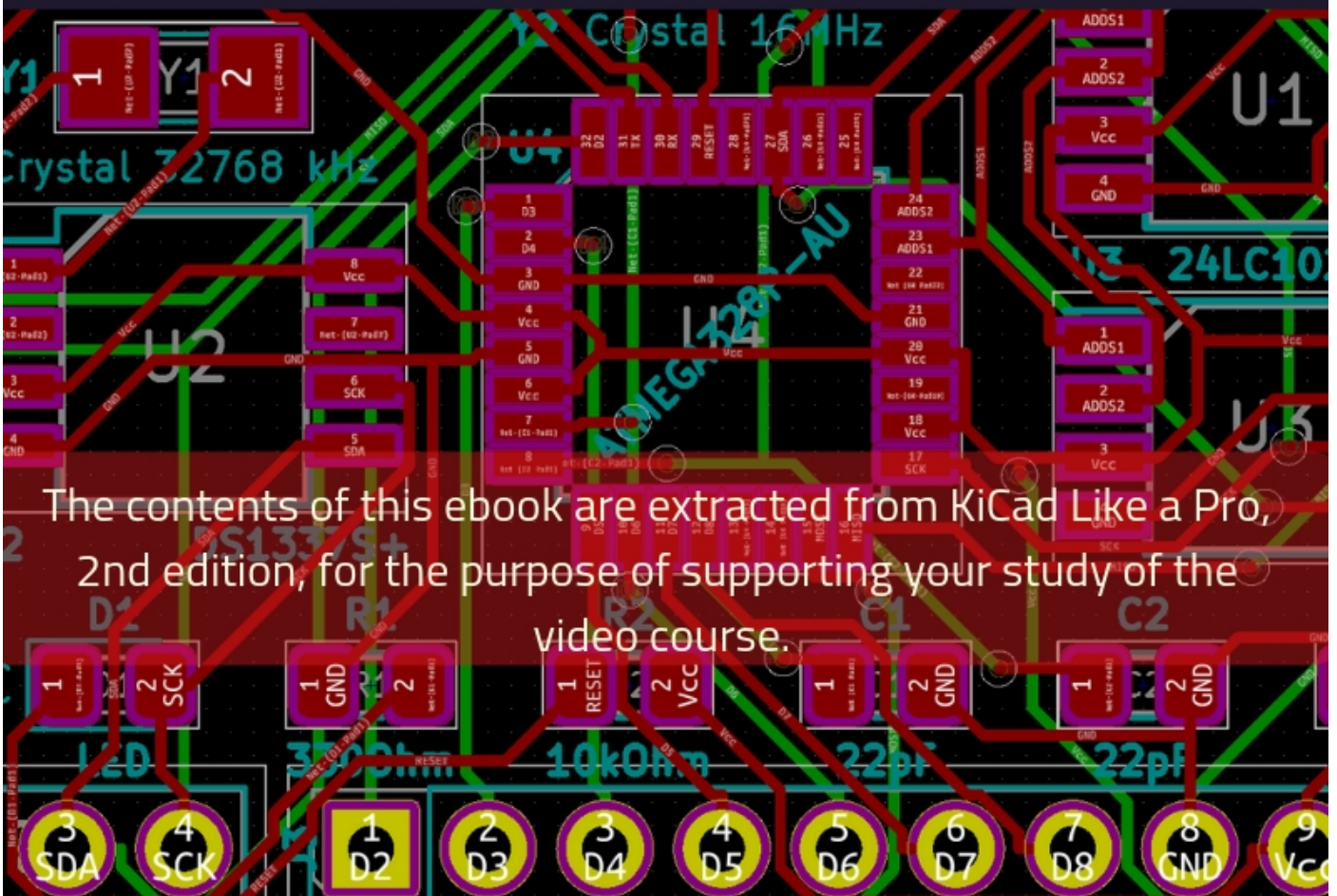


2ND EDITION

KICAD LIKE A PRO

FREE VIDEO COURSE COMPANION

DR PETER DALMARIS



The contents of this ebook are extracted from KiCad Like a Pro, 2nd edition, for the purpose of supporting your study of the video course.

KICAD LIKE A PRO

Dr Peter Dalmaris

KiCad Like a Pro, 2nd Edition

By Dr Peter Dalmaris

Copyright © 2018 by Tech Explorations™

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review.

Printed in Australia

First Printing, 2018

ISBN (PDF) : 978-1-64440-886-5

ISBN (epub): 978-1-64440-885-8

ISBN (mobi): 978-1-64467-528-1

Tech Explorations Publishing

PO Box 22, Berowra 2081 NSW

Australia

www.techexplorations.com

Cover designer: Michelle Dalmaris

Disclaimer

The material in this publication is of the nature of general comment only, and does not represent professional advice. It is not intended to provide specific guidance for particular circumstances and it should not be relied on as the basis for any decision to take action or not take action on any matter which it covers. Readers should obtain professional advice where appropriate, before making any such decision. To the maximum extent permitted by law, the author and publisher disclaim all responsibility and liability to any person, arising directly or indirectly from any person taking or not taking action based on the information in this publication.

Version 1.42

Did you find an error?

Please let us know.

Using any web browser, go to txplo.re/KiCadbook, and fill in the form.

We'll get it fixed right away.

About the author

Dr. Peter Dalmaris is an educator, an electrical engineer, electronics hobbyist, and Maker. Creator of online video courses on DIY electronics and author of several technical books. Peter has recently released his book 'Maker Education Revolution', a book about how Making is changing the way we learn and teach in the 21st century.

As a Chief Tech Explorer since 2013 at Tech Explorations, the company he founded in Sydney, Australia, Peter's mission is to explore technology and help educate the world.

Tech Explorations offers educational courses and Bootcamps for electronics hobbyists, STEM students, and STEM teachers.

A lifelong learner, Peter's core skill lies in explaining difficult concepts through video and text. With over 15 years of tertiary teaching experience, Peter has developed a simple yet comprehensive style in teaching that students from all around the world appreciate.

His passion for technology and the world of DIY open-source hardware, has been a dominant driver that has guided his personal development and his work through Tech Explorations.

Tech Explorations creates educational products for students and hobbyists of electronics who rather utilize their time making awesome gadgets instead of searching endlessly through blog posts and Youtube videos.

We deliver high-quality instructional videos and books through our online learning platform, txplore.com.

Supporting our students through their learning journey is our priority, and we do this through our dedicated online community and course forums.

Founded in 2013 by Peter Dalmaris, Tech Explorations was created after Peter realised how difficult it was to find high-quality definitive guides for the Arduino, written or produced by creators who responded to their reader questions.

Peter was frustrated having to search for Youtube videos and blog articles that almost never seemed to be made for the purpose of conveying knowledge.

He decided to create Teach Explorations so that he could produce the educational content that he wished he could find back then.

Tech Explorations courses are designed to be comprehensive, definitive and practical. Whether it is through video, ebook, blog or email, our delivery is personal and conversational.

It is like having a friend showing you something neat... the "AHA" moments just flow!

Peter left his career in Academia after his passion for electronics and making was rekindled with the arrival of his first Arduino. Although he was an electronics hobbyist from a young age, something led him to study electrical and electronics engineering in University, the Arduino signalled a revolution in the way that electronics is taught and learned.

Peter decided to be a part of this revolution and has never looked back.

We know that even today, with all the information of the world at your fingertips, thanks to Google, and all the components of the world one click away, thanks to eBay, the life of the electronics hobbyist is not easy.

Busy lifestyles leave little time for your hobby, and you want this time to count.

We want to help you to enjoy your hobby. We want you to enjoy learning amazing practical things that you can use to make your own awesome gadgets.

Electronics is a rewarding hobby. Science, engineering, mathematics, art, and curiosity all converge in a tiny circuit with a handful of components.

We want to help you take this journey without delays and frustrations.

Our courses have been used by over 70,000 people across the world.

From prototyping electronics with the Arduino to learning full-stack development with the Raspberry Pi or designing professional-looking printed circuit boards for their awesome gadgets, our students enjoyed taking our courses and improved their making skills dramatically.

Here's what some of them had to say:

"I'm about half way through this course and I am learning so much. Peter is an outstanding instructor. I recommend this course if you really want to learn about the versatility of the amazing Raspberry Pi" -- Scott

"The objectives of this course are uniquely defined and very useful. The instructor explains the material very clearly." -- Huan

"Logical for the beginner. Many things that I did not know so far about Arduino but easy to understand. Also the voice is easy to understand which is unlike many courses about microcontrollers that I have STARTED in the past. Thanks" -- Anthony

Please check out our courses at techexplorations.com and let us be part of your tech adventures.

From the back cover

Printed circuit boards (PCB) are, perhaps, the most undervalued component of modern electronics. Usually made of fibreglass, PCBs are responsible for holding in place and interconnecting the various components that make virtually all electronic devices work.

The design of complex printed circuit boards was something that only skilled engineers could do. These engineers used to expensive computer-aided design tools. The boards they designed were manufactured in exclusive manufacturing facilities in large numbers.

Not anymore.

During the last 20 years, we have seen high-end engineering capabilities becoming available to virtually anyone that wants them. Computer-aided design tools and manufacturing facilities for PCBs are one mouse click away.

KiCad is one of those tools. Perhaps the world's most popular (and best) computer-aided design tool for making printed circuit boards, KiCad is open source, fully featured, well-funded and supported, well documented. It is the perfect tool for electronics engineers and hobbyists alike, used to create amazing PCBs. KiCad has reached maturity and is now a fully featured and stable choice for anyone that needs to design custom PCBs.

This book will teach you to use KiCad. Whether you are a hobbyist or an electronics engineer, this book will help you become productive quickly, and start designing your own boards.

Are you a hobbyist? Is the breadboard a bottleneck in your projects? Do you want to become skilled in circuit board design? If yes, then KiCad and this book are a perfect choice. Use KiCad to design custom boards for your projects. Don't leave your projects on the breadboard, gathering dust and falling apart.

Complete your prototyping process with a beautiful PCB and give your projects a high-quality, professional look.

Are you an electronics engineer? Perhaps you already use a CAD tool for PCB design. Are you interested in learning KiCad and experience the power and freedom of open-source software? If yes, then this book will help you become productive with KiCad very quickly. You can build on your existing PCB design knowledge and learn KiCad through hands-on projects.

This book takes a practical approach to learning. It consists of four projects of incremental difficulty and recipes.

The projects will teach you basic and advanced features of KiCad. If you

have absolutely no prior knowledge of PCB design, you will find that the introductory project will teach you the very basics. You can then continue with the rest of the projects. You will design a board for a breadboard power supply, a tiny Raspberry Pi HAT, and an Arduino clone with extended memory and clock integrated circuits.

The book includes a variety of recipes for frequently used activities. You can use this part as a quick reference at any time.

The book is supported by the author via a page that provides access to additional resources. Signup to receive assistance and updates.

How to read this book

I designed this book to be used both to learn how to use KiCad, and as a reference.

If you have never used KiCad and have little or no experience in PCB design, I recommend you read it in a linear fashion. Don't skip the early chapters in parts 1, 2 and 3, because those will set the fundamental knowledge on which you will build your skill later in the book. If you skip those chapters, you will have gaps in your knowledge that will make it harder for you to progress.

If you have a good working knowledge of PCB design, but you are new to KiCad, you can go straight to Part 2, zoom through it very quickly, and then proceed to the projects in Part 4.

Once you have the basic KiCad concepts and skills confidently learned, you can use the recipes in Part 5 as a resource for specific problems you need solved. These recipes are useful on their own. Throughout the text, you will also find prompts to go to a particular recipe in order to learn a specific skill needed for the projects.

Images: Throughout this book, you will find numerous figures that contain screenshots of KiCad. To create these screenshots, I used KiCad 5 running on Linux Ubuntu. If you are using KiCad under Windows or Mac OS, do not worry: KiCad works the same across these platforms, and even looks almost the same.

Although I took care to produce images that are clear, there are cases where this was not possible. This is particularly true in screenshots of an entire application window, meant to be displayed in a large screen. The role of these images is to help you follow the instructions in the book as you are working on your computer. There is no substitute to experimenting and learning by doing, so the best advice I can give is to use this book as a text book and companion. Whenever you read it, have KiCad open on your computer and follow along with the instructions.

This book has a web page with resources designed to maximise the value it delivers to you, the reader. Please read about the book web page, what it offers and how to access it in the section 'The book web page', later in this introductory segment.

Finally, you may be interested in the video course version of this book. This course spans over 17 hours of high-definition video, with detailed explanations and demonstrations of all projects featured in the book. The

video lectures capture techniques and procedures that are just not possible to do so in text.

Please check in the book web page for updates on this project. Be sure to subscribe to the Tech Explorations email list so I can send you updates.

Requirements

To make the most out of this book, you will need a few things. You probably already have them:

- A computer running Windows, Mac OS or Linux.
- Access to the Internet.
- A mouse with at least two buttons and a scroll wheel. I use a Logitech MX Master 2S mouse (see <https://amzn.to/2ClySq0>).
- Ability to install software.
- Time to work on the book, and patience.

The book web page

As a reader of this book, you are entitled access to its online resources.

You can access these resources by visiting the book's web page at <http://txplo.re/klpr>.

The two available resources are:

1. **The book forum.** This is a place where you can ask book-related questions and have a conversation about your projects. I will be spending time in the forum weekly, answering questions and participating in discussions.

2. **An errata page.** As I correct bugs, I will be posting information about these corrections in this page. Please check this page if you suspect that you have found an error. If you an error you have found is not listed in the errata page, please use the error report form in the same page to let me know about it.

3. **Chapters that didn't fit in the book.** I was not able to fit everything I wanted in this book. Deadlines and other constraints result in decisions about what can go in and what can't. I plan to post those chapters that didn't make it in the book, online.

From time to time, I will be posting additional KiCad resources and updates in that page, so please check regularly. By subscribing to the Tech Explorations email list, your'll be sure to receive my regular book updates and news. The subscription form is in the book page.

Part 1: A quick introduction to PCB design

Part 2: A hands-on tour of KiCad with a very simple project

Part 3: Design principles and basic concepts

Part 4: Projects

16. About this Part

In this part, you will design three printed circuit boards on KiCad. Each circuit board project will give you the opportunity to consolidate and expand on your knowledge. You will become proficient in using KiCad features. You will also gain experience in schematic design and layout.

The topic of the first project is a breadboard power supply. This project will teach you how to design a board with specific mechanical constraints, on top of walking through the full design process.

The topic of the second project is a tiny HAT for the Raspberry Pi. In this project, you will learn how to use third-party board layouts and customised them to your needs. Board layouts can save you a lot of time since you do not need to take your own accurate measurements.

The topic of the third is to design an Arduino Uno clone, with features useful for building mobile, battery-powered gadgets. In this project, you will learn, among many things, how to shrink the size of your PCBs using SMD components, how to use the autorouter, and how to use hierarchical sheets.

Let's begin!

In this first project, you will design the PCB for a 5V power supply for your projects. This power supply will use the LM7805 voltage regulator, a bridge rectifier composed of four 1N4007 (or 1N4004) rectifier diodes, a couple of smoothing capacitors, and a couple of indicator LEDs with their current-limiting resistors.

To draw power from mains safely, you will need a walled power supply from an appliance that you don't need anymore, that is capable of outputting AC or DC between 7V to 24V.

Here are the parts you will need:

1. One walled DC or AC power supply capable of outputting between 7V and 25V.
2. One LM7805 voltage regulator IC.
3. Four 1N4007 rectifier diodes.
4. One 470 μF capacitor, with a working voltage of at least 25V.
5. One 47 μF capacitor, also with a working voltage of at least 25V.
6. One red and one yellow LED
7. Two 330 Ω resistors to protect the LEDs

The exact capacitance values of the capacitors are not important. Larger capacitors will produce a smoother voltage in the output. What is important, however, is their working voltage. If you use a capacitor with a working voltage of 25V, be careful to not use a mains power supply that provides an output voltage of more than 25V, as this will exceed the maximum rated voltage of the capacitor. In Figure 17.2, you can see the parts that we'll be using in this project.



Figure 17.2 The components that we'll use in our power supply.

In regards to the schematic, our power supply is depicted in Figure 17.3:

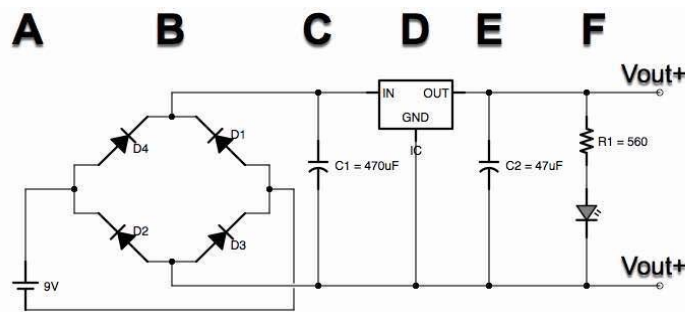


Figure 17.3: The power supply schematic diagram.

In the schematic, you can see the stages that lead to a stable 5V output voltage.

In stage A, we connect a step-down transformer mains power supply. In the schematic, I have marked it a 9V DC, but it can be any power supply that produces AC or DC between 7 and 25V.

The voltage from this mains power supply is taken through stage B, the rectifier stage. If the input is an AC waveform voltage, the output of the rectifier will be a DC waveform voltage, always positive. If the input is a DC voltage, then the output will be the same DC voltage.

To learn about rectifiers and how they work to produce DC voltage from an AC input, please read the [relevant article](#) on Wikipedia¹.

In the next stage, C, the voltage is further stabilised. The capacitor stores energy as the voltage rises and releases it as the voltage drops, and in effect, it 'bridges' the gaps in the waveform.

In stage D, we use the voltage regulator LM7805 to produce the 5V regulated voltage that we aim for. At this point, we can go ahead and connect the circuit that we want to power. However, we can use an additional smaller capacitor to smoothen out this voltage even further, in stage E.

Stage F is where we can connect an indicator LED that can tell us when the power supply is operating.

Figure 17.4 shows the assembled circuit on a breadboard. You can try this yourself and confirm that it is operating as expected.

¹ <https://en.wikipedia.org/wiki/Rectifier>, visited December 11, 2018.

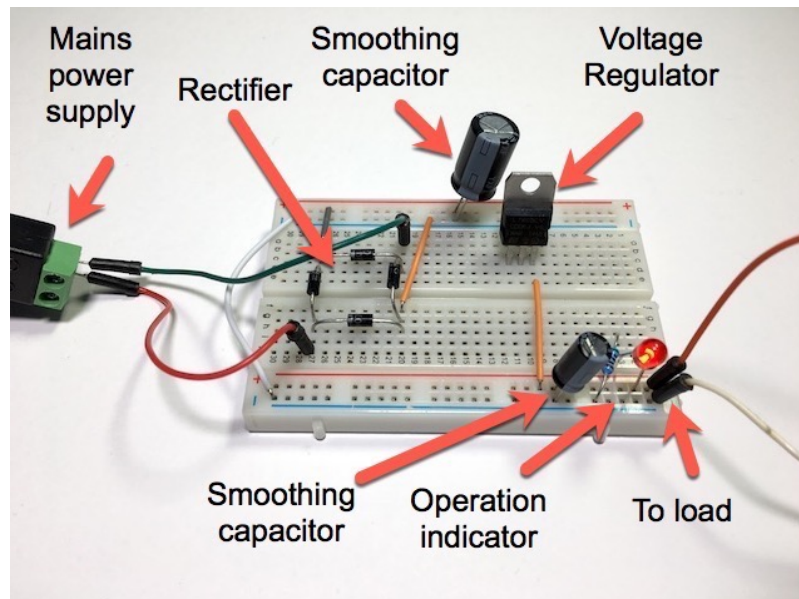


Figure 17.4: The power supply implemented on a breadboard for testing.

In my testing, I supplied the input with 9V DC using a spare mains power supply and tested the output. I used my multimeter to get the actual values of the voltage from the mains power supply, and the output from my breadboard power supply.



Figure 17.5: The actual voltage of the 9 V DC mains power supply I use to power the breadboard power supply.

In Figure 17.5 you see that the actual voltage of my 9 V mains power supply is slightly higher than what is advertised on the label. This is often done by design, to compensate from the voltage drop that occurs when a load is connected. In this measurement, I had only connected an LED as a load.



Figure 17.6: Actual output voltage, when input power comes from a 9 V DC mains power supply.

In Figure 17.6 you can see the actual output voltage of the circuit when I use my 9 V DC mains power supply. It is very close to 5 V. In another test, I applied a 16 V DC input voltage from my bench power supply to the breadboard circuit and measured the output voltage at 5.010 V. At 20 V input, the output voltage was 5.013 V. Based on these tests, we can be confident that this breadboard power supply works as expected.

Now that you have a good overview of the objective of this project, let's turn our attention back to KiCad.

What you will learn

In this project, you will use all of KiCad's main features that you learned about in Parts 2 and 3 of the book. As with the rest of the projects in this book, we will follow the design process outlined in Figure 17.7. You should download your copy of the graphic from the [book's](#) page,² print it, and use it as a quick reference guide.

To keep things simple, the process you will follow assumes that all of the components we need for designing this PCB are available in KiCad's

² The book page is at <http://txplo.re/klpp>. Valid as of December 11, 2018.

standard libraries³ and that we are not interested in implementing any 'fancy' design and aesthetic features. You will have the opportunity to do both in later projects.

Nevertheless, you will learn a great deal. Here's a list of your learning objectives for this first project:

1. Learn how to design a schematic using Eeschema, KiCad's schematic editor.
2. Learn how to do an electrical test and confirm that the schematic is correct.
3. Learn how to associate the components in Eeschema with footprints, using KiCad's Cvpcb, KiCad's associations editor.
4. Learn how to create a file that contains the information that we need in order to do the PCB layout in Pcbnew. Pcbnew is KiCad's layout editor. The file that contains the information is known as the 'netlist'.
5. Learn how to use Pcbnew to design your PCB, place the components on it, and connect them by creating routes.
6. Learn how to prepare the PCB for manufacturing, including testing for defects.
7. Learn how to upload the files that contain the PCB design data to an online manufacturer.

While this process may feel like a lot of work, over time, it becomes just part of your prototyping process. Personally, designing a PCB is work that I enjoy immensely. Creating a PCB involves engineering and design/aesthetic elements.

Creating a PCB is an opportunity to not only create something useful, but also an object of art. Once all your electrical tests are done, once you have placed and routed all the components on the board, you will find yourself obsessed with small details, like the exact position of a silk screen label, or the size of a contour that defines the shape of a corner.

But to get to the art, you must first take the first step.

Let's do that now with the schematic design in Eeschema.

Project repository

This project has a Git repository. You can access it at txplo.re/klpp1.

³ A library is a collection of components (the symbols that we will use to create the PCB schematic), and footprints (which we will use to design the layout of the PCB). KiCad comes with a lot of very useful standard libraries, but we can also import and use libraries created and shared by KiCad users.

This repository contains multiple commits at each step of the design process, as I was going through it. Feel free to clone this repository on your computer and make any modifications you feel like.

18. Project 2: Design a small Raspberry Pi HAT

18.1. What you will built and list of parts

My motivation for creating this simple Raspberry Pi HAT, is to make it easier for students of my course Raspberry Pi Full Stack to connect the course hardware to their Raspberry Pi. Using this HAT, students will be able to simply snap-on the board, and continue with their study, instead of spending time manually assembling the components on a breadboard.

In this project, you will create a tiny board for the Raspberry Pi. The board will contain four SMD components (three resistors and an LED) to help in reducing its size, a button, and a DHT22 sensor. We'll use a standard 2.54 mm pitch header with two rows to connect the board to the Raspberry Pi. To secure the board on the Raspberry Pi, we'll also include a mounting hole that matches the one that is located on the Raspberry Pi board (Figure 18.3).

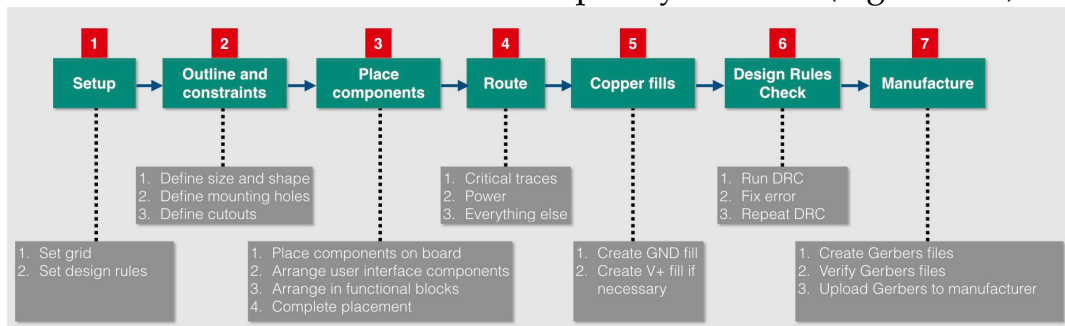


Figure 18.1: A serialised view of the PCB layout process.

An interesting aspect of this project is that instead of following precisely the layout process depicted in Figure 18.2, you will iterate between steps 2 and 3.

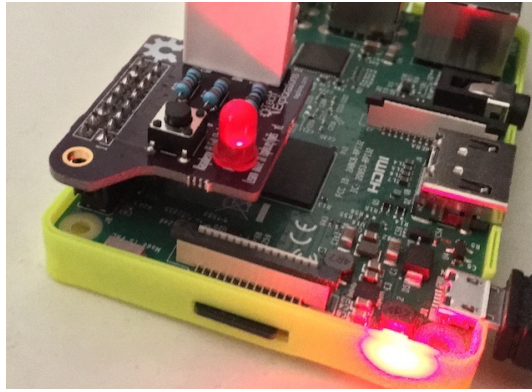


Figure 18.3: The board of this project will be similar to the one in the photograph, except that we will use SMD versions of the resistors and LED.

You will use your experience from the first project to speed up the process, and at the same time extend your skills.

The board will contain these components:

1. An 8x2 header, female, with the standard 2.54 mm pitch.
2. A TH momentary button.
3. Two 10 k Ω resistors, SMD 0805 package.
4. One 330 Ω resistor, SMD 0805 package.
5. One red LED, SMD 0805 package.
6. One DHT22 sensor.

For the SMD components, I have chosen the 0805 package because that is large enough to make hand-soldering possible.

Let's begin.

18.2. What you will learn

In this project you will learn and practice:

- Adding surface-mounted modules (SMD) to your board
- Minimising the amount of space needed for a board
- Creating non-rectangular shapes for the board outline
- Adding mounting holes
- How to revise the board outline after you place the footprints in order to conserve space

18.3. Project repository

This project has a Git repository. You can access it at txplo.re/klpp2.

This repository contains multiple commits at each step of the design process, as I was going through it. Feel free to clone this repository on your

computer and make any modifications you feel like.

