In this part of the ongoing KiCad series, let's look at some techniques to increase accuracy when aiming to create a PCB to be used as a mechanical structure



**Jo Hinchliffe**

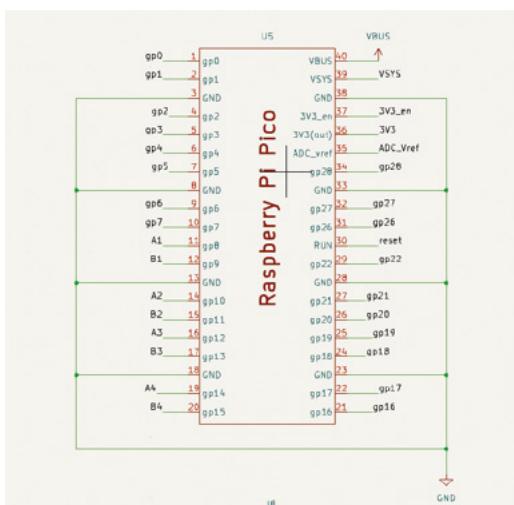
Jo Hinchliffe is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high-power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and CNC kit!

**t's increasingly common for projects to incorporate PCBs as a mechanical part of the mechanism.** In our last section, we looked at hierarchical sheets and laid out a motor driving circuit that we could copy and paste to add motor drivers to a project. In this part, we are going to create a simple robot rover that we're calling 'stoRPer'. StoRPer is a tongue-in-cheek reference to a favourite childhood toy from the 1980s: the 'Stomper'. The Stomper, by toy company Schaper, was the first ever four-wheel drive electric toy car. Despite no form of remote control, they were great fun to try and build obstacle courses for, or to test on steep gradients. We wanted the reasonable torque and the four-wheel drive aspects

of the Stomper but with the addition of a Raspberry Pi Pico to make it a more interesting and controllable platform – so, 'stoRPer' it is. It's designed with all-wheel drive (AWD) so that Mecanum drive systems can be built and experimented with.

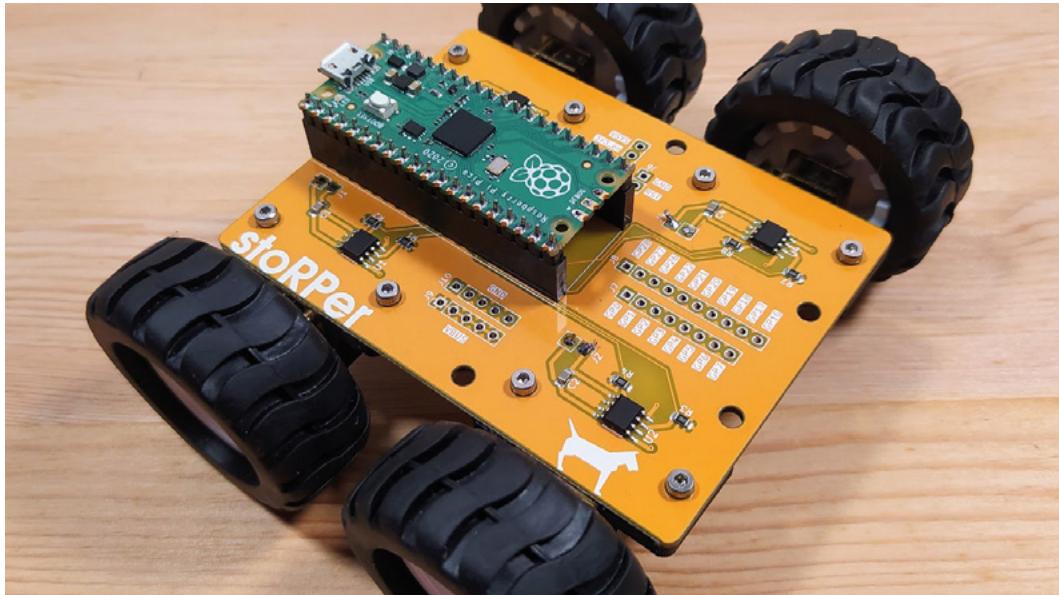
We are going to use Pico as a module on this build and focus on some aspects of particular importance when we are using a PCB as a mechanical part as well as for electronic purposes. The idea for the project is that the PCB will form the chassis of the stoRPer, with the motors being clamped to the PCB chassis using some 3D-printed parts. Therefore, we need to be capable of placing components and general PCB geometry accurately in order for everything to fit together. We'll also look at how we can check our PCB and 3D-printed models will fit together before we print or send the PCB for fabrication.

One of the first jobs is to create a Pico symbol component in the Symbol Editor. We covered creating symbols in the earlier sections of this series, so we won't recap that process too much. We decided not to include the Pico's three debug pins on either the schematic symbol or the PCB footprint. This was partly because, across the different Pico models, they are physically in different positions on the board and also, as we intend to have a Pico mounted onto this project, we can still interact/wire to the debug pins if needed. As such, we laid out a simple 40-pin component in the Symbol Editor and brought it into our Schematic Editor. After quickly connecting all the ground points, we set about connecting four hierarchical sheets, each with an L9110S motor driver



**Right ♦**

A custom Pico symbol has been created, with the majority of the pins broken out



**Left** ◇  
The stoRPer robot prototype using the PCB as its main chassis component

**Figure 1** ◇  
The layout of the L9110S motor driver circuit cloned into four hierarchical sheets

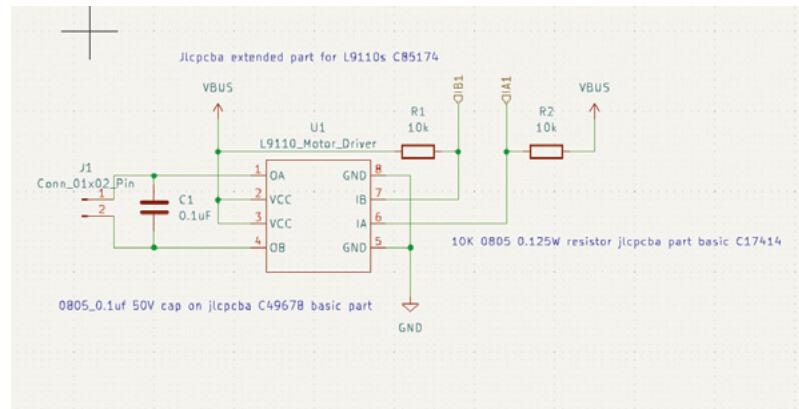
IC-based circuit inside. We covered working with hierarchical sheets in the last section of this series, but you can see the circuit layout in **Figure 1**. Each of the four motor drivers has its own sheet and has two pins broken out. We've connected these sets of pins to the Pico symbol using labels A1, B1, A2, B2, etc. The rest of the Pico's pins are broken out and connected to some multi-pin connectors, ready to be broken out on the PCB.

For the stoRPer project, we've decided to mount the Pico using the through-hole header pads on the Pico rather than the castellated edge connectors.

“  
The rest of the Pico's pins are broken out and connected to some multi-pin connectors  
”

This means that we won't be mounting the Pico flush to the project, but it does mean that the Pico footprint is thinner. We can also choose to use header sockets or not to allow the Pico to be permanently or temporarily mounted to the PCB.

The header pin pads on the Pico lie in a 2.54 mm pitch grid, with the 20 pins on either side being separated by 7\*2.54 mm. This makes them easy to lay out – simply add pads on a 2.54 mm grid in the Footprint Editor (**Figure 2**). We also want to be able to place a rectangle on the silkscreen layer that accurately shows the position of the board.

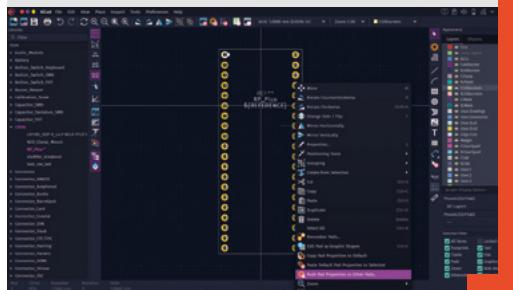


Consulting the Pico documentation, we can find a technical drawing and see that the outer edge of the Pico is 51 mm x 21 mm. We also need to consider the position of this rectangle relative to the pads that we have just created. We can see in the technical drawing, for example, that relative to the centre of the upper left-hand pin (pin 1), the upper-left corner of the Pico is 1.37 mm higher in the Y axis and 1.61 mm over to the left in the X axis. To use this information, we can go back into the Footprint Editor and place our pointer on the grid point in the centre of the pad we placed for pin 1. If we then press the **SPACE** bar, we will set the local origin of the page to be 0,0 at this point. We can check this by looking at the bottom of the screen as we move our pointer, the distance should increase relative to this point. We can then set a user grid to 1 mm spacing and use this grid to draw our 51 mm x 21 mm rectangle. →

