A Tutorial on PCB Design

— using KiCad

Instructor: Xiaoguang "Leo" Liu
University of California Davis
lxgliu@ucdavis.edu

September 30, 2015

Contents

1 KiCad **3**

2 Example 1: Arduino dice **4**
2.1 Starting KiCad . . . 4
2.2 Schematic Capture . . . 5
2.3 Creating Schematic Symbols . . . 11
2.4 Associating schematic symbols with footprint . . . 16
2.5 Creating or Editing Footprint . . . 16
2.5.1 SMD capacitor . . . 16
2.5.2 Push button switch . . . 21
2.5.3 ATmega328P microcontroller . . . 23
2.5.4 The rest of the components . . . 23
2.6 Laying out the PCB . . . 25
2.7 Generating Fabrication Files . . . 31

3 Example 2: RF amplifier **34**
3.1 Some Unique Concepts to High Frequency PCB Design . . . 34
3.1.1 Controlled Impedance Lines . . . 34
3.1.2 Microstrip lines with Top Copper Shield . . . 38
3.1.3 Transmission Lines Bends . . . 39
3.1.4 Short transmission lines . . . 40
3.2 Circuit schematic and component selection . . . 41
3.2.1 RF-choke inductor . . . 41
3.3 Component footprints . . . 43
3.3.1 ADL5602 footprint . . . 43
3.3.2 Inductor footprint . . . 43

3.3.3 SMA connector footprint 43
3.3.4 VCC and GND footprint 46
3.4 PCB Layout 46
3.4.1 Copper fill 46
3.4.2 Via fences 48

4 Further Reading **51**

1 KiCad

In the early days, PCBs are designed and laid out literally by hand. See Fig. 1 for an example board from that era. As technologies developed, it become more common to do the job with the help of a computer. Today, there are numerous software tools for PCB design. On the high end, industry-grade packages, such as Cadence Allegro [1], Mentor Graphics Xpedition, and Altium Designer, offer extensive features and capabilities with a high price tag and often a very steep learning curve. On the lower end, popular choices include CadSoft EAGLE, ExpressPCB, and DesignSpark, all of which offer a reasonable set of features at an affordable price.

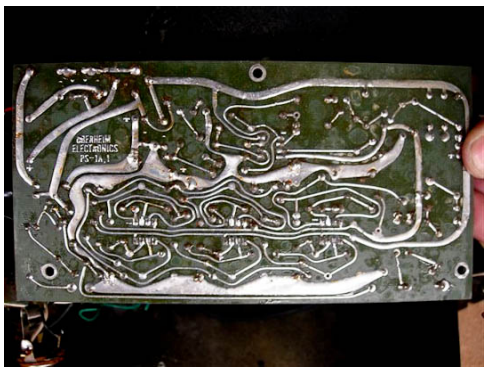


Figure 1: A vintage PCB laid out by hand. Source: Link.

In recent years, KiCad has emerged as a popular open-source software package for designing and laying out PCBs. KiCad is available on all three major personal computer operating systems, Windows, Linux, and Mac OS. Compared with the above mentioned software packages, KiCad is completely free of charge or any other limitation. Although KiCad is not as sophisticated as industry-level tools, it is capable of dealing with fairly complicated designs, and the active developer community is working hard to improve its capabilities. In fact, as of this writing, KiCad has not had an official stable release for the last two years because of the constant development progress being made. In this tutorial, we will be using the 2015 stable release candidate 4.0.0 RC1.

[1] UC Davis students have access to the full suite of Allegro PCB design tools through a donation from Cadence.

2 Example 1: Arduino dice

In the first example we will make a small eletronic dice consisting of a ATmega328P microcontroller[2], a switch, a 7-segment LED display, and some misc resistors and capacitors. Every time you press and release the switch, the microcontroller will generate a random number (1–6) for the dice value and display it on the 7-segment LED. This example is a stripped down version of a project from PrinceTronics.

Table. 1 lists the components that are needed for this example.

Table 1: List of components for Example 1.

Item Description	Quantity	Digikey Part #
Arduino Uno board, DIP version	1	1050-1024-ND
16-MHz crystal oscillator	1	300-6034-ND
22-pF ceramic capacitor, SMD, 0603, 5%	2	445-1273-1-ND
1-uF ceramic capacitor, SMD, 0603	1	1276-1041-1-ND
10k-Ohm resistor, SMD, 0603, 1/10W, 5%	2	P10KGCT-ND
Push button switch, 0.05 A, 24 V	1	SW400-ND
7-segment 1-digit display, common cathode	1	516-2734-ND

2.1 Starting KiCad

Fig. 2 shows the main window of KiCad. The main window serves as a project management panel where you can launch the individual PCB tools.

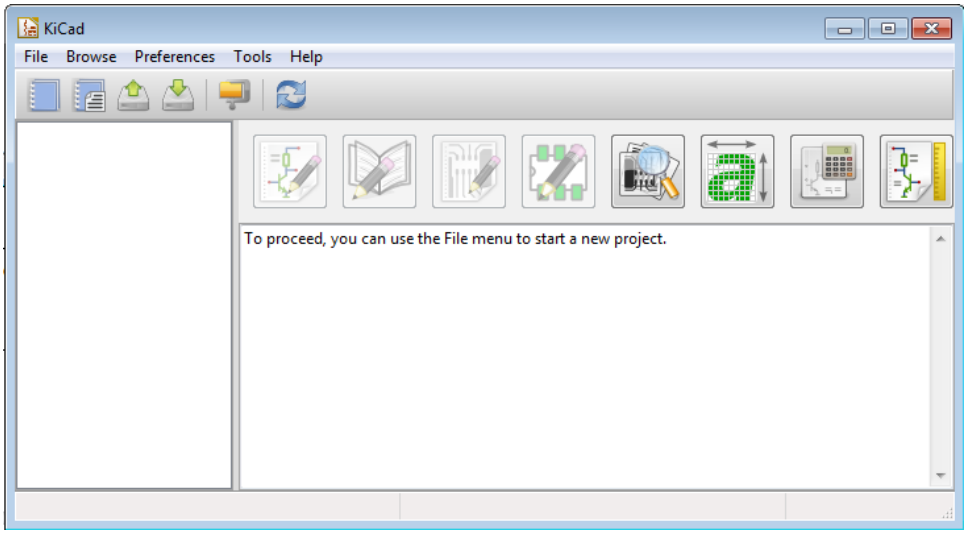


Figure 2: The main window of KiCad.

[2]The heart of the Arduino UNO platform.

Table 2: Individual tools within KiCad.

	Eeschema: schematic editor/capture tool		Gerbview: Gerber file viewer
	Schematic symbol editor		Bitmap2Component: A tool for creating component symbol from a picture
	Pcbnew: PCB layout tool		A calculator for common PCB design related calculations
	Component footprint editor		Schematic sheet layout editor

2.2 Schematic Capture

1. Click on the Eeschema icon. A new schematic window should appear.
2. Save your schematic design with the file name "arduino_dice.sch".
3. The default library that comes with KiCad installation has schematic symbols for many ATmel micro-controllers, including the ATmega328P that is used on the Arduino platform. You can place the symbol on your schematic by the following steps.
   a) Click on the "Place a component" button from the toolbar on the right side.

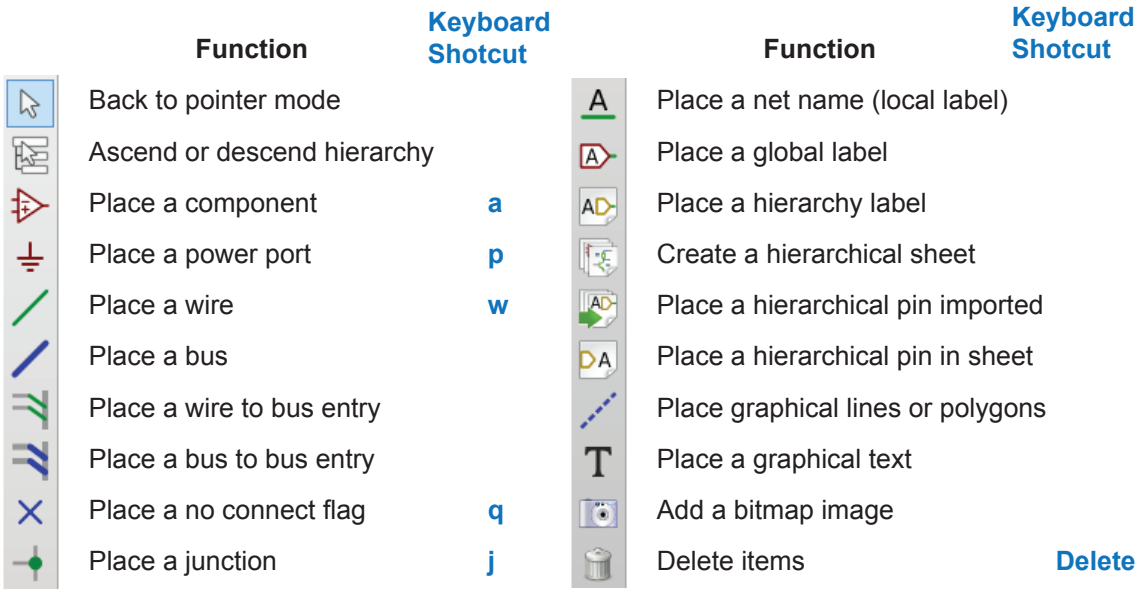


Figure 3: Eeschema toolbar icons.

b) Click anywhere on the schematic, a dialog box should appear.

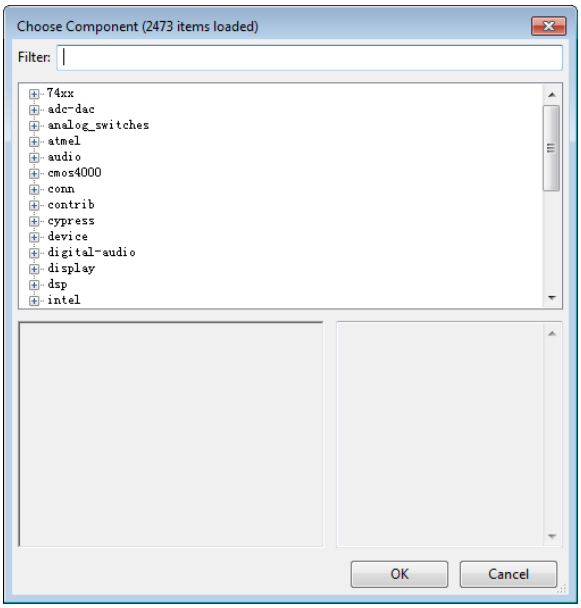


Figure 4: Place component dialog box.

c) We'll add our first item, the ATmega328P microcontroller, from the "atmel" library. Select the "atmel" entry, and click "OK". A new dialog box appears for you select the particular device, the ATMEGA328P-P. Click "OK". Note that if you already know the name of the component, you can simply start typing the name and Eeschema will filter out the components with the same initial characters. It wouldn't take long before you arrive at your desired component.

d) The ATmega328P symbol should now cling to your mouse cursor. Click on the schematic to place it at a location you like.

e) A number of component editing operations are available by right clicking on the component. Some of the operations have keyboard shortcut. The general way to use the shortcut is to place the cursor on the component and press the corresponding shortcut key. Pressing the "ESC" key will cancel the current operation.

   i. "Move component" will move the component and break all circuit connections to it. To retain the connections, use "Drag component".

   ii. "Orient component" has further options to rotate and mirror the component. Experiment the shortcuts by pressing "r" or "y" while placing the cursor on the component.

   iii. "Edit component→ Edit" will bring up a dialog box that allows you to edit all of the component properties.

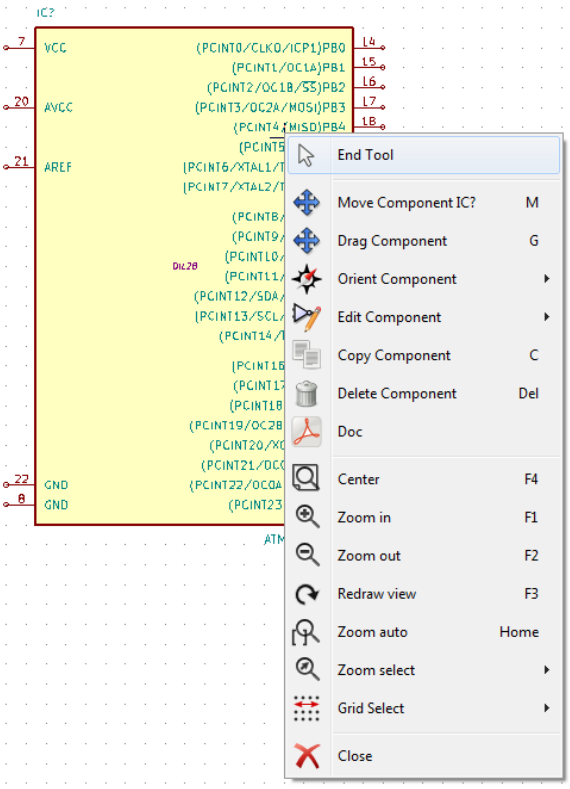


Figure 5: Edit component pop-up menu.

A. The "Reference" and "Value" are the two properties that you are most likely to edit in this dialog box. "Reference" is the annotation of the component. In this case, it should read "IC1". You may also write it as "IC?" where the "?" is a placeholder for a numeric value. KiCad can auto annotate the components and assign a unique value for each component; we will look at how to do this shortly.

B. The "Value" entry is usually used to mark the component value. For resistors, capacitors, and inductors for examples, the "Value" can simply be their corresponding resistance, capacitance, and inductance values. For this ATmega microcontroller we will simply use the "Value" to mark the components name.

C. For now, we will delete the value for the "Footprint" field.

4. Follow a similar procedure to place the other circuit components. Table. 3 lists which library they belong to and their names in the library.

   Note that the *Vcc* and *ground* symbols, and all other symbols related to providing power to the circuits, are organized into the *power* library. They can be accessed directly by clicking the *Place a power port* button from the toolbar on the right. Place a *Vcc* and several *GND* pins where necessary.

Table 3: Schematic components for Example 1.

Component	Library	Name in Library	Value
ATmega328P	atmel	ATMEGA328P-P	
Two 22-pF capacitors	device	C	22 pF
One 1-uF capacitor	device	C	1 uF
One 16-MHz crystal oscillator	device	CRYSTAL	16 MHz
One 7-segment LED display	display	7SEGMENTS	
One push button	device	SW_PUSH	
Two 10-k resistors	device	R	10 k
2-pin header	conn	CONN_01X02	
Vcc	power	vcc	
Ground	power	GND	

5. Connect the components together according to Fig. 6 by placing wires between corresponding pins. The components should be arranged to make connecting them together easier with less wire clutter. To start placing a wire, click on the "Place a wire" button, then click on the starting point, and finish by clicking on the endpoint of the wire. It is often easier to use the keyboard shortcut. First move your cursor to the starting point and press the "w" key. A wire is started at the cursor location. Move the cursor to the endpoint and click to finish the connection.