19. Project 3: Arduino clone with build-in 512K EEPROM and clock

19.1. Project details

In this project, you will build on the knowledge and skill of the previous two projects and design an Arduino-compatible board using, mostly, surface-mounted components. The board will contain a microcontroller, two EEPROM modules, a clock and calendar module, two crystal oscillators, and several SMD-sized resistors, capacitors, and LEDs. In addition, it will include headers for connecting peripherals to the microcontroller.

All this, in a relatively small board, measuring 44x28 mm. Working on this board will give you the opportunity to explore some of KiCad's powerful features, such as splitting schematic diagrams into more than one sheets, using an autorouter, and converting a two-layer board into a four-layer board.

The name I have given to this board is 'Battery powered Arduino with clock and extended EEPROM'. Or, let's call it 'BACEE', for short.

Here is the list of components:

Component	Description	Part
1	Battery (multiple cells)	Battery
2	Polarised capacitor	CP1
3	Unpolarized capacitor	C
4	LED generic	LED
5	Generic connector, single row, 01x09, script generated (KiCad-library-utils/schlib/autogen/connector/)	Conn_01x09_Male
6	Generic connector, single row, 01x04, script generated (KiCad-library-utils/schlib/autogen/connector/)	Conn_01x04_Male

7	Generic connector, double row, 02x03, odd/even pin numbering scheme (row 1 odd numbers, row 2 even numbers), script generated (KiCad-library-utils/schlib/autogen/connector/)	Conn_02x03_Odd_Even
8	Generic connector, single row, 01x04, script generated (KiCad-library-utils/schlib/autogen/connector/)	Conn_01x04_Male
9	Resistor	R
10	Resistor	R
11	I2C Serial EEPROM, 1024Kb, DIP-8/SOIC-8/TSSOP-8/DFN-8	24LC1025
12	IC MCU 8BIT 32KB FLASH 32TQFP	ATMEGA328P-AU
13	Clock/timer IC	DS1337S+
14	Two pin crystal	Crystal
15	Two pin crystal	Crystal

I designed this board because I wanted an Arduino-Uno compatible board, small in size, that I can power using alkaline batteries. I'd like this board to be able to keep the time, to have sufficient built-in memory for data logging, and to be able to connect a variety of sensors using standard headers. I also wanted a design that I can use as a basis for other similar projects. For example, I can design a variation of the base BACEE board with an integrated environment sensor, or wireless communications.

In Figure 19.1, you can see the four-layer 3D rendering of the board. The Atmega328P-AU MCU is in the centre of the board. The headers and the indicator LED are placed along the edges. Using SMD components helps to keep the board size to a minimum. I selected SMD packages that are large enough to be hand soldered.

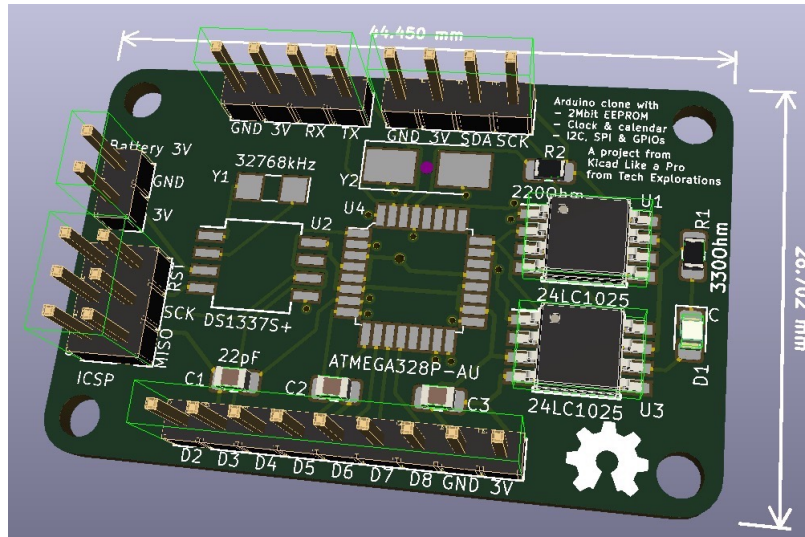


Figure 19.1: The outcome of this project.

For the most part, we will follow the schematic and layout design processes depicted in Figure 19.2 and Figure 19.3.

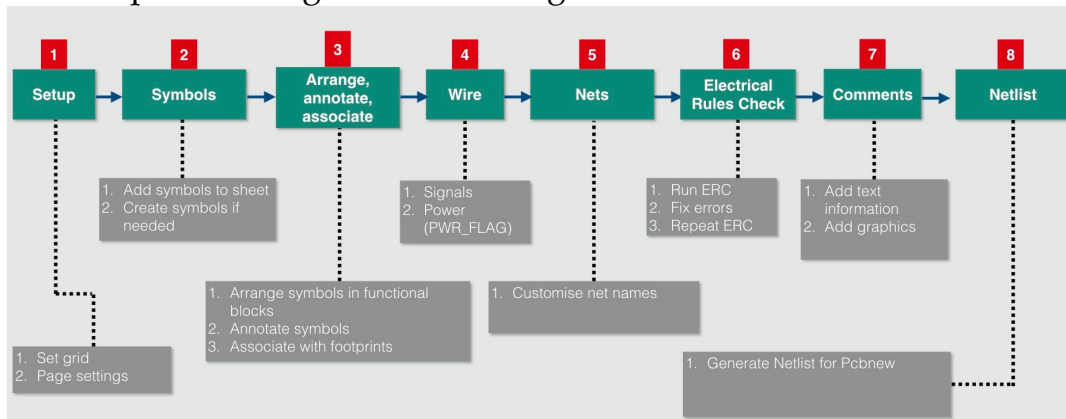


Figure 19.2: The schematic design process.

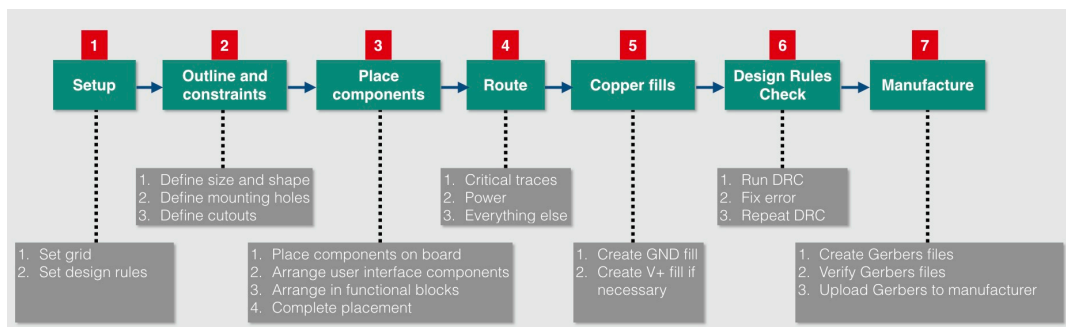


Figure 19.3: A serialised view of the PCB layout process.

However, as you are now more skilled in this type of work, you will feel free-er to iterate through the steps. For example, during the layout work, you may decide to redraw the outline of the board after you complete the footprint placement in order to accommodate a header. You can do that easily by adjusting your drawings in the Edge.Cuts layer.

Let's begin.

19.2. Project repository

This project has a Git repository. You can access it at txplo.re/klpp3.

This repository contains multiple commits at each step of the design process, as I was going through it. Feel free to clone this repository on your computer and make any modifications you like.

19.3. Schematic design in Eeschema

Start a new KiCad project, and store the project files in a new folder. I called mine 'BACEE'.

Part 5: Recipes

20. Adding a schematic symbol library in Eeschema

One of KiCad's great strengths is the sheer number of symbol and footprint libraries contributed by individual users and organisations. In this recipe, you will learn how to find, download and install symbol libraries to KiCad. Once you install a symbol library, you will be able to use its symbols in your schematics, precisely as you can with KiCad's built-in symbols.

You can find libraries for KiCad using Google, through searches like 'KiCad library download.' You may also know the location of contributed libraries as they are often shared in email lists and social media. Quality libraries are contributed by organisations like [Digikey](#), [Freetronics](#) and [Snapeda](#). Sources like Snepada allow you to use a search engine to find individual symbols, footprints or 3D representations of a component and import it into KiCad. Others, like Digikey, allow you to download full repositories of symbols, and footprints and install them in bulk into KiCad. Either way, the process is the same.

In this recipe, you will learn how to import the symbol libraries published by Digikey.

First, use your browser to visit the Github repository from where you can download the libraries archive. The repository looks like the example in Figure 20.1.

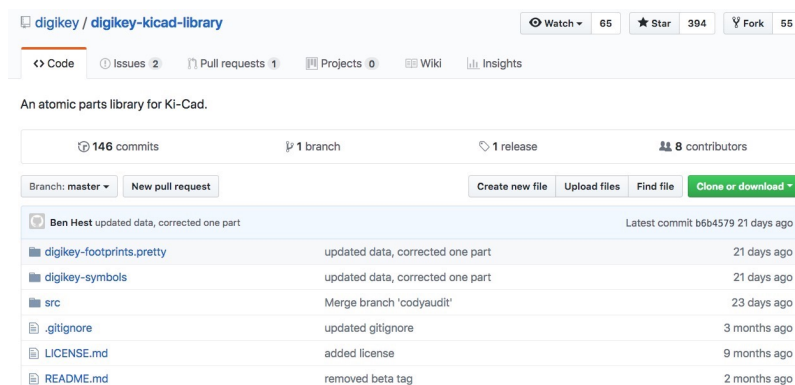


Figure 20.1: The Digikey KiCad libraries repository.

The folder titled 'digikey-symbols' contains symbols. The folder titled 'digikey-footprints.pretty' contains footprints. In this recipe we concentrate on the symbols, and in the next one on the footprints. Before you start the download, click on the symbols directory and have a look inside. You will see a long list of files with '.lib' and '.dcm' filename extensions. The files with the '.lib' extensions are the actual library files that contain the symbols. Those with

the '.dcm' extension contains descriptions, aliases, and keywords for the symbols.

Go back to the root of the repository, and click on the green button to download the ZIP archive of the repository. Expand the Zip file, and store its contents in a folder where you would like to keep third-party libraries. I placed mine in a folder titled 'KiCad Libraries' inside my Documents folder.

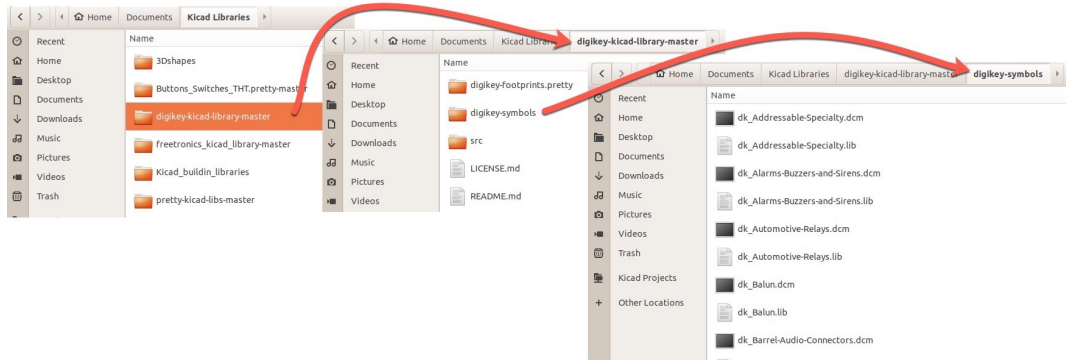


Figure 20.2: The Digikey libraries are now in the KiCad Libraries folder.

You can visit the contents of the Dikigey folder to confirm that it contains everything available on the Github online repository.

Continue by starting KiCad and Eeschema. Open the Library Manager by clicking on Preferences, 'Manage Symbol Libraries...' (Figure 20.3).

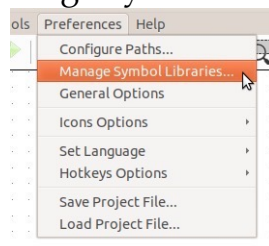


Figure 20.3: Start the Library Manager in Eeschema.

In the Library Manager, click on the 'Browse Library' button (Figure 20.4).

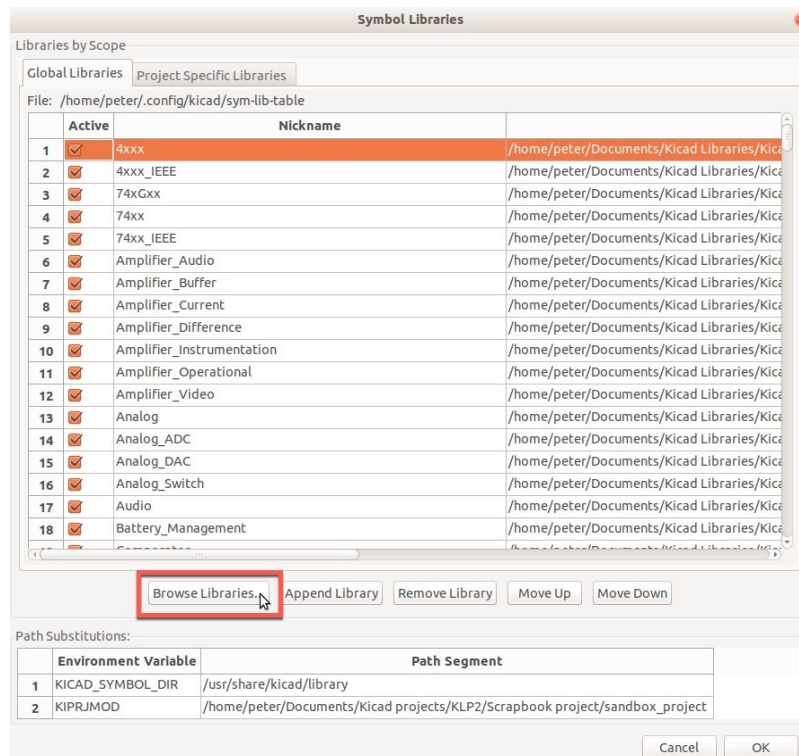


Figure 20.4: The Library Manager.

Using the browser, navigate to the location where the newly downloaded Digikey library files are. Since you want to import all of the symbols to KiCad, multiple select and highlight all files with a '.lib' extension. You can do this by selecting the first file in the list, hold down the shift key, scroll to the bottom of the list and click on the last file (Figure 20.5). When all lib files are selected, click 'Open'.

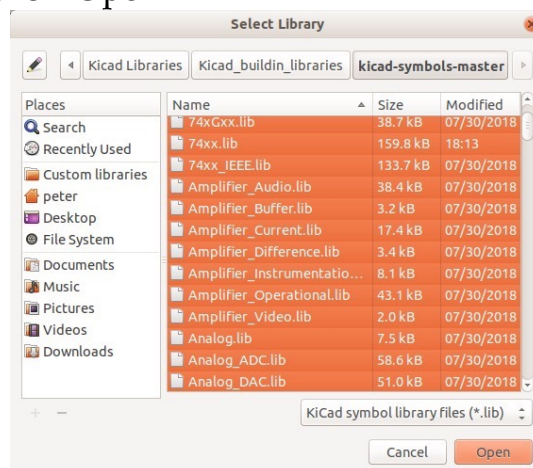


Figure 20.5: Select all '.lib' files.

All new symbol libraries are now in KiCad. You can verify this, first, by scrolling down the list in the Symbol Libraries manager window until you find the libraries with the 'dk' prefix (Figure 20.6). These are all the Digikey symbol libraries that you just imported.

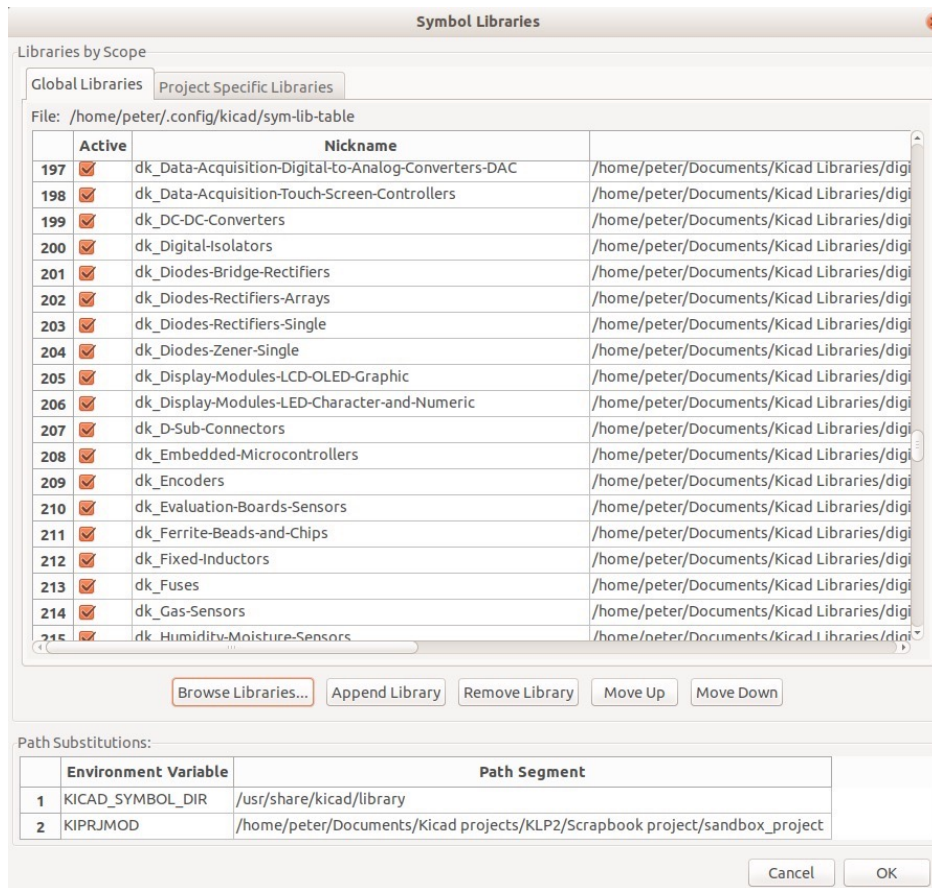


Figure 20.6: The newly imported symbol libraries have a 'dk' prefix.

Click Ok to dismiss this window and try to add a symbol from the new libraries to the Eeschema sheet. Click on the Place Symbol button (right toolbar) and click on the sheet, or type 'A' to bring up the symbol chooser. Scroll in the list, or type 'dk' in the filter to find the Digikey symbols (Figure 20.7).

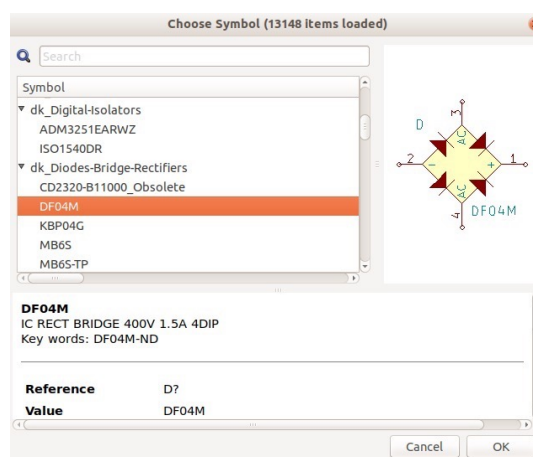


Figure 20.7: One of the symbols in the imported symbol libraries.

Select one of them, and double-click on it to add it to the Sheet. I selected the DF04M rectifier. At this point, you should have the symbol on the Eeschema Sheet, like in Figure 20.8.

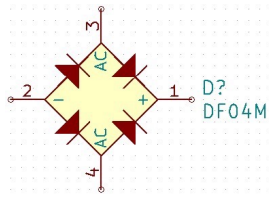


Figure 20.8: One of the new symbols, added to the Eeschema project sheet.

Using the exact same method, you can import individual symbols.

21. Adding a footprint library in Pcbnew

In this recipe, you will learn how to import third-party footprint libraries to your KiCad instance. In the example that follows, we will import the footprint libraries that are published by Digikey. In the previous recipe ('20. Adding a schematic symbol library in Eeschema'), you downloaded, expanded and save the library files to your computer. I assume that you have completed this step. If not, please go to the '20. Adding a schematic symbol library in Eeschema' recipe and complete the steps described there before you continue here.

Start KiCad, and Pcbnew. Open the Footprint Libraries manager through the Preferences menu (Figure 21.1).

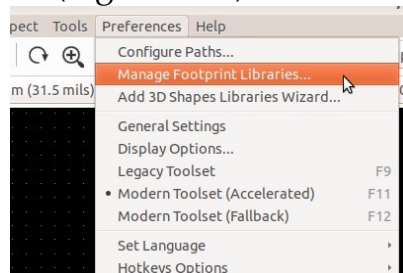


Figure 21.1: Start the Footprint Libraries manager.

In the Footprint Libraries, click on the 'Browse Libraries' button.

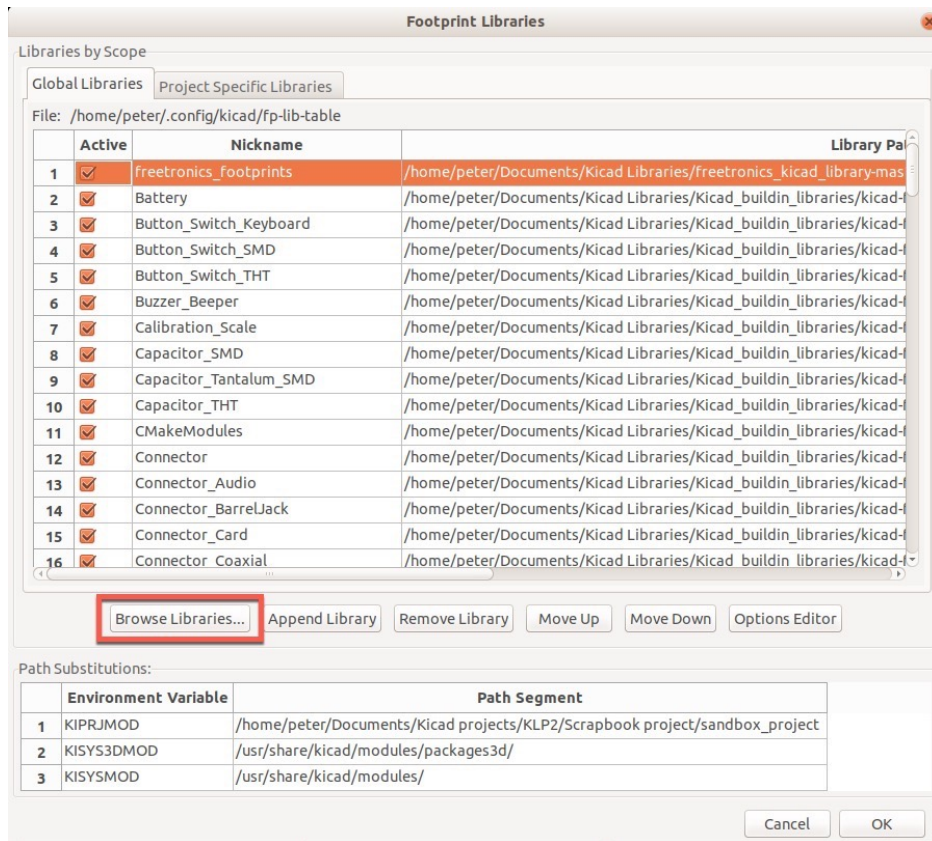


Figure 21.2: The Footprint Libraries manager; click on 'Browse Libraries' to add a new library.

Use the browser to navigate to the location where you stored the Digikey footprints library. This is the folder with the '.pretty' extension, inside the Digikey KiCad library folder. You should only select the .pretty folder, not browse to its contents. Inside this folder are multiple files with the '.KiCad_mod' extension. Each of those files contains one footprint (Figure 21.3).

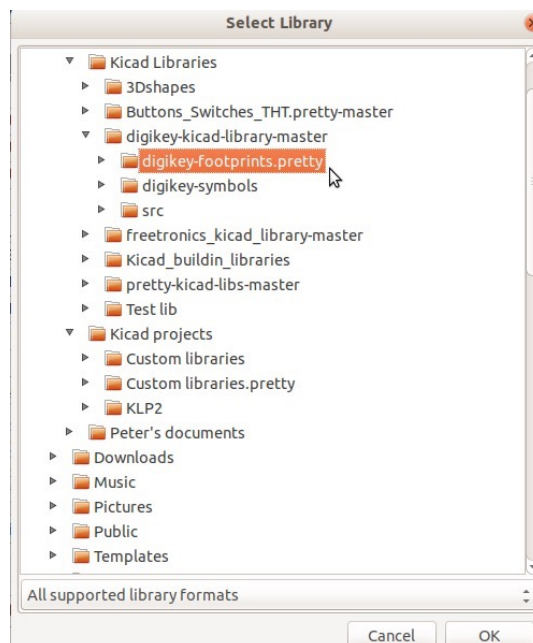


Figure 21.3: Find and select the .pretty folder that contains the footprints.

Click on the .pretty folder to select it, and then click on 'Ok'. Back in the Footprint Library manager, you will see a new row that contains the 'digikey-footprints' library (Figure 21.4). Click Ok to dismiss the manager window. The new library is now ready to use.

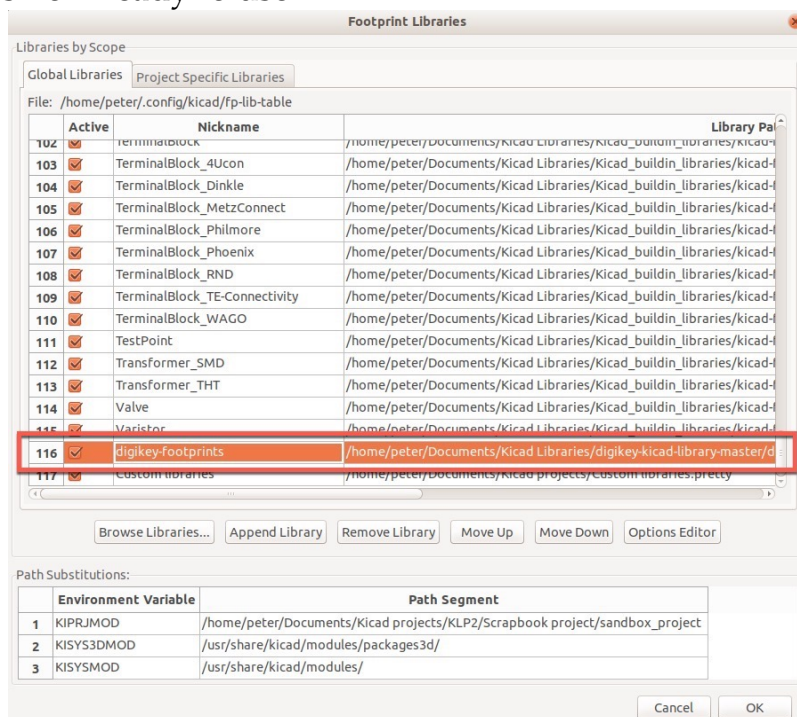


Figure 21.4: The new library appears in the Footprint Libraries list.