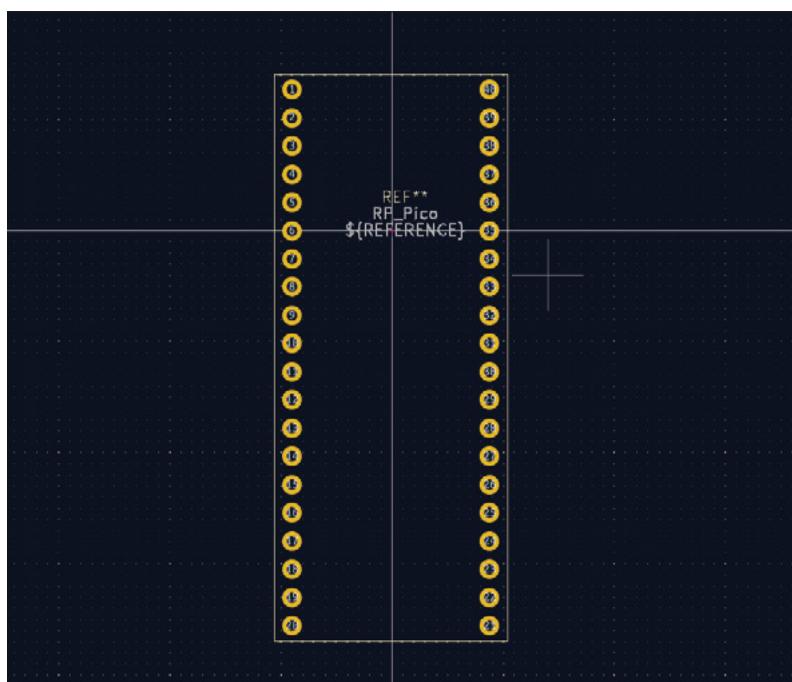
## TUTORIAL

### LOTS OF HOLES

When creating footprints with lots of through-hole pads, KiCad makes it pretty simple: you click to add a pad and then the tool indexes to the next numerical pad for you to place. If you've placed and positioned a lot of pads though, it can be annoying to realise that you need to change an aspect of the pad's properties for all of them. KiCad has you covered, though. As an example, when we made the footprint for a Raspberry Pi Pico and decided that after laying out 40 standard through-hole pads, we wanted to increase the internal hole diameter and the overall outer diameter to increase the size. The Footprint Editor conveniently recognises that this is a common situation and, as such, you can simply change one pad to your desired pad properties and then, with your adjusted single pad highlighted, you can right-click and select 'Push Pad Properties to Other Pads...' to make all compatible pads change to the new characteristics.



**Figure 2** Creating the simple yet accurate Pico footprint

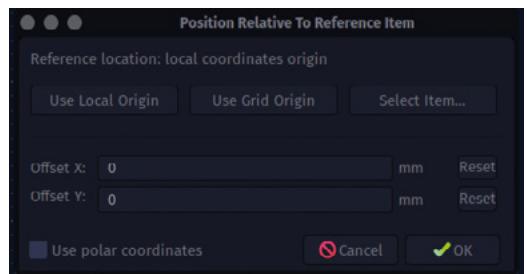


If we then select the rectangle, we can right-click and scroll in the drop-down menu to Positioning Tools > Position Relative To.... Selecting this, we will see a dialog box – in the dialog box, click to select 'Use Local Origin' and then adjust the 'Offset X' and 'Offset Y' by the amounts we derived from the technical drawing (**Figure 3**). Note that, by default, the origin corner of the rectangle is the top left-hand corner. Using this method, you should be able to place items with incredible accuracy.

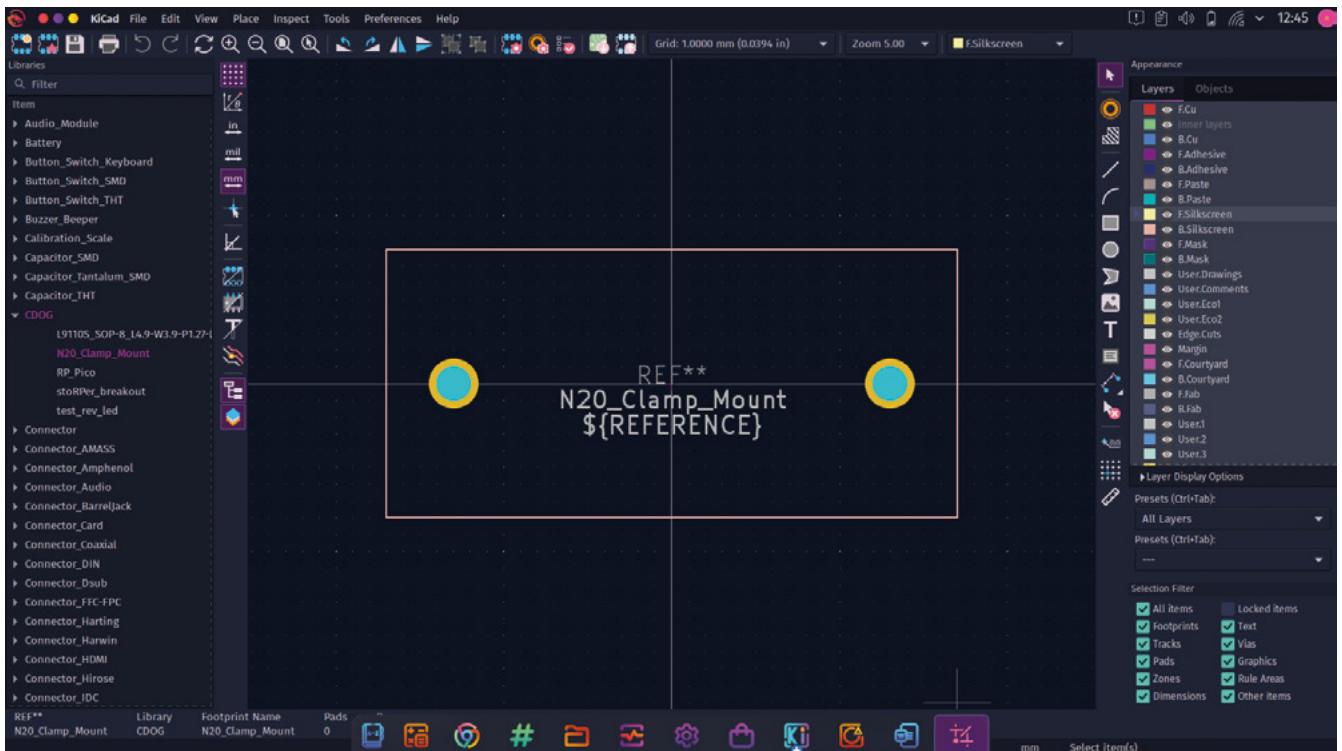
One thing of note is that despite our stoRPer robot design being relatively simple mechanically – a rectangular PCB – we do want to be able to place footprints accurately within the edge cut area. When designing this and other footprints, it's worth considering where your origin point is in the Footprint Editor and placing the device in a

We want to 3D-print some brackets to clamp the motors into position

known position relating to it. We opted to place the Pico footprint so that the upper left-hand corner of the silkscreen box depicting the edge of the Pico was the origin point on a 1 mm grid spacing. This meant that later, when we placed a rectangle in the PCB Editor that represented the edge of the PCB, we could place it in a position such that the Pico is dead centre, with the larger box also placed on a 1 mm grid coordinate. After playing with a few test boxes in KiCad, we decided our rectangular chassis dimensions would be 64 mm × 86 mm. We used Inkscape to draw our rectangle as we could then easily add a 2 mm radius to each corner



**Figure 3** Using the 'Position Relative To...' positioning tool to accurately place objects in the Footprint Editor



of the rectangle. We've again covered importing graphics in an earlier section of this series, but we easily imported the rectangle we drew as an SVG in Inkscape into our edge cuts layer using the File > Import > Graphics function.

With the Pico placed and the PCB edge defined, we need to consider the physical mounts for the motors. We want to use the excellent and available N20-style geared motors, mounting one for each of the four motor driver circuits. We want to 3D-print some brackets to clamp the motors into position, so we need to leave some space for the 3D print material around the motor, and we need to take this into account when creating a footprint for the motor mount. After some consideration, we created a custom footprint which consisted of two non-plated through-hole (NPTH) mechanical pads placed in-line. These were placed at a distance between centres of 26mm, placed on a 1 mm grid spacing. To place an NPTH mechanical hole, you place a regular pad and then press **E** to change the pad type in the Pad Properties dialog. We set each NPTH hole to 2.1 mm internal diameter to create clearance for a small M2 bolt. To finish the footprint for the N20 motor mount clamps, we added a silkscreen rectangle set to the dimensions of the base of our 3D-printable mount design (**Figure 4**). Note that this is the first time in the series that we have placed extra components which aren't connected to anything or included in the schematic in the PCB Editor. To do this, we click



the 'Add a footprint' tool icon and select a footprint in a similar manner to how we would place a symbol in a schematic.

Adding and removing text-based elements to a silkscreen layer is reasonably straightforward in KiCad 7. On more technical PCBs, as opposed to artistic PCBs, we often lay out our PCB design with little regard for the silkscreen and then sort the silkscreen layer out later in the development. Often, one of the first tasks is to remove unwanted elements on the silkscreen that have been automatically placed by the use of default library footprints. We can select the correct silkscreen layer, often the front silkscreen 'F.Silkscreen', and for items such as footprint reference annotation, we can simply left-click to select them, then move them or press the **DELETE** key to remove the item. It's common for this reference to not be placed →