6. The schematic drawing can become difficult to read if there are too many wire crossovers. "Named netlist" can be used to alleviate the issue. Although our example circuit is quite simple and easy to read, we will still use it to illustrate how to "clean up" the schematic with named net.

   a) Delete the wire between the ATmega328P's *PB1* pin and the 7-segment LED's *DP* pin.

   b) Draw a short section of wire on the ATmega328P's *PB1* pin; one of the ends of the wire is now floating.

   c) Click the "Place a net name – local label" button and then click on the schematic. A dialog box will appear, input "DP" in the "Text" field, and click "OK". The text "DP" can now be seen to cling on the cursor.

   d) Click on the floating terminal of the short wire on the *PB1* pin to finish naming a net. You should now see the text DP attached to the wire on the *PB1* pin; the little hollow square at the floating end of the wire has also disappeared.

   e) Repeat steps b–d for the *DP* pin of the 7-segment LED. The ATmega328P's *PB1* pin and the 7-segment LED's *DP* pin are now connected by the net name "DP" even though there is no direct wire connection on the schematic view.

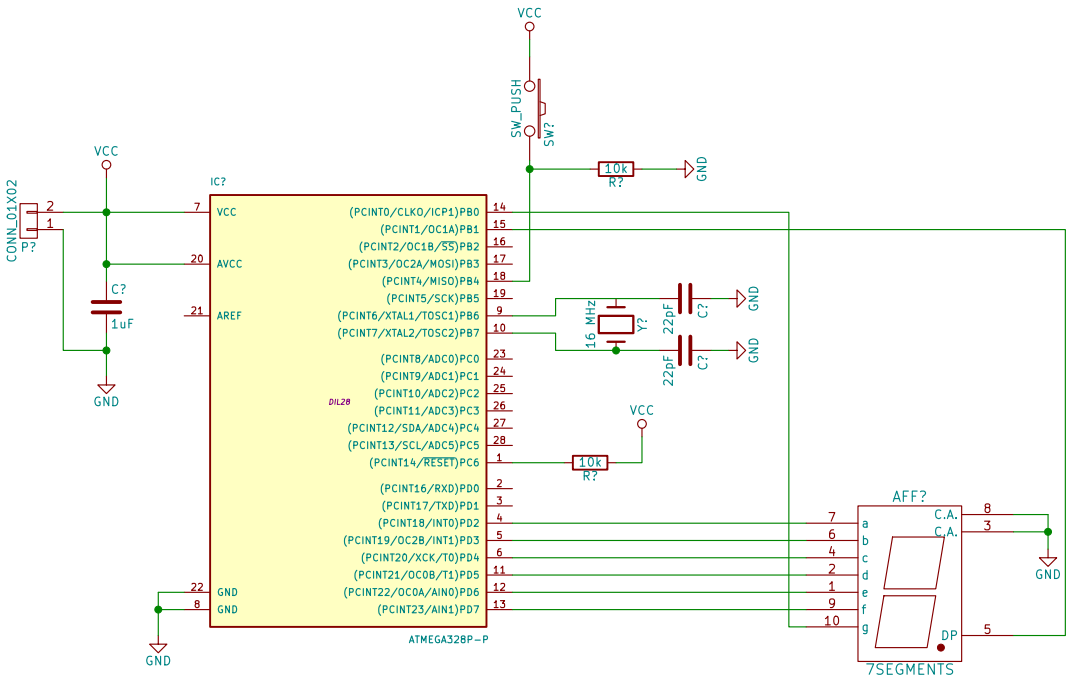


Figure 6: Initial schematic drawing of Example 1 circuit.

f) Repeat steps a–e for the connection between ATmega328P's *PB4* pin and the push button. Refer to the final schematic (Fig. 7) for how it looks.

7. In most circuits, the unused pins can be left floating. In this example, however, we will terminate all the unused pins by placing a no connect label on them; this tells KiCad to ignore these pins during the electrical rule check (ERC).

8. The schematic capture is now almost done. Notice that the references to some of the components still have question marks. For example, the two capacitors connected to the crystal oscillator look identical to each other; we need to differentiate them. In KiCad, we do this by annotating the schematic, i.e. giving each component a unique identifier (reference). Annotation can be done manually by changing the "Reference" property of a component and making sure that each reference is unique, but it is much easier to let KiCad do the annotation automatically.

   a) Click the "Annotate" button.

   b) A dialog box appears. The options are all self-explanatory.

   c) Click the "Annotate" button to finish. You must have noticed that you can "un-annotate" the schematic by clicking the "Clear annotation" button.

   d) After annotation, you should see that all the components are numbered.

9. It is always a good idea to run an ERC before proceeding. ERC checks the electrical connections between components and try to detect potential errors in

the schematic.

10. This will create a netlist file that describes the circuit connections. The netlist file will be used to guide the PCB layout process.

11. The final schematic should look like that in Fig. 7.

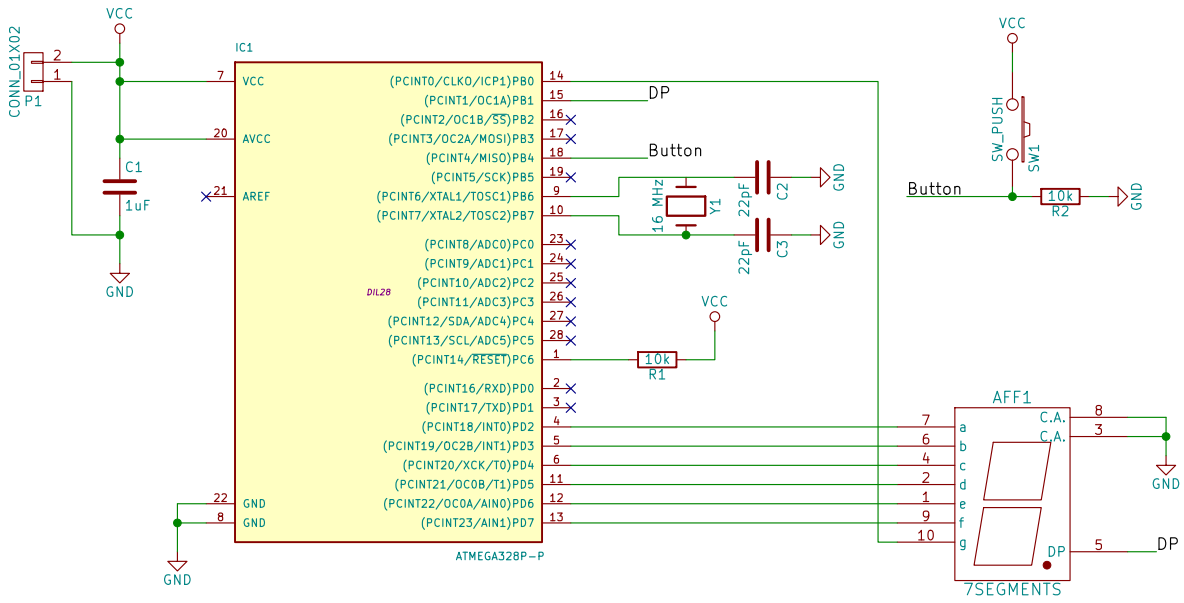


Figure 7: Completed schematic drawing of Example 1 circuit.

2.3 Creating Schematic Symbols

Sometimes you run into a situation where you cant find a proper symbol in the default KiCad library for the component that you want to use. The following steps will show you how to create a schematic symbol of your own.

1. From the KiCad project window, click the "Schematic library editor" button to launch the schematic symbol editor and library manager. Alternatively, you can launch it by clicking the same button from Eeschema. The library editor window should appear.

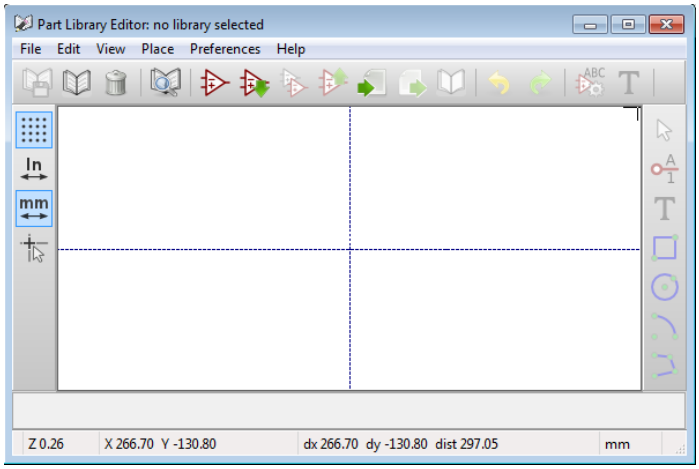


Figure 8: The schematic library editor program

2. Click the "Select working library" button to set the current working library. A dialog box should appear with the same list of component libraries that we saw when we placed components on the schematic.
   a) If you are creating a new component, it is recommended that you select the project library, which is by default named "*project_name*-cache" and should be the last one in the list. In this example, select the library named "arduino_dice-cache".
   b) If you are trying to edit an existing component, click the corresponding library.
   c) Click "OK" to finalize the selection.

3. In this example, we are actually not missing any schematic symbol. Just for the sake of demonstrating how to use the library editor, we will build our own version of the ATmega328P microcontroller IC. Fig. 9 shows the pin diagram of the ATmega328P.

4. Click the "Create a new component" button. A dialog box will appear. Put "ATmega328p" as the component name. Leave everything else as default. Worth mentioning is the "Number of parts per package" setting. Sometimes a

component may have many pins and it becomes difficult to route the schematic (or to be more precise, it becomes difficult to read the schematic when you have lots of wire connections). It may be easier to split a component into several schematic symbols so that routing can be easier. In such a scenario, you can have multiple parts per package.

5. Click "OK" and you should see the component "Reference" and "Value" strings appearing near the center of the library editor main window.

6. Click the "Add graphic rectangle to the component body" button , and use the mouse cursor to draw a rectangle from $(x = -0.400'', y = 0.700'')$ to $(0.400'', -0.700'')$ (Fig. 11). The current position of the cursor is displayed at the bottom of the main window. Notice that the positive direction of the $y-$axis points downward.

7. Add component pins.
   a) Click the "Add pins to the component" button
   b) Click anywhere in the main drawing window. A dialog box should appear. Put "PC6" for the "Pin name", "1" for the "Pin number", "Right" for the "Orientation", "Input" for the "Electrical type", and leave everything else as default.
   c) Click "OK", and then click at $(-0.600, -0.650)$ to add the pin.

8. Repeat the above process and add the rest of the pins as follows.

Pin Name	**Pin number**	**Orientation**	**Electrical Type**	**Pin Name**	**Pin number**	**Orientation**	**Electrical Type**
PD0	2	Right	Input	PC5	28	Left	Input
PD1	3	Right	Input	PC4	27	Left	Input
PD2	4	Right	Input	PC3	26	Left	Input
PD3	5	Right	Input	PC2	25	Left	Input
PD4	6	Right	Input	PC1	24	Left	Input
VCC	7	Right	Power Input	PC0	23	Left	Input
GND	8	Right	Power Input	GND	22	Left	Power Input
PB6	9	Right	Input	AREF	21	Left	Input
PB7	10	Right	Input	AVCC	20	Left	Power Input
PD5	11	Right	Input	PB5	19	Left	Input
PD6	12	Right	Input	PB4	18	Left	Input
PD7	13	Right	Input	PB3	17	Left	Input
PB0	14	Right	Input	PB2	16	Left	Input
				PB1	15	Left	Input

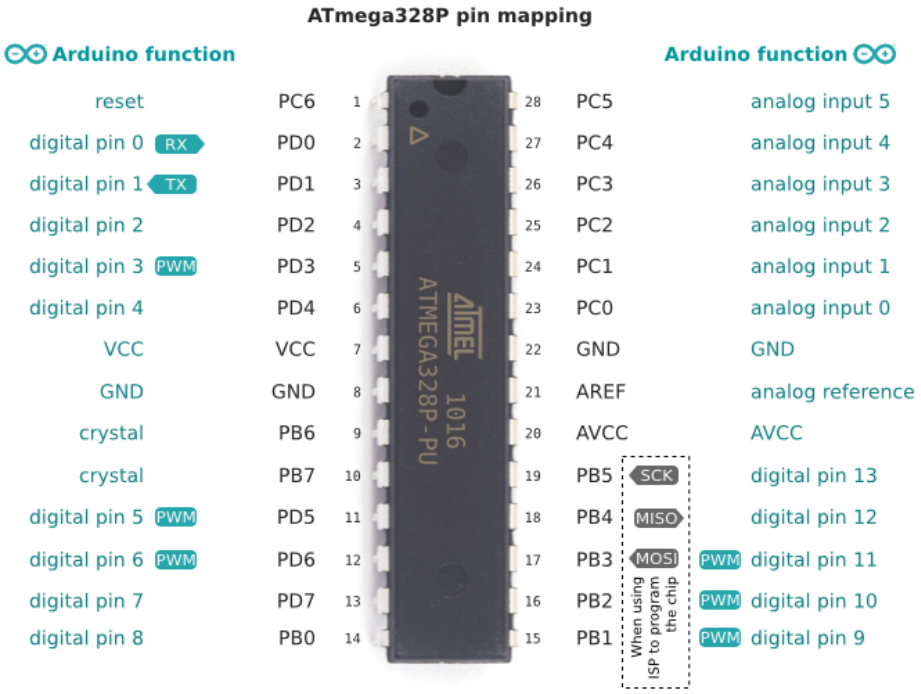


Figure 9: Pin mapping for the ATmega328P microcontroller. Source: Link.

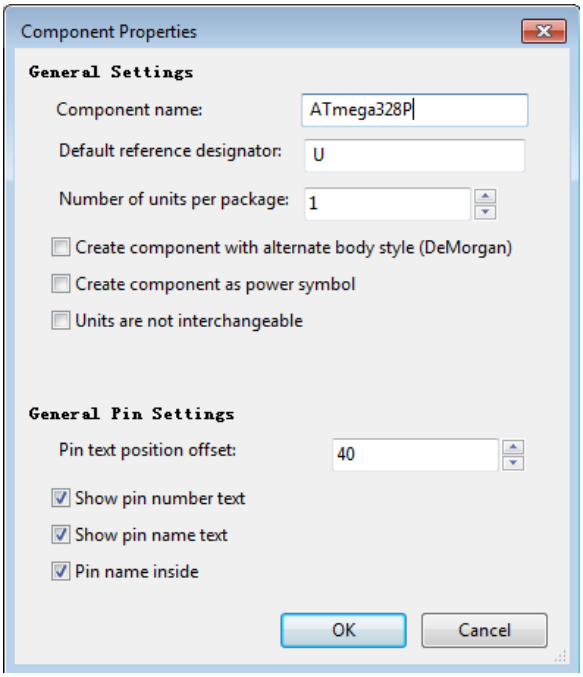


Figure 10: Creat component dialog box.

9. Now move the "Reference" string above the rectangle (only for aesthetic reasons) and the "Value" string below the rectangle. You can also add a half circle (arc) to the top of the rectangle to indicate the orientation of the component. The completed component looks that in Fig. 12.

10. Click the Save current library to disk to save your work. This should complete the creation of the schematic symbol. To verify that you have successfully created the symbol, go back to the Eeschema window and check whether you can find the ATmega328P symbol you just built in the "arduino-dice-cache" library.