Back in Pcbnew, use the 'O' shortcut or select the 'Add footprint' button to add a new footprint. The footprint chooser will come up. Click on 'Select by Browser' button to bring up the browser. Let's find a Digikey footprint that matches the bridge rectifier symbol from the previous recipe. In the Browser, scroll down the left pane to find the digikey-footprints library, and then scroll down to find the DIP4_W7.62mm footprint. Click on this footprint to see it in the right pane (Figure 21.5).

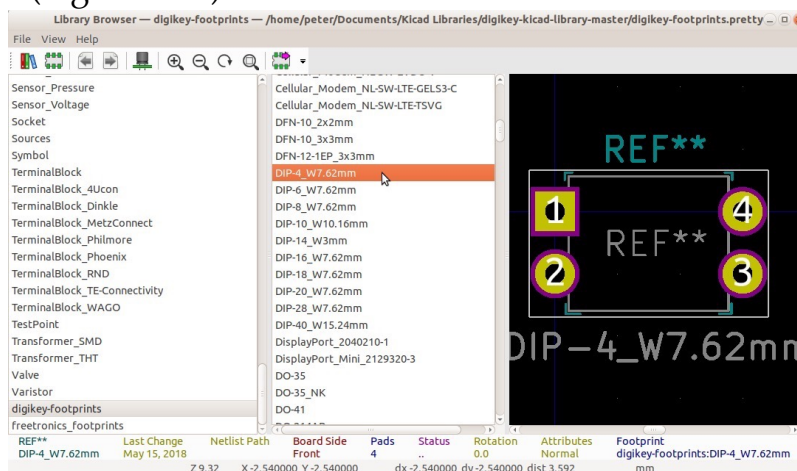


Figure 21.5: Browsing the new Digikey footprints library.

Double-click on it to select it and add it to the Pcbnew sheet. The selected footprint will now appear in the Pcbnew sheet (Figure 21.6).

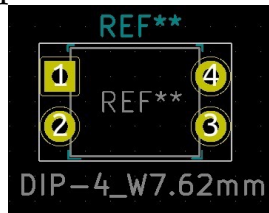


Figure 21.6: A footprint from the Digikey library in the Pcbnew sheet.

22. Using footprint libraries offline

You can opt to download Pcbnew footprints and use them locally instead of accessing the online Github repository. To do that, follow these steps:

1. Go to <https://github.com/KiCad/KiCad-footprints>
2. Click on the green 'Clone or download' button to download the repository.
3. Extract the downloaded Zip file into a folder of your choice. Consider creating a folder for all KiCad libraries in a reasonable place like the 'Documents' folder.
4. Start Pcbnew
5. Open the Footprint Libraries window, under the Preferences menu
6. Select all libraries and click on 'Remove Library' to delete them. If you have any third-party libraries that you want to keep, leave those unselected.
7. Click on the 'Browse Libraries...' button. The browser will appear.
8. Navigate to the location where you extracted the footprints, and open the folder that contains all the '.pretty' subfolders. Hold the Shift key pressed, click on the first .pretty folder on the list and then on the last one. This allows you to multiple-select all of the folders.
9. With all the .pretty folders selected, click the 'Ok' button to exit the browser.
10. The Footprint Libraries now contains a list of all the footprint libraries saved on your local machine, in the Local Libraries tab (Figure 22.1). This means that you can use these footprints in all your projects.
11. Click Ok to exit the Footprint Libraries window.
12. Test that Pcbnew can access the local footprints. While still in Pcbnew, click on the Add Footprints button from the right toolbar, and then anywhere in the page to add a footprint.
13. In the Choose Footprint window, click on 'Select by Browser.'
14. Use the footprint browser to select libraries and footprints, and ensure that all of them display a footprint representation in the right pane (Figure 22.2).

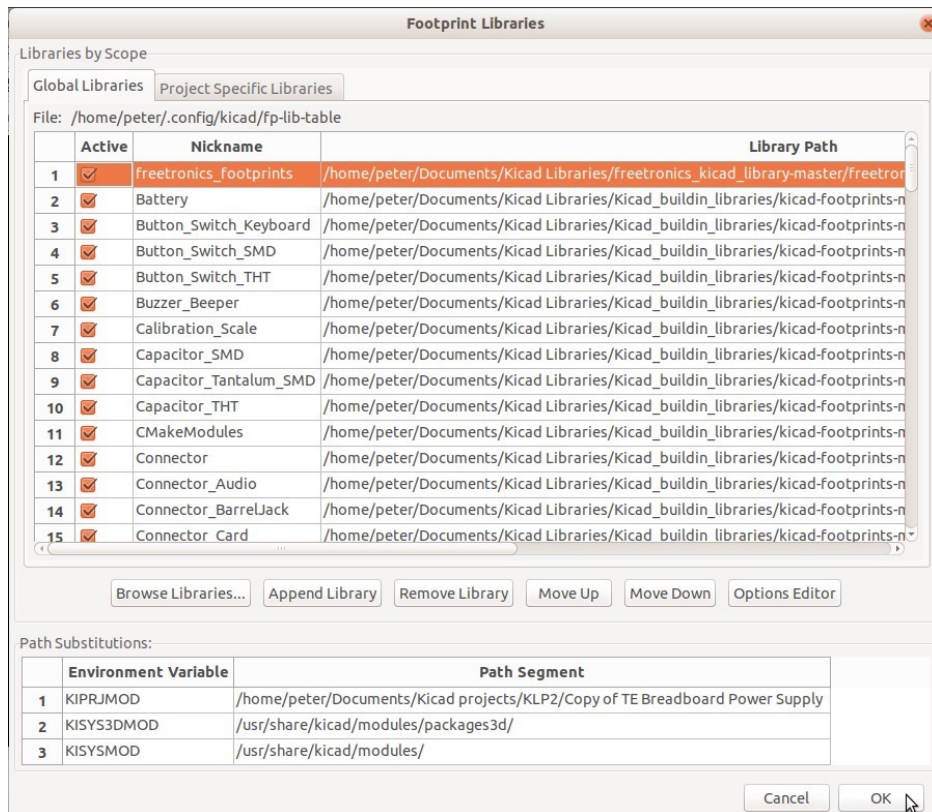


Figure 22.1: These footprints are stored locally.

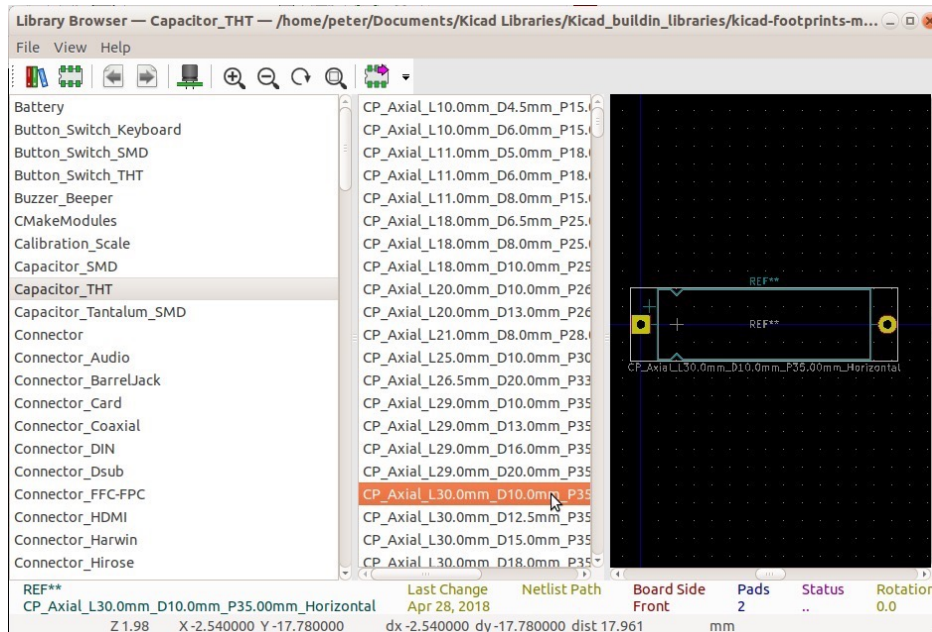


Figure 22.2: Testing local footprints.

As you are now using local footprints, remember to regularly download the latest version of the repository from GitHub, and repeat the process described above so that you always have the newest set of libraries.

23. Using symbol libraries offline

As with footprints, you can opt to download Eeschema symbols and use them locally instead of accessing the online Github repository. To do that, follow these steps:

1. Go to <https://github.com/KiCad/KiCad-symbols>.
2. Click on the green 'Clone or download' button to download the repository.
3. Extract the downloaded Zip file into a folder of your choice. Consider creating a folder for all KiCad libraries in a reasonable place like the 'Documents' folder.
4. Start Eeschema.
5. Open the Manage Symbol Libraries window, under the Preferences menu.
6. Select all libraries and click on 'Remove Library' to delete them. If you have any third-party libraries that you want to keep, leave those unselected.
7. Click on the 'Browse Libraries...' button. The browser will appear.
8. Navigate to the location where you extracted the footprints, and open the folder that contains all the '.lib' subfolders. Hold the Shift key pressed, click on the first .lib folder on the list and then on the last one. This allows you to multiple-select all of the folders.
9. With all the .lib folders selected, click the 'Ok' button to exit the browser.
10. The Symbol Libraries now contains a list of all the footprint libraries saved on your local machine, in the Local Libraries tab (Figure 23.1). This means that you can use these footprints in all your projects.
11. Click Ok to exit the Symbols Libraries window.
12. Test that Eeschema can access the local footprints. While still in Eeschema, click on the Place Symbol button from the right toolbar, and then anywhere in the page to add a symbol.
13. In the Choose Symbol window, expand the tree and click on any item. For each item you click on, you should see the symbol in the right pane (Figure 23.2).

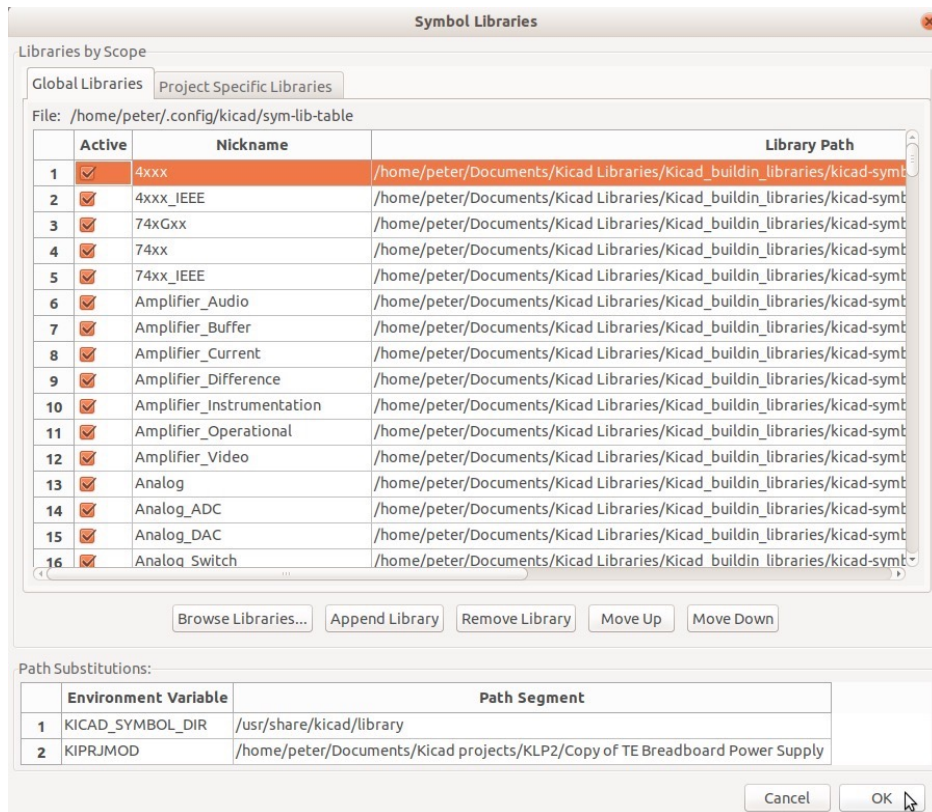


Figure 23.1: These symbols are stored locally.

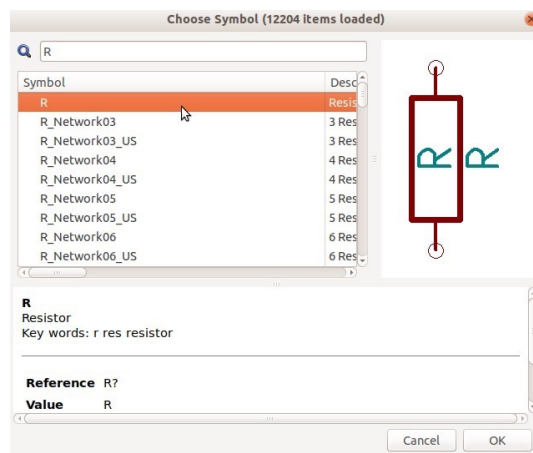


Figure 23.2: Testing local symbols.

As now you are using local symbols, remember to regularly download the latest version of the repository from GitHub, and repeat the process described above so that you always have the latest set of libraries.

26. How to calculate the width of a trace

KiCad includes a calculator that you can use to precisely work out what the width of a track should be based on various parameters, like the current you wish to convey through the trace, its total length, and the maximum temperature rise when that current is actually flowing through it. You can use this calculator to find out the minimum trace width, or you can rely on your experience and choose a width that is much larger than the standard width of signal traces.

To use the calculator, open the KiCad launcher window and click on the calculator icon (Figure 26.1).

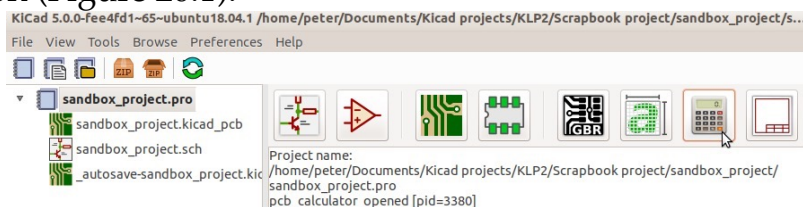


Figure 26.1: The calculator is available through the KiCad launcher window.

The calculator app actually contains multiple calculators. One of them is the Track Width calculator. Select it by clicking on its tab. Fill in the values that best describe your power track requirements. For a typical Arduino gadget, the values that you see in Figure 26.2 are reasonable. I have only altered the conductor length value to 30mm to better match the power trace length of one of my PCB projects. I tend to overshoot these values to ensure that the trace width that the calculator returns can comfortably cover the requirements.

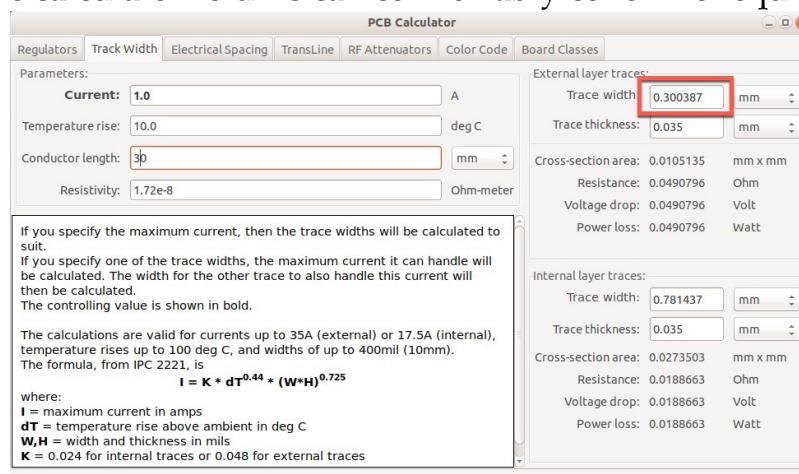


Figure 26.2: The Track Width calculator.

At the top right corner of the calculator, there is a field where you can

provide the trace thickness. This is a value that you don't have control over and is defined by the manufacturer's specifications (some manufacturers allow you to select the weight of your copper trace, but for simplicity let's assume here that this is fixed). The default value, 0.035 mm, seems to be an industry standard. Manufacturers typically make their boards with that trace thickness. To be sure, either search your preferred manufacturer's website for their trace thickness or ask them.

As you type in the parameters, the calculator returns the suggested trace width. In the example of Figure 26.2, the suggested width is 0.30 mm.

28. How to add silkscreen text and simple graphics

In KiCad, adding silkscreen text and simple graphics to your boards is very easy because there are tools dedicated to this task. In this recipe, you will learn how to do it.

I will demonstrate using the breadboard power supply board (Figure 28.1). This board is already routed and contains a ground plane which I have made invisible in order to make it easier to work with the silkscreen.

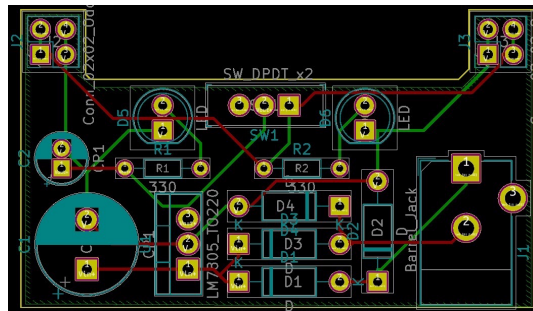


Figure 28.1: Let's add some custom silkscreen elements to this board.

The board already has several silkscreen elements on it, courtesy of the footprints we have used on it. The silkscreen elements are marked with a light-blue colour in Figure 28.1, and with white in the board's 3D representation (Figure 28.2).

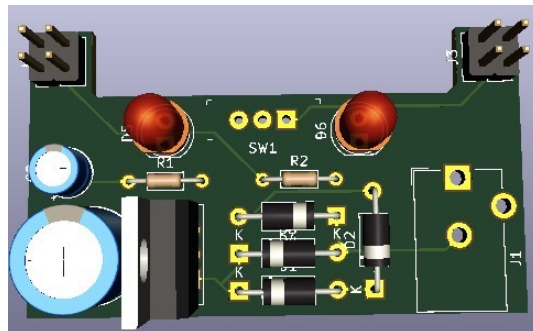


Figure 28.2: The existing silkscreen elements are shown in white in the 3D view of the board.

In the right bottom corner of the board, you can see the outline of the barrel connector footprint. There is a white line that marks the edges of the footprint, and the name of the footprint 'J1'.

Let's say that you want to make the text 'J1' to not be printed in the manufactured board. To do that, you must edit the text properties and make it invisible. Place your mouse course over the text, and type the 'E' shortcut (for 'Edit'). The Properties window in Figure 28.3 will appear.

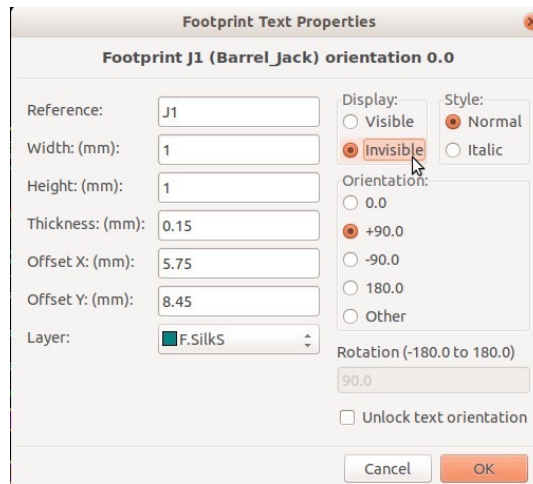


Figure 28.3: The Text Properties window.

Click on the 'Invisible' radio button to select it, and click on the Ok button. The 'J1' text should no longer be visible on the board, or in its 3D representation. To bring it back, either type 'Ctr-Z' (Windows, Linux) or 'Cmd-Z' (Mac OS) to undo the change. Or, edit the barrel connector footprint (place your mouse pointer on the footprint and type 'E'), then click on the Reference text 'Edit' button to bring up the text properties for the footprint reference text, and choose 'Visible' from the Display box.

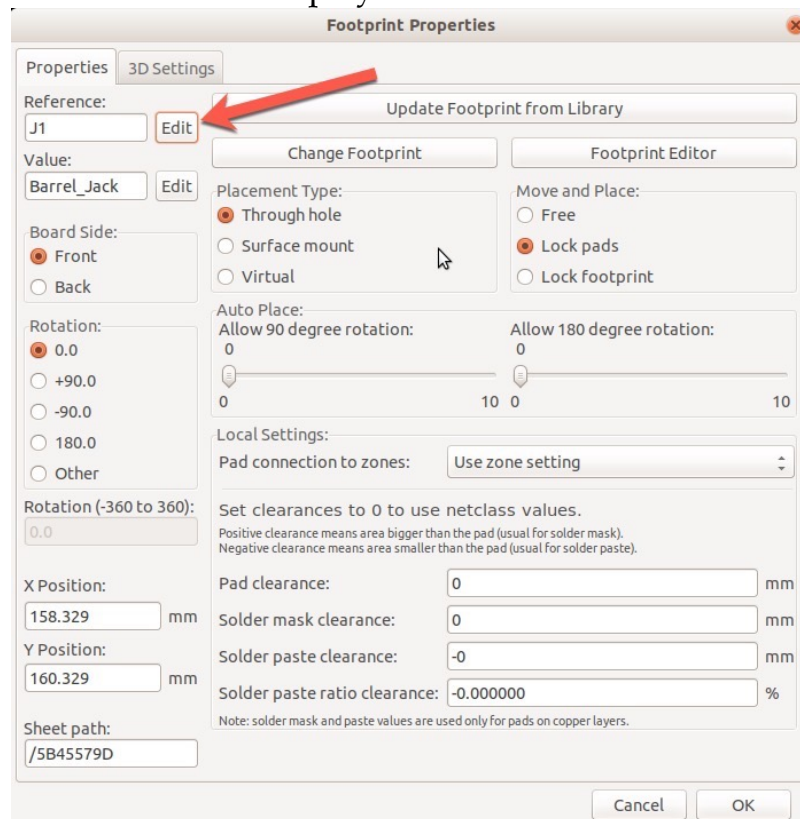


Figure 28.4: You can access the footprint reference text from the footprint properties.

Let's continue to add custom text and simple graphics to this board. First, because we want to insert silkscreen elements into the top layer of the

board, select 'F.Silks' from the Layers Manager. If we wanted to do the same to the bottom layer, we would select 'B.Silks' (we will do this later because there is an additional consideration for doing so).

To add text, click on the Text tool from the right toolbar (Figure 28.5).



Figure 28.5: These tools are used to create silkscreen elements.

Now click on the location on the board where you would like the text to appear. I will place mine over the barrel connector footprint. The Text Properties window will appear (Figure 28.6). Type your text in the text box. You can also control the size and other attributes of your text (I encourage you to experiment with them).

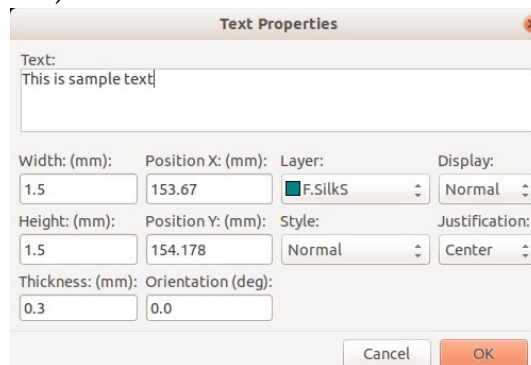


Figure 28.6: Text properties.

With the text and properties as it appears in Figure 28.6, click on Ok. The silkscreen should look like the example in Figure 28.7.

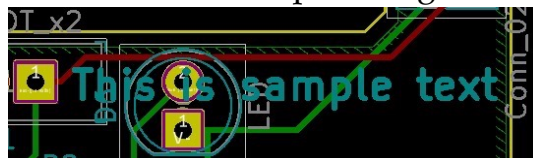


Figure 28.7: The new text appears in a single line.

The text appears in a single line, and it crosses with other elements, like the LED footprint and other text. It would be better to have this text appear in three lines, justified to the left, and moved to the right edge of the board. To do that, bring up the Properties window again (place your mouse pointer over the text and type 'E'), and break the text into three lines, and you can see in Figure 28.8. Also, change the justification to 'Right'. You can reduce the text

size by reducing the Width and Height to 1.0 mm.

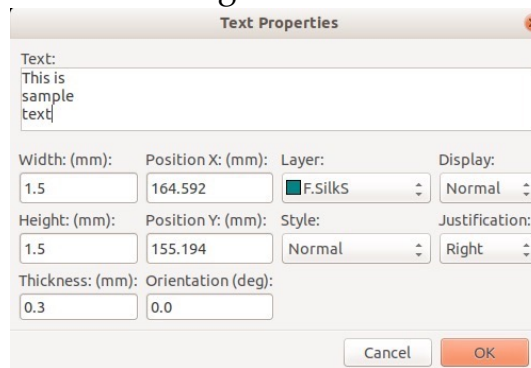


Figure 28.8: Text now appears in three lines, justified to the right.

Click Ok to commit the changes, and notice how this text appears on the board.



Figure 28.8: The text appears in three lines, justified to the right..

If you need to move the text, use the 'M' shortcut ('Move').

Next, let's add a box around the text. We'll use the polygon tool for this. If you wanted to draw a circle you would use the circle tool, and if you wanted to draw an arc you would use the arc tool. While you are still working on the F.Silks layer, click on the polygon tool (the first from the top in Figure 28.5), and click on the four corners around the text to create the outline. Double click to exit the drawing mode. The result should look like the example in Figure 28.9.



Figure 28.9: Added a box around the text.

In 3D, the board looks like the example in Figure 28.10.

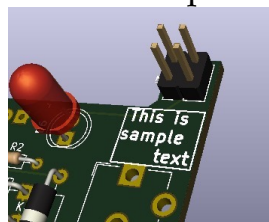


Figure 28.10: Added a box around the text, in 3D.

Now, let's say that you'd like to print the same text and box in the

bottom silkscreen layer. Start with the text. Place your mouse pointer on the text and type 'E' to show the text properties. Change the Layer to 'B.Silks' and the Display to 'Mirrored' (Figure 28.11).