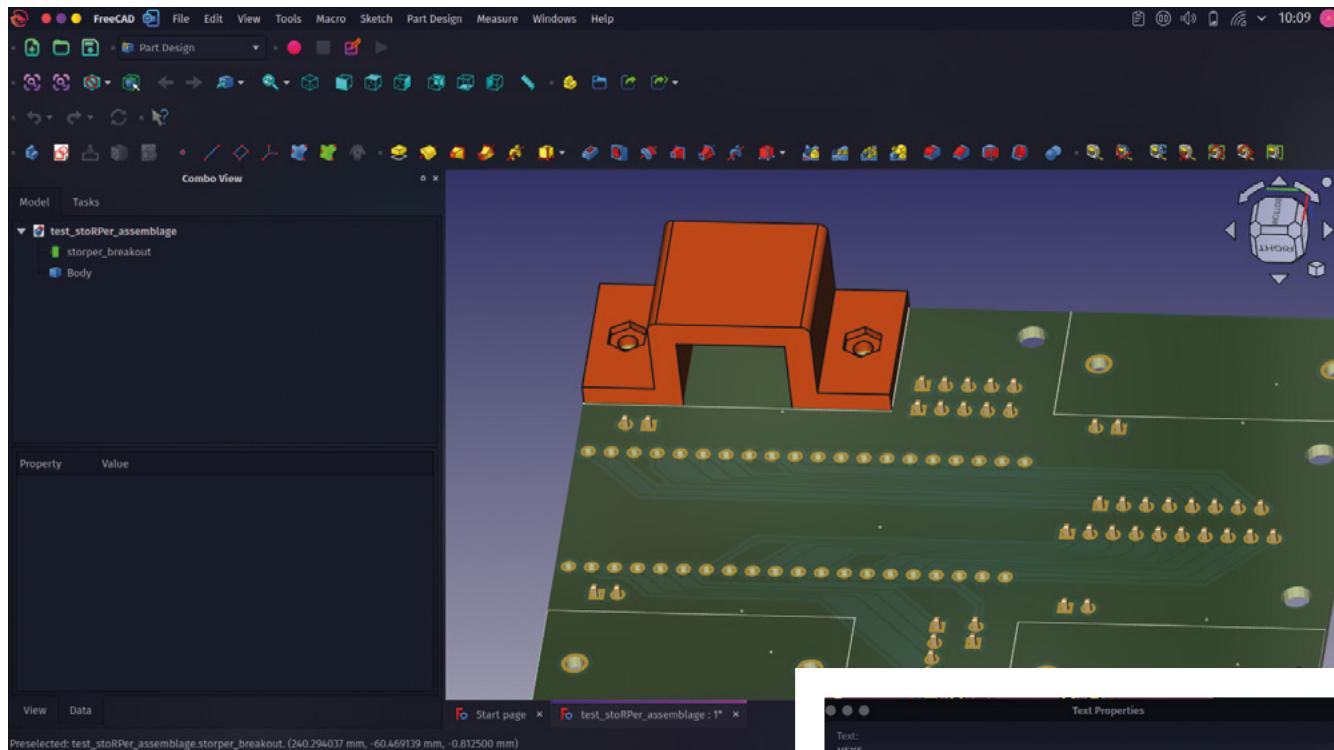
**Figure 4** ◊  
The mechanical footprint that will mount the N20 motor and clamp

**Figure 5** ◊  
Editing a footprint with the component selected and opened in the Footprint Editor from the PCB Editor gives the option of only editing that individual instance of the footprint

## QUICK TIP

As the stoPer design evolved, we used simple rectangular boxes drawn in KiCad on either the F.Silkscreen or the User.Comments layer as guides and visual aids.

## TUTORIAL

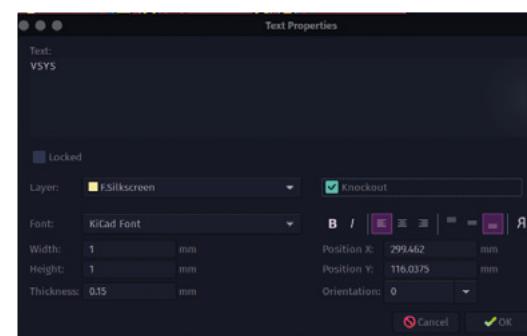


### Above

The combination of KiCAD and FreeCAD make a great open source toolchain

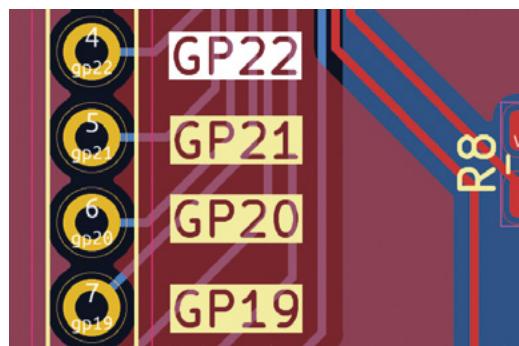
## FREE BOOK

In the free-to-download book *FreeCAD for Makers* from Raspberry Pi Press, we looked extensively at the use of the KiCad StepUp workbench which enables and simplifies importing KiCad projects as 3D objects into FreeCAD as well as the creation of 3D components for inclusion into KiCad's 3D PCB viewer. It's an incredibly powerful suite of tools and is definitely worth exploring. For this project, however, we just wanted to check if the motor clamp we had created in FreeCAD would fit our PCB chassis. You can use File > Export and select the 'STEP...' option to export a STEP file which can be imported into FreeCAD; however, this will lack the details of the copper layers and silkscreen which you might need to see to check if mechanical components cover aspects of your PCB design. One simple approach that solves this is to export a WRL file. WRL files are file types often used by assets destined for use in virtual reality, but they have the advantage in KiCad that a WRL export contains all the visual details of your PCB. We used File > Export > 'VRML...' to export a WRL file, and then we used File > Import in a new document in FreeCAD to import the file. We'd made a simple N20 clamp component which had 2mm radius corners on two corners matching our PCB and N20 motor clamp footprint. While we could have used an Assembly workbench, such as A2plus in FreeCAD, to constrain the clamp in position, for a simple check, we can move the part into alignment to visually check how it looks.



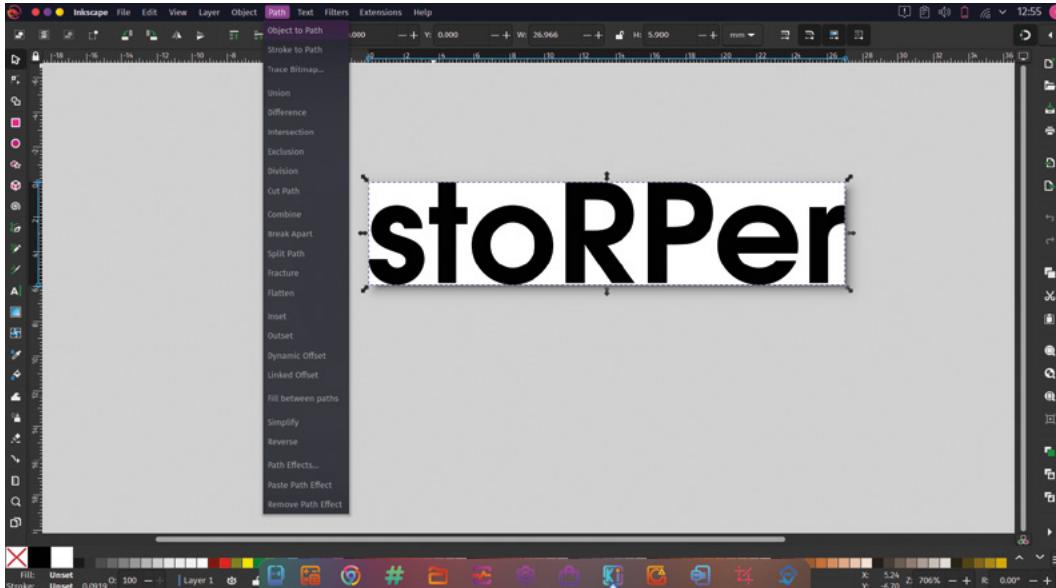
**Figure 6**

The Text Properties dialog where you can set text features, including the new 'Knockout' feature



**Figure 7**

A new feature in KiCad 7 is the ability to add knockout text items, where the text is subtracted from a small block on the silkscreen layer



**Figure 8** ◇  
Converting a text object to a path in Inkscape ready for import into KiCad

optimally and may sit under or across other parts and components. The annotated reference is formed from both the automatic annotation of the schematic during the footprint association process and the type of component it is, so R for resistor, C for capacitor, J for connector, U for IC, etc. As they replace the placeholder Ref\* designator, they are independent of the main footprint design and, as such, can be removed with ease. If, when tidying the PCB design, you want to move a part of the silkscreen design of a footprint, you will need to edit that in the Footprint Editor. KiCad makes it easy to edit the footprint and apply the changes just to the individual footprint within this project rather than pushing the changes to the global footprint library. With a target footprint selected in your PCB, press **CONTROL** and **E** to open the footprint in the Footprint Editor. You should see the footprint in the editor with a message in the upper left-hand corner of the window that reads 'Editing J4 from board. Saving will update the board only', where 'J4' will be the reference of whatever footprint you have opened (**Figure 5**). You can now make any changes to the footprint that you require, including deletion or changes to the graphical silkscreen elements.

Of course, often we want to add text-based components to our board designs. Again, KiCad makes this pretty straightforward. We can simply click the 'Add a text item' tool icon and then left-click in the PCB design. The 'Text Properties' dialog

is pretty straightforward and we can insert text, make changes to the font and size as well as change the orientation of text. One interesting new addition to KiCad 7 is the 'Knockout' option (**Figure 6**). If you input some text into the Text Properties dialog and click the Knockout checkbox, then the text will be created as a solid silkscreen block with the text removed. It's a great effect, looks smart, and is very readable – a welcome new feature (**Figure 7**).

Finally on adding text, sometimes you might like to add text to the silkscreen layer as a graphic rather than directly as text. We've briefly looked at importing graphics before for either creating edge cuts geometry or for importing logo graphics from Inkscape. One notable point is that if you use the text creation tools in Inkscape and then directly try to load them as a graphic element, this will fail as KiCad SVG import doesn't recognise the text elements. This is pretty easy to rectify. As an example, we created our stoRPer text in Inkscape and then, with the text object selected, we click Path > Object to Path (**Figure 8**). As usual, we edit the document properties in Inkscape so that the document is the size of the text object – we then save the file as a standard SVG. In the PCB Editor, we then select File > Import > Graphics to import the file, ensuring to select the correct 'F.Silkscreen' as the graphic layer. The text graphic then imports correctly and can be placed in the design where required. □

# KiCad, different PCB substrates

In this part of the ongoing KiCad series, let's look at some of the different materials PCBs can be fabricated from



**Jo Hinchliffe**

Jo Hinchliffe is a constant tinkerer and is passionate about all things DIY space. He loves designing and scratch-building both model and high-power rockets, and releases the designs and components as open-source. He also has a shed full of lathes and milling machines and a CNC kit!