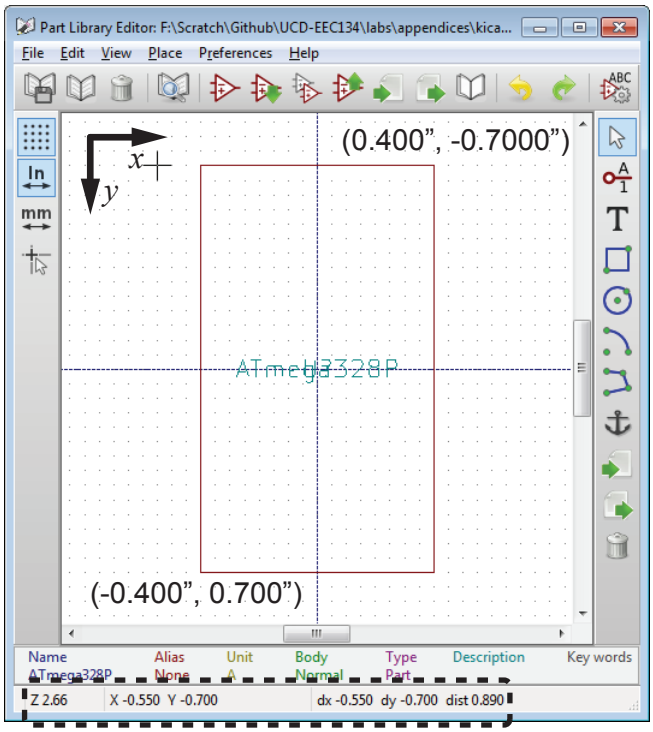


Figure 11: Drawing a rectangle in the library editor main window with the help of the built-in cordinate system.

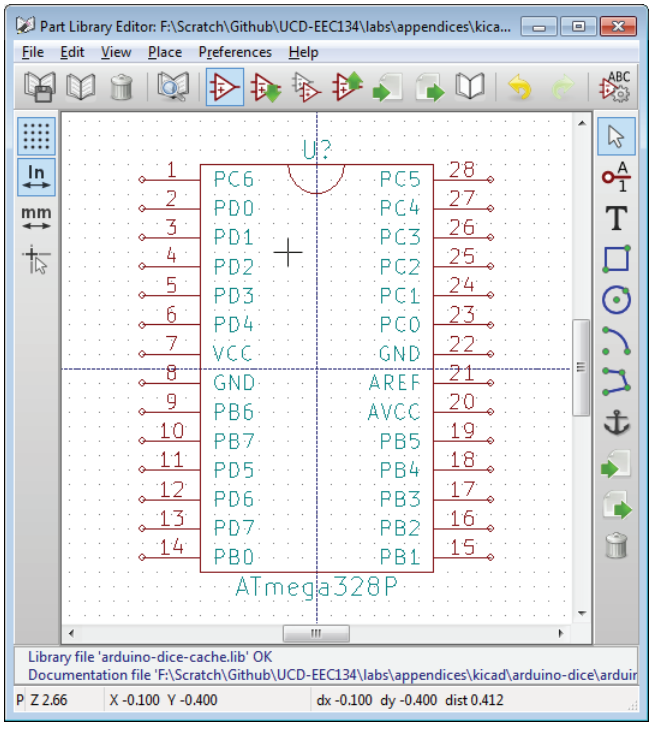


Figure 12: Completed schematic symbol for the ATmega328P microcontroller.

2.4 Associating schematic symbols with footprint

In KiCad, the schematic symbol and the footprint of a device are stored in separate files (.lib for schematic symbols and .mod for footprint). In order for the PCB layout tool (Pcbnew) to put the correct footprints in the layout, we need to assign a relationship between the schematic symbols and the footprints in a circuit. The program CvPcb is used to do this.

1. Open CvPcb from the Eeschema window. By default, CvPcb should automatically loaded the netlist of the project. If not, you can go to "File→Open" to open the desired netlist. Ignore any error messages at this stage.

2. The CvPcb window consists of three panes. The leftmost one lists the footprint libraries. The center one lists all the schematic components used in the circuit. The right one presents a list of available footprints. These footprints are installed with the KiCad program. By default, CvPcb presents only footprints that it thinks that are relevant to the components. To see a full list of available components, uncheck the "Filter footprint list by keywords" button.

3. Our job in CvPcb is to associate the schematic components with their corresponding footprint. This is done by first selecting the component in the left pane, and then double clicking on the right footprint in the right pane. You can preview the drawing of the footprint by clicking the "View selected footprint" button .

4. **A very important rule in PCB design is to never trust footprints provided by others unless 1) they are directly from the component vendors; 2) you have verified them yourself.** The default footprint library that comes KiCad is particularly error prone. For this reason, we will be creating all the footprints by ourselves in this example. The next section outlines this process.

2.5 Creating or Editing Footprint

2.5.1 SMD capacitor

We will start with the simplest component in the circuit, the 22 pF capacitor. This capacitor is a 0603 SMD capacitor produced by TDK Corporation. "0603" refers to its lateral dimensions in thousandth of an inch (normally referred to as a **mil** or a **thou**), that is, the capacitor is roughly 60 mil long and 30 mil wide. If we look at the dimension drawing provided by the vendor (Fig. 13a), we see that its actual dimensions are 63 mil×31 mil. In fact, the capacitor is realy sized in the metric system, with a nominal dimension of 1.6 mm×0.8 mm. So this capacitor is a "0603" in imperial units and "1608" in metric units. **The difference between the imperial and metric units is a common source of confusion when choosing components. Alwasy double check!**

On the Digikey product page, we can find the datasheet to the capacitor "C Series, Gen Appl & Mid-Voltage Spec". Page 15 of the datasheet, copied here in Fig. 13b, shows the recommended land pattern (footprint) for the capacitors in this serie of products. The recommended dimensions for the 0603 (1608 metric) capacitor is highlighted; here we will use the median value of 0.7 mm for A, B, and C.

Now let's start drawing the footprint for the capacitor.

1. Start the PCB footprint editor by clicking on the "PCB footprint editor" button.

2. Click the "New footprint" button to create a new footprint. Put "smd_0603" as the name in the pop-up dialog box. Click "OK".

3. By default, the footprint name "smd_0603" and the Reference string "REF**" will appear in the center of the drawing window.

4. Set the working unit to "mm" (millimeters). Then set the grid size to "User grid". The user grid size can then by set by going to menu "Dimensions→User Grid Size". Set the unit to "Millimeters", "Size X" to "0.05", and "Size Y" to "0.05".

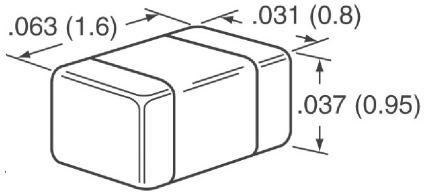
5. Click the "Add pads" button. Move the cursor to position (-0.7,0) and click. A donut-shape patch should appear. This is because by default KiCad assumes a through-hole type pad.

6. To change the pad properties, move the cursor on top of the pad and type keyboard short "e" for editing. The *Pad Properties* dialog box should appear. Change the properties as follows:

Table 4: Pad properties for the SMD capacitor.

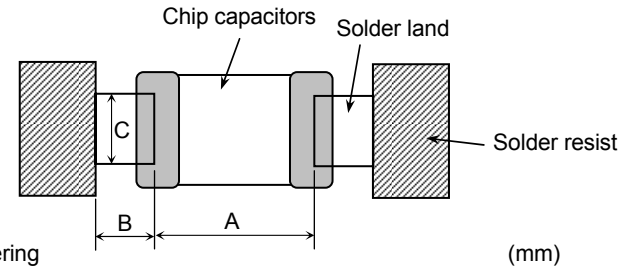
Pad number	1
Pad type	SMD
Shape	Rectangular shape
Position X	-0.7
Position Y	0
Size X	0.7
Size Y	0.7
Layers	F.Cu
Technical Layers	Check "F.Paste", "F.SilkS", and "F.Mask"

7. Repeat Step 5 and 6 to add the second pad at the correct location. You will notice that KiCad assumes the properties of the previous pads as the default properties of the new pad.



(a)

3	Designing P.C.board	The amount of solder at the terminations has a direct effect on the reliability of the capacitors. 1) The greater the amount of solder, the higher the stress on the chip capacitors, and the more likely that it will break. When designing a P.C.board, determine the shape and size of the solder lands to have proper amount of solder on the terminations. 2) Avoid using common solder land for multiple terminations and provide individual solder land for each terminations. 3) Size and recommended land dimensions.



Flow soldering (mm)

Type / Symbol	C1608 (CC0603)	C2012 (CC0805)	C3216 (CC1206)
A	0.7 - 1.0	1.0 - 1.3	2.1 - 2.5
B	0.8 - 1.0	1.0 - 1.2	1.1 - 1.3
C	0.6 - 0.8	0.8 - 1.1	1.0 - 1.3

Reflow soldering (mm)

Type / Symbol	C0402 (CC01005)	C0603 (CC0201)	C1005 (CC0402)	C1608 (CC0603)	C2012 (CC0805)
A	0.15 - 0.25	0.25 - 0.35	0.3 - 0.5	0.6 - 0.8	0.9 - 1.2
B	0.15 - 0.25	0.2 - 0.3	0.35 - 0.45	0.6 - 0.8	0.7 - 0.9
C	0.15 - 0.25	0.25 - 0.35	0.4 - 0.6	0.6 - 0.8	0.9 - 1.2

(b)

Figure 13: (a) Dimensions of the TDK C-series 22 pF SMD capacitors. (b) PCB design recommendation for TDK C-series SMD capacitors. Land pattern dimensions for the C1608 type is highlighted in red.

8. Draw the courtyard outline of the capacitor using "Add graphic line or polygon" tool. The courtyard is the actual dimension of the device plus some margin. It serves as a guide to prevent the PCB designer from placing components too close to each other. Here we have put a decent margin of 0.45 mm in the x-direction and 0.65 mm in the y-direction around the pads (or 0.7 mm in x and 0.6 mm in y around the component). This margin can be adjusted according to the assembly requirement and tolerances in the PCB assembly process.

9. The finished footprint drawing should look like that shown in Fig. 14.

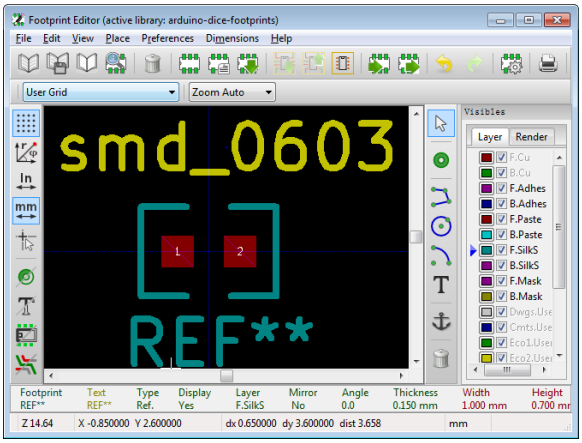


Figure 14: Completed footprint of the SMD 0603 capacitor.

10. We wish to create a new library to contain our newly created footprint. To do this, click the "Create new library and save current module" icon. Set the location and name of the library and click "OK". The "Library Path" field should contain the full path to your desired library location. In this example, we will create a footprint library with the name "arduino-dice-footprints" (KiCad will automatically add the ".pretty" suffix) under the project folder "...\arduino-dice". Fig. 15 shows the proper settings. At this point, a new folder named "arduino-dice-footprints.pretty" should appear in the project tree in the KiCad main window (Fig. 16). Click on the "+" sign will expand the library and show all the footprints in this library; in this case, we only have the "smd_0603" footprint.

11. Although we now have the new footprint library in our file system and the project tree, KiCad still does not recognize it as part of the available libraries that we can pull footprints from. We will need to manually add the library to KiCad's *library tables*.

    a) Open CvPcb and click the "Edit footprint library table" icon. The "PCB Library Tables" dialog box should appear.

    b) Click the "Append with Wizard" button.

    c) In the pop-up dialog box, select "Files on my computer" and click "Next".

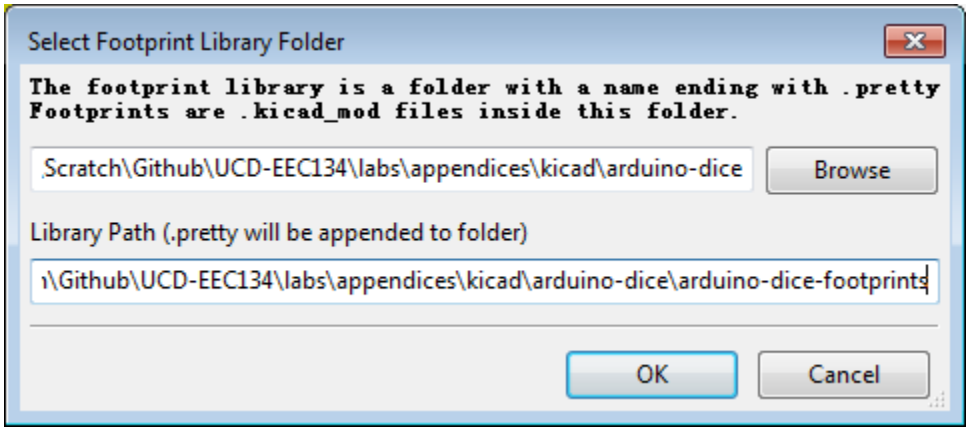


Figure 15: The "Select Footprint Library Folder" dialog box.

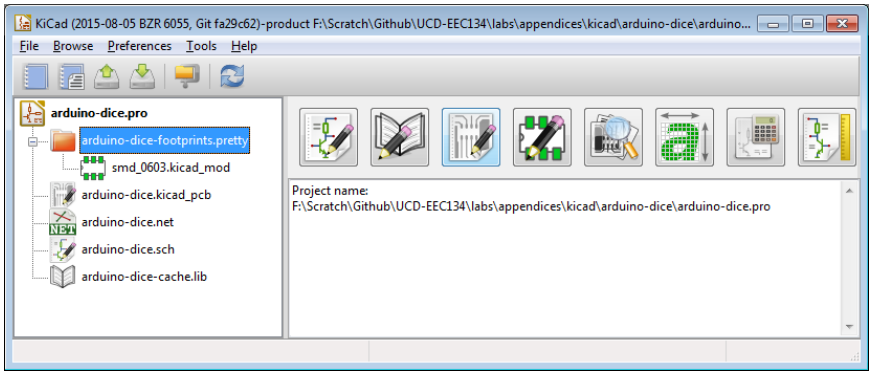


Figure 16: The new footprint library is now visible in the project tree.

d) Browse to and select the footprint library folder we just created, click "Next". Click "Next" again in the next window to confirm.

e) Select "To the current project only" and click "Finish" to finish adding the footprint library to this project. If you have created a generic library that you want to use across multiple projects, you could select the "To global library configuration" option. Fig. 17 shows how the "PCB Library Tables" looks like now. Notice how KiCad replaces the absolute path with the "KIPRJMOD" variable.