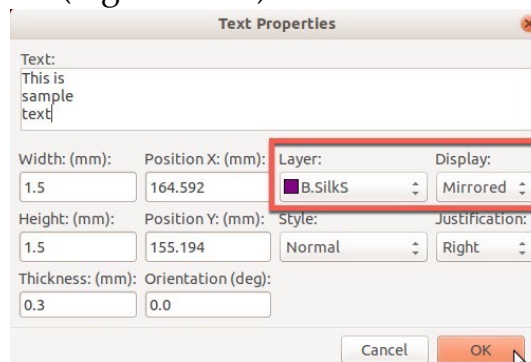


Figure 28.11: Moving the text to the bottom silkscreen.

Because the text is now placed in the bottom silkscreen, it is necessary to display it in mirrored orientation. Click Ok, and then used the 'M' key to move the text in position. Repeat the same process for each of the four lines that make up the box (Figure 28.12).

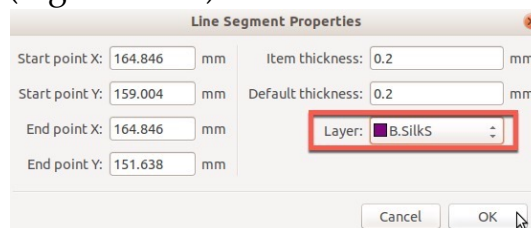


Figure 28.12: Moving the line to the bottom silkscreen.

You can see the result and its 3D representation in Figure 28.12.



Figure 28.12: The text and box appear in the bottom silkscreen.

The bottom silkscreen elements appear in purple. In the 3D view, turn the board around to see the bottom layer, where the silkscreen elements appear in white.

To learn how to add a custom logo to the silkscreen, please continue with the next recipe.

29. How to add a custom logo to the silkscreen

To add your logo or other graphics on your PCB, you first need to determine the dimensions of the area on the PCB where the logo will be placed. An important consideration is that the graphics file must be a bitmap (BMP or PNG), and monochrome. For the end result to be a crisp and high-quality graphic, you must also use a file with a high DPI (Dot Per Inch) count, at least 500 DPI. This is something I discovered experimentally.

In this recipe, you will learn how to add a logo to your PCB. KiCad provides a tool that allows you to convert a bitmap file into a footprint that can be imported into Pcbnew. Because there is a plethora of image manipulation programs and this book is not about image manipulation, I leave it up to you to create a suitable image file.

For this example, I will use a BMP file that has a size of 1140 x 448 pixels to create a footprint that will fit in an area that is around 10 mm wide on the PCB. The bitmap file is far larger than what is necessary for our 10 mm end result, but after a lot of trial and error I have discovered that including more data in the raw file (a larger file) will result in a silkscreen graphic with better detail, that looks crisp and professional.

From this starting point, follow this process to create the graphics footprint and add the logo to your PCB in the front silkscreen:

1. Open the main KiCad window and click on the 'Bitmap to Component Converter' button.

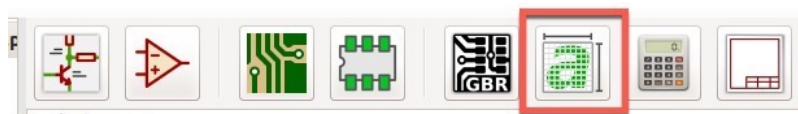


Figure 29.1: The 'Bitmap to Component Converter' button.

2. Click on the 'Load Bitmap' button and import the BMP file.
3. The image of the logo will appear on the left pane of the tool. Click on the 'Black&White Picture' tab.

4. The Bitmap to Component Converter is a very simple tool that allows you to do some basic manipulation of a bitmap image file. The most important modification you will need to do here is to match the size of the footprint you are about to create, with the area in which you want to place the logo. In this example, I would like to place the logo in an area that is around 10 mm in width. I'd like to the height to be proportional to the width. In

Figure 29. <\$n#figure: P5_Bitmap_to_Component_Converter_tool>, find the Bitmap Info box at the top right corner of the Converter. In it, you can see the raw dimensions of the image. The only item that you can edit is the resolution. Increase or decrease the resolution until the size matches your requirements. In my case, I had to increase the resolution to 2500 DPI (height and width) to get my images to around 10 mm. In fact, 11.6 mm across is still fine for the available space so I left the resolution at 2500 DPI.

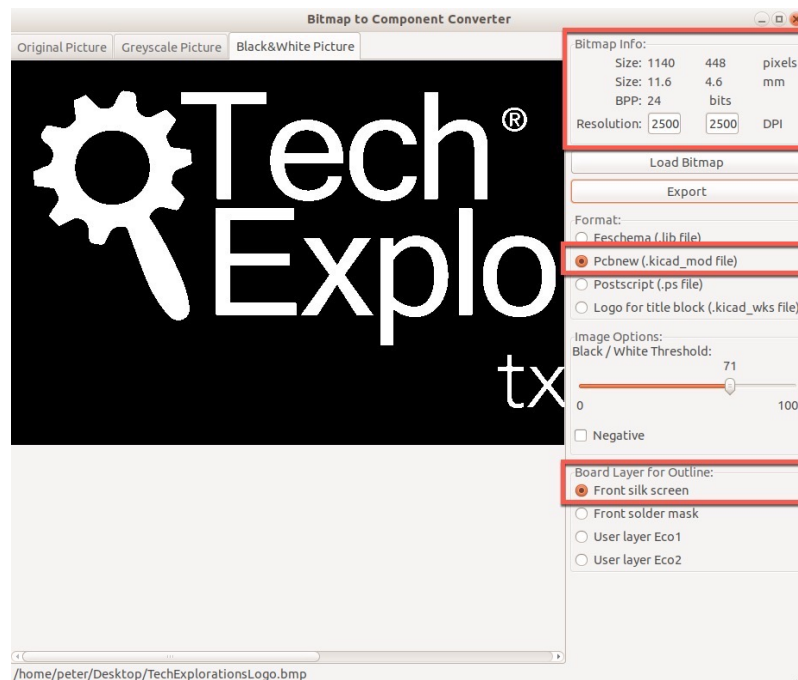


Figure 29.2: The Bitmap to Component Converter.

5. Ensure that Pcbnew and 'Front silkscreen' are selected, and then click on Export.

6. 'Export' will prompt you for a location for the new footprint library you are about to create. If you want to use this logo among all your project, create and select a new folder outside of your current project folder. In my case, I created a new folder inside my current project directory, named 'ProjectLibraries', gave this new library a suitable name 'TechExplorationsLogo_2500DPI.KiCad_mod', and saved the file. You can now close the Converter.

7. Go in Pcbnew, and click on Preferences, ManageFootprintLibraries. Before you use the new footprint you must import it. You can choose to import the new library for the current project only, or for all projects. In either case, click on the preferred tab, and then on the 'Browse Libraries' button.

8. Navigate your file system and click on the folder that contains the new library. Click Ok to select it. In Figure 29.3 you can see the imported library in my 'Project Specific Libraries' tab.

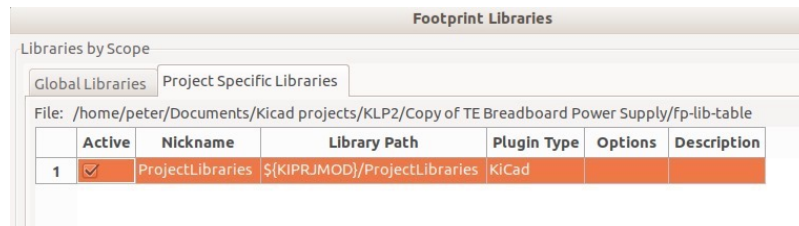


Figure 29.3: The new footprint library is specific to this project.

9. Select the front silkscreen 'F.Silks' from the Layers Manager.
10. Press the 'O' key (to add a footprint) or click on the 'Add Footprints' button from the right toolbar, and then click on the location on the board where you would like to add your logo.
11. The 'Chose Footprint' window will appear. Click on the 'Select by Browser' button. The Library Browser will appear (Figure 29.3).

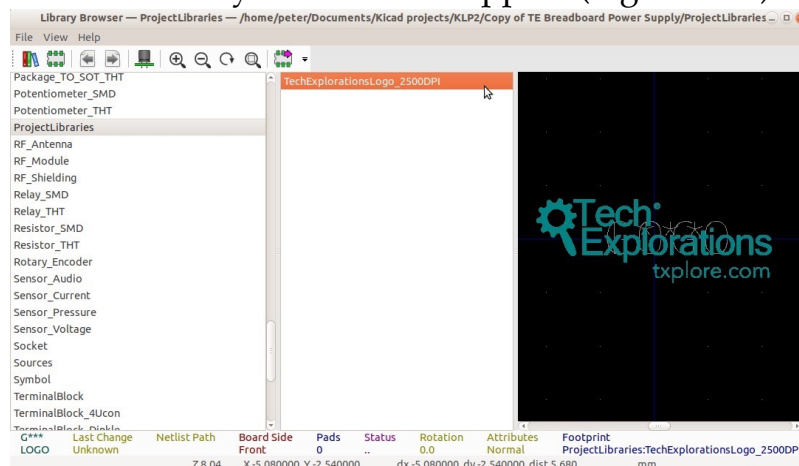


Figure 29.3: Find your new library in the Library Browser.

12. Look for your new library that contains your logo footprint. The library's name is the same as the name of the folder in which the logo footprint is.
13. When you find the library and footprint, double-click on it to select it and drop it on your board.
14. The logo will now appear in the front silkscreen layer of your board. You can verify that it looks as you want it by opening the 3D viewer (Figure 29.4).

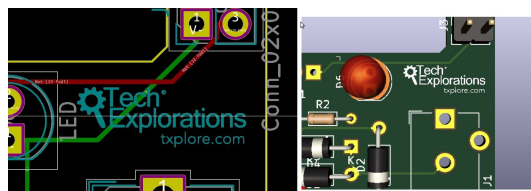


Figure 29.4: The logo appears on the front silkscreen.

What if you wanted to place this logo on the back silkscreen? All you have to do is to edit the logo footprint properties and select 'Back' as the board

side (Figure 29.5).

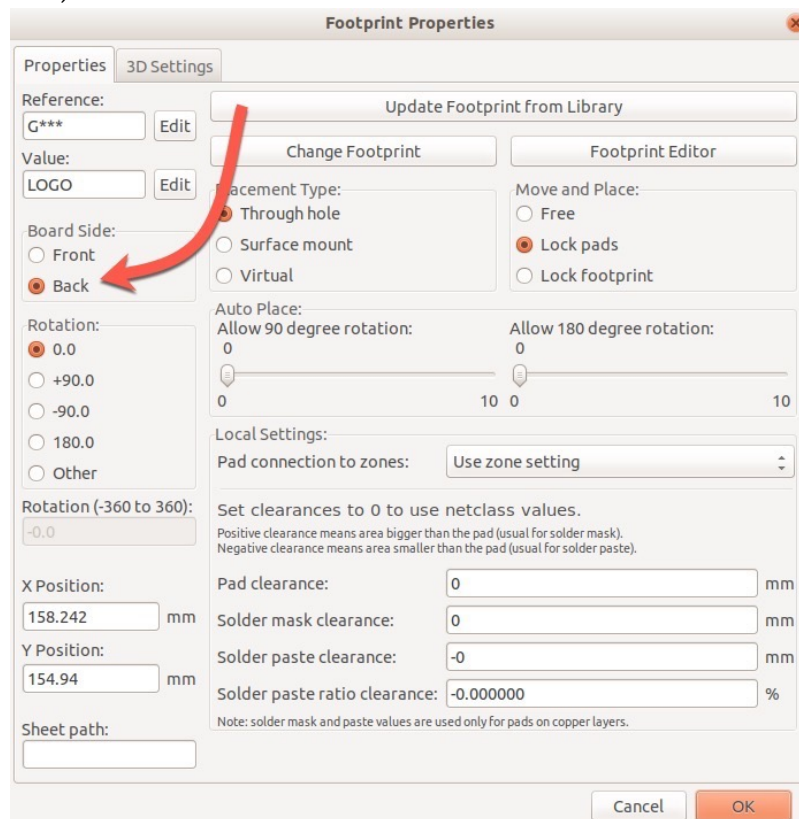


Figure 29.5: Switch your logo to the back silkscreen layer via the footprint properties window.

The PCB should now look like the example in Figure 29.6.

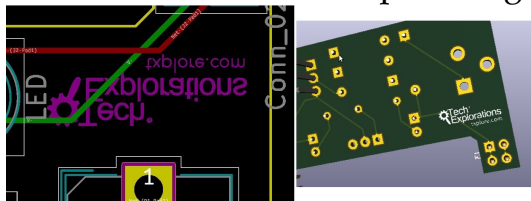


Figure 29.6: The logo now appears on the back silk screen.


As long as your original bitmap image is of decent quality, you will be able to decorate your boards with beautiful graphics.

31. How to make and test Gerber files

The vast majority of online PCB manufacturers accept Gerber files. Gerber files consist of an industry standard, and virtually every PCB CAD application can produce them. In this recipe, you will learn how to export the necessary Gerber files, package them in a Zip archive, and upload them to a manufacturer. For the manufacturer, we will use pcbway.com, but the process is very similar across the industry.

PCBWay is a quality manufacturer located in China. Apart from competitive pricing, they offer a vast range of customisations. While Oshpark aims for simplicity, PCBWay aims to satisfy every conceivable PCB manufacturing desire.

In Pcbnew, to generate the Gerber files, follow this process:

1. Click on the Plotter button  in the top toolbar to bring up the Plot window.
2. PCBWay provides information about which Gerber files to produce and the various settings that you will need to enable. Other manufacturers should have similar information available on their website. There are two sets of files that you must create. The first set contains individual files for each layer of the PCB. The second set contains a file for the drill so that the manufacturer can know how to drill the holes and vias. You can see the main Plot window in Figure 31.1. If you are uploading your Gerber to PCBWay, you can set up your Plot window as in this example. Before you click on the Plot button to generate the files, click on the folder button on the top right corner of the window to change the output directory to a new directory. I usually name directories that contain Gerber files 'Gerber'. Once you set the output directory, click on the Plot button. In the output messages text area, you will see green text indicating that the files were created. Do not close this window yet!

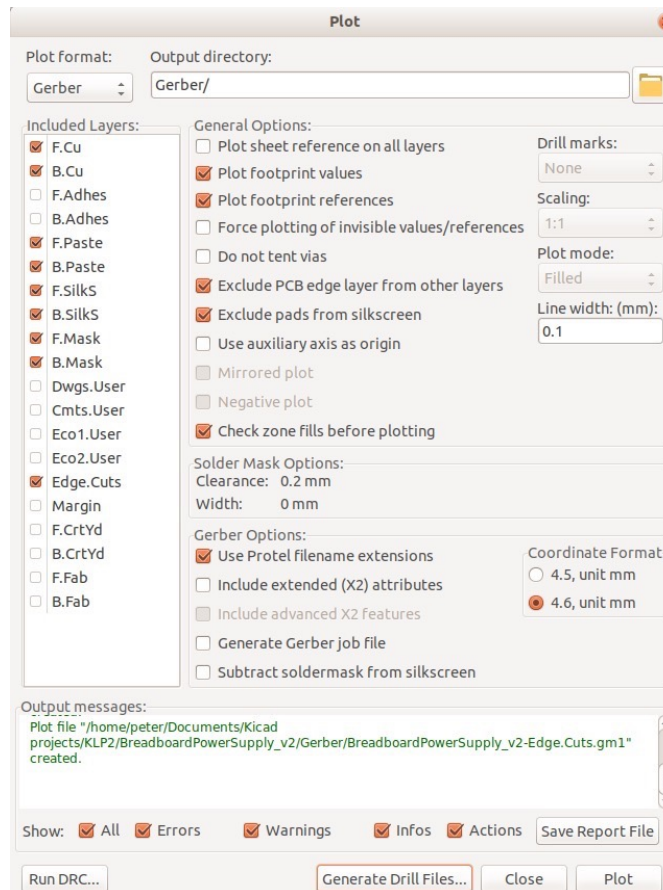


Figure 31.1: The Plot window, where you create the individual layer files.

3. You still need to create the drill files, unless your PCB has no holes. Click on the Generate Drill Files button in the bottom of the Plot window to do that. If you are uploading your Gerber to PCBWay, copy the setting from Figure 31.2, and click on Generate Drill File. This file will be stored in the Gerber directory along with the rest of the Gerber files. You can now Close both Drill and Plot windows.

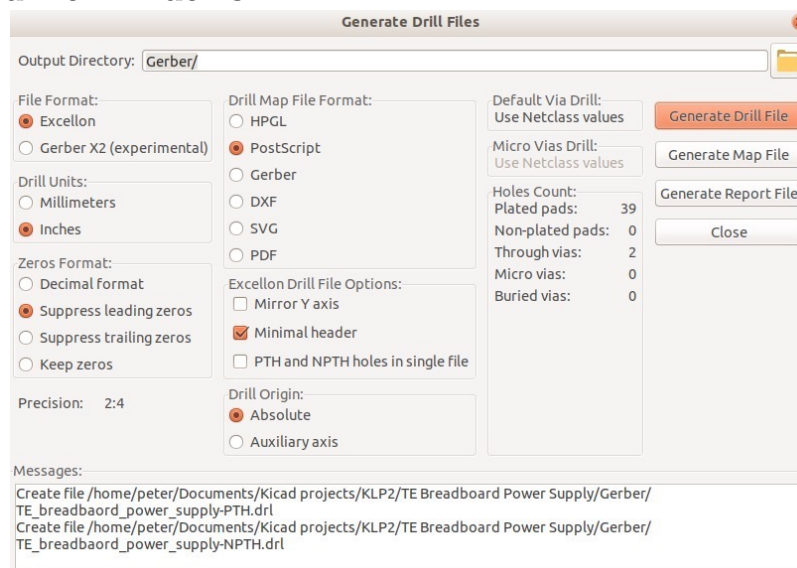


Figure 31.2: The Generate Drill Files window.

4. Use your file manager and go to your project's directory. Have a look inside the Gerber's directory and confirm that the Gerber files are there.

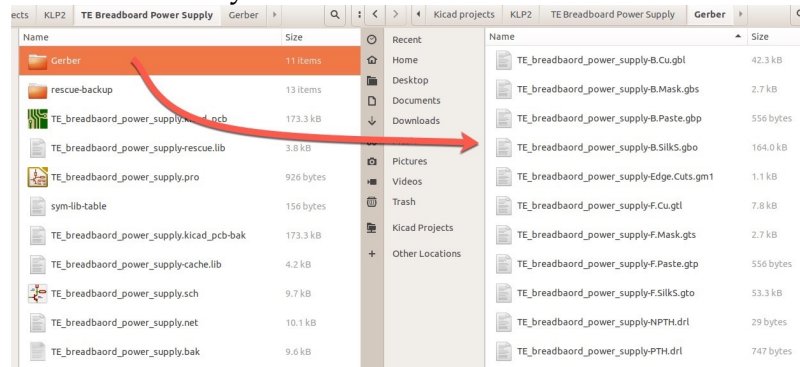


Figure 31.3: The Gerber files inside the Gerber directory.

5. Next, you must create a Zip archive that contains the Gerber directory with all the files in it. In Ubuntu, this can be done with a right-click, and by choosing the 'Compress...' option (Figure 31.4). The Gerber.zip files should be in your project directory (or wherever your archiving utility stored it).

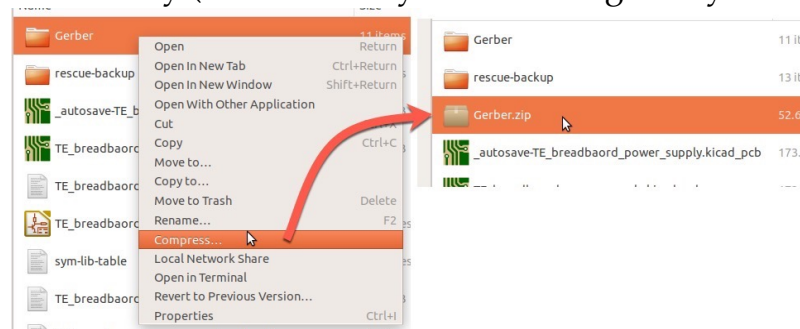


Figure 31.4: Create a Zip archive of the Gerber directory.

6. Upload the Zip file to a service like gerblook.org to ensure that they are correct. Gerblook will render the board layers, and you can visually inspect potential issues.

You are now ready to order your PCB from an online manufacturer using your Gerber files archive. To learn how to do this with pcbway.com please read the next recipe.

35. Creating a new component (symbol)

In this recipe, you will learn how to create a custom symbol. You use symbols in schematic diagrams that you can create in Eeschema. Much of what you will learn in this recipe you will be able to reuse for modifying existing symbols. There is a separate recipe in this book that explains how to do that.

Most likely, you want to create a custom schematic symbol because you have a physical component but can't find an existing symbol to represent it. Perhaps you have searched through the symbols that come with KiCad, and Googled for suitable third-party symbols, but couldn't find any.

It is also likely that you will want to associate your new symbol with a footprint. If your physical component has a standard package, like DIP, for example, then you will be able to associate an existing footprint with your custom symbol. If not, you will also need to create a custom footprint. You can learn how to do this in the relevant recipe, also available in this book.

In this recipe, you will learn how to create a custom symbol by creating a symbol for the 555 timer integrated circuit. There are plenty of libraries for KiCad that contain this symbol, but for the sake of learning, let's pretend that we can't find it.

What we want to create is a symbol like the one in Figure 35.1.

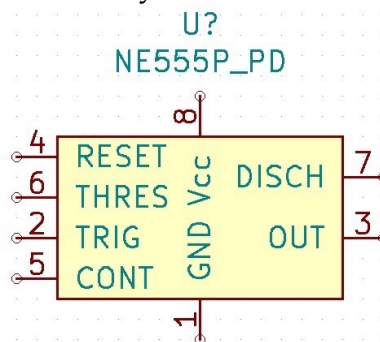


Figure 35.1: A custom-made symbol for the 555 IC.

Our objective is to create a symbol that complies with the convention. In regards to IC symbols:

1. We arrange pins around a rectangle.
2. We group pins according to function (like inputs, outputs, power etc.)
3. We place the Vcc pin on the top of the rectangle.

4. We place the GND pin on the bottom of the rectangle.
5. We choose an appropriate name and designator for the symbol.

Symbol designators are standardised; [you can learn more about them here](#).

The physical component that we are working with is in Figure 35.2. This component comes in a standard DIP package with 8 pins.

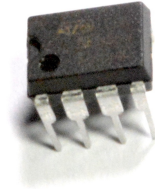


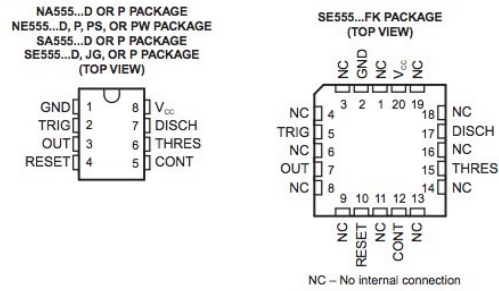
Figure 35.2: The physical component is a 555 timer in a DIP package with 8 pins.

Because we are working to create a symbol, we don't need to know anything about the physical characteristics of the physical component other than the total number of pins that its package contains. You need to know details about the physical characteristics when you are working on the footprint of the component.

However, it is beneficial to have access to the data sheet of the component. The data sheet contains information that you need: The names, numbers, and roles of each pin, whether they are input, output, bidirectional, power, signals, etc. All this is useful information, and the more you have on hand, the better.

The data sheet for the example component is available from its [manufacturer](#). The information you need is on page 6, and I have included it in Figure 35.3 for your convenience.

6 Pin Configuration and Functions



Pin Functions

NAME	PIN		I/O	DESCRIPTION
	D, P, PS, PW, JG	FK		
CONT	5	12	I/O	Controls comparator thresholds, Outputs 2/3 VCC, allows bypass capacitor connection
DISCH	7	17	O	Open collector output to discharge timing capacitor
GND	1	2	–	Ground
NC		1, 3, 4, 6, 8, 9, 11, 13, 14, 16, 18, 19	–	No internal connection
OUT	3	7	O	High current timer output signal
RESET	4	10	I	Active low reset input forces output and discharge low.
THRES	6	15	I	End of timing input. THRES > CONT sets output low and discharge low
TRIG	2	5	I	Start of timing input. TRIG < 1/2 CONT sets output high and discharge open
V _{cc}	8	20	–	Input supply voltage, 4.5 V to 16 V. (SE555 maximum is 18 V)

Figure 35.3: Pin configuration and functions from the IC's datasheet.

Let's start the process of creating a new symbol. In the main KiCad window, click on the Symbol Library Editor button (Figure 35.4).

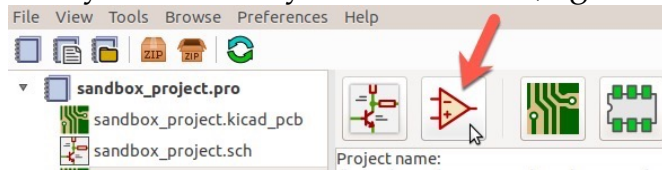


Figure 35.4: Start the Symbol Editor.

You must store each symbol inside a library file, so before you start creating the symbol create a new library. Create a new library by clicking on

the 'New library' button (), or through the File menu.

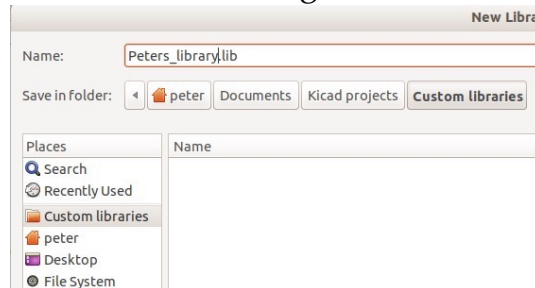



Figure library; I have placed mine in a directory named 'Custom libraries'.

KiCad will ask you if you would like this library to be available to all projects ('Global') or only to the current project ('Project'). Choose the most appropriate one for your circumstances (I chose 'Global'). You are now

working in the new library, and you will store the new symbol in it. You can confirm this by looking at the header of the Editor window. The path and name of the library you just created must be showing there.

Next, click on the 'Create new symbol' button in the top toolbar (). A prompt will ask you to select the library in which you will store the symbol. The library you just created should be listed. Click on it to select it and click 'ok' to continue. The Symbol Properties window will show up. The most important values that you need to complete are the symbol name and designator. The name typically consists of the physical component's model name and any other information that helps to identify it. When you use it later, you have to search for it in the symbol library and having a good name will help you find it quickly. In my example, because I want to differentiate my 555 symbol to those that exist in other libraries, I add my initials 'PD' at the end of the name.

For the designator, you should not guess. Visit Wikipedia to see the [Reference Designators](https://en.wikipedia.org/wiki/Reference_designators) table (https://en.wikipedia.org/wiki/Reference_designator). You can see a section of this table in Figure 35.5. The designator for integrated circuits is 'U,' so type this in the Default Reference Designator field.