**S**o far in this KiCad series, we have designed our PCBs and had them manufactured using the common PCB material known as FR4. On FR4, our copper traces sit on top of fibreglass, and this is what most people think of when they hear the phrase 'PCB'. However, it's not the only option, and most of the PCB fabrication houses have a variety of different materials that they can make a PCB from. The circuit is typically still copper, but these traces sit on top of other materials. Let's take a look at the alternatives.

FR4 is a standard of material for the dielectric, the insulating non-conducting part of a PCB. FR4 is a type of composite fibreglass material, made up from fine glass reinforcing fibres and epoxy resin,

and is very non-conductive. The 'F' and 'R' are an abbreviation of 'fire-retardant', which is one of the many benefits and safety features of the material. Whilst fibreglass composite materials have inherent fire-resistant qualities, FR4 has bromine added which characteristically will reduce the spread of fire.

As well as being essentially fireproof, FR4 has a low thermal expansion coefficient – this means that it won't expand or contract very much in hot or cold environments. This is important because if a PCB expands or contracts a large amount, then it is likely that traces and components on the board will be damaged, crack, or disconnect. Fibreglass composites are strong and most PCB houses will offer a choice of thickness of FR4 boards, allowing you to choose a thickness and strength suitable for your application.



**Right** Some small polyamide flexible PCB antennas fabricated by OSH Park

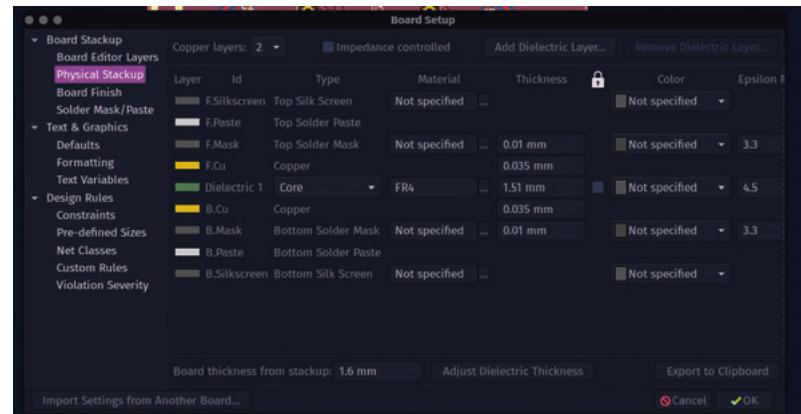
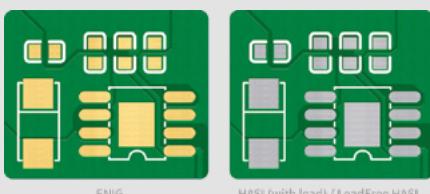
## GOLD OR SILVER

There are lots of other choices that can be made when getting a PCB fabricated. One choice is the type of surface finish applied to any exposed copper pads. These take the form of differing types of covering or plating, which act to both allow the easy fitting of components, solder, or solder paste, whilst stopping bare copper pads from oxidising if left uncovered.

'Hot Air Solder Levelling' (HASL) is a common option where the board is dipped fully into molten solder, removed, and excess solder driven off. This results in all copper areas being covered in a flat, thin layer of solder. It's an older technology and has been used in PCB fabrication for multiple decades and, as such, it's usually an affordable option. It is easy to solder onto, offers good protection against oxidisation, and is stable and long-lasting. It has a long shelf life, so if you don't get around to populating your PCBs for a long time, they will still be in good condition. Finally, HASL is often offered with options that contain lead or options that are lead-free.

'Electroless Nickel Immersion Gold' (ENIG) is a newer technology that applies a two-layer coating to the exposed copper parts of a PCB. The layer directly on top of the copper is a thin layer of hard nickel which creates a barrier stopping the copper from oxidising. To stop the nickel plating from oxidising, a fine layer of gold is immersion-plated over the top. When components are soldered onto ENIG surfaces, they create a strong bond with the nickel layer, and therefore the copper underneath. ENIG is also suitable for covering larger planes on the PCB, so has become a popular choice.

There are other surface finishes, such as 'Immersion Tin', 'Organic Coating' (OSP), and 'Immersion Silver', but HASL and ENIG are most common. Most PCB manufacturers use either HASL, lead-free HASL, or ENIG, or some PCB manufacturers will offer them as choices. For example, JLPCB has HASL selected as default, but you can switch to lead-free HASL or ENIG. OSH Park PCBs are all created with an ENIG surface finish. Over on PCBWay, you can select from a larger range that includes HASL, ENIG, OSP and more – you can even specify to have no coating whatsoever.



As FR4 is the most common PCB material, there aren't many special considerations or changes needed when designing a PCB for FR4. If you want to set up KiCad to display your target board thickness when using the 3D viewer, you can change this by adjusting the dielectric thickness value via the Board Setup. This dialog is found at File > Board Setup in the PCB Editor, then you need to select the 'Physical Stackup' option from the list, and then adjust the 'Dielectric 1' thickness variable (**Figure 1**).

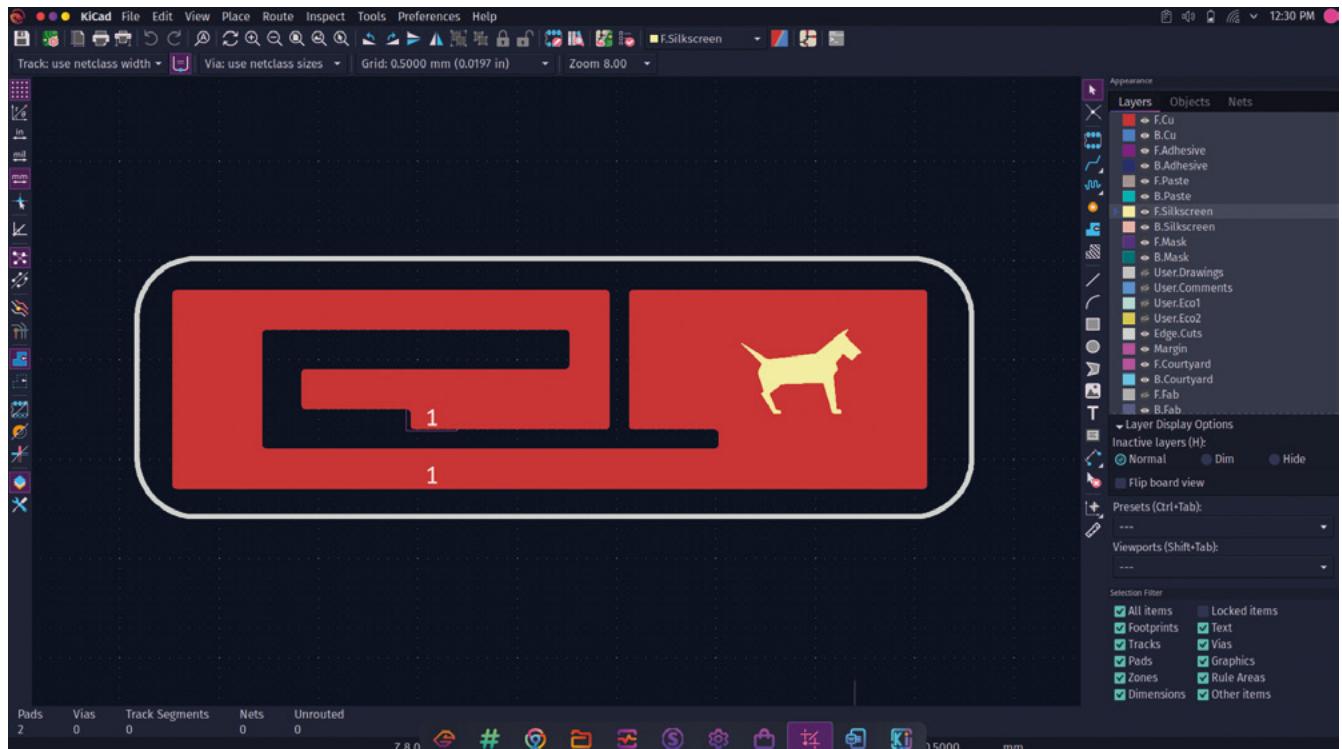
Flexible PCB designs are commonplace in modern electronics, with many use cases. Probably the most

**“** There are numerous types of flexible PCB substrate, with the most common being polyamide film **”**

common to come across are simple flex connectors. These have obvious advantages, in that they can connect components or PCB modules in different locations in a system. Flex connectors can be inserted into specific clamping sockets or, indeed, can be designed to directly solder onto copper pads. They have the benefit of being able to fold and bend around, allowing complex layups of PCBs which do not need larger header pins or sockets. There are numerous types of flexible PCB substrate, with the most common being polyamide film. Traces and pads work in a very similar manner to any PCB, →

**Figure 1** ◆  
Changing the thickness of the 'Dielectric 1' layer to change the overall thickness of our PCB design

**QUICK TIP**  
Most PCB fabrication services have detailed information on the thickness of each layer of their PCBs. You can use this to emulate the PCB accurately in KiCad.