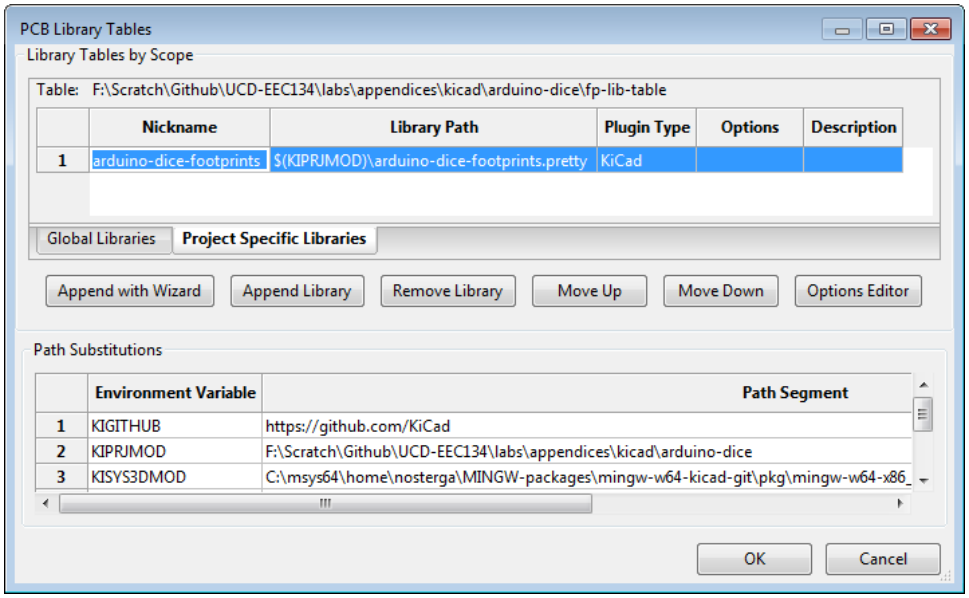


Figure 17: Add a project footprint library.

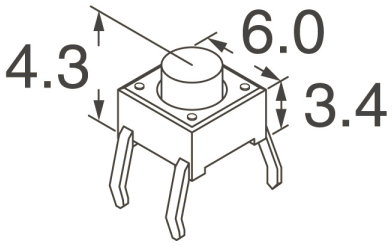
f) Finally, if you plan to create more footprints, it's a good idea to set the active library in the Footprint Editor. In the Footprint Editor window, click the "Set active library" icon, select the "arduino-dice-footprints" library which should be at the bottom of the list, and click "OK".

2.5.2 Push button switch

The Omron B3F-1000 push botton switch is a through-hole component. Page 225 of its datasheet shows the detailed dimensions of the device (Fig. 18) and can be used to guide the footprint design.

The footprint for the switch consists of 4 1 mm diameter holes spaced 6.5 mm apart in the x-direction and 4.5 mm apart in the y-direction. Use similar steps as in Sec. 2.5.1 to create the switch footprint. Use the following parameters for the through hole pads.

Fig. 19 shows the completed footprint drawing for the switch.



(a)

Dimensions OMRON

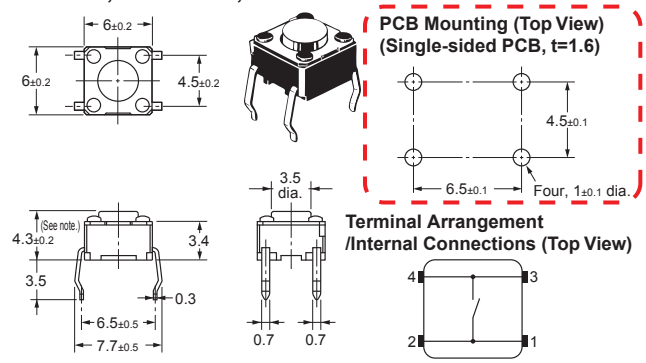
Note: 1. Unless otherwise specified, all units are in millimeters and a tolerance of ±0.4 mm applies to all dimensions.

2. Terminal numbers are not indicated on this switch. With the switch turned over so that the logo mark "OMRON" is visible on the upper part of the rear side of the switch base, the terminal on the right of the logo mark is numbered "1" and that on the bottom right is "3." Accordingly, two terminals on the left side are numbered "2" and "4" respectively.

■ 6 x 6 mm Models

Standard, Flat Plunger Type (without Ground Terminal)

B3F-1000, B3F-1002, B3F-1005, B3F-1020*, B3F-1022*, B3F-1025*, B3F-1002-G, B3F-1022-G*



(b)

Figure 18: (a) Dimension drawing and (b) suggested footprint of the B3F-1000 switch.

Table 5: Parameters for through-hole pads for the B3F switch.

Pad type	Through-hole
Shape	Circular shape
Position X	± 3.25
Position Y	± 2.25
Size X	2.032 (80 mil)
Drill: Size X	1
Layers	All copper layers
Technical Layers	Check "F.SilkS", "F.Mask", and "B.Mask"

2.5.3 ATmega328P microcontroller

The ATmega328P microcontroller powers the popular Arduino UNO platform. There are two variants of the UNO, one with the ATmega328P-PU which is a through hole package and the other with an SMD ATmega328P. In this example, we will be using the through-hole version (ATmega328P-PU) because it allows us to program the microcontroller on the UNO, unplug the microcontroller, and put it on our own PCB.

The ATmega328P-PU follows a standard 0.3" 28-pin dual inline package (28-DIP), i.e. there are two rows of pins with a 0.1" pitch (distance between pins) and 0.3" row spacing. We can follow the same procedures as in Sec. 2.5.1 and 2.5.2 to create its footprint, but this time we'll show you how to do it by modifying an existing footprint from the default KiCad library.

1. In Footprint Editor, click the "Select active library" button. In the pop-up dialog box, select the "Sockets_DIP" library and click "OK".

2. Click the "Load footprint from library". In the pop-up dialog box, click "List all". In the component listing, click "DIP-28_300", and then click "OK". The default 28-DIP package should appear in the Library Editor window.

3. Verify that all the pads have the right dimensions (drill size, pitch and row spacing).

4. Mouse over the string "DIP-28_300" and change it to "atmega328p-pu" so that it's more recognizable when we do the layout later on. Fig. 20 shows completed

5. Now click the "Select active library" icon and select the "arduino-dic-footprints" library. Click the "Save footprint in active library" icon to save the footprint in our project footprint library.

2.5.4 The rest of the components

Similar to the ATmega328P-PU microcontroller, the Avago HDSP-313E 7-segment LED display follows a standard 0.3" 10-DIP pin arrangement. Therefore we can modify a standard 10-DIP footprint (Library: "Sockets_DIP→DIP-10_300") for the 7-segment. Again, always double check all the dimensions if you are basing your work on someone else's footprint. Also notice that the 7-segment display has a much wider body than a typical 10-DIP IC. You will need to modify the courtyard and the silkscreen markings according to the datasheet.

The footprint of the 16 MHz crystal can be created manually from its datasheet. Given the practice we have done so far, it should be relatively straightforward to do.

The 1 uF capacitor and the 10 k resistor are both 0603 components and they will share the same footprint as the 22 pF capacitor. Please DO double check their datasheets to verify that the same footprint can be used!

Fig. 21 shows the footprints for the 7-segment display and the crystal.

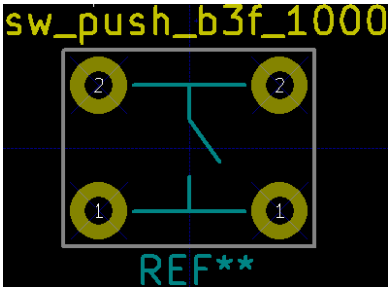


Figure 19: Completed footprint for the Omron B3F-1000 push button switch.

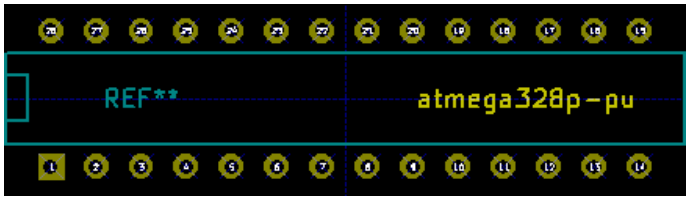


Figure 20: Completed footprint for the ATmega328P-PU microcontroller.

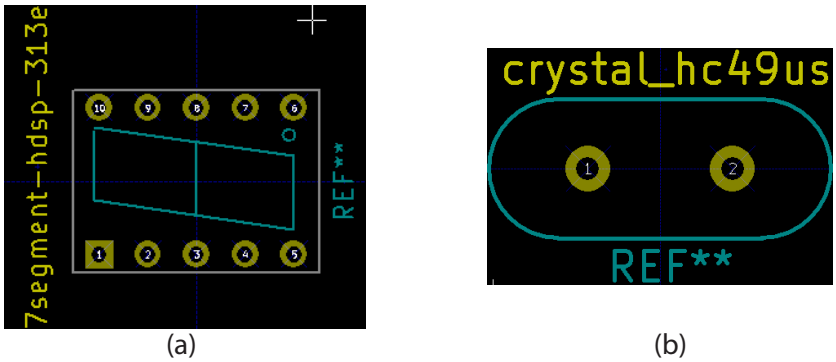


Figure 21: Completed footprints for the (a) HDSP-313E 7-segment LED display and HC-49US 16 MHz crystal.

Lastly, we will use the "Pin_Headers:Pin_Header_Straight_1x02" as the footprint for the 1x2 pin-header connector.

Now having finished building and selecting all the footprints, we need to go back to CvPcb (Sec. 2.4) to make the footprint associations. Fig. 22 shows a screenshot of the completed association. After you are satisfied with the association, click the "Save" button and CvPcb will save the footprint assignment into the "Footprint" field of the schematic symbols. This also means that you can directly add footprint information when you are editing or creating schematic symbols. Nevertheless it is a good idea to run CvPcb to check the association is correctly assigned.

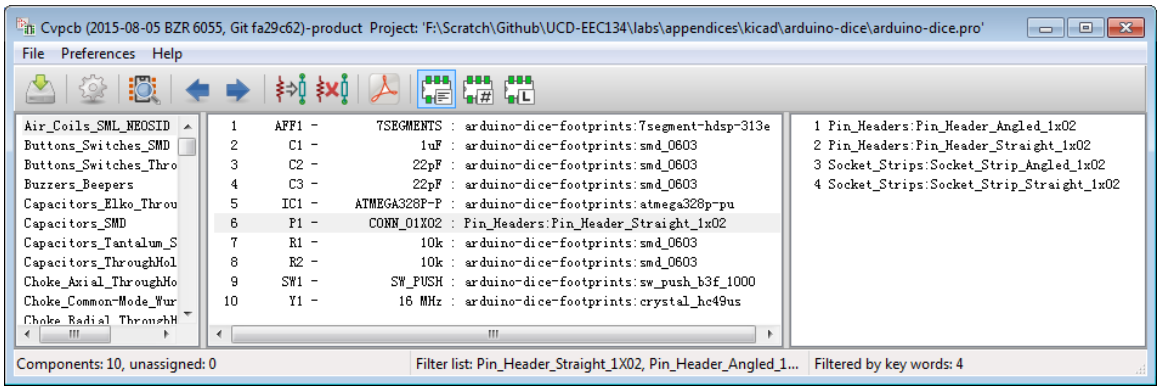


Figure 22: Final schematic symbol and footprint association in CvPcb.

2.6 Laying out the PCB

1. In Eeschema, double check that the newly assigned footprint information appears in the "Footprint" field of the schematic components. Click the "Generate netlist" button to generate a netlist file. Simply use the default file name and location.

2. Start Pcbnew.

3. Click the "Read Netlist" button. The Netlist dialog box will appear.
   a) Click the "Browse netlist files" button, and choose the netlist file generated from Eeschema.
   b) Click the "Read Current Netlist" button. If the schematic and footprint association has been done properly, you should see no error messages. Otherwise, the Netlist dialog box would complain about missing footprint(s).
   c) If there is indeed no error, close the Netlist dialog box.

4. You should now see a bunch of footprints appear in the upper left corner of the Pcbnew main window (Fig. 23). The electrical connections between the components are shown as thin white lines. This initial blob of footprints and lines are often called "rat's nest".

5. Now move the components into the layout area. Do do this, you can:
   a) Right click on a footprint, select the name of the component from the pop-up menu, select "Move" from the secondary menu, move your cursor to the desired location and click to finish the move operation.

      or

   b) Hover the mouse cursor on top of a footprint (oftern called "mouse over"), press the "m" shortcut key, move the cursor to the desire location, and click to finish the move operation.
   c) When a footprint are sitting on top of other footprints, the above operation may become ambiguous because KiCad does not know which footprint you want to select. In such case, a pop-up menu will appear for you select your desired component.
   d) Let's tentatively arrange the components as shown in Fig. 25.

6. For this example, we will layout a two layer board, which is the default setting in KiCad. If you need to route more than two layers, you can go to "Design Rules → Layers Setup". In the dialog box that pops up, you can choose from one of the pre-set layer settings provide by KiCad or create your own. KiCad is able to handle up to 32 layers of copper.

   For the default two-layer board, Table. 6 explains the funtions of each layer. The blue arrow to the left of the layer name indicates the current layer that you are working on. You can also turn the visibility of the layers on and off by checking and un-checking the boxes to the right.

7. Before we start the actual routing the circuit traces, let's set up the layout constraints. These constraints usually represent the manufacturing limitations of the PCB manufacturer. For example, Table. 7 lists typical process parameters from two PCB vendors that we (UC Davis students and researchers) often use.

8. To set up the constraints, go to "Design Rules→Design Rules". A dialog box should appear.
   a) In the "Global Design Rules" tab:
      i. In the "Minimum Allowed Values", set the minimum track width, via drill diameter, and via drill diameter (via diameter + 2×annular ring width) according to Fig. 26a. We won't use micro-via settings in this tutorial so their contraints are arbitrarily set.
      ii. In this dialog box, you can also set up a list of trace widths and via sizes that you can use in the layout. The above figures shows an example of a list of 10 mil, 20 mil, and 40 mil traces. When you do the layout, these trace widths and via diameters can be selected from drop down menus in the top toolbar.
   b) In the "Net Class Editor" tab, set the default net class constraints according to Fig. 26b.

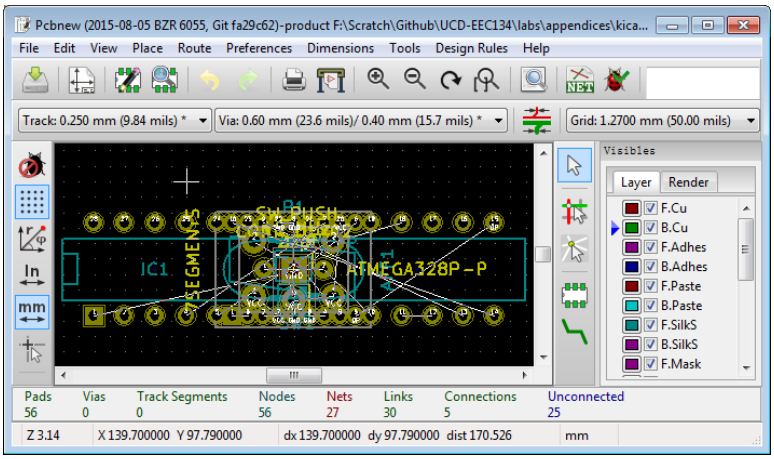


Figure 23: "Rat's nest" of footprints and connections after initial import of the netlist.

Table 6: Layers in Pcbnew.

