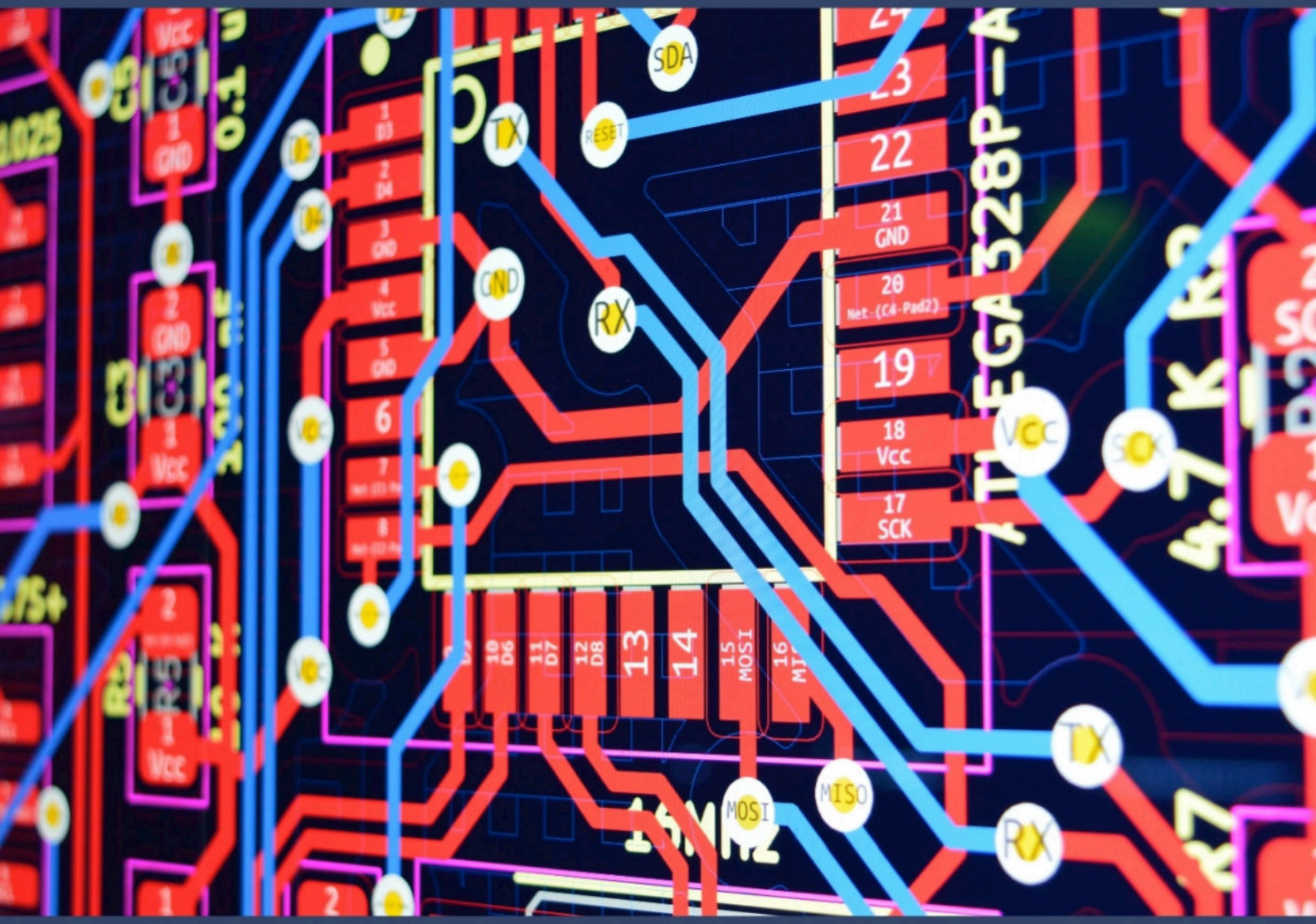
3RD EDITION

KICAD LIKE A PRO

A COMPREHENSIVE HANDS-ON GUIDE FOR LEARNING THE WORLD'S FAVOURITE OPEN SOURCE PRINTED CIRCUIT BOARD DESIGN TOOL. UPDATED FOR KICAD 6.0.

DR PETER DALMARIS



Tech Explorations
www.techexplorations.com

KICAD LIKE A PRO, THIRD EDITION

KiCad Like a Pro, 3rd Edition
By Dr Peter Dalmaris

Printed in Australia

First Printing, 2021

ISBN (PDF) : 978-1-68489-093-4
ISBN (epub): 978-1-68489-094-1
ISBN (mobi): 978-1-68489-089-7

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 4

Did you find an error?

Please let us know.
Using any web browser, go to **txplo.re/kicadr**, 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.

About 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 the 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 (PCBs) 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 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.

All examples, descriptions and procedures are tested on the nightly releases of KiCad 6 (also known as KiCad 5.99) and in KiCad 6 RC1.

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 to 8 because those will give you 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 Parts 7 and 8, zoom through them quickly, and then proceed to the projects in Part 9.

Once you have the basic KiCad concepts and skills confidently learned, you can use the recipes in Parts 7, 8 and 13 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.

Throughout this book, you will find numerous figures that contain screenshots of KiCad. To create these screenshots, I used KiCad 5.99 and KiCad 6.0 RC1 running on Mac OS. If you are using KiCad under Windows or Linux, 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 maximize 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 25 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.

Foreword by Wayne Stambaugh

In 1992 Jean-Pierre Charras started the KiCad project. From it's humble beginnings as one man's goal to provide an electronics design application for his students to a full fledged open source project with a significant number of contributors, KiCad has become the choice for users who prefer an open source solution for electronics design. As the feature parity gap from proprietary EDAs continues to close, KiCad will continue to become more widely accepted and influential.

One of the enduring hallmarks of a successful open source software project is the publication of a "how to" book. With the publication of "KiCad Like a Pro", the KiCad project joins the other well known open source projects with that distinction. Whether you are a beginner or a seasoned professional user, this book has something for everyone. From properly configuring and using KiCad to design your first printed circuit board to advanced topics such as using git for version control this book is a valuable resource for any KiCad user.

Thanks to the generosity of author Peter Dalmaris and the publisher Tech Explorations, fifty percent of the profits from the sales of the special fundraising edition of this book will be donated to the KiCad project through CERN. On behalf of all of the developers and contributors of the KiCad project, I thank you for your continued support of the project and your generous contributions.

Wayne Stambaugh
KiCad Project Leader

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 *txplo.re/kicadr*.

The two available resources are:

1. **Photos and schematics**. Get high-res copies of the photos, schematics, and layouts that appear in the book.

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 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.

Table of Contents

An introduction: Why KiCad?

Since KiCad first appeared in the PCB CAD world in 1992, it has gone through 6 major versions and evolved into a serious alternative to commercial products. I have been using KiCad almost daily since version 4 when I published the first edition of KiCad Like a Pro.

Once thought clunky and barely usable, it is now a solid, reliable CAD application. KiCad has been consistently closing the feature and performance gap against its commercial competitors. It has made leaps in adding powerful features and has significantly improved its stability.

Combined with the benefits of free and open-source software, I believe that KiCad is simply the best PCB CAD software for most use cases.

One of those benefits is KiCad's very active and growing community of users and contributors. KiCad has a dedicated developer team, supported by contributing organizations like CERN, the Raspberry Pi Foundation, Arduino LLC, and Digi-Key Electronics. The community is also active in contributing funds to cover development costs. Since joining the Linux Foundation, the KiCad project has received around $90,000 in donations. The project used this money to buy development time and funding developer conference travel and meetups. To a large extent, this alone guarantees that KiCad's development will accelerate and continue to in the future.

Supporting the KiCad core team is the KiCad community. The community consists of over 250 thousand people worldwide that have downloaded a copy. These people support the KiCad project in various ways: they write code, create and share libraries, and help others learn. They write documentation, record videos, report bugs, and share hacks. During the KiCad 6 development cycle, the KiCad repository had around 14600 commits from the community. Based on this number, KiCad 6 is the most significant KiCad version ever in terms of changes.

Another signal of the strength of the KiCad community is that KiCad 6 includes completed or nearly completed translations to nearly 20 languages. No other CAD software that I am aware of can boast this.

PCB manufacturers have also taken notice. Many of them now publish Kicad-specific tutorials, explaining how to order your boards. Some have made it possible to upload the KiCad native layout file from your project instead of generating multiple Gerber files.

And finally, KiCad is part of an expanding CAD ecosystem. You will find KiCad-compatible component libraries on the Internet's major

repositories, such as Snapeda and Octopart, as well as native support in PCB project version control software for teams, such as CADLAB.io.

KiCad's development and prospects have never been brighter than now. KiCad's roadmap has exciting new features and capabilities such as grouping board objects into reusable snippets and a stable Python API.

Why do I use KiCad? Because it is the perfect PCB software for my use case.

I am an electrical engineer with a background in electronics and computer engineering. But, above all, I am a technology educator and electronics hobbyist. The majority of my PCB projects eventually find themselves in my books and courses. My projects are very similar to those of other hobbyists in terms of complexity and size. I make things for my Arduino and Raspberry Pi courses. As a hobbyist, KiCad proved to be the perfect tool for me.

Your use case may be different. You may be a university student completing an engineering degree. You may be a hobbyist or solo developer working in a startup company. You may be part of a team working on commercial projects that involve highly integrated multi-layer PCBs.

To help you decide whether KiCad is right for you, I have compiled a list of 12 KiCad Benefits. This list contained ten items in the second edition of the book. I added the last two items to highlight additional benefits brought about with KiCad 6.

Here they are:

Benefit 1: KiCad is open source. This is very important, especially as I spend more time creating new and more complicated boards. Open source, by definition, means that the code base of the application is available for anyone to download and compile on their computer. It is why Linux, Apache, and WordPress essentially run the Internet (all of them open-source). While I am not extreme in my choices between open source and closed source software, whenever a no-brainer open-source option does appear, like KiCad, I take it.

Benefit 2: It is free! This is particularly important for hobbyists. CAD tools can be expensive. This is worsening with most CAD software companies switching to a subscription-based revenue model. When you are a hobbyist or student or bootstrapping for a startup, regular fees do add up. Not to mention that most of us would not be using even half of the features of commercial CAD software. It is hard to justify spending hundreds of dollars on PCB software when there is KiCad. This brings me to Benefit 3

Benefit 3: KiCad is unlimited. There are no "standard", "premium" and "platinum" versions to choose from. It's a single download, and you get everything. While there are commercial PCB tools with free licensing for students or hobbyists, there are always restrictions on things like how many layers and how big your board can be, what you can do with your board once you have it, who can manufacture your board, and much more. And there is always the risk that the vendor may change the deal in the future where you may have to pay a fee to access your projects. I'll say again: KiCad is unlimited and forever! This is so important that I choose to pay a yearly donation to CERN that is higher than the cost of an Autodesk Eagle license to do my part in helping to maintain this.

Benefit 4: KiCad has awesome features. Features such as interactive routing, length matching, multi-sheet schematics, configurable rules checker, and differential routing are professional-grade. While you may not need to use some of them right away, you will use them eventually. You can add new features through third-party add-ons. The external autorouter is one example. The ability to automate workflows and extend capabilities through Python scripts is another.

Benefit 5: KiCad is continually improved. Especially since CERN & Society Foundation became involved in their current capacity, I have seen a very aggressive and successfully implemented roadmap. When I wrote the first version of this list (August 2018), KiCad 5 was about one month old. The funding for KiCad 6 was already complete, and the road map living document was published. Three years later, KiCad 6 was delivered with promises fulfilled. Now, with KiCad 6 published, the road map for the future looks just as exciting.

Benefit 6: KiCad's clear separation of schematics and layout is a bonus to learning and using it. Users of other PCB applications often find this confusing, but I believe that it is an advantage. Schematic design and layout design are indeed two different things. Schematic symbols can be associated with different footprints that depend on the project requirements. You can use the schematic editor independently of the layout editor or in sync. I often create schematic diagrams for my courses that I have no intention of converting into PCBs. I also often create multiple versions of a board using the same schematic. This separation of roles makes both scenarios easy.

Benefit 7: I can make my boards anywhere: I can upload my project to any online fabricator that accepts the industry-standard Gerber files; I can

upload it to an increasing number of fabricators that accept the native KiCad layout file; and, of course, I can make them at home using an etching kit.

Benefit 8: KiCad works anywhere. Whether you are a Mac, Windows, or Linux person, you can use KiCad. I use it on all three platforms. I can take my KiCad 6 project from the Mac and continue working on Windows 10 without worrying about any software or project files glitches.

Benefit 9: KiCad is very configurable. You can assign your favorite keyboard hotkeys and mapping, and together with the mouse customizations, you can fully adapt it to your preferences. With the additions of the plugin system and the Python API, , it will be possible to extend your instance of KiCad with the exact features you need (or write them).

Benefit 10: If you are interested in creating analog circuits, you will be happy to know that KiCad ships with SPICE. You can draw the schematic in Eeschema and then simulate it in SPICE without leaving KiCad. This integration first appeared in KiCad 5, and it is now a stable feature.

Benefit 11: In the past, KiCad's release cycle was somewhat chaotic. New major versions would come out every two or three years, but no one knew ahead of time. In the future, KiCad will operate in a yearly release cycle. This is good for two reasons: One, commercial users who can now better predict how the software they depend on will change and when. Two, as KiCad users, all of us will be able to expect a reliable development schedule that prioritizes reliability. KiCad is now mature enough to be able to evolve predictably.

Benefit 12: KiCad is now a serious productivity tool for businesses. If you are an electronics engineer, you can proudly list it in your resume. If you are using it in your business, you can contract the KiCad Services Corporation, to customize the software to your exact requirements. I am talking about deep customization, not just changing the theme and the menu bars. This means that KiCad can fit precisely with your business. As far as I know, no commercial CAD application can do that. For the non-business users among us, we can expect many of these business-led improvements to flow into future software versions in the tradition of open-source software.

These are the twelve most important reasons I have chosen KiCad as my tool of choice for designing PCBs. These reasons might not be suitable for you, but I hope you will consider reading this book first before making your own decision.

Over the last seven years, I have packed almost everything I have learned as a KiCad user in this book. I have organized it in a way that will

make learning KiCad quick. The objective of this book is to make you productive by the time you complete the first project, in part four.

If you come from another PCB CAD tool and have experience designing PCBs, I only ask that you have an open mind. KiCad is most certainly very different from your current PCB tool. It looks different, and it behaves differently. It will be easier to learn it if you consciously put aside your expectations and look at KiCad like a beginner would. As per the Borg in Star Trek, "resistance is futile", and in learning, like in so many other aspects of life, you are better off if you go with the flow.

Let's begin!

Part 1: Introduction

1. What is a PCB?

As a child, I remember that my interest in electronics grew from admiration of what these smart engineers had come up with to curiosity about how these things worked. This curiosity led me to use an old screwdriver that my dad had left in a drawer (probably after fixing the hinges on a door) to open anything electronic with a screw large enough for the screwdriver to fit in.

A record player, a VCR, a radio; all became my "victims." I am still amazed that a charged capacitor didn't electrocute me. At least, I had the good sense to unplug the appliances from the mains. Inside those devices, I found all sorts of wondrous things: resistors, transformers, integrated circuits, coils, and power supplies.

Engineers had attached those things on small green boards, like the one in Figure 1.1.1. This is an example of a printed circuit board, or PCB, for short.

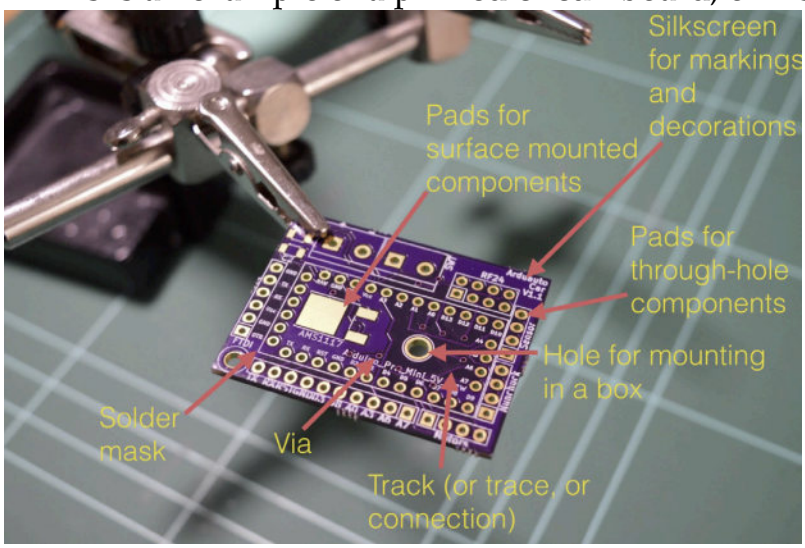


Figure 1.1.1: The top side of a printed circuit board.

Let's look at the components of a PCB, what a PCB looks like, and the terminology that we use. The example PCB is one I made for one of my courses (Figure 1.1.1).

The top side of the PCB is the side where we place the components. We can place components on the bottom side, too.

In general, there are two kinds of components: through-hole or surface-mounted components. We can attach through-hole components on the PCB by inserting the leads or the pins through small holes and using hot solder to hold them in place. In the example pictured in Figure 1.1.1, you can see

several holes to insert the through-hole component pins. The holes extend from the top side to the bottom side of the PCB and are plated with a conductive material. This material is usually tin, or as in the case of the board in the image, gold. We use solder to attach and secure a component through its lead onto the pad surrounding the hole (Figure 1.1.2).

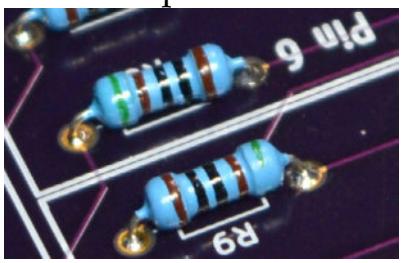


Figure 1.1.2: A through-hole component attached to a PCB.

If you wish to attach a surface-mounted component, then instead of holes, you attach the component onto the surface of the PCB using tin-plated pads. You will use just enough solder to create a solid connection between the flat connector of the component and the flat pad on the PCB (Figure 1.1.3).

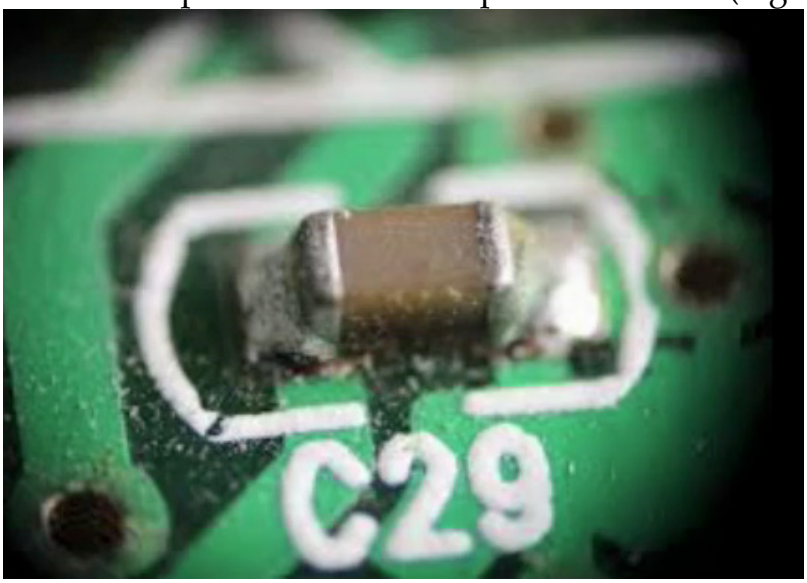


Figure 1.1.3: A surface-mounted component attached to a PCB.

Next is the silkscreen. We use the silkscreen for adding text and graphics. The text can provide helpful information about the board and its components. The graphics can include logos, other decorations, and useful markings.

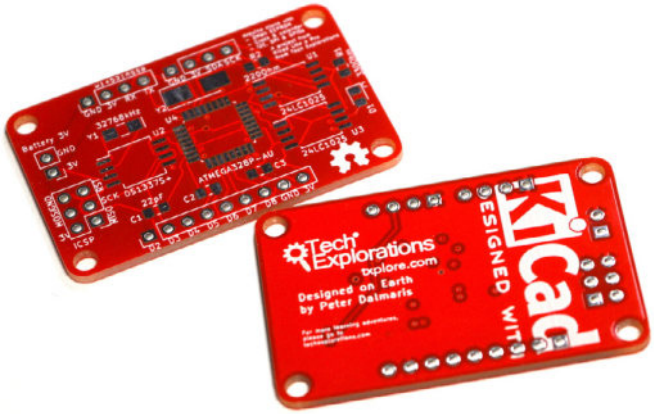


Figure 1.1.4: The white letters and lines is the silkscreen print on this PCB.

In Figure 1.1.4, you can see here that I've used white boxes to indicate the location of various components. I've used text to indicate the names of the various pins, and I've got version numbers up there. It's a good habit to have a name for the PCB and things of that sort. Silkscreen goes on the top or the bottom of the PCB.

Sometimes, you may want to secure your PCB onto a surface. To do that, you can add a mounting hole. Mounting holes are similar to the other holes in this board, except they don't need to be tinned. You can use a screw with a nut and bolt on the other side to secure the PCB inside a box.

Next are the tracks. In this example (Figure 1.1.5), they look red because of the color of the masking chemical used by the manufacturer.

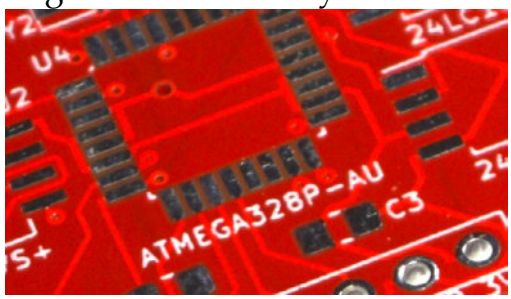


Figure 1.1.5: The bright red lines connecting the holes are tracks.

Tracks are made of copper, and they electrically connect pins or different parts of the board. You can control the thickness of a track in your design. You can also refer to a "track" as a "trace."

Notice the small holes that have no pad around them? These are called 'vias.' A via looks like a hole but is not used to mount a component. A via is used to allow a track to continue its route in a different layer. If you're using PCBs with two or more layers, you can use vias to connect a track from any one of the layers to any of the other layers. Vias are handy for routing your tracks around the PCB.

The red substance that you see on the PCB is the solder mask. It does a couple of things. It prevents the copper on the PCB from being oxidized over time. The oxidization of the copper tracks negatively affects their conductivity. The solder mask prevents oxidization.

Another thing that the solder mask does is to make it easier to solder by hand. Because pads can be very close to each other, soldering would be complicated without the solder mask. The solder mask prevents hot solder from creating bridges between pads because it prevents it from sticking on the board (Figure 1.1.6). The solder mask prevents bridges because the solder cannot bond with it.

Figure 1.1.6: A solder bridge like this one is a defect that a solder mask can prevent.

Often, the tip of the solder, the soldering iron, is almost as big or sometimes as bigger than the width of the pads, so creating bridges in those circumstances is very easy, and a solder mask helps in preventing that from happening.

In Figure 1.1.7 you can see an example of the standard 1.6mm thick PCB.

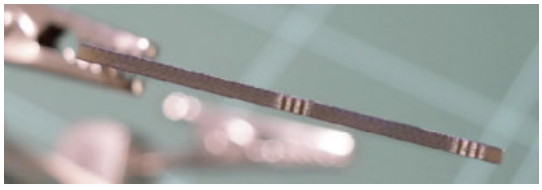


Figure 1.1.7: This PCB has a thickness of 1.6mm, and is made of fiberglass.

Typically, PCBs are made of fiberglass. The typical thickness of the PCB is 1.6 millimeters. In this closeup view of a PCB picture (Figure 1.1.8), you can see the holes for the through-hole components. The holes for the through-hole components are the larger ones along the edge of the PCB. Notice that they are tined on the inside, electrically connecting the front and back.

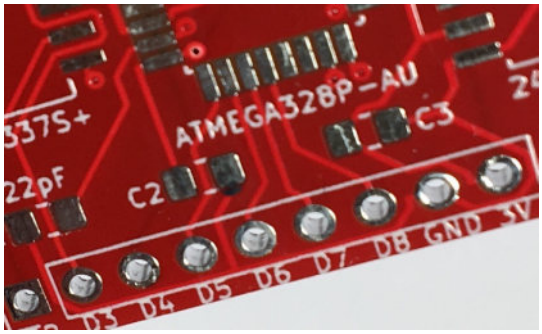


Figure 1.1.8: A closeup view of the top layer.

In Figure 1.1.8, you can see several vias (the small holes) and tracks, the red solder mask, and the solder mask between the pads. In this closeup, you can also see the detail of the silkscreens. The white ink is what you use in the silkscreen to create the text and graphics.

Figure 1.1.9 is interesting because it shows you a way to connect grounds and VCC pads to large areas of copper, which is called the copper fill.

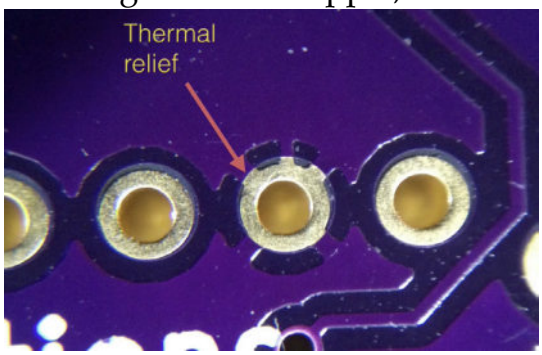


Figure 1.1.9: Thermal relief connects a pad to a copper region.

In Figure 1.1.9, the arrow points to a short segment of copper that connects the pad to a large area of copper around it. We refer to this short segment of copper as a 'thermal relief.' Thermal reliefs make it easier to solder because the soldering heat won't dissipate into the large copper area.

Figure 1.1.10 gives a different perspective that allows us to appreciate the thickness of the tracks.

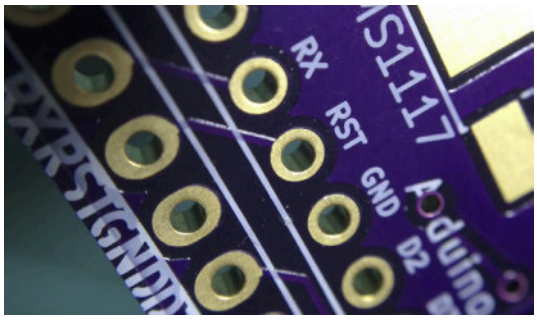


Figure 1.1.10: The plating of the holes covers the inside of the hole and connects that front end with the back end.

Notice the short track that connects the two reset holes (RST)? The light that reflects off the side of the track gives you an idea of the thickness of that copper, which is covered by the purple solder mask.

In this picture, you can also see a very thin layer of gold that covers the hole and the pad and fills the inside of the hole. This is how you electrically have both sides of the hole connected.

Instead of gold plating, you can also use tin plating to reduce manufacturing costs.

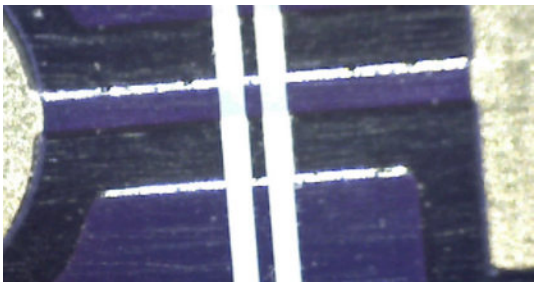


Figure 1.1.11: A detail of this example board at 200 times magnification.

The image in Figure 1.1.11 was taken at 200 times magnification. You can see a track that connects two pads and the light that reflects off one side of the track.

2. The PCB design process

To design a printed circuit board, you have to complete several steps, make decisions, and iterate until you are satisfied with the result.

A printed circuit board is a physical device that takes time and money to manufacture. It must be fit to perform its intended purpose, and must be manufacturable. Therefore, your design must be of high quality, safe, and possible to manufacture by your chosen manufacturer.

Apart from the practical considerations of designing a PCB, there are also the aesthetic ones. You want your work to look good, not just to function well. Designing a PCB, apart from being an engineering discipline, is also a form of art.

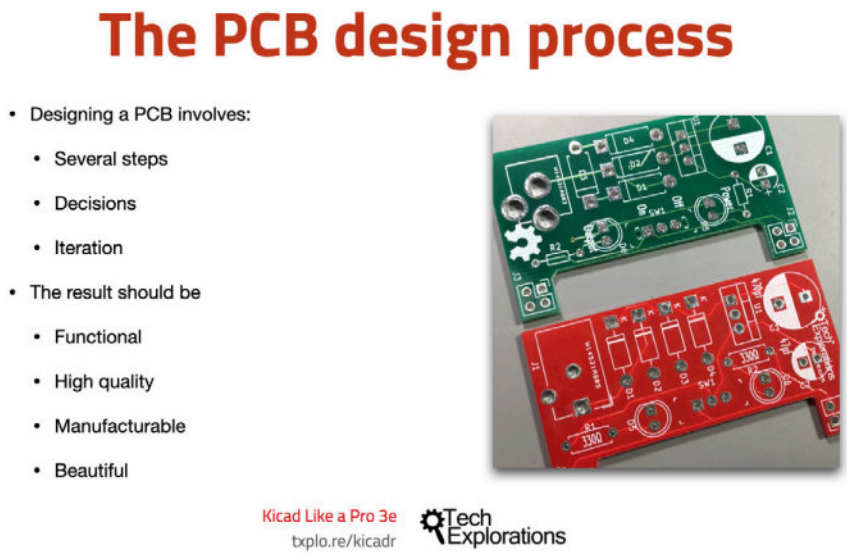


Figure 1.2.1: Some considerations of the PCB design process.

In this book, you will learn about the technical elements of designing a PCB in KiCad, but I am sure that as you start creating your PCBs, your artistic side will emerge. Over time, your PCB will start to look uniquely yours.

PCB design is concerned with the process of creating the plans for a printed circuit board. It is different from PCB manufacturing. In PCB design, you learn about the tools, process, and guidelines useful for creating such plans.

In PCB manufacturing, on the other hand, you are concerned about the process of converting the plans of a PCB into the actual PCB.

As a designer of printed circuit boards, it is useful to know a few things about PCB manufacturing, though you surely do not need to be an expert. You need to know about the capabilities of a PCB manufacturing facility so that you can ensure that your design does not exceed those capabilities and that your PCBs are manufacturable.

As a designer, you need to have an understanding of the design process, and the design tools. To want to design PCB, I assume that you already have a working knowledge of electronics. Designing a PCB, like much else in engineering, is a procedural and iterative process that contains a significant element of personal choice. As you build up your experience and skills, you will develop your unique designing style and process.

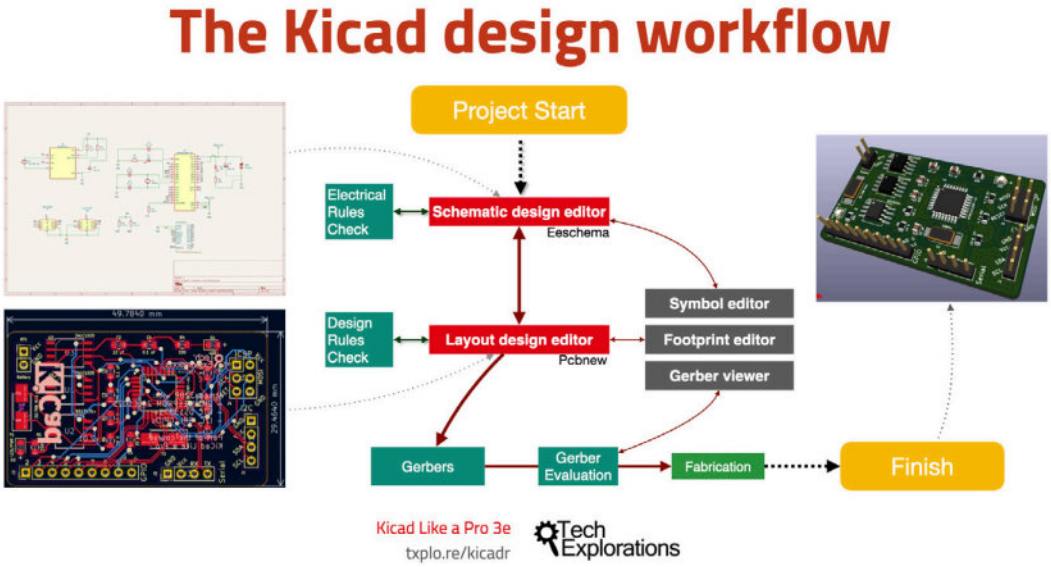


Figure 1.2.2: KiCad is a suite of applications.

KiCad is not a single application. It is a suite of apps that work together to help you create printed circuit boards. As a result, it is possible to customise the PCB design process to suit your particular style and habits.But when you are just starting up, I think it is helpful to provide a workflow that you can use as a model.

In Figure 1.2.2 you can see my KiCad PCB design workflow model. You can use it as it is, or you can modify as you see fit. I distilled this workflow by drawing from my own experience and learning from other people's best practices. I also tried to simplify this process and make it suitable for people new to PCB design.

In this book, I will be following this PCB design workflow in all of the projects.

From a very high-level perspective, the PCB design workflow only has two major steps:

1. Step 1 is the schematic design using the schematic design editor (Eeschema);
2. Step 2 is the layout design using the layout editor (Pcbnew).

Once you have the layout design, you can export it and the manufacture it.

The goal of the schematic design step is to capture information about the circuit that will be implemented in the final PCB. Once you have a schematic design, you can use the layout editor to create a version of the PCB. Remember, a schematic design can have many different layouts.

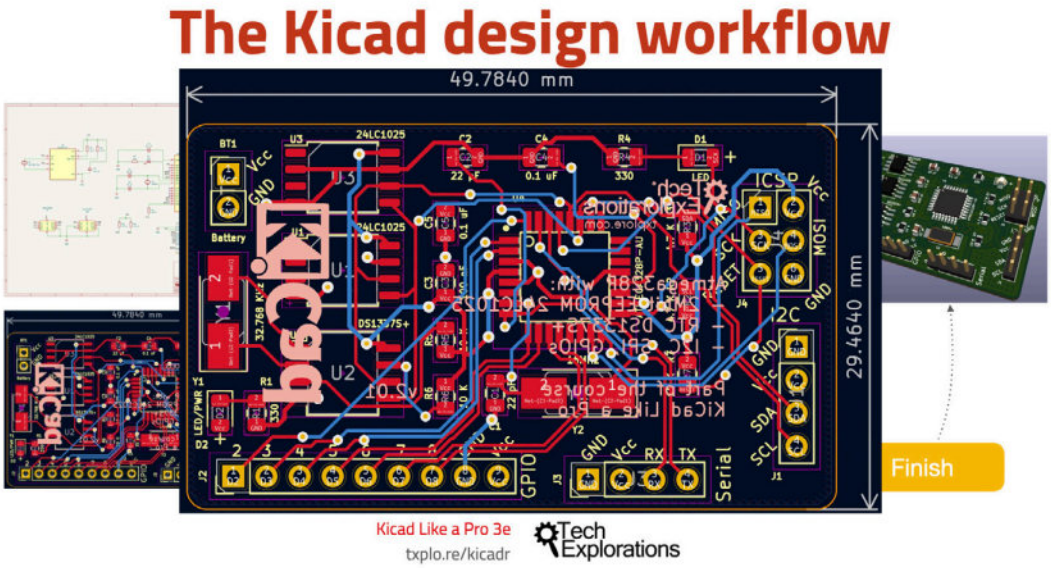


Figure 1.2.3: The KiCad layout file contains information about the physical PCB.

The KiCad layout file contains information about your board, which the manufacturer can use to create the board. The layout must contain information about the size and shape of the board; its construction (such as how many layers it must have); the location of the components on the board, the location of various board elements, like pads, holes, traces and cutouts; the features of these elements (such as the sizes of holes and traces); and much more (which you will learn in detail later in this book).

Let's walk through through the workflow now using the diagram in Figure 1.2.3 as an aid.

For the discussion that follows, keep in mind these definitions:

- A symbol is a symbolic representation of a real component in the schematic; a symbol represents a component's function, not its physical appearance or location in the final PCB.

- A footprint is a graphical depiction of a real component in the layout. It relates directly to a real physical counterpart. It contains information about the real component's location and dimensions.

In this book, I will be using the terms "symbols" and footprints according to the definitions above.

In KiCad, the process begins with Eeschema, which is the schematic editor.

In Eeschema you create the electrical schematic that describes the circuit that eventually will be manufactured into the PCB. You draw the schematic by selecting symbols from the library and adding them to the schematic sheet. If a component that you need doesn't exist in the library, you can search for it on the Internet, or create yourself with the help of the schematic library editor.

Running regular electrical rules checks helps to detect defects early. Eeschema has a built-in checker utility for this purpose.

Pcbnew (KiCad's layout editor) has its own validator, the Design Rules Checker.

These two utilities help to produce PCBs that have a low risk to contain design or electrical defects.

Before you finish work in Eeschema and continue with the layout, you must first associate the schematic symbols with layout footprints.

In KiCad 6, many symbols come with preset symbol to footprint associations, but many don't, so you'll have to do this yourself. Also keep in mind that, as I said earlier, KiCad is very flexible. It is possible to assign many different footprints to the same schematic symbol (one at a time, of course).

Once you have completed the Electrical Rules Check and symbol to footprint associations, you can continue with layout using the KiCad Layout design editor, or Pcbnew.

You use Pcbnew to position the footprints on the sheet and connect the footprint pins using wires. You'll also add an outline that marks the outer limit of the PCB, and other design elements like mounting holes, logos, and instructional text.

Once you have your PCB laid out and have its traces completed, you can go ahead and do the design rules check. This check looks for defects in the board, such as a trace that is too close to a pad or two footprints overlapping.

Let's look at some of the PCB terminology before we continue.

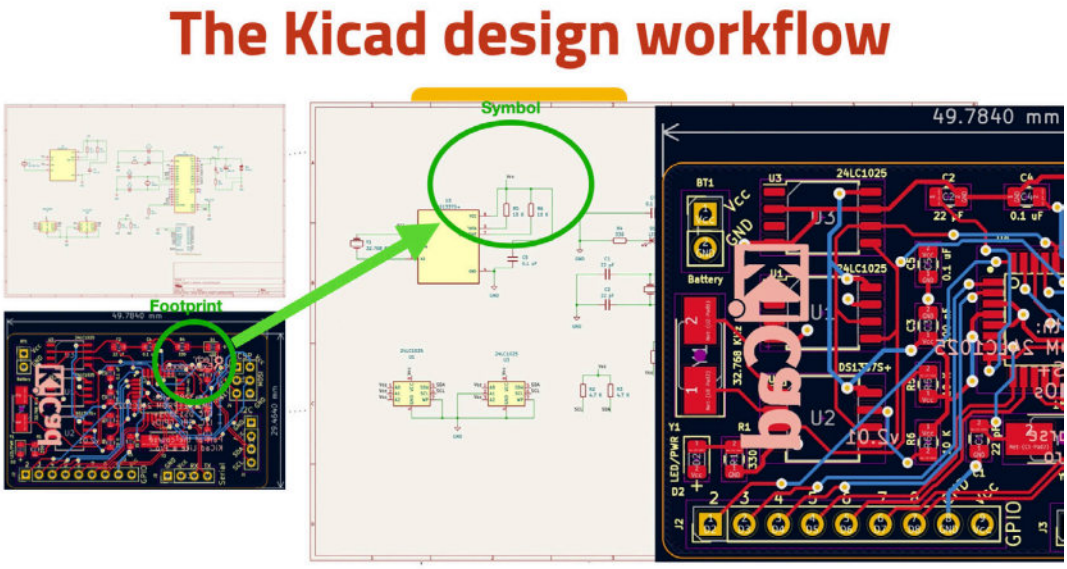


Figure 1.2.4: Symbols and footprints.

As you know, a symbol is a symbolic representation of a real component in the schematic. A footprint is a graphical depiction of a real component in the layout.

You, as the designer, must tell KiCad which footprint you want to use in your PCB by associating it to a particular symbol. Take the example of a resistor. A resistor uses a specific symbol in the schematic, but on the PCB it can be realized as a through-hole or SMD device of varying sizes.

When you are finished working on the layout, you can continue with the last step which involves exporting the layout information in a format that is compatible with your board manufacturer's requirement.

The industry standard for this is a format called 'Gerber.' Gerber files contain several related files, with one Gerber file per layer on your PCB, and contain instructions that the fabrication house needs to manufacture your PCB.

Let's move on to the next chapter where we'll talk about fabrication.

3. Fabrication

Imagine that you have finished laying out your board in KiCad, and you're ready to make it. What are your options? One option is to make your PCBs at home. There's a guide available on the Fritzing website.

The process described in the Fritzing guide is called etching. It involves the use of various chemicals in chemical baths. Some of these chemicals are toxic. You have to have special safety equipment and keep your children and pets away. The process emits smelly and potentially dangerous fumes. Once you have your board etched, you still need to use a drill to make holes and vias and then figure out how to connect your top and bottom layers.

If this sounds like not your kind of thing (I'm with you!), then you can opt for a professional PCB manufacturer service. PCBWay, NextPCB, and OSH Park are very good at what they offer.

You can get a professionally made PCB for around $15 for several copies and without danger to yourself as well. I've used OSHPark (great for beginners thanks to its straightforward user interface) and PCBWay (great for more advanced projects that need an extensive array of manufacturing options) extensively. I'm always happy with the result. Using an online manufacturer takes a little bit of planning because once you order your PCBs, it can take up to several weeks to be delivered. If you're in a hurry, there are options to expedite the process if you are willing to pay a premium.

The typical small standard two-layer order costs around $10 for a two square inch board; you get three copies of that. This price works out to around $5 per square inch. The pricing is consistent in the industry, where the main cost factor is the size of the PCB. There is a strong incentive to make your PCBs as small as possible. Be aware of this when you design your layout.

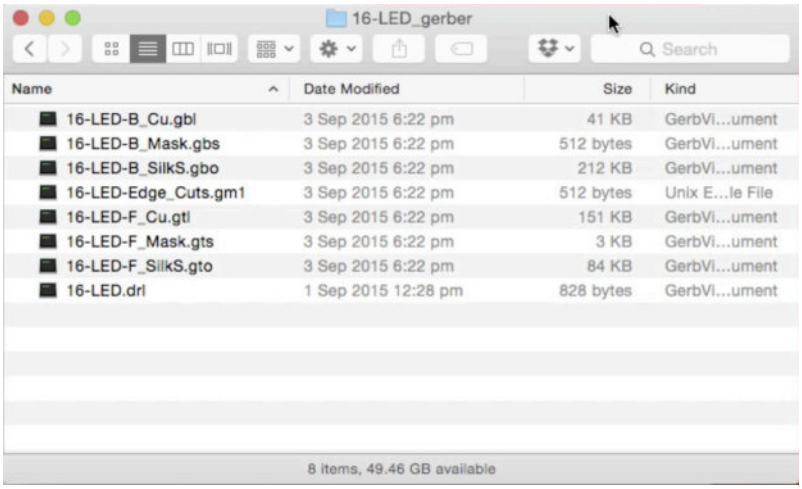


Figure 3.3.1: An example of the Gerber files that the manufacturer will need in order to make your PCB.

Now, let's turn our attention to the files you need to upload for these services — and the files are Gerber files. Each layer on your PCB has its own Gerber file, which is simply a text file. Figure 3.3.2 shows the contents of an example Gerber file.

```
1  G04 #@! TF.FileFunction,Soldermask,Bot*
2  %FSLAX46Y46*%
3  G04 Gerber Fmt 4.6, Leading zero omitted, Abs format (unit mm)*
4  G04 Created by KiCad (PCBNEW (2015-07-06 BZR 5891, Git 351914d)-product) date
   03/09/2015 18:22:08*
5  %MOMM*%
6  G01*
7  G04 APERTURE LIST*
8  %ADD10C,0.100000*%
9  %ADD11R,1.727200X2.032000*%
10 %ADD12O,1.727200X2.032000*%
11 G04 APERTURE END LIST*
12 D10*
13 D11*
14 X122555000Y-42545000D03*
15 D12*
16 X120015000Y-42545000D03*
17 X117475000Y-42545000D03*
18 X114935000Y-42545000D03*
19 X112395000Y-42545000D03*
20 M02*
21
```

Figure 3.3.3: Gerber files contain text

You can see that this is just a text-based file that contains instructions. An advantage of this text format is that you can use a version control system like Git to maintain your project history and store and share via online repositories like Github.

Ucamco has designed the Gerber files system and standard. They make equipment and write software for PCB manufacturers — things like PreCAM software, PCB CAM, laser photoplotters, and direct imaging systems. If you're curious about how to read these Gerber files, you can look up the Gerber format specification on Ucamco's website.

4. Get KiCad for your operating system

It is now time to download your copy of KiCad and install it on your computer.

Kicad has support for a variety of operating systems. The major operating systems, Mac OS and Windows, are supported. Of course, there is support for Ubuntu and a lot of different flavors of Linux. I have tested, and I frequently use KiCad on Mac OS. Mac OS is my primary operating system, but I'm also working on Windows and Kubuntu instead of Ubuntu.

Kubuntu is based on Ubuntu in its core but uses the KDE Desktop and related software. I find Kubuntu to offer a much better experience compared to Ubuntu. Of course, this is my personal preference, and opinions vary greatly.

I'm not going to show you how to install KiCad on each one of those operating systems. The KiCad developer team has refined the installer over the years. The KiCad installation process on the supported operating systems is just like that of any other refined application.

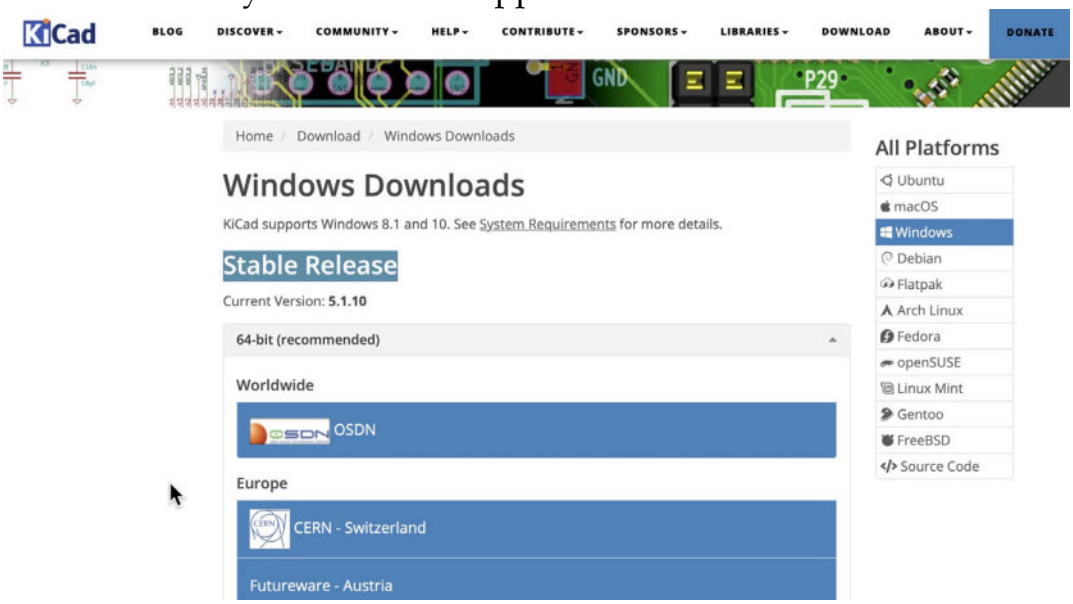


Figure 1.4.1: Windows stable release download page.

For example, to get the KiCad installer for Windows, go to the KiCad Windows page and download the stable version of KiCad from your preferred source. Double-click on the installer icon and follow the installation wizard instructions to complete the installation on your computer.

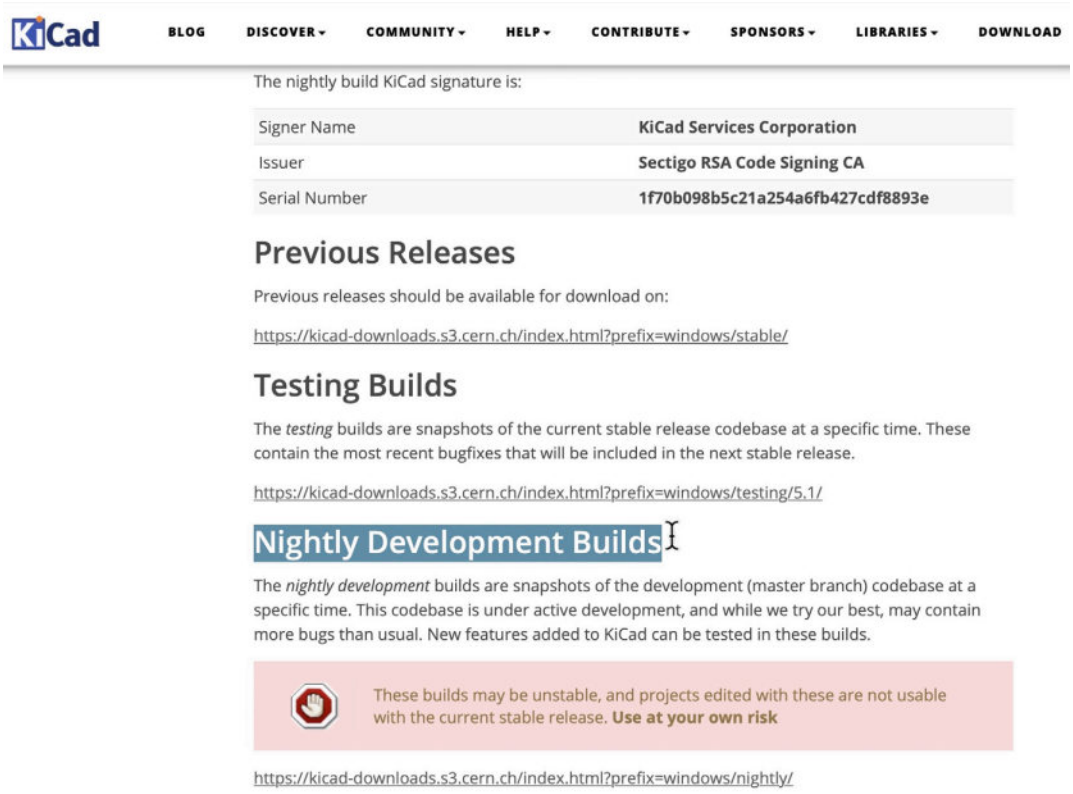
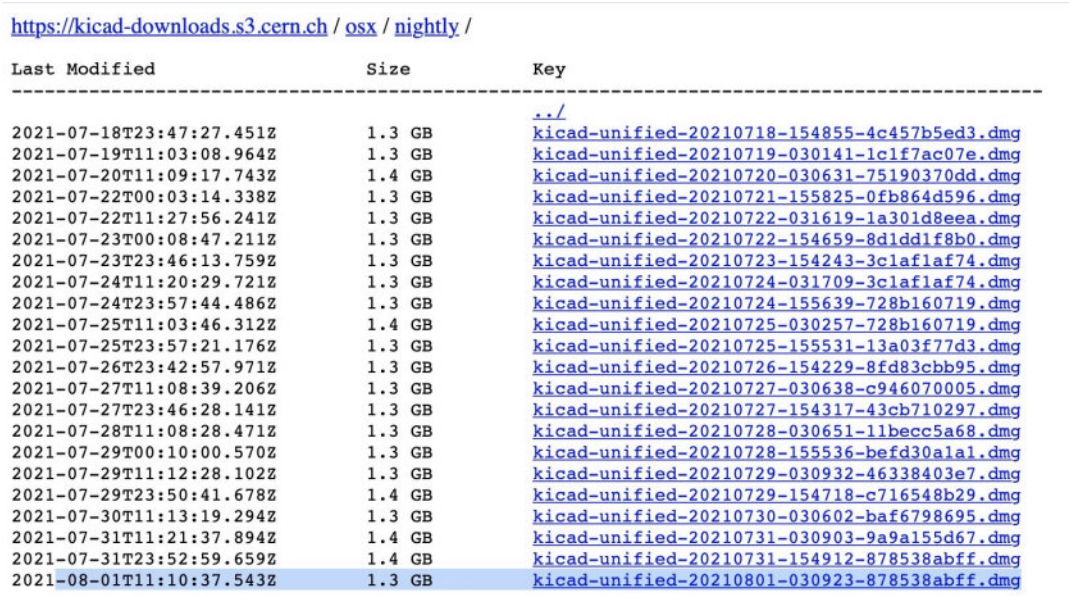


Figure 1.4.2: Nightly build download

You can download and install the latest available version of KiCad that is available as a nightly build. Nightly builds are work-in-progress. They contain the latest code committed by the KiCad developers but are considered "unstable." Therefore, you should not use it for work that you do not want to lose. The major operating systems have a nightly build generated (almost) every night. If you want to look at the cutting-edge version of KiCad and you are not afraid of weird behaviors and strange crashes, then go to the nightly releases page for your preferred operating system and download the installer. Example: Windows.



https://kicad-downloads.s3.cern.ch / osx / nightly /

Last Modified	Size	Key
		../
2021-07-18T23:47:27.451Z	1.3 GB	kicad-unified-20210718-154855-4c457b5ed3.dmg
2021-07-19T11:03:08.964Z	1.3 GB	kicad-unified-20210719-030141-1c1f7ac07e.dmg
2021-07-20T11:09:17.743Z	1.4 GB	kicad-unified-20210720-030631-75190370dd.dmg
2021-07-22T00:03:14.338Z	1.3 GB	kicad-unified-20210721-155825-0fb864d596.dmg
2021-07-22T11:27:56.241Z	1.3 GB	kicad-unified-20210722-031619-1a301d8eea.dmg
2021-07-23T00:08:47.211Z	1.3 GB	kicad-unified-20210722-154659-8d1dd1f8b0.dmg
2021-07-23T23:46:13.759Z	1.3 GB	kicad-unified-20210723-154243-3c1af1af74.dmg
2021-07-24T11:20:29.721Z	1.3 GB	kicad-unified-20210724-031709-3c1af1af74.dmg
2021-07-24T23:57:44.486Z	1.3 GB	kicad-unified-20210724-155639-728b160719.dmg
2021-07-25T11:03:46.312Z	1.4 GB	kicad-unified-20210725-030257-728b160719.dmg
2021-07-25T23:57:21.176Z	1.3 GB	kicad-unified-20210725-155531-13a03f77d3.dmg
2021-07-26T23:42:57.971Z	1.3 GB	kicad-unified-20210726-154229-8fd83cbb95.dmg
2021-07-27T11:08:39.206Z	1.3 GB	kicad-unified-20210727-030638-c946070005.dmg
2021-07-27T23:46:28.141Z	1.3 GB	kicad-unified-20210727-154317-43cb710297.dmg
2021-07-28T11:08:28.471Z	1.3 GB	kicad-unified-20210728-030651-11becc5a68.dmg
2021-07-29T00:10:00.570Z	1.3 GB	kicad-unified-20210728-155536-befd30a1a1.dmg
2021-07-29T11:12:28.102Z	1.3 GB	kicad-unified-20210729-030932-46338403e7.dmg
2021-07-29T23:50:41.678Z	1.4 GB	kicad-unified-20210729-154718-c716548b29.dmg
2021-07-30T11:13:19.294Z	1.3 GB	kicad-unified-20210730-030602-baf6798695.dmg
2021-07-31T11:21:37.894Z	1.4 GB	kicad-unified-20210731-030903-9a9a155d67.dmg
2021-07-31T23:52:59.659Z	1.4 GB	kicad-unified-20210731-154912-878538abff.dmg
2021-08-01T11:10:37.543Z	1.3 GB	kicad-unified-20210801-030923-878538abff.dmg

Figure 1.4.3: Nightly build download

If you're working on Mac OS, go to the Mac OS downloads page and download the latest available stable release. You can also download a nightly build if you are comfortable with the inherent risk. Both stable and nightly builds come as a regular DMG file. The download file contains the entire KiCad suite with all its applications, the documentation, and the libraries for the schematic symbols, footprints, and templates. It also includes several demos projects.

The installation process makes use of Ubuntu's apt-get system. For Ubuntu, you can find installations instructions on the Ubuntu page. For using nightly development builds in Ubuntu, you will find instructions on the same page.

There is there are similar instructions for the various other operating systems like Suse and Fedora.