**TUTORIAL**

**Above** Our flexible antenna design in KiCad

## COPPER CONUNDRUM

Copper weight describes the thickness of copper at any given point on a PCB. It's often expressed as ounces per square foot, so a 1 oz/ft<sup>2</sup> copper weight will be thinner than 2 oz/ft<sup>2</sup>. Copper weight can be important in terms of designing PCB traces, as the weight affects the depth of the trace. Different trace widths and weights may need to be used when considering the amount of current particular parts of a circuit may be passing. Similarly, track impedance and RF qualities of tracks may well come into play, as well as track sizes, when trying to match lengths of track in more complex PCB designs.

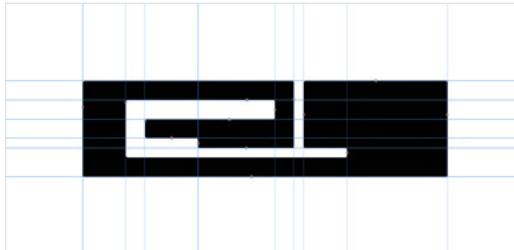
Most common copper weightings are 1 oz or 2 oz, and many PCB houses will offer these as a choice. If you require a precise copper weighting, it's possible, at a price, for some PCB fabricators to offer more bespoke weight of copper by plating or etching extra material to or from the board.



It's good advice to read any guidance your PCB fabrication house has to offer and to speak to them directly

in that they are a thin layer of copper. Flexible PCBs can become quite complex to design for, and many PCB fabrication houses will have options for custom layer materials in flex PCBs. These can also include rigid sections, which means you have to supply a design that can distinguish different substrates within it. If you plan to use this technology, it's good advice to read any guidance your PCB fabrication house has to offer and to speak to them directly to explain your concept. At the simpler end of flex PCBs, it can be as straightforward as laying out a regular PCB design, exporting some Gerbers, or sending a project file to your PCB fabricator.

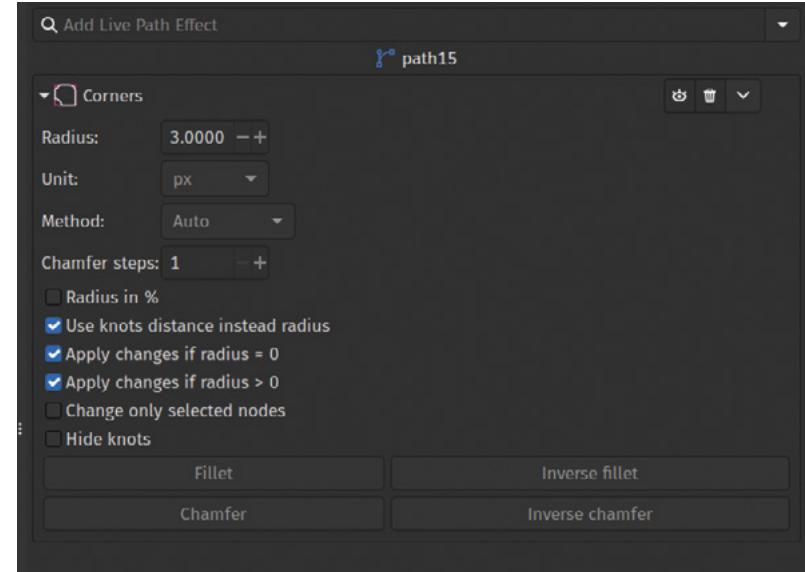
A good use case for flex PCBs, and a good simple example, is a flexible antenna. We'd found a paper online with an image and dimensions for a dual-band 2.4GHz and 5GHz patch antenna, which piqued our curiosity, so we set about laying it out in KiCad. We actually began in Inkscape: the PDF source we



**Figure 2** Using Guidelines in Inkscape to manually trace the antenna design image

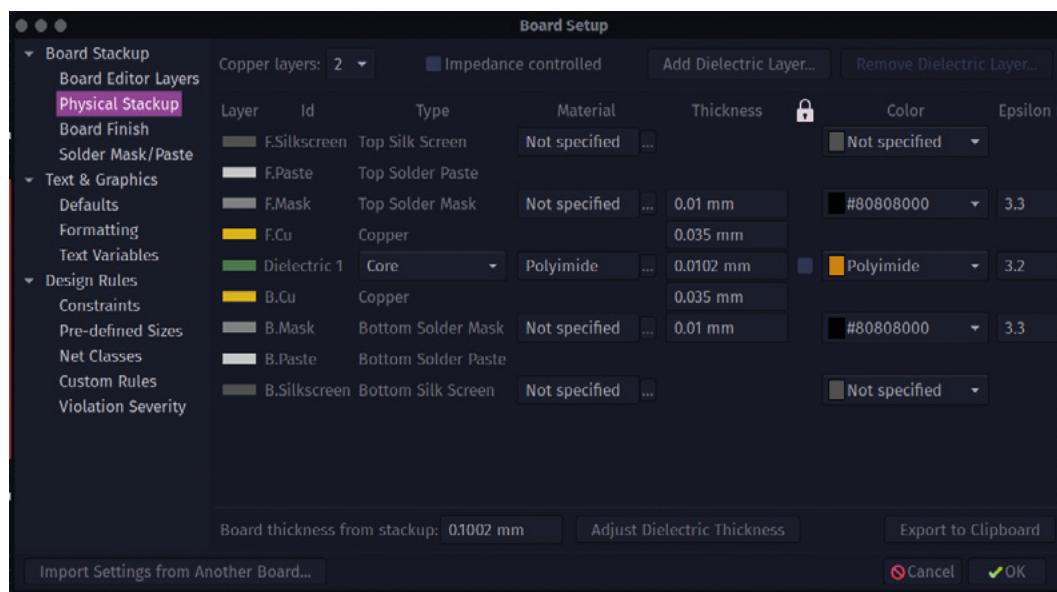
had for the design was imported into Inkscape, but it had a gradient fill and therefore wouldn't export correctly as the whole object was full of nodes. If this was a bitmap image, it would have been a good candidate for Inkscape's 'Trace Bitmap' function, but as it was a vector file, this wouldn't work. However, as a vector image, it was straightforward to drag in vertical and horizontal guide lines and snap these to the edges of the PDF antenna drawing (**Figure 2**).

Once we had a guideline for every part of the antenna in place, it was a simple job to use the pen tool to draw a continuous line around the part, closing it to form a solid object. There is some discussion online around not using sharp 90-degree angles in traces when aiming for a flex PCB as, if the PCB is flexed, the sharp corners can be origin points for tears and failures. Whilst not a massive concern for this design, it was easy to add internal and external radial chamfers to the antenna object



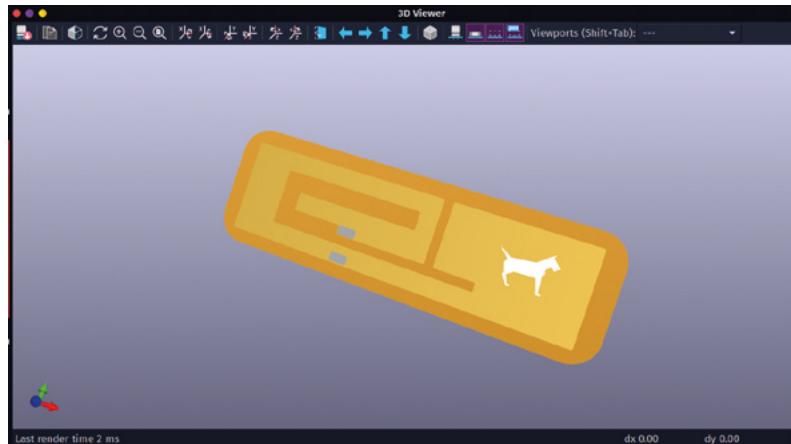
by selecting the object, then using the 'Path effects' dialog to apply the 'Corners' path effect chamfer option (**Figure 3**). Finally, we deleted the original imported PDF, resized the document to fit our antenna design, and then saved it as an SVG.

In a new KiCad project, we actually ignored our usual workflow of creating a schematic and associating parts to schematic symbols and went straight to the PCB Editor. You can directly add components and create traces in the PCB Editor with no schematic in place. For a more complex project, we wouldn't recommend this approach, as you →



**Figure 3** Using the 'Corners' path effect in Inkscape to add internal and external chamfers

**Figure 4** By setting the board material, dielectric thickness, and solder mask colours, we can emulate a flex PCB in the KiCad 3D viewer

**TUTORIAL**

**Figure 5** A 3D render of our flex circuit

have no means of checking connectivity or creating net connections, but for small simple projects like this, it's pretty easy.

We imported our antenna SVG, making sure to import it onto the Front Copper layer (F.Cu). We then used the 'Add a Footprint' tool to add two small SMD pads. We positioned these on the points in the antenna design that the original design had indicated. With no net connections due to no schematic, these pads will be directly connected to the large copper antenna design that we just placed, but of course, the pads will have no solder mask

“  
All that remains is to then quickly draw another SVG for the outline of the antenna  
”

over them, allowing us to solder on a connecting coaxial cable. All that remains is to then quickly draw another SVG for the outline of the antenna, which we did in Inkscape, but you could just draw in KiCad. With the outline imported to the edge cuts layer, we have a completed design. To get the design fabricated, we used OSH Park which has a flex PCB offering. One of the nice features of OSH Park is that you don't have to produce Gerber files to upload – you can upload your KiCad PCB file directly to the website for previewing and ordering. Conveniently, either project files or Gerber files don't particularly specify the board substrate or thickness, so we don't need to specify a thin flex board design in KiCad. However, we might want to model the board

**FILLING HOLES**

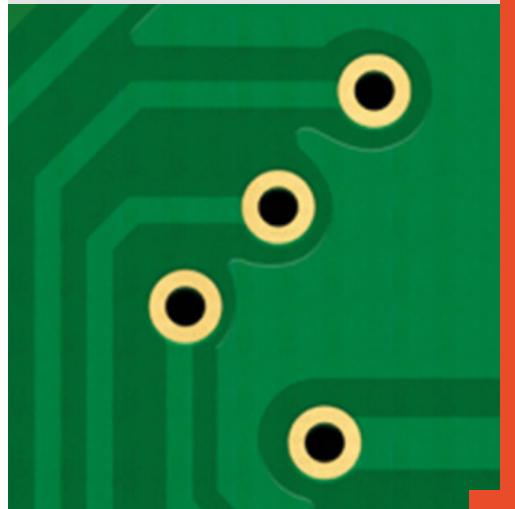
Vias, the small plated through-holes that connect different layers of the PCB, have different ways of being finished. For many projects, the PCB fabrication house default will be fine, but it's worth looking at the common options offered.