Layer name	Function
F.Cu	Front side copper trace
B.Cu	Back side copper trace
F.Adhes	Front side adhesives layer. Used to hold SMD components in place during reflow or wave soldering; this layers is used for mass production only.
B.Adhes	Back side adhesives layer.
F.Paste	Opening for front side solder paste. This layer is used to create the solder paste stencils. The solder paste layer openings are usually slightly larger than the pad openings.
B.Paste	Opening for back side solder paste.
F.SilkS	Front side silkscreen printing layer.
B.SilkS	Back side silkscreen printing layer.
F.Mask	Front side solder mask layer.
B.Mask	Back side solder mask layer.
Edge.Cuts	Circuit board outline drawing.

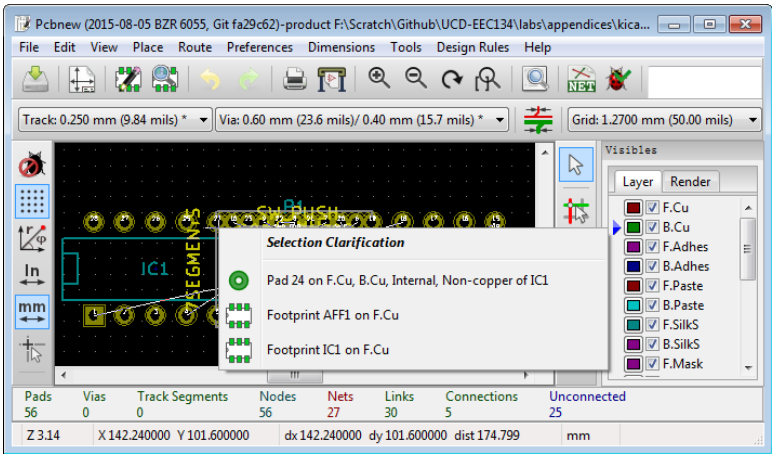


Figure 24: Component selection ambiguity.

9. Now follow the guidance of the rat's nest lines and connect the components together using copper traces. To draw a trace, you can either:

   a) Click the "Add tracks and vias" button , click on the component pin that you want the trace to start from, move the cursor to the pin that you want the trace to end, and double click to complete the trace.

   or

   b) Press the "x" shortcut key to start drawing traces, click on the component pin that you want the trace to start from, then move the cursor to the pin that you want the trace to end, and double click to complete the trace.

10. To add vias, you must start a trace first, then press "PageDown" to add a via to the next layer down, or "PageUp" to add a via to the next layer up.

11. Expect a lot of going back and forth between layout the traces and placing the components.

12. It is often a good practice to fill the PCB with copper to increase the isolation between components and provide easy access to ground.

    a) Click the "Add filled zones" button . The next steps will define a rectangular boundary for the copper fill.

    b) Click on a starting point for the fill zone (i.e. one corner of the rectangle). A dialog box should appear.

    c) We'll fill the top layer first so select the "F.Cu" from the "Layer" list. Select "GND" from the "Net" list; this will assign GND to the filled copper. Leave everything else as default; you can play with the different settings later on your own. Click "OK".

    d) Now draw the outline of the fill area. It can be an arbitrarily shaped polygon, but we'll make it a rectangle here. Double click on the starting point to complete the drawing.

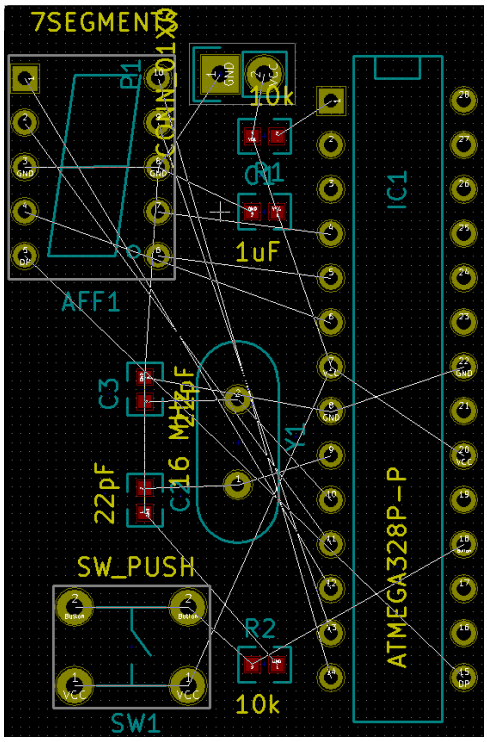


Figure 25: Component arrangement for the Arduino dice PCB.

Table 7: PCB fabrication process parameters for two PCB vendors.

	Bay Area Circuit student special	**OSH Park**
Layers	2	2
Substrate material	FR-4	FR-4
Substrate thickness	62 mil	62 mil
Copper thickness	0.5 oz	1 oz
Minimum Trace Width/ Minimum Spacing	5 mil	6 mil
Minimum Hole Size	15 mil (drill); 5 mil (annular ring width)	13 mil (drill); 7 mil (annular ring width)
Finish	HASL (tinned)	ENIG (thin gold)
Solder mask color	Green	Purple
Silkscreen color	White	White

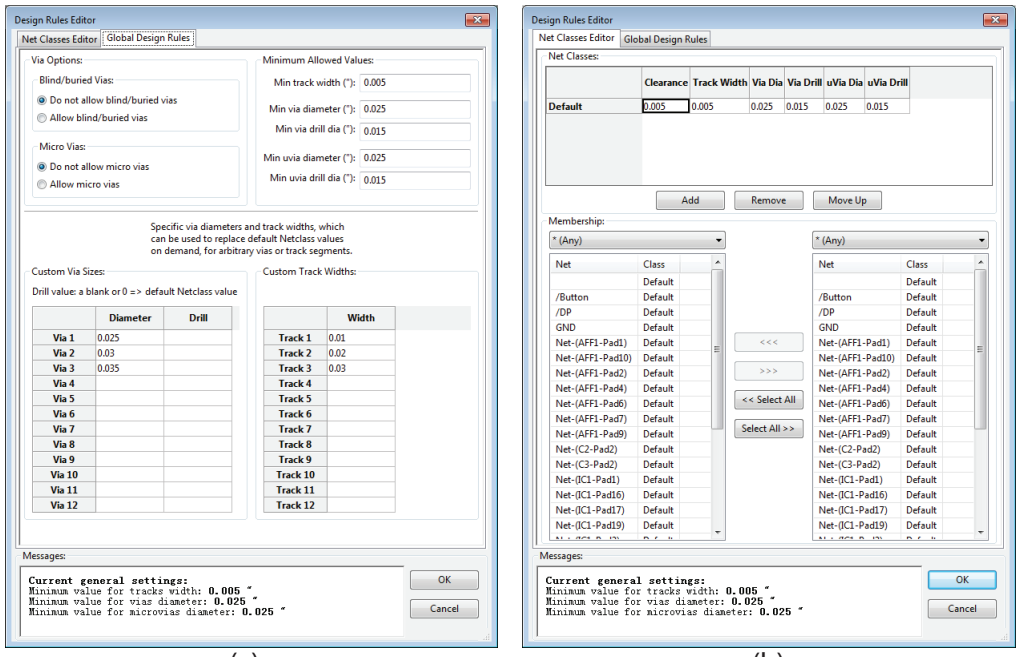


Figure 26: Design rules set up.

e) Within the area that you've drawn above, right click and select "Fill or Refill All Zones" to complete the copper fill.

f) You can repeat the above steps to do a copper fill for the back side of the PCB.

g) **Tip:** When you intend to do a copper fill that is connected to ground, you can actually leave most of the *GND* terminals (nets) unconnected while laying out the circuit. They will be automatically connected to the nearest copper fill, which is connected to ground. Same thing could be said for *VCC* terminals if you intend to do a copper fill connected to *VCC*.

13. In the last step we will generate the outline of the PCB. This is the physical boundary of the PCB. Fig. 27 illustrates an example PCB with a non-rectangular outline.

    To add the board outline:

    a) Select the "Edge.Cuts" layer. In KiCad, the "Edge.Cuts" layer is reserved for the board outline.

    b) It may be helpful to switch to the full-screen cursor mode by toggling the "Change cursor shape" button from the left-side toolbar. In the full-screen mode, the horizontal and vertical lines of the cursor extends all the way across the screen, making it easier to find out the board outline that enclose all your layout.

c) Use the "Add graphic line or polygon" tool from the right-side toolbar to draw the board outline.

14. Fig. 28 shows a layout that I produced in the end.

2.7 Generating Fabrication Files

1. After you are satisfied with the PCB layout, you need to generate the final artwork files for the PCB manufacturer to produce your PCB. The most common artwork file format is the Gerber format.

2. To generate the gerber files, in Pcbnew, go to "Files→Plot". A dialog box should appear. For most PCB houses, the necessary files include: *F.Cu*, *B.Cu*, *F.SilkS*, *B.SilkS*, *F.Mask*, *B.Mask*, and *Edge.Cuts*.

3. You can use the *F.Paste* and *B.Paste* layers to generate solder paste stencils.

4. Click "Plot" to generate the gerber files for the artwork and click "Generate Drill File" to generate the drill file (.drl) for the vias.

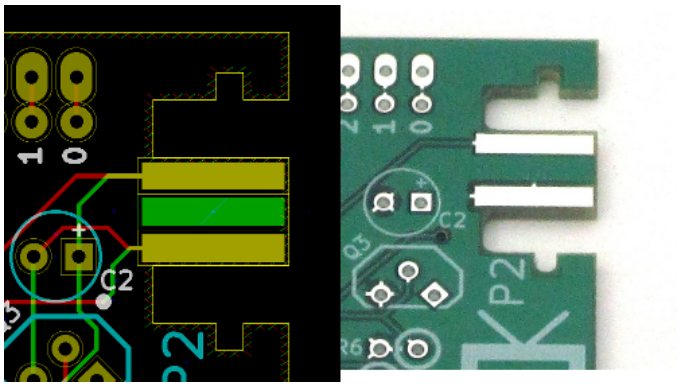


Figure 27: An example of PCB outline. Source: Link.

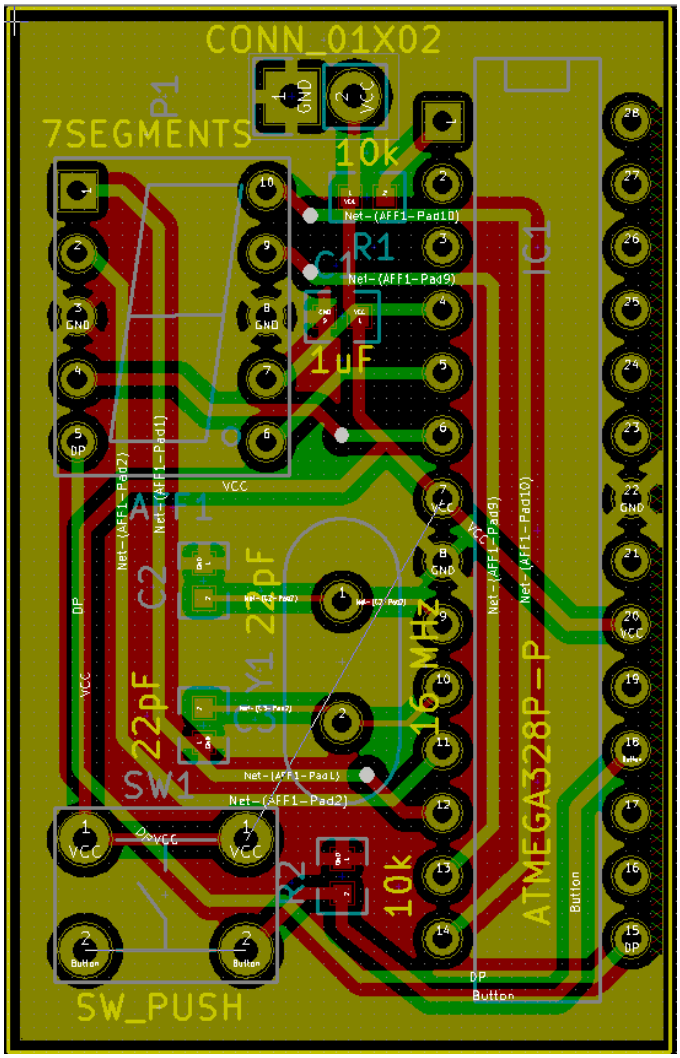


Figure 28: Final layout of the Arduino dice PCB.

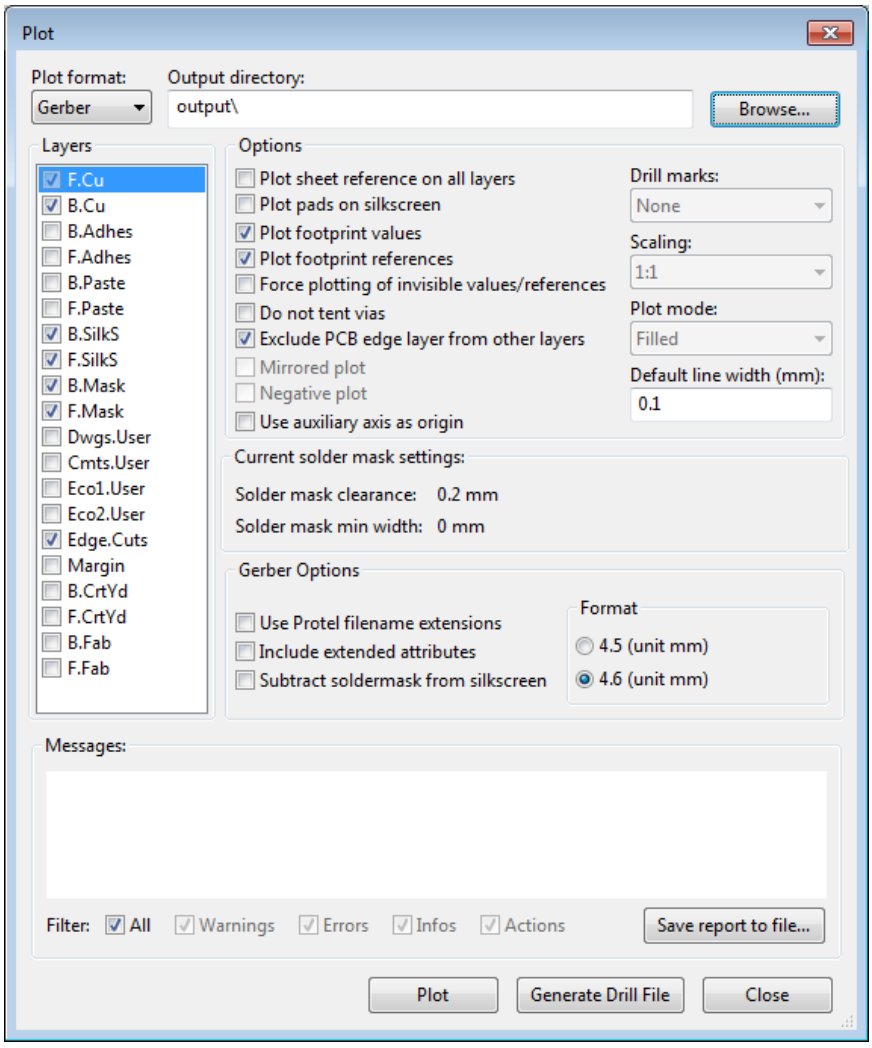


Figure 29: The *Plot* dialog box in Pcbnew.