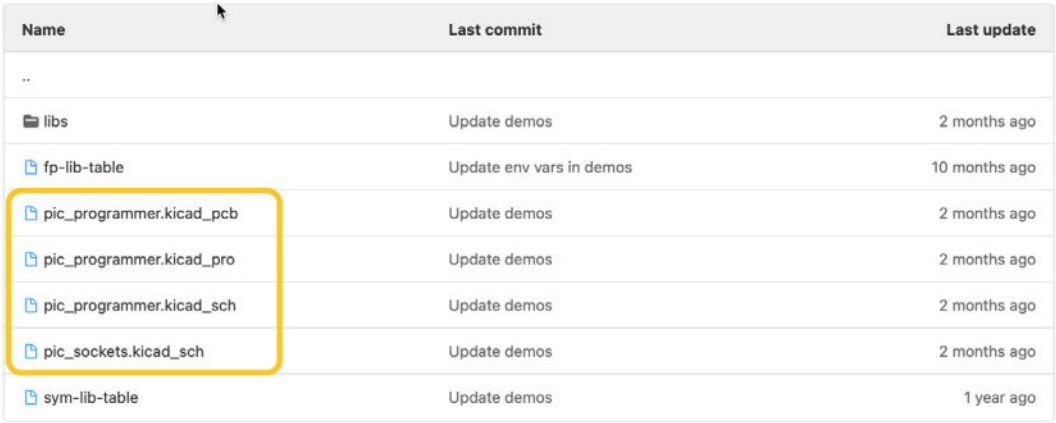
You also have the option to download the source code and build from the source on your operating system. This is not something that I usually do unless I want to play around with it and experiment. Luckily, the operating systems I use or have excellent binary builds, so I never needed to build my KiCad instance from the source. But if you are someone who enjoys doing that, then go to the source code page and follow the detailed instructions.

At this point, I invite you to download the version of KiCad that is suitable for your operating system and install KiCad on your computer. Once you finish installing KiCad, verify that it's up and running by starting KiCad.

In the next chapter, you will use your brand new instance of KiCad to look at some of the demo projects that ship with KiCad.

5. Example KiCad projects

Now that you have installed your instance of KiCad let's start your familiarisation with it by looking at one of the examples that come with it. Browse to the KiCad demos folder, and download the one titled 'pic_programmer' (Figure 1.5.1). You can also download the entire "demos" folder if you wish.



Name	Last commit	Last update
..		
libs	Update demos	2 months ago
fp-lib-table	Update env vars in demos	10 months ago
pic_programmer.kicad_pcb	Update demos	2 months ago
pic_programmer.kicad_pro	Update demos	2 months ago
pic_programmer.kicad_sch	Update demos	2 months ago
pic_sockets.kicad_sch	Update demos	2 months ago
sym-lib-table	Update demos	1 year ago

Figure 1.5.1: The contents of the 'pic_programmer' demo project folder.

The demo project folder contains several files that make up the project. For now, the ones to focus on have the extensions' kicad_pro,' 'kicad_pcb' and 'kicad_sch.' The file with the 'kicad_pro' extension contains project information. The 'kicad_pcb' file contains layout information. The files with the 'kicad_sch' extension contain schematic information. There are two 'kicad_sch' files because this project includes two schematics.

Double-click on the 'kicad_pro' (project) file. The main KiCad window will appear. This window is the launchpad for the other KiCad apps, like Eeschema (the schematic editor) and Pcbnew (the layout editor). You can see the main KiCad window in Figure 1.5.2.

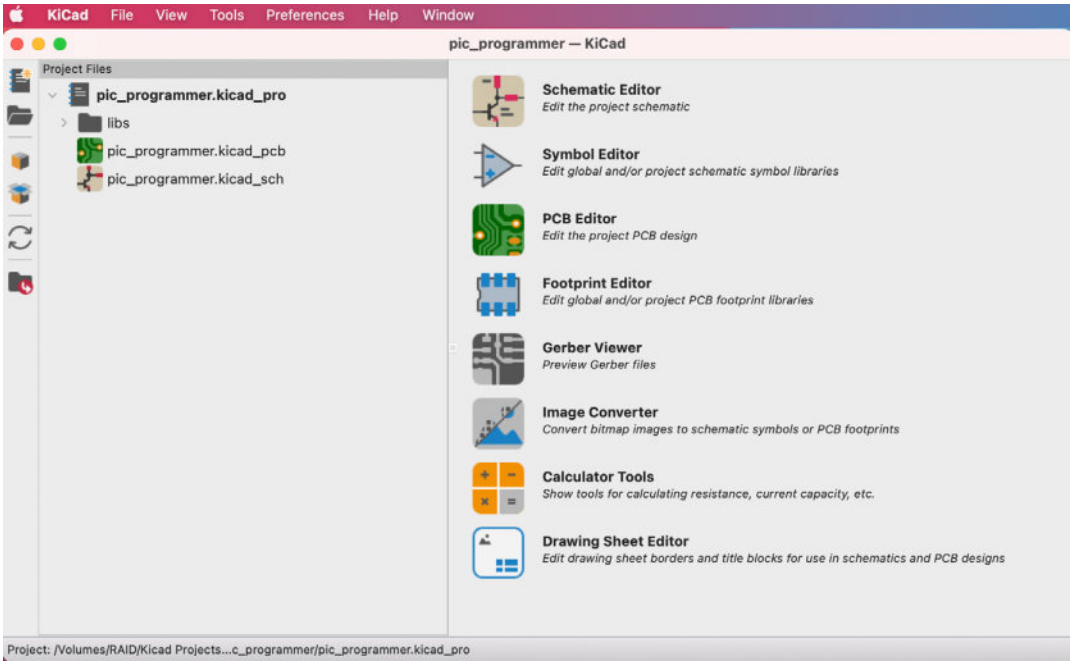


Figure 1.5.2: The main KiCad window.

Let's explore the schematic of this demo project. The main KiCad window shows the project files in the left pane, the various app buttons in the top-right pane, and various status messages in the bottom right pane. In the right pane, click on the Schematic Editor button. This button will start the Eeschema application, the schematic layout editor. You should see the editor as in the example in Figure 1.5.3.

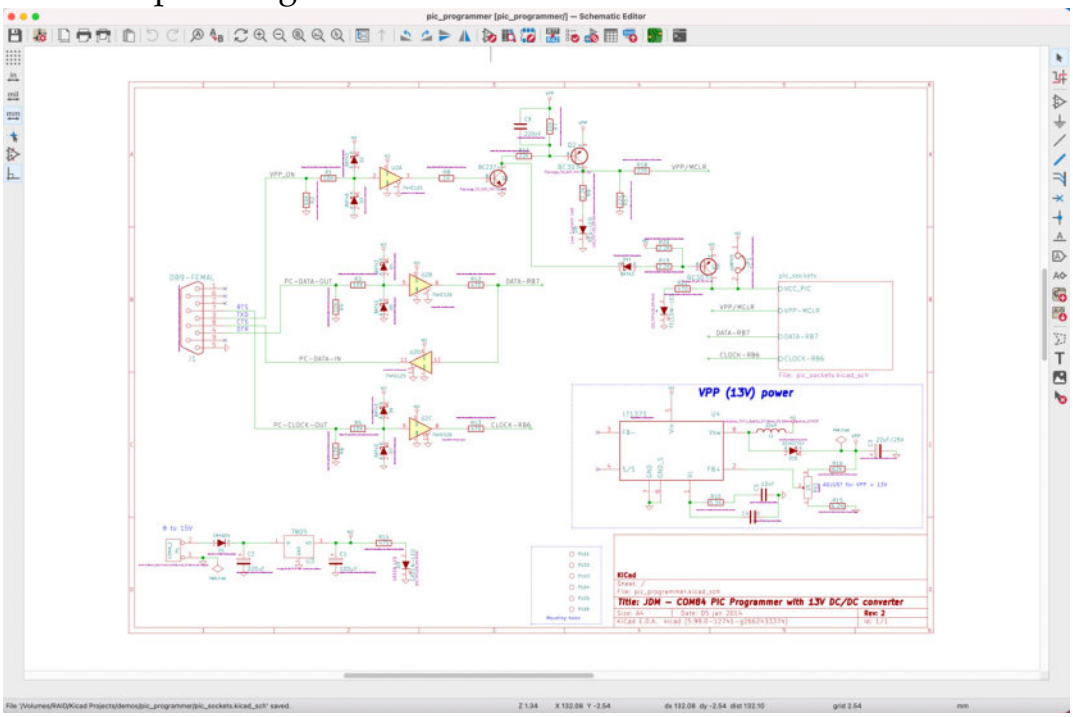


Figure 1.5.3: The schematic editor.

A few things are going on here. At first, this window might seem overwhelming. Don't worry about the various buttons and menus; concentrate on the schematic itself. Look at the various symbols, like those for the diodes, the transistors, and the operational amplifiers. There are symbols for resistors, and connectors, with green lines connecting their pins. Notice how text labels give names to the symbols but also the wirings between pins. Notice how even the mounting holes at the bottom right side of the schematic have names. Even though these mounting holes are not electrically active, they are depicted in the schematic. The values of the capacitors and resistors are noted, and any pins that are not connected to other pins are marked with an 'x's.

There is a rectangular symbol on the right side of the schematic with the title 'pic_sockets' (Figure 1.5.4).

Double click on it. What happened?

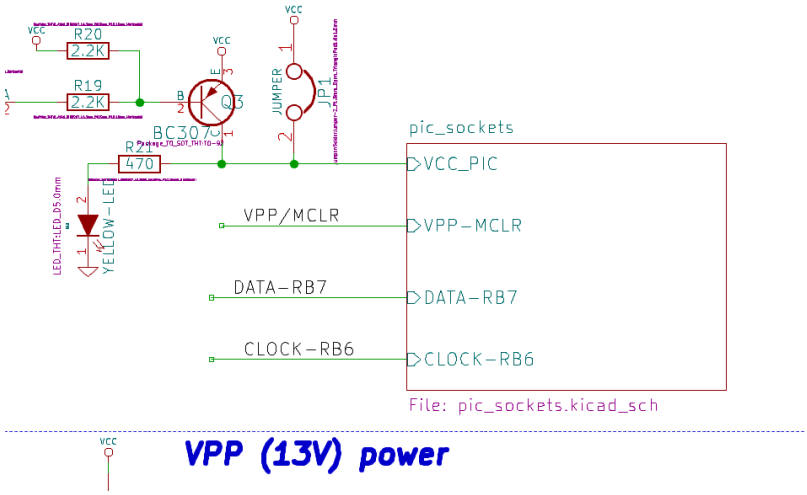


Figure 1.5.4: A link to another sheet.

This symbol links to another sheet, which contains additional symbols that are part of the same schematic. It looks like the example in Figure 1.5.5.

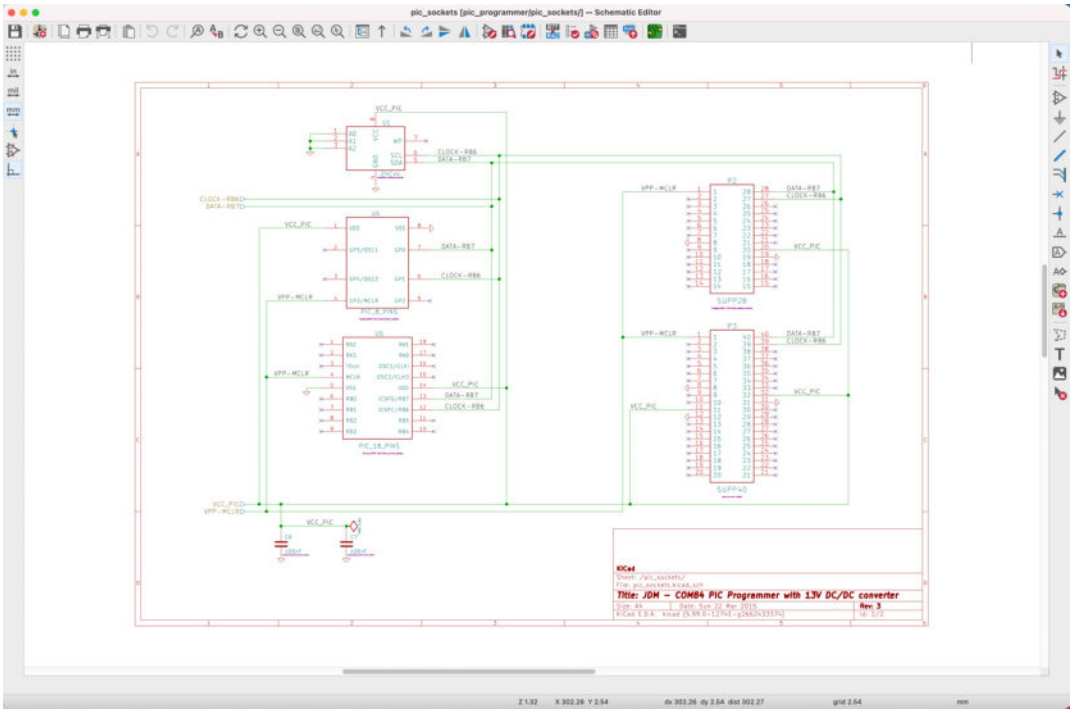


Figure 1.5.5: KiCad's schematics can span over multiple sheets.

KiCad's schematics can span over multiple sheets. Add more if your schematic is too large to fit in one sheet comfortably (you will learn how to do this in this book).

I encourage you to spend a bit of time studying this schematic. You can learn a lot about drawing good schematic diagrams by studying good schematic diagrams, just like you can learn programming by studying good open-source code.

Go back to the main KiCad window. Click on the button labeled "PCB Editor." This will launch Pcbnew, the layout editor. The window that appears will look like the example in Figure 1.5.6.

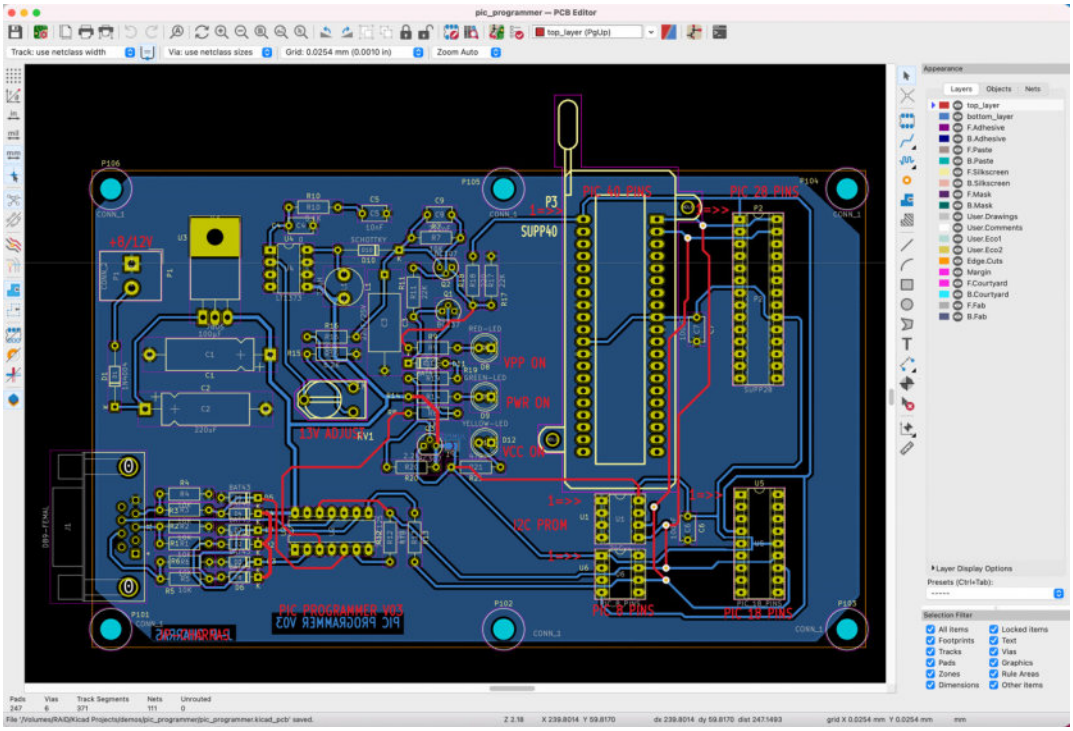


Figure 1.1.5.6: Pcbnew, the layout editor.

Again, don't worry about the various buttons and menus; concentrate on the layout inside the sheet. Use your mouse's scroll wheel to zoom in and out and the Alt+right mouse button to pan (you should also be able to pan by holding down the middle mouse button). Zoom in and look at some of the layout details, such as the pads, how they are connected to traces, the names that appear on the pads and traces, and the colors of the front copper and back copper layer traces. Note: in Linux, panning is done with the middle mouse button, and the alt key is not used.

Also, compare how a footprint in the layout compares to the symbol in the schematic. You can see a side-by-side comparison in Figure 1.5.7.

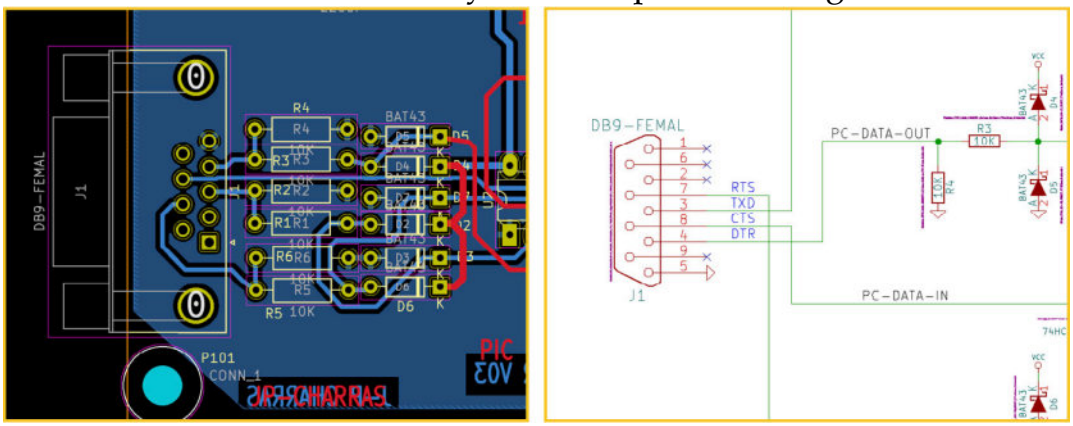


Figure 1.5.7: A side-by-side comparison of a footprint (left) and its schematic symbol (right).

Associated symbols and footprints have the same designator (J1, in this example) and the same number of pins. The layout shows the traces that correspond to the wires in the schematic.

Everything you see here is configurable: the width of the traces, which layer they belong to, the shape, size, and configuration of the pads. You will learn all of this in this book. In the layout, zoom in on the J1 connector to see one of its details: the name of the trace that connects pad 7 of J1 to pad 1 of R5. Traces, like everything else in KiCad, have names. The names of everything that you see in Pcbnew are defined (manually or automatically) in Eeschema.

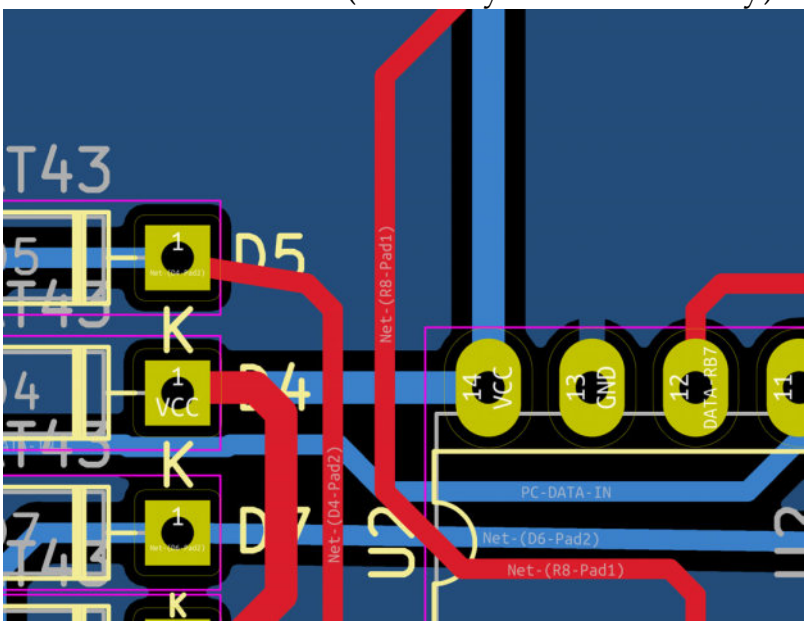


Figure 1.5.8: Traces have names.

Try one more thing: In Pcbnew, click on the View menu and choose the 3D Viewer. The 3D Viewer will show you a three-dimensional rendering of the PCB, with remarkable detail. You can zoom in and turn the board around to see it from any angle you want (Figure 1.5.9). Many components are populated, like the LED, resistors, and some of the integrated circuits. For the rest, you can still see their pads and outlines on the board.

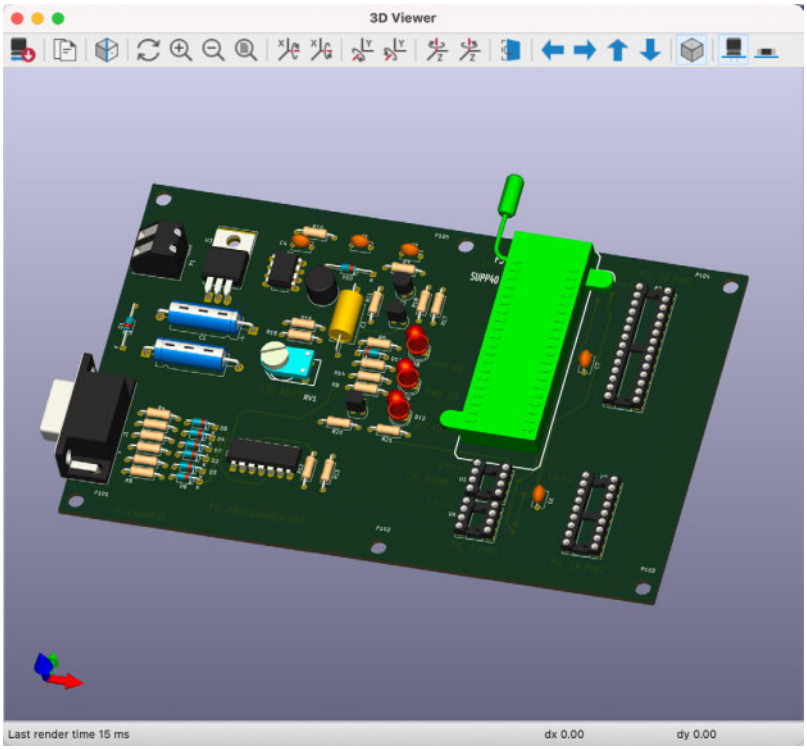


Figure 1.5.9: The 3D viewer will give you a realistic rendering of your board that you can examine in 3D.

As with the schematic editor, I encourage you to spend a bit of time studying the layout of this demo project. Later in this book, you will learn about the most important layout guidelines that will help you design well-functioning and elegant PCBs.

Apart from the demo projects that KiCad ships with, you should also look at some of the very impressive showcased projects of boards "made with KiCad". For example, the CSEduino is a 2-layer PCB that contains an Atmega328P microcontroller and implements a simple Arduino clone. You will be able to easily create a board like this by the time you finish this book. Go to txplo.re/madewkicad for more examples of projects made with KiCad.

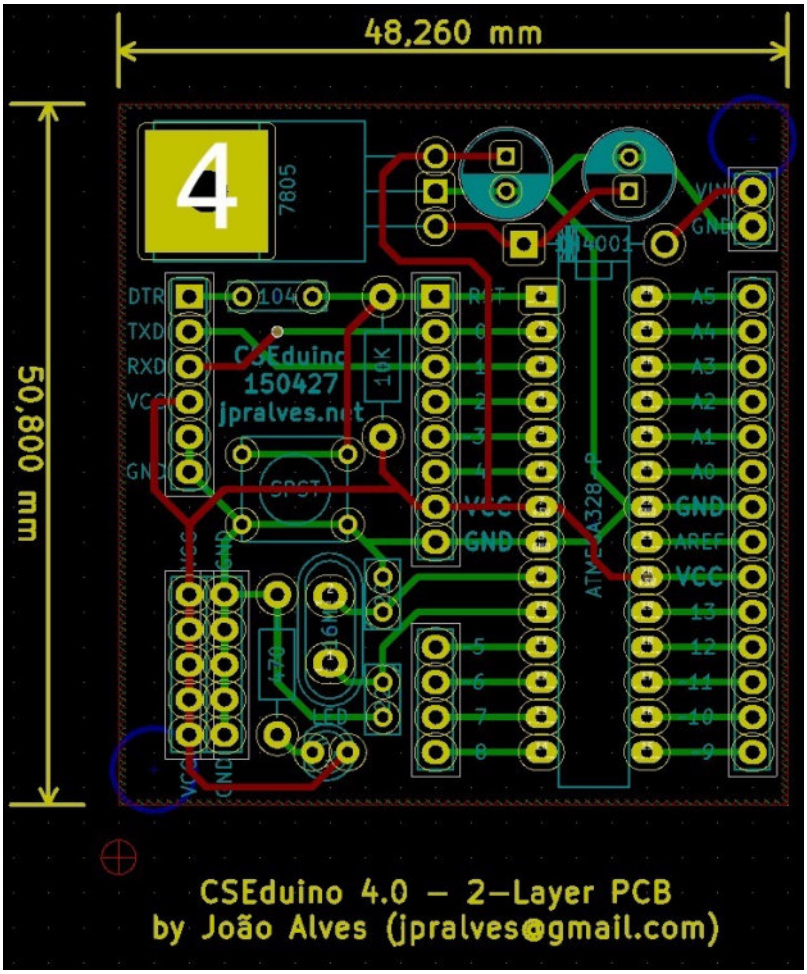


Figure 1.5.10: Featured board 'Made with KiCad': CSEduino.

Another featured board is Anavi Light, a HAT board for the Raspberry Pi. This is also a 2-layer board that allows you to control a 12V LED strip and get readings from sensors.

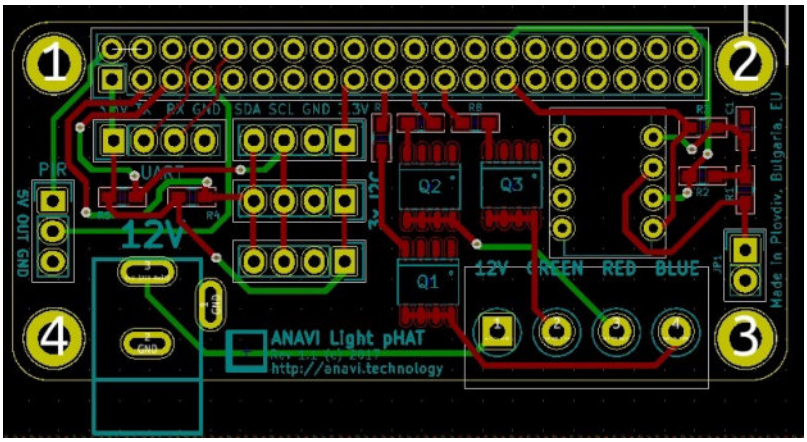


Figure 1.5.11: Featured board 'Made with KiCad': Anavi Light.

Finally, a truly impressive board made with KiCad is Crazyflie (Figure 1.5.12). Crazyflie is a dense 4-layer PCB with a rather elaborate shape. The board implements the flight controller of a tiny drone. The shape is

specifically designed to implement the drone's body and arms. You will also learn how to create PCBs with complicated shapes in this book.

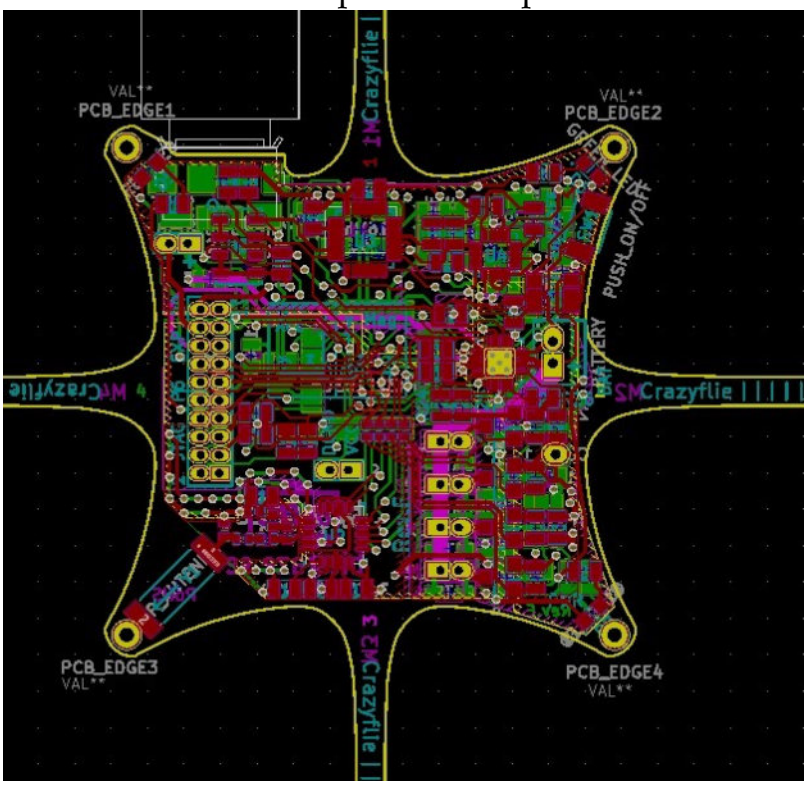


Figure 1.5.12: Featured board 'Made with KiCad': Crazyflie.

With this chapter complete, you should now understand the kinds of projects that people use KiCad. These are also the kinds of boards that you will design by the time you complete this book. Let's get straight into the first project so that you can start discovering this fantastic tool by doing.

Part 2: Getting started with KiCad 6

1. Introduction

Welcome to Part 2 of this book.

In the chapters of this Part, I will give you a brief overview of KiCad 6. This overview will help you with the first hands-on activity of this course, in which you will create your first PCB in the chapters of the following two Parts.

Ensure that you have installed KiCad on your computer so that you can follow along. If you haven't done so yet, please go back to chapter "4. Get KiCad for your operating system," where I provide information on installing KiCad on Mac OS, Windows, and Linux.

In the following chapters, I will introduce the individual apps that make up the KiCad software suite. I will also explain the roles of paths to the symbol, footprint, 3D model, and template libraries, show you how to create a new project from scratch and a template.

I will also compare KiCad 6 as it runs on the three supported platforms. If you have experience with KiCad 5, read the relevant chapter at the end of this Part.

Let's continue with the following chapter, where I'll give you an overview of KiCad's core apps.

2. KiCad Project Manager (main window)

This chapter will give you an overview of the KiCad project manager, otherwise known as the "main" KiCad window.

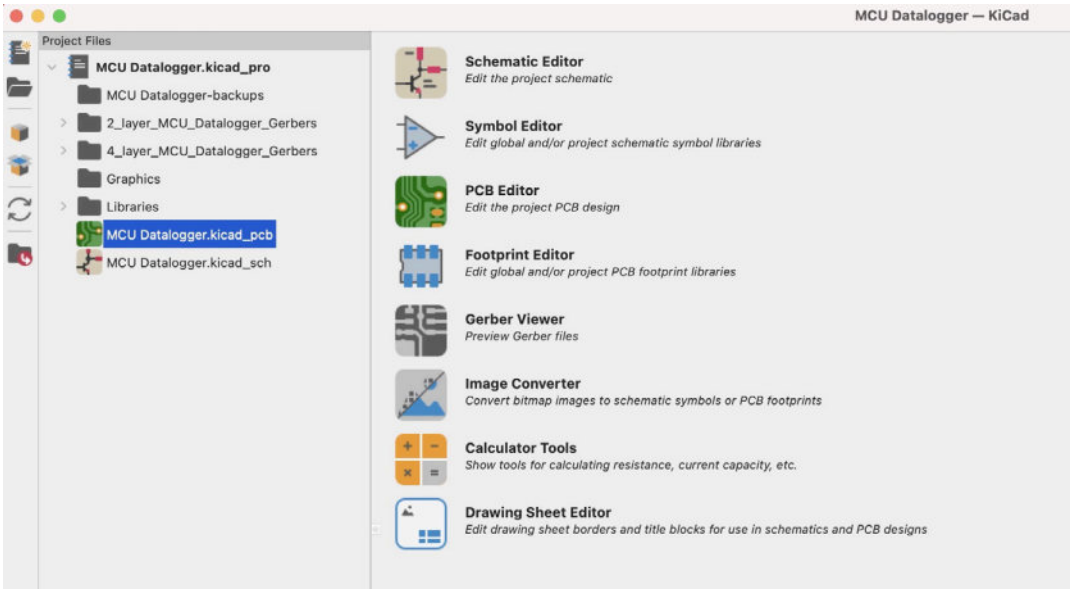


Figure 2.2.1: The KiCad Project Manager window.

This is the window that you will see first when you start KiCad. The project manager gives you access to the various KiCad applications, like the schematic and symbol editors, and shows you the project files.

The main window contains:

- A toolbar on the left.
- The project files are in the middle.
- The application buttons are on the right side.

The left toolbar has buttons to create a new project or open an existing project and archive/unarchive.

The middle pane shows the project files and folders. This is essentially a file browser that gives you access to the individual files and folders inside the main KiCad project directory.

The right pane contains buttons for the individual applications. Say that you want to start the schematic editor. You can do this in three ways:

1. Double-click on the file with the extension "kicad_sch" in the middle pane (file browser).
2. Click on the Schematic Editor button in the right pane.

3. Click on "Schematic Editor" under Tools in the top menu (see below).

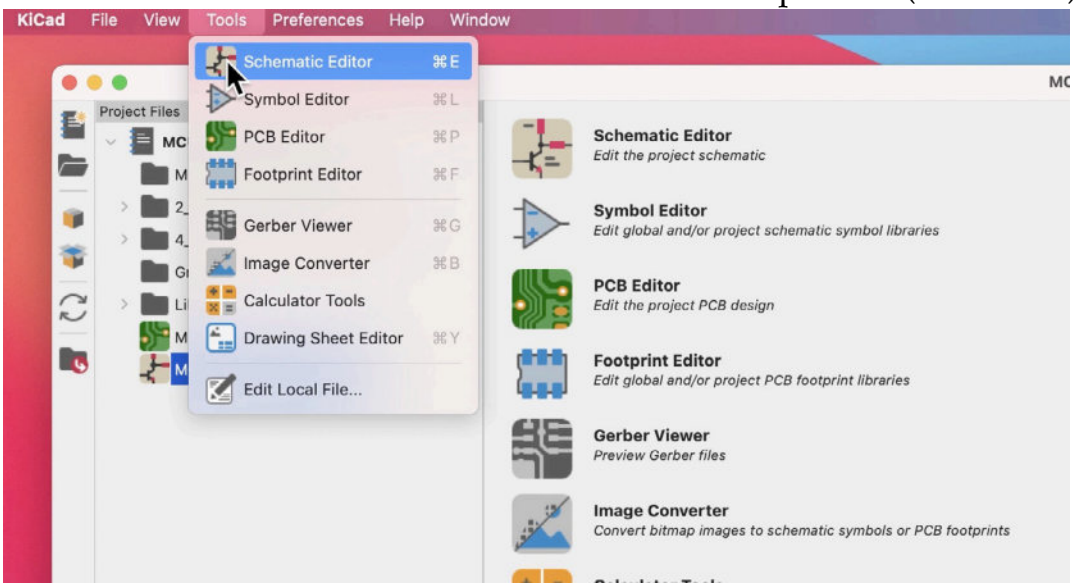


Figure 2.2.2: Starting the Schematic editor.

If you create a new directory via your operating system's file manager or create a new file, the middle pane will display those items. Remember that a KiCad project will contain files that KiCad creates and files created by other tools, like the Autorouting autorouter and Git. You will learn about the core files in KiCad later in this book.

You will learn about the buttons in the right pane in the next chapter.

First, let's do a tour of the items in the top menu bar. The top menu bar appears at the top of the screen on Mac OS, and the top of the KiCad window in Microsoft Windows.

Below you can see the KiCad main app in Mac OS with its menu bar in the top of the screen. I have opened the KiCad menu to reveal the "About KiCad" option.

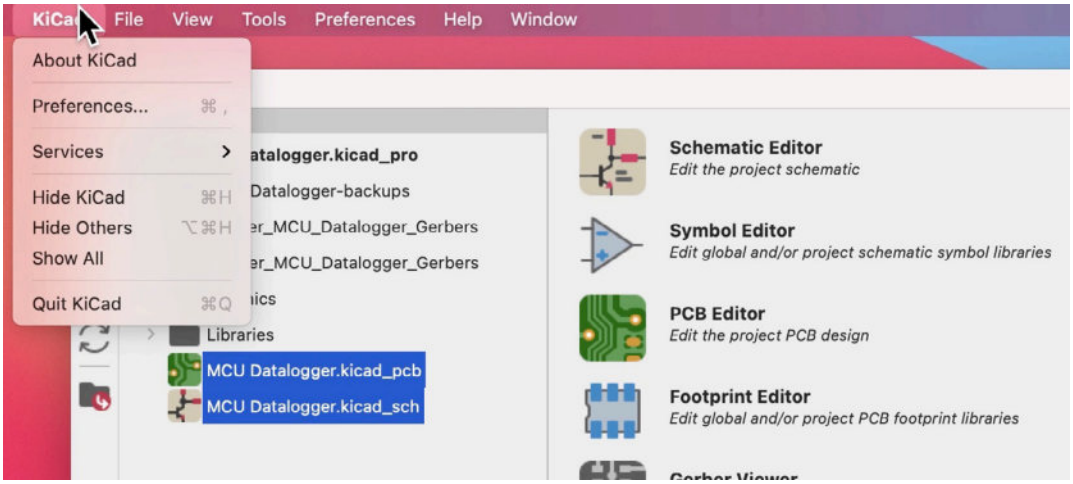


Figure 2.2.3: The top menu bar in KiCad (Mac OS).

Below you can see the KiCad main app in Microsoft Windows with its menu bar in the top of the KiCad window. I have opened the Help menu to reveal the “About KiCad” option.

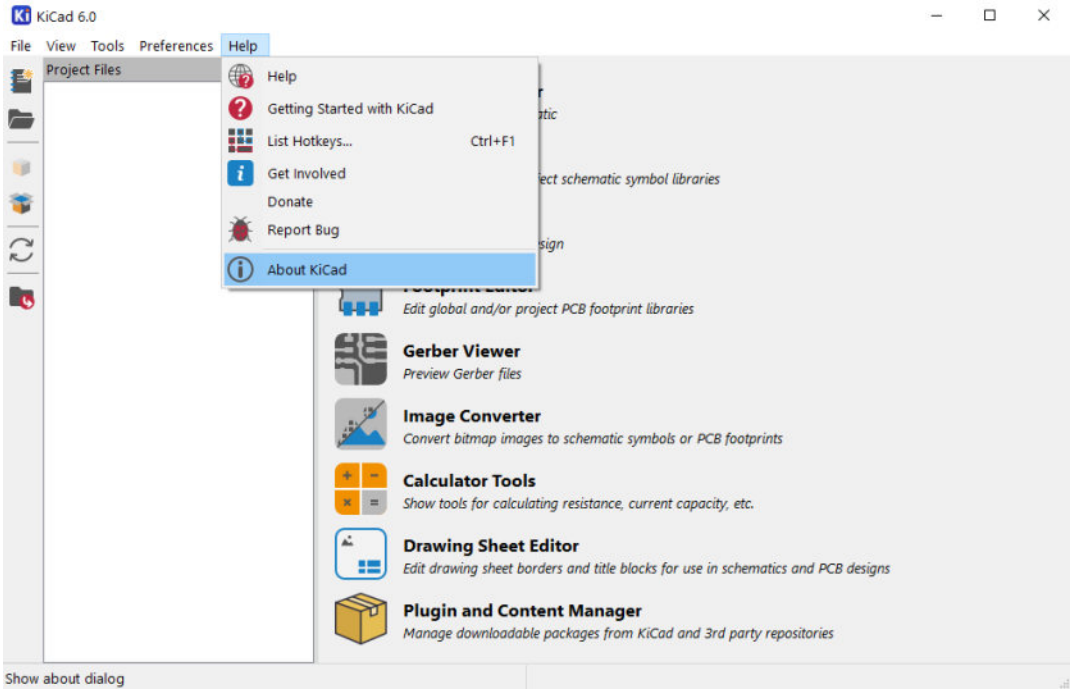


Figure 2.2.4: The top menu bar in KiCad (Windows).

To get information about your instance of KiCad, click on "About KiCad" under the KiCad menu item. You will need to use the information provided in this window if you have found a bug and wish to report it to the development team. Below you can see the “About KiCad” window in Mac OS, next to the New Issue page in Gitlab.

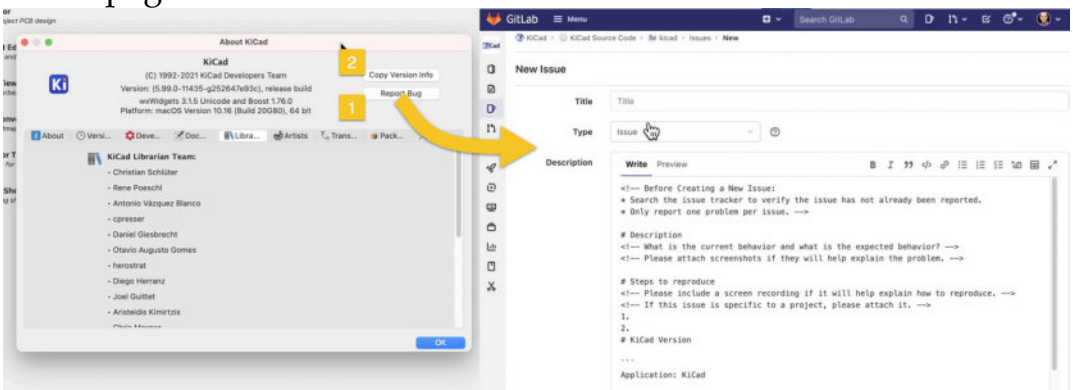


Figure 2.2.5: Report a bug.

Below you can see the “About KiCad” window in Microsoft Windows. The Linux version looks very similar.

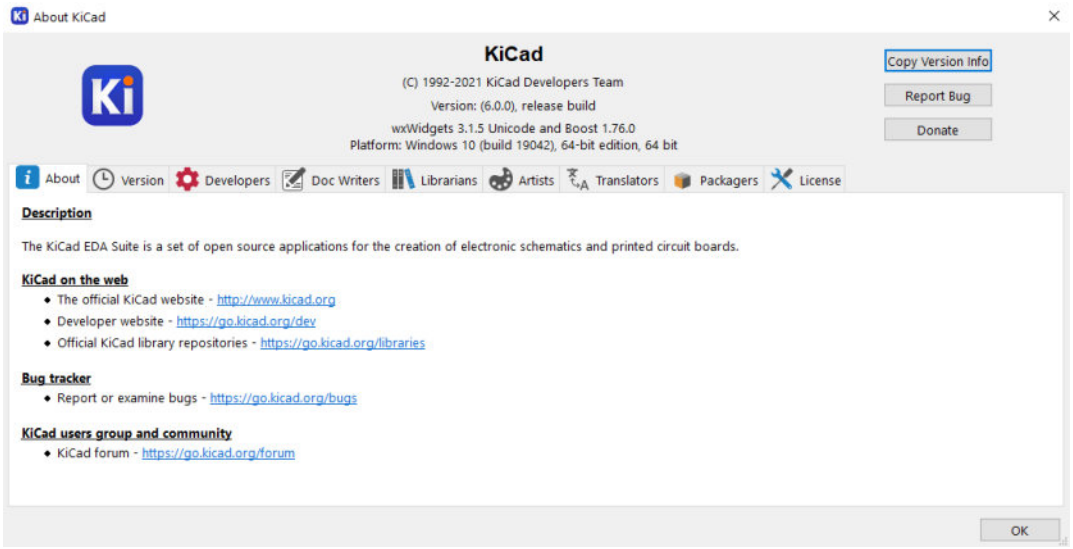


Figure 2.2.6: The KiCad "About" window in Microsoft Windows.

To report a bug, open the About KiCad window, and click "Report Bug" (see "1" above). This will use your web browser to open the New Issue page in Gitlab. You will need to include your KiCad instance version information, which you can get from the About KiCad window ("2", above).

Also, from the KiCad menu item, you can bring up the Preferences window.

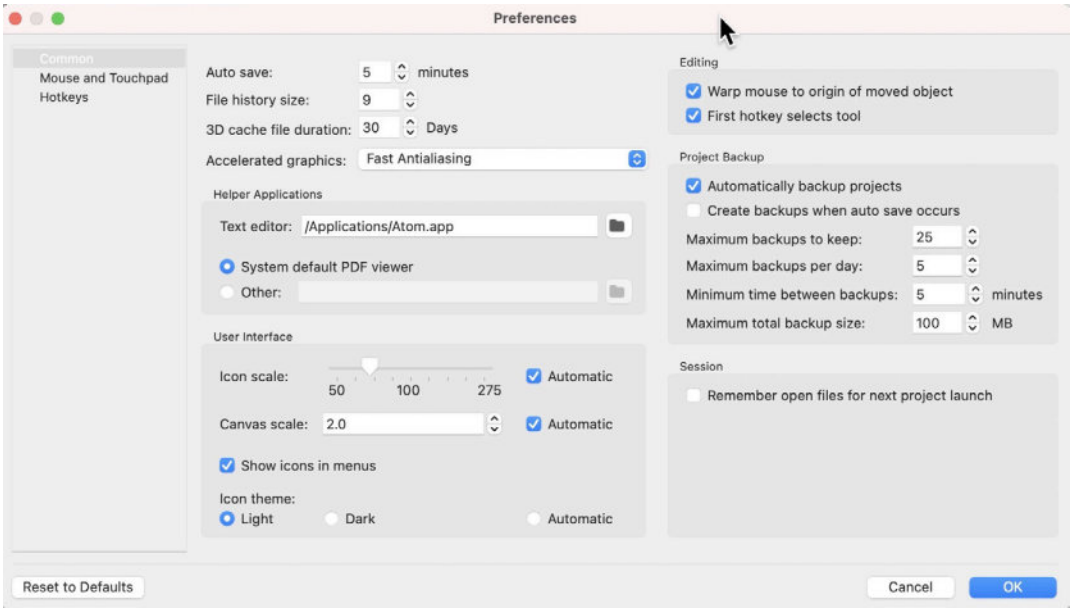


Figure 2.2.7: The KiCad Preferences window.

In the Preferences window contains several tabs with widgets that allow you to customize KiCad. Exactly what you see here depends on which applications are open. In the example above, only the main KiCad project window is open. The right pane would contain additional items if Eeschema

or Pcbnew were also open. You can learn about the details in dedicated chapters later in this book (Eeschema, and Pcbnew).

Under the File menu, you see the standard options for file and project management. You can open/close a project, create a new project, archive/unarchive a project, and import non-KiCad projects. You will find some of those options as buttons in the right toolbar of the main KiCad window.

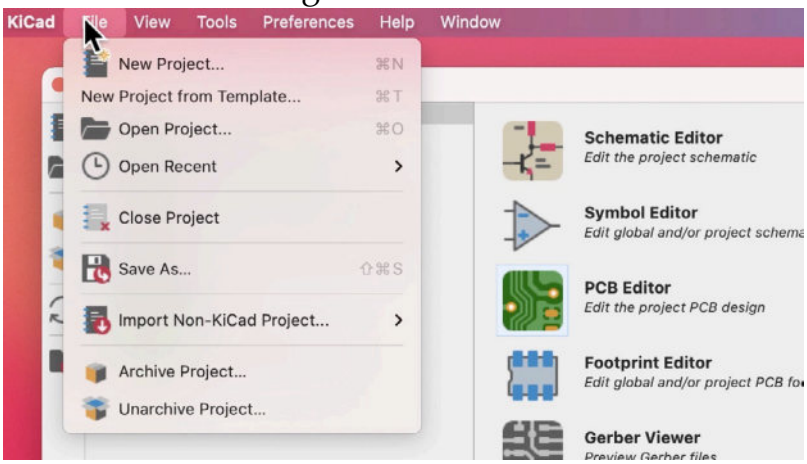


Figure 2.2.8: The KiCad File menu.

You will be using those options in the projects through this book. In the Recipes part of this book, you can learn how to import a non-KiCad project, and how to archive/unarchive.

Under View, you can use a text editor to view any of KiCad's project files. You can define your preferred text editor in the Preference window in the Common tab. Below you can see an example of a KiCad schematic file loaded in the Atom text editor.

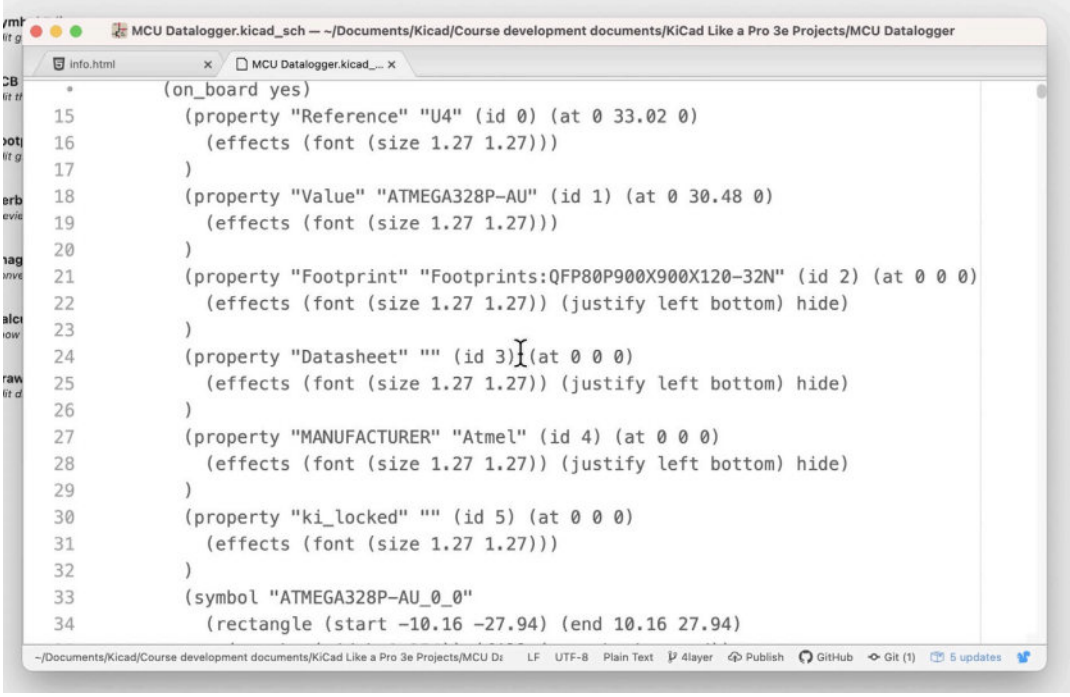


Figure 2.2.9: A schematic design file in a text editor.

All KiCad files are text files, and as such, you can open them in a text editor. It is also possible to programmatically edit those files using automation implemented in a language like Python directly, without needing an API. Beware, though: modifying these files by hand or programmatically without knowing precisely what you are doing will most likely damage your KiCad project. Always back up your work before any such experimentation!

The Tools dropdown menu gives you access to the individual apps in the KiCad software suite. The items in this menu replicate the application buttons in the right pane of the KiCad main window.

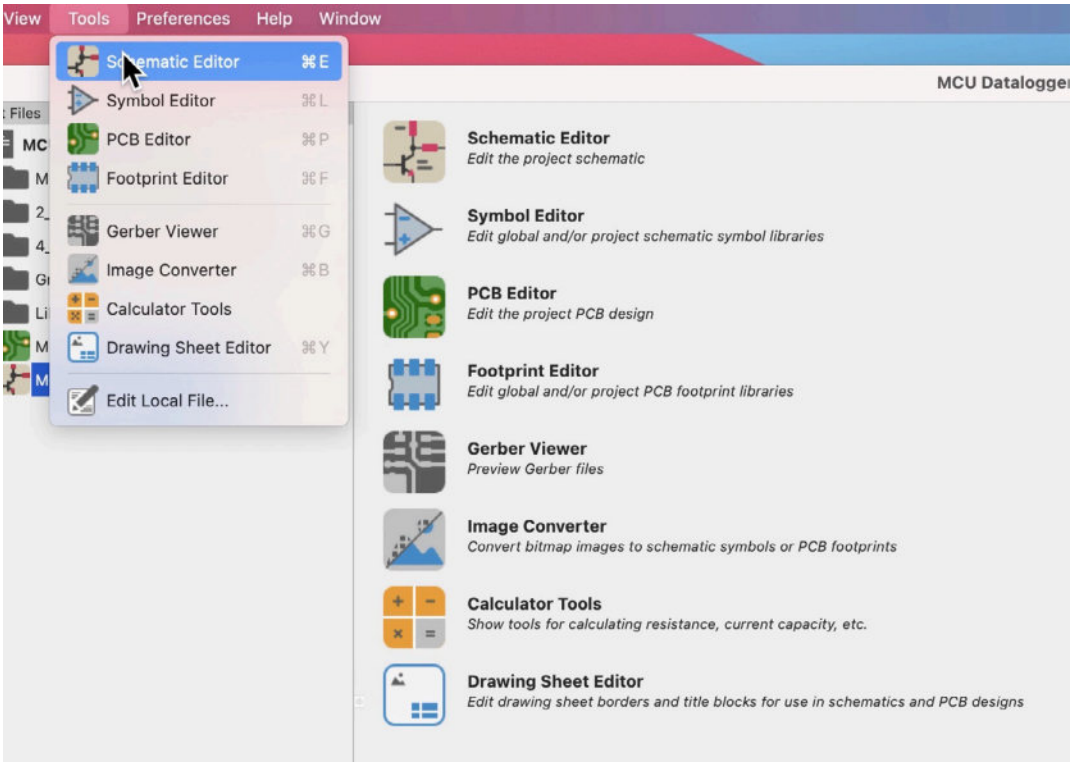


Figure 2.2.10: The Tools menu items.

I will describe these applications in the next chapter.

Under Preferences (not to be confused with the Preferences window under "KiCad"), you can access the Paths, Symbol Libraries, and Footprint Libraries manager windows.

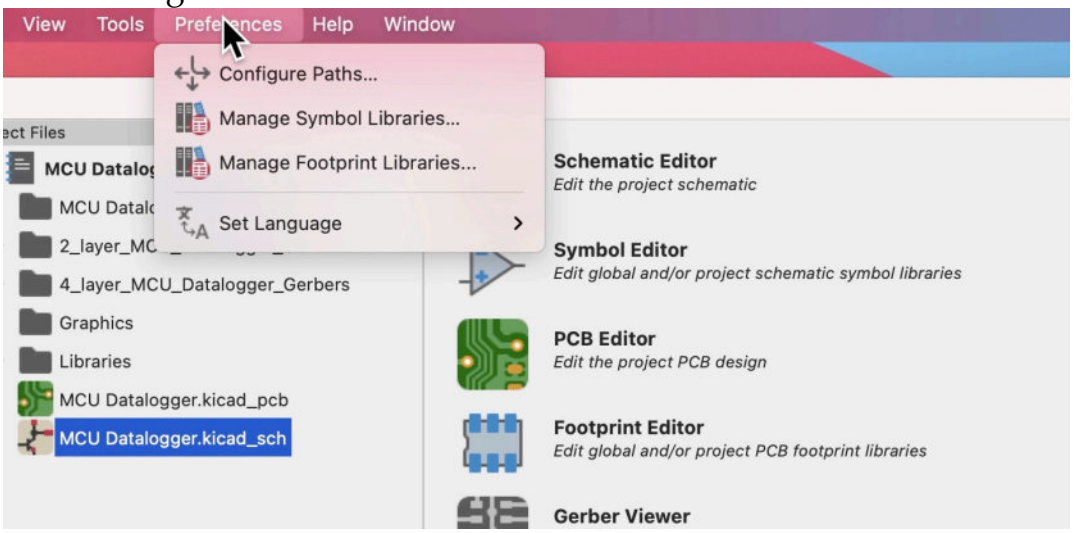


Figure 2.2.11: The Preferences menu.

I have written a dedicated chapter on these manager windows with details later in this Part of the book.

Finally, the help menu. It allows you to access a local copy of the official KiCad documentation, which opens in your browser, and a window that contains a list of hotkeys. Be mindful that this documentation may be old.

When I am writing this, this documentation has not been updated since KiCad 5.0.0-rc2, and most of the links are not working.

The Hotkeys window, apart from listing current hotkeys, allows you to make changes. I prefer to keep the default hotkeys unless there is a conflict with other applications running on my computer.

This was an overview of the main KiCad window, the KiCad project manager. In the next chapter, you will learn about the individual applications that make up the KiCad software suite.

3. Overview of the individual KiCad apps

In the previous chapter, you learned about the KiCad Project Manager. This chapter will give you a tour of the individual applications that make up the KiCad software suite.

As you may recall from the previous chapter, you can access the KiCad applications via the project manager's right pane or the Tools menu. To open Eeschema or Pcbnew, you can also double-click on the schematic and layout files listed on the middle page of the project manager.

Let's take a closer look at each of the KiCad applications.