'Tented vias' are covered with the solder mask, so no solder would stick to them. The hole may or may not be filled, depending on the size of the via. Other benefits of tenting are that you reduce the risk of unintended shorts when boards are being assembled into products or being handled etc.

'Un-tented vias' have no covering, so are finished in the selected surface finish in the same way as exposed pads and other copper features. Whilst this may well be fine, there is a risk of accidentally soldering to vias or for short circuits.

'Plugged vias' are filled in a couple of different ways, or you may have a choice. One way is to fill the via with solder mask; another is to fill the via with epoxy resin. Some manufacturers may only be able to fill vias up to a certain diameter or, indeed, some PCB houses can offer custom approaches where you can ask for all vias of a certain diameter to be filled. The benefit of plugging vias is that the via can't accidentally become filled with solder or other conductive material.

'Conductive plugged vias' are not the most common choice, but some PCB houses can fill vias with conductive material. This can increase the amount of current the via is capable of passing. There are trade-offs in that the conductive filler may thermally expand at different rates than the other board materials, causing small flexes leading to potential failures. As an example, JLCPCB offers the option to fill vias with conductive copper-filled epoxy.





accurately in KiCad, especially if we are using either renders of the KiCad design in promotion or if we are exporting the board 3D model for use in other CAD programs.

Again, we can use the board setup dialog and the 'Physical Stackup' tab in the PCB Editor to emulate a flex circuit. Your PCB fabrication house will have data about all the thicknesses of each layer of their flexible PCB offerings and you can use this to set thicknesses in the board setup. If you just need a close enough PCB view that looks like your flex design, you can simply adapt the major thickness of the board by changing the material to 'Polyamide' and setting the thickness of that layer to 0.0102 mm (this is the OSH Park flex polyamide layer thickness), as in **Figure 4**. You can then set the colours of the top and bottom solder mask layers to transparent by reducing the opacity to zero using the 'Custom colour' option. This, in the 3D viewer, will then give you a reasonable approximation of a flex PCB (**Figure 5**). Often, PCB manufacturers offer metal substrates for PCBs, commonly copper or aluminium. These substrates can be useful when you need to dissipate heat quickly through a system. Often, aluminium substrates can make sense for

temperature sensor modules where you want the sensor to be accurate but the board not to soak up heat. Another application for metal substrate PCBs is where you want the PCB to act as a heatsink. For example, we had JLCPCB make and assemble some 1-watt LED modules (**Figure 6**). Running at 1 watt, these LEDs generate a reasonable amount of heat and, to promote their long life, it's useful to have some kind of heatsink attached. The reverse side of the LED has a large thermal pad which connects to the board, and the reverse of our aluminium board is bare metal. This acts as a heatsink for the LED module and the temperature is reduced. One thing of note is that aluminium and copper substrate PCBs can be bent if they are put under pressure or load. In fact, when your aluminium PCBs arrive in panels, it can be quite hard to remove them without bending the PCB!

FR4, flexible, and aluminium are not the only options often when we move to high-speed designs, designs running at microwave frequencies, or when we have very special applications like medical devices – there are other substrates available. Rogers, PTFE, Teflon, Copper Core – it's fun to read around some of these more esoteric materials. □

**Figure 6** ◇  
Our small aluminium  
LED module

## Part 1

# Design a circuit with KiCad



**Stewart  
Watkiss**

MAKER

Also known as Penguin Tutor. Maker and YouTuber who loves all things Raspberry Pi and Pico. Author of *Learn Electronics with Raspberry Pi*.

[penguintutor.com](http://penguintutor.com)

[twitter.com/  
stewartwatkiss](https://twitter.com/stewartwatkiss)



## Warning! Electrical Safety

Whilst 12V will not cause electrocution, it can cause a fire. Ensure power supplies have over-current protection and consider adding a fuse.

[magpi.cc/  
electricalfires](http://magpi.cc/electricalfires)

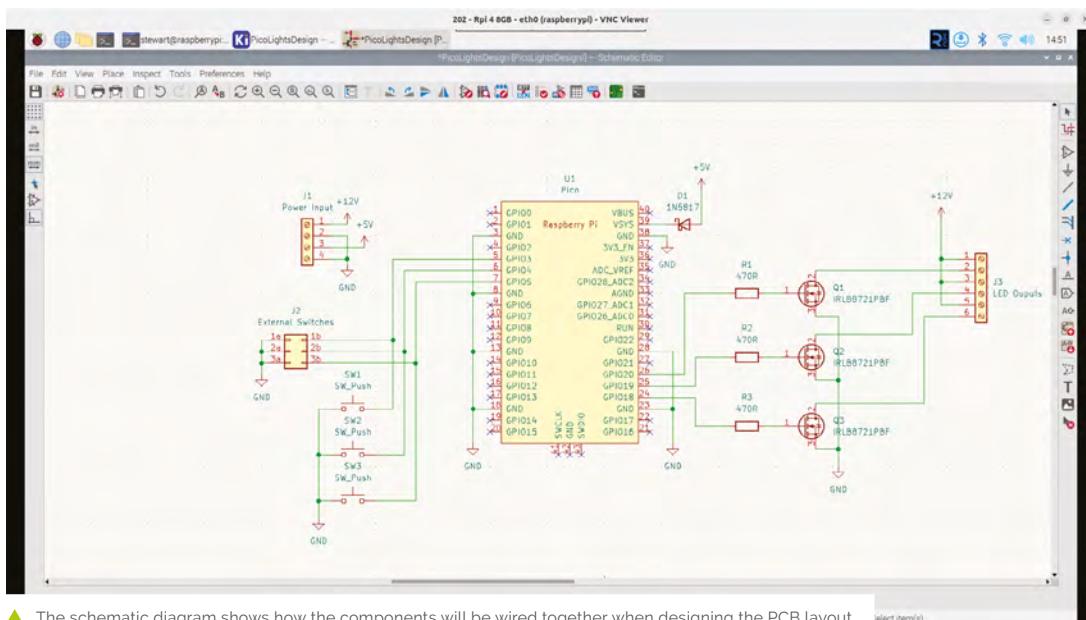
This tutorial will provide guidance on how to design your own circuit using KiCad. It will show how you can design a circuit that can be used with Raspberry Pi Pico. This will include choosing suitable components and designing a schematic diagram. This will then lead to creating your own custom printed circuit board (PCB) in the next tutorial.

This circuit is to control three sets of lights using buttons on the PCB or through a web interface. The design can be used for 5V or 12V lights, making it suitable for either home automation or model railway lighting.

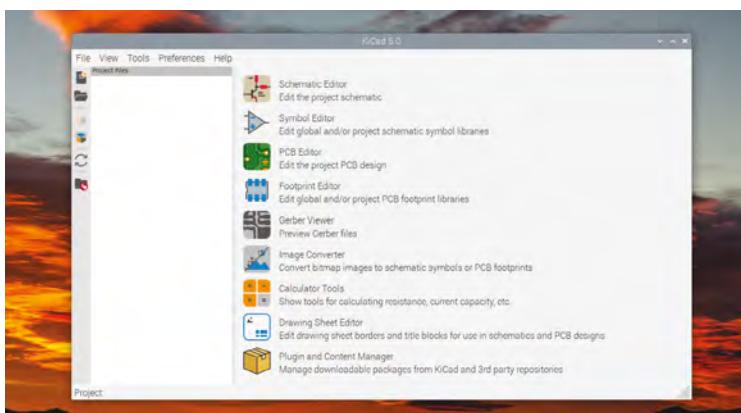
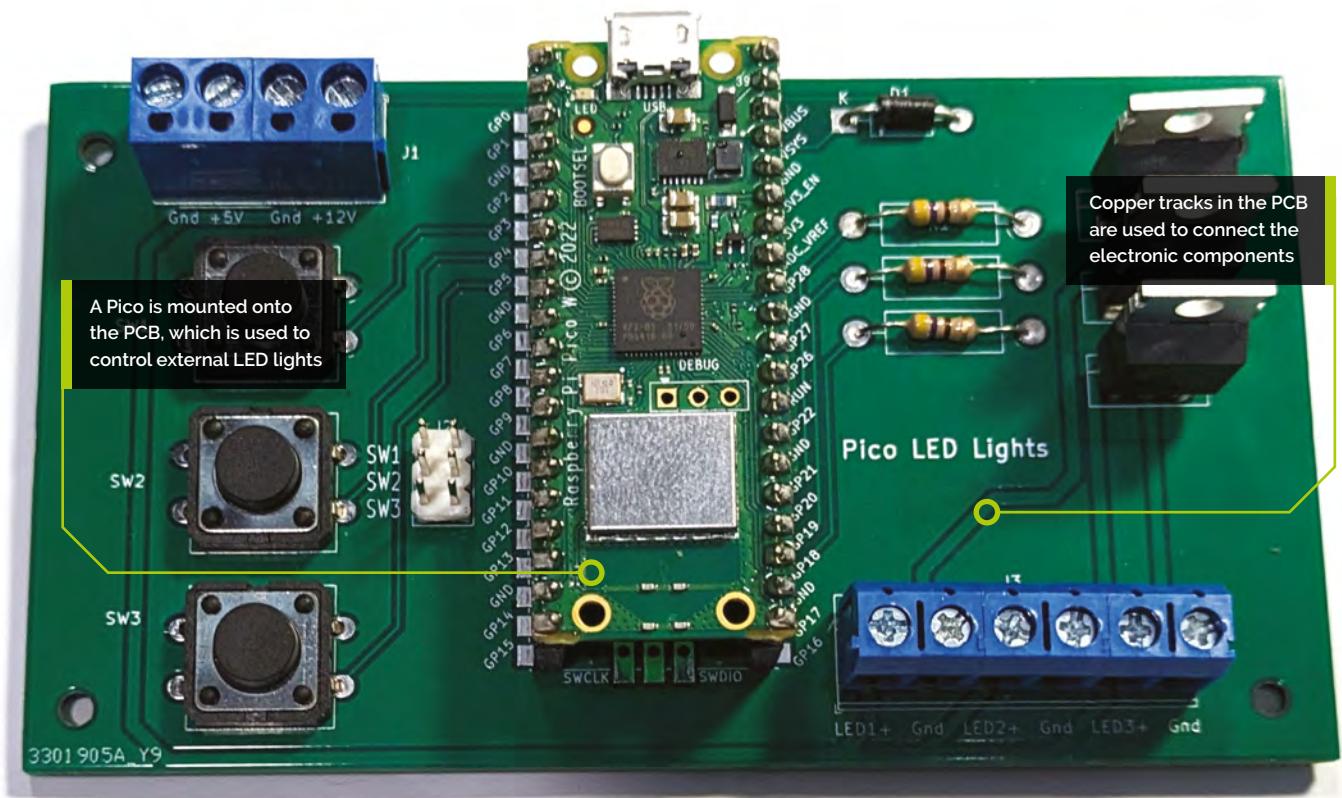
## 01 Design idea