RV	Varistor
S	Switch (all types, including push-buttons)
T	Transformer
TC	Thermocouple
TP	Test point
TUN	Tuner
U	Integrated circuit (IC) 
V	Vacuum tube
VR	Variable resistor (potentiometer or rheostat)

Figure 35.5: A section of the reference designators standard IEEE 200-1975/ ANSI Y32.16-1975.

Figure 35.6, shows the values I have entered in the Symbol Properties window. Other than the symbol name and the designator, everything else remains as per the default.

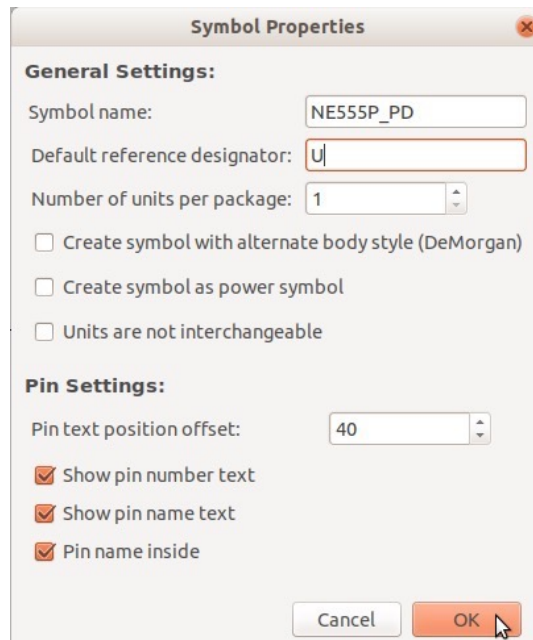


Figure 35.6: The properties for the new symbol.

Click Ok to commit and continue. KiCad will place the designator and symbol name in the middle of the sheet, on top of the other. Use the 'M' hotkey to relocate the two blocks of text. You should have something similar to the example in Figure 35.7.

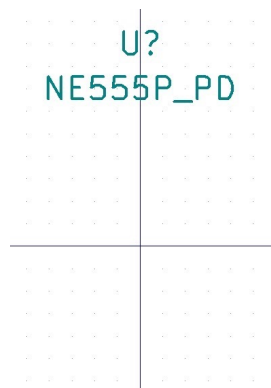




Figure 35.7: An empty new symbol.

Continue by drawing the outline of the symbol. You can either use the polygon tool () or the rectangle tool () to do this. Your outline should look like the example in Figure 35.8.

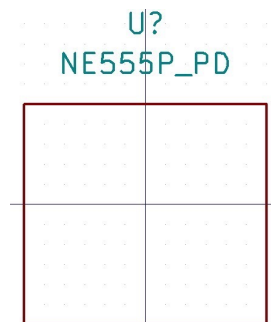


Figure 35.8: The outline of the footprint.

Add the background colour that is consistent with other IC symbols by opening the drawing properties window for the rectangle (place your mouse pointer on the rectangle line and type 'E'). Under 'Fill Style', check the 'Fill background' radio button (Figure 35.9).

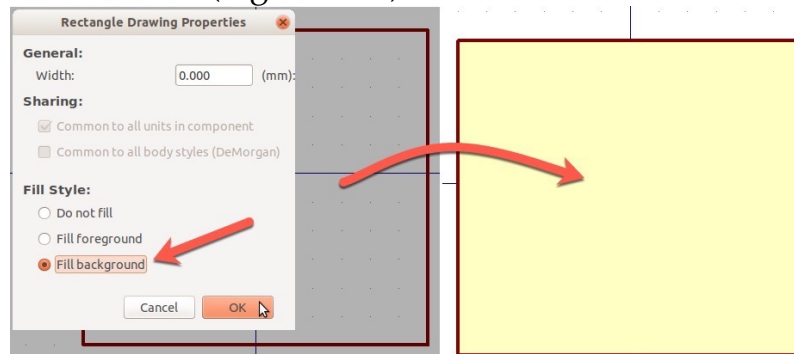



Figure 35.9: Fill the rectangle background.

Next, work on the pins. Keep the datasheet open because you need the information in it. For your convenience, refer to the excerpt in Figure 35.3.

Click on the pin button in the right toolbar (). Place the 8 pins around the perimeter of the symbol outline, as in the example of Figure 35.1. The thing to remember here is that the convention is to group similar pins together and place the two power pins to the top and bottom of the rectangle. Let's start with the Vcc pin. According to the datasheet, the Vcc pin is number 8, and according to the convention, it should go to the top edge of the rectangle. Click on the pin tool, and then click in the middle of the top edge. The Pin Properties window will come up. Fill it as you can see in the example in Figure 35.10.

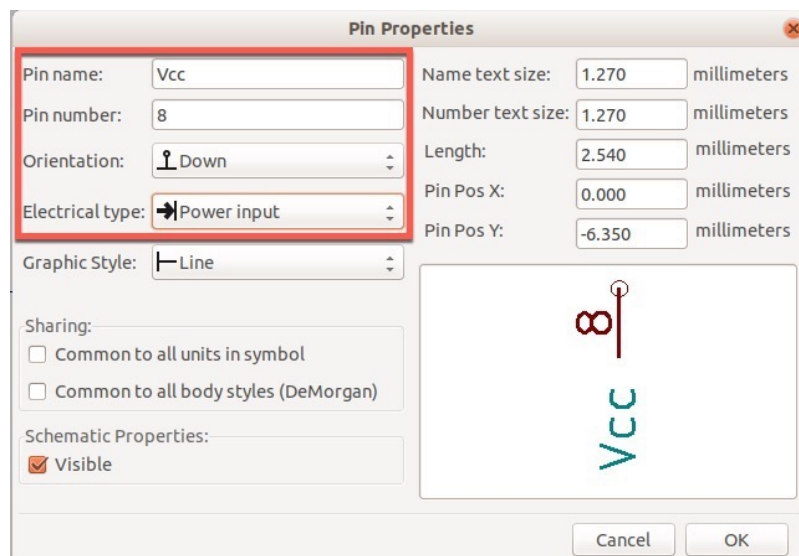


Figure 35.10: The Vcc pin properties.

In Figure 35.10, the fields inside the box are the ones that you need to focus on. The pin name is arbitrary, but of course, you should use a name that is appropriate. I usually use the same name that I see in the documentation for this pin. The Pin number, on the other hand, is very important. The Pin number is how schematic symbols and footprints can associate physical and symbolic pins. When you design the custom footprint for this physical component in the 'Creating new footprint' recipe, it is the number that you put in the Pin number field that dictates which net in the schematic diagram connects to the correct pad in the footprint. Take the pin number for the Vcc pin from the documentation ('8') and type it in this field.

In the orientation drop-down, select the option that matches the side of the rectangle where you are attaching the pin. The Vcc pin should go on the top of the rectangle and should have its circular connector pointing away from the rectangle. The horizontal line of the pin orientation icon represents the rectangle. If you wanted to place the pin on the left of the rectangle, you would choose the icon with the circular connector printing towards the left.

Finally, because the Vcc pin is a power pin, I have selected the 'Power input' electrical type. You should choose the same type for the GND pin.

Click Ok to commit the changes. Place the pin in the middle of the top side of the rectangle, as you can see in Figure 35.11. I have moved the text block so that they don't overlap with the pin.

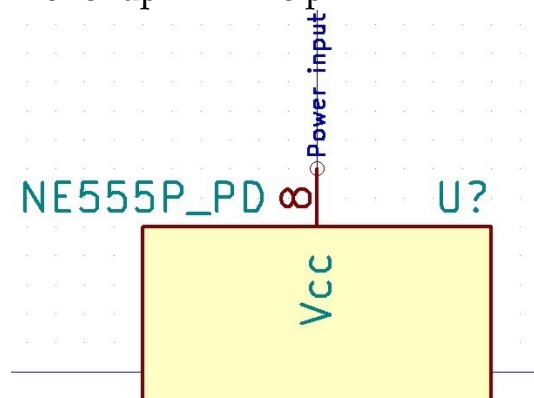


Figure 35.11: The Vcc pin, in place.

Follow the same process to add the GND pin (pin 1) in the bottom edge of the rectangle. Copy the pin name and number from the datasheet, and mark it also as a Power Input.

Continue with the left side of the rectangle where you should place the input pins. According to the datasheet, the input pins are RESET, THRES, TRIG. There is one bidirectional pin, 'CONT,' which you can place either on the left or the right of the rectangle. I have placed it on the left. Your symbol should look like the example in Figure 35.12.

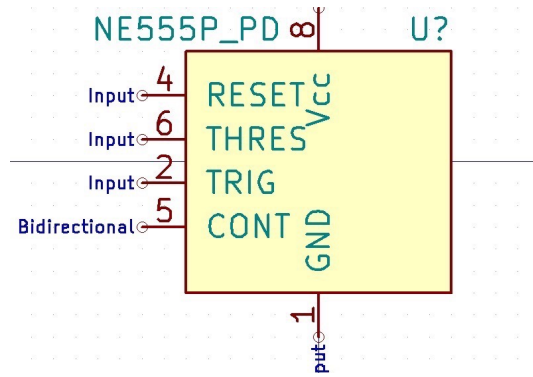


Figure 35.12: Input pins are placed on the left side.

The pin attributes for the input pins on the left of the symbol look like the example in Figure 35.13. Remember that pin 5 is bi-directional so its electrical type should be 'bidirectional'.

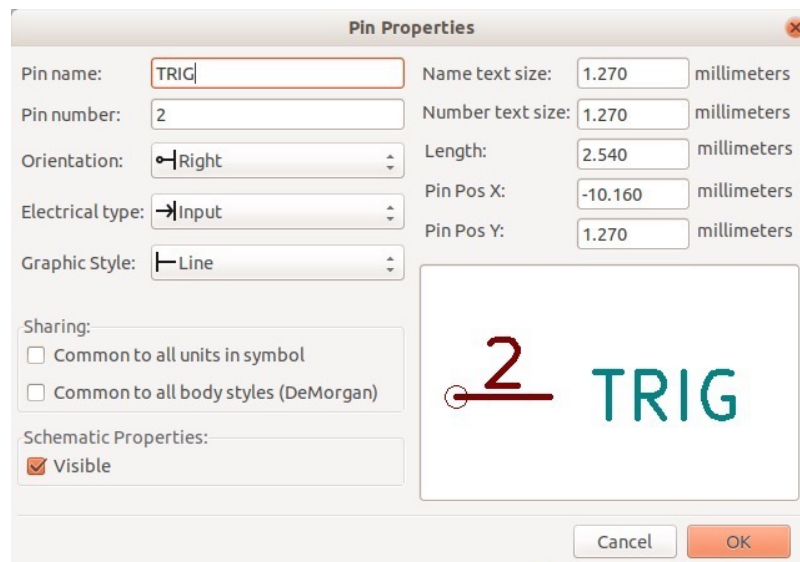


Figure 35.13: The input pin attributes.

Continue in the same way to create the last two pins. Those are output pins, as per the datasheet. Use the pin names and numbers as those appear in Figure 35.3 for pins 7 and 3. When completed, you should have a symbol that looks like the example in Figure 35.14.

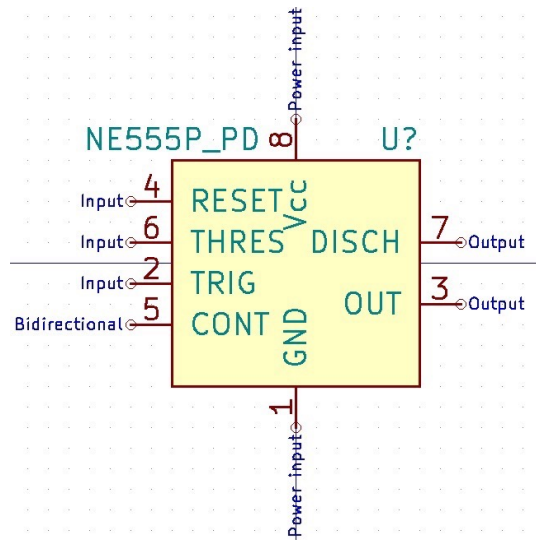


Figure 35.14: The completed custom symbol.

The last thing to do is to add the URL for the real-world component datasheet to the symbol properties. This will be useful to you for future reference. It will be surely useful when you go ahead to create a custom footprint to match the symbol. To add the datasheet URL, click on the Symbol, 'Fields...' to bring up the Field Properties window. Click on the Datasheet row to select it and then copy / paste the URL in the Field Value field (Figure 35.15). Click Ok to commit the changes.

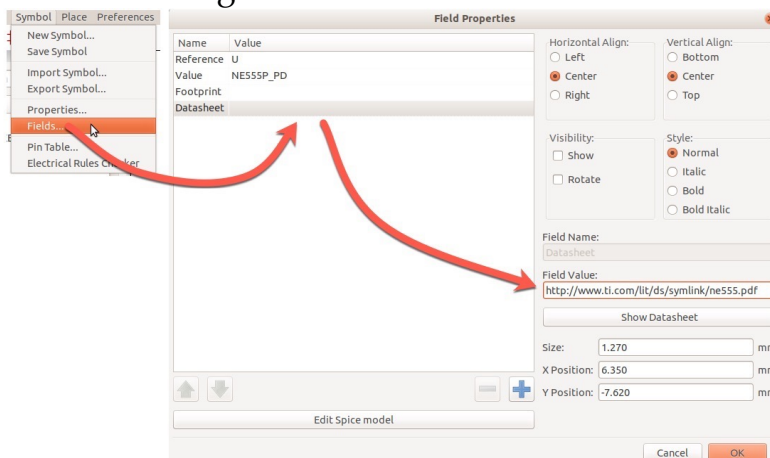



Figure 35.15: The datasheet is very useful to have readily available.

Your work is complete. Save the symbol to the selected library by

clicking on the Save Current Symbol button (). Then, test that you can use it in Eeschema. Open Eeschema. Go into Symbol Libraries from the Preferences menu and add the new library (read the relevant recipe if you don't know how to do this). Place your cursor on the Sheet and type 'A' to add a new symbol. Search for the name of your library by typing part of its name in the filter field (Figure 35.16). The library should appear. Double-click on the

symbol to drop it on the sheet.

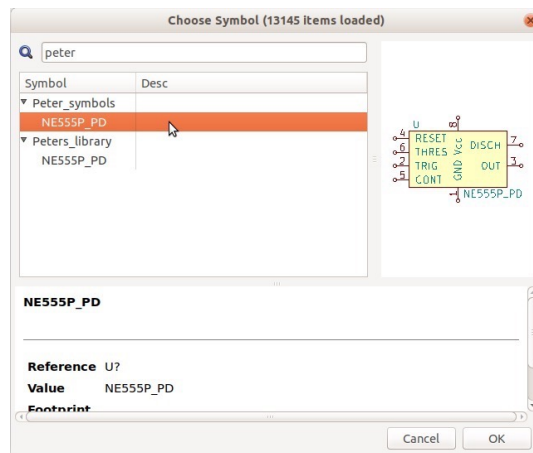


Figure 35.16: Find your new library and symbol.

The custom 555 symbol should now be in place inside Eeschema, and you can go ahead to use it as you do with any other symbol (Figure 35.17).

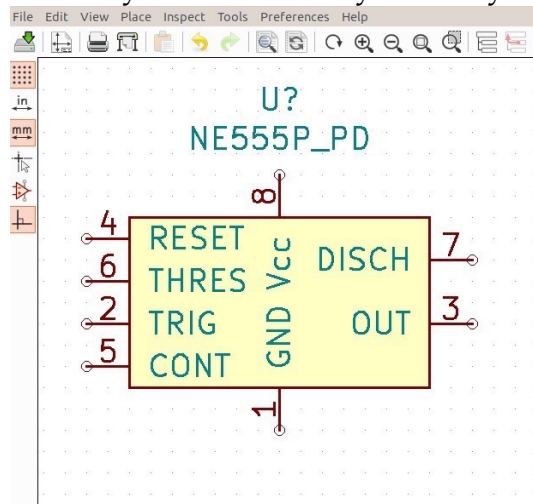


Figure 35.17: Your new custom symbol in Eeschema.


In this recipe, you learned how to create a brand-new symbol. What if you have found a symbol that is close to what you want, but could be perfect with a bit of tweaking? In other words, what if you want to modify an existing symbol? You can learn how to do this in the '36. Modifying an existing component (symbol)' recipe.

36. Modifying an existing component (symbol)

In this recipe, you will learn how to modify an existing symbol. It builds on knowledge that you have learned in the recipe on how to create a custom symbol. If you have not read that recipe (titled 'Creating a new component (symbol)'), please do that now and come back when you have.

The process of modifying an existing symbol starts with finding the symbol that you want to modify. Then, you create a copy of the original symbol, make the changes, and save it to an existing or new symbol library.

Let's begin. Your objective is to make a few cosmetic changes to a symbol that ships with one of the standard symbol libraries of KiCad 5. The symbol name is '74HC595'. This is a common 8-bit shift register that you will find in many Arduino projects. The symbol itself is complete as it comes out of the box, but for the sake of this recipe, let's say that you would like to re-arrange some of the pins, change the location of the text blocks, and add some descriptive text.

First, in Eeschema, find the symbol you want to modify and add it to the sheet. You can do this using the 'Place Symbol' tool from the right toolbar (). With the symbol on the sheet, right click on it to reveal the context menu and click on Properties, 'Edit with Library Editor'.

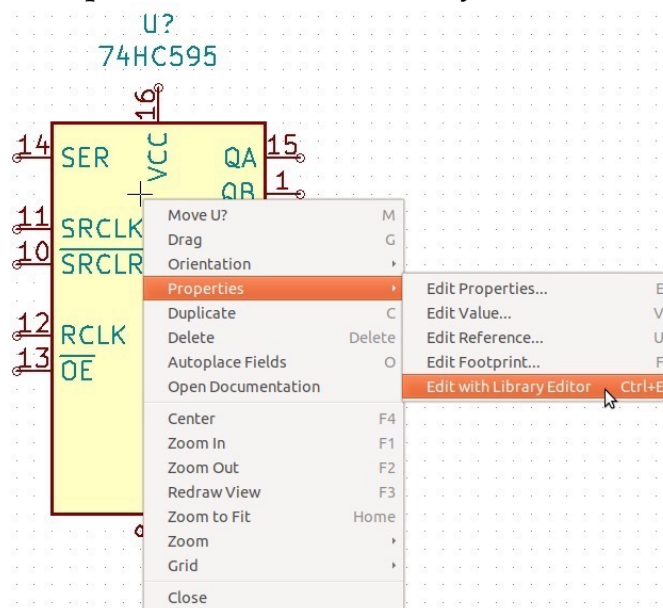


Figure 36.1: Edit an existing symbol with the Library Editor.

You have already used the Library Editor to create a new symbol in the

'35. Creating a new component (symbol)' recipe, so the editor window should be familiar (Figure 36.2).

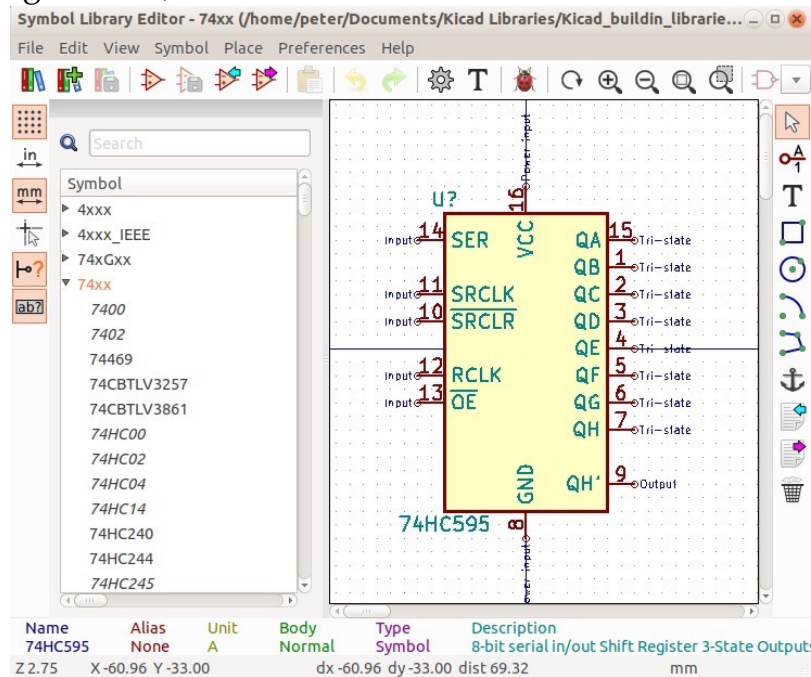


Figure 36.2: An existing symbol, waiting for your modifications.

Using the knowledge you gained in the '35. Creating a new component (symbol)' recipe, move the pins around, change their text, modify and move the text blocks, or add new text as you see fit. I modified the symbol to look like the example in Figure 36.3. I have added a block of text that describes the device and moved the input pins along the left side of the rectangle.

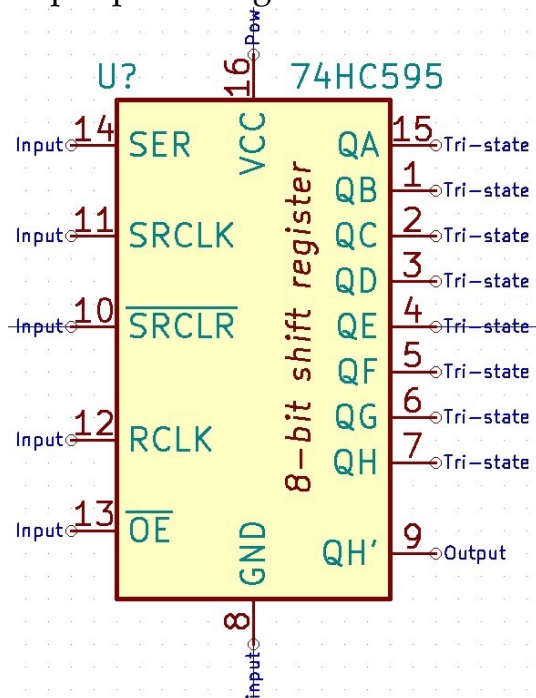




Figure 36.3: A slightly modified version of the build-in 74HC595 symbol.

When your modifications are complete, you must save the footprint in a library. If you click on the Save button (), then you will overwrite the original symbol with the modified version. A better option is to save the modified symbol in a new library file. I prefer this option. Click on the Export button (), or select Symbol, 'Export Symbol' from the menu. Navigate to your libraries directory, type in a new name for the new symbol and click Ok. I simply added my initials after the original name of the symbol (Figure 36.4).

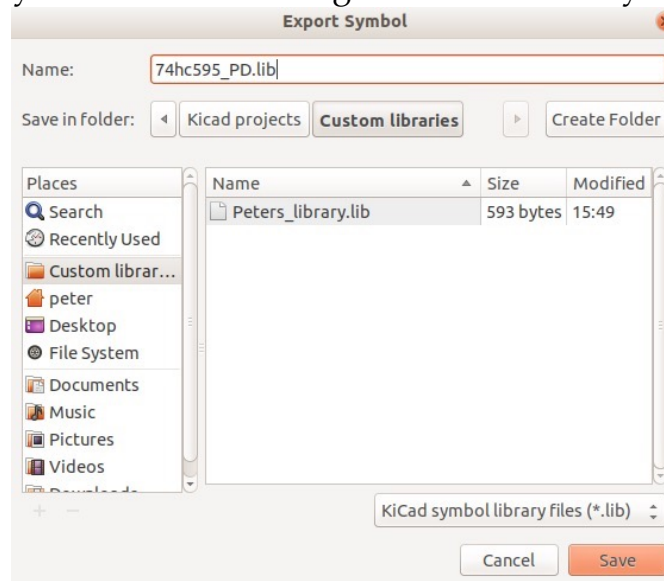


Figure 36.4: Saving the modified symbol.

Finally, let's test your modified symbol. In Eeschema, add the new symbol library to your libraries table if it isn't there already. Add a new symbol to the sheet, which will reveal the symbol chooser window. Use the filter to search for the filter, by typing '74HC595' in the field (Figure 36.5).

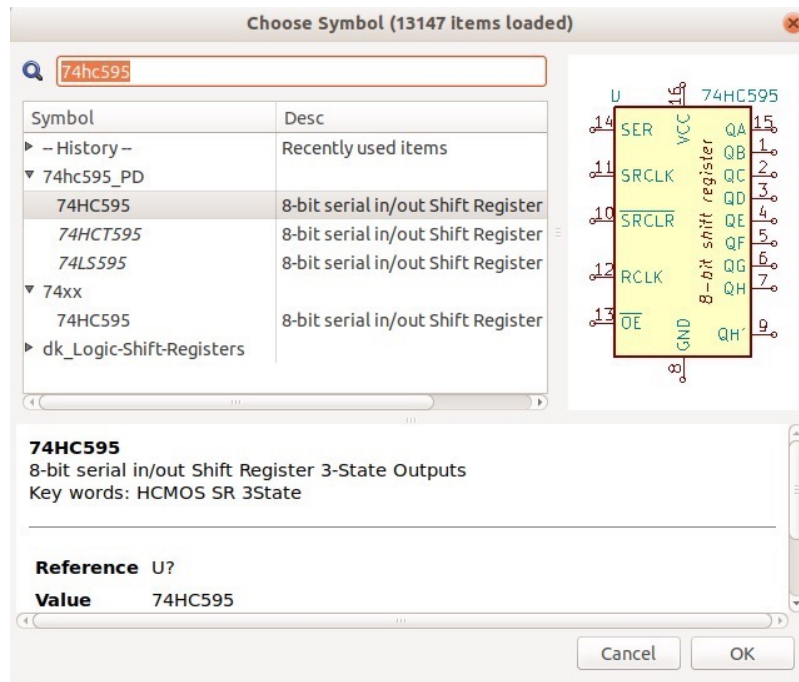


Figure 36.5: The modifier symbol appears in the symbol chooser results.

Notice that there are three items inside the '74hc595_PD' library. Apart from the actual symbol name, there are two more: '74HCT595' and '74LS595'. These are aliases, pointing to the same symbol. To edit the aliases, use the symbol properties window from the Library Editor's 'Symbol' menu.

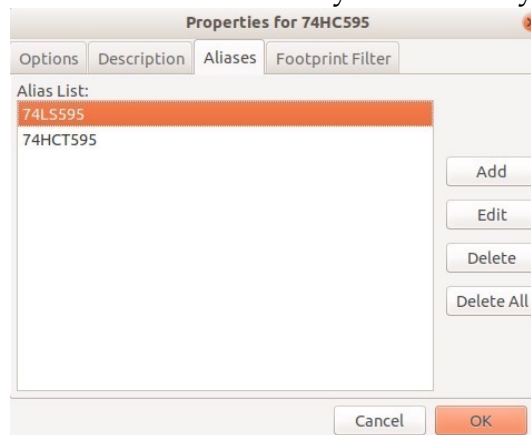


Figure 36.6: Symbols can have aliases, configurable from the Properties window.