Schematic editor: Eeschema

Click on the Schematic Editor button to open the application. You can see the editor window below.

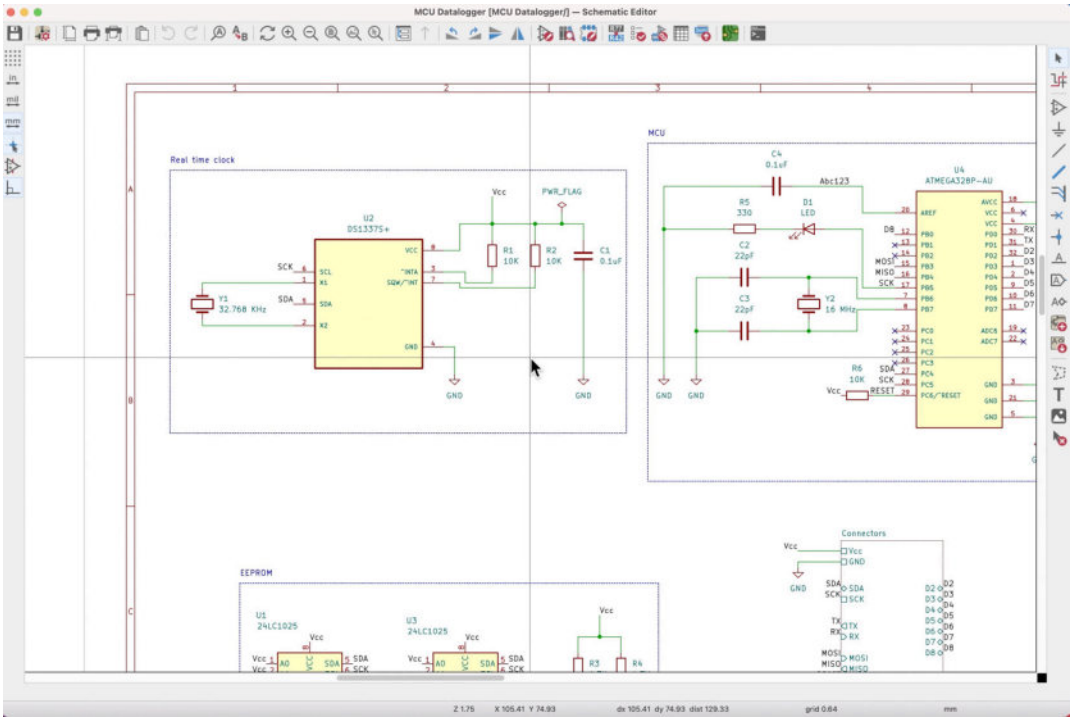


Figure 2.3.1: Eeschema, or the Schematic Editor.

You use Eeschema to draw the schematic of the PCB. Although KiCad is flexible enough and allows you to create PCBs without a schematic, this is rarely a good idea. The schematic diagram captures all necessary information that the layout editor uses: components (as symbols), wires that connect pins

nets, and various kinds of netlabels, busses, power nets, and much more. Eeschema is the first KiCad application you will use when you start a new KiCad project.

In the example above, you can see a schematic from one of the projects in this book. You can see the symbols (such as U2, R1, and R2), green wires connecting pins, special symbols representing unconnected pins and power nets, and other elements like graphics and text labels.

You can learn how to use and configure the schematic editor in a dedicated Part of this book.

Layout editor: Pcbnew

Once you have completed work in Eesceham, you will continue with the Layout editor, or "Pcbnew." To open Pcbnew, you can click on the Pcbnew button in the KiCad project manager or the Pcbnew button in the top toolbar of Eesceham. Below you can see an example instance of Pcbnew.

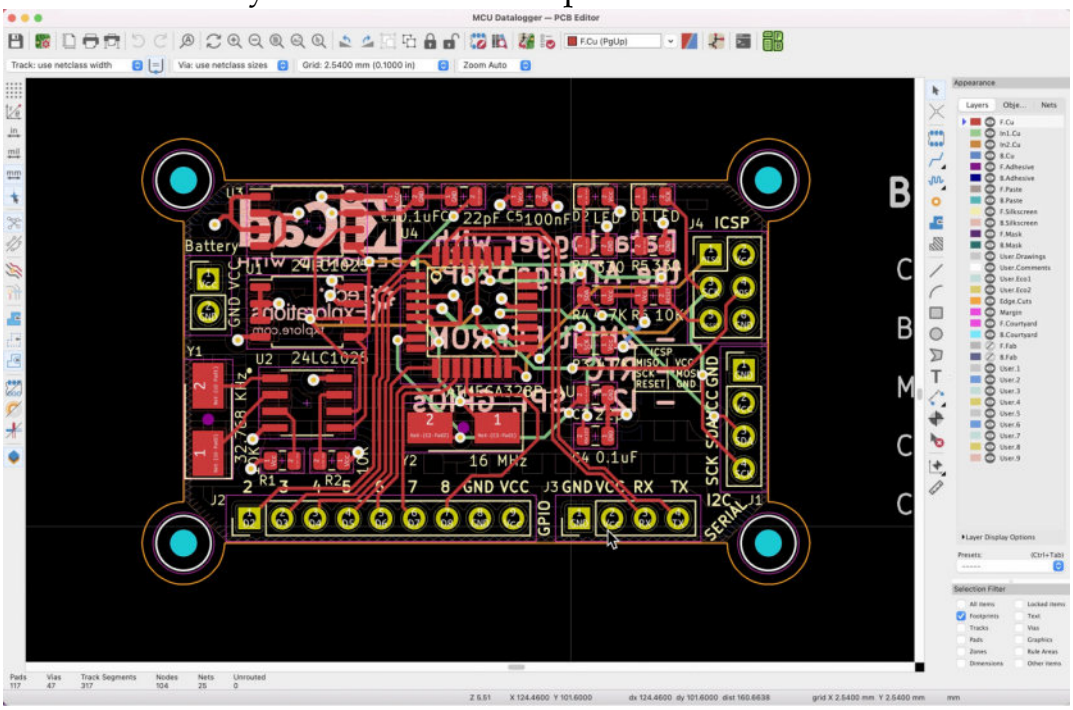


Figure 2.3.2: Pcbnew, or the Layout Editor.

In the example above, you can see the finished PCB design from one of the projects in this book. The layout editor allows you to select the layers and design elements you want to see. For example, you can enable or disable the visibility of layers, footprints, tracks, zones, and vias. In the example above, I have enabled the visibility of all layers and elements and an outline of the top and bottom copper zones.

The layout editor includes various sophisticated tools, such as an interactive router and a 3D viewer. You can see a 3D rendering of the PCB from Figure Figure 2.3.2 below:

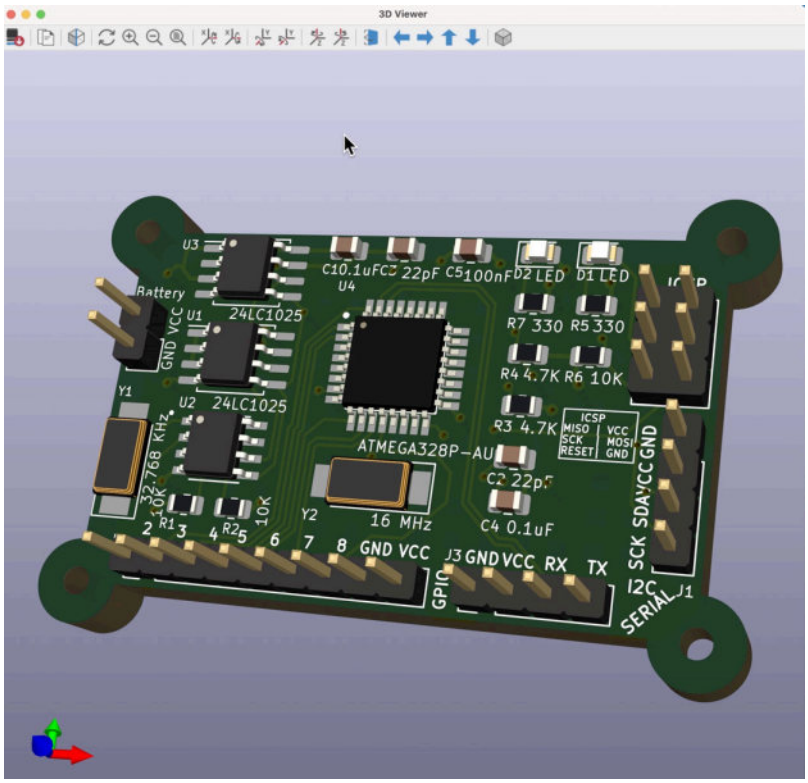


Figure 2.3.3: The 3D viewer in Pcbnew.

You can learn how to use and configure the layout editor in a dedicated Part of this book.

Symbol Editor

Let's continue with the Symbol Editor. You can open this application from the KiCad Project Manager or the top toolbar of Eeschema.

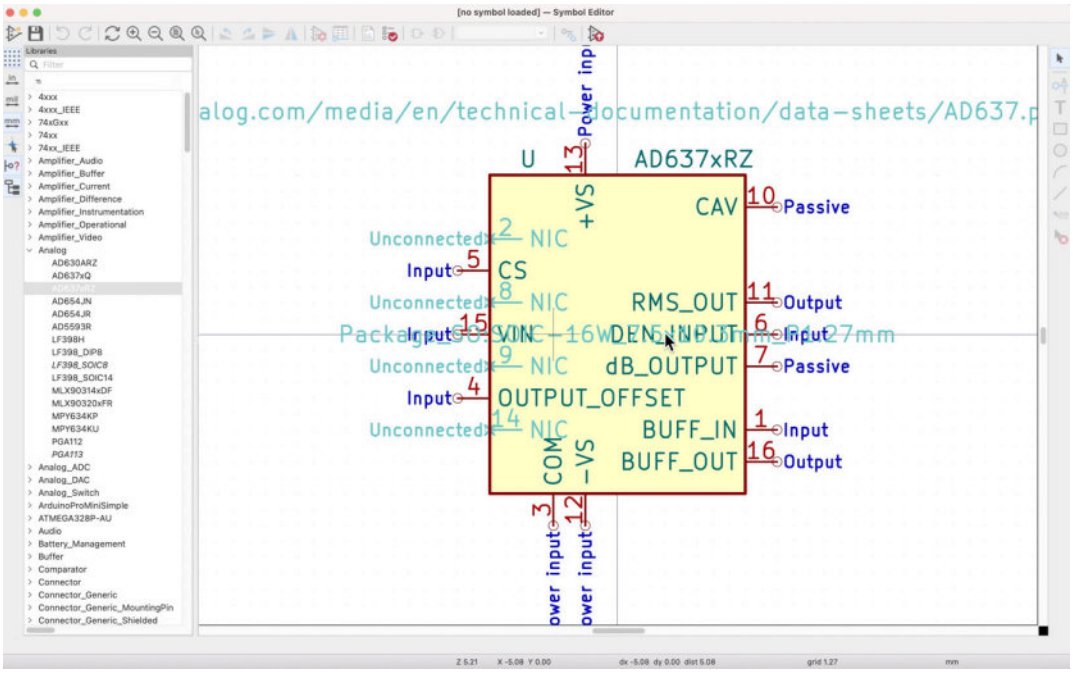


Figure 2.3.4: The Symbol Editor.

With the Symbol Editor, you can modify existing symbols or create new ones. You can think of the Symbol Editor as a simplified version of the schematic editor. In the Symbol Editor, you can work with a single symbol at a time.

KiCad 6 comes with an extensive set of symbol and footprint libraries. There are also thousands of third-party symbols and footprints that you can import. However, you will eventually need to create a symbol, and that's when the Symbol Editor comes in.

You can learn how to create new symbols from scratch later in this book.

Footprint editor

Similar to the symbol editor, there is also the footprint editor. You can open the footprint editor from the KiCad project window or the Pcbnew top toolbar.

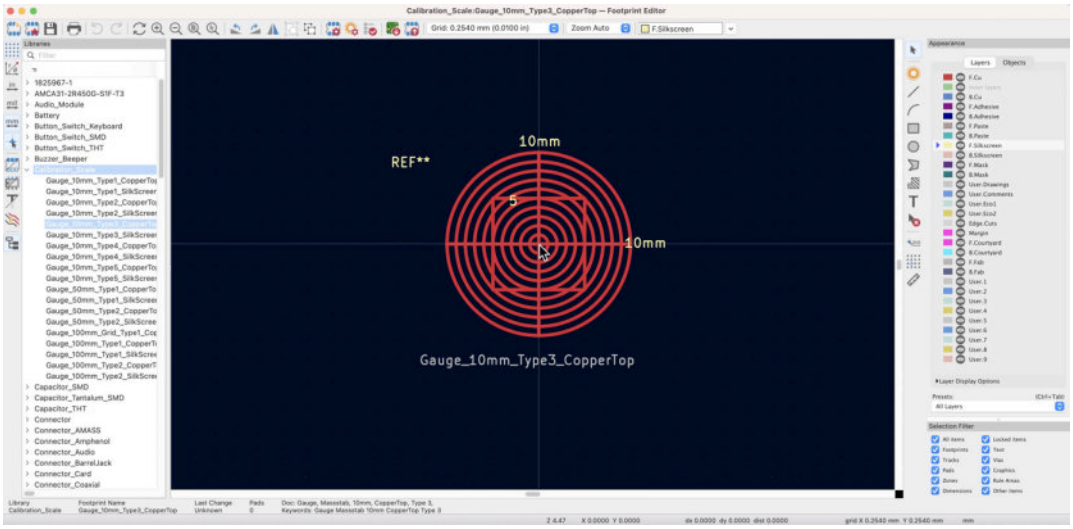


Figure 2.3.5: The Footprint Editor.

With the footprint editor, you can create a footprint from scratch or modify an existing footprint. The footprint editor also contains a wizard that allows you to quickly generate footprints that follow convention, such as those that use BGA, QFN, DIP and SOIC, packages.

You can learn how to use the footprint editor in a dedicated chapter.

Gerber Viewer

When you have completed work on your PCB and wish to order it from an online manufacturer, the most common way is to export a set of Gerber files from Pcbnew. Before you upload those files to your preferred manufacturer, you should take the time to inspect them. KiCad has a tool for this: the Gerber Viewer.

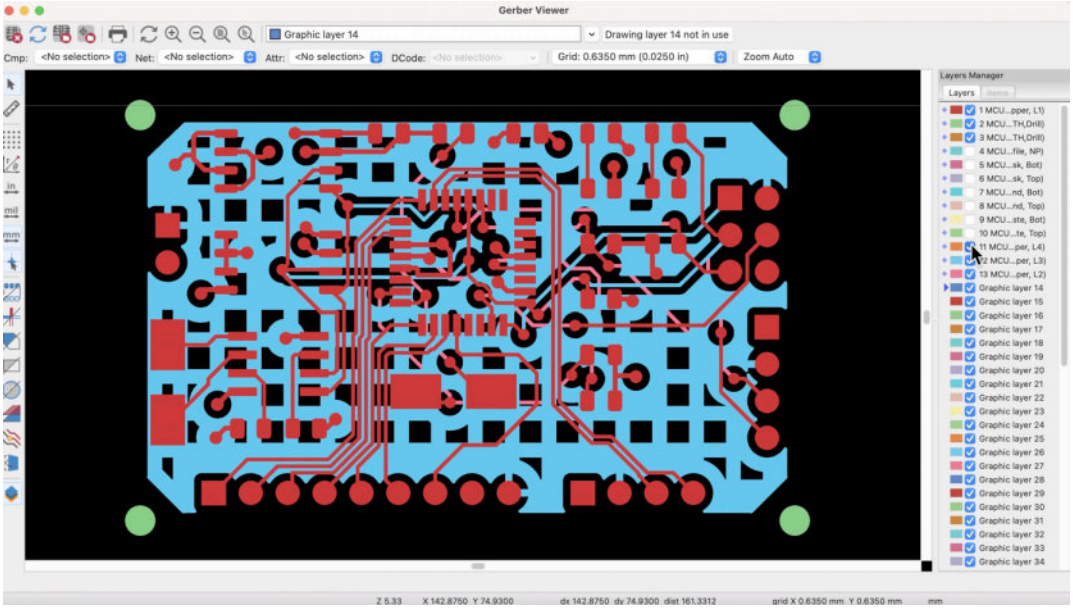


Figure 2.3.6: The Gerber Viewer.

With the Gerber Viewer, you can examine the project Gerber files visually, layer by layer. This way, you can ensure that all its elements are correct. Silkscreen text and graphics, drills, copper fills, the board outline, and cutouts, etc.

Think of the Gerber Viewer as a quality control tool. Use it to reduce or eliminate the risk of ordering a defective PCB.

You can learn how to export the Gerber files and use the Gerber Viewer (and online Gerber viewers) in dedicated chapters later in this book.

Image Converter

You can open the image converter app from the KiCad Project Manager. With the Image converter, you can convert a bitmap image into a footprint. Typical uses of the converter are to create a graphics footprint (such as a company logo) or a footprint with an irregular shape that would be too tedious to design in the footprint editor.

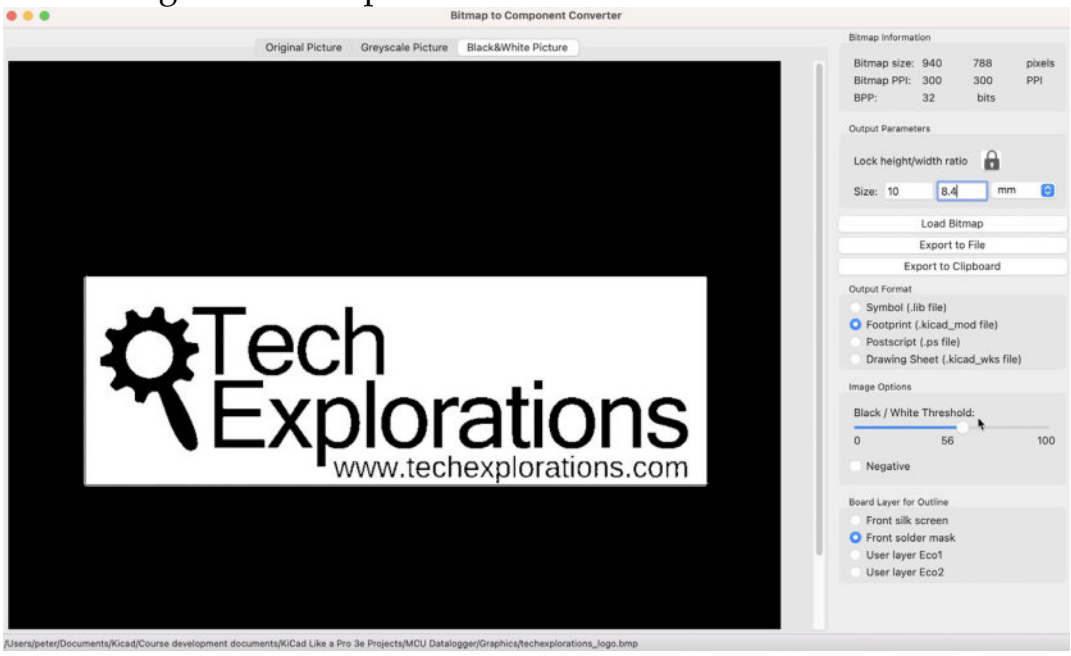


Figure 2.3.7: The Image Converter.

In the example above, I use the Image Converter to create a logo that I can include in my PCBs. You can learn how to use the Image converter in a dedicated chapter in Part 13 of this book.

Calculator tools

The calculator tool contains multiple calculators. Here is a list of tools:

1. Voltage regulators.
2. RF Attenuators.

3. E-Series.
4. Resistor color codes.
5. Transmission lines.
6. Via size.
7. Track Width.
8. Electrical spacing.
9. Board classes.

In the example below, I am using the Track Width calculator to calculate the correct width given a set of parameters.

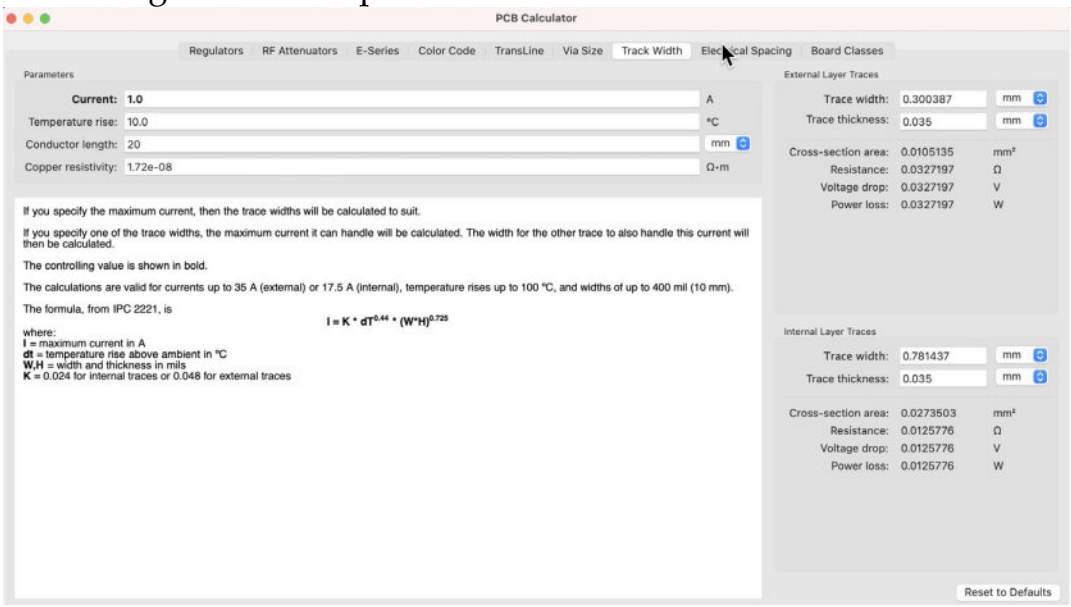


Figure 2.3.8: The Calculator tool.

You can learn how to use the Track Width calculator by reading the relevant chapter in the Recipes part of this book. The mode of operation for the rest of the calculators is similar.

Drawing Sheet editor

The last main application in the KiCad suite is the Drawing Sheet editor. You can use this editor to customize your schematic editor sheet. You can see the editor in the example below.

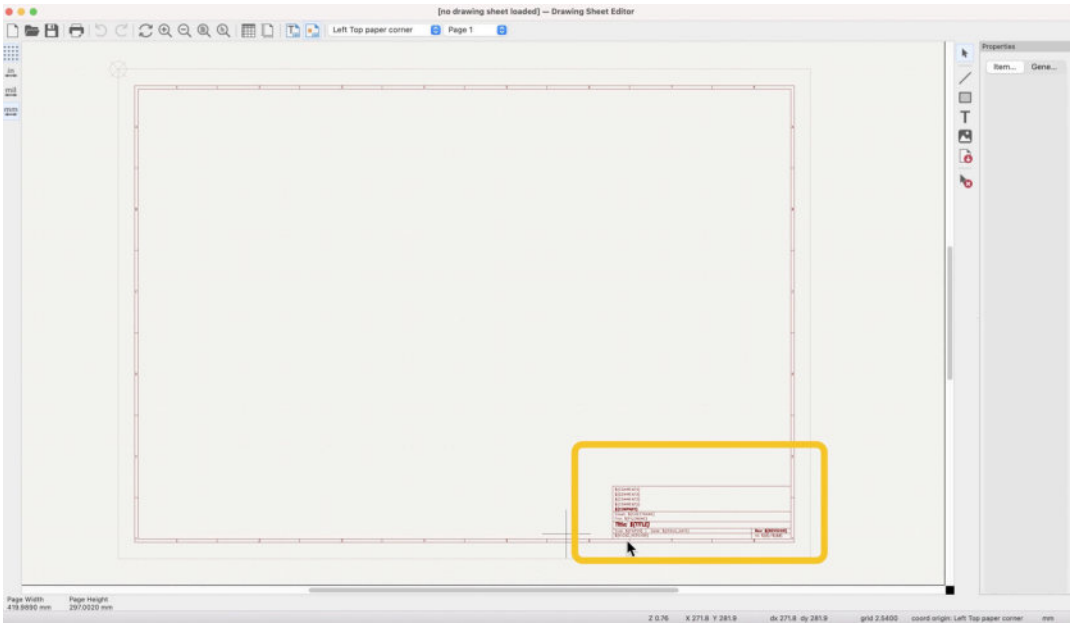


Figure 2.3.9: The Drawing Sheet editor.

With the Drawing Sheet Editor, you can change the size of the schematic sheet and everything within it. For example, you can remove or change the size and location of the information container. You can also change the setup of the text placeholders inside the information box.

To learn how to use the Drawing Sheet Editor, please read the relevant chapter in the Recipes part of this book.

4. Paths and Libraries

In the KiCad project window, you will find the paths and libraries configurations options under the preferences menu item.

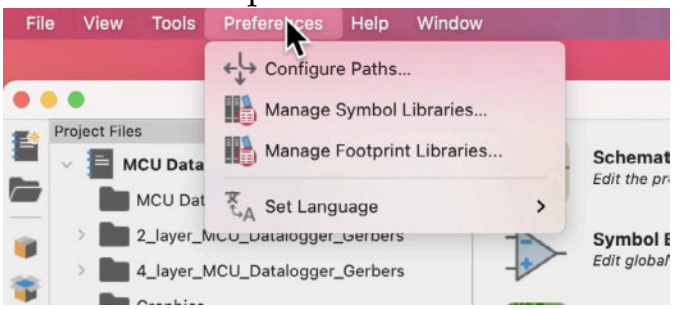


Figure 2.4.1: The Preferences menu.

Let's look at each one.

Configure Paths

Bring up the "Configure Paths" window from the Preferences menu.

This window contains a table to environment variables that contain paths to important collections of files.

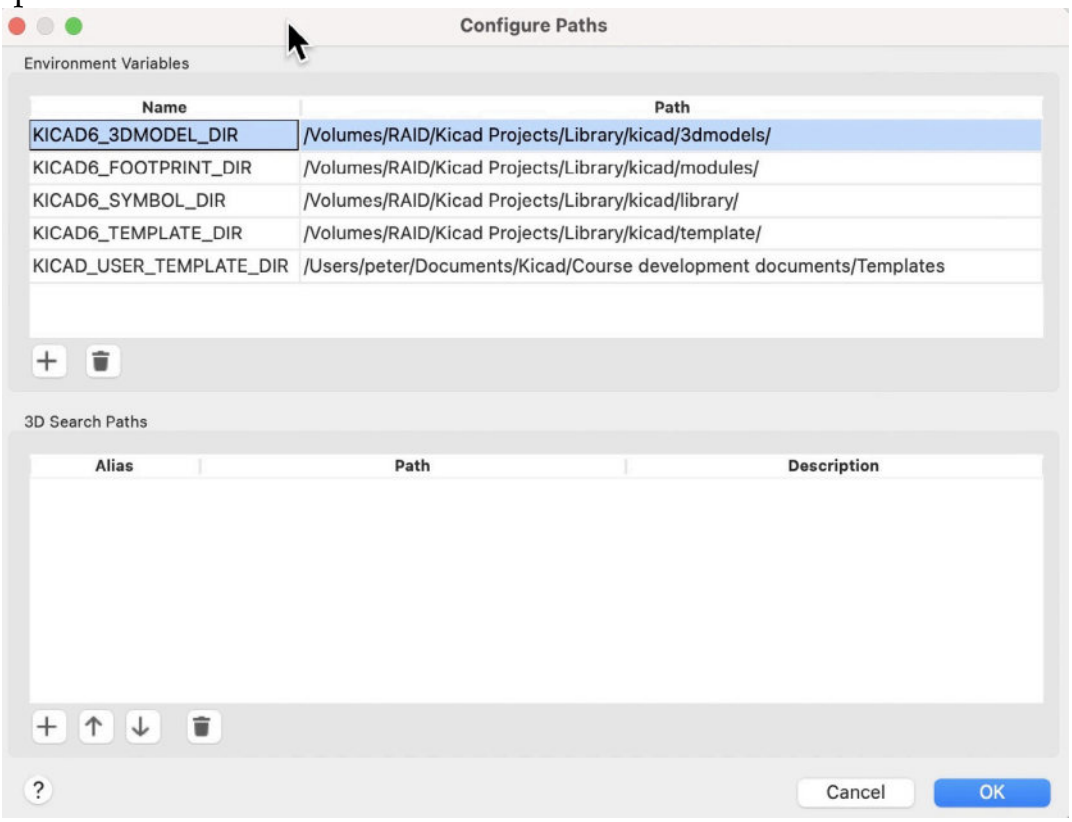


Figure 2.4.2: The "Configure Paths" window.

As you can see in the figure above, there are five path environment variables:

- KICAD6_3DMODEL_DIR: points to a directory that contains 3D models of components for use by the 3D viewer. Learn more about this in a dedicated chapter.
- KICAD6_3RD_PARTY: points to a directory that contains 3rd party plugins, libraries, and other downloadable content.
- KICAD6_FOOTPRINT_DIR: points to a directory that contains footprint files for use by Pcbnew. Learn more about this in a dedicated chapter.
- KICAD6_SYMBOL_DIR: points to a directory that contains symbol files for use by Eeschema. Learn more about this in a dedicated chapter.
- KICAD6_TEMPLATE_DIR: points to a directory that contains sheet template files for use by Eeschema. Learn more about this in a dedicated chapter.
- KICAD_USER_TEMPLATE_DIR: points to a directory that contains project template files created by the user. You can use these template files to start a new project quickly. Learn more about this in a dedicated chapter.

When you install KiCad, these variables will inherit default values that point to the KiCad application installation folder. You can use the Configure Paths window to change these values.

For example, my computer has a solid-state drive with a limited amount of available space on it. Because the libraries (especially the 3D models) take several gigabytes of storage, I have opted to use my external RAID drive for those resources. As you can see in Figure 2.4.2 above, the footprint, symbol, and 3D model paths point to my external RAID drive, while the rest point to locations on the internal SSD.

Manage Symbol Libraries

Use the symbol libraries manager to:

- Add new symbol libraries.
- Delete symbol libraries.
- Activate or deactivate symbol libraries.

The Symbol Libraries window contains a list of active or inactive libraries installed in your KiCad instance. Each library may contain one or more schematic symbols. When a library is installed and activated, you can use its symbols in your schematics in Eeschema.

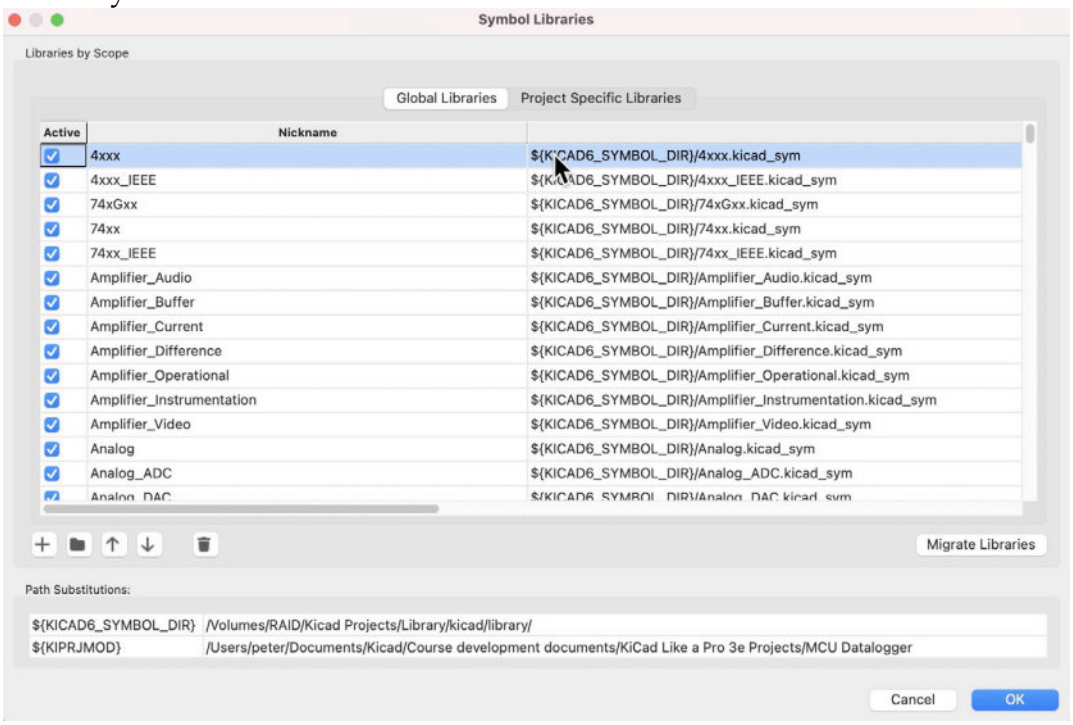


Figure 2.4.3: The “Symbol Libraries” window.

In the figure above, you can see the Symbol Libraries window with several of the libraries installed in my instance of KiCad.

Notice that:

- The table contains two tabs: “Global Libraries” and “Project Specific Libraries.” You can manage libraries under each tab to control the library visibility (global or project-specific).
- Each library has a name and a path. The path can use an environment variable, as in the example above. Alternatively, you can set an absolute path to a library; this is often a good option when you want to install a library stored outside the standard environment paths.
- If you forget the environment variable paths, look at the bottom of the window. In the table “Path Substitutions,” you can see the actual path stored in the environment variables.

Learn how to use the symbol libraries manager in a dedicated chapter later in this book.

Manage Footprint libraries

Use the footprint libraries manager to:

- Add new footprint libraries.
- Delete footprint libraries.
- Activate or deactivate footprint libraries.

The footprint libraries manager window works similarly to the symbol libraries manager.

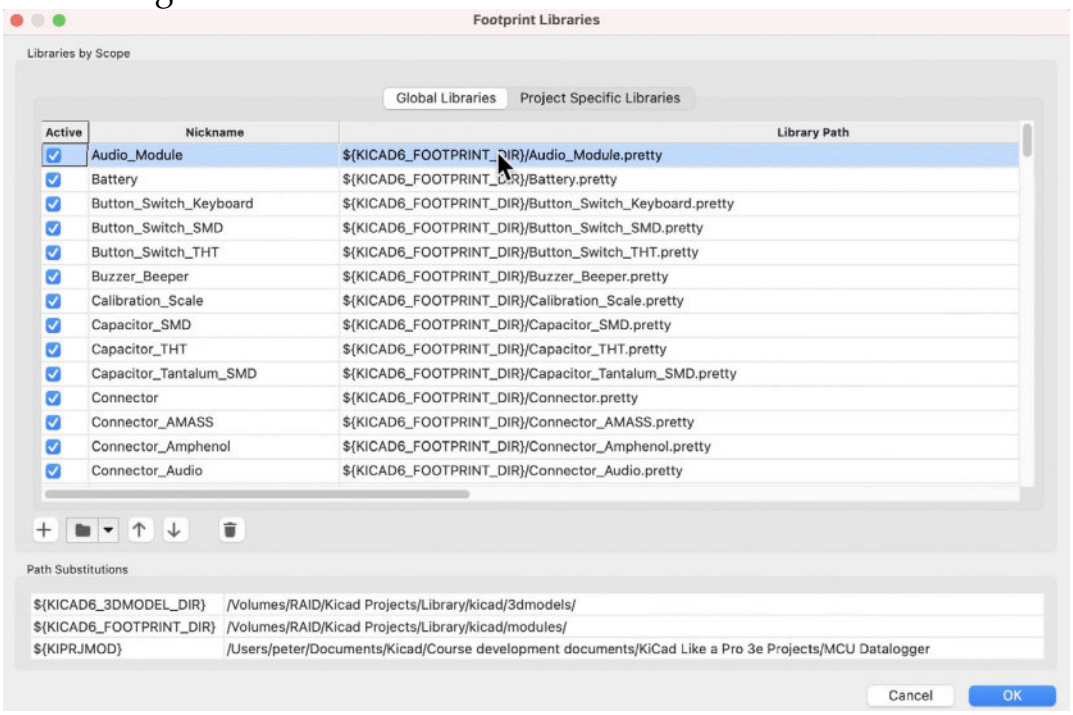


Figure 2.4.4: The “Footprint Libraries” window.

You can control the context of a library by listing them under the “Global Libraries” or “Project Specific Libraries” tab. Each library has a name and a path, and the path may contain an environment variable or an absolute path. Learn how to use the footprint libraries manager in a dedicated chapter later in this book.

5. Create a new project from scratch

In this chapter, you will learn how to create a new KiCad project.

Kicad offers you two ways to start a new project:

1. A new blank project.
2. A new project from a template.

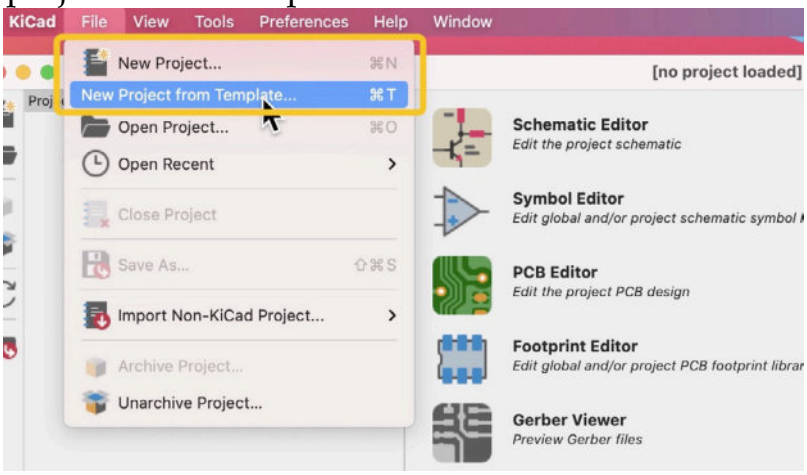


Figure 2.5.1: KiCad offers two ways to start a new project.

When you start a new project from a template, you can take advantage of work that you (or the original author of the template) have done in the past. Project templates offer an excellent way to speed up the initial time-consuming steps for projects that share a common base. For example, if you create Arduino shields, you can set up an Arduino shield base template and use it to create new Arduino shield projects. You can learn more about project templates in a dedicated chapter in the Recipes part.

In this chapter, you will create a new blank project. In the File menu, click on "New Project…". In the window that appears, set a name ("1", below), check the new folder box to have KiCad automatically create a new folder for your project ("2"), and click Save ("3").

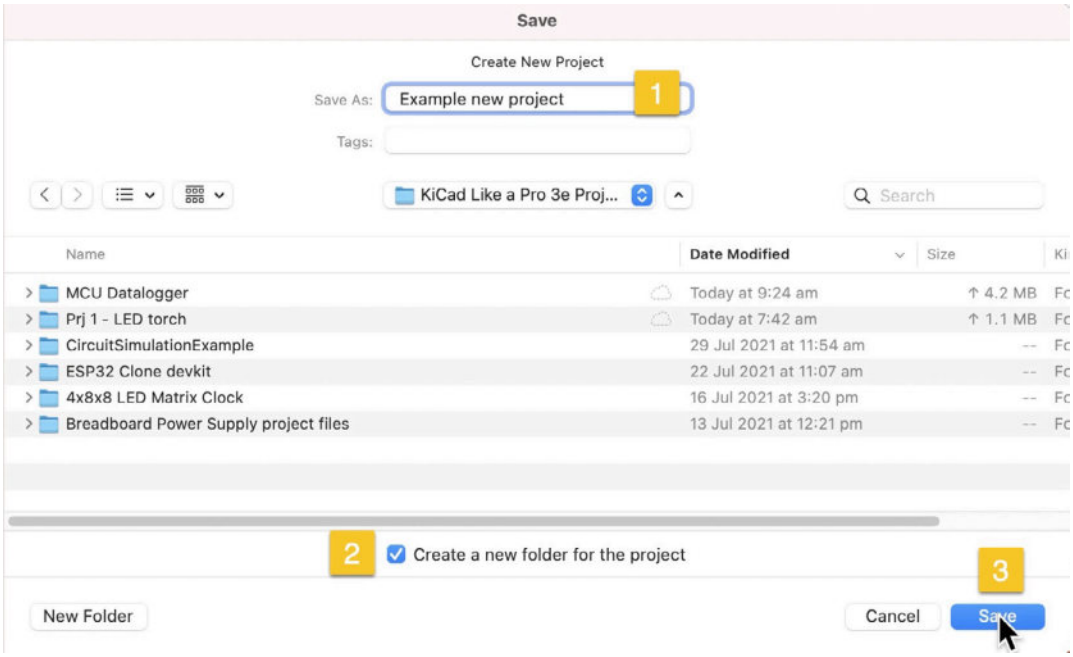


Figure 2.5.2: Set a name and directory for the new project.

KiCad will set up your new project. In the project folder, you will see three new files:

1. The main project file with extension ".kicad_pro."
2. The schematic design file with extension ".kicad_sch."
3. The layout design file with extension ".kicad_pcb."

The KiCad project window will show the project as a hierarchy tree. At the top of the hierarchy is the project file (".kicad_pro"), and inside of that are the schematic and layout files.

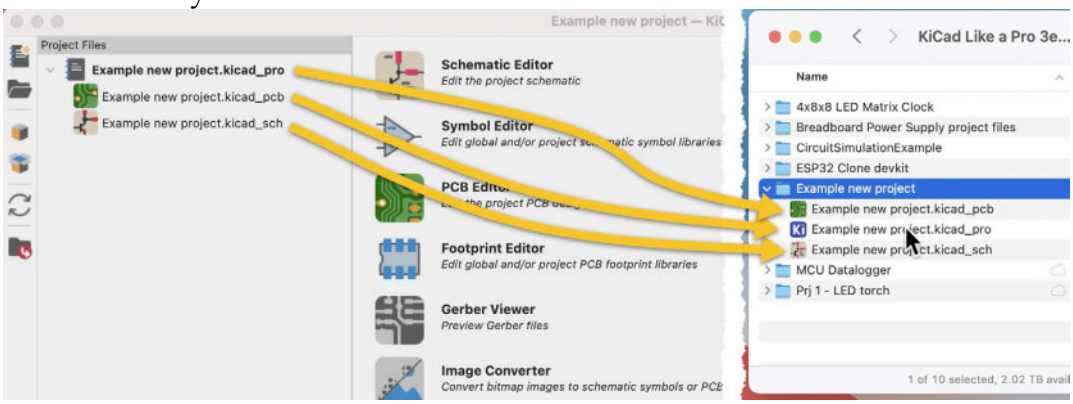


Figure 2.5.3: The new project is ready.

At this point, your new project is ready. You can open the schematic editor and begin work on the schematic. This is where you will begin work in the next part of this book, in which you will work on your first KiCad project. In the next chapter, you will learn how to create a new KiCad project from a template.

6. Create a new project from a template

In this chapter, you will learn how to create a new project from a template. KiCad comes with several project templates ready to use, but you can also create yours. You can read a dedicated chapter in the Recipes part if you are interested in creating custom project templates.

Click on “New Project from Template” in the File menu to create a new project from a template. The project templates window will appear (see below).

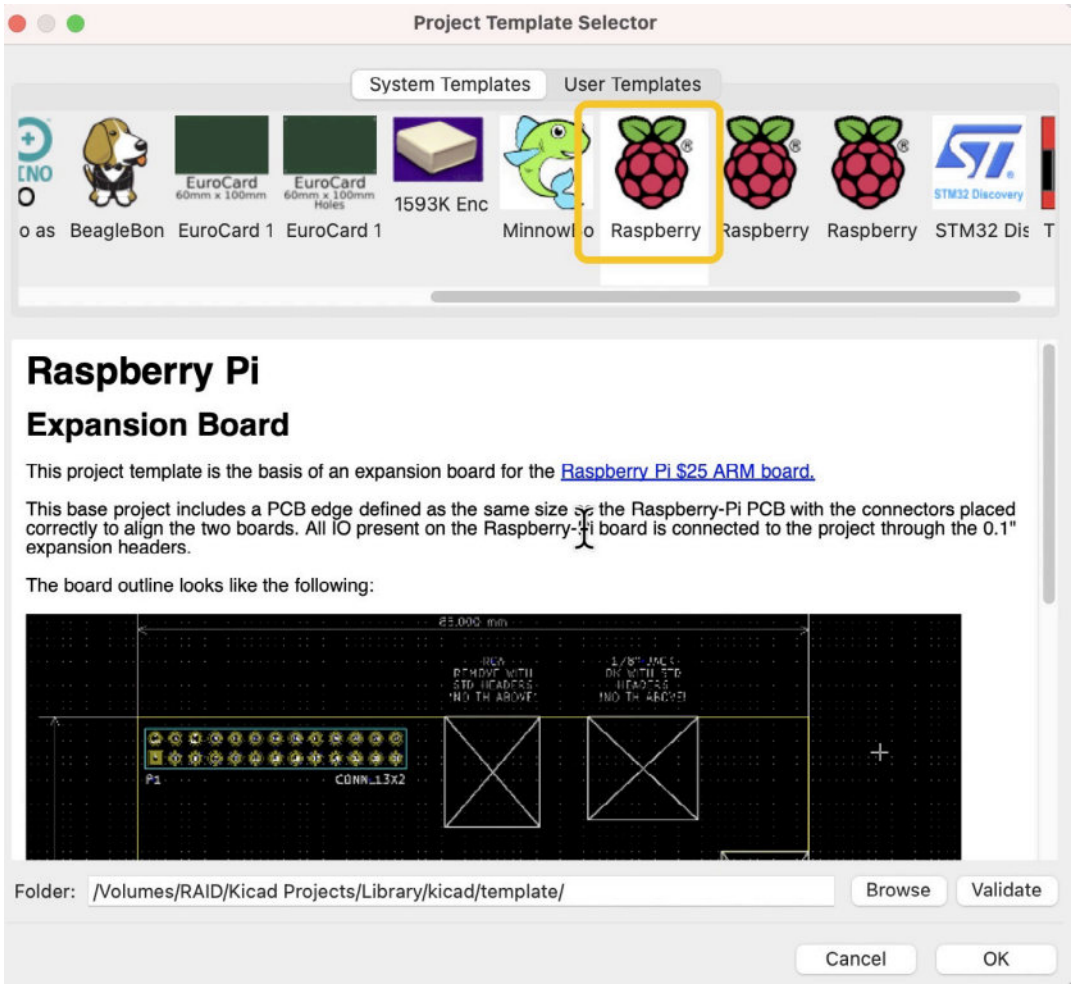


Figure 2.6.1: The project templates selector.

The selector window contains two tabs: System Templates and User Templates.

In a new KiCad installation, the User Templates tab will be empty until you create a new template and store it in the appropriate template directory (learn how to do this in the relevant chapter in the Recipes part).

The System Templates tab shows a collection of built-in templates. Click on a template icon to see information about it. For this example, I have selected one of the Raspberry Pi templates. The information box shows a description of the template. The description is composed of regular HTML so that you can include text, links, and images.

After selecting the template, you want to use, click OK. This will bring up the Save dialog box. This is identical to the dialog box that appears when you create a new blank project. Give the new project a name and location, and click Save.

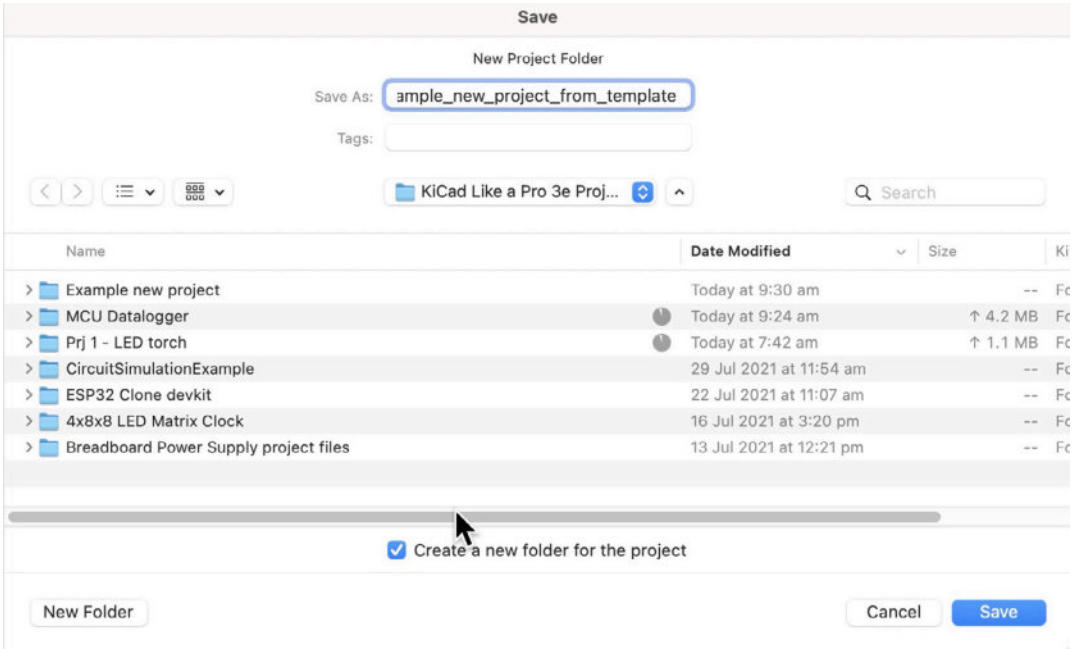


Figure 2.6.2: The name and location of the new project.

When KiCad finished creating the new project from the Raspberry Pi template, you will see several new files in the project folder (right, below) and the project hierarchy in the KiCad project window (left, below).

Figure 2.6.3: The new project created from a project template.

In the project folder (above, right), notice that several additional files also appear in addition to the project, schematic, and layout files. These additional files have been copied from the Raspberry Pi project template.

In the KiCad project window, click on the Schematic Editor button to open Eeschema. In a new blank project, the schematic editor is empty. But this is a new project from a template; the schematic and layout editors are already populated with seeding content.

Below is the schematic editor showing a header and mounting holes for a Raspberry Pi project:

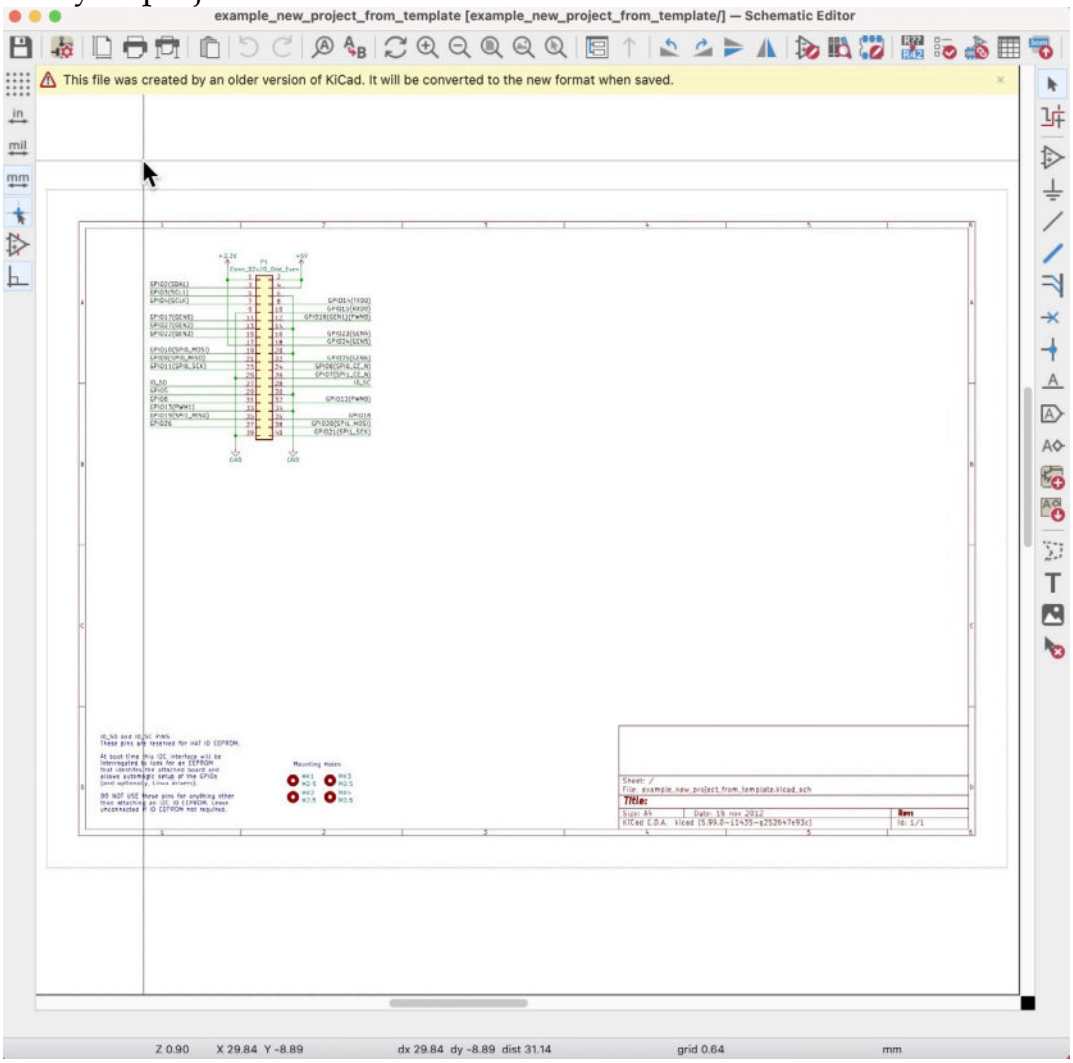


Figure 2.6.4: The new project schematic is already populated with content from the template.

Similarly, the layout editor is already populated with content from the template:

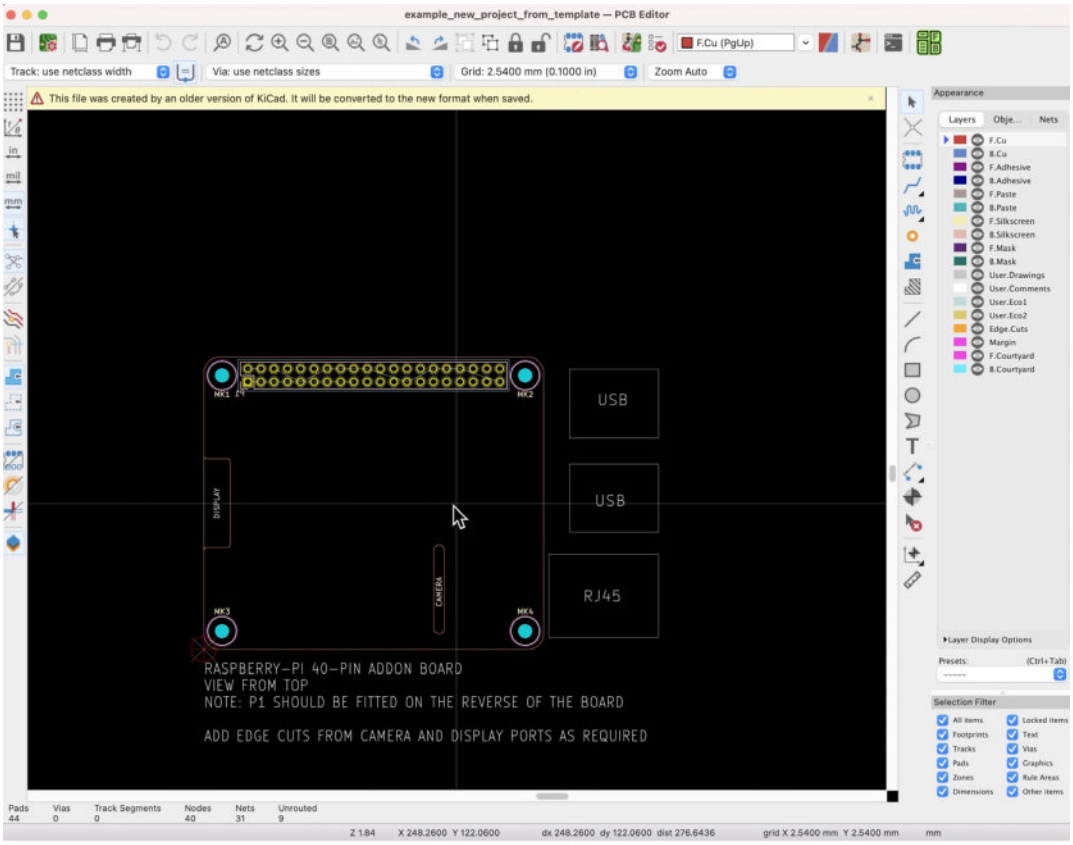


Figure 2.6.5: The new project layout is already populated with content from the template.

As you can see, much of the work has already been done. In the layout editor, the design of the board outline requires exact measurements, which are time-consuming. The placement of the mounting holes and connectors, likewise, must be exact and, as a result, very time-consuming. All this is work that you can avoid when you create a new project from a template.

Creating a new project from a template is an example of a productivity-boosting tool that KiCad provides. You will learn about many more in this book.