All projects start with an idea. When creating a breadboard circuit, you have an opportunity to experiment and change the design as required. Creating a custom PCB involves additional time and cost, so it is important to spend additional time in the design phase to get the circuit just how you want it.

It is often useful to create a design specification that lists the features that you want, anything you want to avoid, and any restrictions it needs to be designed for. There may be restrictions on size, or you may want to provide additional flexibility to



The schematic diagram shows how the components will be wired together when designing the PCB layout



add extra features. Sometimes these may conflict with each other, in which case you may need to make some compromises. Listing these up-front helps to keep your design on track.

## 02 Creating the initial design

With the idea and specification ready, you can start to make basic decisions about the circuit. Our first decision was to use a Raspberry Pi Pico. This project could be made using a Raspberry Pi computer, but it doesn't need

that amount of power for simple switching. Without the overhead of an operating system, Pico is more responsive, more reliable, and cheaper. A Pico W can be used to provide Wi-Fi access.

The plan is to switch high-power LEDs which are more than can be powered just using the GPIO pins on a Pico, so this is going to need MOSFET switch circuits.

It also needs to use switches for input, and these can be wired between GPIO pins and ground, using the internal pull-ups in Raspberry Pi Pico.

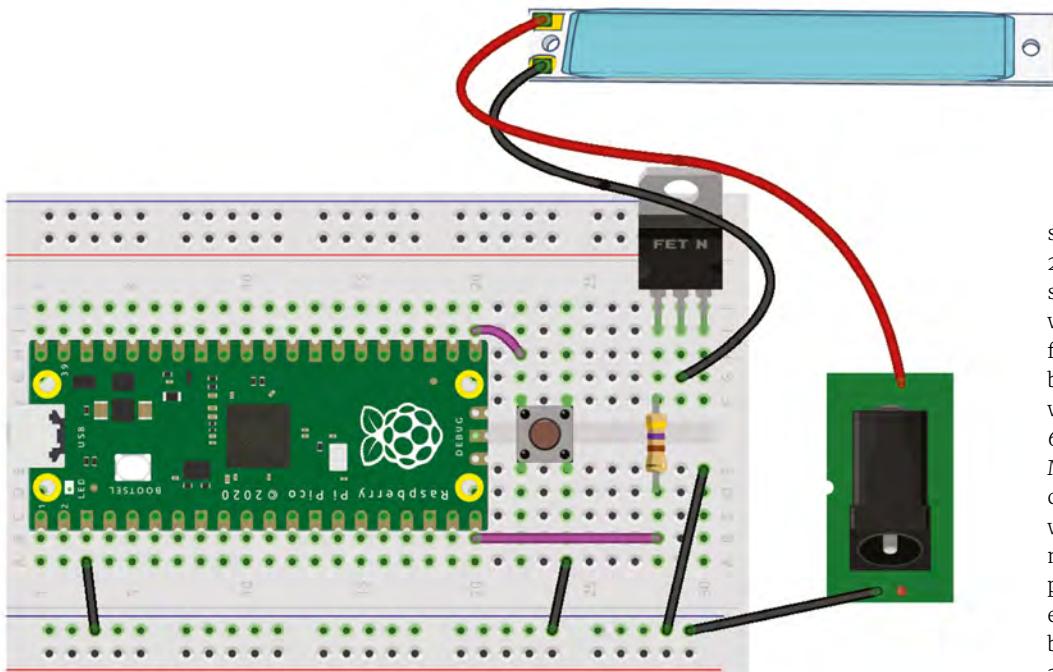
## 03 Flexible design

One thing to consider when designing a PCB is whether it will be used for a single circuit or whether it can be used for multiple purposes. It can be useful to include additional flexibility as that helps justify the cost of having a PCB made, but adding extra features will increase the size and cost of the PCB.

► **Figure 1:** KiCad includes different tools which can be used to help in designing printed circuit boards

## You'll Need

- ▶ 470Ω resistors [magpi.cc/470ohm](https://magpi.cc/470ohm)
- ▶ 1N5817 diode [magpi.cc/1n5817](https://magpi.cc/1n5817)
- ▶ Switches [magpi.cc/12x6switches](https://magpi.cc/12x6switches)
- ▶ PCB screw terminals [magpi.cc/pcbterminal](https://magpi.cc/pcbterminal)
- ▶ IRLB8721 MOSFET [magpi.cc/mosfet](https://magpi.cc/mosfet)
- ▶ 5V COB LED light [magpi.cc/cobled](https://magpi.cc/cobled)
- ▶ Power adapter [magpi.cc/jacktoscrew](https://magpi.cc/jacktoscrew)



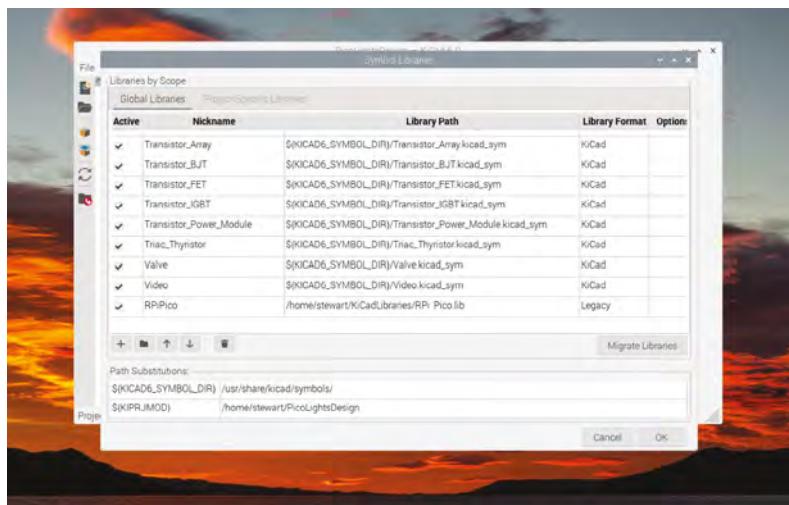
▲ Figure 2: A breadboard prototype with a COB LED light strip and a connector for a 5V power supply

The main thing here is what LED lights are to be controlled. In the specification for this build, it was decided that the board should be capable of controlling both 12V or 5V LED lights. The 12V lights would be useful for camping lights or model railways, and 5V would be useful for COB (chip-on-board) lighting strips. An appropriate voltage power supply is needed to match the LEDs used.

▼ Figure 3: To include a Pico in the design, add the RPi\_Pico.lib library through the Manage Symbol Library menu option

## 04 Component selection

Having decided on the LEDs, you can now choose a MOSFET that is sufficient for controlling them.



A typical MOSFET for switching LEDs is the 2N7000. This MOSFET can switch up to 200mA, which will likely be sufficient for model railway lights, but not for the light strips which can draw up to 600mA. Looking at what MOSFETs are available, you can find the IRLB8721PBF which supports up to 62A, more than we need with plenty to spare. It is more expensive than the 2N7000, but the savings in having a single transistor type for different requirements

means that we can make savings by using a single PCB design for multiple projects.