Double click on any of the symbol aliases to add the modified footprint to the sheet. The example in fig shows the original symbol on the left, and the modified version on the right (Figure 36.7).

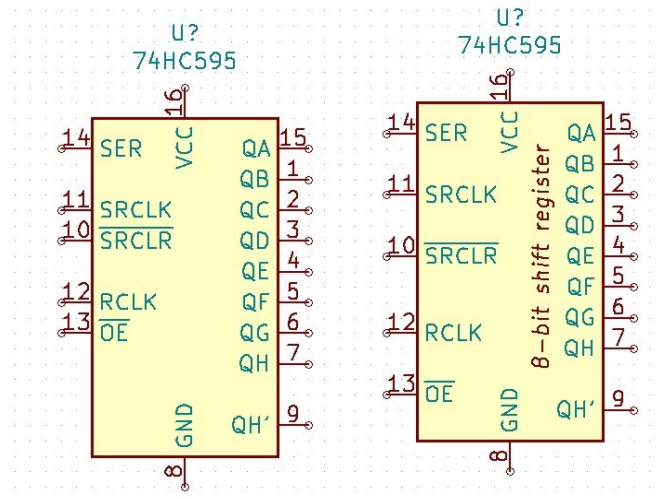


Figure 36.7: The original (left) and modified (right) symbols.

37. Creating a new footprint - manually

After creating a new schematic symbol, you may need to create a matching footprint. In this recipe, you will learn how to do this manually. For some types of footprints, KiCad has introduced a wizard that can accelerate the process. There is a separate recipe that covers that option.

As with creating a new symbol, to create a new custom footprint you will need some mechanical information from the real-world component's datasheet. If you don't have a datasheet, you can use a calliper to take measurements. I usually use both datasheet and calliper together. The datasheet gives me the mechanical values I need for the drawing of the component (width and height of the package, pins and their positions, pin attributes etc.), and then I use the calliper to confirm the dimension and distance values.

In recipe 'Creating a new component (symbol)' you created a custom symbol for the 555 integrated circuit. In this recipe, you will create a footprint for that symbol.

Creating footprints is a slightly more involved process than creating a symbol because we have to consider the physical characteristics of the device, and the manufacturing requirements. We have to think about how to place information about the footprint in the various physical and design layers that KiCad uses for this purpose. For example, information about the boundary of the footprint is placed in the courtyard layer, pads are placed in the copper layers (front and back), the outline of the footprint goes in the fabrication layer, and any artwork (text and graphics) goes to the silkscreen. In Figure 37.1 you can see the elements that make up a typical KiCad footprint, and the layers where those elements are placed. The footprint in Figure 37.1 is what you will work towards creating in this recipe, even though it already exists in a library that you can import.

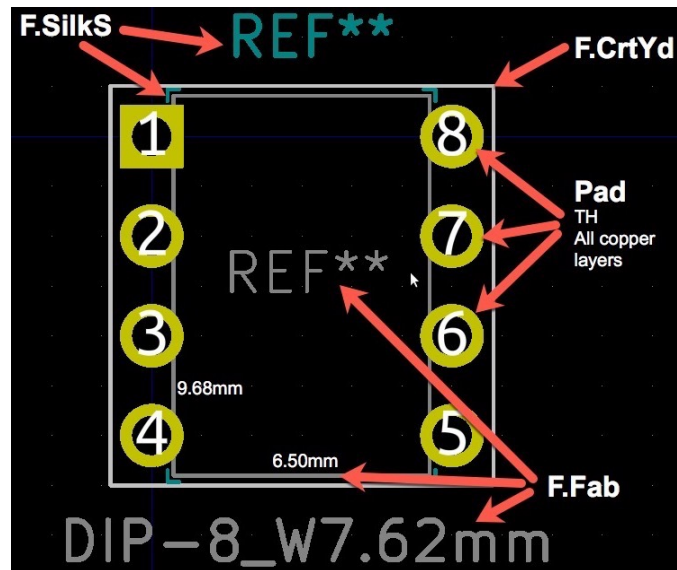


Figure 37.1: The elements and layers of a typical footprint.

To create the footprint in Figure 37.1, you will work through a process in which you place the elements in one layer. The real component for which you will design this footprint is shown in Figure 37.2.

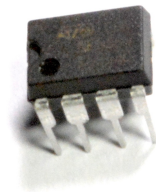


Figure 37.2: You will create a footprint for this component, the NE555N timer integrated circuit.

The component in Figure 37.2 has a standard DIP package with 4 pins. The data sheet is available from its [manufacturer](#). From the datasheet, you will need the mechanical data illustration towards the end of the document. For your convenience, I have included this illustration in Figure 37.2.

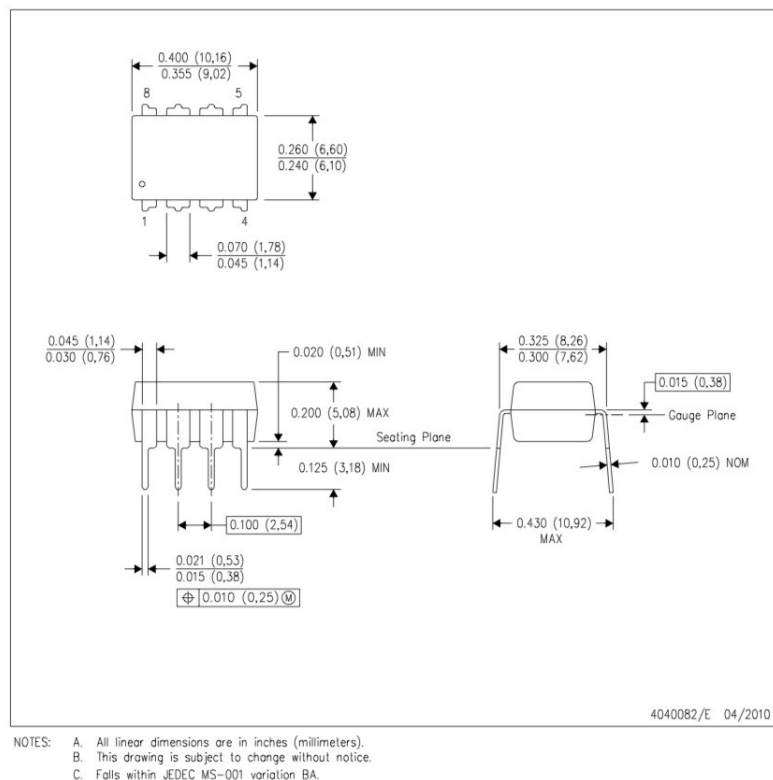


Figure 37.2: The mechanical characteristics of the R-DIP-T8 package.

You will create this footprint by following this process:

1. In the front fabrication layer ('F.Fab') you will draw the outline of the footprint. The fabrication layers (front and back) are used by the manufacturer. Their elements do not appear in the end result.
2. In the top and bottom copper layers, you will draw the pads.
3. In the front courtyard layer ('F.CrtYd'), you will draw the external outline of the footprint. No other footprint will be allowed within this outline.
4. In the front silkscreen layer ('F.Silks'), you will add text and graphics, such as the corners of the DIP package and the side where pin 1 is facing.

Let's begin. From the main KiCad window, click on the Footprint Library Editor button (Figure 37.3).

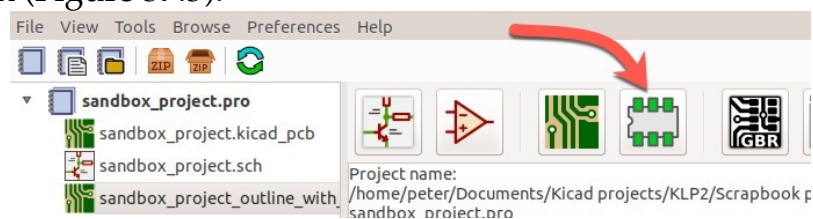


Figure 37.3: Start the Footprint Library Editor.

The Footprint Editor window will appear. From the File menu, select 'New Footprint' (Figure 37.4).

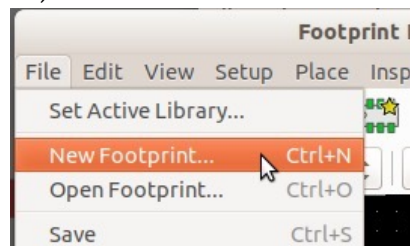


Figure 37.4: Create a new Footprint.

The Editor will prompt for a name for the new footprint. Since we are creating a footprint for a DIP component with 8 pins and 7.62mm width (including the pins), type in this name: 'DIP-8_W7.62mm_PD'. I added my initials, to differentiate from other DIP-8 footprints that might be available in other libraries. Of course, use your own initials. Click on Ok to dismiss the dialog. The Editor will show an almost blank sheet. Its only content is two text blocks, one in the front fabrication area (the name of the footprint), and one in the front silkscreen area ('REF**').

Front Fabrication layer ('F.Fab') - Outline

Continue with the front fabrication layer. Select 'F.Fab' from the layers manager, and use the polygon tool to draw a rectangle in the dimensions of the DIP package. Those dimensions appear in the documentation. Working in millimetres, you should draw a rectangle that is 6.60 mm in width and 10.16 mm in length. You will need to adjust the grid to make it easier to achieve those dimensions, or at least as close to them as you can get. I set my grid to 0.1270 mm and I was able to create a rectangle with the exact 6.60 mm x 10.16 mm dimensions. Also, change the cursor shape so that the crosshairs extend to the edges of the sheet, and use the dx and dy values in the status line to know the length of each segment of the rectangle as you draw it. In Figure 37.5, the arrows point to the tools that you use and functions you should enable to draw the outline of the footprint. The exact point where you start drawing is not important.

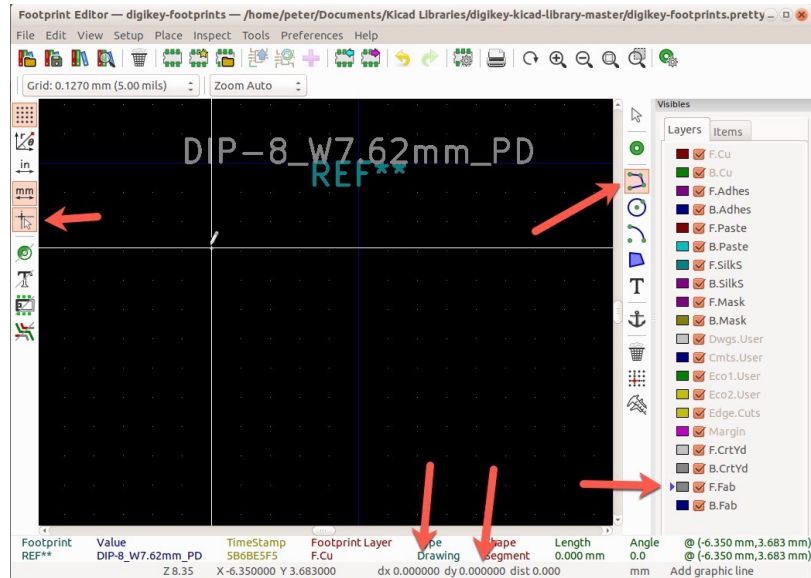


Figure 37.5: Using the polygon tool, start drawing the rectangle.

Draw the first horizontal line of the outline to a length of 6.60 mm. In Figure 37.6 notice that the dx value is 6.60 mm. Click at that point, and continue with the first vertical line.

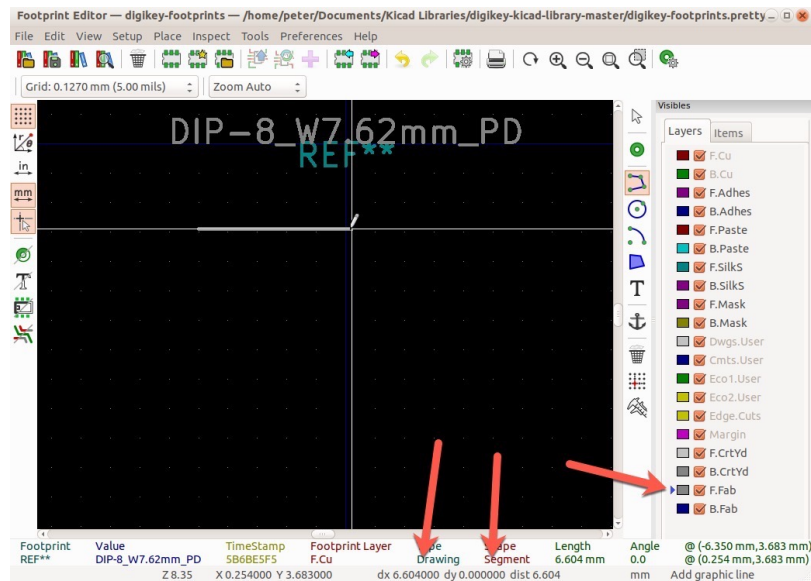


Figure 37.6: The first line, at 6.60 mm, is complete.

Extend the vertical line to 10.16 mm. Use the crosshairs to help you create a perfect 90-degree angle between the two lines (Figure 37.7).

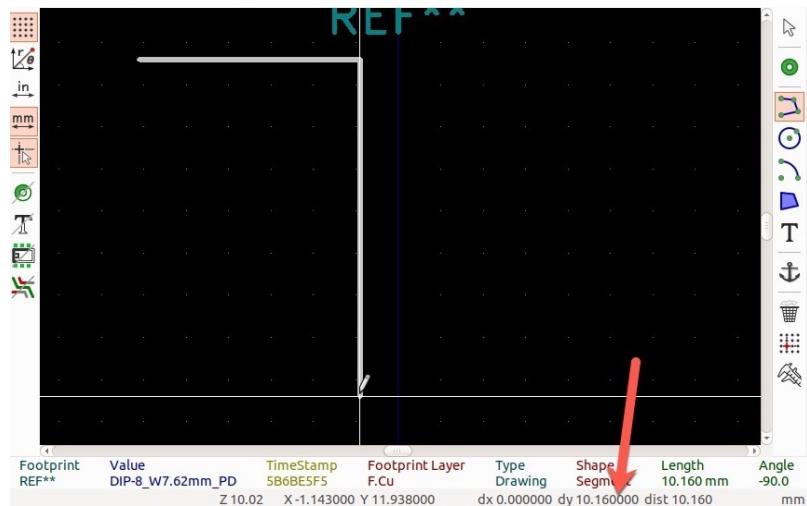


Figure 37.7: The first vertical line is 10.16 mm in length.

Complete the rectangle so that in the end you have something like the example in Figure 37.8.

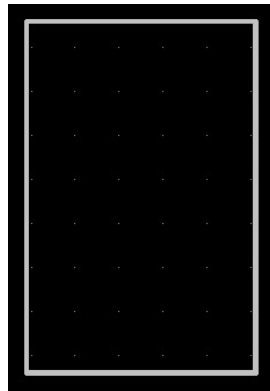



Figure 37.8: The completed outline in the F.Fab layer.

Pads

Continue with the placement of the pads. As per the data sheet's mechanical specifications, the pad centres must be 2.54 mm apart. The pin diameter is 0.25 mm, so the pad drill must be slightly larger than that. The specification doesn't give us the exact offset of the pins from the body of the device, so we will have to use our judgement or a calliper. I used my calliper to find that the offset is around 1 mm. With this information, let's go ahead and place the pads. Keep the grid size to 0.127 mm since this works well with the 2.54 pad pitch we are working with. That is because 2.54 is a multiple of 0.127.

- From the right toolbar, click on the pad tool (). Follow these steps:
1. Move the pad to the top left of the outline.
 2. Align the crosshair centre exactly on the vertical line, around 1 mm from the corner.
 3. Press the space bar to zero the dx , dy and $dist$ values of the status

bar.

4. Move the pointer slightly to the left, to separate the pad from the outline. At dx at 0.63 mm, I think this distance is good. It is smaller than the 1 mm that I measured with my calliper, but since the device pins are flexible, I opt to a narrower than a wider footprint.

5. Ensure that dx is still at 0.63 mm and click to commit the pad in place. After clicking your mouse, do not move it until you press the space bar again to reset dx and dy . This will allow you to measure the distance between this pad and the next. Don't worry about the negative sign in the dx value, we are only interested in absolute values when we measure distances for the purposes of creating this new footprint.

The result is similar to what you can see in Figure 37.9.

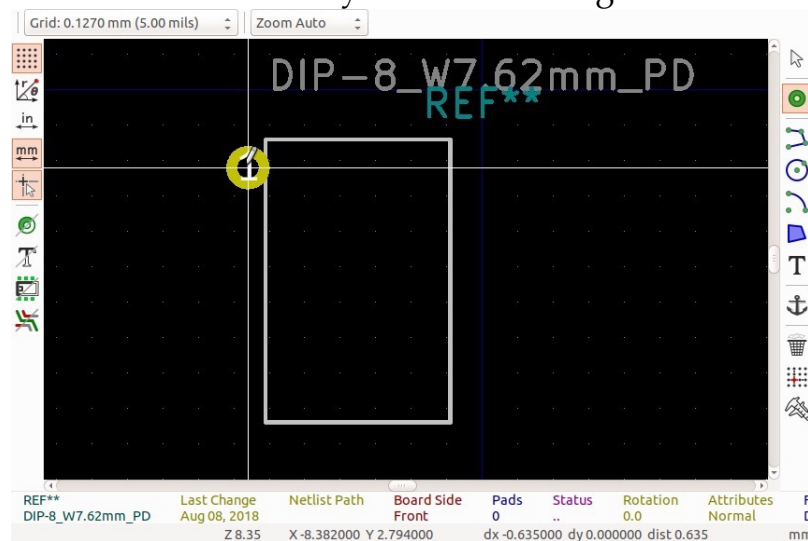


Figure 37.9: The first pad is in place.

The pad tool is still enabled, and the second pad is attached to the cursor, waiting to be placed. Hopefully, you reset dx and dy as soon as you clicked to commit pad 1. If you didn't, move the cursor over pad 1, and press the spacebar to reset the counters. Then, follow this process:

1. Move the mouse downwards making sure that dx is always zero.
2. Look at the dy counter, and stop moving the mouse when it reads '2.54 mm'.
3. When dy becomes 2.54 mm, click your mouse to place pad 2 in position.
4. Press the space bar to reset dy to zero.
5. Repeat this processes until you have all four pads on the left of the footprint's outline.

When you place pad 4 in position, you must mouse the mouse across to the other side of the outline. To ensure that pad 5 is exactly horizontally across

pad 4, place your mouse pointer exactly over pad 4, and press the spacebar. Then move the mouse to the right, making sure that dy is always zero. Place pad 5 in position so that it is exactly 0.63 mm from the right vertical line and click to commit it in place. Your footprint should look like the example in Figure 37.10.

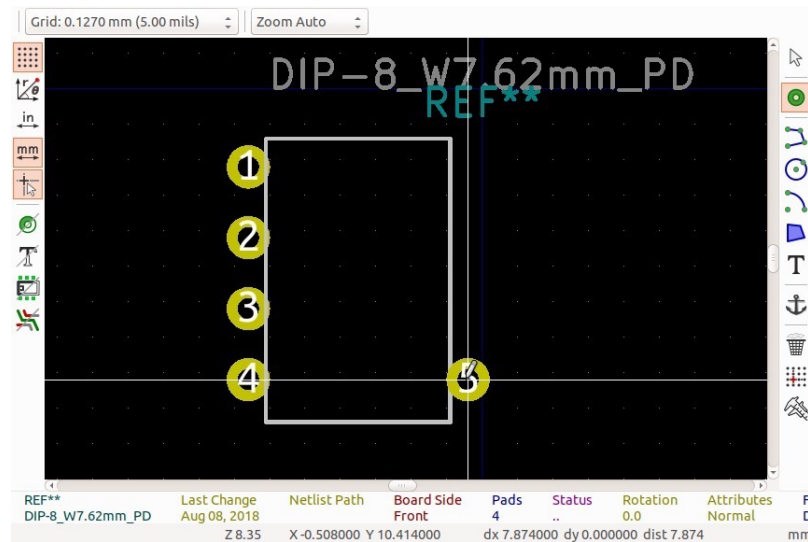


Figure 37.10: Left side has four pads, continue on the right side.

Continue to place pads 6, 7 and 8 in the same way. Eventually, your footprint will look like the example in Figure 37.11.

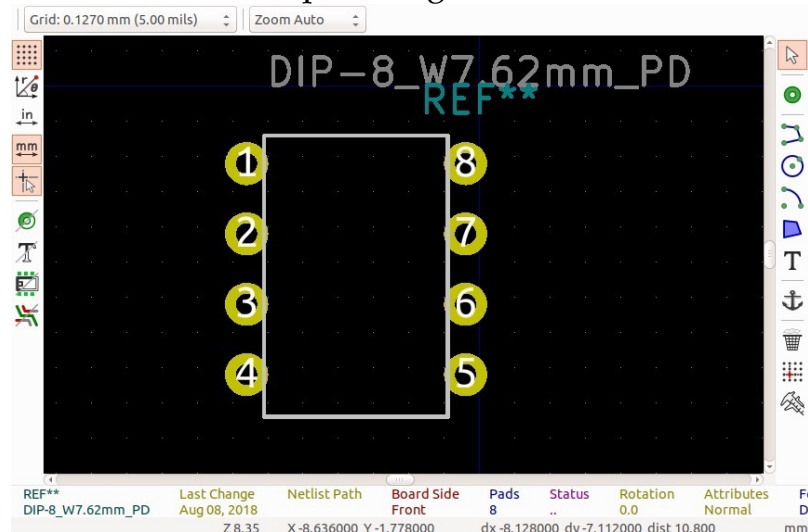


Figure 37.11: All footprints in place.

Before you continue work in the other layers, you should check that the footprint drills are appropriate for the pin size of the physical device. You should also change the pad type of pad 1 to rectangular instead of circular. This is due to a convention that holds that the first pin of an integrated circuit should be square to make it possible to identify the correct orientation of the chip during assembly. We will do this now, but we will also add graphics in the silkscreen to reduce the risk of error in assembly.

To configure a pad, use the pad's Properties window. Place your mouse cursor over the first pad, and type 'E'. This will bring up the pad's Properties window. As you can see in Figure 37.12, you should change the Pad shape to rectangular. You can also confirm that the hole size is 0.762 mm, which is sufficiently larger than the pin diameter, so there is no need to change this. You can also confirm that for this pad, copper will be poured in all copper layers. If you have a good reason for doing so (perhaps you are designing a single layer board that you want to etch at home?), then you can change this setting to a bottom copper layer. If you were creating an SMD pad, you could specify the top or bottom copper layer.

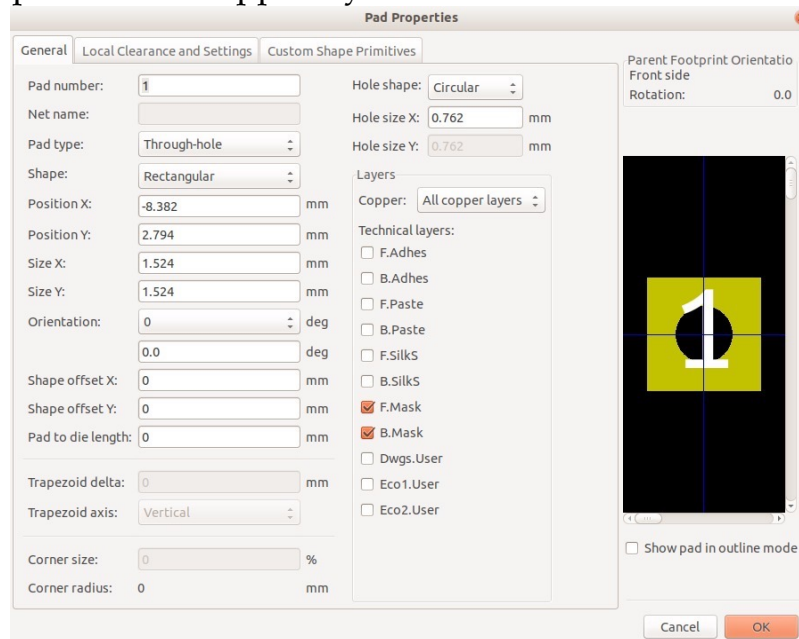


Figure 37.12: Pad 1 is now a square.

You can leave the rest of the pads as they are, no changes are needed. Your footprint should look like the example in Figure 37.12.

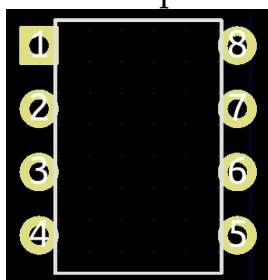


Figure 37.12: The footprint, with fabrication layer and pads completed.

Front Courtyard layer ('F.CrtYd')

Let's continue with the front courtyard layer ('F.CrtYd') where you will create the outline of the boundary for the footprint. From the Layer Manager, select 'F.CrtYd'. Your objective is to draw a rectangle around the footprint that encloses the pads and the footprint outline. Allow sufficient space around the

pads to ensure that they cannot overlap with other footprints. Select the polygon drawing tool and start drawing from the top right corner of the footprint (Figure 37.13).

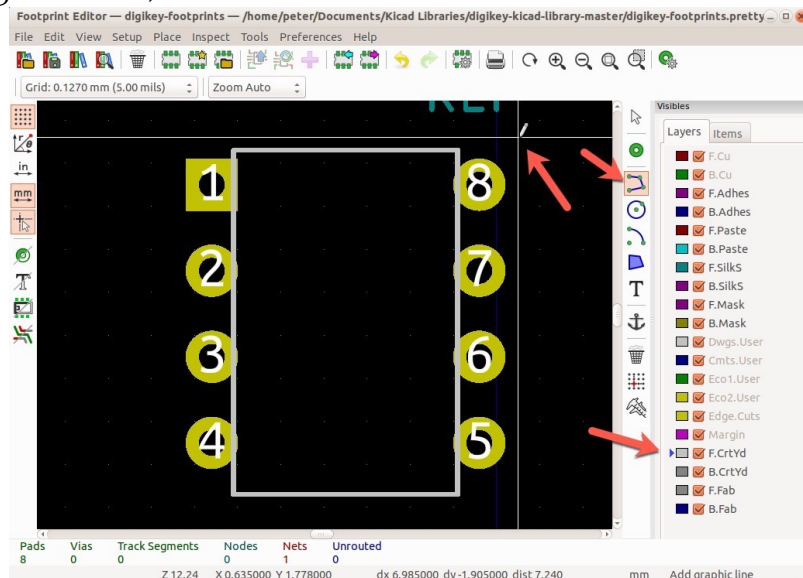


Figure 37.13: Drawing in the front courtyard layer.