7. KiCad 6 on Mac OS, Linux, Windows

KiCad has supported multiple operating systems from its early days. When I started using KiCad in version four, I used it on Windows, Mac OS, and Linux (Ubuntu). However, there were differences between those platforms, both in terms of reliability (I found Windows, generally, worked better) and how the user interface looked and behaved.

I have been using KiCad 6 almost daily for almost nine months now, and I feel that KiCad works seamlessly on the three operating systems I have used (Mac OS, Windows 10, and Linux).

I spent a lot of time comparing the two. My testing consisted of a single project that I opened and edited across the three operating systems. I used KiCad's ”archive project” function, which you can find under "File" in the KiCad project window. Opening and working on a project that I previously edited on a different operating system were trouble-free.

Below, you can see the same project's main KiCad project window in Mac OS, Windows, and Kubuntu. They look identical while following the UI conventions of their host operating system.

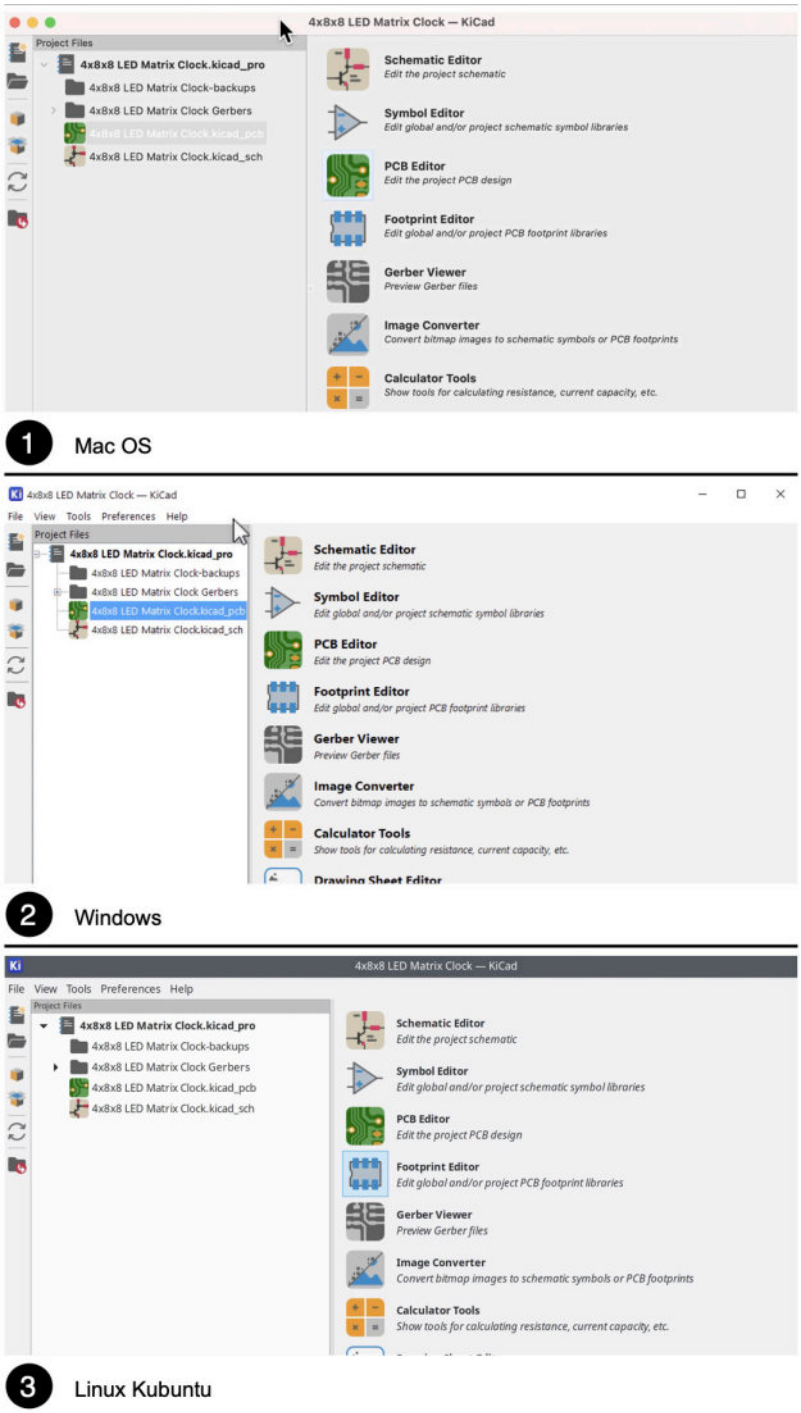


Figure 2.7.1: KiCad project window on three OSs.

There were no surprises in terms of KiCad's main applications, Eeschema and Pcbnew, and how those work. Shortcuts, mouse conventions, menus, buttons, colors; all work as expected in a truly cross-platform compatible application suite.

Below is an example of Eeschema in the three operating systems:

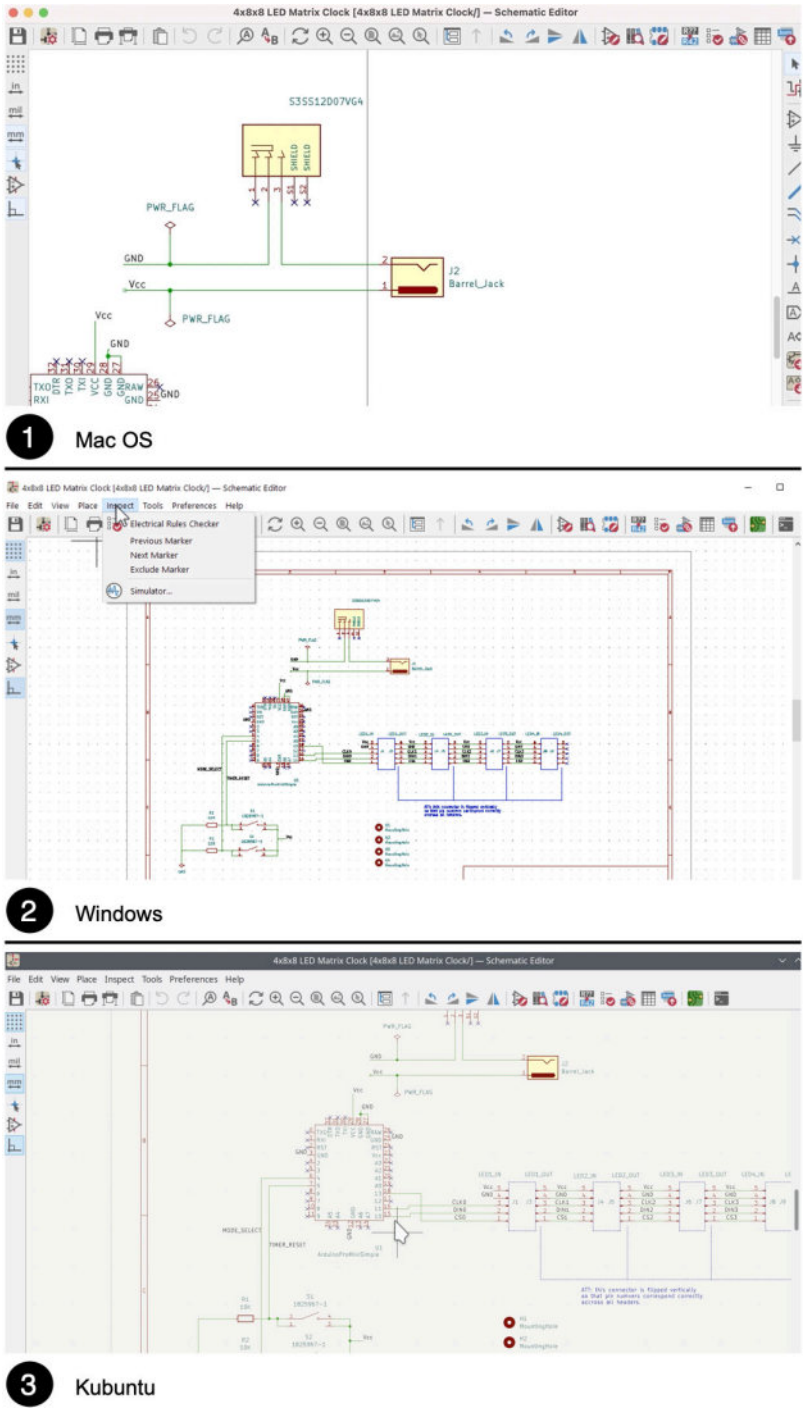


Figure 2.7.2: Eeschema in the three OSs.

And here is Pcbnew:

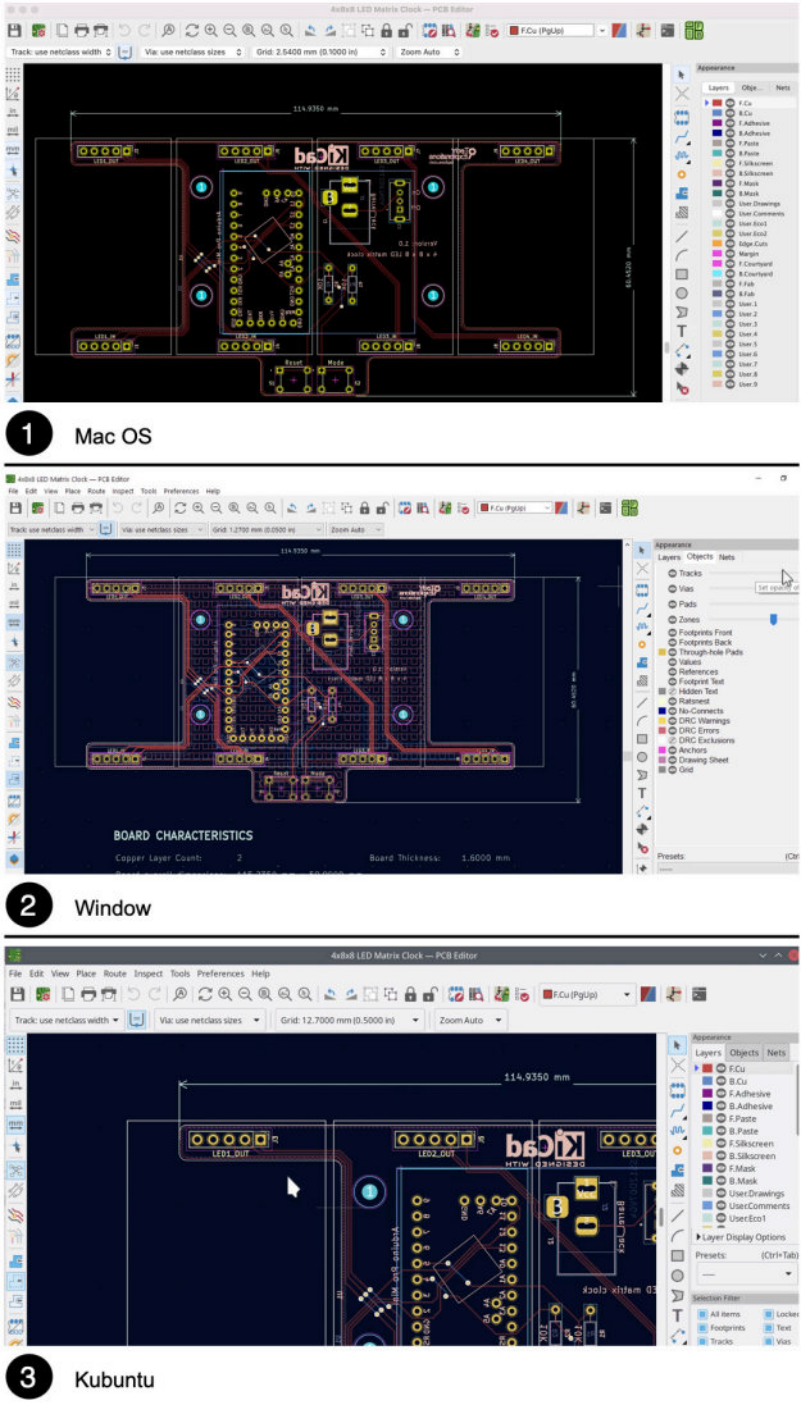


Figure 2.7.3: Pcbnew in the three OSs.

The same uniformity appears when testing other KiCad applications, such as the 3D viewer, the various preferences windows, and the interactive router. Even secondary widgets and features work well across the supported platforms.

Figure 2.7.4: Schematic Setup in Mac OS and Windows.

The quality of the implementation of Kicad in the three operating systems I have tested is excellent. The implication for solo users and teams is that you can use KiCad 6 with high confidence that you can edit the same projects across platforms. If you are in a team, your team members will work using their preferred operating system.

8. Differences between KiCad 6 and 5

KiCad 6 is a significant upgrade over KiCad 5. If you are new to KiCad, and KiCad 6 is the first KiCad you have ever used, you can safely ignore this chapter. Go ahead to Part 3, and begin work on your first KiCad project.

However, if you have used a previous version of KiCad and created one or more projects, you take some time to read a blog post that I wrote in early 2021. In that blog post, I go into detail to highlight and explain the differences between KiCad 6 and KiCad 5.

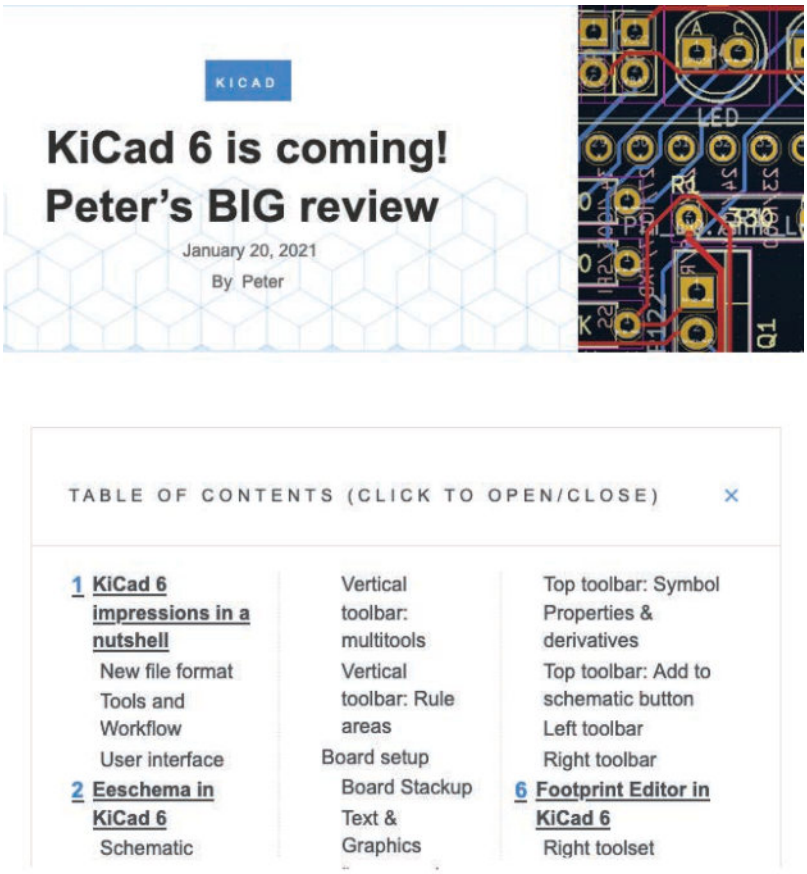


Figure 2.8.1: Peter's Big KiCad 6 review.

Here, I will list my top-three most significant changes in KiCad 6:

1. KiCad 6 has a new file format. The transition into this format, based on the S-Expressions standard, started in KiCad 5. With KiCad 6, the transition is complete.
2. The user interface is refreshed and modernized. While in KiCad 6, the user interface is still recognizable from the earlier versions, it follows

modern conventions on how the mouse and keyboard work. If you are coming from an earlier version of KiCad, you will use your existing KiCad knowledge. Icons have been redesigned. The menus and toolbars are better placed and organized. There is a single Preferences window.

3. The schematic editing paradigm is updated. Now, when you click on an element in the schematic editor, the element is selected. This was not the case in KiCad 5 and prior, causing much confusion and frustration.

Get the full details of what's new in KiCad 6 in my comprehensive blog post.

Part 3: Project - A hands-on tour of KiCad - Schematic Design

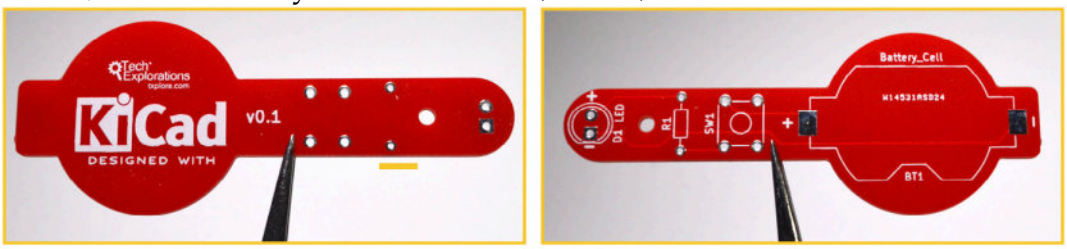
1. Introduction to schematic design and objective of this section

In Part 3 of the book (which you are reading now), you will learn about the basics of KiCad by working and completing a simple PCB project. In Part 3, the focus is on the schematic design, while in Part 4, the focus shifts to the layout design and the manufacturing. By the end of this project, you will have experienced the PCB design process using KiCad from start to finish.

While this first project is relatively simple, it will teach you the most important KiCad features and tools. You will develop skills that you will use in every future project regardless of its complexity.

As you work your way through this project, remember that you may need to reference the chapters in Part 13, Recipes, if you want to learn more details about specific features. To keep the size of the project concise, I have moved detailed descriptions of various features and tools to the end of the book.

The practical objective of this project is to design and manufacture a simple LED torch, like the one you see in 3.1.1 (below):



Final PCB, back Final PCB, front

Figure 3.1.1: The manufactured project deliverable.

Most of the work will be in Eeschema (the schematic design editor) and Pcbnew (the layout design editor). At the end of this Part 3 of the book, the schematic design will look like this (Figure 3.1.2):

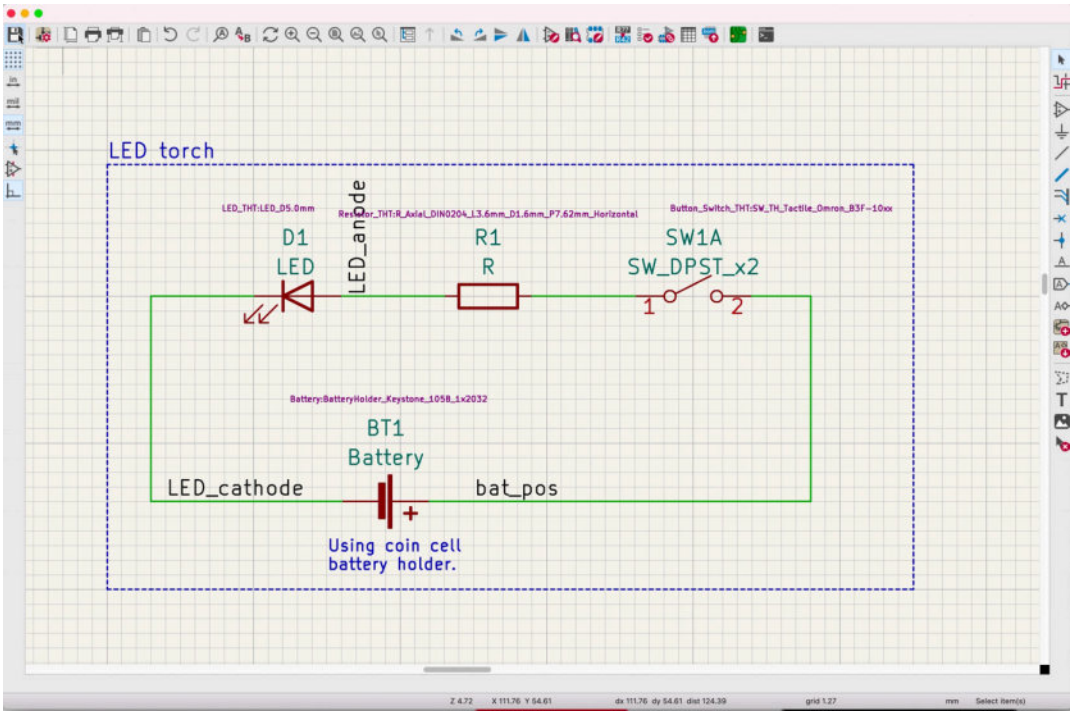


Figure 3.1.2: The final project schematic design.

The final layout will look like this (Figure 3.1.3):

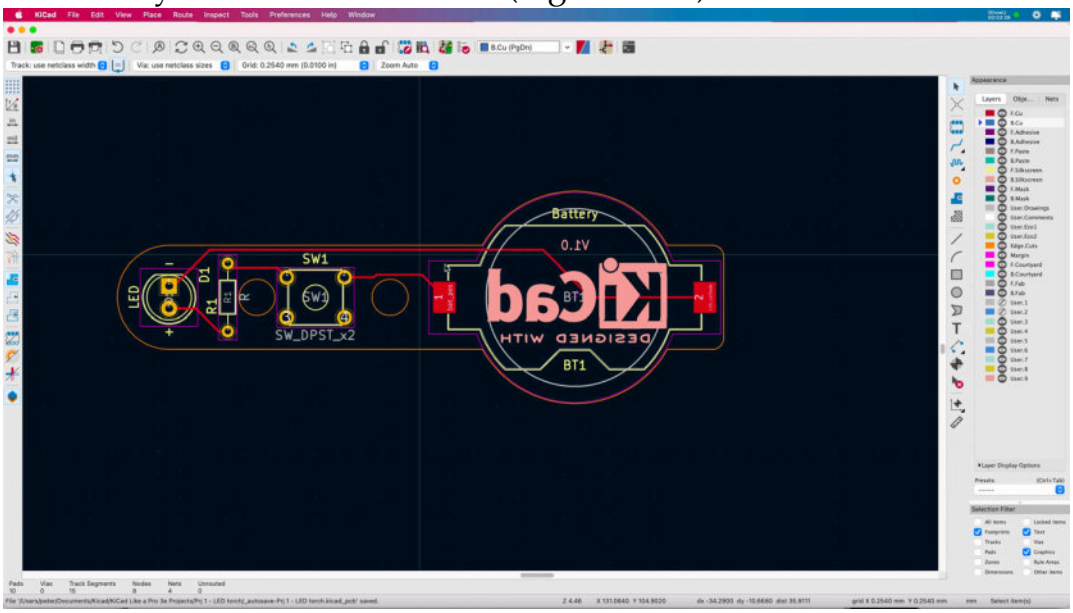


Figure 3.1.3: The final project layout design.

To guide the design of the PCB, I will be using the PCB design workflow that I outlined earlier in this book. I am also providing a summary in the next chapter.

The schematic (see Figure 3.1.2) contains only a few standard component symbols: an LED, a resistor, a button switch, and a battery holder. All these symbols are available in the KiCad libraries, so you will not need to get them

from external sources. Electrically, the circuit contains a single loop. When you press the button, the circuit closes, and the LED turns on.

Despite this being a simple project, you will learn how to find and add symbols to the editor, associate them with layout footprints, annotate them, wire them, create named nets, run the Electrical Rules Checker, and decorate the schematic with text and graphics.

In Part 4, you will learn how to import the schematic in Pcbnew and design the physical layout, complete with beautifully rounded corners, mounting holes, silkscreen graphics, and, of course, pass the design rules check before sending it to manufacturing.

2. Design workflows summary

This chapter will give an overview of the model design workflow that I use to guide me through the PCB design process.

You can see the model in Figure 3.2.1 below. For a comprehensive discussion, please refer to Part 6 of this book, specifically Chapter One (the KiCad Schematic Design Workflow) and Chapter Two (the KiCad Layout Design Workflow).

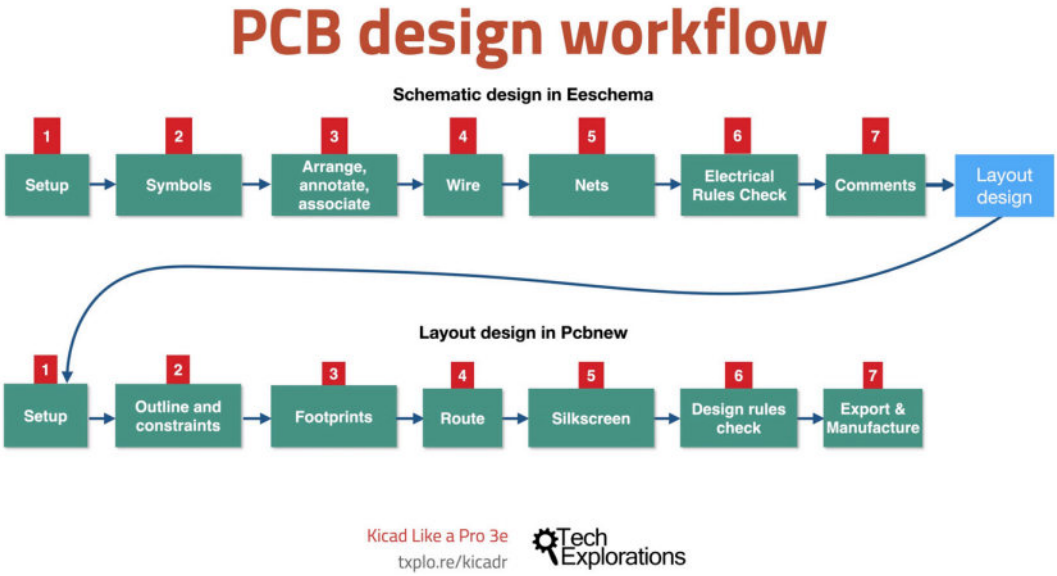


Figure 3.2.1: The KiCad model PCB design workflow.

The model consists of two workflows: The Schematic Design Workflow and the Layout Design Workflow. We use Eeschema to create the schematic design and Pcbnew to create the layout design.

Throughout this first project, I will be referencing the steps you see in 3.2.1; you may want to bookmark this page to jump back here when you need to quickly.

As you can see, work begins in Eeschema. The schematic design workflow consists of 7 distinct steps, which you can complete linearly, one after the other. In real life, it is more common than not to iterate through these steps as needed. For example, you may need to change the wirings in step 4 after finding electrical errors in step 6.

When you complete the schematic design, you will continue with the layout design workflow using Pcbnew. Again, the layout design workflow

consists of another seven distinct steps, which you can complete linearly. As with the schematic design workflow, real-life progression is typically iterative. It is common for a designer working in the layout design to jump to a much earlier step in the schematic design editor to fix a design bug, add new components, rewire, at make a change that affects the layout.

In this first book project, I will keep the workflows as linear as possible to reduce the overall complexity and improve your learning outcomes.

In Figure 3.2.2 (below), you can see a detailed depiction of the schematic design workflow. This depiction provides details about some of the tasks that we complete in each step.

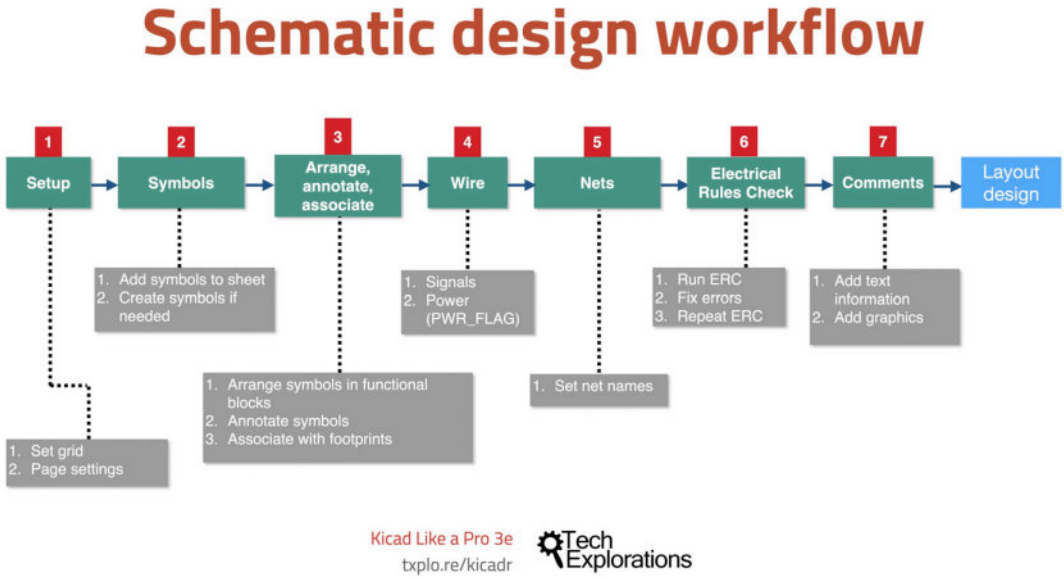


Figure 3.2.2: A detailed depiction of the schematic design workflow.

For example, you can see that in step four, you will draw the signal and power wires, while in step 5, you will set the names of the various nets. I will be helping you through each of the tasks in these seven steps through the project.

Similarly, in Figure 3.2.3 (below) you can see a detailed depiction of the layout workflow:

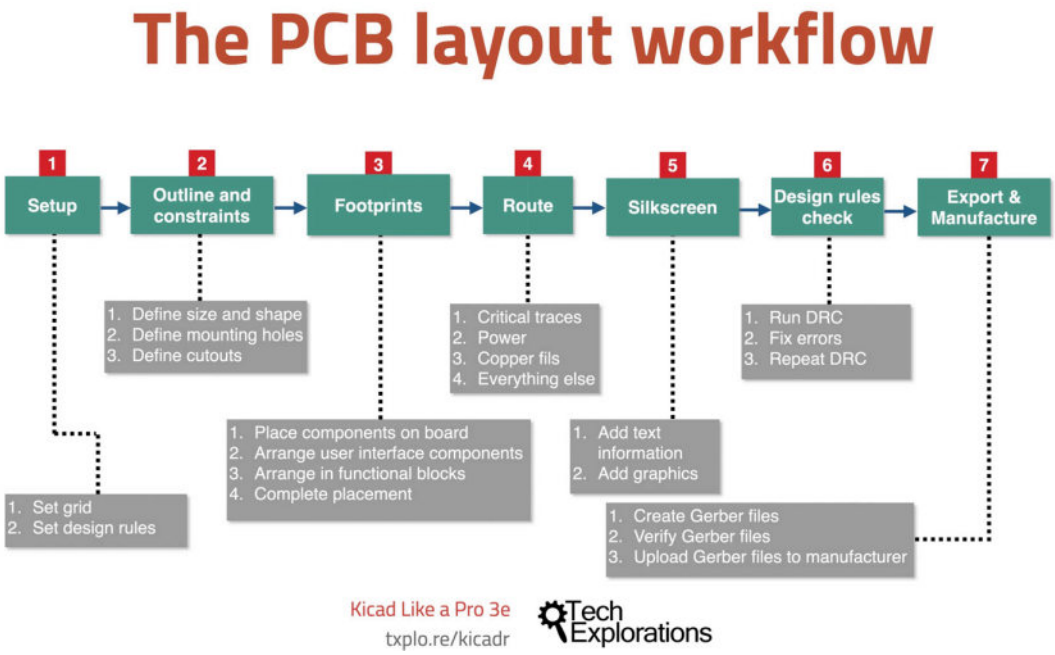


Figure 3.2.3: A detailed depiction of the layout design workflow.

We will use the workflow in Figure 3.2.3 in Part 4 of this book which covers the layout workflow for this first project.

3. The finished KiCad project and directory

Before I start this project, I want to take a few minutes to show you the completed KiCad project. First, let's look at the project contents as they appear in the main KiCad project window (Figure 3.3.1).

Figure 3.3.1: The project contents in the KiCad main window.

The main project file has the extension ".kicad_pro", and you can see it at the top of the hierarchy tree in the project files pane of Figure 3.3.1. Within the project structure, the two most important files are the schematic (".kicad_sch") and layout (".kicad_pcb").

The project hierarchy may contain secondary files and directories, such as the Gerber and backups directory. I will show you how to set up the automated backup feature in the next chapter.

In Figure 3.3.2 (below), you can see how a typical KiCad project looks on the file system.

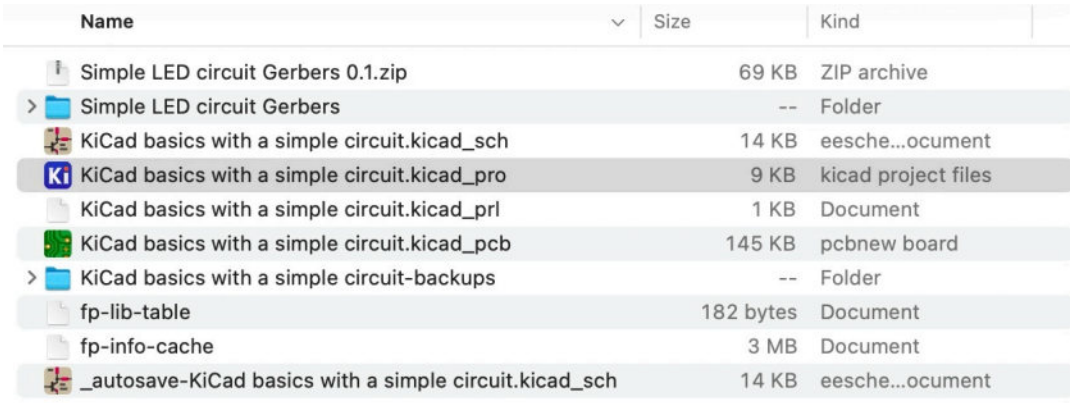


Figure 3.3.2: The Kicad project on the file system.

KiCad projects on the file system have a flat hierarchy. You can see the ".kicad_pro" file at the same level as the ".kicad_sch" and ".kicad_pcb" files. In Figure 3.3.2, you can also see several other files:

- fp-lib-table : the global footprint library table; it contains a list of the libraries always available to a project, regardless of which project is loaded.
- fp-info-cache: a file that speeds up access to the footprint repository information.
- _autosave-…kicad_sch: this file helps restore the last saved state of the schematic file in case KiCad crashes.

The files in the table above are examples of files that KiCad generates and manages. Under normal circumstances, you will not need to do anything with these files and can even choose not to include them in an archive of your projects.

If you compare the contents of Figures 3.3.1 and 3.3.2 you will notice that there is a one-to-one correspondence between the main KiCad files. The KiCad project window does not show any of the secondary files (including the ZIP archive of the Gerber directory).

4. Start Kicad and create a new project

In this chapter I will show you how to start work on a new project in KiCad.

Open KiCad. The main project window will look like this (Figure 3.4.1):

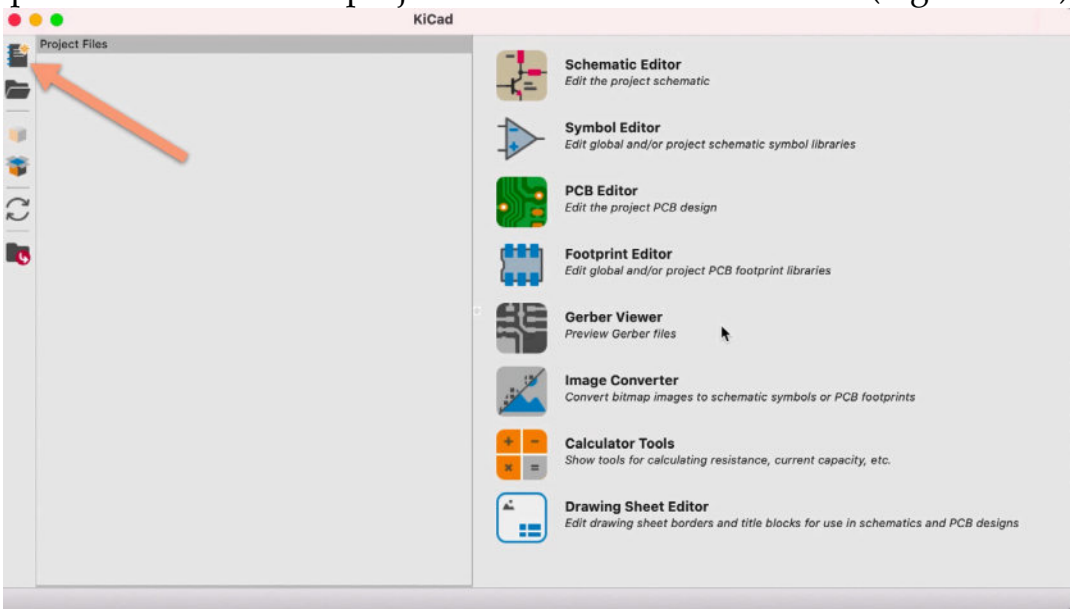


Figure 3.4.1: The main KiCad project window, about to create a new project.

The arrow points to the new project button. Don't click on it just yet!

You will need a location on your computer's file system to store the project. I have a central location for all projects in this book, which you can see below:

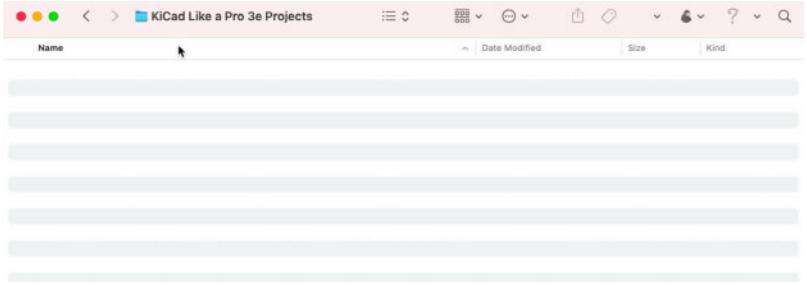


Figure 3.4.2: My central project directory.

I have named this directory "KiCad Like a Pro 3e Projects", and it is empty at the moment. I have placed this directory on a file system that is backed up to the cloud automatically. Using an automated cloud backup service is an additional layer of safety for my important projects.

Go ahead and click on the new project button (see arrow in Figure 3.4.1). KiCad will ask you for a name and location for your new project. In the

dialog that appears (Figure 3.4.3), provide a name (1), location (2), enable the new folder creation option (3) and click Save (4).

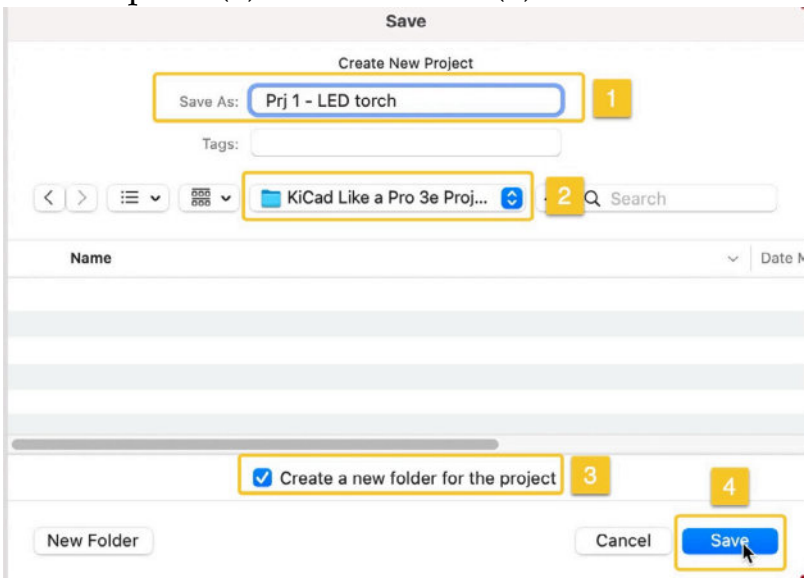


Figure 3.4.3: The new KiCad project name and location.

KiCad will create a new folder to contain the new project files (Figure 3.4.4):

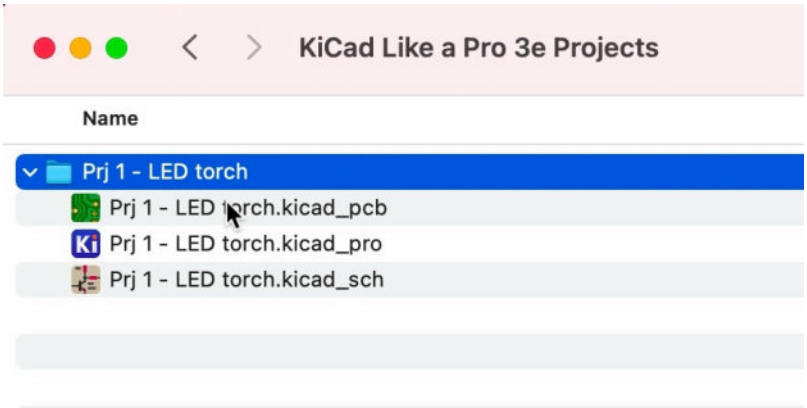


Figure 3.4.4: The new project folder and files.

The new project directory contains the three primary files: project, schematic and layout.

You now have a new KiCad project, and you are ready to continue work with step one of the schematic design workflow. Before you do this in the next chapter, take a few moments to familiarise yourself with the available apps through the main KiCad window. You can open those apps using the buttons in the right pane of the main KiCad window (Figure 3.4.1). Apart from the Schematic and Layout editors, you can see:

- The Symbol Editor.

- The Footprint Editor.
- The Gerber Viewer.
- The Image Converter.
- The Calculator tools.
- The Drawing Sheet Editor.

In the following projects in this book, you will learn how to use these tools, especially the symbol and footprint editors and the Gerber viewer. There are dedicated chapters about the symbol editor, footprint editor, and image converter. I have provided information on how to use these tools in all projects.

Now is a good time for you to take a few moments and "play" with these apps before you dive into this first project.

You should also be familiar with the contents of the Preferences menu in the main KiCad window. Under preferences, you will see three main items: Configure Paths, Manage Symbol Libraries, and Manage Footprint Libraries (Figure 3.4.5).

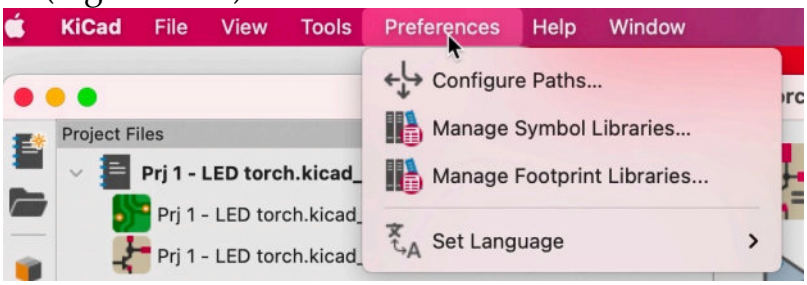


Figure 3.4.5: The contents of the Preferences menu item in the main KiCad app.

The symbol and footprint library manager windows have a similar structure. Each one allows you to set libraries that are accessible globally (i.e. used in all KiCad projects) or only by the currently open project. You can add, remove and edit library entries. You can learn how to use these managers in dedicated chapters (symbol library manager and footprint library manager).

Also under the Preferences menu item is the Configure Paths window. It looks like this (Figure 3.4.6):

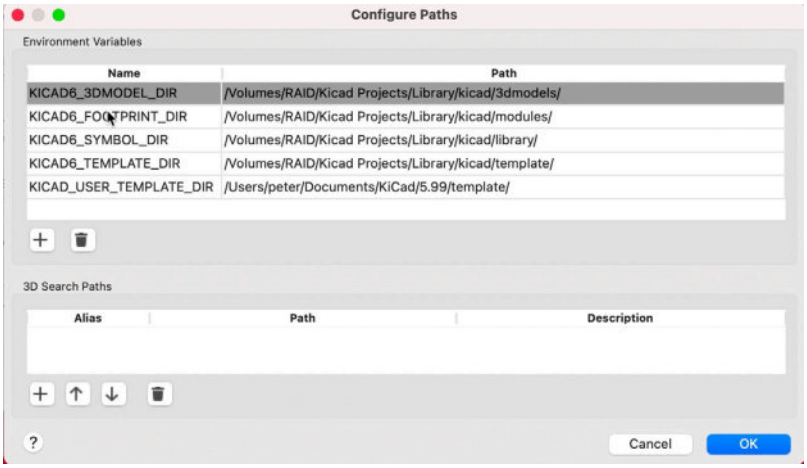


Figure 3.4.6: The Configure Paths window.

The settings in this window apply to all KiCad projects. As you can see in the example above, I have edited the default paths to point to my external RAID drive. The default paths point to a location on the main computer drive. My computer's primary disk drive is a small but fast SSD, while the external RAID is large and redundant. Because KiCad's libraries (especially the 3D models) can occupy tens of gigabytes of space, I chose to store them on the external drive. I have not noticed any performance penalty since KiCad keeps library information cached within the project folder. Before you continue, decide if you would like to make any changes to your KiCad instance paths. I found that doing this mid-project can cause problems that are easy to avoid by planning.

Finally, you may want to set up the common KiCad preferences found in the Preferences window. Open the Preferences window and browse through the contents of the Common, Mouse and Touchpad, and Hotkeys tabs (Figure 3.4.7).

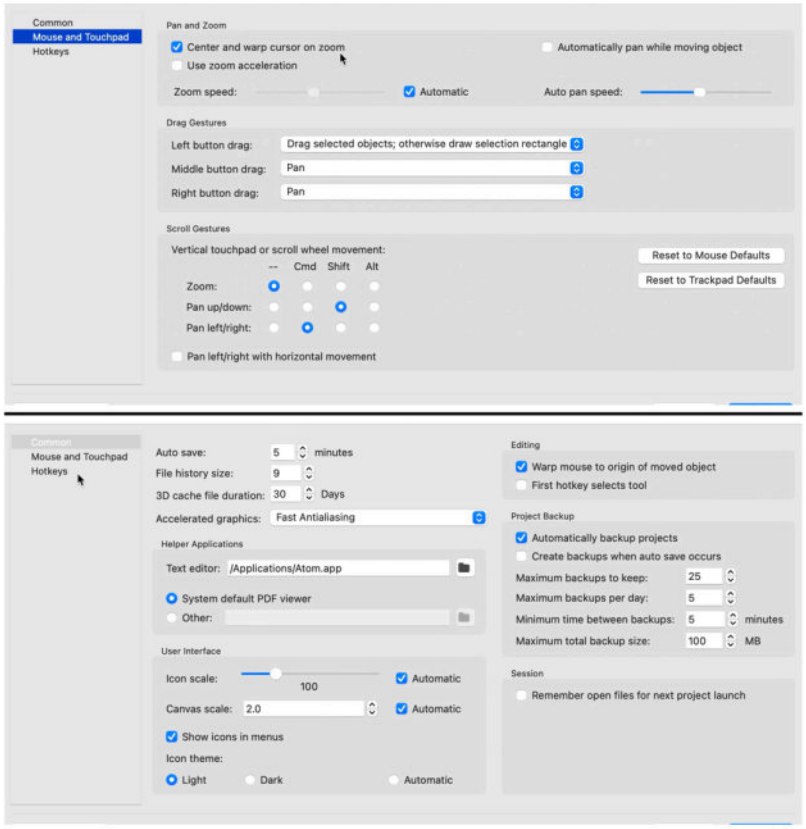


Figure 3.4.7: My settings in the KiCad Preferences window.

I find that the defaults work well. You can see the settings I am using throughout the projects in this book in Figure 3.4.7. You may want to experiment with the options in the Accelerated Graphics dropdown (in the Common tab) to find a good balance between graphics quality and performance that works best with your computer hardware.
Ready to continue? Go on to the next chapter.

5. 1 - Start Eeschema, setup Sheet

In this chapter, you will complete step one of the schematic design workflow that you learned about in the previous chapter.

I will continue where I left off in the previous chapter. At this point, I have created a new KiCad project, and the main KiCad window is open. Click on the schematic editor button at the top of the right pane in Figure 3.5.6 (below).

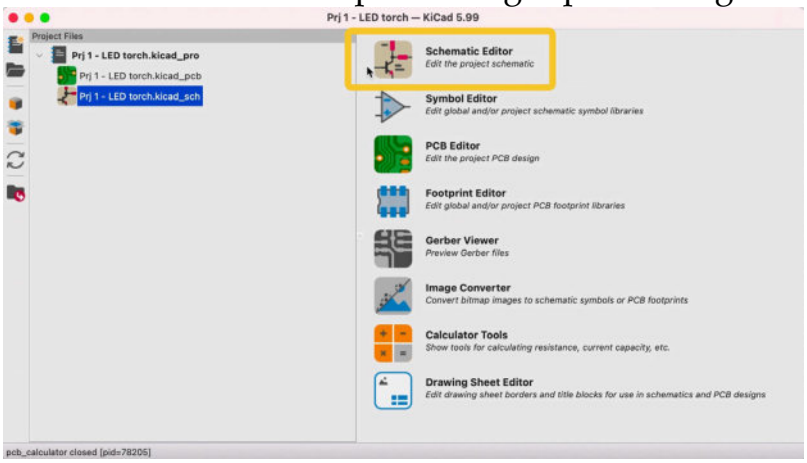


Figure 3.5.1: Start the schematic editor.

This will bring up the Eeschema window with a blank design editor (Figure 3.5.2):

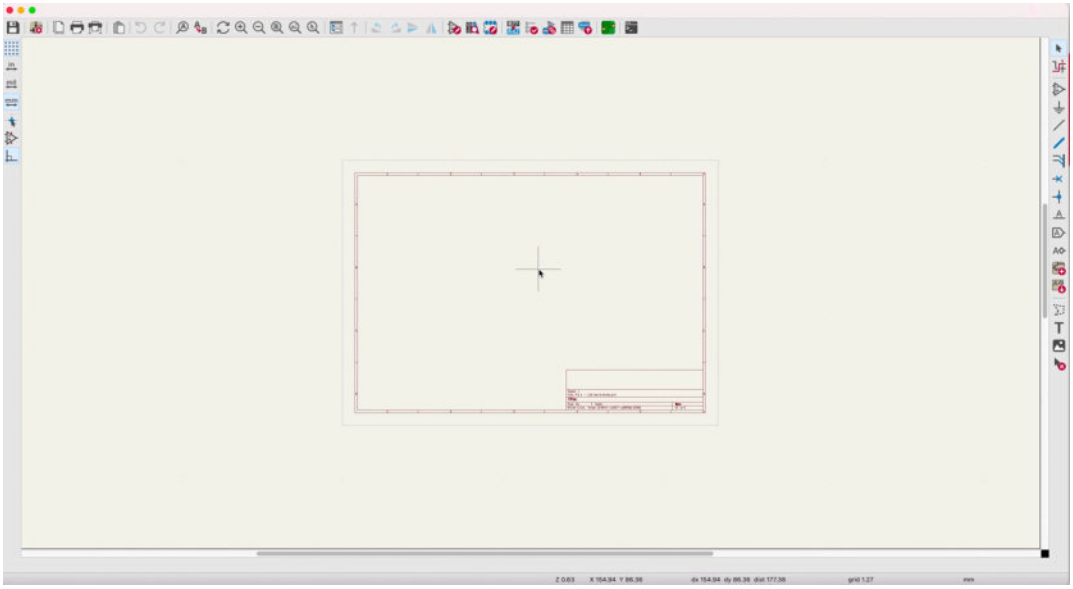


Figure 3.5.2: Eeschema.

It is worth taking a few minutes to look at some of the most important user interface tools and techniques you will use in all KiCad projects. Let's start with the mouse.

I strongly recommend that you use a mouse with two buttons and a scroll wheel. I use a Logitech MX Master 2S (Figure 3.5.3):

Figure 3.5.3: My Logitech mouse.

Apart from the regular functions assigned to the left and right buttons, I use the scroll wheel to zoom in and out. In addition, the scroll wheel of this mouse is a middle button that allows me to pan. Because zooming and panning are so helpful in any CAD application, it is important to have a mouse that provides easy access to those functions.

To zoom, move the mouse pointer to the location in the editor that you want to zoom in, and then turn the scroll wheel. The zoom will always centre on the crosshair pointer.

To pan, press and hold the scroll wheel button and move the mouse to pan. It is also possible to zoom while you pan.

You can expose the context menu by pressing the right mouse button. Just click anywhere in the editor sheet, and click. The exact contents of the context menu will depend on what it is that you clicked on. A wire will give you a different context menu to a symbol, and so on (Figure 3.5.4).

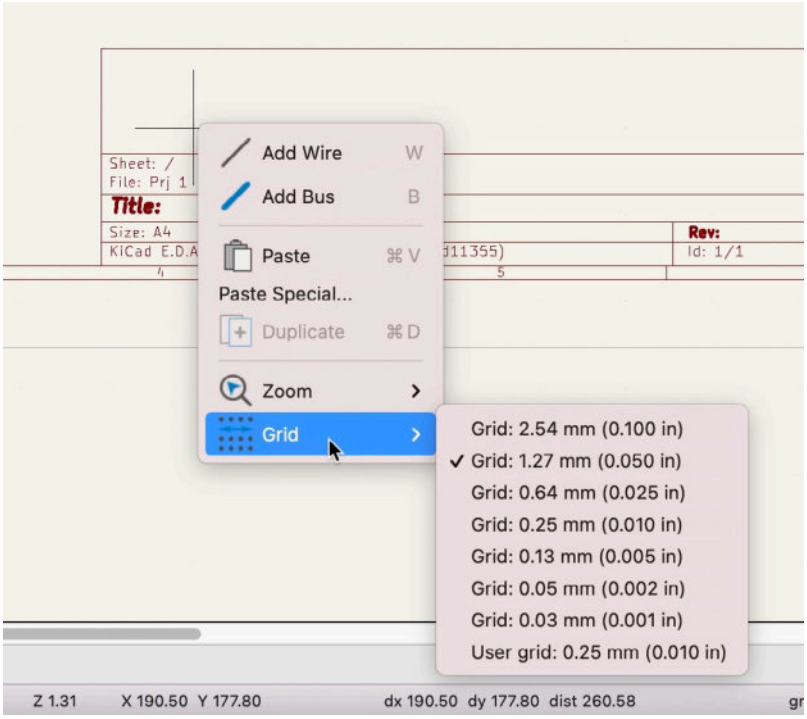


Figure 3.5.4: Example context menu.

In the example above, I have right-clicked in an empty part of the editor sheet. Notice that the context menu contains submenus such as Zoom and Grid.

The editor sheet has a coordinate system that starts at the top left corner. The horizontal axis is the X, and the vertical is Y. There is a grid that supports drawing using a visual guide for the alignment of the various objects. There is also a snap-to-grid option to ensure perfect alignment. I use this option to ensure that wires and pins are aligned. At the bottom of the Eeschema window is the status bar. On its right side, you will find information about the current coordinates of the cursor, the distance between the cursor and the location where your reset the ruler, the grid size, and the measurement unit used (Figure 3.5.5).

Figure 3.5.5: The status bar.

In the example above, my grid is set to 1.27 mm. You can change your grid and unit settings via the buttons of the left toolbar (Figure 3.5.6):

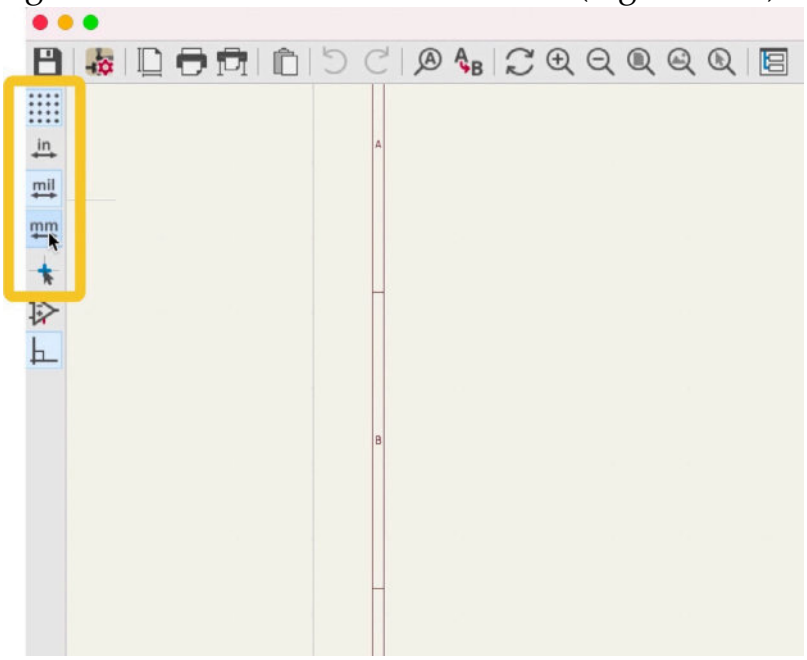


Figure 3.5.6: Grid and unit settings.

You can choose your preferred unit between millimeters, mils and inches, turn the grid lines on or off, and choose the type of cursor (regular or full crosshairs).