## 05 Prototyping

The next stage is to test if your design will actually work. This is where the breadboard comes in useful because it allows you to try out

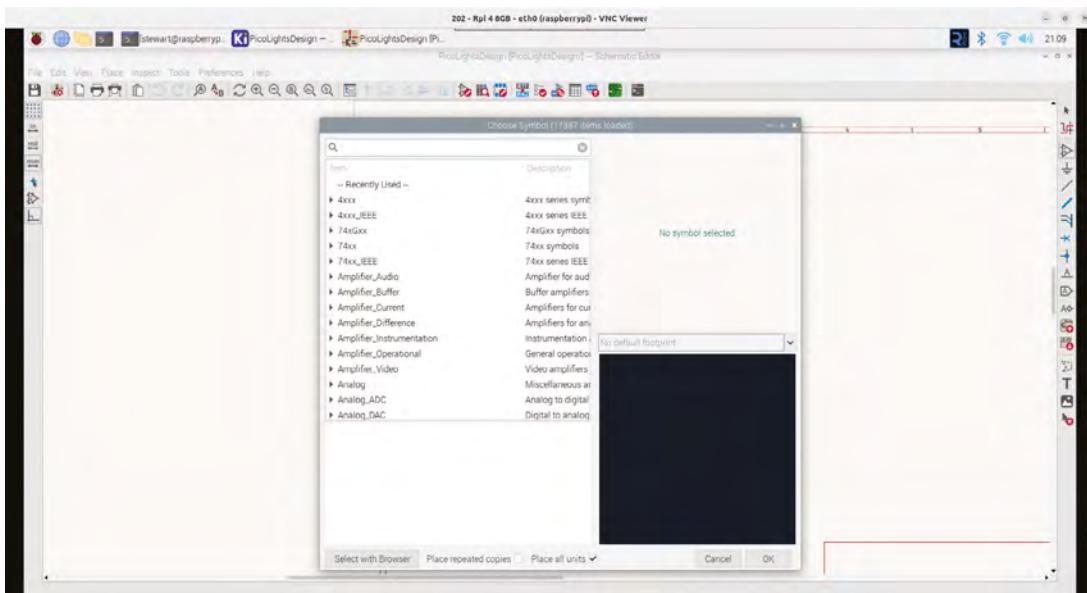
**■ The next stage is to test if your design will actually work ■**

different circuits and values. You may want to use a multimeter, or even an oscilloscope to check that the outputs are what you expect.

In this case, the electronics are made up of common circuit configurations, but as it's a different MOSFET than before, you may want to test it to see if it behaves in the way that you expect. The diagram in **Figure 2** shows an example using single switches and a single COB LED light strip to test the main components.

## 06 Moving to KiCad

In the design so far, you've hopefully been making notes about the decisions you've made. Now it's time to convert those to create a schematic diagram. The schematic diagram is



useful because it shows each of the components wired together without the difficulty of trying to follow individual wires.

The tool used here is KiCad, which is open-source software capable of creating professional circuits. You can run KiCad on most computers, including Raspberry Pi. To install on a Raspberry Pi, use the terminal command:

```
sudo apt install kicad kicad-packages3d
```

## 07 Creating the schematic diagram

After launching KiCad, you should see the project window. There should be no project selected and the right-hand side of the window shows the various tools which make up KiCad. This is shown in **Figure 1** (previous page). Create a new project by clicking on the new project icon and giving it a name, such as PicoLights. It will create a new directory and create a KiCad project file. You can then click on the Schematic Editor on the right to start a new schematic diagram.

You are presented with a blank drawing area. You can move around using a mouse with the right button pressed, and zoom using your mouse scroll wheel.

## 08 Adding a Pico footprint to KiCad

Whilst various models of Raspberry Pi are included in KiCad's component library, it does not currently include a Pico. To add a Pico symbol, first download the file from [magpi.cc/kicadzip](http://magpi.cc/kicadzip) (note: direct download).

This file is a complete PCB design, but within the zip file are the files **RPi\_Pico.lib** and **RPi\_Pico\_SMD\_TH.kicad\_mod**. Copy both files to a suitable directory (eg. **KiCadLibraries**) and choose Tools > Manage Symbol Libraries.

Click on the '+' icon to create a new library, name it 'RPiPico', then click on the folder icon in the Library Path column and select the lib file. It will add it as a Legacy library. This is shown in **Figure 3**. That .kicad\_mod file will be used later when creating the PCB.

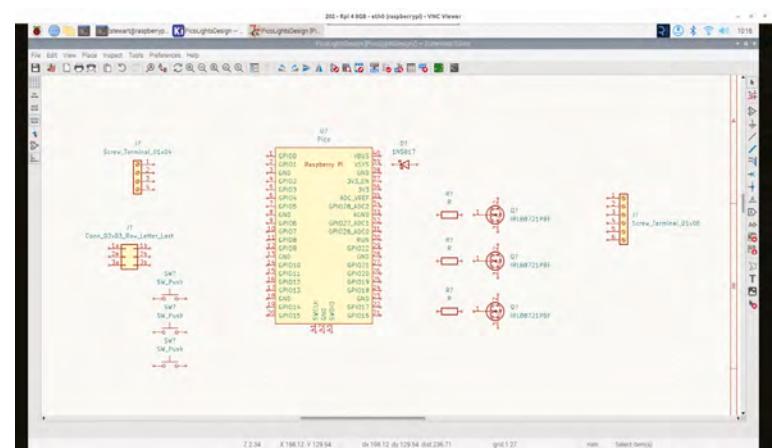
## 09 Adding components to the schematic

You can now add components by choosing Add Symbol from the Place menu, or by using the icon on the right-hand side of the schematic editor. Click the screen to bring up the symbol selector dialog, search for Pico, and select the one inside

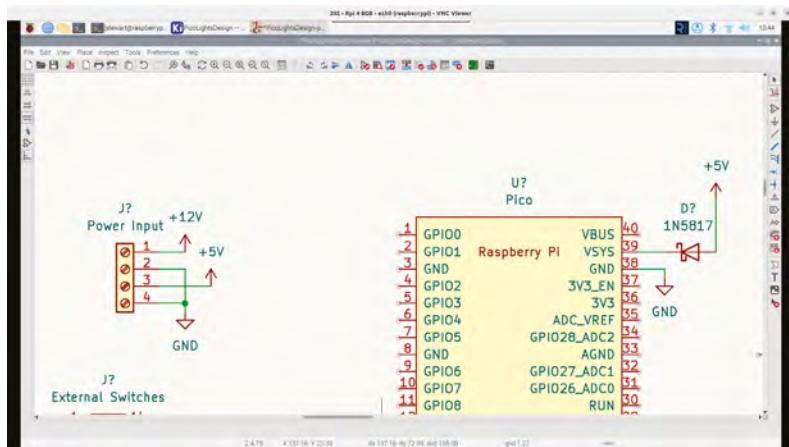
## Top Tip

### Power options

Your Pico can be powered through a 5V connection to the screw terminal, or by using the USB port on your Pico



**Figure 5:** All components are added and can be arranged by moving into position using the M key



▲ Figure 6: Power label symbols are used to indicate the power supplies. Labels with the same name are treated as though they are connected

the RPiPico library, then click OK to add it to the schematic diagram. The symbol selector is shown in **Figure 4**.

For the button switches, use SW\_Push. This will also provide headers which can be used to connect to an external switch that can be added as 'Conn\_02x03\_Odd\_Even'.

The MOSFET is the IRLB8721PBF and add gate resistors (search for 'R' for the resistor symbol).

## simple-lights.py

► Language: MicroPython

**DOWNLOAD THE FULL CODE:**



[magpi.cc/picolights](http://magpi.cc/picolights)

```

001. from machine import Pin
002. import utime
003.
004. sw1 = Pin(3, Pin.IN, Pin.PULL_UP)
005. sw2 = Pin(4, Pin.IN, Pin.PULL_UP)
006. sw3 = Pin(5, Pin.IN, Pin.PULL_UP)
007.
008. out1 = Pin(20, Pin.OUT)
009. out2 = Pin(19, Pin.OUT)
010. out3 = Pin(18, Pin.OUT)
011.
012. while(1):
013.     if (sw1.value() == 0):
014.         # Toggle out1
015.         out1.value(1-out1.value())
016.         utime.sleep (0.5)
017.     if (sw2.value() == 0):
018.         out2.value(1-out2.value())
019.         utime.sleep (0.5)
020.     if (sw3.value() == 0):
021.         # Toggle out3
022.         out3.value(1-out3.value())
023.         utime.sleep (0.5)

```

## 10 Powering the circuit

To provide power, a screw terminal is used. This needs two power supplies: one for your LEDs (12V or 5V depending upon LEDs) and one for the Pico (5V). These will be supplied using a 4-way terminal connector, Screw\_Terminal\_01x04. A 1N5187 diode is also needed to allow either the USB or screw terminal to be used.

The output will also be provided through screw terminals to connect to your choice of LED. Choose 'Screw\_Terminal\_01x06'.

The components can be moved into an appropriate position and rotated or mirrored using the right-click menu. Arrange the components so that they are similar to **Figure 5**.

## 11 Adding power connections

The value of the components can be changed using the 'Edit' value from the right-click menu. Change the resistors to 470R and rename the connectors to represent their purpose.

Rather than having to wire up all the different places where power is needed, power symbols can be used. Use +12V, +5V, and a common ground for both supplies' GND. If you place multiple power symbols with the same reference (you can use duplicate from the right-click menu) then they will all be connected together, reducing the number of wires needed. Connect the power symbols to the various components using 'Add wire' from the Place menu. The power connections for the input and Pico are shown in **Figure 6**.

## 12 Wiring

The rest of the components can be wired together to complete the circuit. When joining wires, there will be a circle over the connection to show that they are connected. If there is no circle, then crossing wires are not electrically connected.

You should also annotate the components by giving each a unique reference. You could do this manually using the properties of each component, changing 'J?' to 'J1' etc., or use 'Annotate Schematic' from the tools menu.

Code is provided (**simple-lights.py**) to allow you to test the prototype using the button LED. In the next tutorial, you will use the schematic to create a professional printed circuit board.