3 Example 2: RF amplifier

In this example, we will design a test PCB for the Analog Devices ADL5602 RF/IF gain block IC. The ADL5602 has a fixed nominal gain of 20 dB over a fairly wide frequency range of 50 MHz–4 GHz[3]. Both the input and output ports of the ADL5602 are internally matched to 50 Ω, making it easier to use this IC as no external matching circuits are needed.

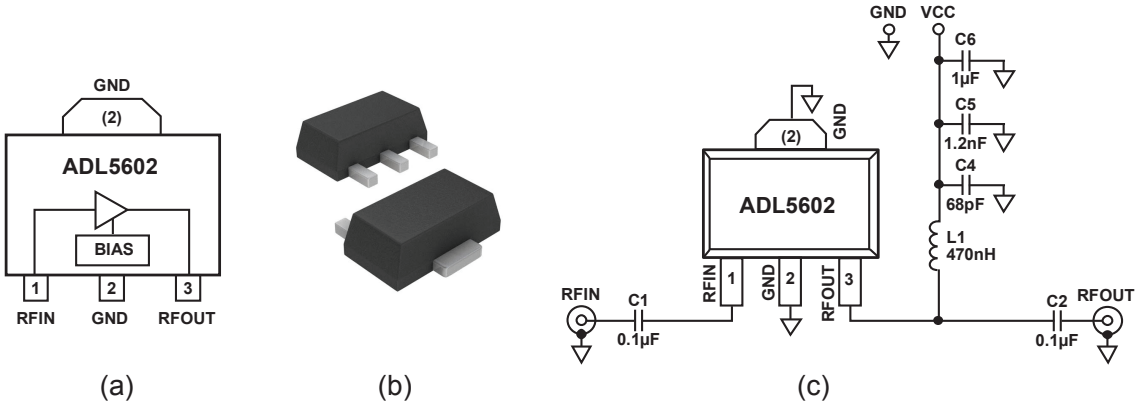


Figure 30: The ADL5602 RF gain block IC. (a) Functional block diagram. (b) Picture of the package. (c) Basic peripheral circuit. Source: ADL56032 datasheet.

RF IC vendors usually provides suggestions on the peripheral circuit schematic and layout of their ICs. In the case of the ADL5602, the vendor does a good job in providing the recommended circuit connections (Fig. 16 of the datasheet), land pattern (Fig. 17), and an example layout (Fig. 19). Note that Fig. 19 is actually the layout for the evaluation board. If you have read enough datasheets, you will realize that an evaluation board is really an example of the recommended layout of the IC's peripheral circuit. *Needlessly to say, when you do your own design, you should follow these vendor recommendations as much as you can, unless you really know what you are doing.*

We will then follow the evaluation board's design and reproduce it in KiCad. A few modifications are needed for the design to work for our PCB vendor. But before we delve into the design details, we need to understand a few key concepts.

3.1 Some Unique Concepts to High Frequency PCB Design

3.1.1 Controlled Impedance Lines

Hopefully by this time you have learned enough about RF design to realize that impedance matching is very important at high frequencies[4] In general, any trace that carries high frequency (> a few hundred MHz) signal should be treated as a

[3] If you look at the datasheet carefully, you can see that the gain does fluctuate within a 2 dB range over the specified frequency range.

[4] If not, please do review Lecture 2b on transmission lines.

transmission line and its *characteristic impedance* must match those of the source and load impedances to minimize loss due to reflections.

In the majority of RF systems, all components are matched to a common system impedance of 50 Ω. Therefore, RF signal traces must have a constant impedance all along. This is why transmission lines are also called *controlled impedance lines* in the PCB design world.

Several forms of transmission lines are amenable for PCB implementation. These include the "microstrip line," the "strip line," and the "coplanar waveguide (CPW)" (Fig. 31). A variation of the CPW adds a ground plane underneath the coplanar signal traces and the grounds, and can be considered as a combination of microstrip line and a CPW (Fig. 31). We call it "CPW over ground (CPWG)" or "conductor backed CPW (CBCPW)".

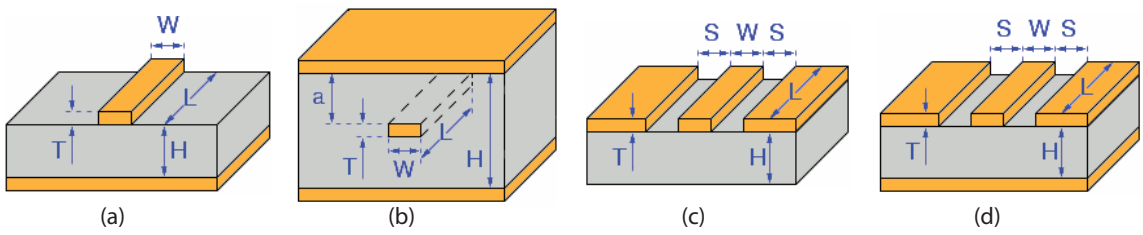


Figure 31: Common PCB transmission line implementations. (a) Microstrip; (b) Stripline; (c) CPW; (d) CPWG. Pictures are taken from KiCad's PCB calculator.

The impedance of the above transmission lines are determined by their cross-section geometry only. Therefore, to maintain a constant impedance, the cross-section geometry of a trace must remain constant. Some PCB vendors publicize their ability to control line impedance accurately; it is basically another way of saying that they can control the pattern accuracy well.

Synthesizing the transmission line geometry for a particular impedance is a subject that has been well studied. In fact there are numerous computer programs that can help you do the job. In this example, we will use an open-source one called *Wcalc*. A similar tool exist under the "TransLine" tab within KiCad's PCB calculator tool.

Microstrip lines are the most commonly used transmission lines on a PCB. The impedance of a microstrip line is primarily determined by its trace width, substrate dielectric constant, and the thickness of the substrate. The metal thickness has a minor effect on the impedance. It is also worth noting that the impedance of a microstrip line varies with frequency. However, narrow-band operation, this variation is relatively small and can often be ignored. For circuits that have instantaneous bandwidth spanning multiple GHz, this frequency dependent impedance variation needs to be investigated.

The loss of the transmission line is primarily determined by the loss tangent of the substrate. For circuits operating above a few GHz, low-loss substrates, such as the Rogers Duroid 5880 and 4350, are usually preferred. These substrates tend to be more expensive, in terms of both procurement and fabrication.

In the ideal case, the RF PCB designer can choose the optimal substrate stack-up to achieve the best high frequency performance. In an less-than-ideal case, other factors may come into play. For low-GHz range, for example, cost usually dominates the selection of substrate and low-cost substrates such as the FR-4 are more common, although they do not have great loss characteristics. In the case of EEC 134 designs, we are bound to using low-cost PCB vendors whose only offerings are on FR-4 substrates with a fixed thickness.

So the only variable left to us is the trace width. Using the specifications of Bay Area Circuits (substrate thickness = 62 mil, substrate dielectric constant = 4.8[5]), we can synthesize the trace width needed for a 50 Ω impedance. It turns out to be 108 mil, or 2.74 mm (Fig. 32). This is quite large! As a comparison, the smallest pin width of the ADL5602 chip is only 0.32–0.52 mm. The separation between the input and output pins is 3 mm, not much bigger than the 2.74 mm. Obviously it would be quite difficult to layout the circuit with such a wide trace.

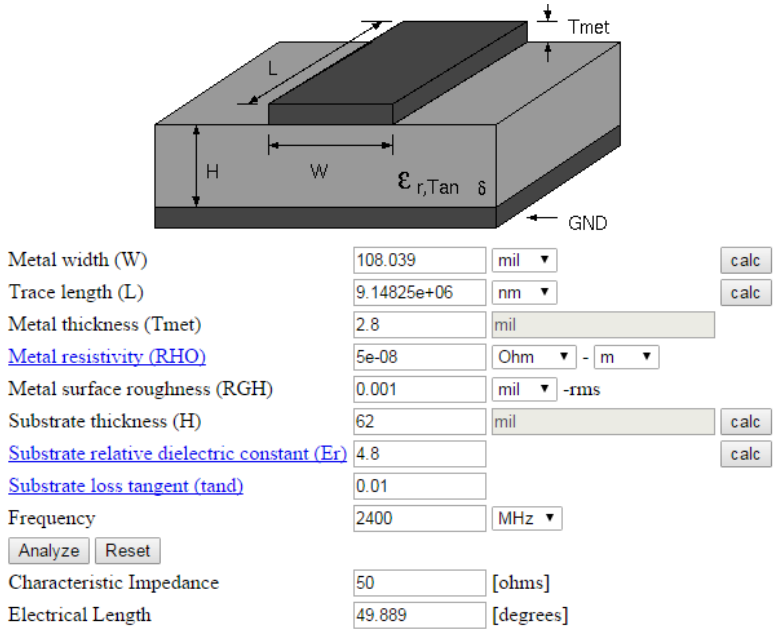


Figure 32: Synthesizing a 50 Ω microstrip line with Wcalc.

One trick we can play to reduce the trace width is to use the CPWG configuration. Compared with the microstrip, the CPWG adds two coplanar ground planes which increases the capacitance per unit length C' of the transmission line. In addition, the separation between the center conductor and the coplanar ground planes can be adjusted to further increase C'. An increased C' means that the center conductor trace width can be reduced.

The minimum trace separation allowed by Bay Area Circuits is 5 mil. If we allow some tolerance on it and use 6 mil as the separation, we could synthesize the required center conductor width using Wcalc. For a 50 Ω transmission line, this width is 37 mil,

[5]Note that these constraints apply only to student special bundles.

or 0.94 mm. Although it's still quite a bit larger than the ADL5602 pin size, it's much better than that of the microstrip case.

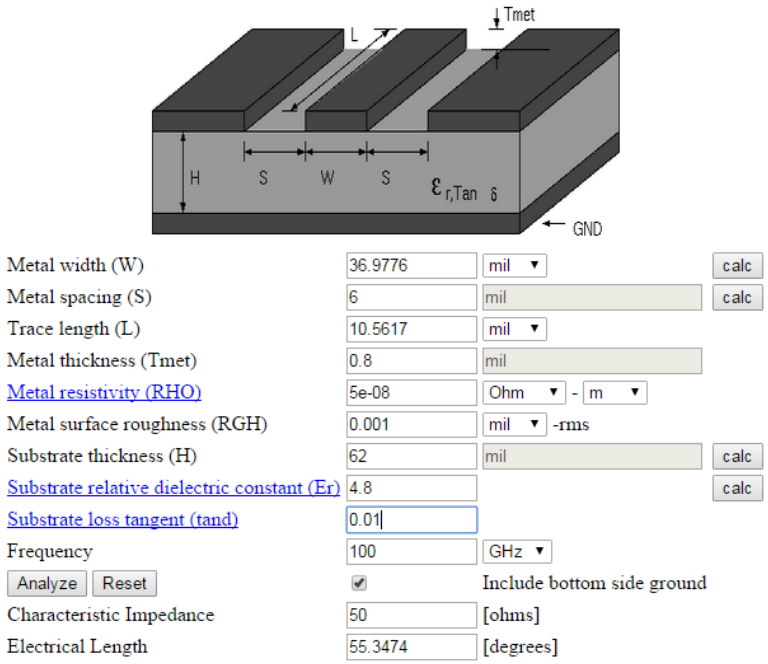


Figure 33: Synthesizing a 50 Ω CPWG line with Wcalc.

To implement a CPWG on a PCB, it is critical to connect the three (two on the top plus the bottom) ground planes together. This is usually done with vias. Fig. 34 shows an example of CPWG design on a PCB. Later in this tutorial, we will show you how to implement it in KiCad.

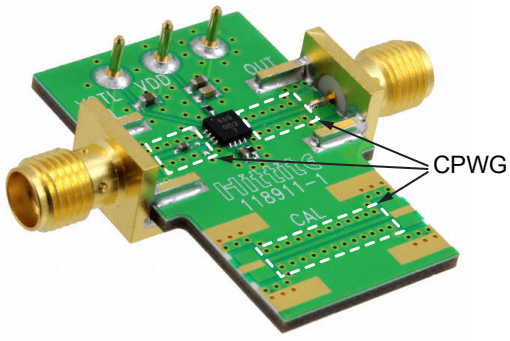


Figure 34: CPWG on a Hittite Microwave evaluation board.

3.1.2 Microstrip lines with Top Copper Shield

One of the many challenges in RF PCB design is to provide good isolation between signals of different frequencies and strength. The contention and interference between signals may come from desired or undesired sources. For example, in RF transceivers, we are often dealing with very weak signals coming into the receiver whereas a transmitting signal several tens of dB stronger may be right close by on the same PCB. Another example exists in many digital processors with high clock speed as they tend to generate ample high frequency (up to a few GHz) noise due the sharp rising and falling edges of the digital signals.

One of the techniques for providing good isolation at high frequencies is to shield the signal traces with ground planes and row(s) of vias. Fig. 35 shows an example PCB employing such a technique. The "via fence" and the top ground plane effectively serve as a cage to contain the RF signal traveling on the microstrip line to prevent it from affecting or being affected by other signals.

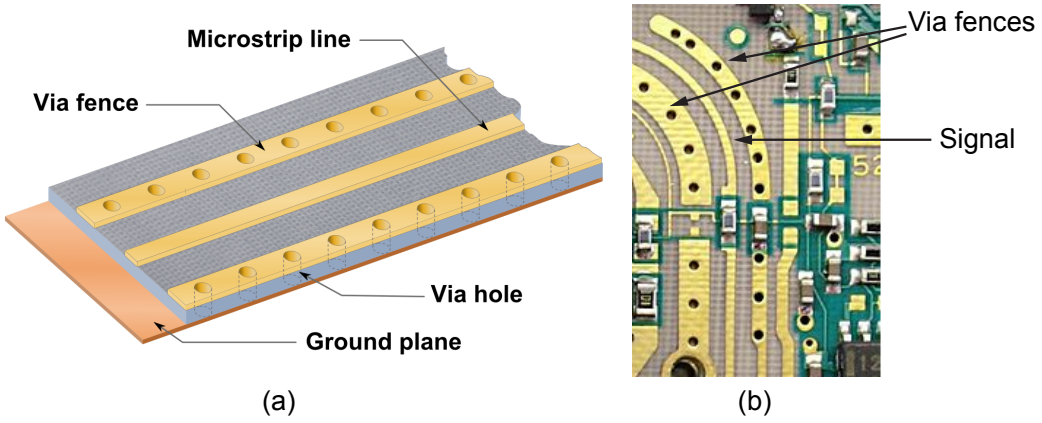


Figure 35: Via fences and top ground plane shields for microstrip lines. Source: Wikipedia.

While Fig. 35 shows islands of shielding structures, many PCB designers opt to fill the entire top layer (signal layer) with ground plane and vias. This is usually called "copper pour" and has been discussed in Example 1.

The acute reader will have noticed that the CPWG on PCB (Fig. 34) inherently has rows of vias to act as shields and in fact it looks rather similar to the microstrip line with via fence shields (Fig. 35). So what exactly is the difference between the two designs?

In principle, there is not much of a difference; both designs are planar transmission lines with ground planes and vias serving as shields. In practice, the biggest distinction is the separation or gap S between the signal trace and the coplanar ground planes. In a CPWG, S is small and contributes significantly to the C' of the transmission line whereas in a shielded microstrip, S is large and its effect on the line impedance is kept small. *A rule of thumb is that S needs to be larger than two or three times the width of the signal trace for a shielded microstrip.* In a CPWG, S must be chosen carefully to get the right impedance.