Draw the rectangle around the footprint until the polygon is fully enclosed. In the end, your footprint should look like the example in Figure 1. The rectangle around the footprint is the boundary as defined in the front courtyard layer.

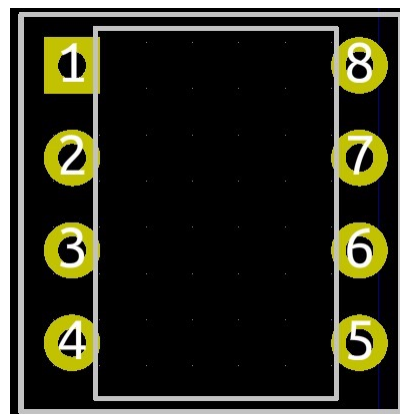


Figure 37.14: The line around the footprint is the front courtyard outline.

Front Silkscreen

While your footprint is now functionally ready, you should spend a few more minutes to add informational text and graphics in the front silkscreen ('F.Silks') layer. Since you are working on the footprint of an integrated circuit, at the very least you should mark the location of pin 1. Let's do that now. Select 'F.Silks' from the Layer Manager. Select the polygon tool. Draw four lines to mark the outline of the IC on the board, like in the example of Figure 37.15.

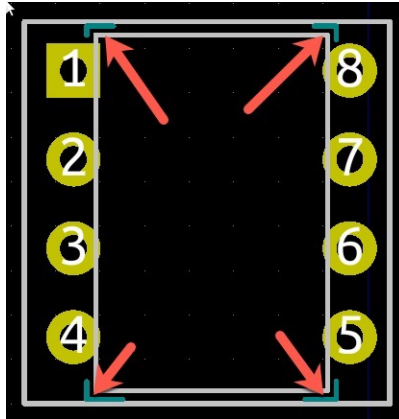


Figure 37.15: In the F.SilkS layer, mark the edges of the footprint.

Also draw a circle that indicates the position of pin 1, using the circle tool. The end result is in Figure 37.16.

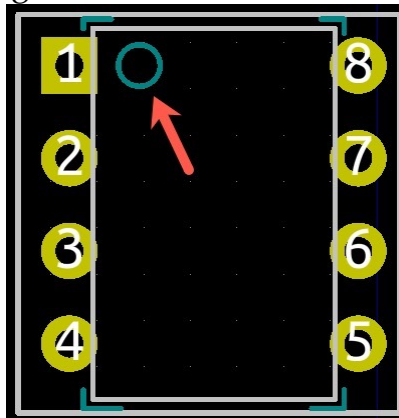


Figure 37.16: A circle in the F.SilkS layer indicates the position of pin 1.

Tidy up

You are almost finished with this design. Time to tidy up before saving the new footprint. Use the 'M' hotkey to move the text blocks over and below the footprint. Select the complete footprint, including the text block, and move it over the center of the axes. The final footprint is in Figure 37.17.

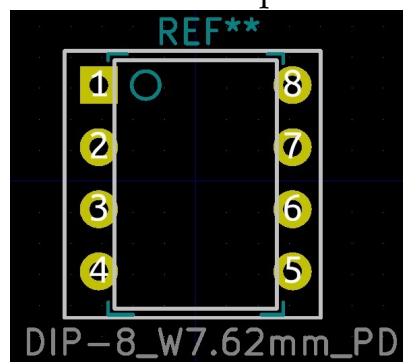


Figure 37.17: The final custom footprint.

Save the footprint

The last thing you must do before you can use your custom footprint is to save it. If you already have a folder for your custom footprints with the

'pretty' extension, you can use that to store the new footprint. If not, create this folder now. Then, click on the 'Export' button from the top toolbar (Figure 37.18).

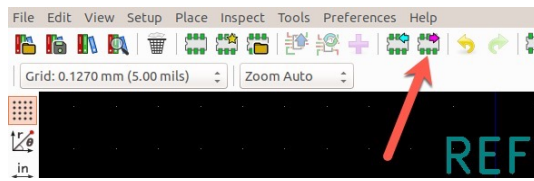


Figure 37.18: Use the Export button to save the new footprint.

This will bring up the Export Footprint window. Navigate to your .pretty folder, give your new footprint a name (or accept the default, as I do), and click on Save (Figure 37.19).

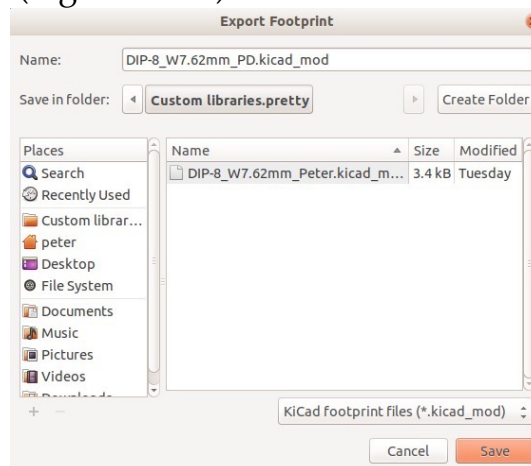


Figure 37.19: Save the new footprint.

Test the footprint

Let's go back to Pcbnew to test your new footprint by adding it to the sheet. If the .pretty folder that contains your new footprint is not in the libraries table, add it now (Figure 37.20). You can learn how to do this in the relevant recipe.

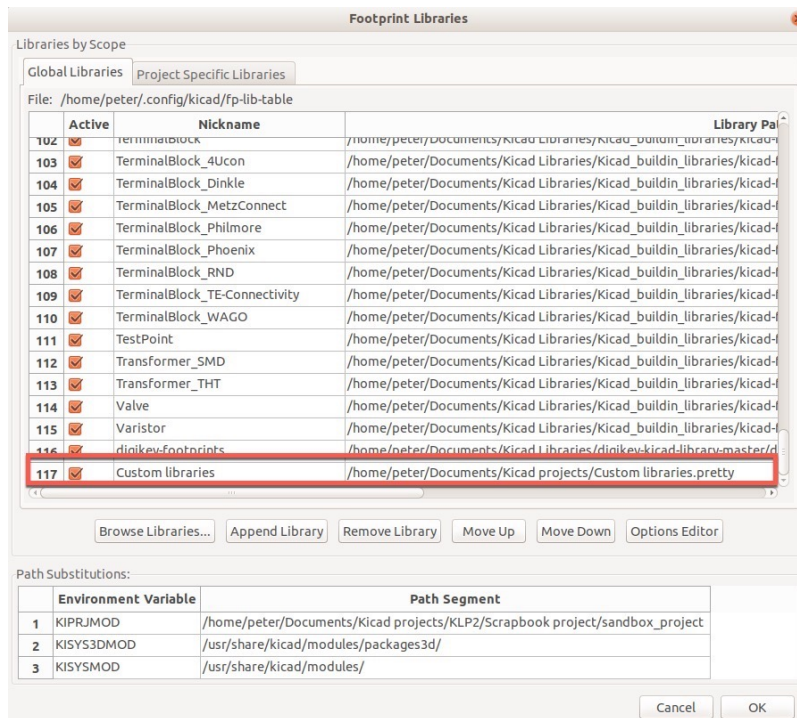


Figure 37.20: My custom .pretty folder is in the libraries table.

Once that is done, use the 'O' hotkey to add a new footprint. Use the browser to navigate the list of libraries. Find your custom library, and in it, you will find your new footprint (Figure 37.21).

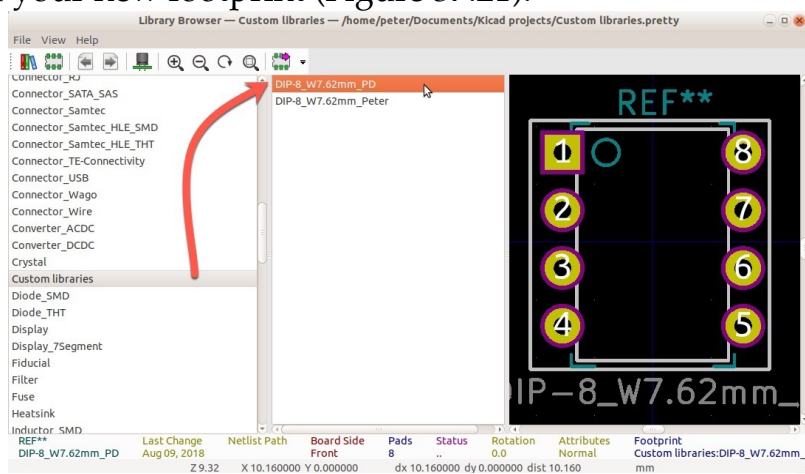


Figure 37.21: Select your new footprint from the library browser.

Double click on the footprint, and Pcbnew will place it on the sheet (Figure 37.22).

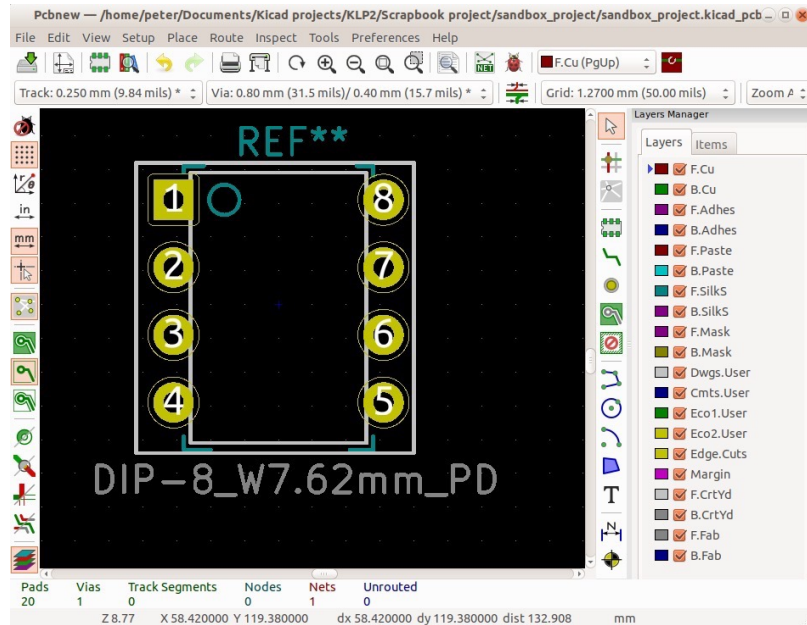


Figure 37.22: Your new footprint in the Pcbnew sheet.

Well done, this was a long process. You now have a custom footprint that you can use in your layouts as you do with any other footprint.

39. Modifying an existing footprint

In this recipe, you will learn how to modify an existing footprint. You may want to do something simple, like relocate silkscreen graphics, or change the drill size of the pads. Whichever the case may be, the process is the same.

Start by finding a footprint to edit. Let's assume that you know which library this footprint is in, and you know its name. If you don't, use the footprint browser in Pcbnew to find it first.

From the main KiCad window, start the Footprint Manager. In the Footprint Manager, use the 'Import footprint' button to find and import the footprint (Figure 39.1). If the Editor already contains a footprint, you will get a prompt asking for confirmation to discard it. Consider saving it if you haven't done so already, and continue.

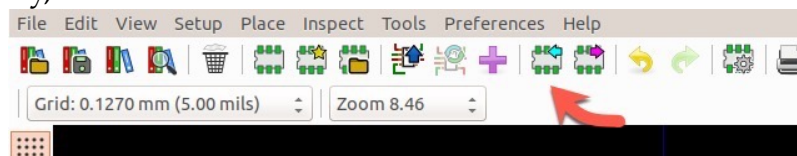


Figure 39.1: Import the footprint you want to edit.

In the browser, navigate to the .pretty folder where the footprint is, and select it. In my example, I navigated to the location of the Digikey footprints directory, and I have selected to modify the LED_3mm_Radial footprint (Figure 39.2).

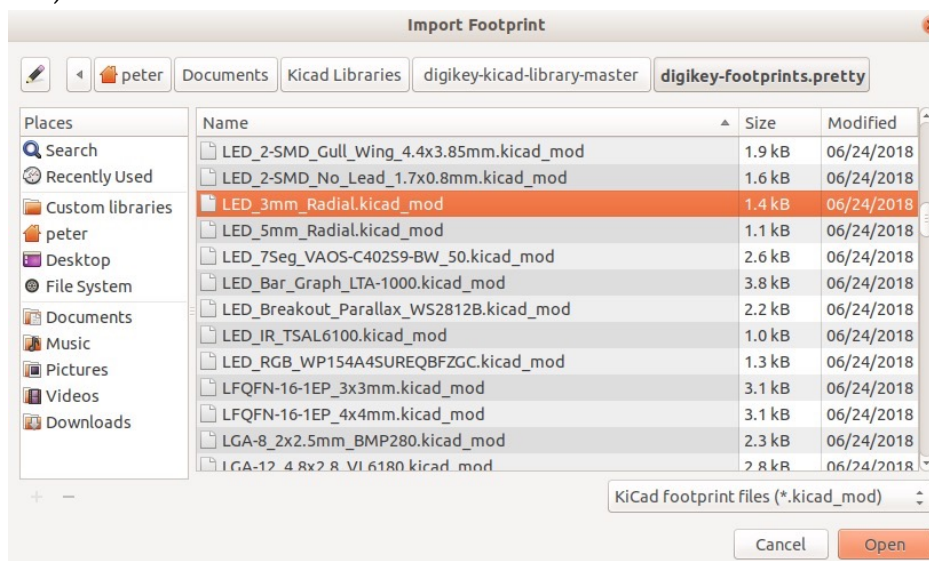


Figure 39.2: Navigate to the footprint's location and select it.

Click 'Open' to import the footprint. The footprint will appear in the Editor sheet. Let's make a couple of simple modifications:

1. Add the block text 'LED' in the silkscreen.
2. Mark the cathode pad with the letter 'C' in the silkscreen.
3. Reduce the drill size to 0.8 mm for both pads.

Let's start with the text. Select the 'F.SilkS' layer from the Layers Manager. Select the Text tool. Click on the right side of the LED footprint; the text properties will appear. Type 'C' in the text box and click on 'Ok'. Position the text block right next to the symbol cathode (the straight section of the circle), and then click below the footprint to create the second text block. Type 'LED' in the text box, and click 'Ok' to create the new block. Click on the sheet to commit the new block.

Continue with the pad drill size. Hover the mouse pointer over pad 1 and type 'E' to bring up the pad properties. In the pad properties window, change the hole size X value to 0.8 mm (Figure 39.3). Then click 'Ok' to make the change effective.

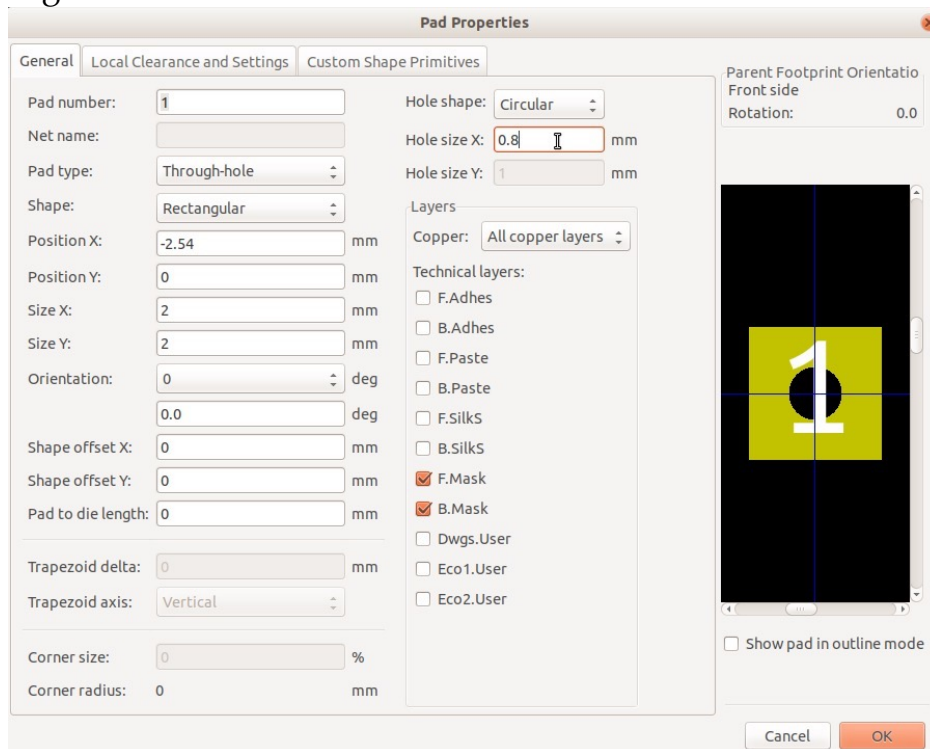


Figure 39.3: Modify the drill size for pad 1.

Do the same for pad 2. The modified footprint should look like the example in Figure 39.4.

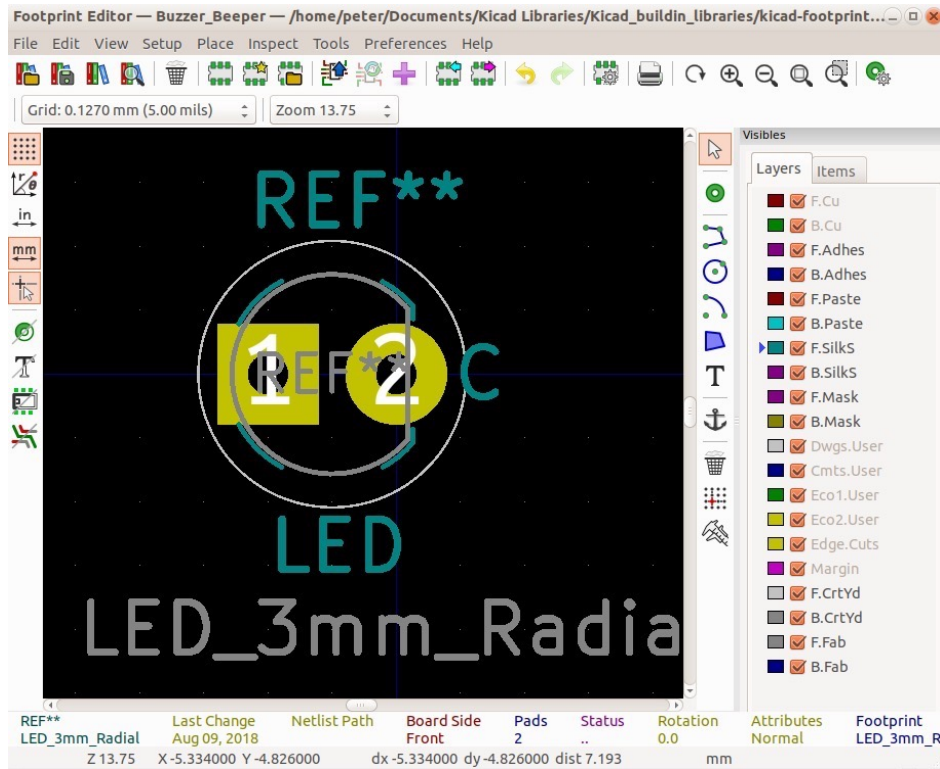


Figure 39.4: The modified footprint.

Before you can use the modified footprint you must save it. Use the Export Footprint button (the one with the red arrow) to do this. Navigate to the location of your custom footprints .pretty folder, and store the modified footprint there (Figure 39.5).

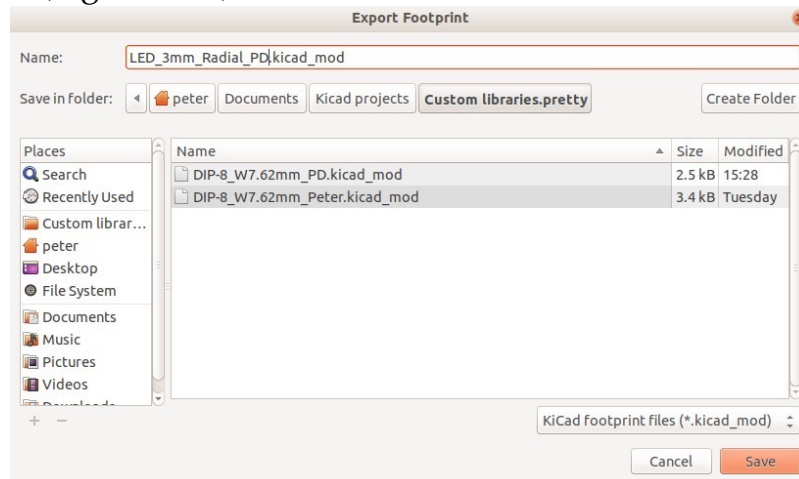


Figure 39.5: Store the modified footprint in the custom footprints folder.

You are now able to use your modified footprint in your layouts.

41. How to create a bill of materials (BoM)

In KiCad, you can create a Bill of Materials (BOM) report via third-party plugins. In this recipe, you will learn how to use one of the plugins to create a BOM in CSV format, like the one in Figure 41.1.

Component	Description	Part	References	Value	Footprint	Quantity	Per	Datasheet	Category
1	Battery (multiple cells)	Battery	BT1	Battery 3V	PinHeader_1x02_P2.54mm_Vertical	1	~		
2	Polarized capacitor	CP1	C3	10uF	C_0805_2012Metric_Pad1.15x1.40mm_HandSolder	1	~		
3	Unpolarized capacitor	C	C1 C2	22pF	C_0805_2012Metric_Pad1.15x1.40mm_HandSolder	2	~		
4	LED generic	LED	D1	LED	LED_0805_2012Metric_Pad1.15x1.40mm_HandSolder	1	~		
5	Generic connector, single row, 01x09, script generated (kicad-library-utils/sc Conn_01x09_J4)			DigitalPins	PinHeader_1x09_P2.54mm_Vertical	1	~		
6	Generic connector, single row, 01x04, script generated (kicad-library-utils/sc Conn_01x04_J1)			I2C connector	PinHeader_1x04_P2.54mm_Vertical	1	~		
7	Generic connector, double row, 02x03, odd/even pin numbering scheme (row Conn_02x03_J2)			ICSP connector	PinHeader_2x03_P2.54mm_Vertical	1	~		
8	Generic connector, single row, 01x04, script generated (kicad-library-utils/sc Conn_01x04_J3)			Serial connector	PinHeader_1x04_P2.54mm_Vertical	1	~		
9	Resistor	R	R2	220Ohm	R_0805_2012Metric_Pad1.15x1.40mm_HandSolder	1	~		
10	Resistor	R	R1	3300hm	R_0805_2012Metric_Pad1.15x1.40mm_HandSolder	1	~		
11	I2C Serial EEPROM, 1024Kb, DIP-8/SOIC-8/TSSOP-8/DFN-8	24LC1025	U1 U3	24LC1025	S01-8_5.3x5.3mm_P1.27mm	2		http://ww1.microchip.com	
12	IC MCU 8BIT 32KB FLASH 32TQFP	ATMEGA328	U4	ATMEGA328P-AU	TQFP-32_7x7mm	1		http://www.integratedc	
13	Two pin crystal	D613375+	U2	D613375+	SO-8_5.3x6.2mm_P1.27mm	1			
14	Two pin crystal	Crystal	Y2	16 Mhz	Crystal_SMD_5032-2Pin_5.0x3.2mm_HandSoldering	1	~		
15	Two pin crystal	Crystal	Y1	Crystal 32768khz	Crystal_SMD_MicroCrystal_CC7V-11A-2Pin_3.2x3.5mm_HandSoldering	1	~		
22	Component Groups:					15			
23	Component Count:					17			
24	Fitted Components:					17			
25	Number of PCBs:					1			
26	Total components:					17			
27	Schematic Version:					1			
28	Schematic Date:					19/8/18			
29	BoM Date:					Mon 20 Aug 2018 05:35:07 PM PDT			
30	Schematic Source:					/home/peter/Documents/Kicad projects/KLP2/Battery-powered Arduino with extended EEPROM/Battery-powered_Arduino_with_Extended_EEPROM/Battery-powered_Arduino_with_Extended_EEPROM.sch			
31	KiCad Version:					Eeschema 5.0.0-fee4fd166-ubuntu18.04.1			

Figure 41.1: A BoM created using the KiBoM Python script.

The plugin you will install is a Python script, available from Github. To install it, follow these steps:

1. Download the ZIP archive of the Github repository from <https://github.com/SchrodingersGat/KiBoM> (Look for the green 'Clone or download' button and select 'Download ZIP')
2. Extract the contents of the ZIP archive.
3. Copy the folder named 'KiBoM-master' to your KiCad documents folder, or anywhere you prefer to keep your work files.
4. Start KiCad, and open Eeschema.
5. Click on the BOM button from the top toolbar (Figure 41.2)
6. The Bill of Material window will appear. Click on the 'Add Plugin' button.
7. Navigate to the location of the KiBOM_CLI.py file and double-click on it to select it (Figure 41.3). You can accept the default name for the plugin.
8. The plugin is now installed. The Bill of Material window will show relevant information in the 'Plugin information' text box, and the command line path (Figure 41.4).

The command line path is of particular interest because you can control the output of the plugin by manipulating the command line arguments in it.

You can find information about the valid arguments and ways to customise the plugin output at the plugin [Github page](#).

Using the default configuration, however, produces a complete BoM. Go ahead and try it out. Load a schematic into Eeschema, and click on the BoM button. Click on the 'Generate' button. The plugin will produce the BOM file, and provide information about the work it did in the 'Plugin information' text box.

Browse to the project directory, and you will find the newly-created '.csv' file, that contains the BoM (Figure 41.5). You can open this file with a text editor or a spreadsheet application. In Figure 41.1, you can see the BoM for project 3 of this book as it appears in Microsoft Excel.



Figure 41.2: The BoM button.

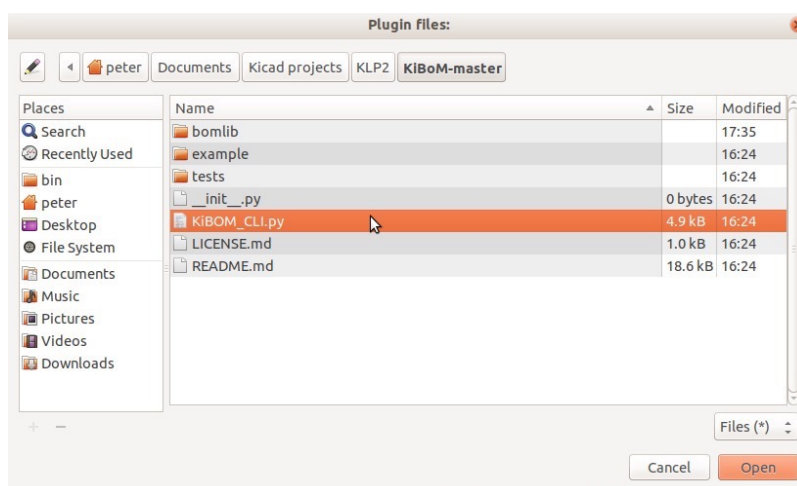


Figure 41.3: Navigate to the location of the KiBOM_CLI.py file.