You can further control the appearance and operation of the grid via the Preferences window. Open the Preferences window, then click on Display Options under Schematic Editor (Figure 3.5.7):

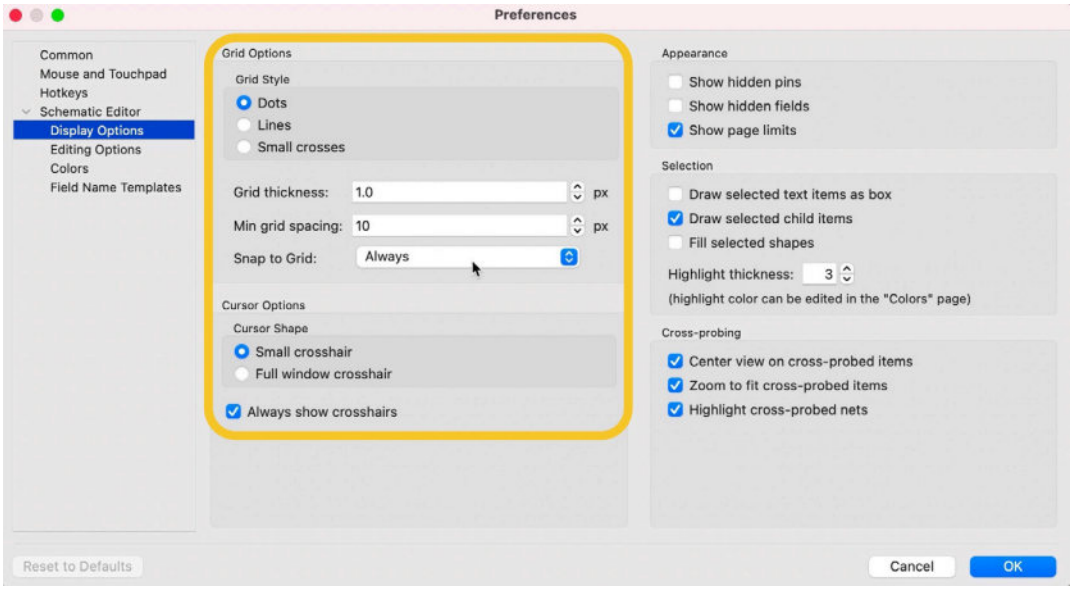


Figure 3.5.7: Grid options in Preferences.

You can choose between three grid styles, two cursors, control the grid line thickness, minimum spacing, and enable the snap to grid feature (default is on, and I suggest you leave it at that).

Next, still in Preferences, click on Editing Options. You can see the various options there with my settings below (Figure 3.5.8):

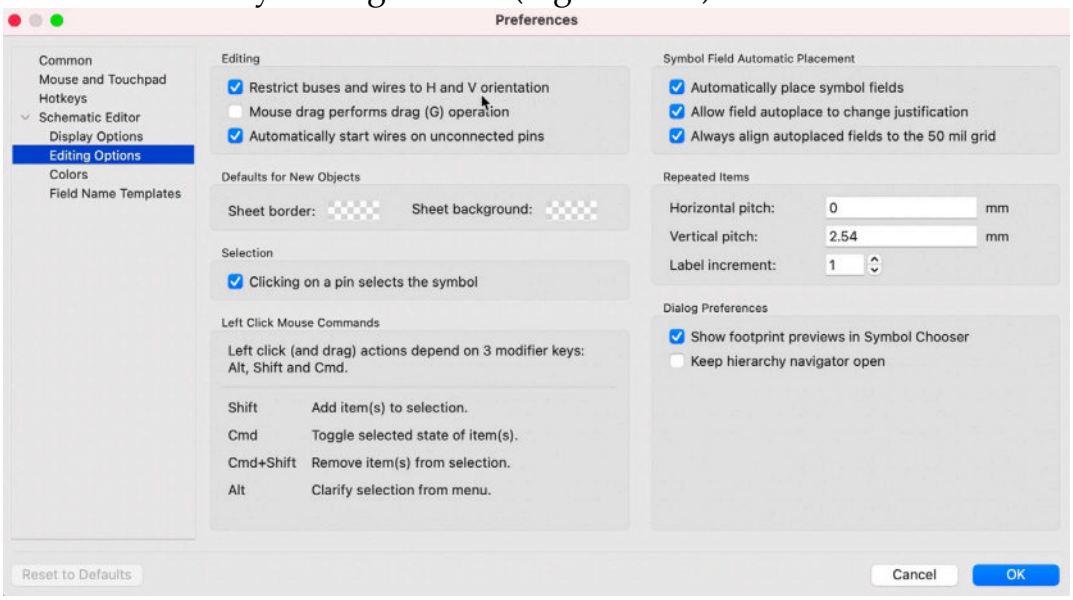


Figure 3.5.8: Editing options in Preferences.

In the Editing group, the first option sets the orientation restrictions for wires. If checked, you will only be able to draw horizontal and vertical wire segments. I believe that with this setting, you will be able to produce more visually pleasing schematics, so I suggest you leave it checked. You can see an example below; the wires in the left contain horizontal and vertical segments

only, and the one in the right has wire segments in non-90 degree angles (Figure 3.5.9).

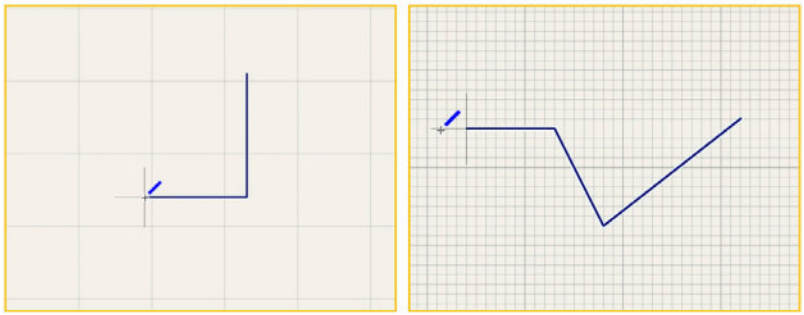


Figure 3.5.9: Wires.

Take a few moments to explore the options in the Schematic Editor tabs in the Preferences window before you continue (especially the ones in Display Options and Editing Options). You can learn more about these options in a dedicated chapter later in this book.

Remember that you can use the ESC key (type ESC twice) to exit any activated tool. To delete multiple items in a schematic, use your mouse to enclose those items in a box and highlight them, and then hit the Delete key (Figure 3.5.10):

Figure 3.5.10: Multi-select of several wire segments.

To configure the grid size and the quick-select settings, you use the Grid Settings window, accessible from the View menu (Figure 3.5.11).

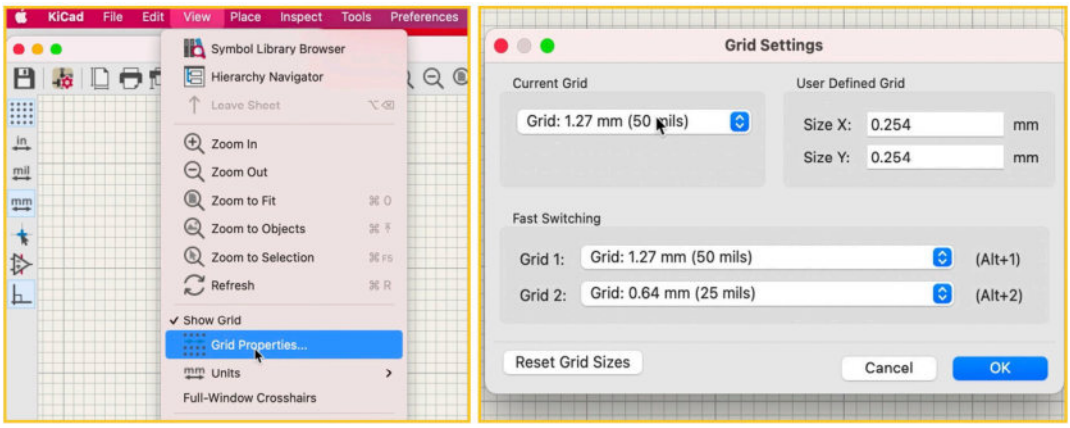


Figure 3.5.11: The Grid Settings window.

Generally, in busy schematics, you will want to use a small grid size. You can also set grid size keyboard shortcuts (Alt-1 and Alt-2) to specific grid sizes. You can see my settings in the figure above.

The last item in my to-do list in this first step of the schematic design workflow is to fill in the project information in the Page Settings window. Access the Page Settings window from the File menu (Figure 3.5.12):

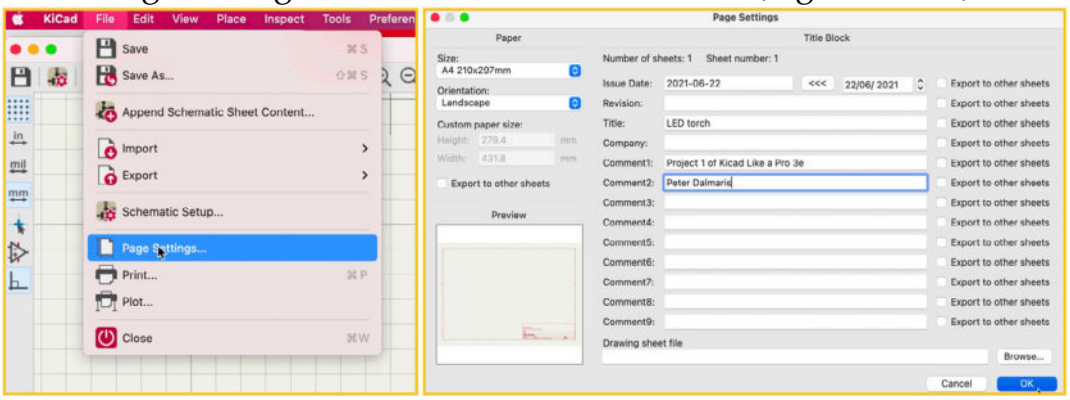


Figure 3.5.12: The Page Settings window.

The fields in the Pages Settings window can contain any text that you type in and will appear in the schematic sheet label (bottom right corner). You can provide any information you think is helpful to the reader of the schematic. Once completed, click OK, and notice how the information you entered appears in the schematic sheet (Figure 3.5.13):

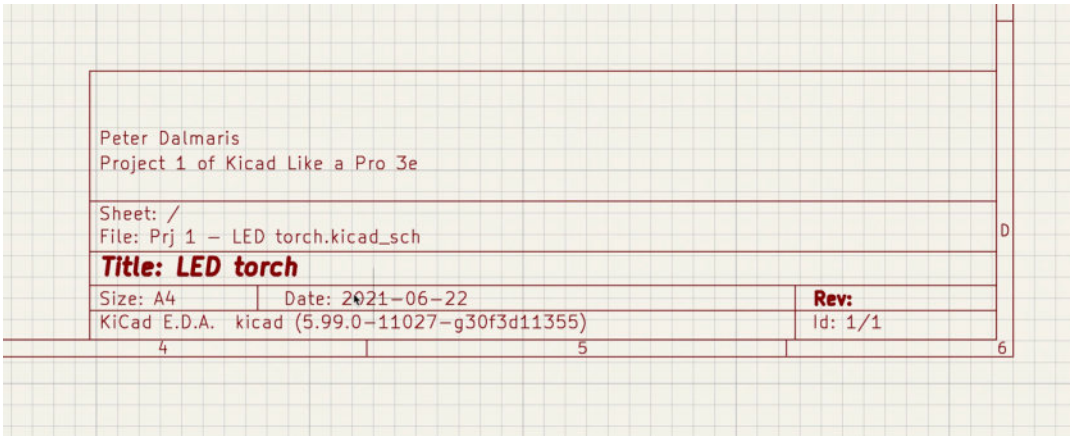


Figure 3.5.13: Project information in the design editor label.

Step one of the schematic design workflow is now complete. Let's continue with step two in the next chapter.

6. 2 - Add symbols

In this chapter, you will complete step one of the schematic design workflow that you learned about in the second chapter of this part of the book.

This chapter will show you how to find symbols using the symbol chooser and place them in the schematic design editor. To keep this first project simple, I will be using symbols that exist in KiCad's libraries.

To drop a symbol to the editor sheet, you need to first bring up the symbol chooser window. The symbol chooser contains a listing of all available symbol libraries and their contents, as well as a search engine. You can look for a symbol by searching for it (if you know its name), or browsing for it. To bring up the symbol chooser, use the "A" hotkey, or choose the "Add Symbol" option under the "Place" top menu, or click the symbol button in the right toolbar (Figure 3.6.1).

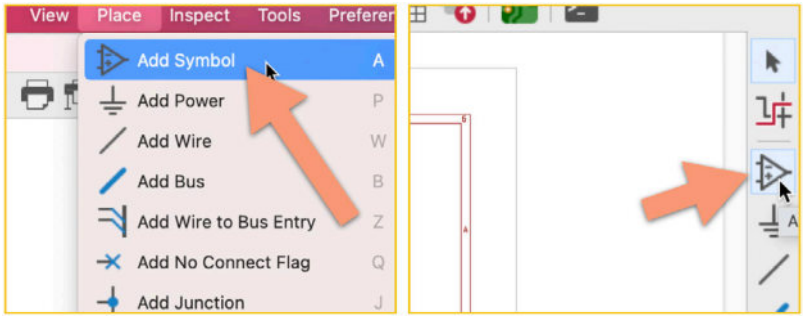


Figure 3.6.1: Getting to the Symbol Chooser.

The symbol chooser window will appear (Figure 3.6.2).

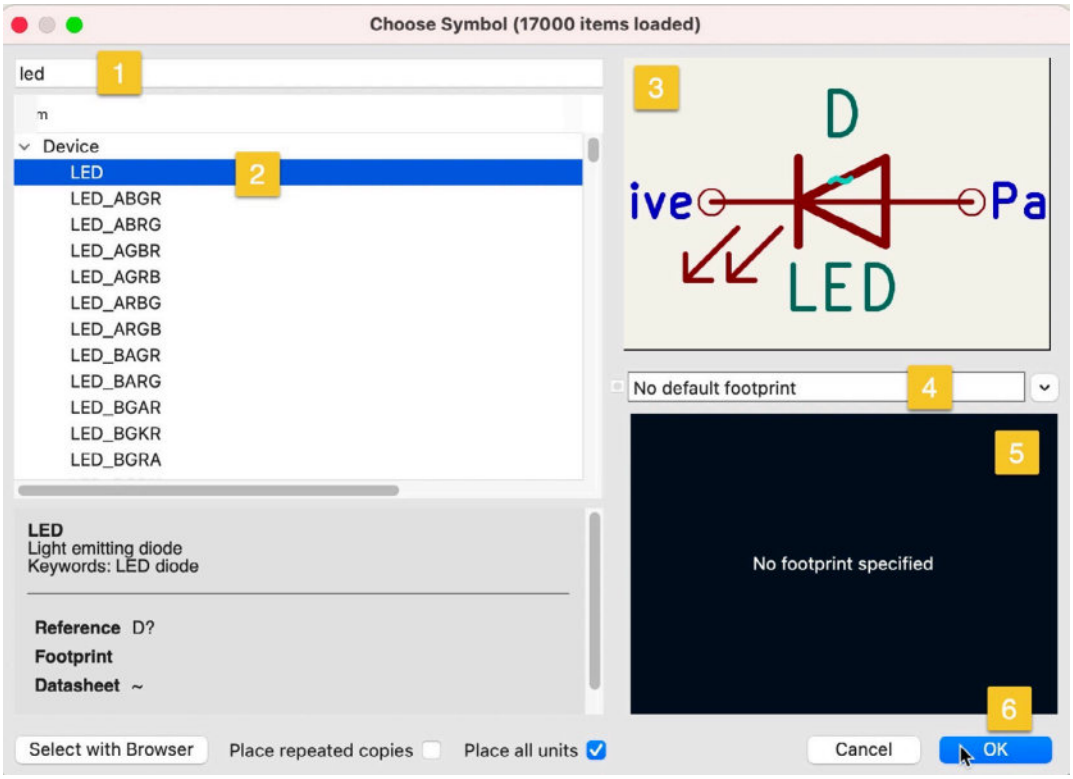


Figure 3.6.2: The Symbol Chooser window.

You can look for a symbol by typing a few letters of its name in the search box (1) or navigate the library and symbol list (2). In the example above, I am looking for the LED symbol, so I have typed “led” in the search box. The symbol chooser narrows down the contents of the listing pane (2) to symbols that contain “LED” in their name. I have clicked on the item in the first row. This prompts the symbol to appear in the symbol preview pane (3). Information about the selected symbol appears in the bottom left corner pane of the window. Some symbols also have associations with one or more footprints. You can choose one of the footprints by using the drop-down menu (4) and verifying that you have the correct association in the footprint preview pane (5). In my example, I don’t want to set an association at this time, so I leave the footprint dropdown unchanged and click OK (you may also double click on the device row (2) to select the symbol and dismiss the window).

After you dismiss the symbol chooser window, the symbol will be tied to the mouse cursor. You will be able to move the symbol around the editor. Find a good location for it, and then left-click to place it (Figure 3.6.3).

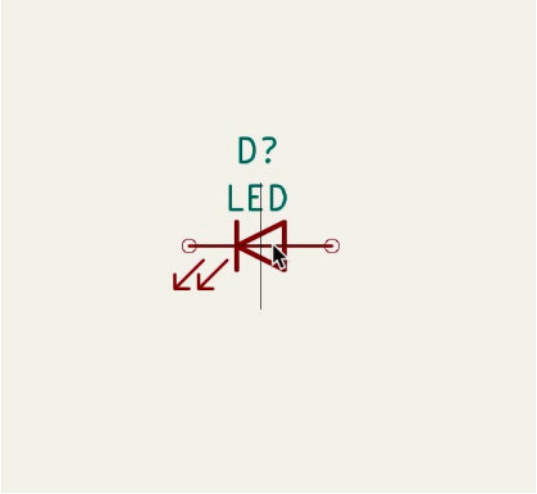


Figure 3.6.3: The new symbol in the editor sheet.

While a symbol is selected (you will see a bluish halo around it), you can use the "R" (counter-clockwise rotation) hotkey to change its orientation. You can select an unselected symbol by left-clicking on it. Remember that to unselect a selected symbol, simply left-click on any blank area. Also, to move a selected symbol, with the cursor over the selected symbol, press and hold the left mouse button and drag the symbol.

You can double-click on the symbol to bring up its properties window (Figure 3.6.4):

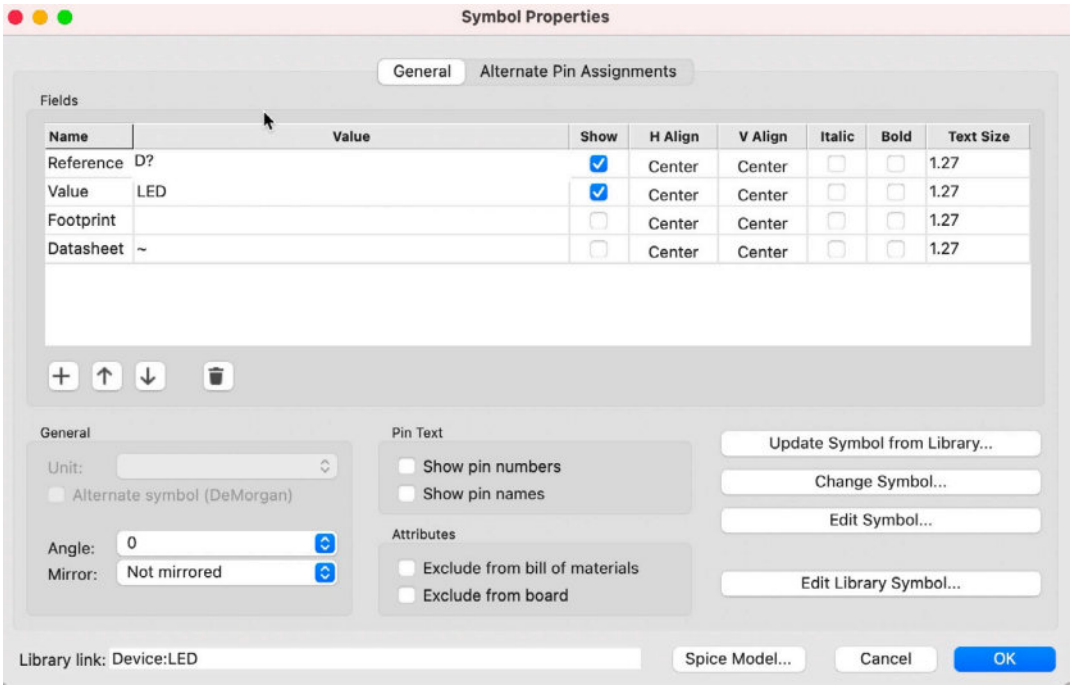


Figure 3.6.4: The symbol properties window.

We will be editing the properties of this symbol later to do things such as assign a footprint or add a URL to a data sheet. For now, take a moment to become familiar with it, and click "OK" to dismiss it.

Repeat the process I described above to add the remaining symbols:

- Resistor (search for "R").
- Switch (search for "SW_DPST_x2").
- Battery (search for "Battery_Cell").

Remember that you are working with symbols and that each symbol can be associated with a variety of "real life" footprints. For example, the battery symbol can be associated with footprints that belong to a 3.3V coin battery cell or a AA alkaline battery holder. We'll do the associations later.

Once you have completed the addition of the four symbols, your schematic diagram will look like this (Figure 3.6.5):

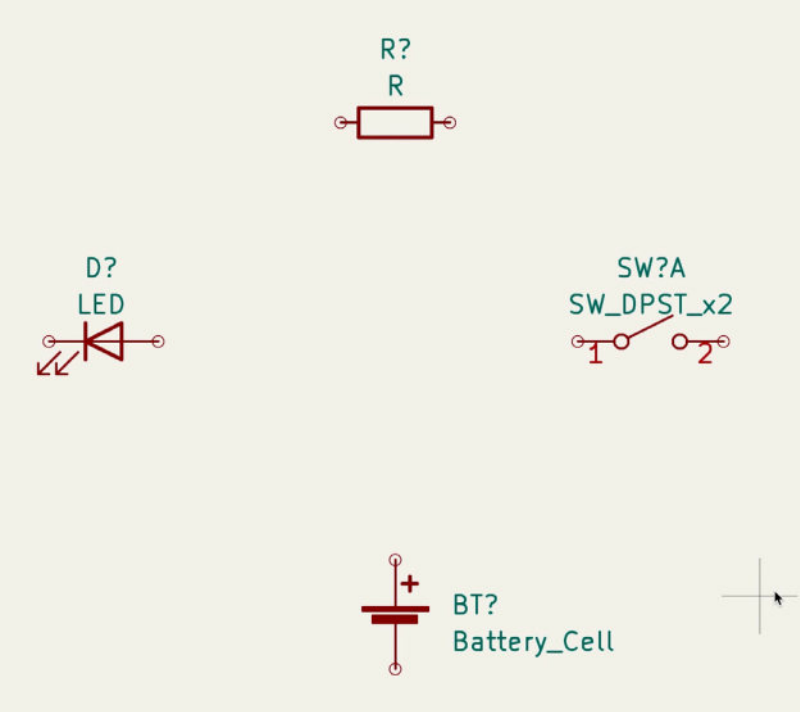


Figure 3.6.5: The circuit symbols in the editor.

If you have made a mistake, it is easy to fix. Say that you changed your mind and want to replace a symbol with another. The easiest way is to delete the incorrect symbol by selecting it and hitting the "delete" key. Then use the symbol chooser to find the replacement symbol. You can also use the interactive delete tool from the right toolbar. Once you enable this tool, you can delete any item in the editor by clicking on it (Figure 3.6.6).

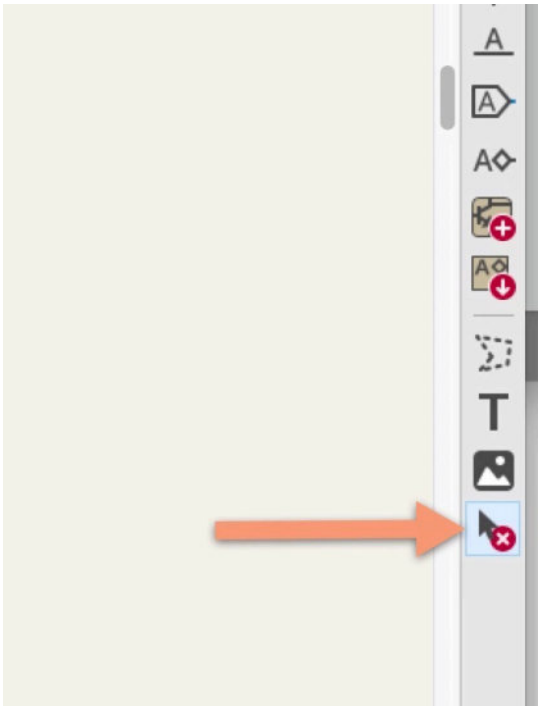


Figure 3.6.6: The interactive delete tool.

You can also change symbols in bulk. To learn more about this, please read the dedicated chapter on this topic.

The schematic editor now contains the four symbols I need for my simple circuit. I have not done the final placement and the wiring yet, as I'll do that in the next two steps of the workflow.

For now, save the document, and continue with the next chapter.

7. 3 - Arrange, annotate, associate

In this chapter, you will complete step three of the schematic design workflow, that you learned about in the second chapter of this part of the book.

This chapter will show you how to arrange the symbols you added to the editor in their final positions, annotate them with unique reference IDs, and associate them with the appropriate footprints.

Arrange

Start by moving the symbols. After finding them in the symbol chooser, I simply placed the symbols in random locations in the previous chapter. Now, I will put them in locations that make it easy to wire them to become part of a valid circuit.

You can move a symbol by selecting it with your mouse, then click on the selected symbol and hold, while you move the mouse. The grid size and snap-to-grid function are important here. You can use the fast-switch grid size to experiment with the placement options. My fast-switch hotkey (Alt-1 and Alt-2) allow me to switch between 2.54 mm and 1.27 mm grid sizes. As the grid size becomes smaller, you can place the symbols with finer positioning control. For the circuit we are working on, 2.54 mm for the grid size is sufficient, so I'll set it to that. You may also turn on the gridlines so that you can see the grid instead of only "feeling" it as a result of the snap-to-grid function.

Go ahead and place the symbols as in Figure 3.7.1 (below).

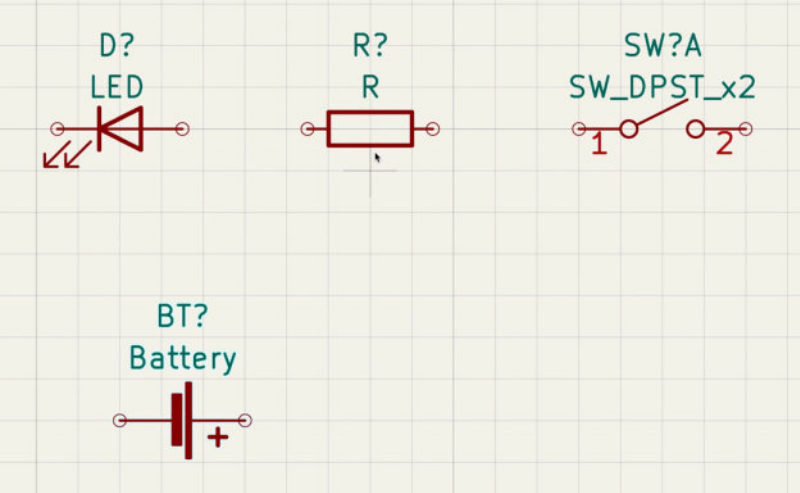


Figure 3.7.1: Symbol placement is complete.

Remember: you can rotate a symbol using the “R” hotkey after selecting it.

Annotate

Next, I will annotate the symbols. Annotation can be done manually or (preferable) automatic, and it entails setting unique reference IDs for each symbol.

First, what is the reference identifier? In Figure 3.7.1, notice that each symbol has a designator such as “D”, “R”, or “BT”, followed by a question mark. This is the symbol’s reference identifier. The reference identifier is a unique name for this symbol that we can use in the schematic and the bill of materials as an identifier for the symbol. The question mark indicates that the designator for the symbol is not yet set. To set it manually, double-click on the symbol to bring up its properties window (Figure 3.7.2).

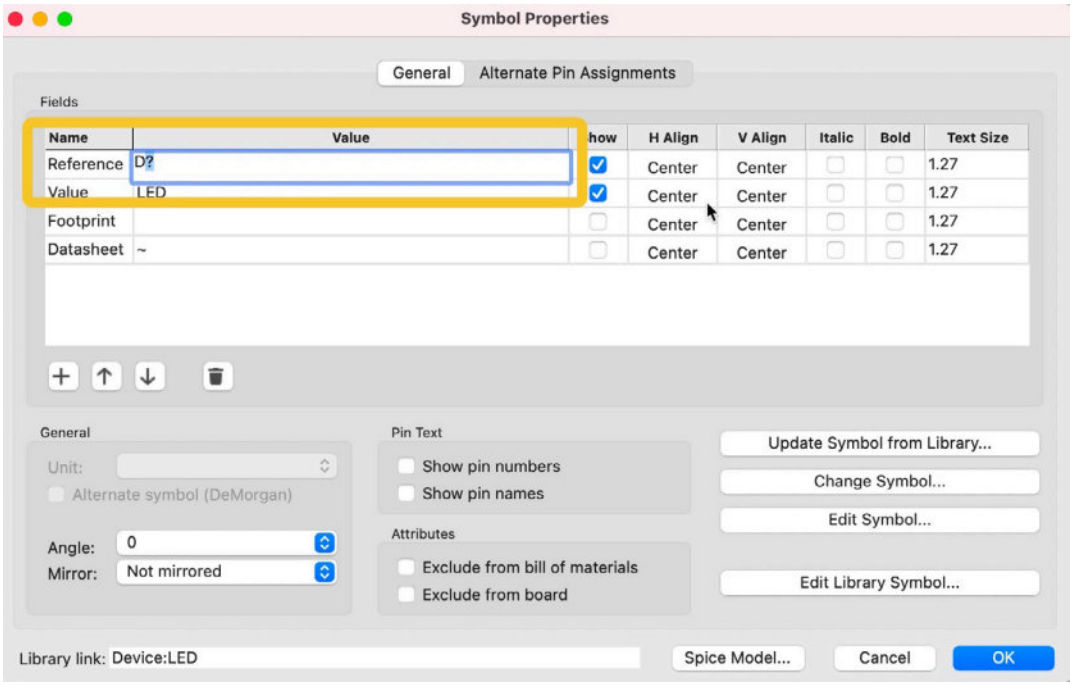


Figure 3.7.2: Setting the reference ID manually.

You can set the reference ID manually by editing the Reference field in the properties window. If you choose the manual method (which I discourage), you will have to keep track of the identifier assigned and ensure there are no duplicates. Click “Cancel” to dismiss the properties window.

A better way to set the identifiers is to use the automatic annotator tool. Bring up the schematic annotator window by clicking the Annotator button from the top toolbar (Figure 3.7.3).

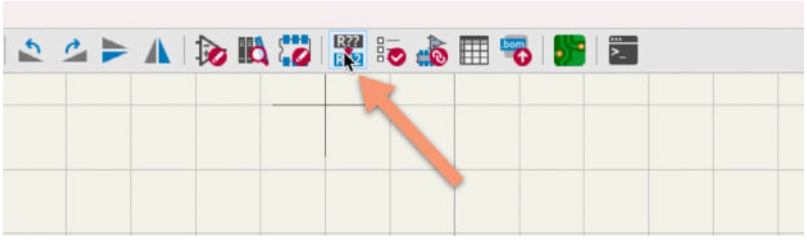


Figure 3.7.3: Invoke the Annotator tool.

This will bring up the Annotator tool window that looks like this (Figure 3.7.4):

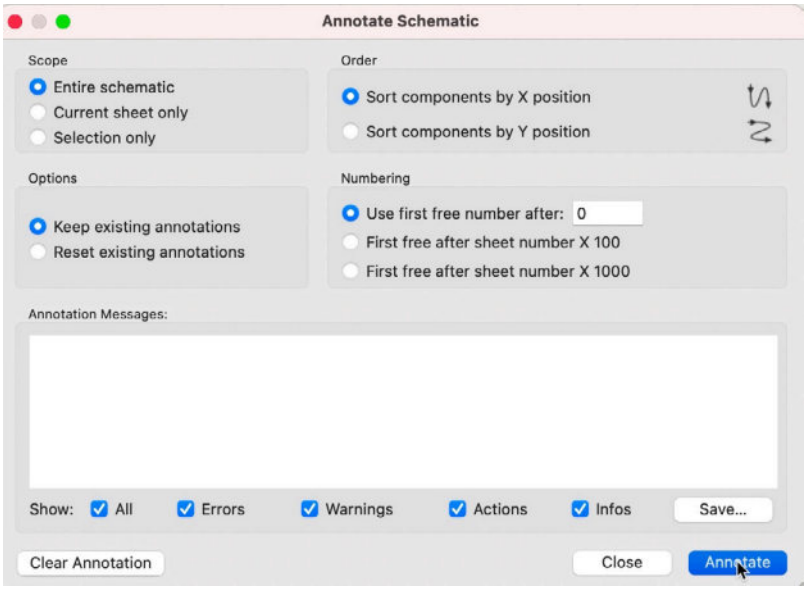


Figure 3.7.4: The Annotate Schematic tool window.

The default setting works well, and I rarely need to change them. Just click "Annotate" to let the tool set the reference IDs and then "Close" to dismiss the window.

Your schematic now looks like this (Figure 3.7.5):

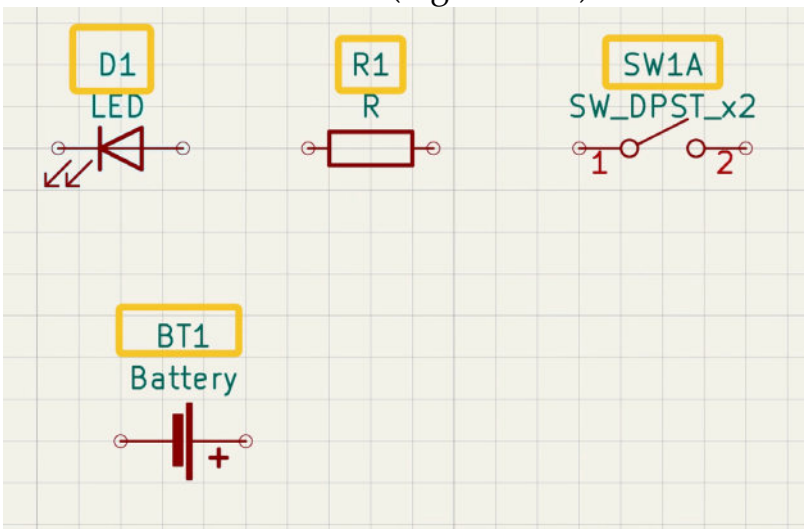


Figure 3.7.5: The annotated schematic.

Notice that the question marks are replaced with numbers, and each symbol now has a unique identifier.

Associate

Next up, association. In association, we choose the desired footprint for a symbol. Remember that the footprint defines the physical attributes of a component in the schematic diagram. For example, take the resistor in Figure 3.7.5. What will this resistor look like in the final PCB? Will it be a through-

hole component or an SMD? What will be its length and diameter? What are its silkscreen and other graphics?

There are several ways to associate a symbol with a footprint. You can assign a footprint, one symbol at a time, via the symbol's properties window. For example, for the LED, double-click on the symbol to bring up its properties. In the properties window, notice the Footprint attribute (Figure 3.7.6).

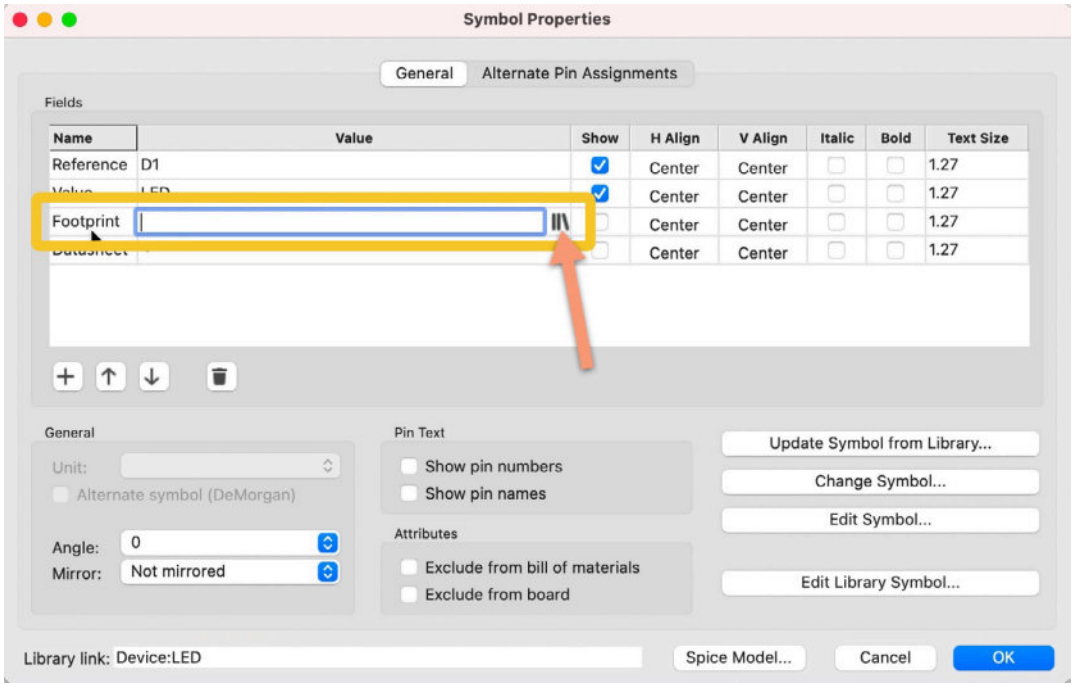


Figure 3.7.6: The Symbol properties window and footprint property.

You can type the footprint identifier in the footprint field, although it is safer to click on the footprint library button to bring up the footprint chooser window (Figure 3.7.7). You can use the footprint chooser to search (1) and browse (2) for the desired footprint. Double-click to select it (3) and associate it with the symbol (4) when you find it. Before you close the symbol properties window, click on the "Show" check box of the Footprint property to display the footprint reference in the editor.

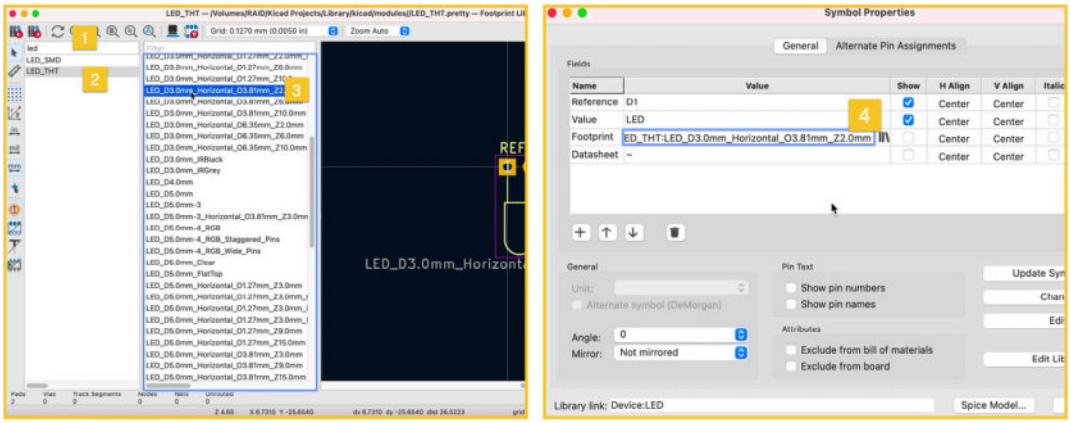


Figure 3.7.7: The footprint chooser and Symbol properties windows.

If your schematic only has a small number of symbols, this one-at-a-time method is sufficient. But for larger schematics, you will need a more streamlined approach. For this, Eeschema offers the association tool, which you can access from the top toolbar (Figure 3.7.8).

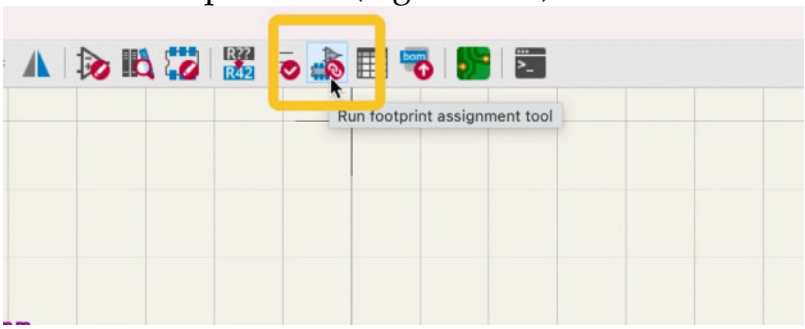


Figure 3.7.8: The Associations tool button.

The associations tool contains three panes (Figure 3.7.9). The middle pane shows the symbols (left side) and their associated footprints (right). The left pane (1) contains a list of footprint libraries, and the right page (3) a list of footprints based on the selected library and the filter settings (top of the window). You can learn how to use the associations tool in detail by reading the dedicated chapter later in this book.

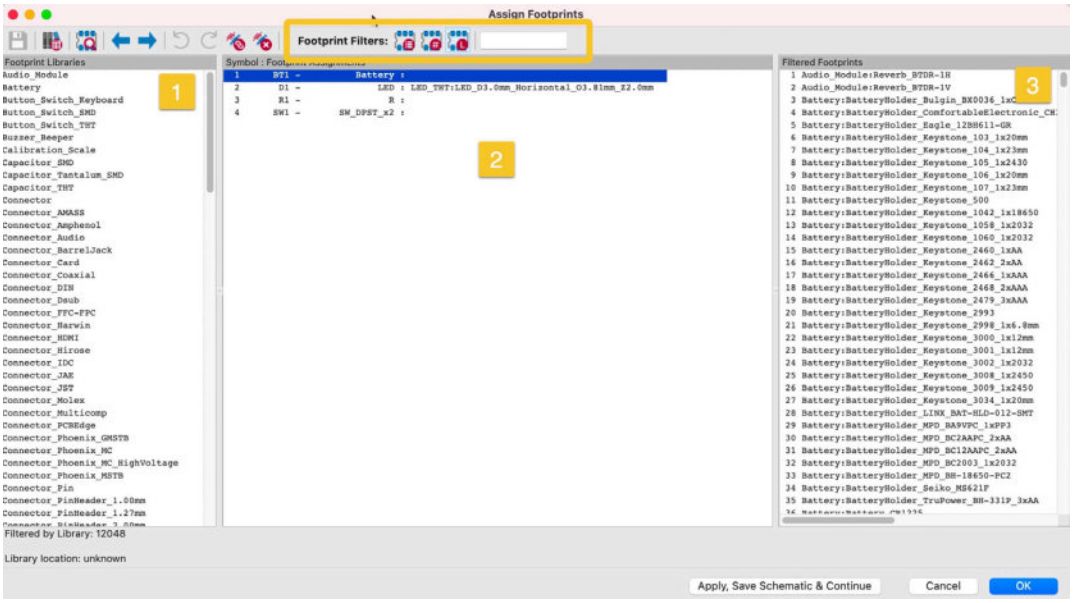


Figure 3.7.9: The Associations tool window.

As you can see in the figure above, the LED symbol already has an associated footprint. I manually assigned this footprint earlier in this chapter. You can change this association by selecting the LED's row and then double-clicking on an alternate footprint from the right pane (3). Also, notice that as you click on a symbol row in the middle pane, Eeschema pans the editor so that you can see the symbol in the schematic.

Let's re-associate the LED symbol to an appropriate footprint. I have enabled all three filters (description, pins, and library) from the top menu bar. I have typed "led" in the search box. In the left library pane (1), I have selected the "LED_THT" library. In the right pane (3), you will see a listing of all footprints in the selected library and match my filter settings (Figure 3.7.10).

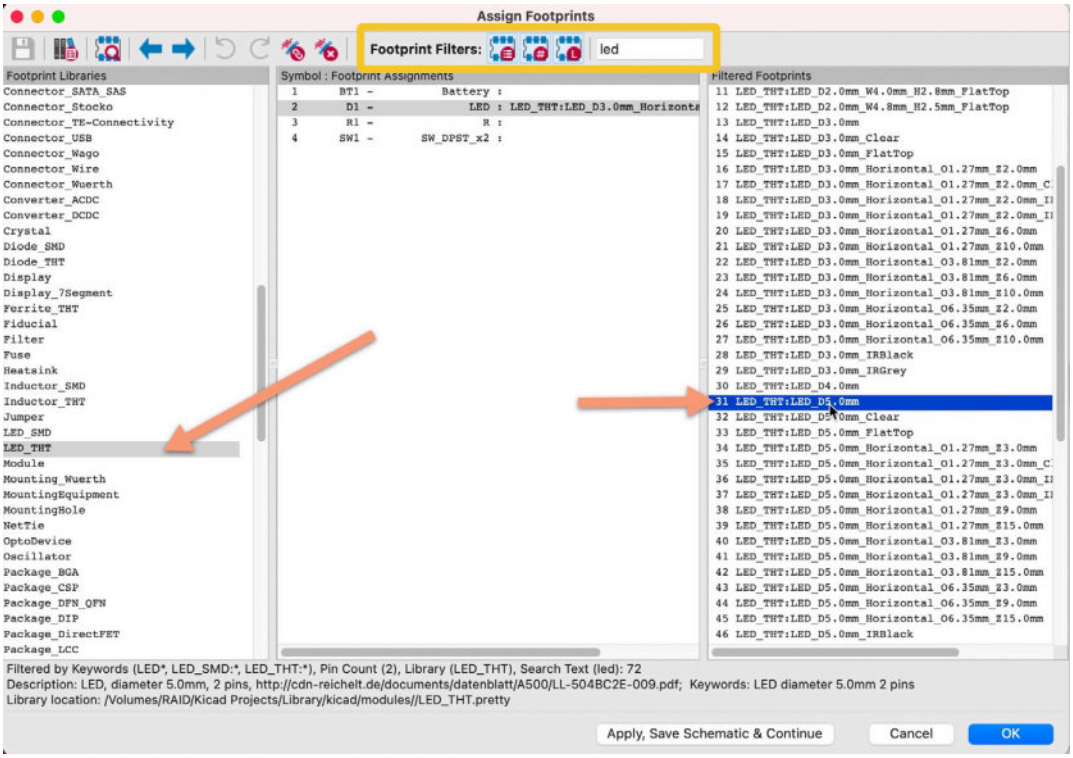


Figure 3.7.10: Making an association.

Double-click on the footprint in the right pane and notice how the association appears in the symbol row in the middle pane to finish the association.

Repeat the process so that all four symbols have their associated footprints. You can see my selected associations in Figure 3.7.11 (below).

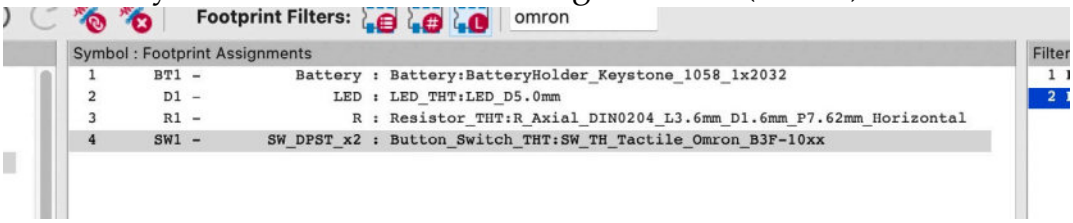


Figure 3.7.11: The completed associations.

My schematic editor now looks like this (Figure 3.7.12):

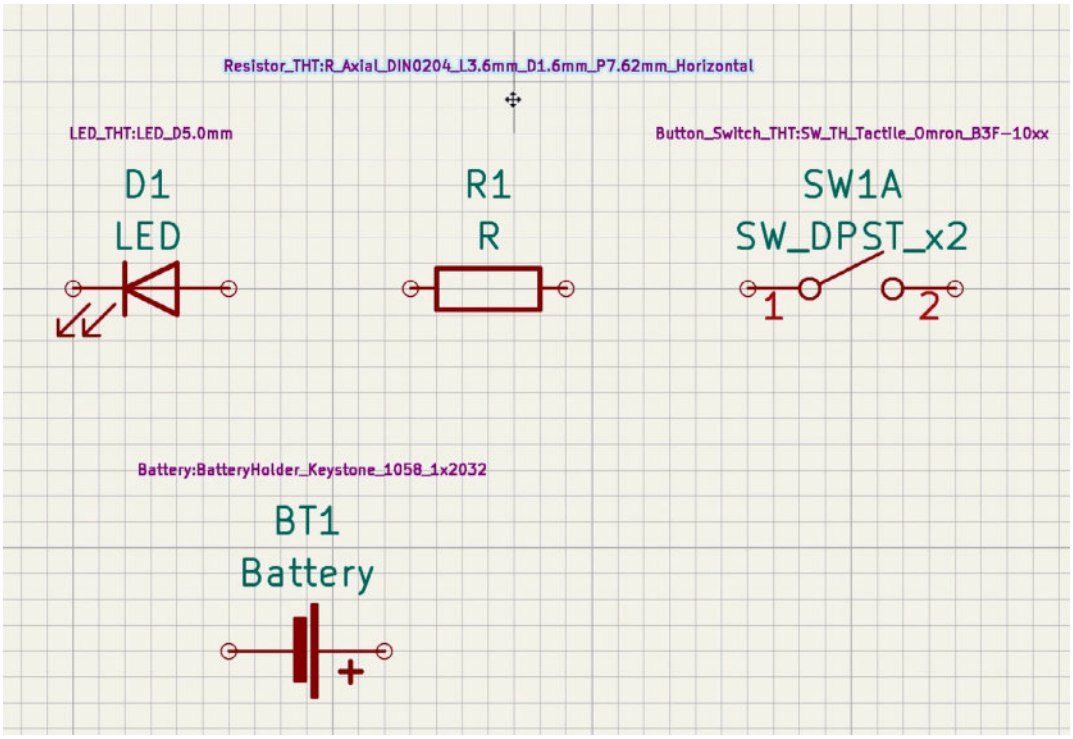


Figure 3.7.12: The schematic with symbols fully annotated and associated.

I have set the symbols to show their footprint properties (see earlier in this chapter on how to do this). I have also changed the appearance of the footprint property text to make it smaller. Learn how to do this in the relevant recipe chapter.

With the symbol and footprint associations complete, step three of the process is also done. Let's continue with the wiring in the next chapter.

8. 4 - Wiring

In this chapter, you will have completed step four of the schematic design workflow, that you learned about in the second chapter of this part of the book.

This chapter will show you how to wire the symbols you arranged and annotated in the last chapter.

To do the wiring, you will use the “wire” tool. You can enable this tool by clicking on its button from the right toolbar or typing “W” on your keyboard (Figure 3.8.1). Beware that there is a different behaviour between the way that the “W” hotkey and the wire button work. When you use the “W” hotkey, the editor will immediately start drawing a wire. However, if you enable the wire tool from the right toolbar, you will need to left-click inside the editor sheet to start drawing a wire.

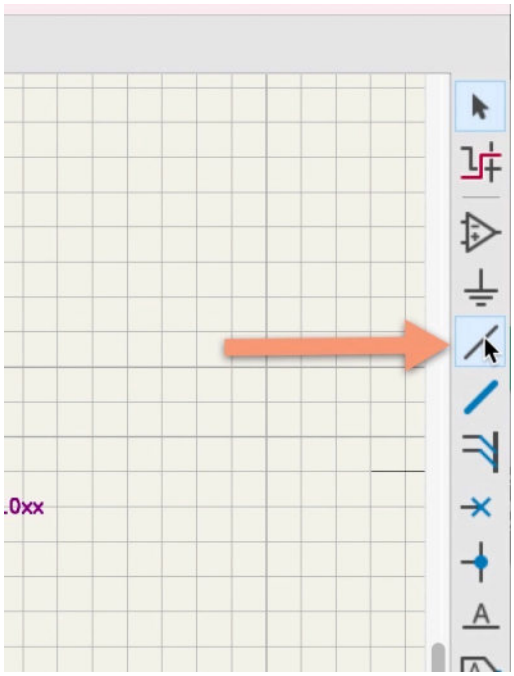


Figure 3.8.1: The Wire tool button.

For this example, click on the wire button to enable the Wire tool, place the cursor over one of the pins, and then left-click to start drawing. To draw a new segment and change the drawing direction, click again. To finish drawing a wire, either double-click or place the cursor over the destination pin and left-click.

Below you can see my first wire, composed of three segments, with 90-degree corners between them (Figure 3.8.2).

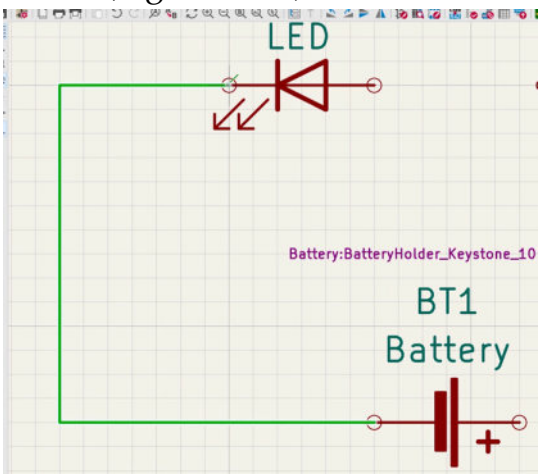


Figure 3.8.2: A wire connecting two pins.

Continue the same process to draw the wires between all pins. Remember, you can zoom and pan during the wiring as needed. Practice zoom in/out with the scroll wheel before you finish drawing a wire to become used to this operation.

By the end of the wiring process, your schematic will look like this (Figure 3.8.3):

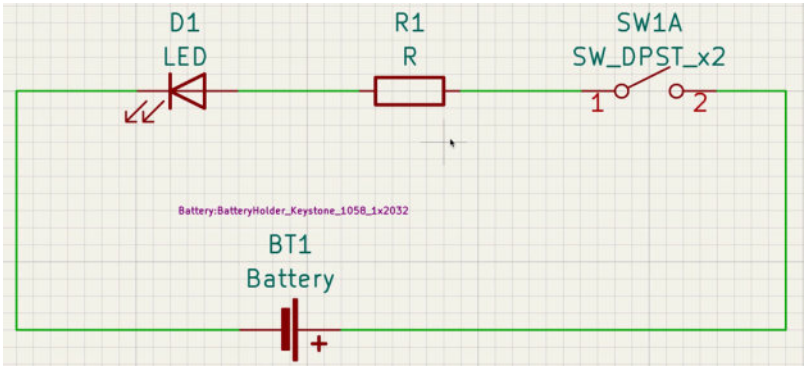


Figure 3.8.3: Completed wiring.

This completes step four of the wiring process. Let's continue with step five, which involves the creation of named nets.

9. 5 - Nets

In this chapter, you will complete step five of the schematic design workflow, that you learned about in the second chapter of this part of the book. This chapter will show you how to give custom names to the nets you created in the last chapter.

But first, what is a net? Think of a net as a representation of an electrical connection between two pins. In the circuit in Figure 3.9.3 you can see four wires, that connect four pairs of pins. For each of those wires there is a net. KiCad uses nets to keep track of which pins are connected to which other pins. While we (i.e. the "designers") see wires, KiCad "sees" nets.

If you have two wires that intersect and are electrically connected, both wires will belong to the same name. Therefore it is possible to have nets that contain more than one wire. You will see this frequently in later projects in this book. In this first project, it happens that each wire corresponds to one net.

It is possible to give nets custom names so that it is easy to identify. While you don't have to name all nets, it is good practice to give custom names to some important nets, such as those for the ground and operating voltage levels and signal nets.

To give a custom name to a net, you will use the net label tool. You can enable this tool from the right toolbar (Figure 3.9.1).

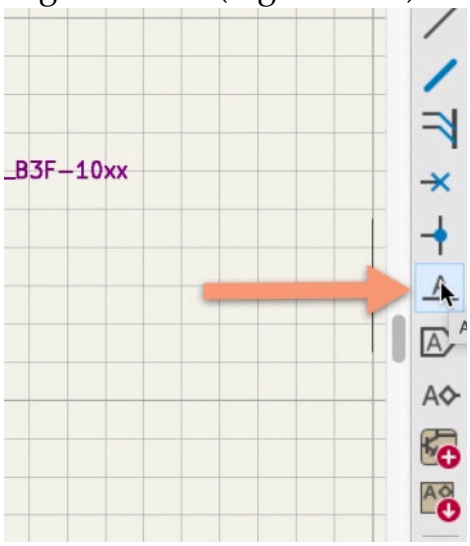


Figure 3.9.1: The net label tool.

To use it, click on the button to enable it (or type the “L” hotkey), and then click anywhere in the editor to bring up the label properties window. In the label text field, type the name of the net. In my example, I have typed “LED_cathode” (Figure 3.9.2, left). Click OK to create the label.