3.1.3 Transmission Lines Bends

To maintain a nice constant impedance along the transmission line, it is desirable to maintain a constant cross-section geometry. In a complex circuit, however, you may often need to route the high frequency signal to different directions, depending on the orientation and alignment of the circuit components. When the signal trace changes direction, the cross-section inevitably needs to change at the bend, resulting in some impedance perturbation. For a right angle bend of a microstrip line, for example (Fig. 36-a), the corner section effectively has a larger width and it results in an extra shunt capacitance. In addition, due to the skin effect, high frequency signals on a planar transmission line tend to travel on the edges of the trace. As the signal passes through the bend, the current/voltage wave on the outer edge of the bend has to travel a longer distance than the wave on the inner edge. This additional length results effectively results in an extra series inductance. The extra capacitance and inductance creates a low-pass frequency response and limits the bandwidth of the transmission line.

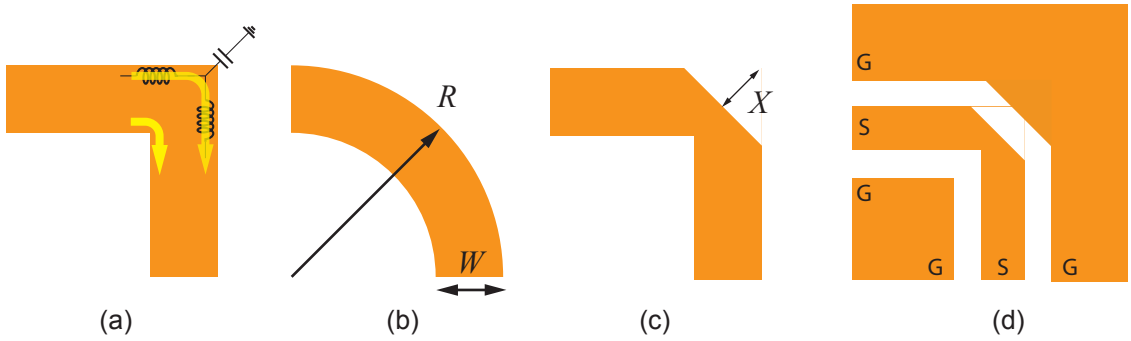


Figure 36: (a) Effects of a right angle bend on the microstrip line characteristics. (b) A microstrip circular bend. (c) A microstrip mitered bend. (d) A CPW mitered bend.

To overcome the adverse effects of a transmission line bend, several bending structures can be employed. A circular bend (Fig. 36-b) smooths out the impedance discontinuity and is often used when board area is not of a major concern. *The rule of thumb here is that the radius R of the bend should be larger than three times the line width W.*

Fig. 36-c&d show another strategy; the corner of the bend is cut off to mitigate the extra capacitance and inductance. This is called a "mitered bend". The optimal design of a mitered bend has been studied thoroughly in the 1970s[6]. An empirical equation for the cut depth X is given by *Douville* and *James*

[6]R.J.P. Douville and D.S. James, Experimental Characterization of Microstrip Bends and Their Frequency Dependent Behavior, 1973 IEEE Conference Digest, October 1973, pp. 24-25.
R.J.P. Douville and D.S. James, Experimental Study of Symmetric Microstrip Bends and Their Compensation, IEEE Transactions on Microwave Theory and Techniques, Vol. MTT-26, March 1978, pp. 175-181.

$$X = \frac{W}{\sqrt{2}} \left\{ 0.52 + 0.65 e^{-1.35 \frac{W}{H}} \right\}$$

where H is the thickness of the substrate. This formula is valid for $0.25 \leq W/H \leq 2.75$.

With the availability of many powerful electromagnetic simulators, it is probably best to simulate the structure to ensure the best performance for a particular PCB stack-up. But it may also be an overkill for non-critical designs. The bottom line is that paying attention to the bend structure is probably better than doing nothing!

3.1.4 Short transmission lines

In previous sections and whatever RF courses you have taken, we have learned that it is very important to maintain good impedance matching on signal traces that carry high frequency signals. New students to the world of high frequency electronics tend to follow the impedance matching principles dogmatically. However, we will see in this subsection that there are certain cases when you don't need to worry about impedance matching too much!

Let's consider a transmission line of characteristic impedance Z_t and length l_t, connected to a load impedance of Z_0. We know that the input impedance looking into the transmission line and the corresponding reflection coefficient are given by

$$Z_{in} = Z_t \frac{Z_0 + jZ_t \tan \beta l_s}{Z_t + jZ_0 \tan \beta l_s},$$

$$\Gamma_{in} = \frac{Z_{in} - Z_0}{Z_{in} + Z_0}.$$

If Z_t is not equal to Z_0, then Z_{in} will generally be complex number that changes with l_s. To get a more intuitive feeling for what Z_{in} looks like, we plot Γ_{in} for various Z_t and l_s values.

It is clear from the graph that as l_s approach 0, Γ_{in} also approaches 0 regardless of the Z_t value. This is hardly surprising because when l_s is extremely small it basically vanishes and the input is connected to the matched load directly. If you remember the teachings from EEC 130A[7], we consider transmission line effects only when electrical length of a line is appreciable compared to a wavelength. For short electrical connections, we treat them as short circuit.

This is why impedance matching is not very important for small l_s. Exactly how small is considered "small" depends on the particular Z_t value and the level of input reflection you can tolerate. In general, it is good practice to keep circuit components as close to each other as possible.

[7] EEC 130A is the introductory electromagnetics course at UC Davis.

3.2 Circuit schematic and component selection

We will create a project named "rf-gain-block".

We will follow the circuit given in Fig. 30-c for this example PCB design. The schematic capture and schematic symbol creation should be straightforward if you have gone through Example 1. Fig. 37 shows an example schematic capture in KiCad.

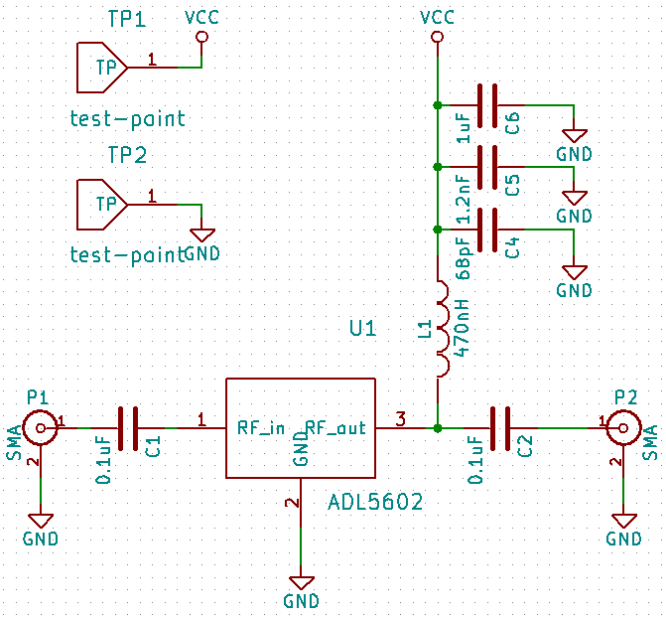


Figure 37: Schematic of the ADL5602 example circuit. Note the component annotation is slightly different those in Fig. 30-c.

The datasheet gives a recommended bill of materials (BOM, Table 6 of the ADL5602 datasheet). It is worth noting that C1 and C2, which are the input and output dc-blocking capacitors, have been sized to 0402 (imperial) to match the width of the input and output 50 Ω transmission line. However, the datasheet recommended circuit is built on a different substrate than what we are going to use. From the calculation above, our 50 Ω line width is about 37 mil, which is closer to the width of a 0603 component. Therefore, in our design, C1 and C2 are going to be size 0603 (imperial). Table. 8 list the modified BOM.

3.2.1 RF-choke inductor

The L1 inductor serves to provide the dc bias current for the amplifier while preventing the RF signal from going into the low-impedance dc source (VCC). It does so by providing a low dc impedance and a high RF impedance, which is consistent with the impedance behavior of an inductor.

In reality, however, all components have parasitics due to their finite physical size and proximity to other components/circuits. These parasitics can be inductive, capacitive, and/or resistive. Fig. 38 shows a simple equivalent circuit for a realistic inductor, where R_s represents the resistive loss in the inductor and C_p represents

Table 8: BOM for the ADL5602 Circuit.

Component	Description	Default Value	Package
C1, C2	dc-block capacitors	0.1 µF	0402
L1	RF-choke inductor	470 nH	0603 (Coilcraft 0603LS-NX or equivalent)
C4	power supply decoupling capacitors	68 pF	0603
C3	power supply decoupling capacitors	1.2 nF	0603
C2	power supply decoupling capacitors	1 µF	1206
VCC, GND	power supply		test pins
P1, P2	RF input and output SMA connectors		SMA

the parasitic capacitance between the input and output terminals of the inductors. Because the impedances of inductors and capacitors are frequency dependent, the frequency response of a realistic inductor exhibit more complex behavior than an ideal inductor.

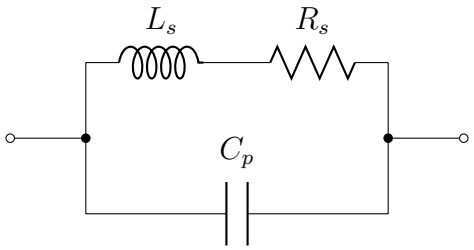


Figure 38: A simple equivalent circuit for a realistic inductor.

For example, if R_s is small, C_p may resonant with L_s to create a parallel resonant tank which present a high impedance. The resonant frequency of C_p and L_s is called the self-resonant frequency (SRF). For an RF-choke inductor, this behavior works to our advantage because it enhances the inductor's capability to reject RF signals around the SRF. However, if the operating frequency is too much higher than SRF, C_p starts shorting out the input and output terminals and the impedance of the inductor starts to drop. As an example, Fig. shows the impedance-frequency profile of the Coilcraft 0603LS-471 ceramic inductor that we are going to use in this example. Although the effective inductance starts to drop around 500 MHz, the impedance reaches a peak around 800 MHz and is above 200 from 100 MHz to above 1.5 GHz.

3.3 Component footprints

Several custom footprints need to be created for this PCB as shown below. All the footprints will be collected in a project footprint library named "rf-gain-block-footprints".

3.3.1 ADL5602 footprint

The footprint of the ADL5602 IC will follow its recommended land pattern in its datasheet. It is copied in Fig. 40 for easy reference. The large exposed pad in the center and the many vias connects serve two purposes: 1) providing a low-impedance electrical path to the ground plane underneath; 2) providing a low thermal impedance path so that the heat generated in the IC could be well dissipated.

Pad 1 and 3 are straightforward to generate in KiCad. Creating of pad 2, the large ground pad, requires a little bit of guidance. Our strategy here is to draw each via as a separate pad with rectangular pad area and a circular drill. The complex land pattern consists of two unit cell types, both labeled in Fig. 40. Pad type 1 has a width of 0.86 mm and a height of $(0.62 + 1.27)/3 = 0.63$ mm. Pad type 2 has a width of 0.762 mm and a height of $(5.37 - 0.62 - 1.27)/6 = 0.58$ mm. The drill diameter for both pad types is 0.20 mm.

1. In the footprint editor, set the unit to "mm".
2. Place pad type 1 at (0, -0.315).
3. Use the array tool to create the rest of the 3 type 1 pads.
   a) Right click on the pad created above, select "Create Pad Array" (shortcut "ctrl+N"). Set up the "Create Array" dialog box according to Fig. 41. Leave the numbering settings to their default values.
   b) Now change the pad numbers of the three pads all to 2. You have to do this one by one. It is a pity that KiCad does not allow automatic numbering of an array with a single number.
4. Create the first type 2 pad at (-0.762,-2.18). Following similar steps as above, create a 3×6 array with pad number 2.
5. The final footprint is shown in Fig. 40-b. A courtyard outline has also been added.

3.3.2 Inductor footprint

The Coilcraft 0603LS series inductors have a recommended footprint in its datasheet (copied here in Fig. 42-a). Fig. 42-b shows the created footprint in KiCad.

3.3.3 SMA connector footprint

The footprint of the SMA connector used in this lab is given in Fig. 43.

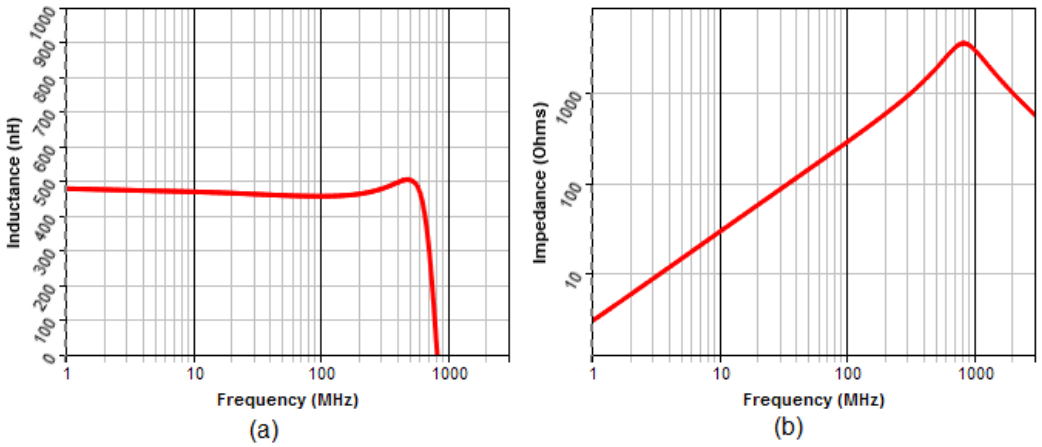


Figure 39: Impedance profile of the Coilcraft 0603LS-471 inductor. Source: Link.

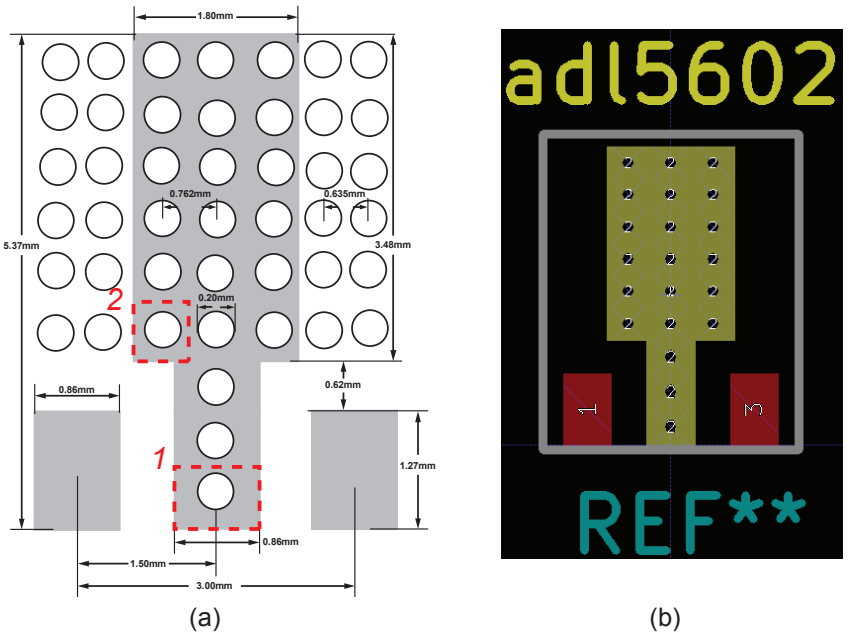


Figure 40: (a) Datasheet recommended land pattern for ADL5602 IC. Source: ADL5602 datasheet. (b) Footprint in KiCad.