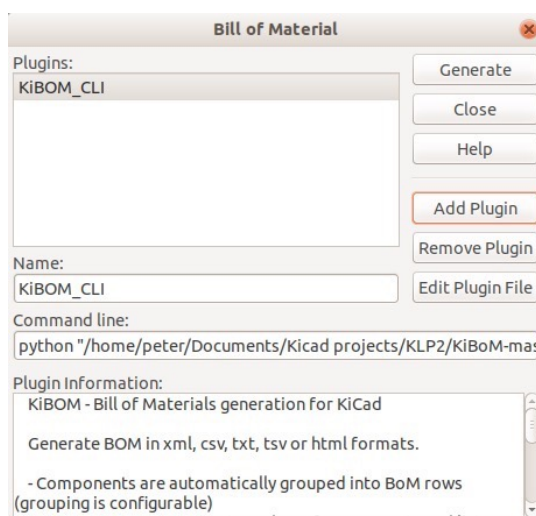


Figure 41.4: The BoM plugin is installed.

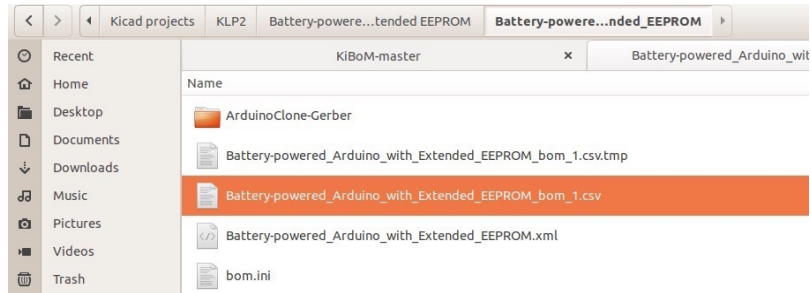
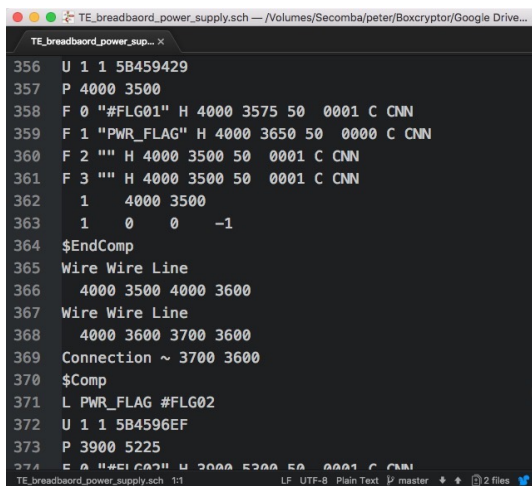


Figure 41.5: The newly-created CSV file that contains the BoM.

If you are interested in alternative plugin to implement the bill of materials feature in KiCad 5, I encourage you to have a look at KiCost (<https://pypi.org/project/kicost/>). As per the KiCost documentation, this plug-in generates part-cost spreadsheets for boards. This is very helpful if you want to make an accurate cost calculation for your project. The plugin can also generate an XML file that you can upload to Digi-Key or Mouser to quickly order the project parts.

47. Using Git for version control

KiCad projects consist of files that contain plain text. You can use any text editor to open them and have a look inside. For example, in Figure 47.1 you can see a section of the schematic '.sch' file for one of my projects, opened using the Atom text editor.



```
TE_breadbaord_power_supply.sch — /Volumes/Secomba/peter/Boxcryptor/Google Drive...
TE_breadbaord_power_sup... x
356 U 1 1 5B459429
357 P 4000 3500
358 F 0 "#FLG01" H 4000 3575 50 0001 C CNN
359 F 1 "PWR_FLAG" H 4000 3650 50 0000 C CNN
360 F 2 "" H 4000 3500 50 0001 C CNN
361 F 3 "" H 4000 3500 50 0001 C CNN
362 1 4000 3500
363 1 0 0 -1
364 $EndComp
365 Wire Wire Line
366 4000 3500 4000 3600
367 Wire Wire Line
368 4000 3600 3700 3600
369 Connection ~ 3700 3600
370 $Comp
371 L PWR_FLAG #FLG02
372 U 1 1 5B4596EF
373 P 3900 5225
374 E 0 "#FLG02" H 4000 5300 50 0001 C CNN
TE_breadbaord_power_supply.sch 1:1 LF UTF-8 Plain Text master + 2 files
```

Figure 47.1: Import the DXF R12 file into Pcbnew.

The fact that KiCad's projects are text files makes it easy to use a versioning system, like Git, to keep track of all changes made across the project. There are several popular versioning systems available, but my personal preference is Git. It is open source, fast, very popular, and extremely versatile. As you will see in this recipe, it is also very easy to use.

What you will learn in this recipe is how to use Git, and the Github online repository, to maintain your project's history. Doing so will allow you to:

1. Preserve your project's history. This will allow you to access past versions of any file in your project.
2. Create experimental branches. This is useful if you want to experiment with alternate design options, or design different versions of the same board. Each one can be stored in a separate branch of the same repository.
3. Merge or discard different branches. This Git function allows you to merge (unify) two branches into one. For example, you may have the main branch of your project, and go on to work on an experimental branch as you are investigating a special board feature. If the experiment succeeds, you can merge the experimental branch to the main branch, and continue from there.

If not, you can just discard the failed experiment branch, and continue with the intact project in the main branch.

These are just three of the many possible scenarios. Those are the three scenarios that I use most often.

If you use Git alongside an online repository, like Github, you will be able to use this versioning system to collaborate with other people on the same project. You will also be able to share your project and its history with other people. In Figure 47.2, you can see part of the history of one of the project in this book, as it appears in the publicly-accessible repository on Github.

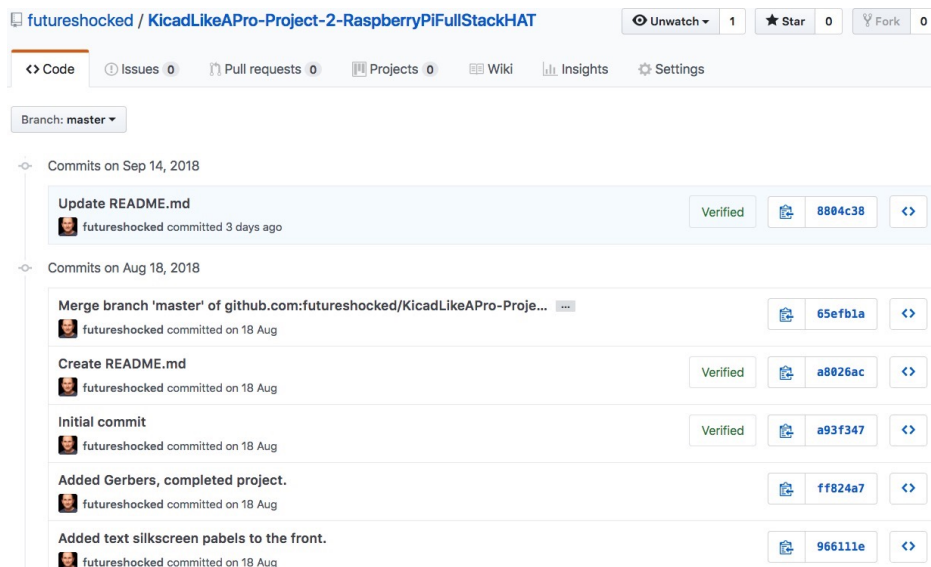


Figure 47.2: Part of a history of one of the projects in this book on Github.

In Figure 47.3, you can see the history of the schematic file for the same project.

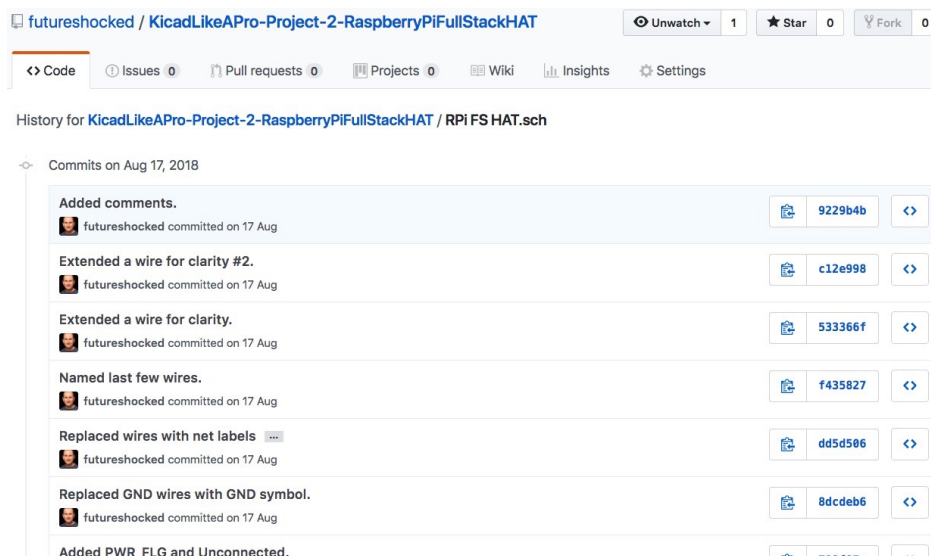


Figure 47.3: Part of the history of the schematic file on Github.

The numbers in the right side of each column are called 'commits'. Each

number is an ID that refers to a commit. The full ID of a commit is a long alphanumeric, like 'f0266d1343a446d068e874f46cf9cc29'. What you see in Figure 47.3 is a short hash of that ID. A commit may contain changes in multiple files, or additions and removals of files. To get detailed information about a commit, you can click on the commit number. The result is a side-by-side comparison of the changes detected in each file, as in the example of Figure 47.4.



Figure 47.4: The changes detected in file 'RPi FS HAT.sch' at commit.

In this tutorial I will give you a basic introduction of Git and Github. Git is a big topic, and I do encourage you to learn more about it by using a specialised source, like this [Getting Started](#) guide from Github.

Read on to learn how to use Git and Github in the context of a KiCad project. You will learn how to:

1. Create the repository on your computer.
2. Commit changes of your project to the repository.
3. See those changes in the log.
4. Checkout past commits.
5. Create branches.
6. Upload your project to Github so you can share it with other people.

Start by installing Git on your computer. For each operating system, you can find detailed instructions on the [Git web site](#). Here, I will show you how to do this in Ubuntu or other Debian operating systems use to terminal:

```
$ sudo apt install git
```

Once the installation is complete, configure your git user settings:

```
$ git config --global user.name 'testuser'
```

```
$ git config --global user.email 'testuser@example.com'
```

From this point onwards, the instructions and the code is the same for all operating systems as long as you have access to the command line. Most of these operations are also possible using the [Git desktop app](#), if you prefer a graphical user interface.

You are now ready to use Git with your KiCad project. Use the terminal to browse into your KiCad project. At the root of the project, initialise the Git repository:

```
$ git init
```

You will get confirmation that your new repository is ready to use, something like this:

```
Initialized empty Git repository in /home/peter/Documents/KiCad
projects/KLP2/BreadboardPowerSupply_v2/.git/
```

Add the current project files into the repository:

```
$ git add .
$ git commit -am 'First commit.'
```

Again, Git will respond with confirmation. If you want to know what the current status of the repository is, use the 'status' command:

```
$ git status
On branch master
nothing to commit, working tree clean
```

Now lets say that you have made a change to the schematic diagram and that you wish to store this in the Git repository. Save your Eeschema file, and check the status of the repository:

```
$ git status
On branch master
Changes not staged for commit:
  (use 'git add <file>...' to update what will be committed)
  (use 'git checkout -- <file>...' to discard changes in
working directory)
```

```
    modified:   BreadboardPowerSupply_v2.bak
    modified:   BreadboardPowerSupply_v2.sch
```

```
no changes added to commit (use 'git add' and/or 'git commit -
a')
```

Git knows about the change, but it is not yet committed to the repository. To commit it, use the commit command:

```
$ git commit -am 'Added text label.'
[master 2f77b7b] Added text label.
 2 files changed, 183 insertions(+), 181 deletions(-)
rewrite BreadboardPowerSupply_v2.bak (68%)
```

Continue to do some more work in Eeschema, and save and commit this new work:

```
$ git commit -am 'Added 2nd text label.'
[master 74cabdc] Added 2nd text label.
 2 files changed, 4 insertions(+)
```

Git is keeping track of these commits in its log. If you want to know what is in the log, you can check with the log command:

```
$ git log
commit 74cabdc120196a703eb71c2290f016fa3b4b65eb (HEAD ->
master)
```

```
Author: Peter Dalmaris <peter@txplore.com>
Date: Sun Aug 12 19:52:07 2018 -0700
```

Added 2nd text label.

```
commit 2f77b7b68037436ed169797e9dbb8bd7678bd69a
Author: Peter Dalmaris <peter@txplore.com>
Date: Sun Aug 12 19:50:23 2018 -0700
```

Added text label.

```
commit 06d8909c726ae8f57ca456c4f9e0fc46e7b37fdd
Author: Peter Dalmaris <peter@txplore.com>
Date: Sun Aug 12 19:44:37 2018 -0700
```

First commit.

The newer commits appear on top. Each commit has an ID, that is useful if you want to operate on it. Let's say, for example, that you changed your mind about the latest change you made, and that you want to go back to a previous version of the Eeschema file. As long as you have the commit ID, you can do that using the checkout command:

```
$ git checkout 2f77b7b68037436ed169797e9dbb8bd7678bd69a
Note: checking out '2f77b7b68037436ed169797e9dbb8bd7678bd69a'.
```

```
You are in 'detached HEAD' state. You can look around, make
experimental
changes and commit them, and you can discard any commits you
make in this
state without impacting any branches by performing another
checkout.
```

```
If you want to create a new branch to retain commits you
create, you may
do so (now or later) by using -b with the checkout command
again. Example:
```

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 2f77b7b Added text label.
```

As an attribute to the checkout command you must provide the commit ID of the commit that you want to retrieve. Git will confirm that it retrieved this commit, and that in the case of this example, the Eeschema files have been restored to the earlier version. Close and reopen Eeschema and you will see that the schematic is, indeed, at its earlier version. The changes you committed since then still exist in the repository, its just that you are now at an earlier time in its history.

As Git is telling you, you are now in detached HEAD state. This simply means that you have rewinded your project to an earlier time. The commit on

which you are working on now is not the latest. You can work on this version and decide what to do next. If you are just looking around, then you can checkout HEAD when you are ready and continue work from the latest version:

```
$ git checkout master
```

If you choose to make changes, you can create a new branch and continue work there. You can create a new branch of your project like this:

```
$ git checkout -b new_branch
```

Where 'new_branch' is the name of the new branch. Continue to work in this branch as normal, saving your files and committing them to the repository. When you decide to return to the master branch, simply check it out:

```
$ git checkout master
```

Close and reopen your KiCad apps and you will see that their contents are updated accordingly. Continue to work in the new branch. The convention is that the master branch contains completed work. When you are ready, you will want to merge your experimental branch with the master branch. To do this, checkout the master branch, and then merge the experimental branch into the master.

```
$ git checkout master  
$ git merge new_branch
```

Sometimes, there are conflicts between the two versions of a file, and Git will not be able to automatically merge them. In that case, Git will let you know, like this:

```
Auto-merging BreadboardPowerSupply_v2.sch  
CONFLICT (content): Merge conflict in  
BreadboardPowerSupply_v2.sch  
Auto-merging BreadboardPowerSupply_v2.bak  
CONFLICT (content): Merge conflict in  
BreadboardPowerSupply_v2.bak  
Automatic merge failed; fix conflicts and then commit the  
result.
```

You will need to resolve the conflict by using a text editor and manually merging the files in question.

Git is a powerful tool for both saving your work at multiple points, and for experimenting. If used correctly, there is essentially no risk of losing work. Knowing that you can return to any point in the history of your project is truly liberating.

Another important benefit of using Git in your KiCad projects is that it is made for collaboration. You can share your repository with other people using a tool like [Github.com](https://github.com). If you work in a team, then the team members will be able to work on the same project at the same time, in their own branch,

and then merge their work into the main branch. This make it possible to use KiCad in teams, even though the KiCad itself does not have any collaboration capabilities.

Uploading your repository to Github

Suppose you would like to publish your project online so that other people can access it. You can do this easily by creating a remote Git repository on a service like Github. The repository on Github is a remote copy of your local repository. Every time you make a change to your local repository, you can push it to the remote so that your collaborators can access the updates. And vice-versa: if a change is accepted in the remote, you can pull it to your local repository so that you can use the changes.

Start by creating a repository on Github (Figure 47.5).

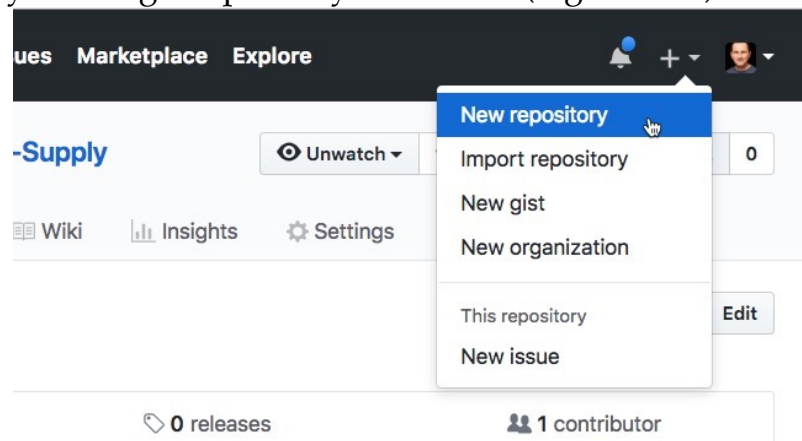


Figure 47.5: Create a new repository on Github.

Give it a name, a license type (I prefer the MIT license for sharing projects), and a description.

Next, go to your command line, and browse to the directory where your project is saved. I assume that you already have a local Git repository for your project in this directory. If not, follow the instructions in the first part of this chapter to create and populate one.

Go ahead to create the remote for your local project. A common name for the remote is 'origin', so let's use that in this example.

Using the command line, working inside the directory of your project, type:

```
$ git remote add origin git@github.com:futureshocked/  
KiCadLikeAPro-Project-Breadboard-Power-Supply.git
```

Replace the remote URL for the one of your project. You can find your remote URL from the repository page on Github (Figure 47.6).

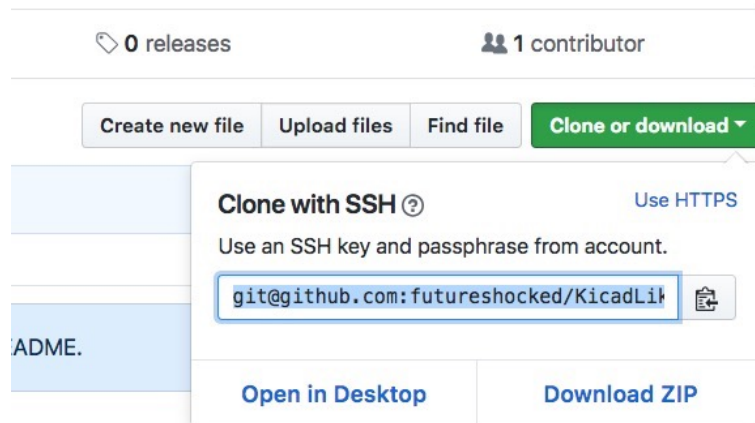


Figure 47.6: Find out your repository URL on Github.

You can confirm that the remote is set by typing:

```
$ git remote -v
origin git@github.com:futureshocked/KiCadLikeAPro-Project-
Breadboard-Power-Supply.git (fetch)
origin git@github.com:futureshocked/KiCadLikeAPro-Project-
Breadboard-Power-Supply.git (push)
```

Before you can 'push' (i.e. upload) your local repository to the remote for the first time, you will need to 'pull' (i.e. download). Pushing and pulling is not the same as uploading and downloading because both also include merging. With merging, the contents of the local and remote repositories are synchronised.

Go ahead to pull the remote to your local repository like this:

```
$ git pull origin master --allow-unrelated-histories
From github.com:futureshocked/KiCadLikeAPro-Project-
Breadboard-Power-Supply
 * branch          master      -> FETCH_HEAD
Merge made by the 'recursive' strategy.
 LICENSE | 21 ++++++
 1 file changed, 21 insertions(+)
 create mode 100644 LICENSE
```

Git will ask you for a comment using your default text editor. Type in something like 'First pull' and save the file to allow Git to complete the pull. For the first pull only, you will need to use the `--allow-unrelated-histories` since the two repositories are not related. From this moment onwards, you will be able to use 'git pull origin master' to pull from the remote.

Finally, go ahead to push your local repository to the remote. Do this:

```
$ git push origin master
Counting objects: 117, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (116/116), done.
Writing objects: 100% (117/117), 332.61 KiB | 2.75 MiB/s,
done.
```



```
Total 117 (delta 61), reused 0 (delta 0)
remote: Resolving deltas: 100% (61/61), done.
To github.com:futureshocked/KiCadLikeAPro-Project-Breadboard-
Power-Supply.git
   9428bca..30dfd14  master -> master
```

The local repository is now merged with the remote on Github. Refresh your web browser to see your project files on Github (Figure 47.7).

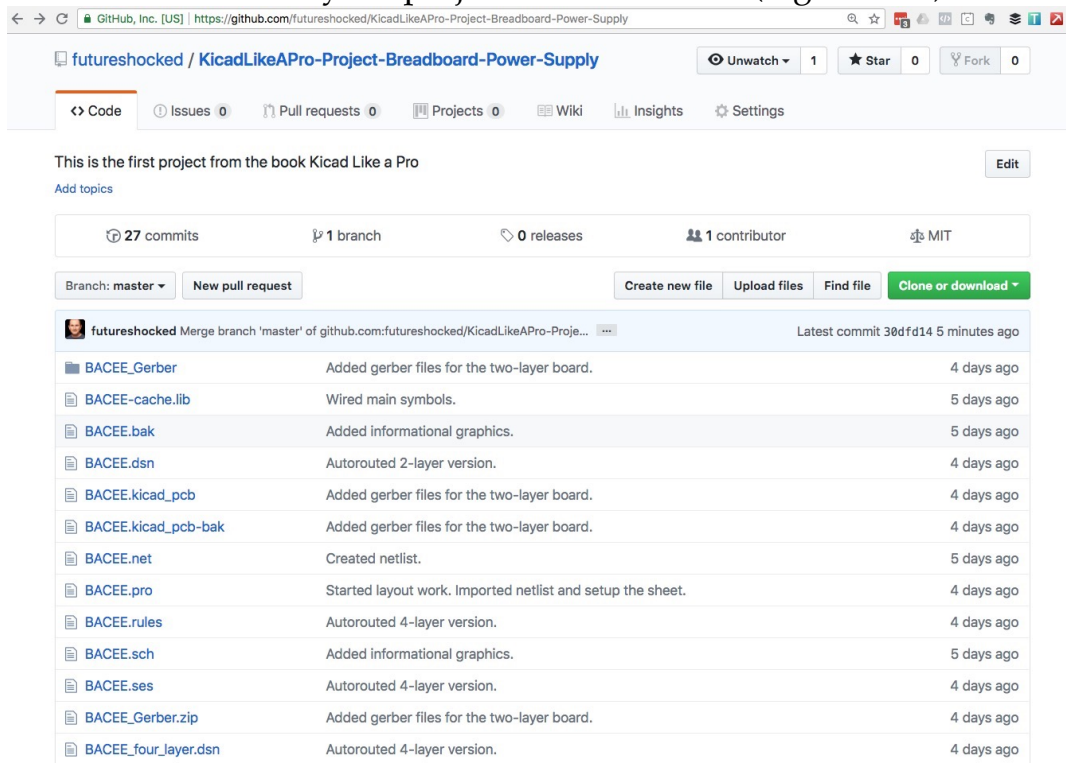


Figure 47.7: After the push, the project files are now on Github

Authentication

To avoid having to type in your Github credentials every time you pull or push, you should setup your SSH keys in your Github account. Each of your computer has a unique SSH key (or you can generate one). Once setup, Github will use that key to authenticate it, in place of a user name and password. My Github SSH keys setup looks like the example in Figure 47.8.

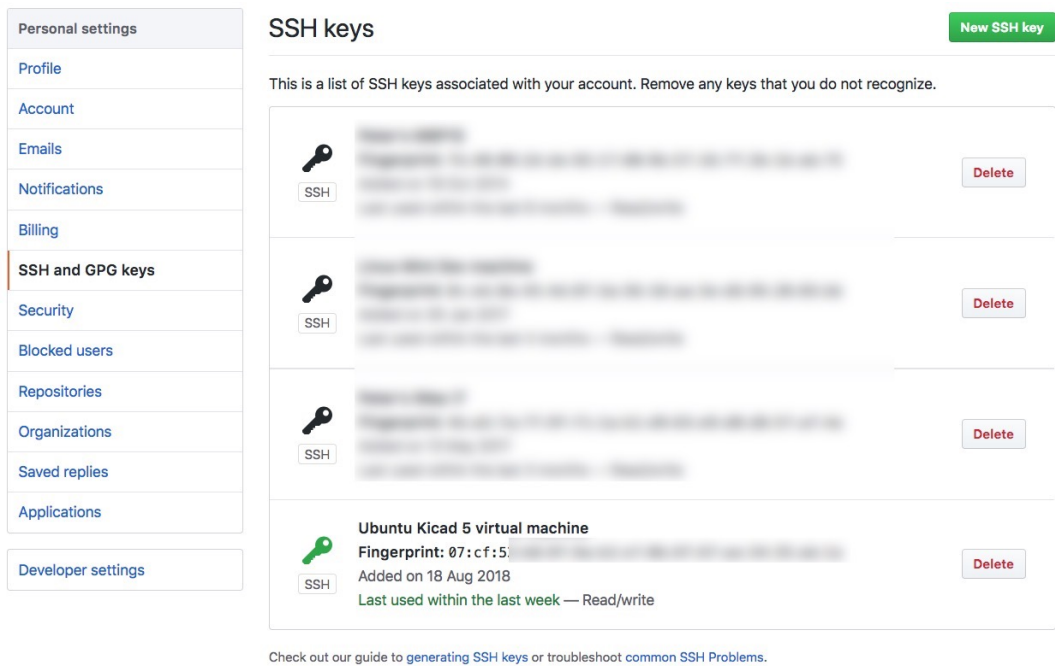


Figure 47.8: SSH keys speed up interactions with Github.

To learn how to setup your SSH keys, please follow the instructions that Github provides in its documentation, at <https://help.github.com/articles/connecting-to-github-with-ssh/>