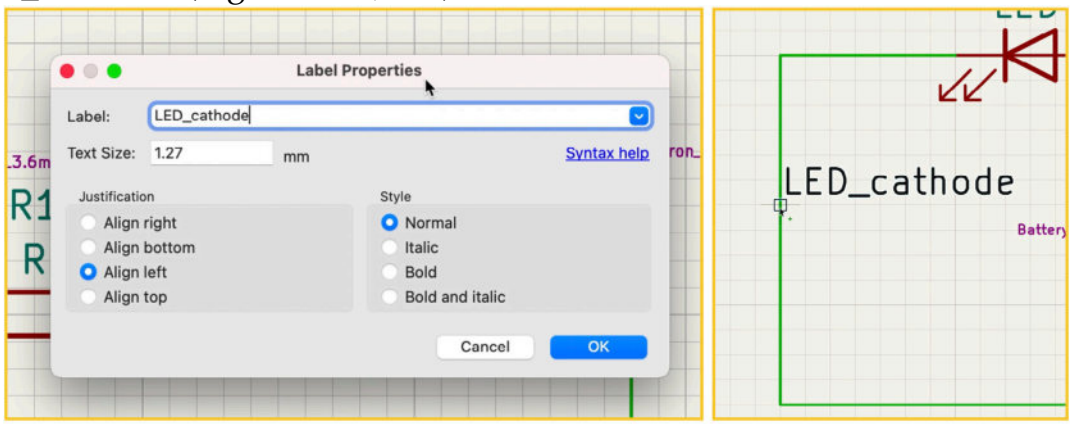


Figure 3.9.2: Creating and attaching a net label.

The new label will be attached to the mouse cursor when you dismiss the net label properties window. Move the small box of the label to overlap any part of the wire you want to name, and click to commit it.

That’s it. The wire and its net have a custom name.

Continue using the same process to create two additional named nets. Below is a list of all nets in this schematic:

- LED_cathode.
- LED_anode.
- bat_pos.

I did not give a custom net name to the wire that connects the resistor to the switch.

Below is the schematic at the end of step five (Figure 3.9.3):

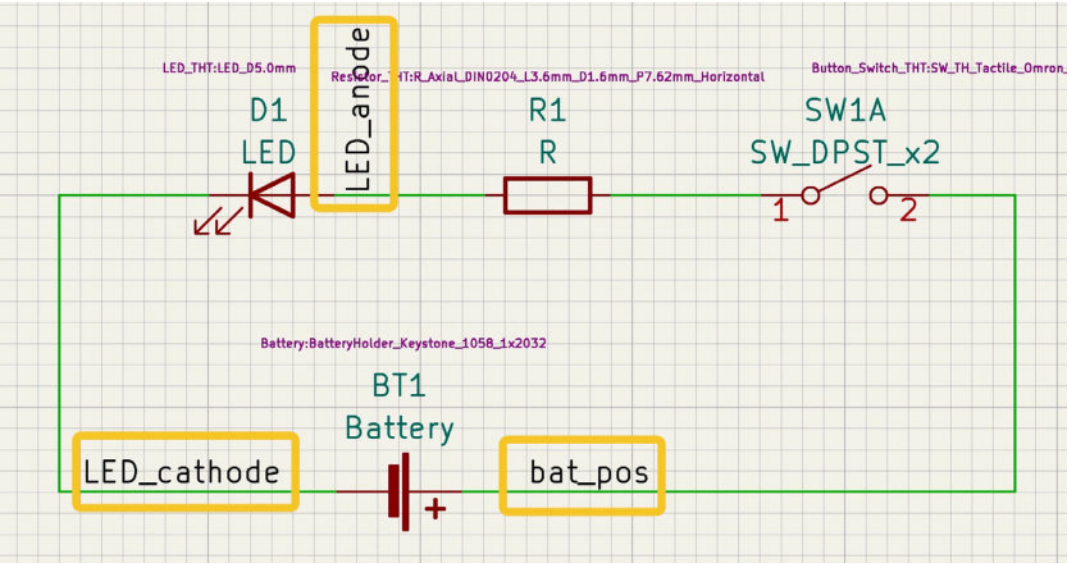


Figure 3.9.3: This schematic contains three named nets.

In the next chapter, I will show you how to perform an electrical rules check.

10. 6 - The Electrical Rules Check

In this chapter, you will complete step six of the schematic design workflow, that you learned about in the second chapter of this part of the book. This chapter will show you how to use the Eeschema electrical rules checker (ERC) tool to ensure that your schematic does not contain errors that should be corrected before continuing with the layout workflow.

To use the ERC, invoke it by clicking on its button in the top toolbar or the Inspect menu (Figure 3.10.1).

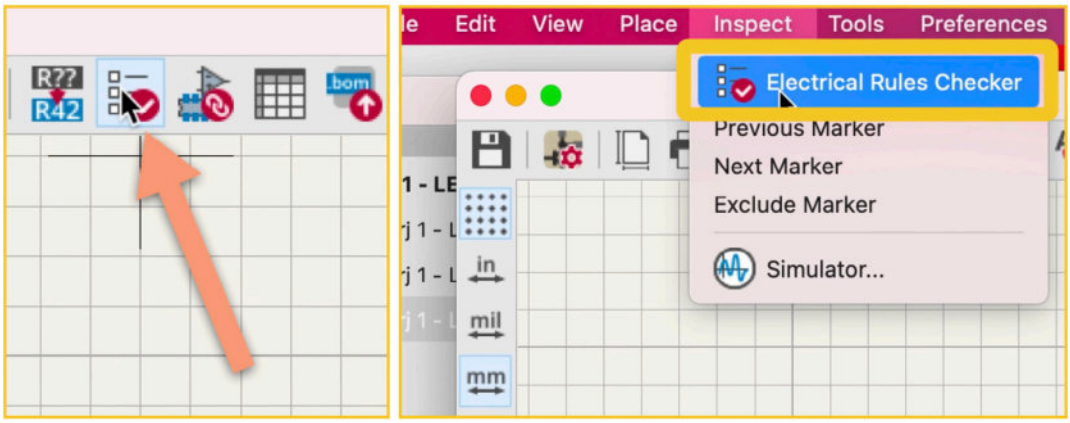


Figure 3.10.1: Starting the ERC tool.

The ERC window will appear (see Figure 3.10.2 below). You can run the check immediately by clicking on “Run ERC” (1). The results in my example show zero errors and warnings (see box below). You can click on the Violations (2) and Messages (3) tabs to switch between the two types of output that the ERC can provide. You can also use the checkboxes at the bottom of the window to enable or disable the various message types.

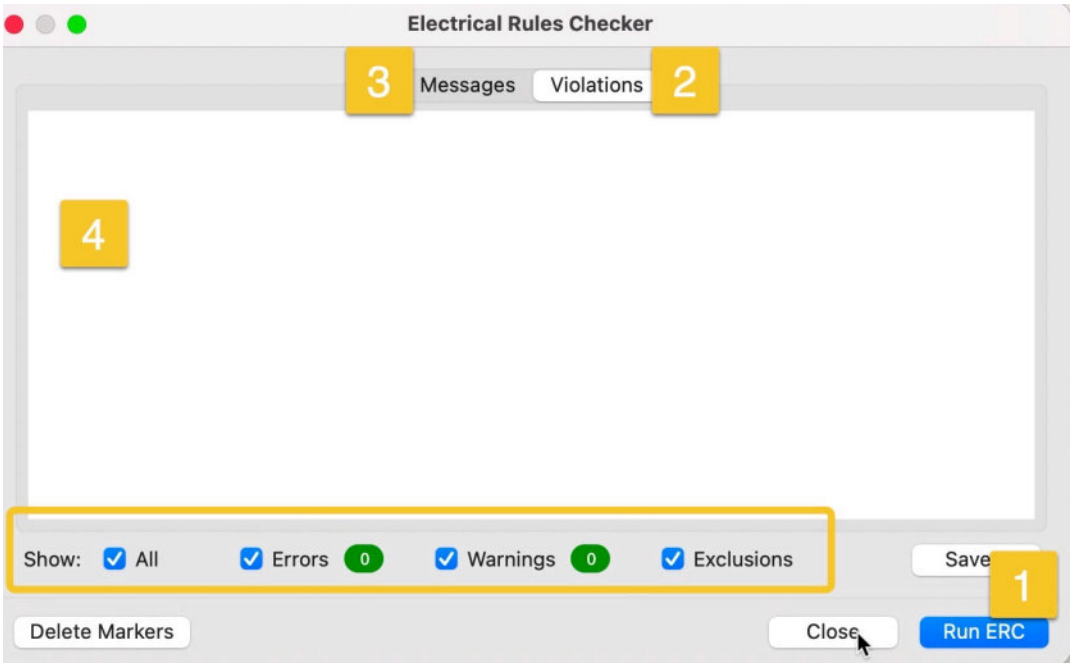


Figure 3.10.2: The ERC window.

Of course, we are working on a simple circuit, and I did not make any mistakes during the schematic workflow steps. I will deliberately delete the wire that connects the resistor to the switch and rerun the ERC. This time, the ERC reveals two violations (Figure 3.10.2):

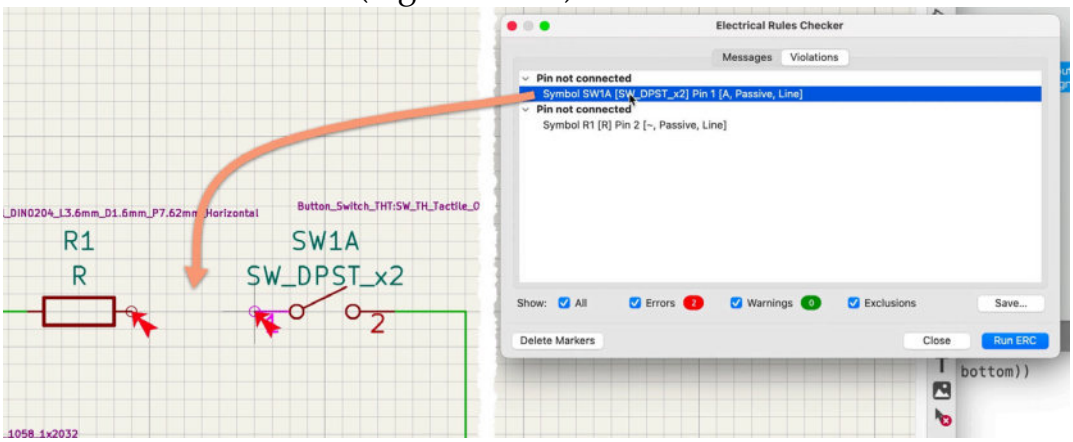


Figure 3.10.2: The ERC reveals two violations.

When you click on a violation in the ERC, the schematic editor in the background will pan to show you the location of the violation. There are also markers (arrows) that give you a visual clue as to its location. Use all the information that the ERC gives you to find and fix these violations before you continue. In the example above, the deleted wire caused two pins to be left unconnected. This is why the ERC lists the unconnected pins instead of the missing wire as the violation.

Go ahead and fix the violations by restoring the wire. Re-run the ERC to make sure that there are no remaining violations.

Now that the ERC passes, there is one step left in the schematic design process. In this step, I will add comments and graphics to the schematic. This is the equivalent of adding explanatory comments to software source code. Let's complete step seven in the next chapter.

11. 7 - Comments with text and graphics

This chapter will show you how to complete the schematic design workflow by adding explanatory comments and graphics. This is the equivalent of adding explanatory comments to software source code.

After completing step six in the previous chapter, your schematic looks like this (Figure 3.11.1):

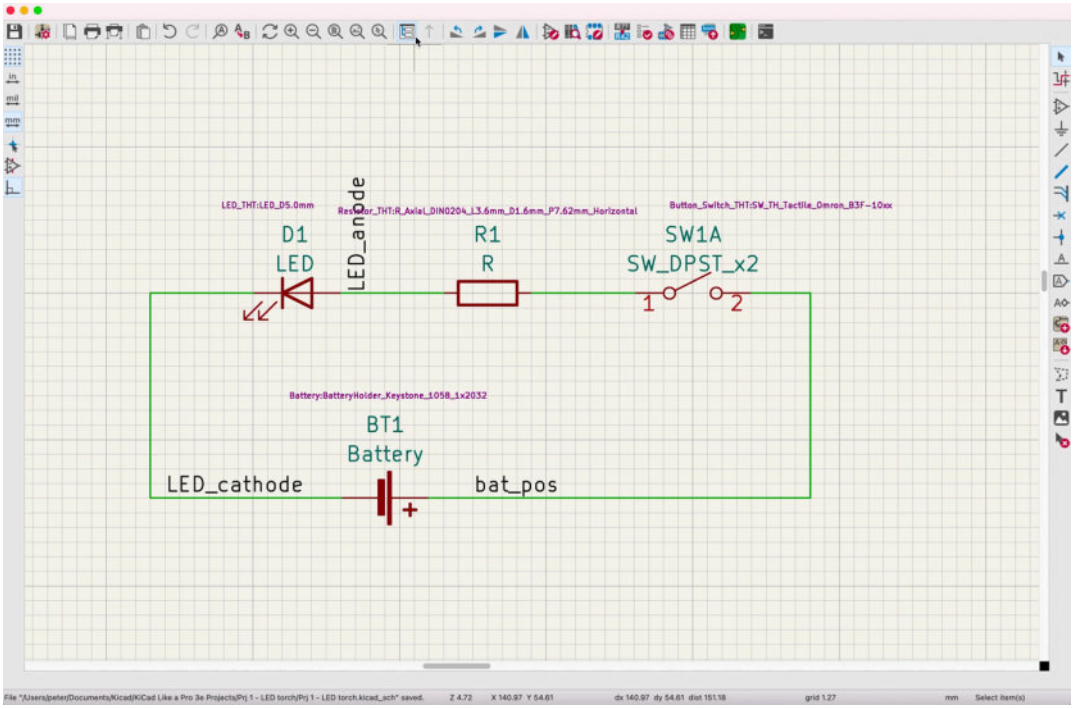


Figure 3.11.1: The schematic before adding annotations.

Similarly to how comments can make software source more readable, annotations in an electronics schematic can improve the document's readability.

To add annotations to the schematic design, you can use the tools at the bottom of the right toolbar (Figure 3.11.2):

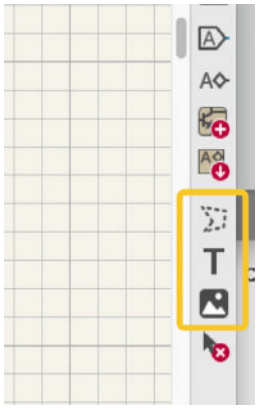


Figure 3.11.2: The line, text and graphics tool buttons.

I use the graphics line tool to create simple boxes that enclose the various components in functional groups, the text tool to insert names and information, and the graphics tool to insert images. For example, in the screenshot below (Figure 3.11.3), I use the line tool to create a box around the circuit.

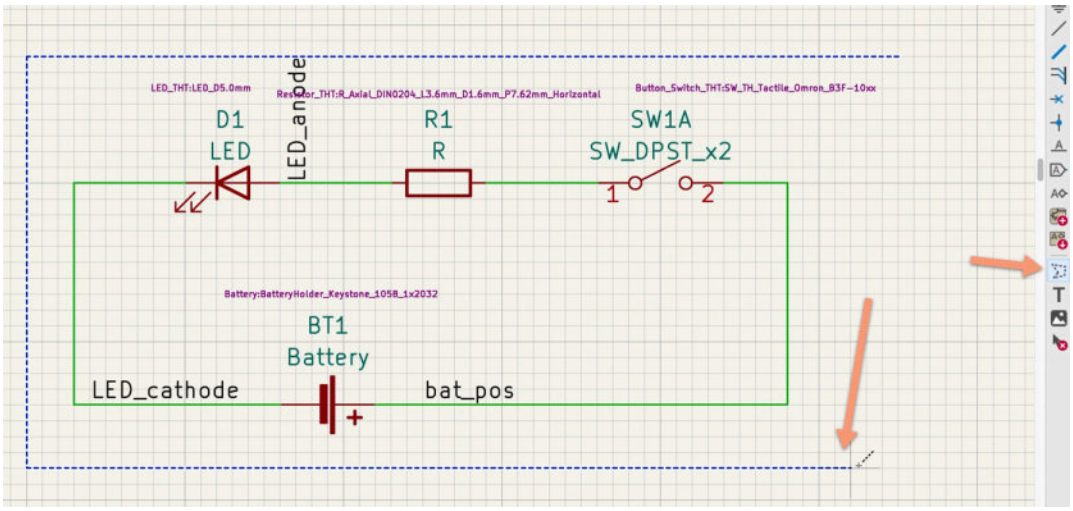


Figure 3.11.3: The line tool in action.

Then, I use the text tool to add a text comment. In this case, the comment is simply a short name for the box (Figure 3.11.3):

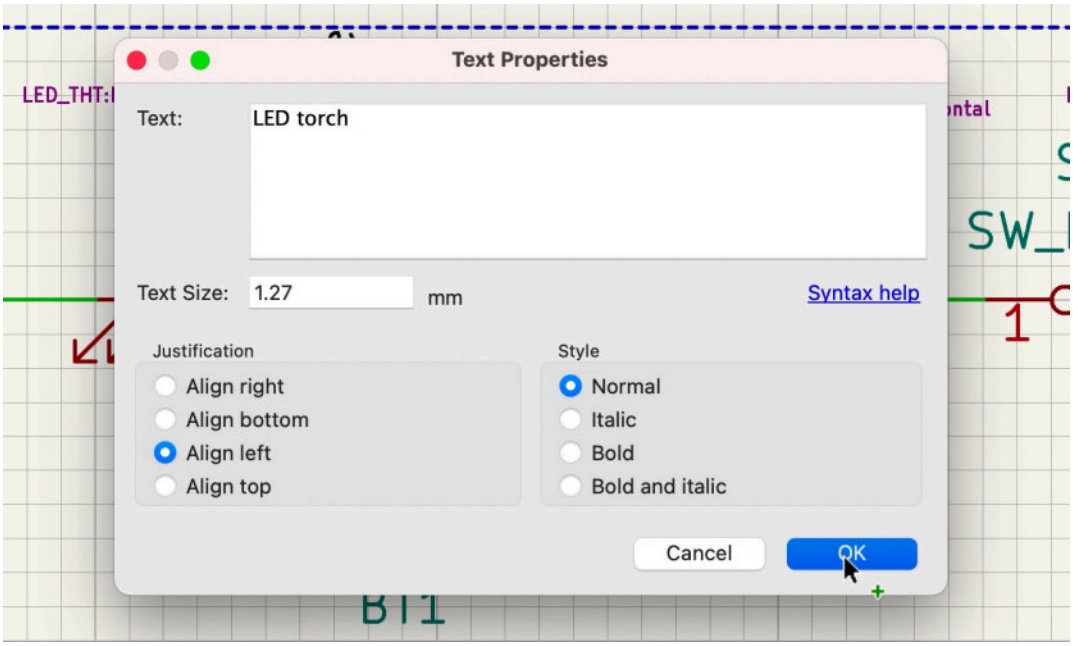


Figure 3.11.4: The text tool in action.

This example circuit is simple enough not to need too much commenting. Apart from the box, I have added two text items and completed the process with the final schematic, as shown in Figure 3.11.5.

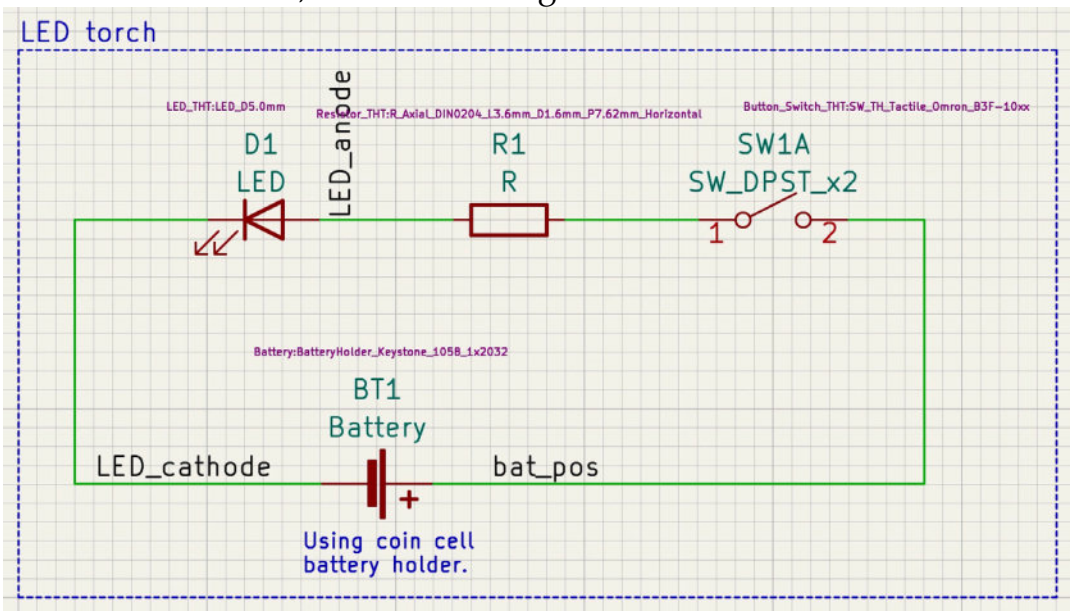


Figure 3.11.5: The final schematic.

And with this, the schematic design is complete.

Let's continue with the layout design workflow in the next Part of this book.

Part 4: Project- A hands-on tour of KiCad - Layout

1. Introduction to layout design and objective of this section

In the chapters of this part of the book, I will continue developing the LED torch project that I started in Part 3. At the end of the previous chapter, I completed the schematic design of the project PCB. I will now continue with the layout design.

To guide me with this work, I will follow the steps outlined in the layout design workflow that I outlined in Part 3.

To design the layout of the PCB, I'll be using Pcbnew. At the end of this part of the book, the PCB will look like this (Figure 4.1.1):

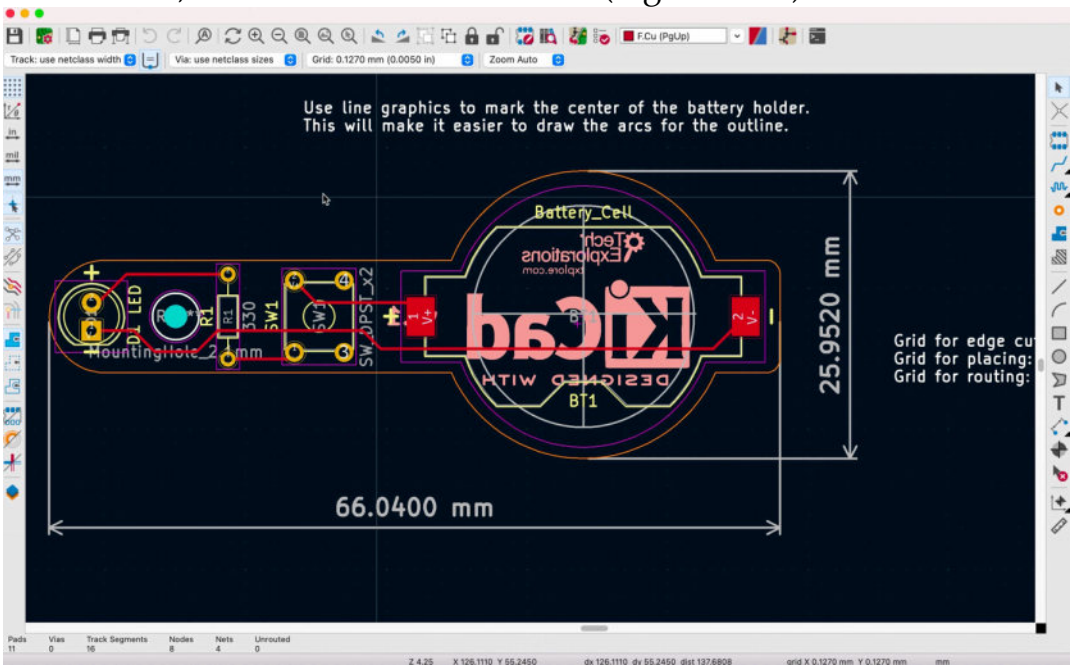


Figure 4.1.1: The final LED torch PCB layout.

Here is a 3D rendering, also made in KiCad (Figure 4.1.2):

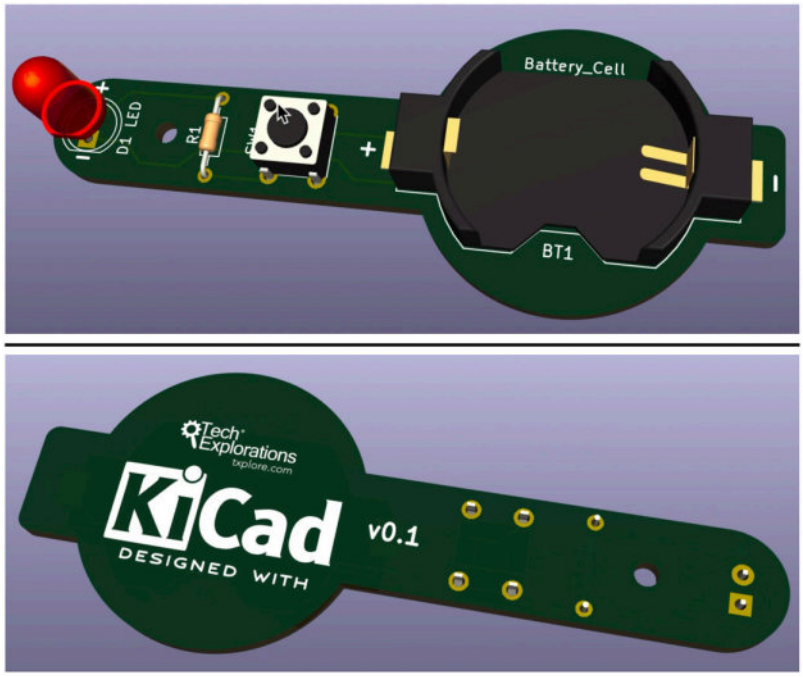


Figure 4.1.2: The final LED torch PCB layout in 3D.

The final PCB layout of this simple project contains several interesting elements:

- Both surface-mounted and through-hole components.
- Rounded edges moulded around the footprints of the PCB components.
- Silkscreen graphics (logos) and text.

Even though this is a simple project, it allows us to practice the essential skills for PCB design using KiCad.

Let's begin the layout design workflow with Pcbnew in the next chapter.

2. 1 - Start Pcbnew, import footprints

In this chapter, I will complete step one of the layout workflow, that you learned about in the second chapter of Part 3 of the book. In this step, I will start Pcbnew for the first time and import the schematic design from Eeschema.

When you start Pcbnew for the first time, it will present you with a blank designer space. When you import the schematic design from Eeschema, two primary elements of the design will appear in the editor:

1. The component footprints.
2. The nets that connect the footprint pins.

In addition to importing the schematic design data, step one of the layout design process is also an opportunity to set up the editor (or simply accept the defaults).

Let's begin.

In the main KiCad window, click on the Pcbnew button in the right pane (Figure 4.2.1):

Figure 4.2.1: Start Pcbnew.

You can also start Pcbnew from the main menu. Click on Tools, PCB Editor. Or, if Eeschema is already open, you can click on the Pcbnew button on its top menu (Figure 4.2.2).

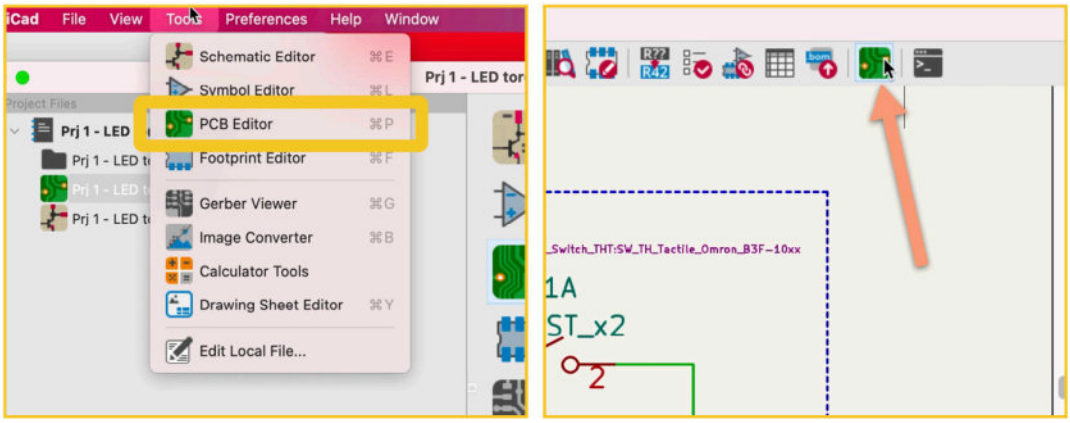


Figure 4.2.2: Other ways to start Pcbnew.

To import data from the schematic editor, you will use the importer tool. You can invoke this tool from the menu bar (Tools, "Update PCB from Schematic") or the top toolbar in Pcbnew (Figure 4.2.3):

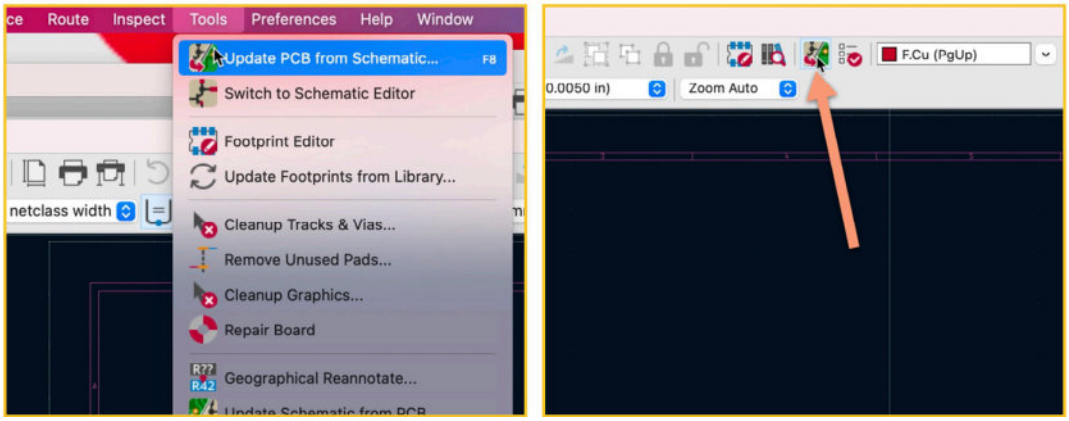


Figure 4.2.3: Start the importer tool in Pcbnew.

The importer tool window will appear (Figure 4.2.4). Because you are importing data into an empty design editor, it doesn't matter what the status of the various options is. These options are helpful when you import and update from the schematic designer into the layout editor. For example, you may decide to delete a symbol from your schematic. When you import the new schematic data into an already populated layout editor, you can ensure that KiCad will delete the footprint for the deleted symbol by enabling the "Delete footprints" option in the importer tool window.

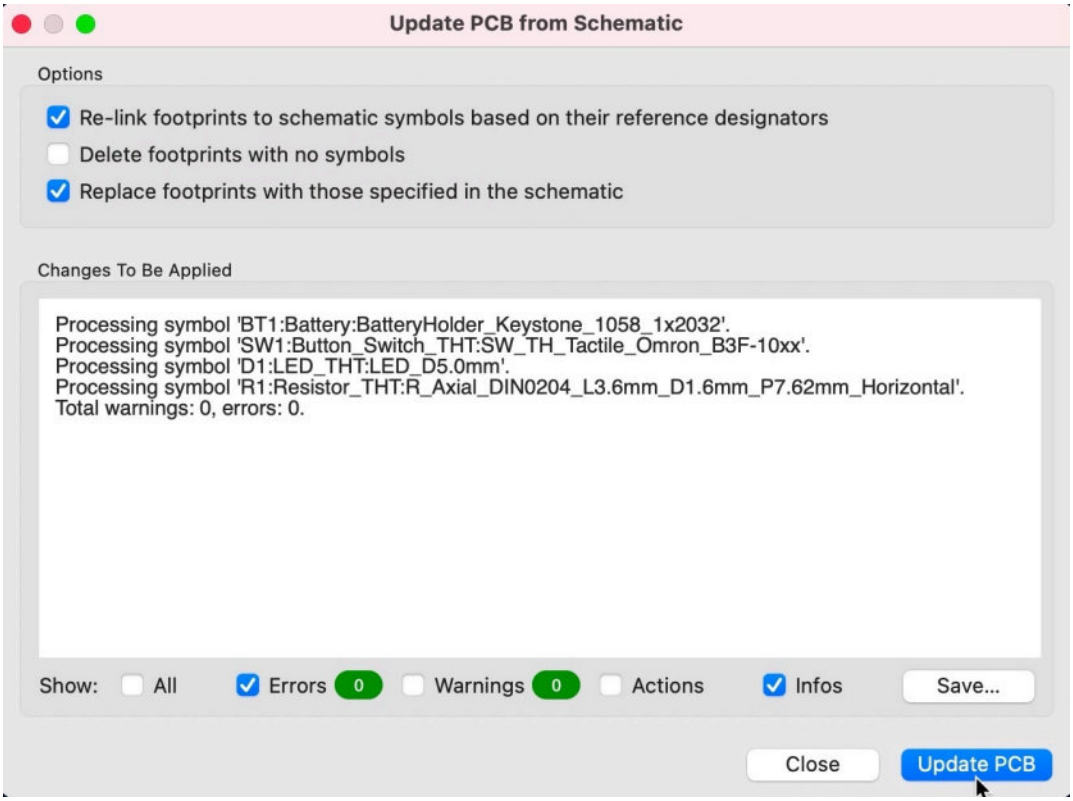


Figure 4.2.4: The schematic data importer tool window.

In this case, simply click “Update PCB” to populate the layout editor and then “Close” to close the importer window.

Your layout editor should now look like this (Figure 4.2.5):

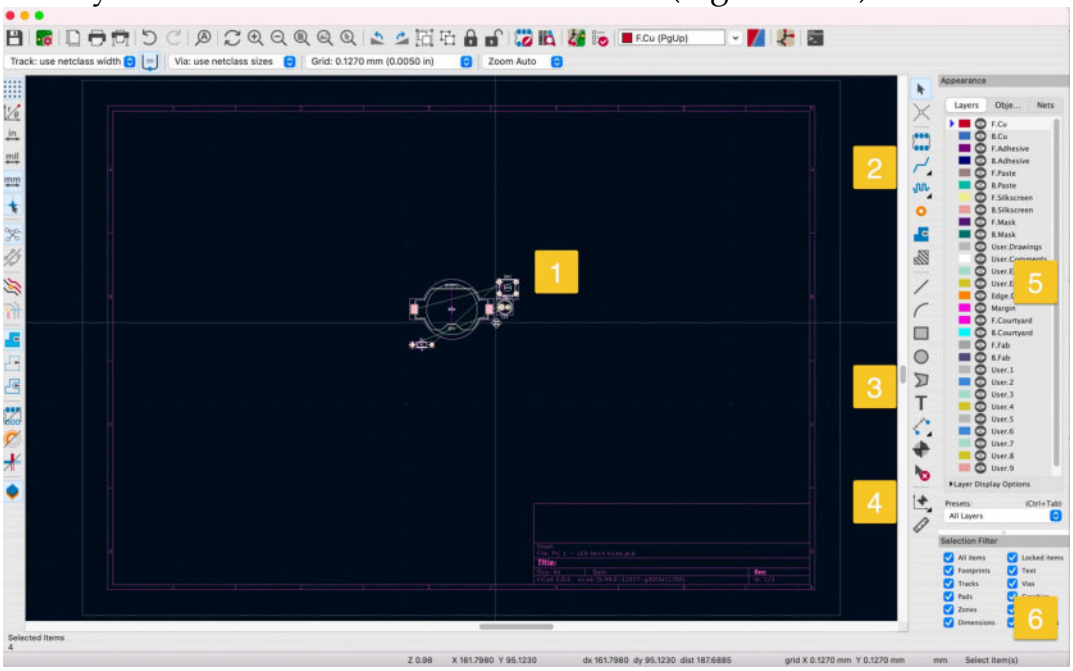


Figure 4.2.5: The schematic data imported into Pcbnew.

The footprints are bundled together and attached to the mouse cursor (see “1” above). You can move your cursor with the attached footprints to a

suitable location and then left-click to drop them in the editor. You can also see the thin “rat nest” lines that depict the nets or connections between the various pads of the footprints.

The layout editor offers various tools to help with the design process. In the right toolbar, you can use the buttons in the top part (“2”) to do things such as add additional footprints to the editor (that can be independent of those defined in the schematic) or draw wires. You can use the buttons in the middle of the right toolbar (“3”) to draw graphics using the line, box and circle primitives. And you can use the buttons in the lower part of the right toolbar (“4”) to measure distances between any two points of the editor.

In the Appearance group (“5”), you can choose the layer you want to use. Most of your work will be done in the front and back copper layers (“B.Cu”, “F.Cu”), the silkscreen layers (“B.Silkscreen”, “F.Silkscreen”) and the edge cuts layer (“Edge.Cuts”).

The Selection Filter (“6”) allows you to select the layout elements that the mouse cursor can select. This is particularly useful in busy layouts where several elements overlap, making it challenging to select a specific one. For example, you could have a silkscreen element in the back silkscreen layer overlying with a wire on the back and the front copper layer. Without the filter, KiCad would not know which element you want to select when you click on one of the overlapping elements and will present you with a context menu from where you can choose one.

You will learn how to use all of the features I described above (and much more) in this book.

There is one last thing I’d like to do before continuing with step two of the layout workflow: edit the page settings form.

From the File menu, select Page Settings (Figure 4.2.6):

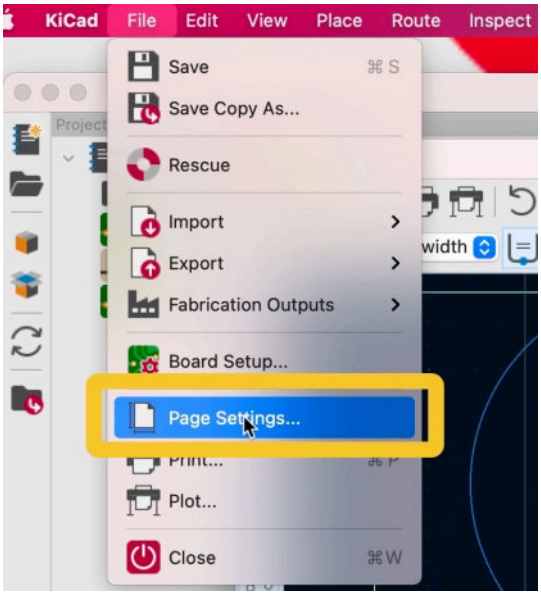


Figure 4.2.6: Open the Page Settings window.

In the Page Settings window, add some helpful content in form fields. The information you enter will appear in the layout sheet information corner (Figure 4.2.7):

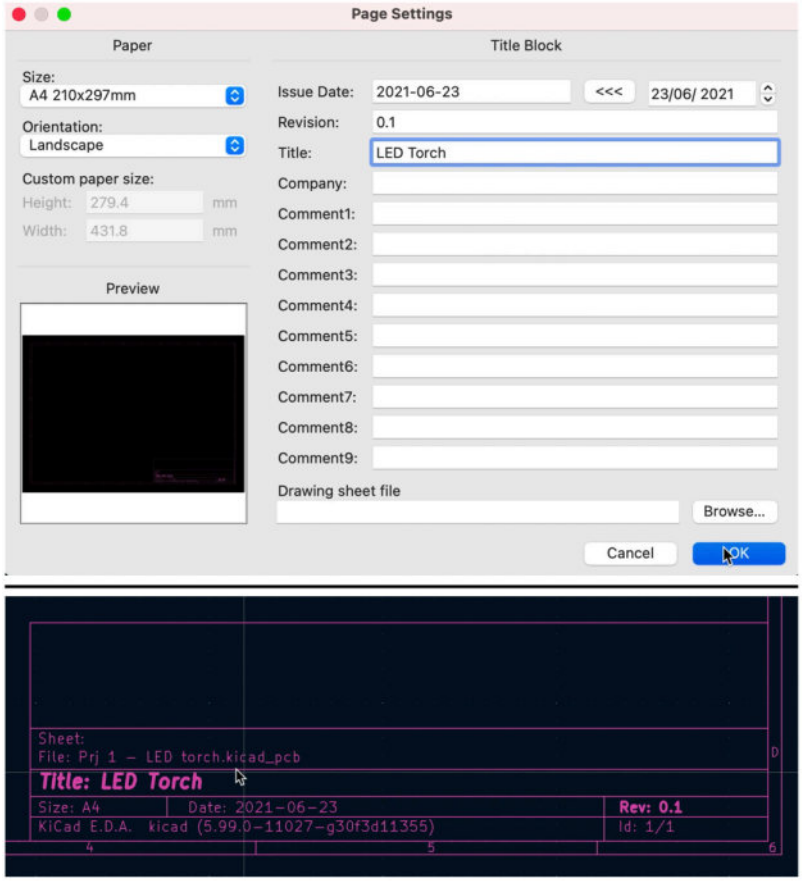


Figure 4.2.7: The Page Settings form (top) and the sheet information corner (bottom).

The first step of the layout design workflow is complete. Save the file and continue with step two in the next chapter, where you will draw the PCB outline based on the geometrical constraints of the project.

3. 2 - Outline and constraints (edge cut)

In this chapter, I will complete step two of the layout workflow, that you learned about in the second chapter of Part 3 of the book. In this step, I will draw the rough outline of the board based on simple geometrical constraints. To define these constraints, I ask these questions:

1. How large would I like my LED torch to be? (This dictates the board's overall size and cost).
2. How will I turn the LED on and off? (This dictates the location of the button or switch).
3. What is the biggest component of the board? (This dictates the shape of the board needed to accommodate the largest component).
4. How will I attach the board to an enclosure? (This dictates board features to help finish the final device).

These questions will help me figure out the physical dimensions and the shape of the board. As every board is different, the geometrical constraints will also differ. At the start of the layout workflow, you should always take some time to think about the appropriate questions to ask and their answers.

Here are my answers to the constraint questions that I asked myself:

1. The LED torch should be large enough to hold in one hand and fit in my pocket easily. Around 65 mm in length and 25 mm in height should be sufficient.
2. I will use a small momentary button. I will be able to press the button with my thumb. When I press the button, the LED turns on.
3. The biggest component of the board is the coin cell battery holder.
4. I will design a 3D-printed enclosure. I will attach the LED torch board to the enclosure utilizing a screw, mounting bosses or slide-in rails. The board itself should include one screw mounting hole.

With these constraints in place, I can proceed to create a rough outline of the board in the edge cuts layer. This outline is "rough" because I expect to refine it after I place the footprints within the outline.

I will select the User.1 layer from the Layers tab (under Appearance, "1", see Figure 4.3.1). The user layers allow me to add text and graphics that the manufacturer ignores. I plan to create a box with the approximate dimensions of my PCB and then confirm that the footprints (especially the battery holder) will fit within that space. After the necessary adjustments, I will switch to the Edge.Cuts layer and draw the actual board outline.

Next, I will select the graphics box tool by clicking on its button from the right toolbar ("2").

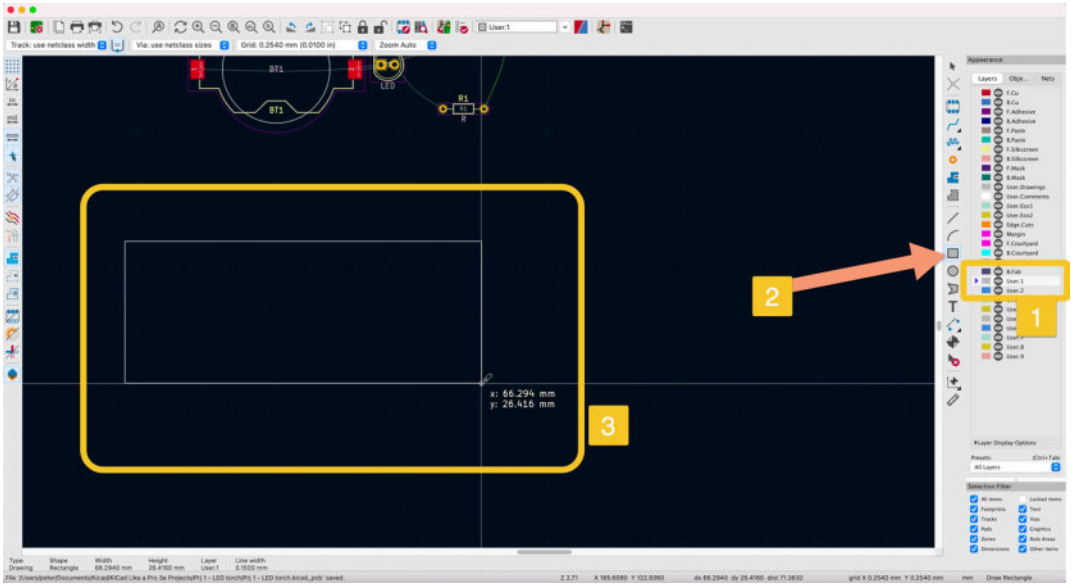


Figure 4.3.1: Drawing the rough outline of the board.

With the box tool selected, I click to start drawing a rectangle and drag the mouse until I have reached the needed dimensions. In the example above, I am drawing a rectangle with 66.29 mm in length and 26.41 mm in height. Click again to finish the drawing.

I now have an outline. Can it fit the components? Let's test. Drag the footprints, starting with the largest one, into the outline (Figure 4.3.2).

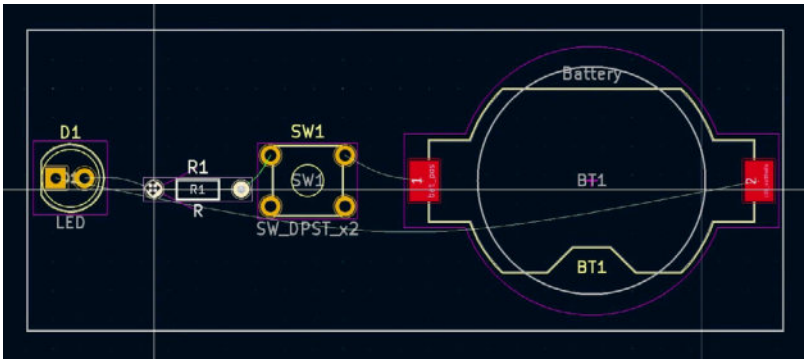


Figure 4.3.2: Yes, this outline can accommodate the footprints.

As you can see in the figure above, this outline can accommodate the footprints with much room to spare. I can decrease the height to reduce the overall size and cost of my PCB. Before I change the height of the outline, I will add the mounting hole, which is constraint four, from the list at the start of the chapter. I will use the graphics circle tool and draw a circle next to the LED footprint. I have move R1 out of the way temporarily (Figure 4.3.3).

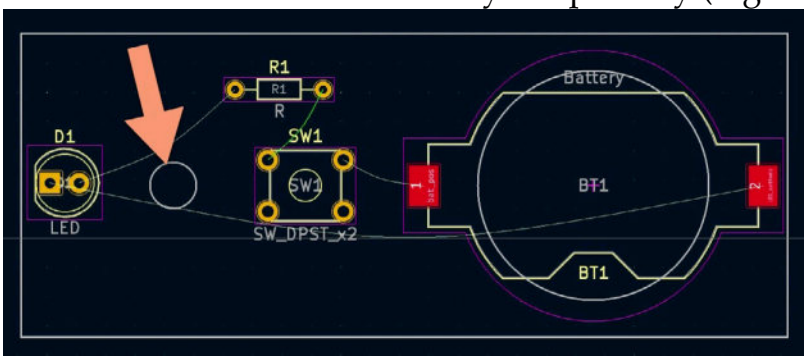


Figure 4.3.3: The circle represents a mounting hole.

I will now change the height of the outline so that the board is thinner. Click on the white outline to reveal the handles in the corners and middles of each line (Figure 4.3.4):

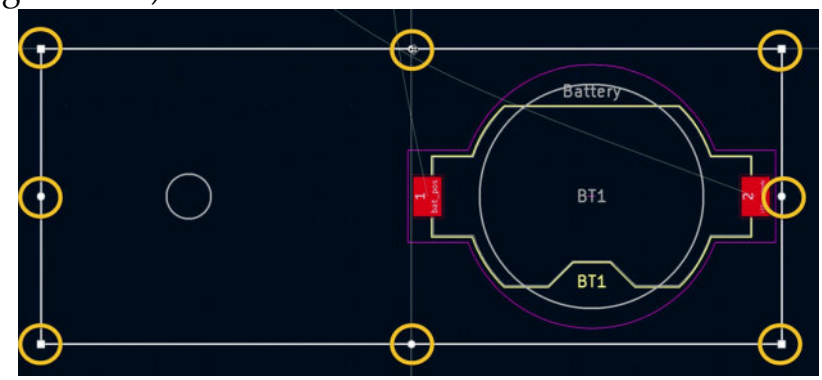


Figure 4.3.4: Use the handles to change the dimensions of the outline.

Use the handle in the middle of the top line to drag the line down. Use the handle in the middle of the bottom line to drag the line upwards. The result looks like this (Figure 4.3.5):

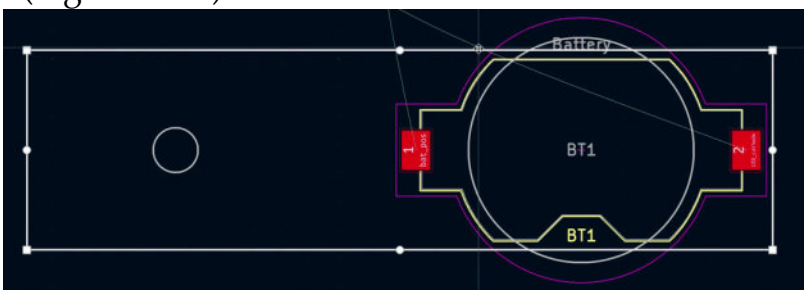


Figure 4.3.5: A board outline with reduced height.

This outline is more streamlined. The battery footprints yellow front silkscreen outline is fully enclosed within the board outline. The purple line is in the front mask layer, and the white line user comment lines are partially outside the board outline. It is safe to ignore this for now. When I refine the outline later, I will use circular segments to fully enclose the entire battery holder footprints within the PCB's outline.

Now that I know what the rough outline of this board should be, I will draw it in the edge cuts layer. Regarding Figure 4.3.6, click on Edge.Cuts ("1") in the Layers tab to switch the active layer. Select the line tool from the right toolbar ("2"). Start drawing the first line from the top right corner of the box ("3") until you reach the other end ("4"). Click to start drawing, click again to add an edge, and double-click to finish the drawing.

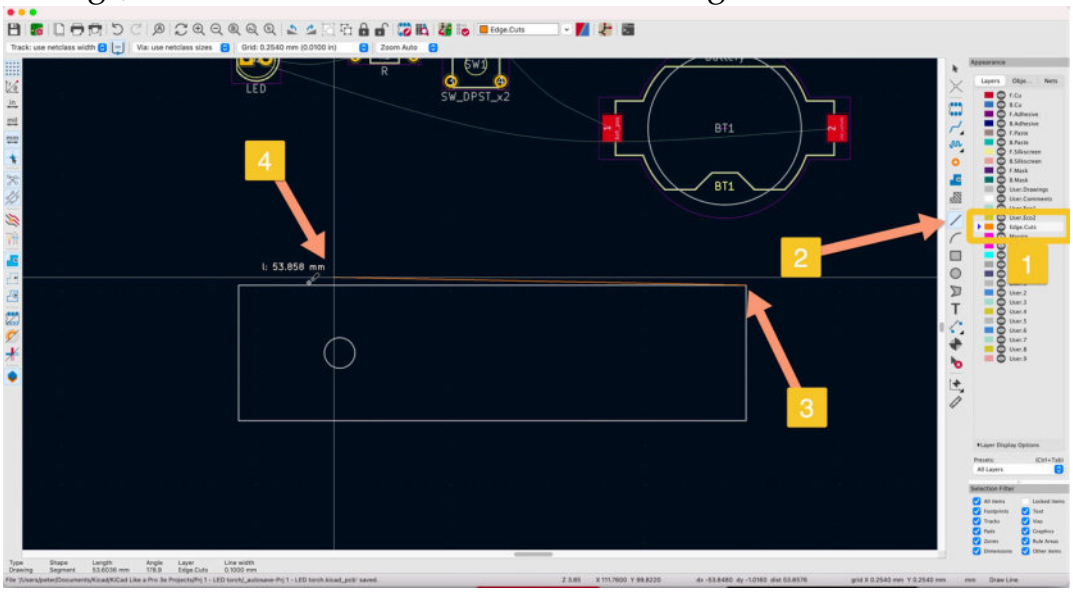


Figure 4.3.6: Drawing in the Edge.Cuts layer.

Below you can see how I traced the outline in the edge cuts layer until I had the entire board outline (Figure 4.3.7).

Figure 4.3.7: Drawing in the layer in five steps.

Once you complete the drawing of the outline in the edge cuts layer, you can use the 3D viewer to see the rough board and its components in 3D space:

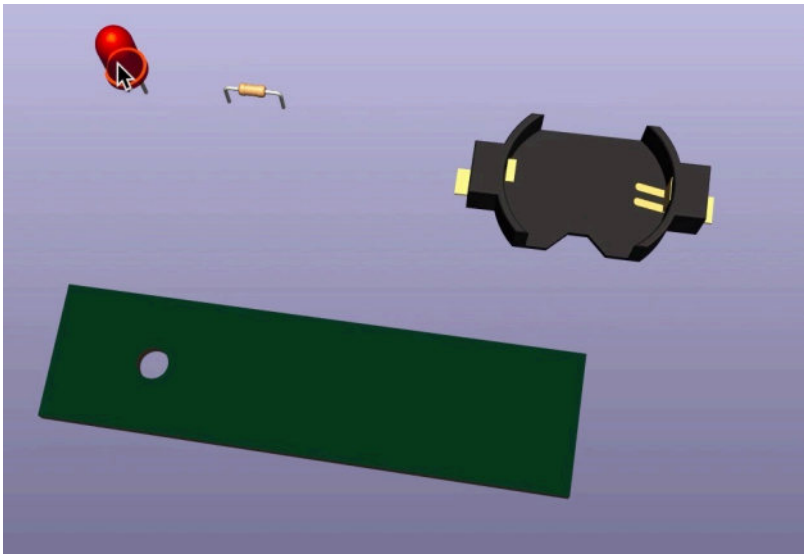


Figure 4.3.8: The board and its components floating in 3D space.

The current version of the board is not particularly appealing, but it is good enough for the next step of the workflow, where I will place the footprints within the board. I will do that in the next chapter and then work on the refinement of the board.

4. 3 - Move footprints in place

In this chapter, I will complete step three of the layout workflow, that you learned about in the second chapter of Part 3 of the book. For my simple design, this means moving the footprints within the PCB outline that I drew in the previous lecture.

The emphasis is on placing the footprints that have a user interface or important functional role first and then continue with the rest. In this example, there are two such footprints: the LED and the button.

I will place the LED at one end of the board to direct its light away from the device. As for the button, I will place it at a location that makes it convenient to press it with my thumb as I am holding the device in the palm of my hand. Each design is unique, so there can also be other considerations that weigh-in in the placement decisions. For example, this simple LED torch also has a large battery holder footprint. It makes sense to place this footprint towards the edge of the PCB, away from the LED and the button. This placement will give the device a better grip as the bulk of its mass will be inside the palm of my hand and will not obstruct the button.

Below you can see the layout as I left it at the end of the previous chapter (Figure 4.4.1).

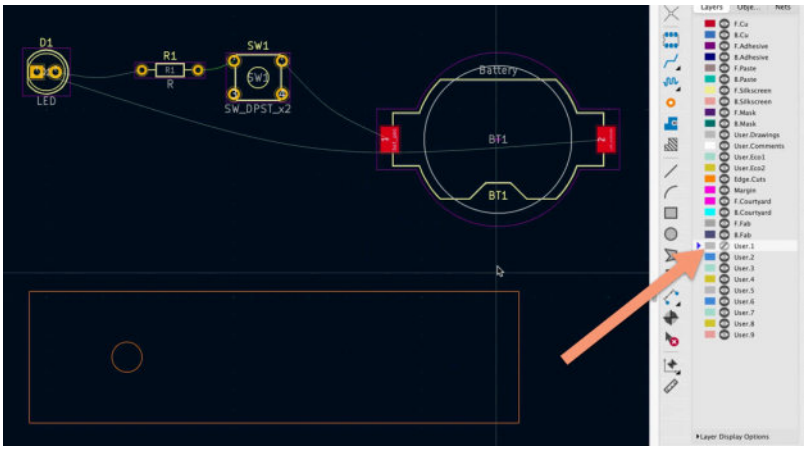


Figure 4.4.1: The rough outline of the PCB and the footprints.