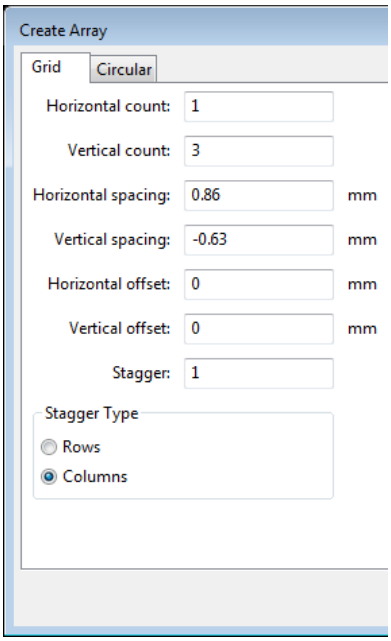
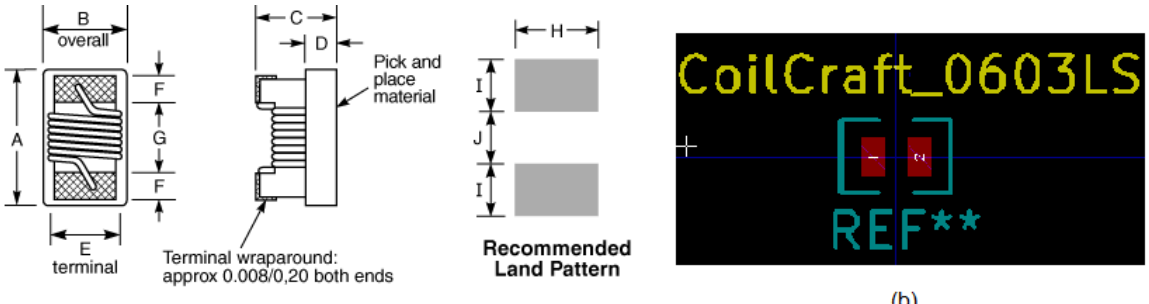


Figure 41: Array setting for pad type 1.



A max	B	C max	D ref	E	F	G	H	I	J	
0.071	See table	0.044	0.015	0.030	0.013	0.034	0.040	0.025	0.025	inches
1,80		1,12	0,38	0,76	0,33	0,86	1,02	0,64	0,64	mm

(a)

Figure 42: Coilcraft 0603LS series footprint.

3.3.4 VCC and GND footprint

The dc power to the circuit is going to be applied through two test points, one for VCC and the other for GND. Fig. 44-a shows a picture of the clip-on test point and its outline drawings. The footprint of this test point is simply a circular pad with a drill diameter of 1.1 mm and a total diameter of 2 mm. The 1.1 mm drill diameter is chosen to be slightly smaller than the widest separation (1.39 mm) between the two legs of the test point so that when the test point is inserted into the hole, mechanical tension could keep it in place for easy soldering.

3.4 PCB Layout

In designing an RF PCB, we usually start with the layout of the RF components and traces. The traces that carry high frequency signals should be kept as short as possible and as straight as possible. In our design, we will align the input and output RF traces in a straight line and connect the input and output pins of the ADL5602 IC in perpendicular to the RF traces. Fig. 45 shows the "ideal" connection to the ADL5602 IC. The input and output traces are tapered to minimize the coupling between the two.

In KiCad, however, all traces have a rounded end shape and tapering cannot be easily implemented. A less-than-ideal approach is to use segments of traces of different width to make the connection. In Fig. 45-b for example, a 20 mil trace segment is used to bridge the pad and the 37 mil trace[8].

In the actual layout, we try to keep the components as close to each other as possible. As such, it may be difficult to fit the 37 mil trace in between the components. As discussed in Section 3.1.4, however, impedance matching is not as important at small length scale and it is OK to connect the components with a thinner trace.

The layout of the rest of the circuit should be straightforward. Fig. 46 shows a preliminary layout. Notice that we have placed the RF components along a straight line and have minimized the distance between the components to minimize the insertion and reflection losses.

3.4.1 Copper fill

At this point, we need to remind ourselves that the 37 mil trace width gives 50 Ω impedance only in a CPWG configuration with a 6 mil gap (Section. 3.1.1). To implement the CPWG, we will use the copper fill feature that we introduced in Example 1.

1. Click the "Add filled zones" button to turn on the copper fill tool.
2. Click at the upper left corner of your intended fill region, a dialog box should appear. Set up the parameters as follows:
   a) Net: GND (This sets the electrical connection of the copper fill to ground.)

[8] The 20 mil trace is invisible because it has the same color as the pad and the 37 mil trace.

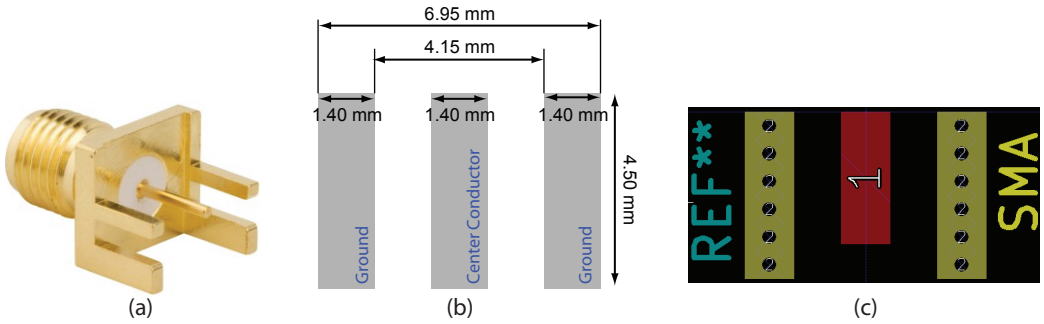


Figure 43: (a) Picture of the SMA end-launch connector. (b) Recommended land pattern. (c) Footprint in KiCad.

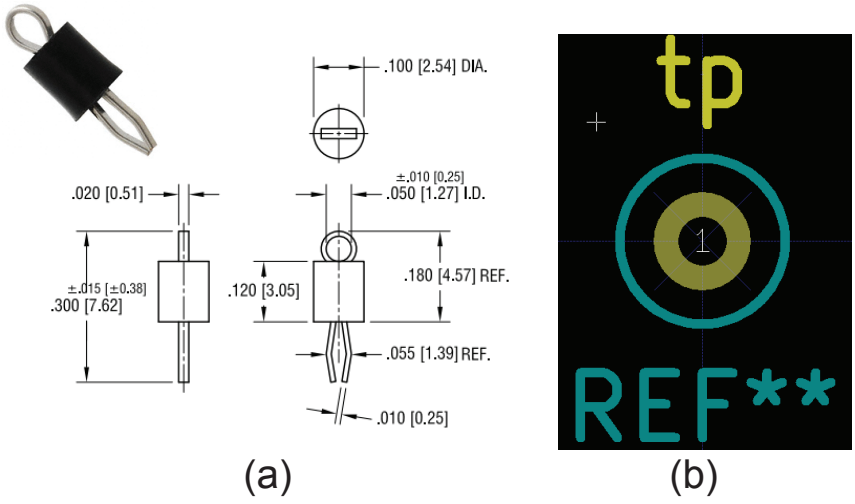


Figure 44: (a) Picture of the test point and its outline drawings. Source: Link. (c) Footprint in KiCad.

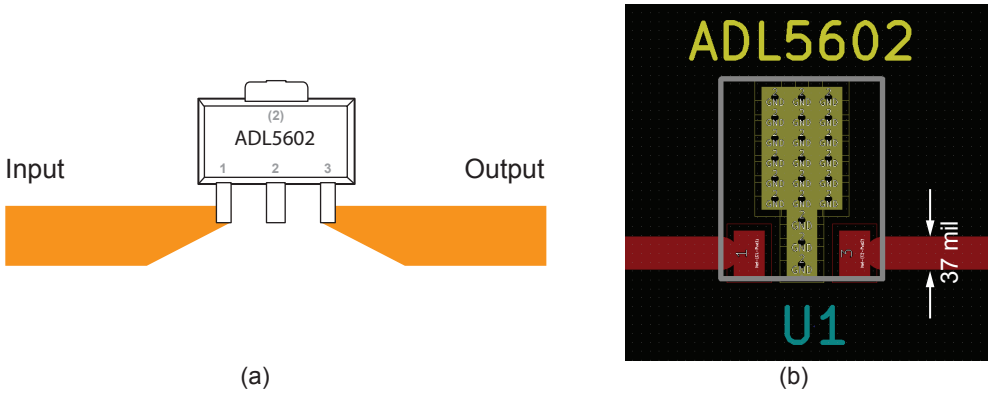


Figure 45: (a) "Ideal" PCB connection to the ADL5602 IC. (b) Implementation in KiCad.

b) Clearance: 0.006" (This sets the separation between the trace and the GND-connected copper fill.)

c) Pad connections: Solid (For RF circuit, we usually don't use thermal relief for pads to minimize the inductance/impedance to ground.)

3. Draw the outline of the fill zone. The fill zone needs to be a closed region. Double click to finish the drawing. You may find it a lot easier to move the cursor around using the keyboard arrow keys. Repeat this for all metal layers that you want to fill. In this example, both the top and the bottom copper layers will be filled.

4. Once the fill zones have been defined, right click on anywhere in the main layout window and select "Fill or Refill All Zones" to create the copper fill. Fig. 47 shows the copper filled layout. Notice that the gap between the trace and the fill is uniform everywhere. This is because we set up only one fill zone. Setting up multiple fill zones will allow you to have multiple separations.

3.4.2 Via fences

The last step (before defining board outline) is to create the via fences that connect the top and bottom ground fills. In PCB terminology, this is often called "via stitching". Unlike more sophisticated PCB tools, such as Cadence Allegro and Altium Designer, KiCad does not have a dedicated tool for via stitching. However, there are several hacks for implementing it. We will follow a procedure proposed in this link.

1. Open the footprint editor. Select "rf-gain-block-footprints" as the active library.

2. Create a new footprint named "via-15mil". Here "15mil" indicate the drill diameter of the via.

3. Insert a pad and set it up as follows:

a) Pad type: Through-hole

b) Shape: circular

c) Size X: 0.030"

d) Drill→Size X: 0.015"

e) Layers→Copper: All copper layers

f) Unselected everything under "Technical layers". By default, "F.SilkS", "F.Mask", and "B.Mask" layers are selected. In particular, unselecting the "F.Mask" and "B.Mask" layers ensures that no solder mask openings are created and that we will have vias covered by the solder mask material in the end[9].

[9] In PCB terminology, such vias are called "tented vias".

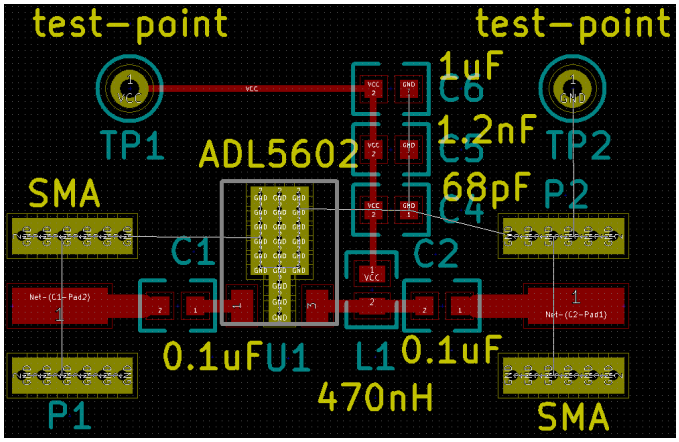


Figure 46: Initial layout of the example 2 circuit.

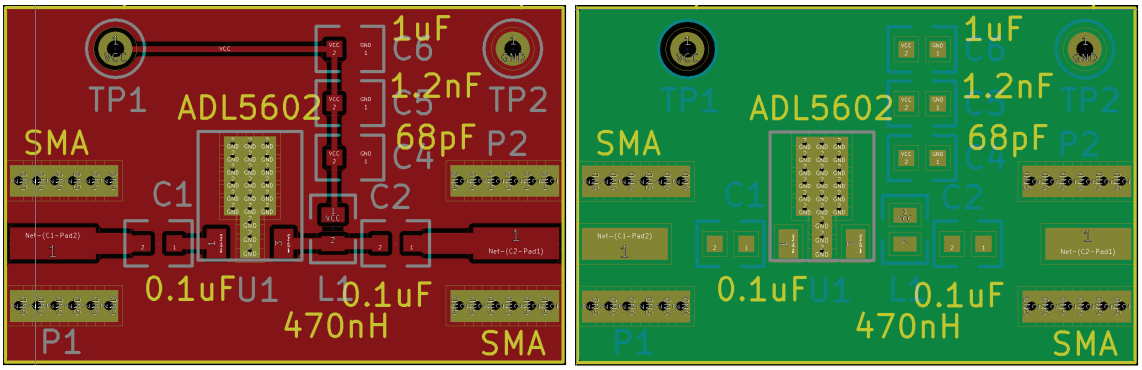


Figure 47: Layout of the example 2 circuit after copper filling.

4. Save the footprint.

5. In the Pcbnew, click the "Add footprints" button and click on the main layout window to insert a footprint directly. Select the "via-15mil" footprint and place it along the edges of the copper fill. For via fences around RF traces, a rule of thumb is to set the via spacing no larger than 1/20th of the propagation wavelength. Obviously you also need to make sure that the via spacing is not too small as to violate the design rules of the PCB manufacturer. Fig. 48 shows the layout after the via stitching.

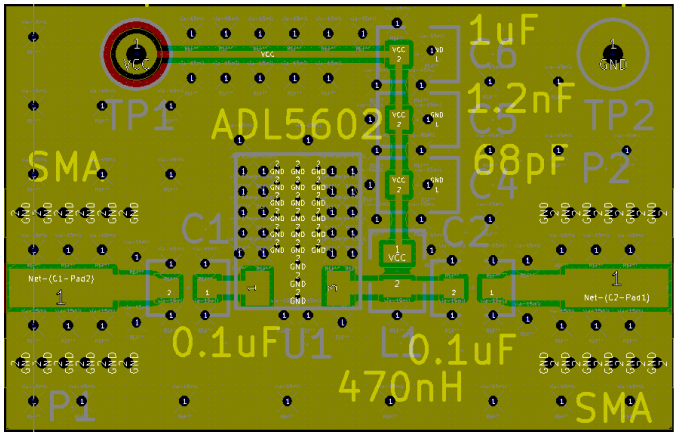


Figure 48: Layout of the example 2 circuit after via stitching.

We have now completed the second example.

4 Further Reading

* Not in any particular order.

- Shields are your friend, except when..., (Part 2), (Part 3).
- KiCAD keyboard PCB design guide
- Online KiCad footprint generator: http://kicad.rohrbacher.net/quickmod.php
- KiCad library management strategies: https://docs.google.com/document/d/1M38ByFyqnhwGo8b_jDDyBceyZtEGeaSAuQaP9REzWrU/edit
- KiCad footprint management (somewhat outdated): http://mithatkonar.com/wiki/doku.php/kicad/footprint_management