The front of the torch is on the left side. In the figure above, notice that I have disabled the User.1 layer. The User.1 layer is where I drew the graphics box that represented the outline of the PCB in the previous chapter. I used this

box as a guide to help me draw the actual outline in the Edge.Cuts layer using the individual line segments. I will not need the contents of the User.1 layer going forward, so I have disabled it to reduce clutter in the layout editor.

Another consideration is the geometry of the routes that will eventually connect the pads of the footprints. The layout editor provides visual clues about the routes by showing the pad-to-pad connections using the "ratnest" lines. In general, try to place and orient the footprints to minimize the amount of overlapping ratnest lines. Fewer overlapping ratnest lines will result in a cleaner and easier to draw a network of routes.

I can now start placing the footprints within the outline of the PCB. To make it easy to select footprints only, I will choose "Footprints" only in the Selection Filter (Figure 4.4.2):

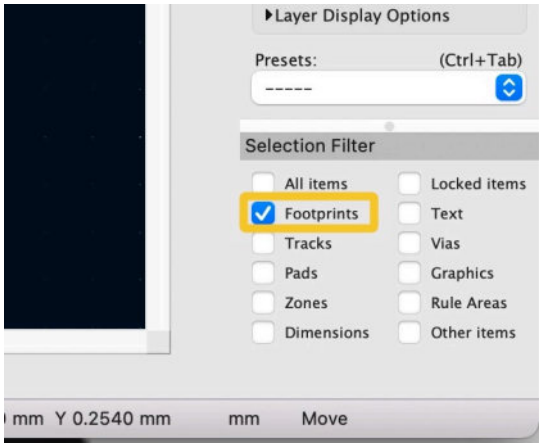


Figure 4.4.2: The Selection Filter.

To move a footprint in place, left-click on it to select it, then left-click again and hold to move it. Release the mouse button to stop moving the footprints. While moving the footprint, you can rotate it using the "R" and "Shift-R" hotkeys.

To help with the precise placement and alignment of the footprints, use the grid and the cursor crosslines. I prefer to use full-window crosslines better to infer the position of a footprint against its neighbours.

After I moved the footprints in their final locations, my PCB looks like this (Figure 4.4.3):

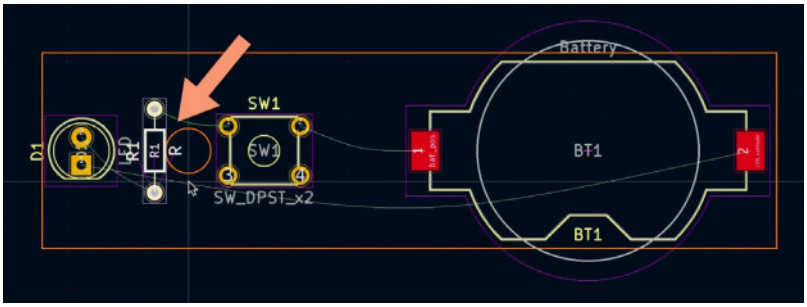


Figure 4.4.3: The final positions of the footprints.

The arrow points to a location on the PCB where the resistor overlaps with the circle graphic in the Edge.Cuts layer. I don't like to move the resistor closer to the LED and don't want to move the resistor elsewhere (even though there is space between the button and the battery holder). So, I will simply move the circle a little closer to the button. In the Selection Filter, select "Graphics", and then click and click-hold the circle to move it by around 1 mm towards the button.

Here is the current state of the layout (Figure 4.4.4):

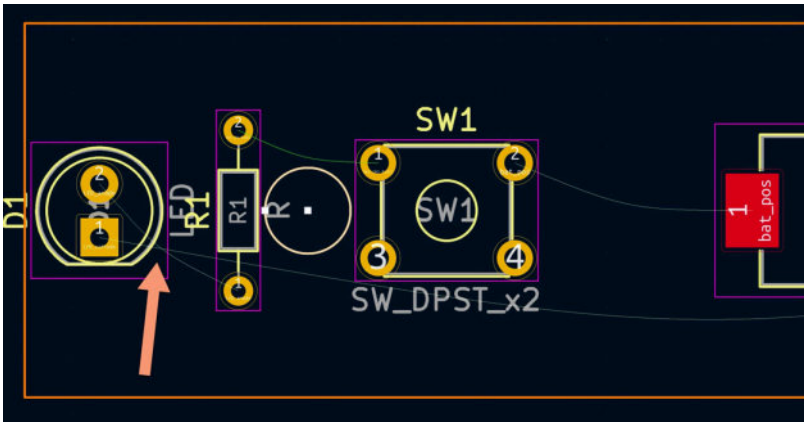


Figure 4.4.4: Circle graphic moved.

One last issue to resolve is the overlapping of two ratnest lines that come out of the diode pads. As I mentioned earlier, reducing the number of overlapping ratnest lines helps reduce the complexity of the route network in the next step of the workflow. I can easily remove the overlap in this example by rotating the LED footprint by 90 degrees (left-click the LED footprint to select it and type "R" twice). Here is the layout now (Figure 4.4.5):

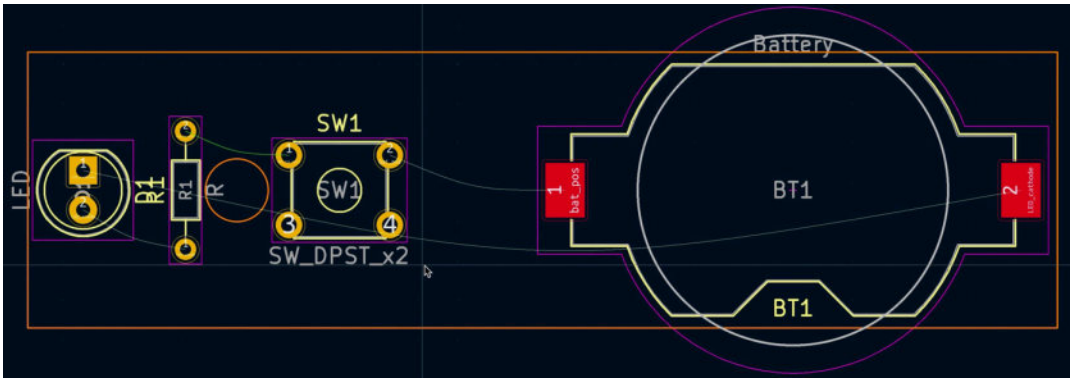


Figure 4.4.5: Final placement; no overlaps.

Is the placement of the footprints complete? Yes. I think that the footprints are where they should be.

Is there anything else I can do to improve this board before continuing with the routing step (followed by a final refinement of the board outline)? Yes. There is a bit of space between the battery holder and the button. I can improve my options of mounting the board in an enclosure by adding a second mounting hole in that space. Since I already have a mounting hole, I can quickly clone it and place it in the available space. You can clone a layout element, like a footprint, text label, or graphic, by selecting the element and typing Ctr-D or Cmd-D. The duplicated element will be attached to the cursor. Move the cursor to place the new element in its final position, and left-click to finish.

Here is the PCB now (Figure 4.4.6):

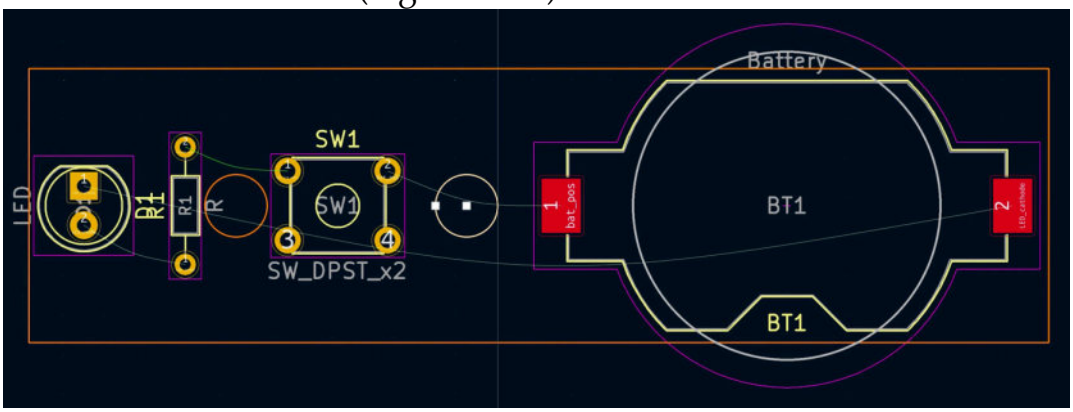


Figure 4.4.6: Final PCB in step three.

Click on Save, and bring up the 3D viewer to see a depiction of the PCB:

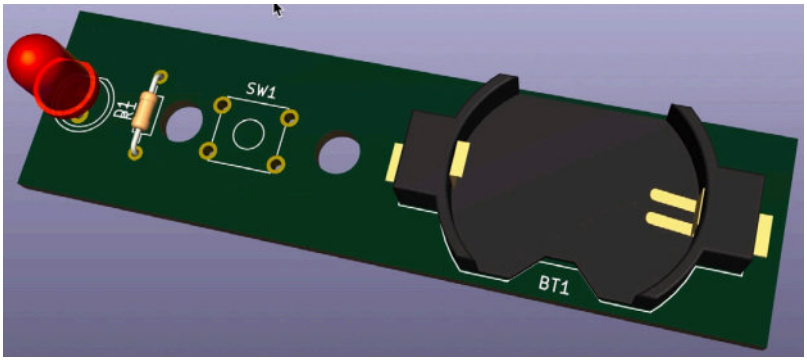


Figure 4.4.7: Final PCB in 3D, in step three.

With the footprints placed inside the PCB outline, I can continue with step four of the workflow and draw the routes.

5. 4 - Route (add tracks)

In this chapter, I will complete step four of the layout workflow, that you learned about in the second chapter of Part 3 of the book. In this step, I will complete the wiring ("routing") of the board. At present, the board looks like this:

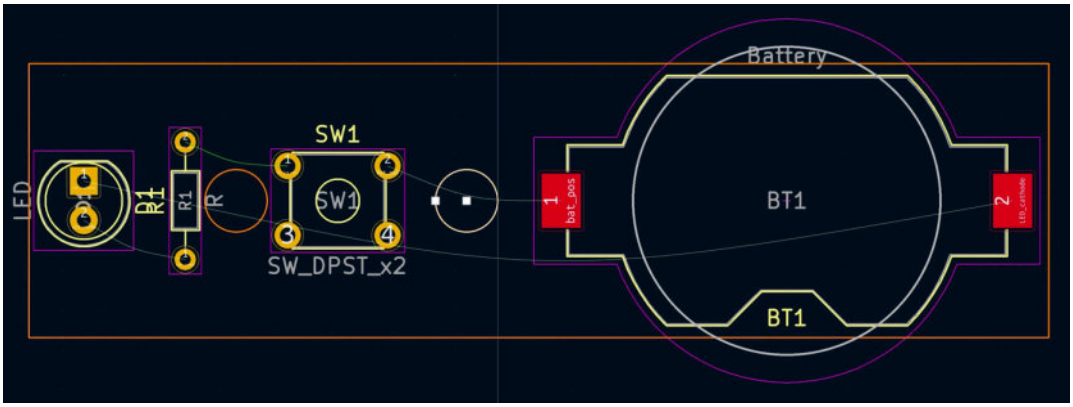


Figure 4.5.1: Final PCB in step three.

The thin ratnest lines will guide me through the routing process. The routing is complete when all ratnests have been replaced by copper routes.

Before I start with the drawing of the routes, I want to point out an interesting detail in the future above. Notice that all ratnest lines are grey-white, except for one that is green (between the R1 and SW1 footprints). This happened because, at some point, I experimented with net colors in the Appearance pane. See Figure 4.5.2 below:

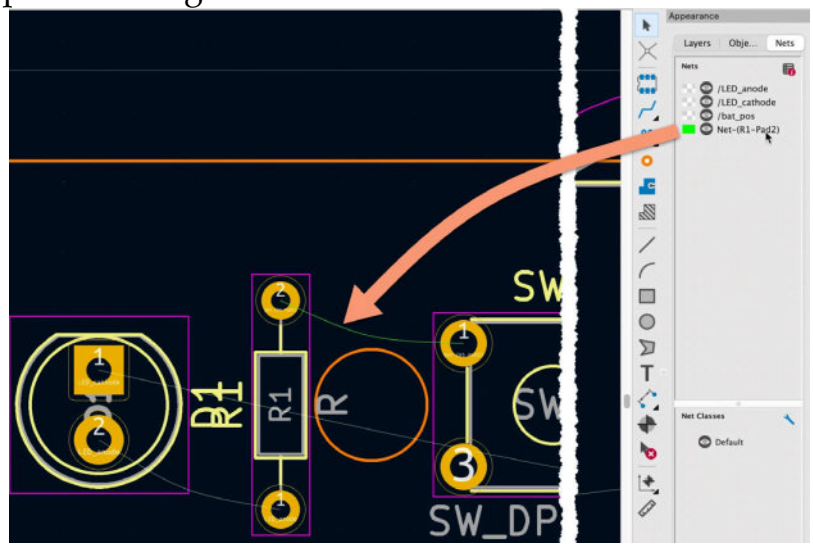


Figure 4.5.2: A green ratnest line.

In this example, I have created a custom color for ratnest lines that belong to a specific net. You can do this in the Appearance Nets tab; click on the color box and select a new color from the color chooser. Going forward, I will remove this customization so that all ratnest lines use the same color.

Let's continue with the routes. This is a simple board, so it is possible to draw all copper routes in a single layer, such as the front copper layer. By default, the layout editor provides a top and bottom copper layer. You can see this in the Layers tab, under Appearance (Figure 4.5.3):

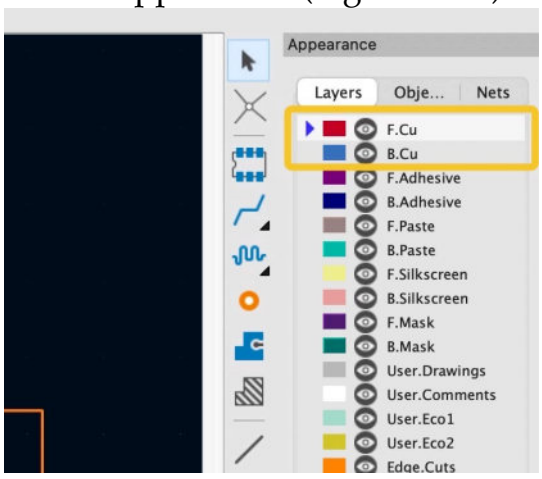


Figure 4.5.3: This board has two copper layers.

You can configure a different set of copper layers if you wish. Go to File, Board Setup, choose the Physical Stackup tab, and use the drop-down menu to select the number of copper layers for your board (Figure 4.5.4).

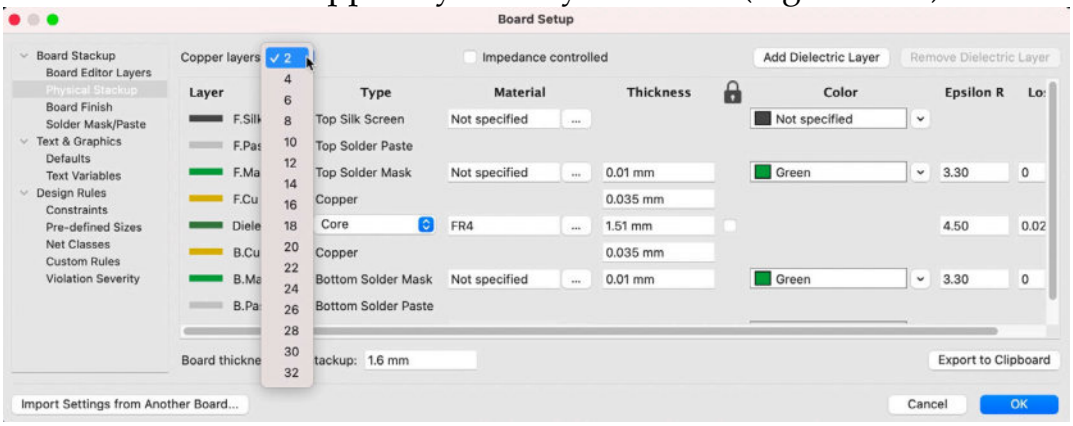


Figure 4.5.4: Set the board copper layers.

In the same tab, you can set other aspects of the board's layers. I will accept all the defaults for this project and draw the routes in the front copper layer. As you can see in Figure 4.5.3, the "F.Cu" layer is already selected so I can start drawing.

Select the “Route tracks tool” by clicking on its button in the right toolbar (Figure 4.5.5), or use the “X” hotkey.

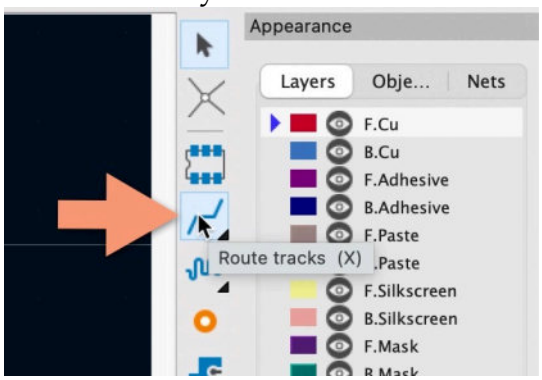


Figure 4.5.5: Select the “Route tracks” tool.

The cursor becomes a pen. To start drawing a route, click on a pad (or anywhere in the editor). To finish drawing, again click on a pad or double-click anywhere in the editor. The advantage of starting the drawing process from a pad is that the ratnest line will guide you to the route’s destination. I will start drawing from pad 2 of R1 (Figure 4.5.6).

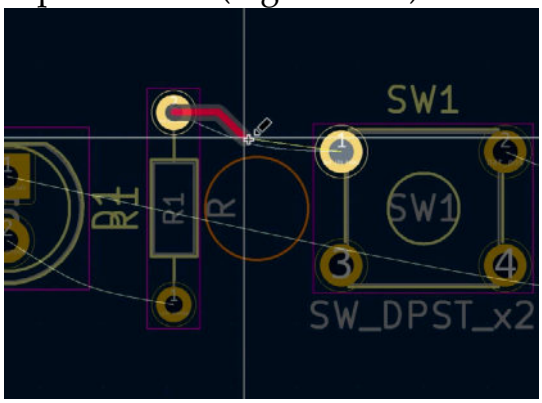


Figure 4.5.6: Drawing the first route.

As you are drawing a route, you can move your mouse pointer and see how the route follows the pointer attached to it. You can define the geometry of the route, such as its corners and angles, by clicking. When you move the mouse pointer over the destination pad, you will notice the snap effect (snap-to-grid). When this happens, left-click to finish drawing this route. Below you can see the first route completed (Figure 4.5.7).

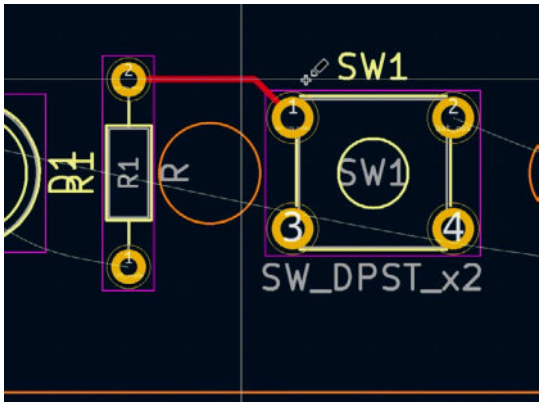


Figure 4.5.7: the first route is complete.

Follow the same process to fully route the board, replacing each ratnest line with a track. Here is my fully routed board at this time (Figure 4.5.8):

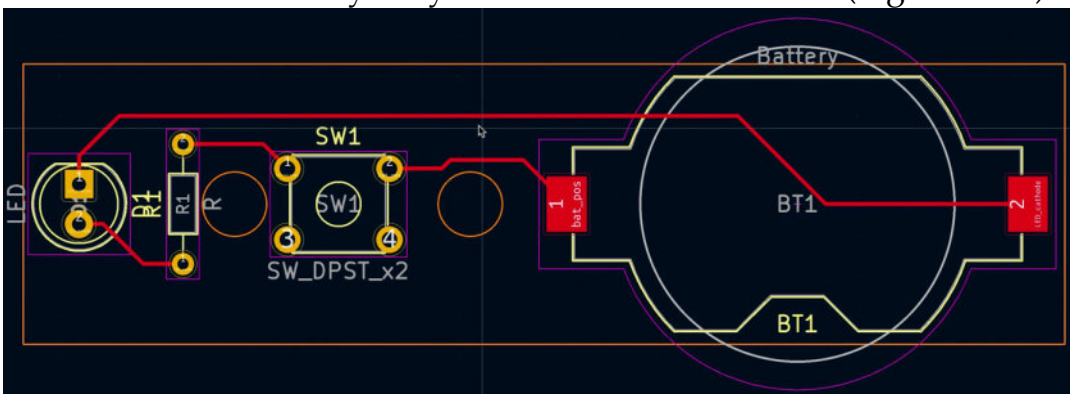


Figure 4.5.8: The fully routed PCB.

Once you have drawn a route, it is possible to modify it by dragging its component segments. Of course, you can also delete it and re-draw it. To select a track, ensure that the "Tracks" checkbox is selected in the Selection Filter. Then, click on a track segment to select and use the "D" (Drag, 45-degree angle) or "G" (Drag, free-angle) hotkeys to choose the type of move option you want. Use your mouse to move the segment. Try these two move options now to get a feel for them. These options are part of the interactive router, and you can learn more about them in the relevant recipe chapter.

At this point, the board is fully routed, and this step is complete. I have the habit of doing a DRC (Design Rules Check) to make sure I have not forgotten anything (Figure 4.5.9):

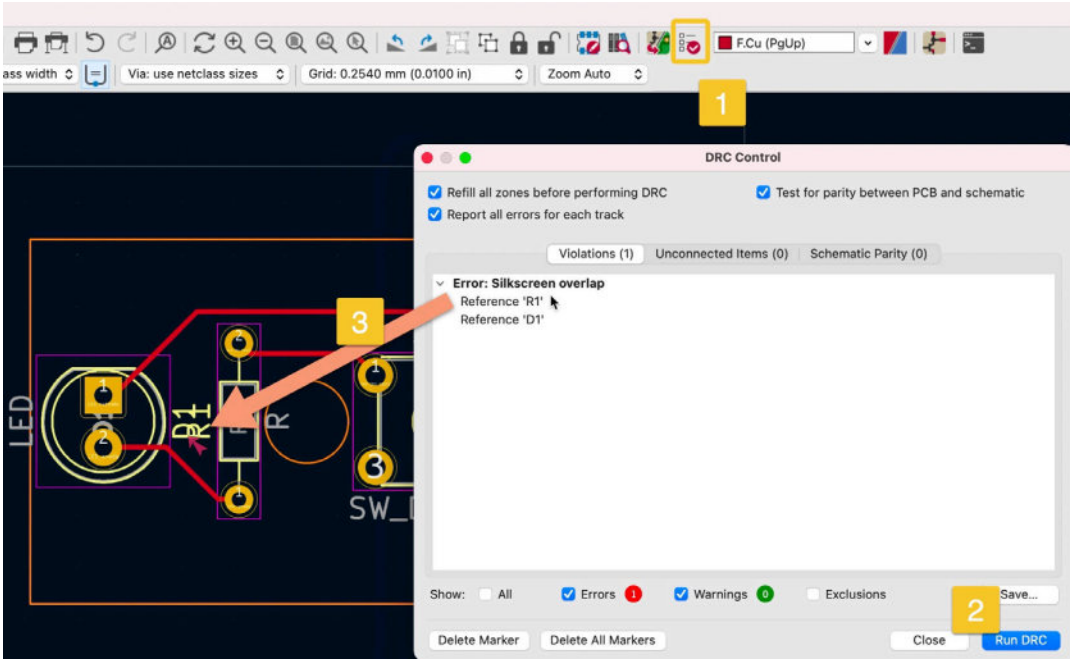


Figure 4.5.9: The DRC shows one error; I will fix this later.

Bring up the DRC window by clicking on the DRC button in the top toolbar (“1”), and then click on “Run DRC” (“2”). The DRC shows one error that relates to an overlap between two silkscreens. This is not a routing error, and I can fix it later in the workflow.

With the routing complete, I will continue in the next chapter by going back to step two to refine the board's outline.

6. 5 - Refine the outline

In this chapter, I will go back to step two of the layout workflow. In this step, I will refine the board outline as now all the footprints are in place, and I don't rely on a provisional set of geometry assumptions. At present, the board looks like this:

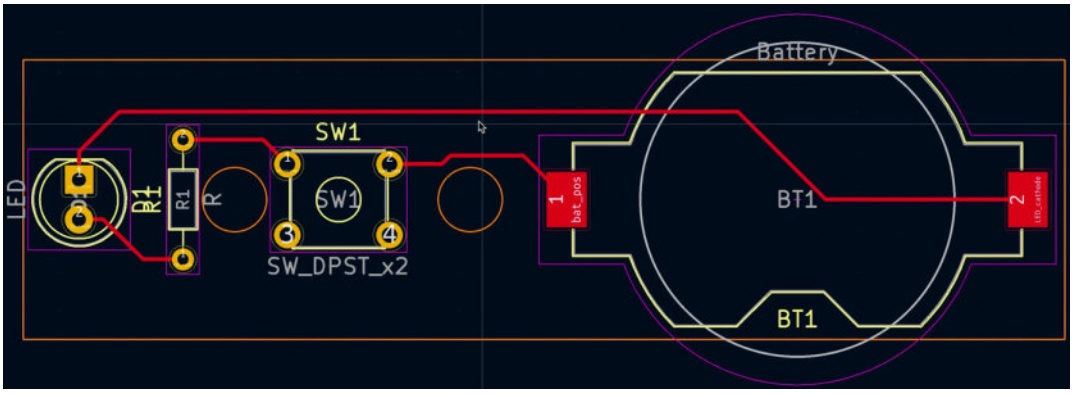


Figure 4.6.1: Final PCB in step four.

At the moment, the PCB outline consists of four straight lines with ninety-degree angles between them. I have drawn the lines in the Edge.Cuts layer. I will make changes that implement the following:

1. Reduce the total size of the board (width and height) to a minimum.
2. Replace the angled corners and rounded corners.
3. Enclose the battery footprint within the PCB outline using two rounded segments for the top and bottom parts of the footprint.

Start the refinement work by selecting the Edge.Cuts layer from the Layers tab of the Appearance pane. In the Selection filter, check the Graphics checkbox (Figure 4.6.2).

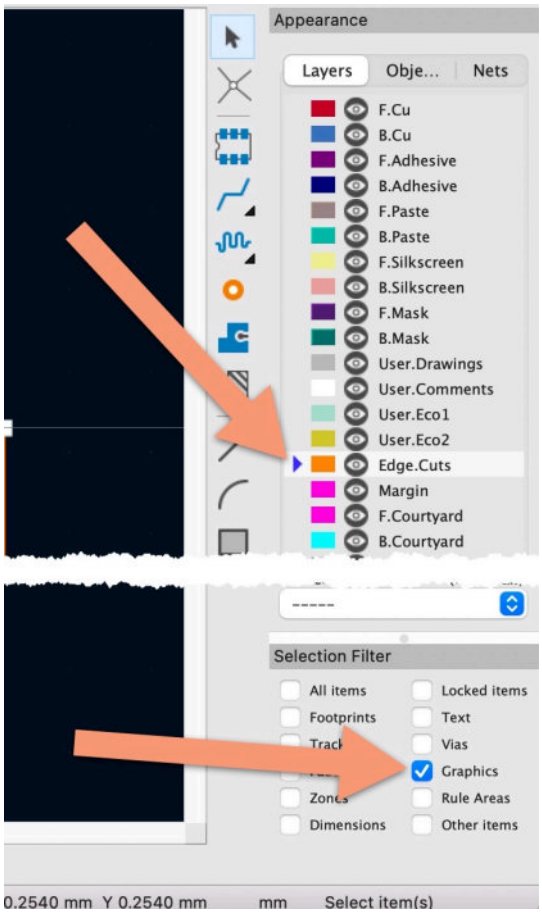


Figure 4.6.2: Continue the graphics work in the Edge.Cuts layer.

To reduce the size of the board, I will move the top and bottom edges of the outline inwards. Since I am working with individual lines (not a box), I can click each line to select it, then click and hold to move it to its new position (Figure 4.6.3).

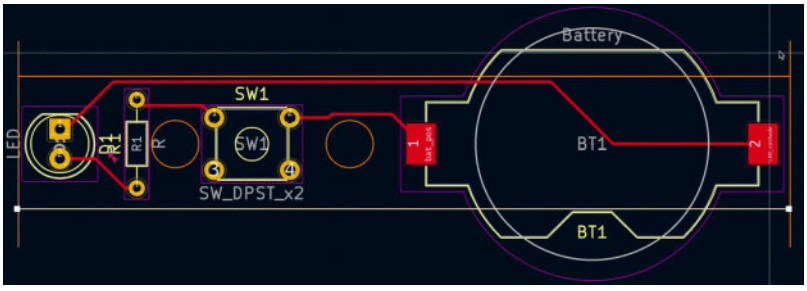


Figure 4.6.3: Reducing the height of the PCB.

The two vertical lines are still in their original length. To resize them, first, I will click anywhere in the right line to select it. This will turn on the handles at either end of the line. Then, I will click on one handle and hold, drag the handle and drop it to the end of the nearest horizontal line (Figure 4.6.4).

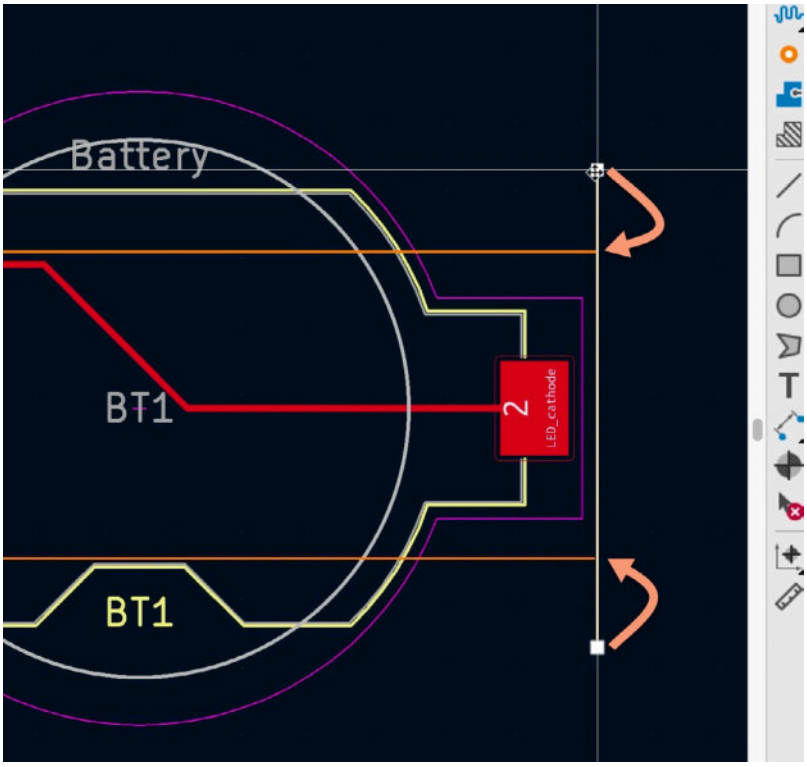


Figure 4.6.4: Resizing the vertical lines.

Repeat the process on the left side. Instead of trying to adjust the existing lines, you can also simply delete them and re-draw them. Either way, the board now looks like this (Figure 4.6.5):

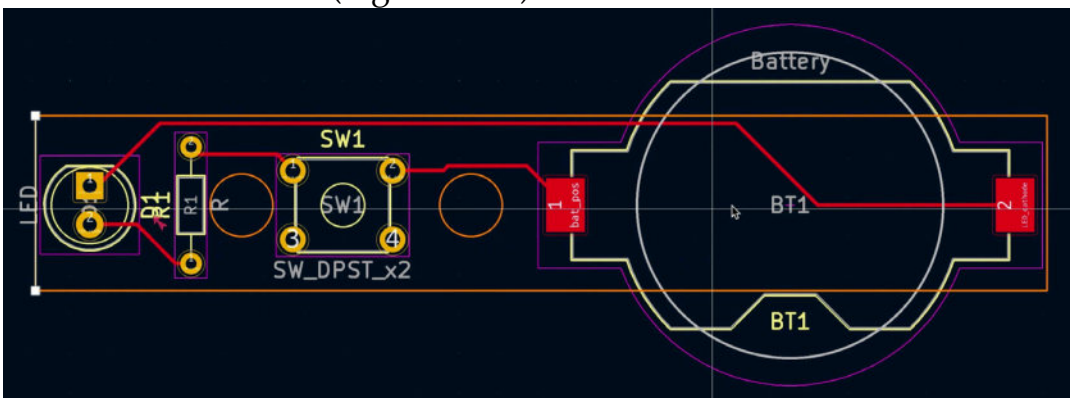


Figure 4.6.5: PCB outline with reduced height.

I will continue with the second item in my improvements list. First, at the left end of the PCB, I will replace the entire left edge of the board with a semicircle.

I start by deleting the left vertical line (even though I only drew it a few minutes ago). Select the arc tool from the right toolbar, and place your cursor in the middle of the LED footprint. Click to start drawing the arc (“1”). The first left-click defines the centre of the arc. Place the cursor at the left end of the bottom line to intersect it, and click again (Figure 4.6.6). The second click

("2") defines the radius of the arc, and starts the drawing. Move the mouse clockwise to draw, until it meets the left end of the top line ("3"). When the arc and the top line meet, click again to finish the drawing. The arc is complete.

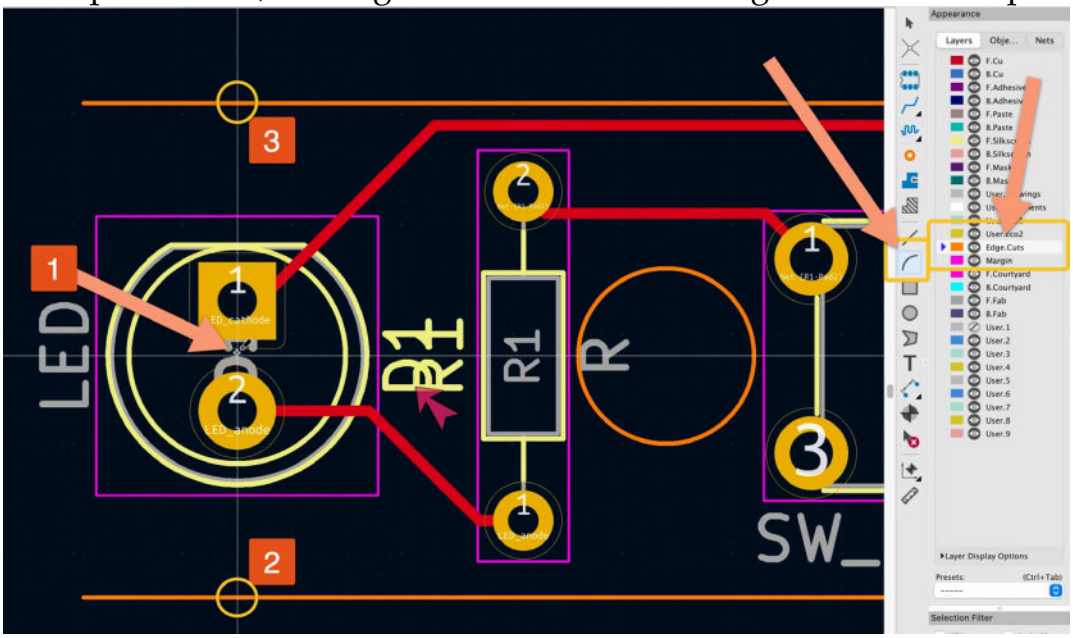


Figure 4.6.6: Drawing a semi-circle with the arc tool.

Because the position and alignment of the two horizontal lines in reference to the LED footprint and the arc line is not perfect, I could not connect the bottom end of the semicircle to the bottom horizontal line. To fix that, I select the bottom horizontal line and move it down so that it "touches" the end of the semicircle. In such cases, you will need to "play" with the position of the lines involved and the grid and grid size so that you can perfectly connect the outline segments. The layout editor uses a small circle line to confirm when two points overlap precisely. When I see this circle, I click to finish the drawing.

Repeat the process to align the top line with the top end of the semicircle. The top and bottom end of the semicircle is now perfectly connected to the top and bottom horizontal lines (Figure 4.6.7):

Figure 4.6.7: Semicircle joins the rest of the outline.

The left end of the PCB now looks like this:

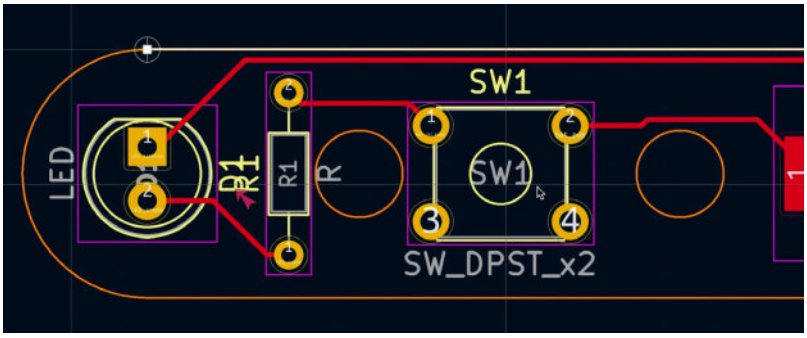


Figure 4.6.8: The left end of the PCB.

Next, I will draw two circular segments around the battery holder. I will need to be more careful with my drawing precision here; before making any changes in the Edge.Cuts layer, I will use the User.2 layer to draw a guide circle. This circle will help me draw the correct arc in the Edge.Cuts layer.

Select the User.2 layer, and then select the circle tool. Place the mouse in the middle of the battery holder footprint, and click to start drawing a circle. You can see the blue circle below:

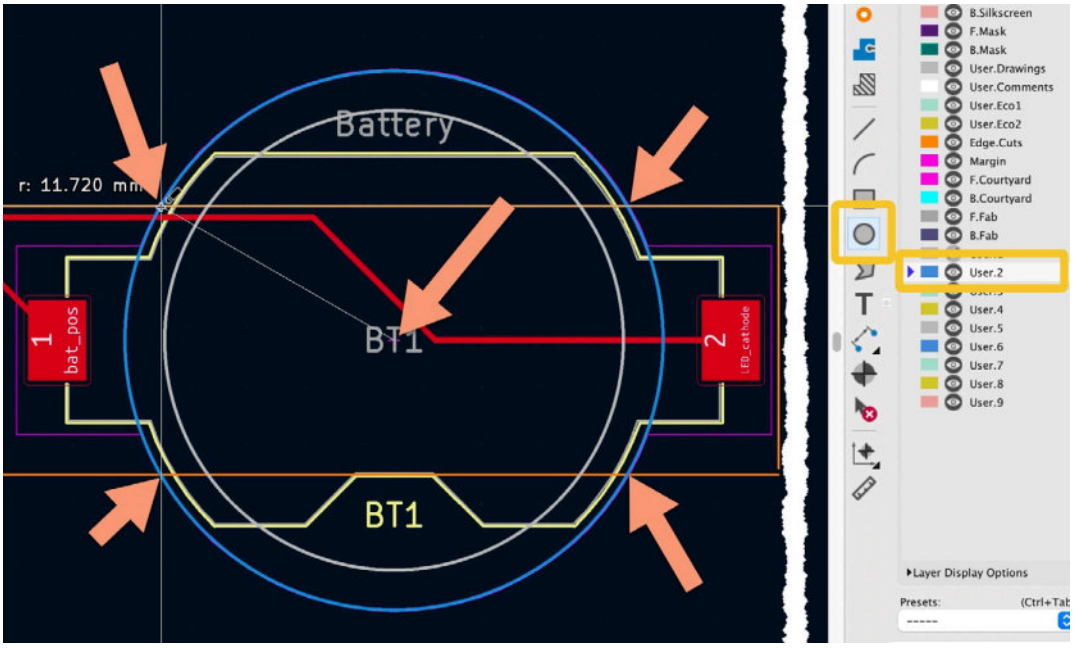


Figure 4.6.9: Drawing a circle in User.2.

I have drawn the blue circle to contain the battery holder footprint fully and intersect with the existing Edge.Cuts lines. In the figure above, the arrows show the centre of the footprint where I started drawing the circle and the points where the circle intersects the Edge.Cuts lines.

Now switch to the Edge.Cuts layer and select the arc tool. Click in the middle of the battery holder footprint to start drawing an arc (Figure 4.6.10).

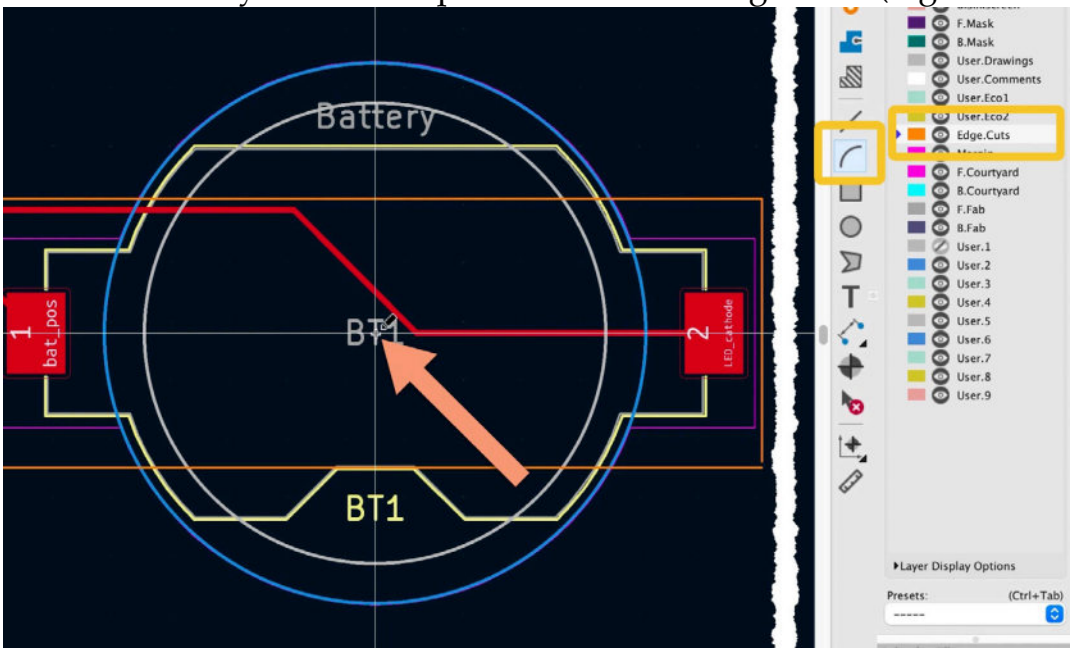


Figure 4.6.10: Drawing the first arc.

Next, (refer to Figure 4.6.11 below) move the cursor towards the location where the blue circle intersects the edge cuts line on the left side of the battery holder ("1"). Click when you see a small circle that indicates the intersect. Continue to draw the arc towards the left side until the arc intersects the edge cuts line on the right side ("2"). I remind you to take care and click only when you see the small intersection circle, like in the example in image two below.

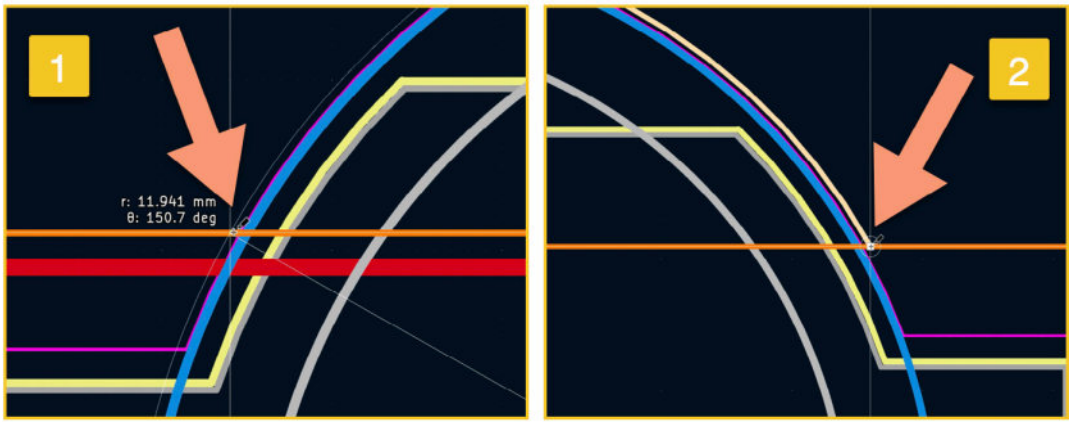


Figure 4.6.11: Drawing the first arc (continuing).

The arc at the top of the battery holder is complete and looks like this:

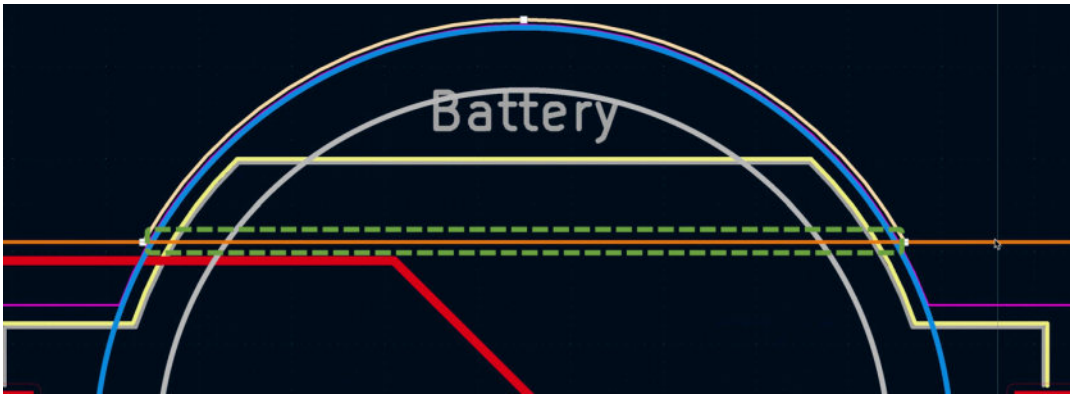


Figure 4.6.12: The first arc is complete.

There is still work left to do at the top side of the battery holder before continuing with the bottom side. Now that I have finished drawing the arc, I have to remove the orange line that defined the original outline for the battery holder as it is redundant. It can cause problems for both KiCad's 3D viewer and the PCB manufacturer. I have marked the line segment that I must remove with the green dotted box in Figure 4.6.12. There are a few ways you can go about doing this. I will finish the work here by resizing the existing line and adding a new line. Concerning Figure 4.6.13, I click on the line to reveal the handle on the right side ("1") and use the handle to resize the line. I attach the right end of the line to the left end of the top arc ("2"). Using the line tool, I create a new line that starts at the right end of the arc ("3") and connects with the top end of the right vertical line ("4").

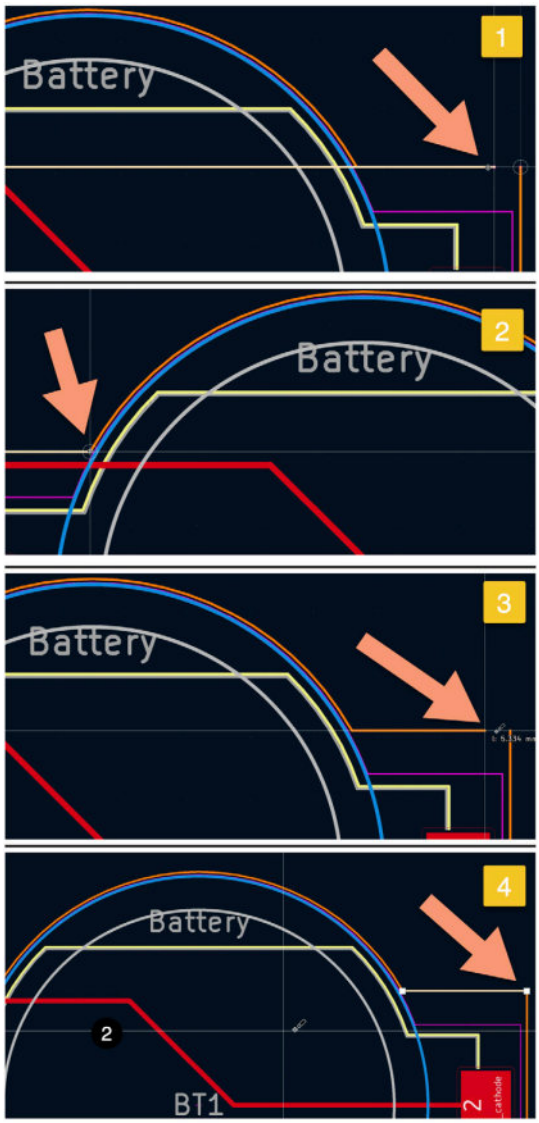


Figure 4.6.13: Removing the redundant line.

Repeat the same process for the second arc. By the end of this process, the battery holder footprint is fully enclosed in the edge cuts outline (I have disabled the User.2 layer to remove the blue circle line):

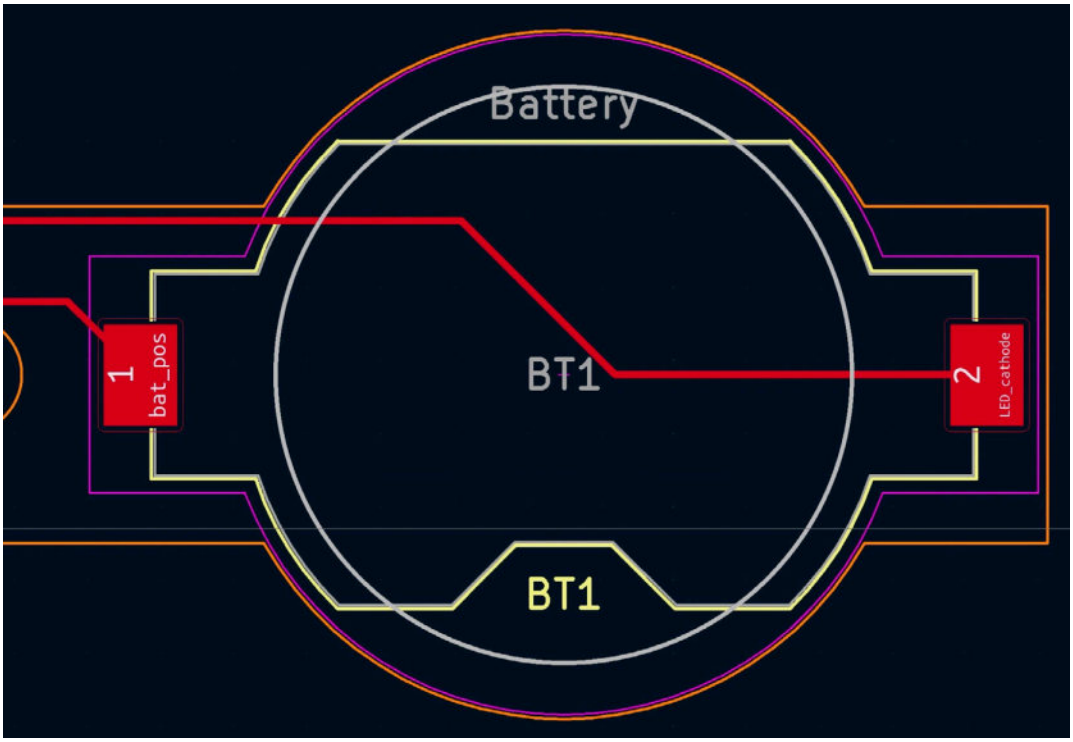


Figure 4.6.14: The battery holder footprint is fully enclosed in the PCB outline.

I will continue with the right side, where I want to replace the 90-degree corners with rounded corners. For this, I will use (again) the arc tool. To help me with the process, I have set the grid size to 0.254 mm and will use the dx and dy values in the status bar as a guide.

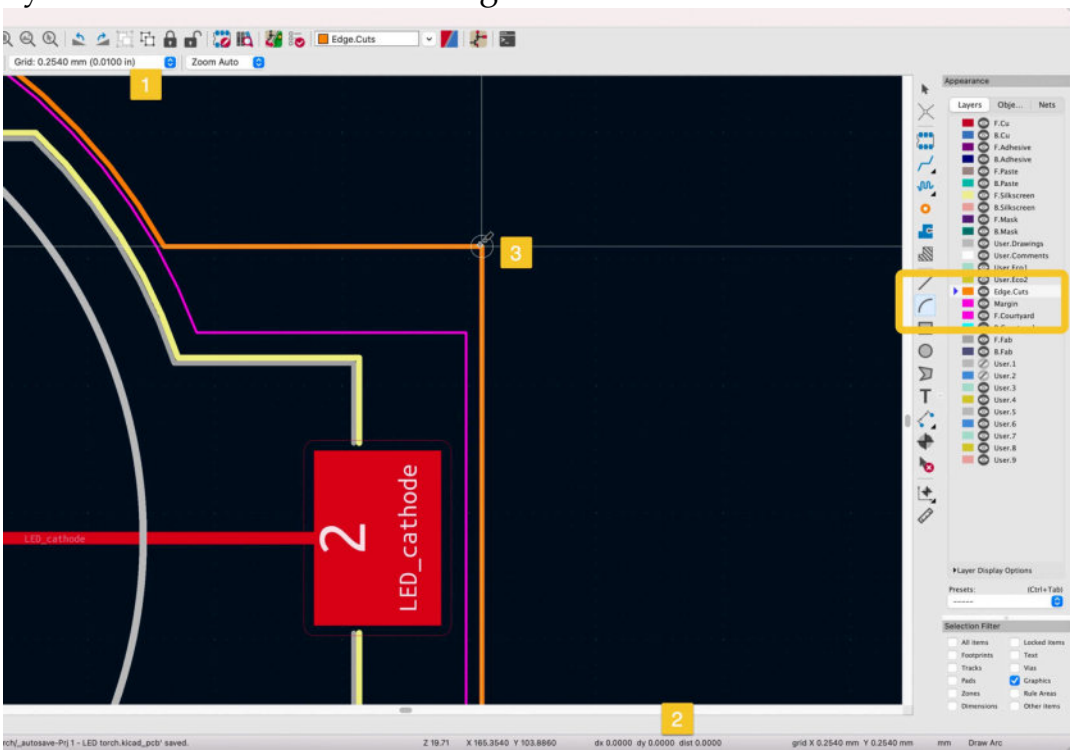


Figure 4.6.15: Replacing the corner with an arc.

I continue with the top-right right corner of the outline. In Figure 4.6.15, notice that I have selected the Edge.Cuts layer and the arc tool. I have set the grid to 0.254 mm (“1”). Keeping an eye on the dx/dy values of the status bar (“2”), I place the cursor on the corner and press the space bar. The space bar press will reset the dx/dy distance counters to zero. Move the mouse pointer diagonal down and left so that dx and dy are both showing 1.27 mm, as showing in Figure 4.6.16 (it does not matter whether negative or positive).

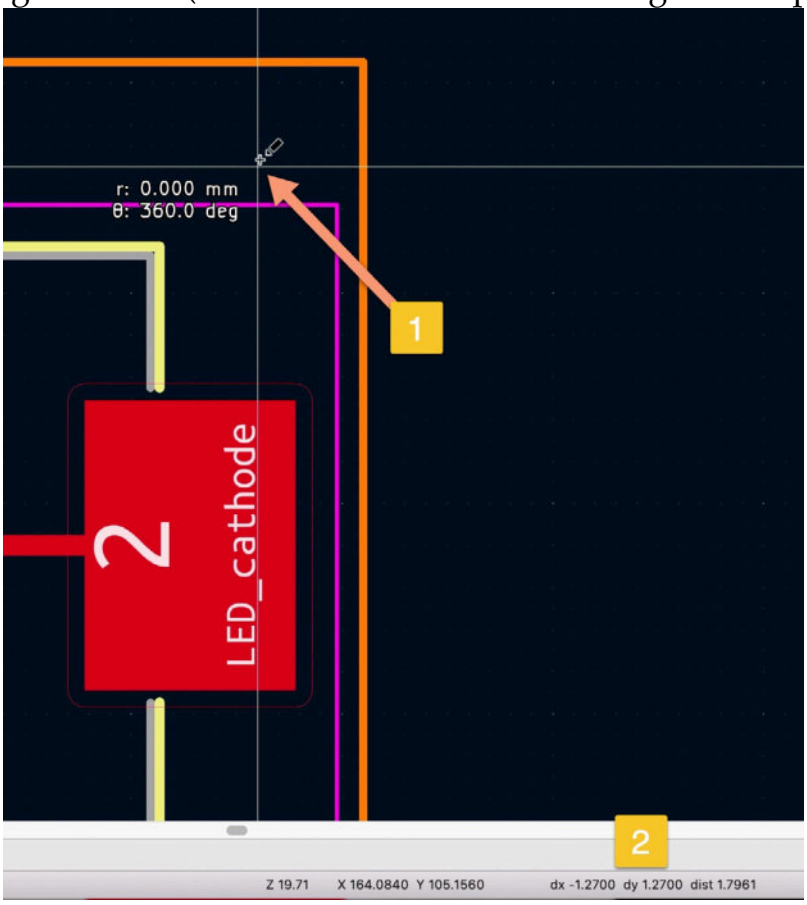


Figure 4.6.16: Placing the pointer to begin drawing the arc.

This position and distance from the corner will produce the arc that I want. Click to start drawing the arc, then move the mouse to touch the horizontal line and click again to define the radius. Then drag toward the right and down to intersect the vertical line and click again to finish the drawing.

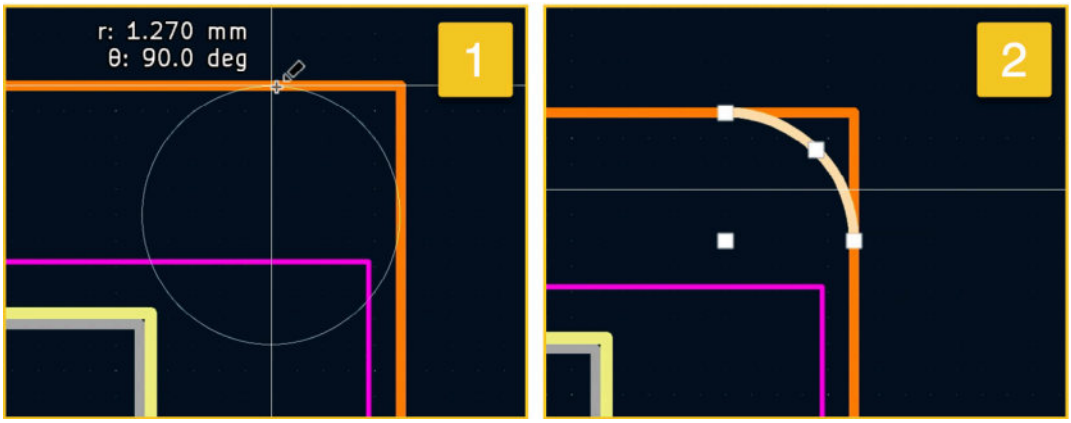


Figure 4.6.17: Draw the top corner arc.

I now have the arc. I will finish this operation by editing the horizontal and vertical lines. As I did with the battery footprint semicircles, I click on a line to reveal its handle, then use the handle to drag the line end on the arc ends. The editor will help me by showing the small alignment circle when the two endpoints meet. The result is in Figure 4.6.18 (below):

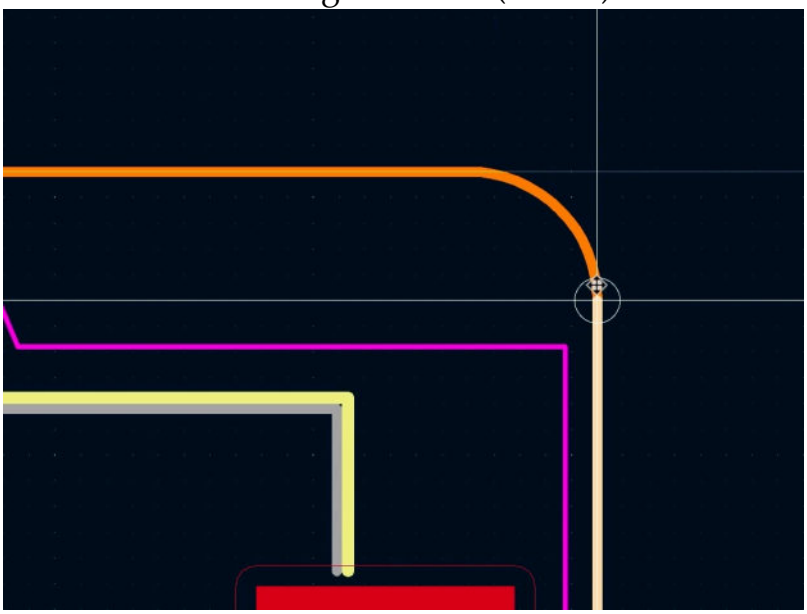


Figure 4.6.18: Arc is complete.

Repeat the same process to replace the bottom right corner with an arc. After this work, the entire PCB will look like this:

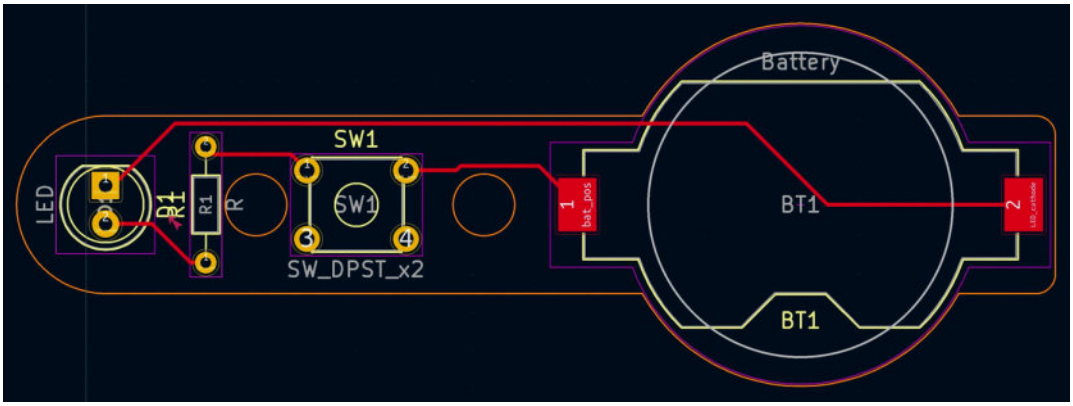


Figure 4.6.19: The refined PCB outline.

And this is the board's 3D rendering:

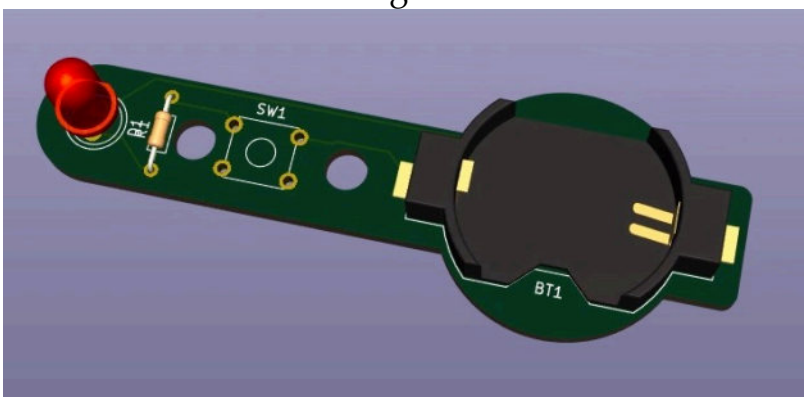


Figure 4.6.20: The refined PCB in 3D.

The second iteration of the second step of the workflow is now complete. I will continue to step five and work on the silk screen text and graphics.

7. 6 - Silkscreen (text and graphics)

In this chapter, I will complete step five of the PCB layout workflow, that you learned about in the second chapter of Part 3 of the book. In this step, I will add informative and decorative text and graphics in the front and back silkscreen layers. For example, I will use silkscreen text to print the name of the various components on the PCB to help the end-user with the assembly and the version number of the layout to differentiate between potential new versions of this PCB. I also like adding decorative logos, such as the "Designed with KiCad" logo.

As I left it in the previous chapter, the layout already has both graphics and text on the front silkscreen. These elements were inherited from the footprints of the various components.

In Figure 4.7.1 (below)), the arrows show a text label and a polygon-arc composite line in the front silkscreen layer. These elements are part of the battery holder footprint. In the same figure, I have used a yellow box to mark the "F.Silkscreen" and "B.Silkscreen" layers that I will be working on in this chapter and the tools that are available to use. Once I enable one of the silkscreen layers, I will use these tools to add new text labels or other graphics.

It is also possible to change the layer where many of the other elements exist. For example, in the figure below, it is possible to change the layer of the "Battery" text item from the current "User.Drawings" layer to one of the silkscreen or even copper layers.

If you wish to learn more about creating and using logos, please refer to the dedicated chapter in the recipes part of the book.

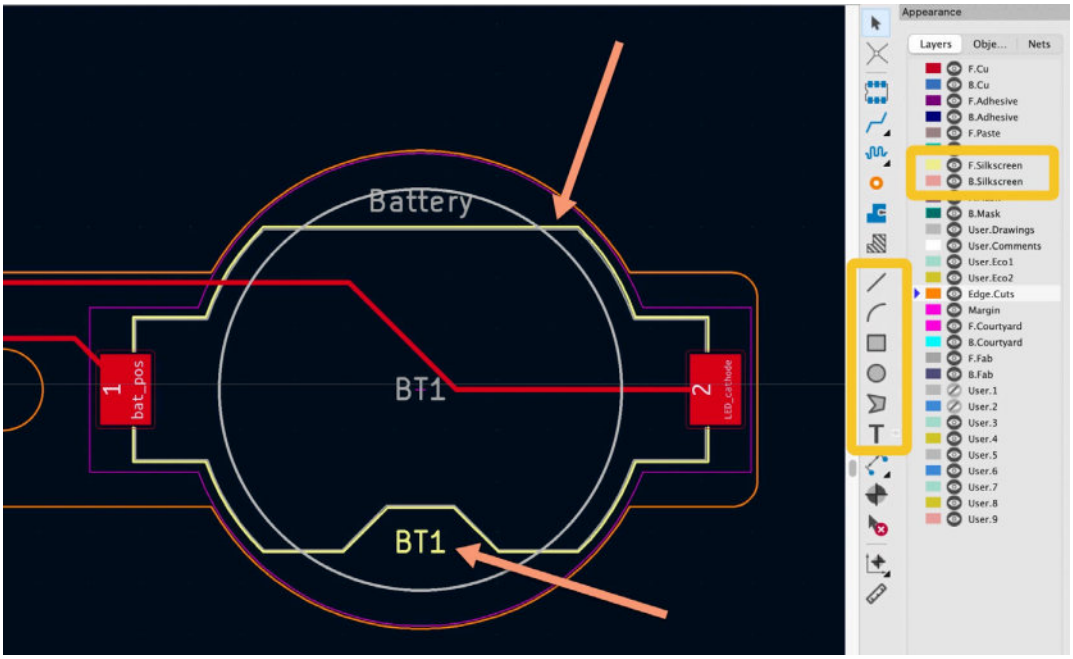


Figure 4.7.1: Existing elements in the front silkscreen and silkscreen tools.

I will start by adding or editing a few items in the front silkscreen. First, I will enable the "Text" item in the Selection Filter. At the left side of the PCB are the "R1" and "D1" text labels. They are yellow, which is the color assigned to items in the front silkscreen layer (confirm this by looking at the "F.Silkscreen" color in the Appearances pane). I will move these items so that they are not overlapping. See their final positions in the figure below:

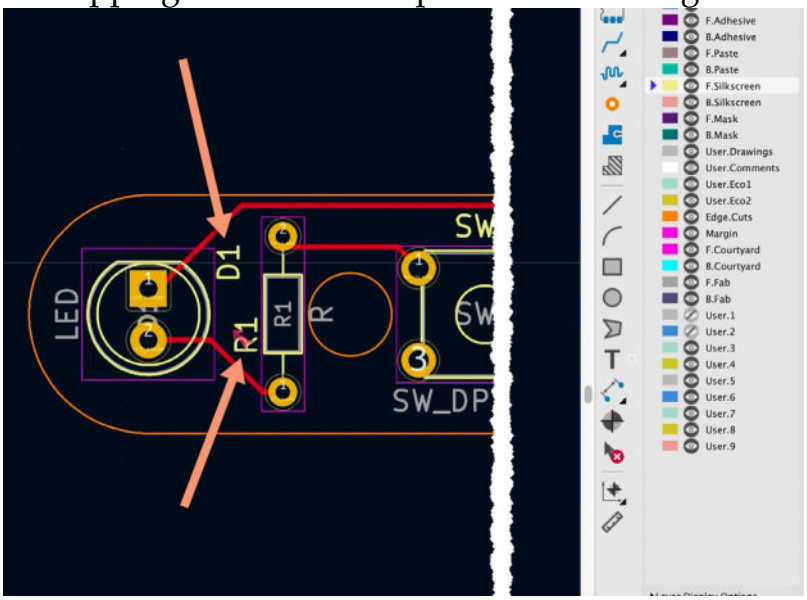


Figure 4.7.2: The final positions for the "D1" and "R1" text labels.

The new positions of the "R1" and "D1" labels will also resolve the DRC issue that I discovered at the end of the routing step chapter.

I will change the layer of the “LED” text label to “F.Silkscreen”. I can do this via the label’s Properties window. Double-click to bring up the window, and use the Layer dropdown to select the new layer (see Figure 4.7.3 below):

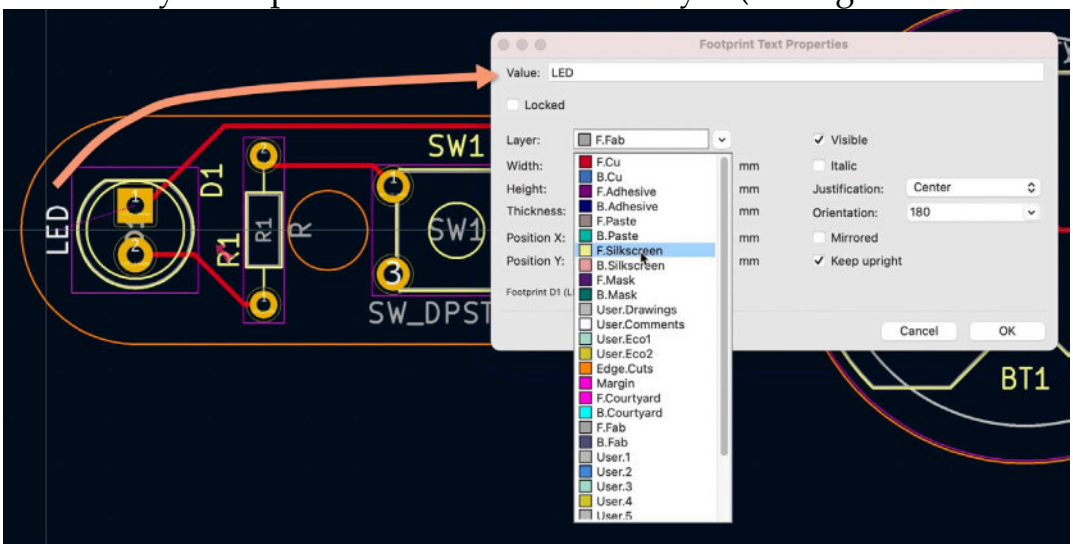


Figure 4.7.3: Changing the assigned layer of an element.

Click OK to commit the change and notice that the “LED” label is now yellow.

To see what will be printed on the silkscreen, you can also use the 3D viewer. Bring up the 3D rendering of the board (View —> 3D Viewer):

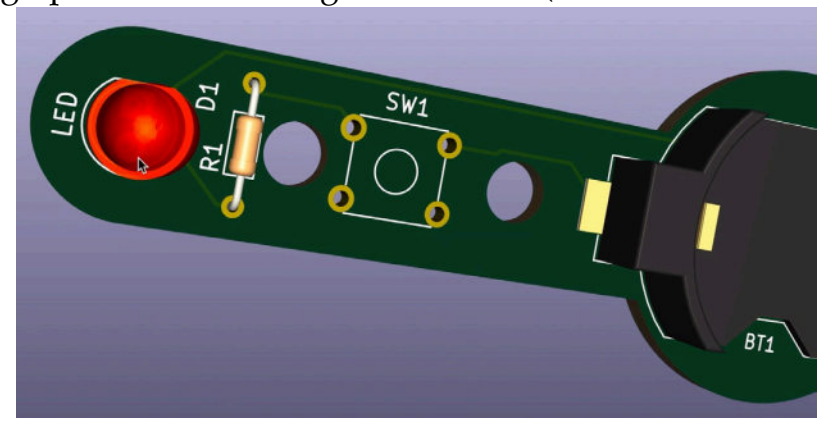


Figure 4.7.4: A 3D rendering of the PCB shows the silkscreen graphics.

As you can see above, the labels “LED”, “D1”, R1”, “SW1”, and “BT1” appear in the front silkscreen, along with other footprint elements.

Go ahead and change the layer for the “Battery” label (which is currently in “F.Fab” to the front silkscreen. This concludes the silkscreen work for the front of the board. The board now looks like this:

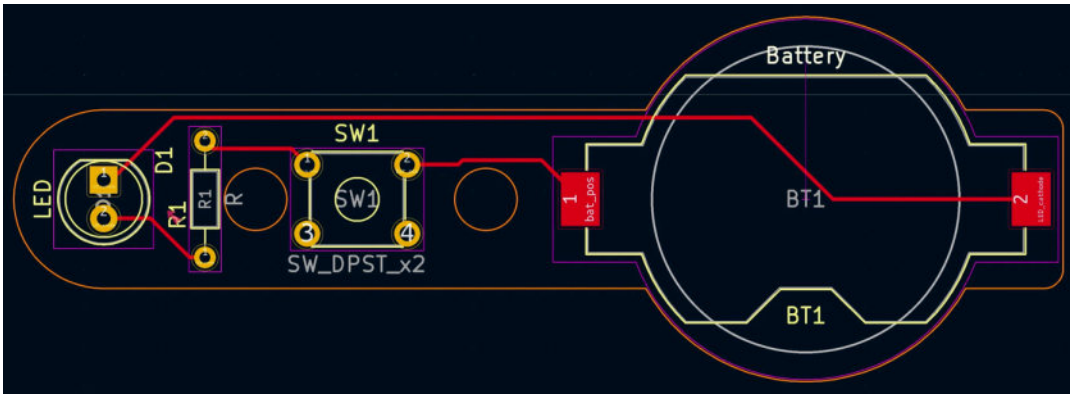


Figure 4.7.5: Front silkscreen work is complete.

In the back silkscreen layer, I will add a KiCad logo and the version number of the board. Logos and similar graphics are treated as footprints that only have information in their silkscreen or copper layers. Therefore, you can find such graphics footprints in the footprint libraries. In the case of the KiCad logo, there are several choices you can make.

Click on the Footprint button from the right toolbar to bring up the footprint chooser ("1" in Figure 4.7.6 below).

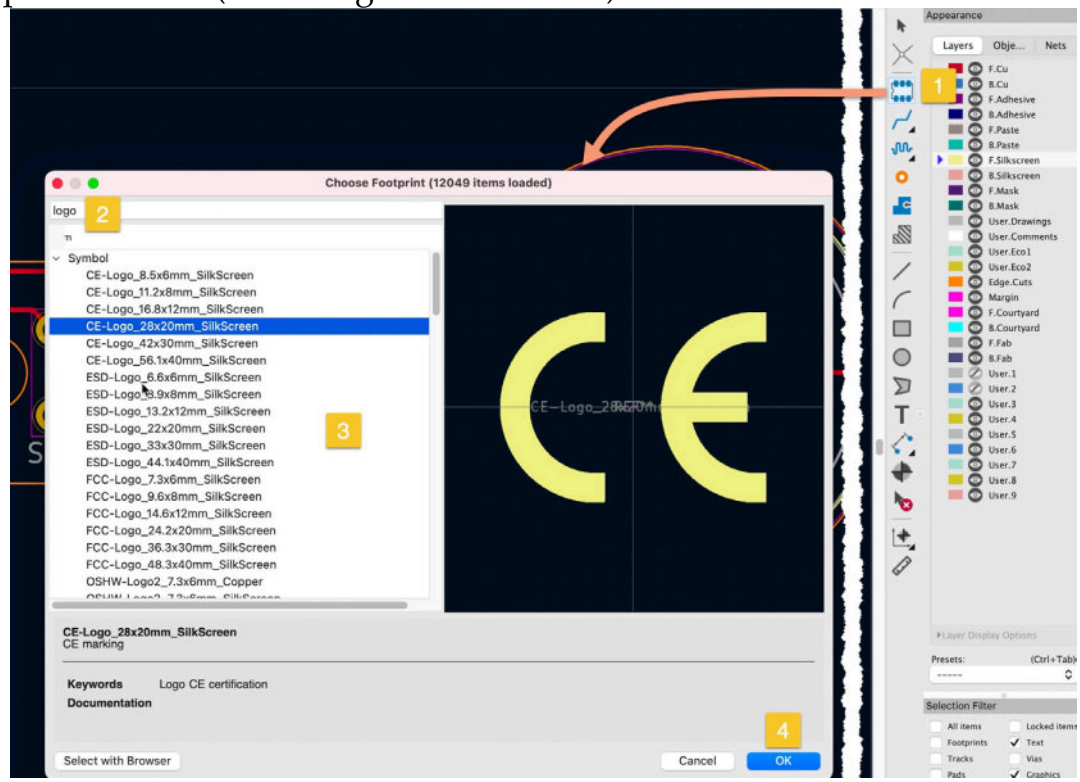


Figure 4.7.6: Find a logo in the footprint chooser.

Type the name (or part of) of the footprint you want to find in the search box. I typed "logo". Among the libraries that KiCad ships with are the "Symbol" library. It contains an extensive collection of logos, such as "CE", "ESD", "OSH" and, of course, "KiCad". I will choose the footprint named

“KiCad-Logo2_40mm_Silkscreen”. Double-click to select it and add it to the board. The logo is now in the editor:

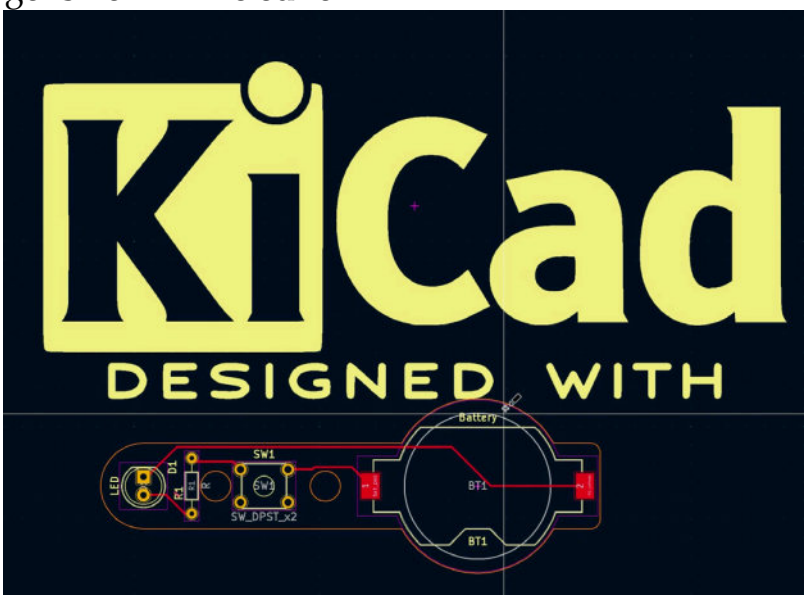


Figure 4.7.7: This logo is too large for the board.

Unfortunately, I did not realize that the logo is much larger than the board. So, I’ll delete it and return to the footprint chooser to look for something smaller. The “KiCad-Logo2_8mm_Silkscreen” should be a better fit. Go ahead and add it to the board. The board now looks like this:

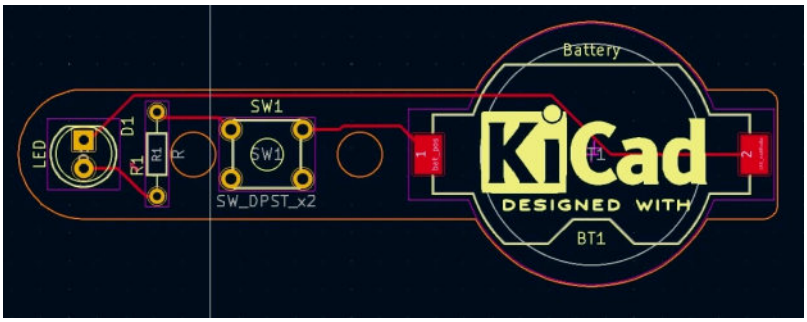


Figure 4.7.8: This logo is a good fit for the board.

This logo is a good fit for the board. By default, it appears in the front silkscreen layer. To switch the logo layer to the back silkscreen, double click to bring up the footprint properties (Figure 4.7.9).

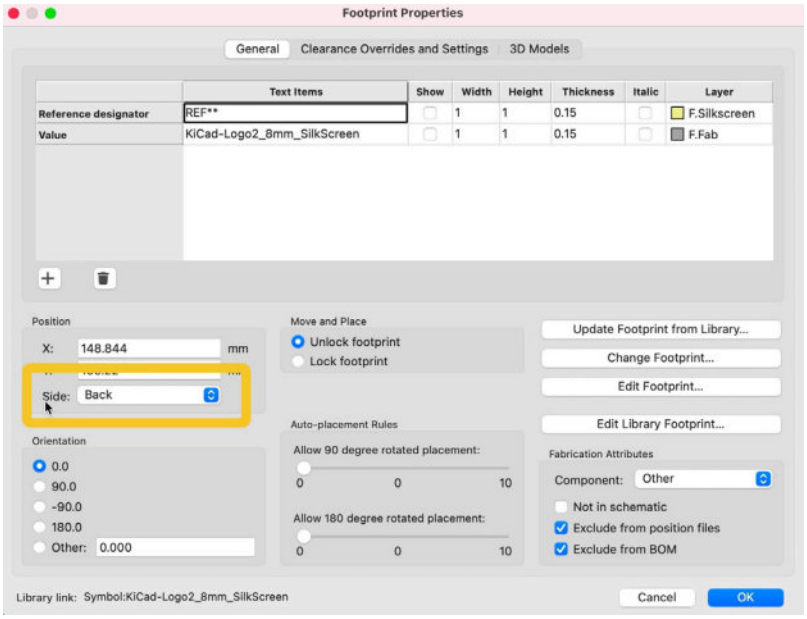


Figure 4.7.9: Switch the logo to the back silkscreen.

In the properties window, select “Back” from the “Side” dropdown and click OK. The board now looks like this:

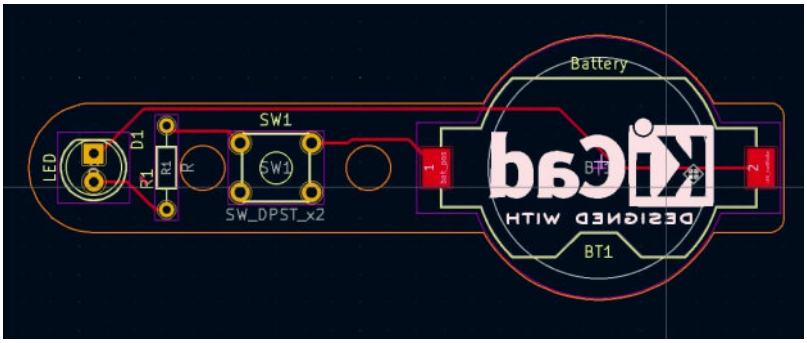


Figure 4.7.10: This logo is the back silkscreen layer.

The logo is now in the back silkscreen layer.

The last text item to add is the version of the board. Click on the text tool from the right toolbar and click just below the logo to bring up the Text Properties window. In the text field, type “V1.0”. In the Layer dropdown, select “B.Silkscreen” and check the Mirrored checkbox.

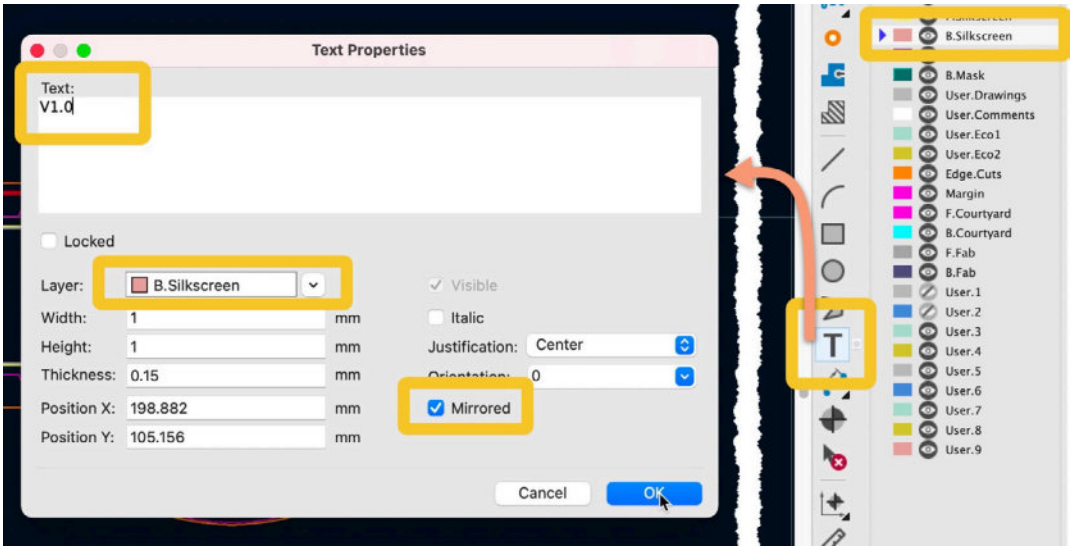


Figure 4.7.10: Adding a new text label in the back silkscreen.

Click OK to dismiss the properties window. The new text label is attached to the cursor, so move the cursor to an appropriate position and click again to finish the placement. I have positioned the label just above the logo:

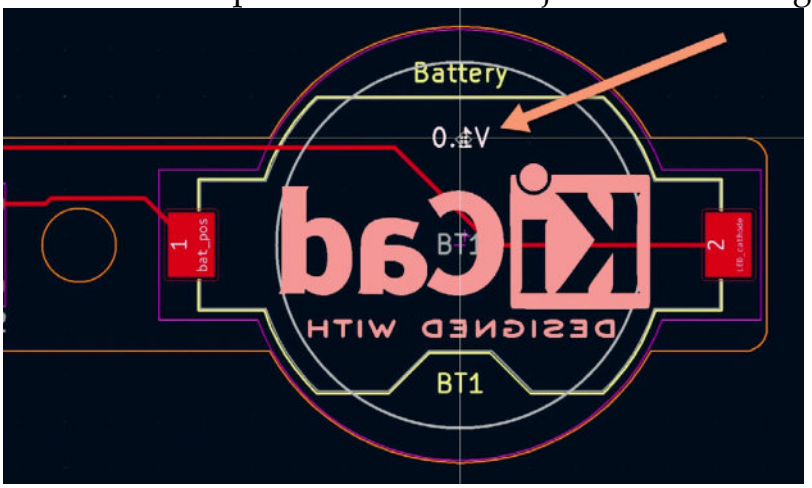


Figure 4.7.11: Added a version number in the back silkscreen.

Here is the 3D rendering of the back of the board:

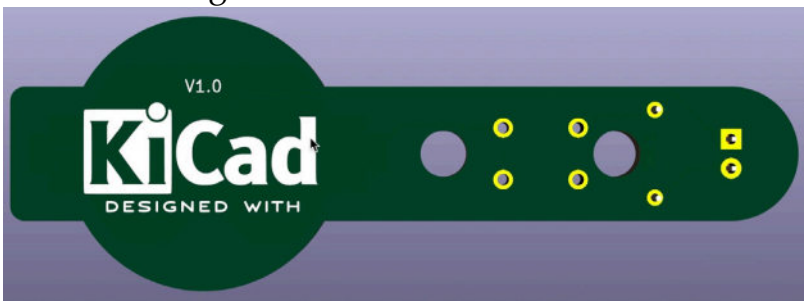


Figure 4.7.12: 3D rendering of the back of the board.

Before finishing work in this step, I will also add a few more text items to help me with the assembly of the board:

- A "-" next to the LED cathode pad.

- A “+“ next to the LED anode pad.

  Below is the PCB as it appears at the end of step five of the workflow:

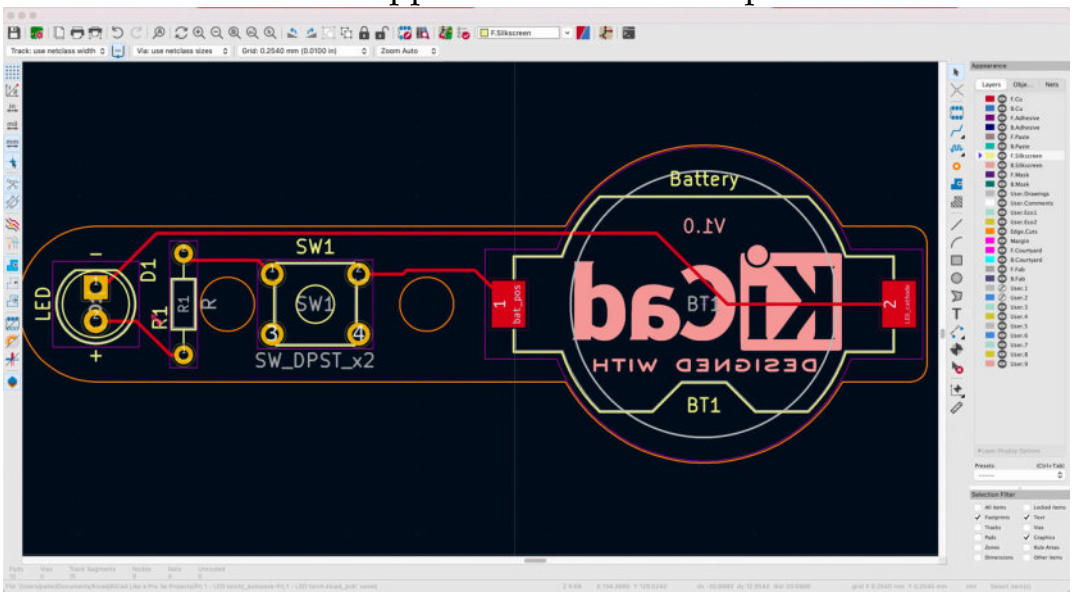


Figure 4.7.13: The PCB at the end of step five of the workflow.

This project is nearing completion. There are only two steps left. In the next chapter, I will complete the final design rules check, and then I will export the Gerber files to have the board manufactured.

8. 7 - Design rules check

In this chapter, I will complete step six of the PCB layout workflow, that you learned about in the second chapter of Part 3 of the book. Although I conduct frequent design rules checks throughout the layout workflow, especially during the routing step, I always run a final check before exporting the Gerber files for manufacturing (next step).

In this simple project, I already run the DRC in step four. During that check, the DRC revealed an issue with overlapping silkscreen elements. I fixed this issue in the previous chapter, and therefore I don't expect the check to show any new violations. I do expect warnings, though; however, I can simply ignore them and continue manufacturing.

Let's go ahead with the DRC. Click on the DRC button in the top toolbar, and then click "Run DRC". Here are the results:

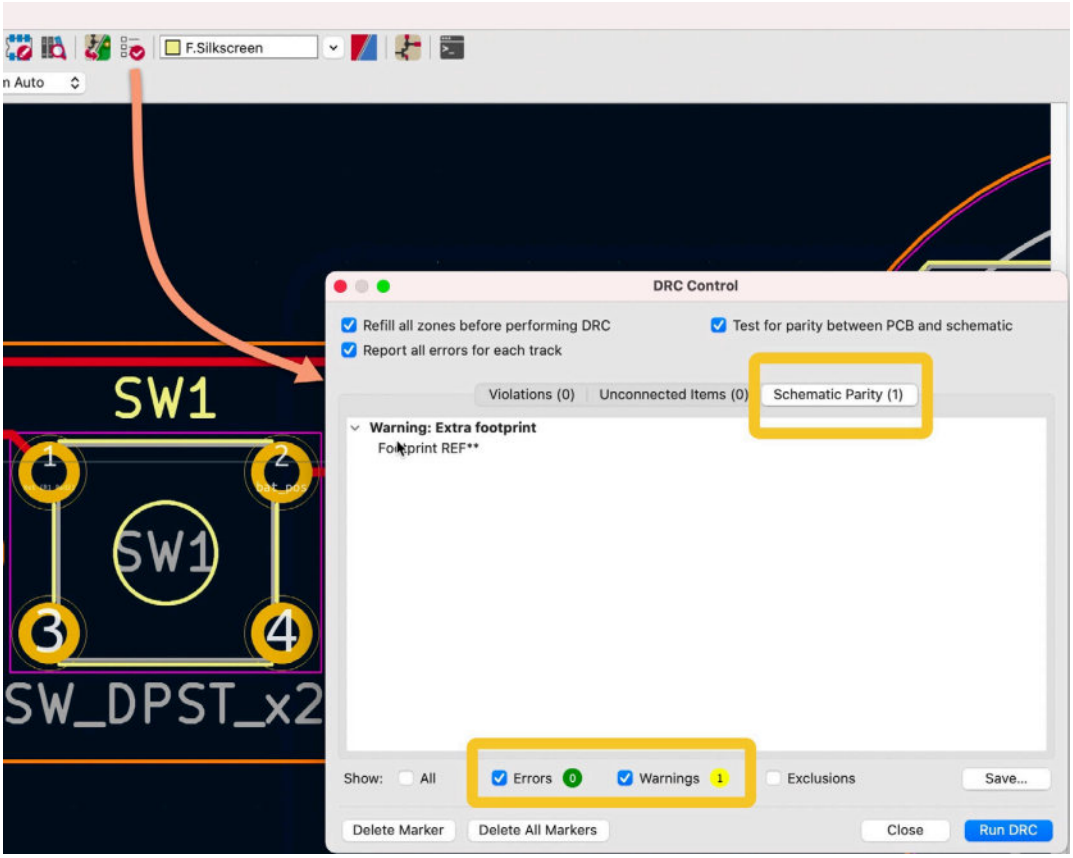


Figure 4.8.1: The results of the last DRC.

After fixing the silkscreen violation found the first time I ran the DRC (I fixed this in the previous chapter), the new DRC shows only a warning. This

warning relates to the KiCad logo I placed on the back of the board in the last chapter. To see the location of the warning, click on the warning. Pcbnew will pan the editor to the location of the warning:

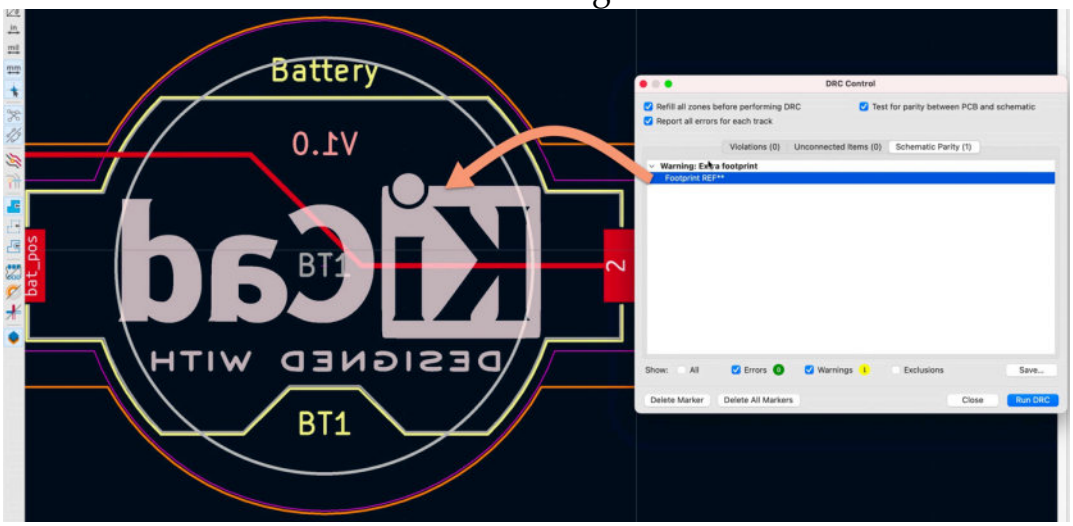


Figure 4.8.1: This warning relates to the logo footprint in the back silkscreen layer.

You can safely ignore this warning and continue with the next step of the workflow (export the Gerber files) since it does not affect any functional components of the board. The warning indicates that the reference designator for the logo footprint contains "**" at the end of its name and therefore is undefined.

Even though I can simply ignore this warning, I will go ahead and fix it so that the DRC returns no errors or warnings.

Double-click on the KiCad logo footprint to bring up its properties:

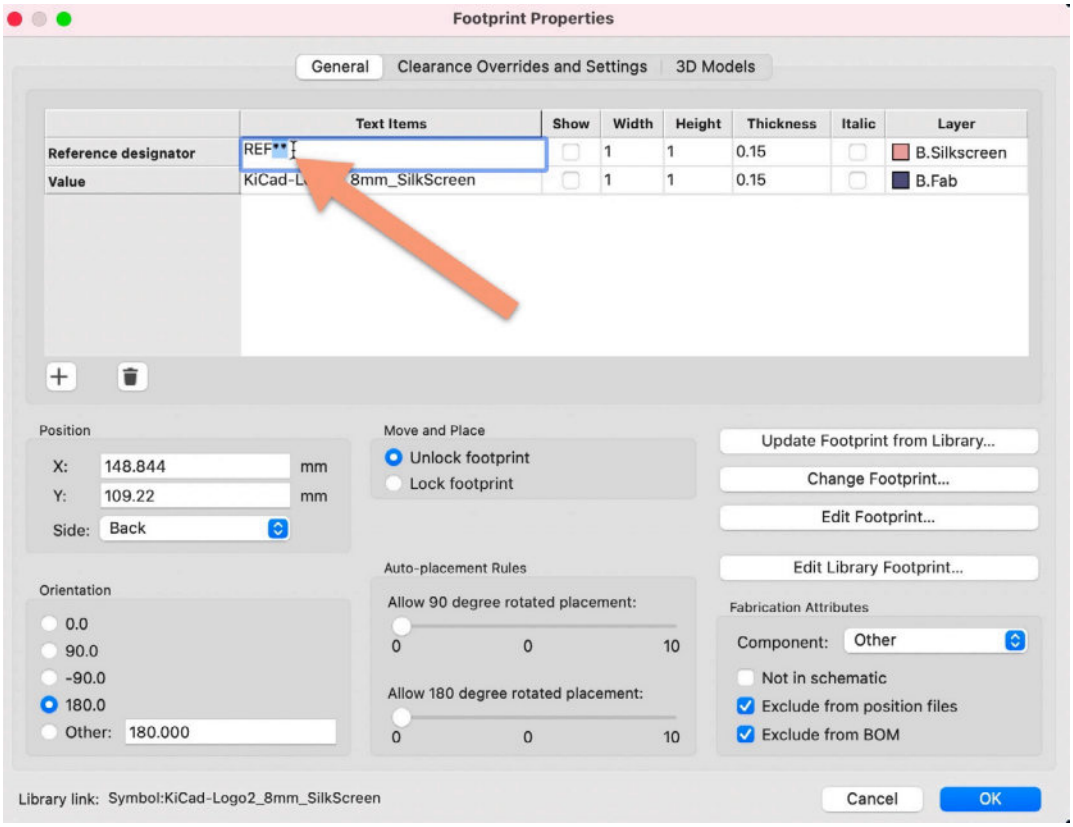


Figure 4.8.2: Set the reference designator.

Notice that the reference designator is "REF**". I will change this to "REF1", which is unique on this board, and click OK.

Run the DRC once more to see if this clears the original warning:

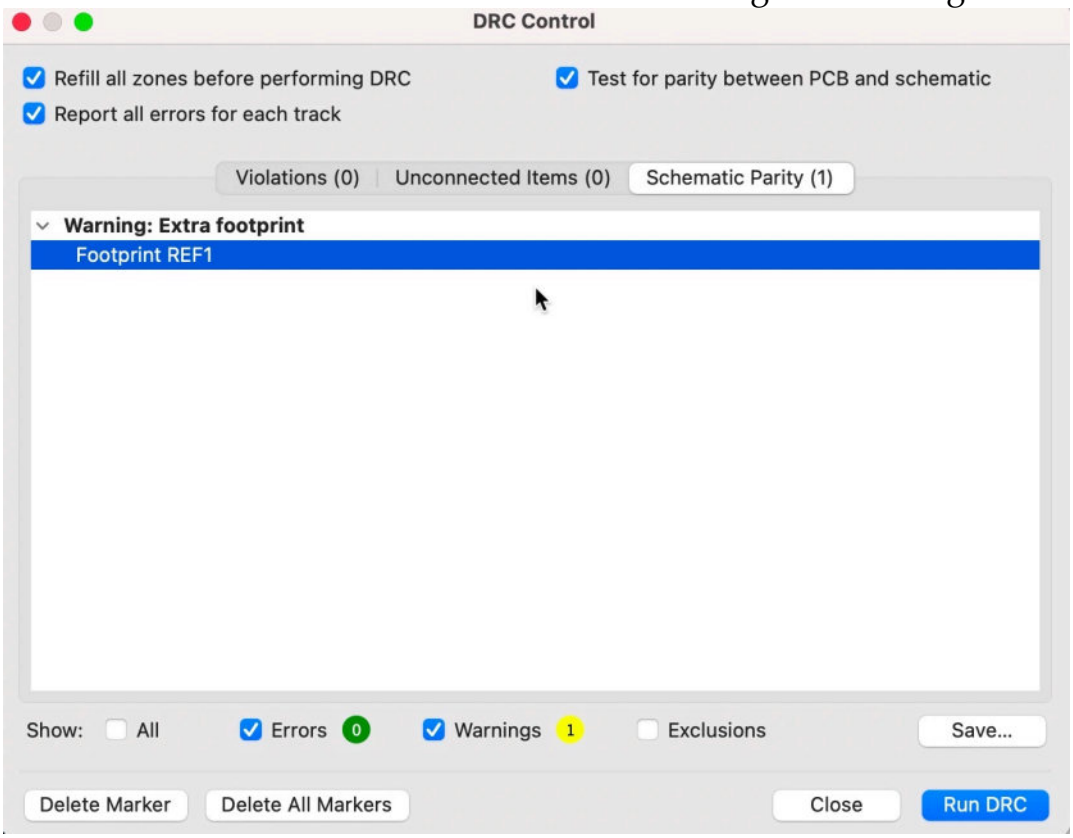


Figure 4.8.3: A new warning appears in the DRC.

This warning indicates that the footprint with designator "REF1" exists in the layout editor but not in the schematic editor. This is true since I added this footprint to the PCB in step five of the layout workflow. Unlike the rest of the footprints in the PCB, REF1 does not have a symbolic counterpart in Eeschema.

Now I have three options:

1. Go back to the schematic editor, add a new symbol and associate it with the logo footprint.

2. Ignore this warning.

3. Open the footprint's properties and check the checkbox "Not in schematic" (see below).

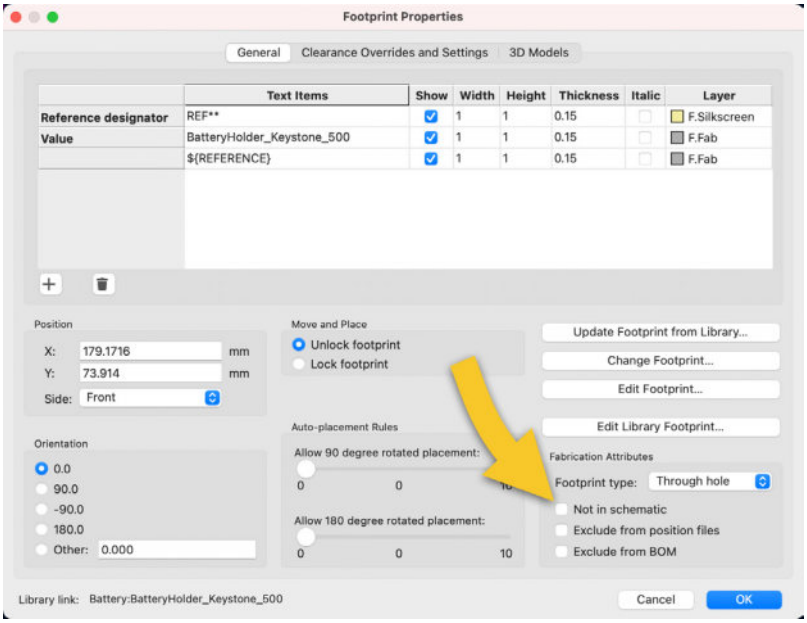


Figure 4.8.4: The "Not in schematic" option in Footprint Properties.

A manufacturer will only use the data present in the Gerber files, which comes from the layout. The data in the schematic editor play no role in the final manufactured PCB. Therefore, any additional work I do now will not affect the final product but only the internal consistency of my project.

Since I want to get on with the project, I will choose option two and ignore the DRC warning. The logo is not a functional component of my PCB, and whether it exists or not in the schematic, the PCB will work the same.

I have now completed the Design Rules Check. I have chosen to ignore the only warning that it returns. I am now ready to export the Gerber files and have my PCB manufactured.

9. 8 - Export Gerbers and order

In this chapter, I will complete step seven of the PCB layout workflow, that you learned about in the second chapter of Part 3 of the book. In this step, I will export the Gerber files, which contain the data that the online manufacturer will need to manufacture my PCB. Before I upload the Gerber files to the manufacturer's website, I will use a special tool to check that the files are correct.

Pcbnew supports Gerber files export as one of its fabrication outputs. You can learn about Gerber files and the process to export and test them in Pcbnew in the dedicated chapter later in this book. In this chapter I will summarise the process.

Go to File, then click on "Fabrication Outputs", and then "Gerbers".

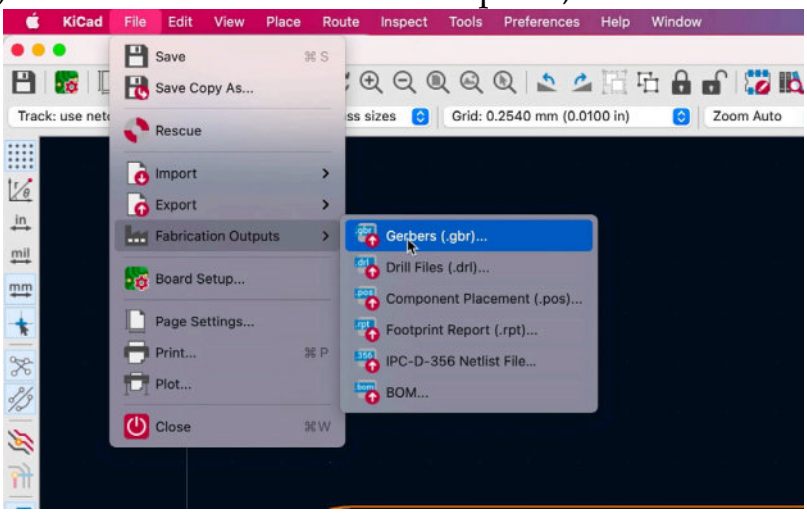


Figure 4.9.1: Starting the Gerber files export process.

In the Plot window that appears, select your Gerbers output directory. I set this directory to exist inside my current project directory. Click on the folder button to navigate your file system and choose an output directory. In Figure 4.9.2, you can see the settings for my Gerber files export. Double-check the output directory, included files (exactly as they appear below), Gerber options (use Protel filename extensions), and enable the extended X2 format.

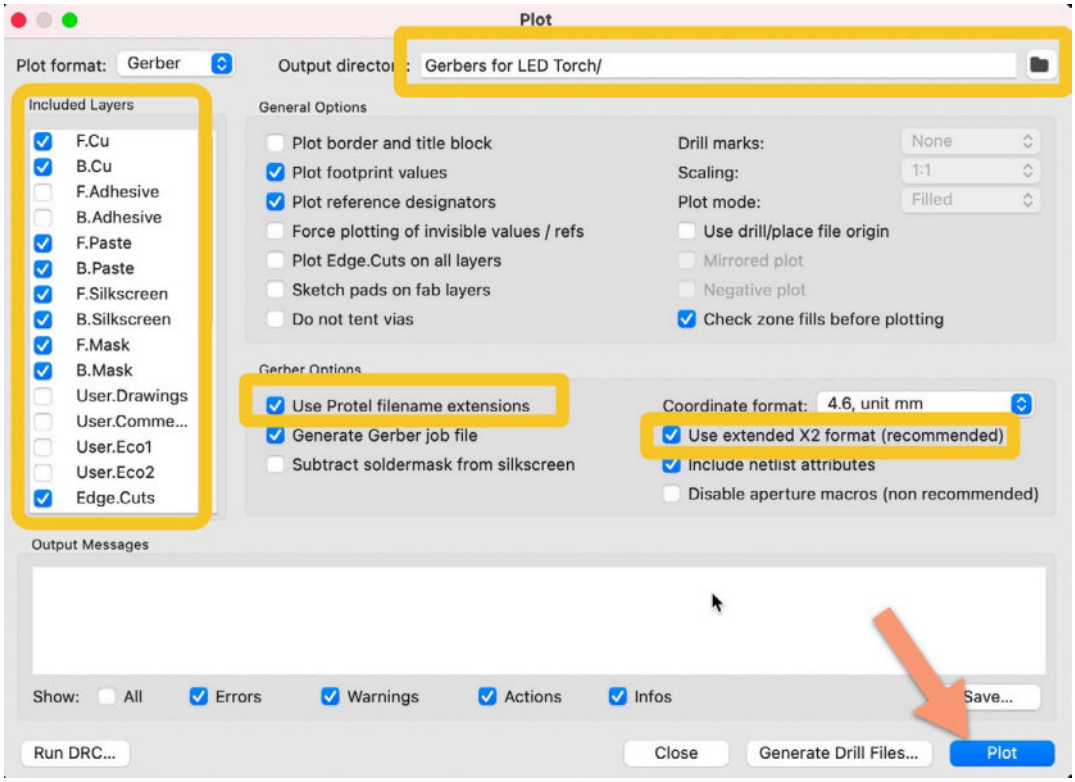


Figure 4.9.2: The settings for the Gerber files export.

Click on Plot to generate those files. In the output messages box, you will see confirmation that the Gerber files were created. You can also check the contents of the output directory to confirm that the files are there.

You will also need to generate the drill files, which contain information about through-holes and vias. Click on the "Generate Drill Files" button. In the window that appears, accept the defaults.

Be careful: The button "Generate Drill Files" appears on both the Plot and the "Generate Drill Files" windows. Click on the "Generate Drill Files" button in the Plot window to open the "Generate Drill Files" window, and then "Generate Drill Files" (in the "Generate Drill Files" window) to actually generate the drill files.

Here's the Generate Drill Files window with the default options:

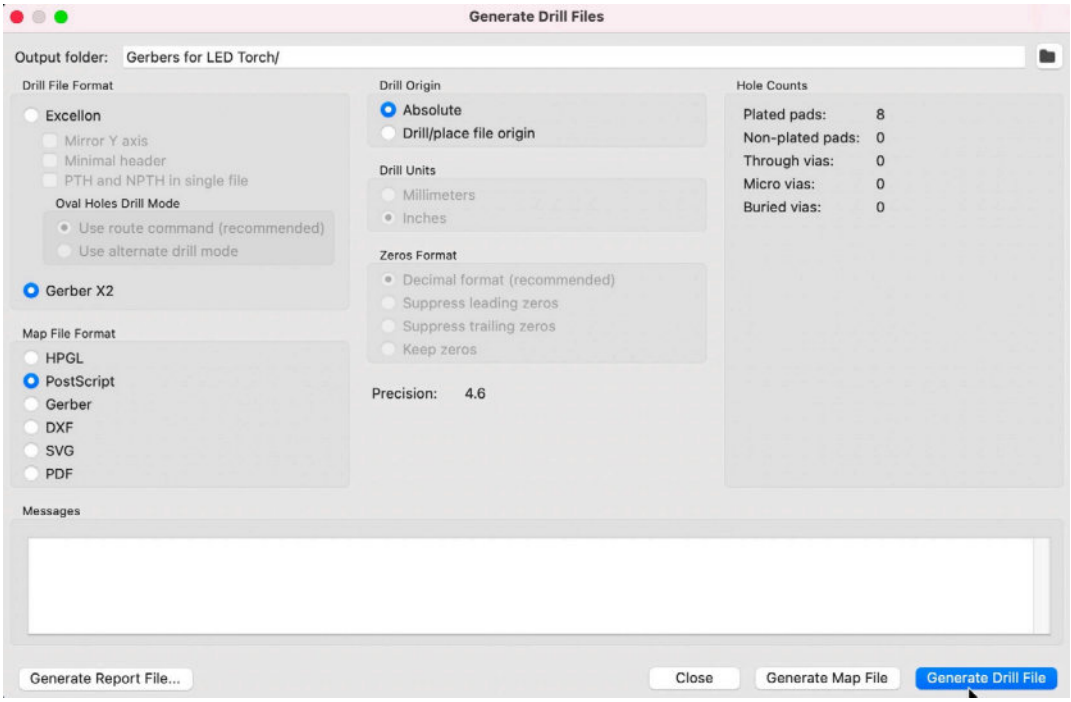


Figure 4.9.3: The settings for the drill files export.

Click the "Generate Drill File" button.

The output folder now contains a collection of Gerber files. It looks like this:

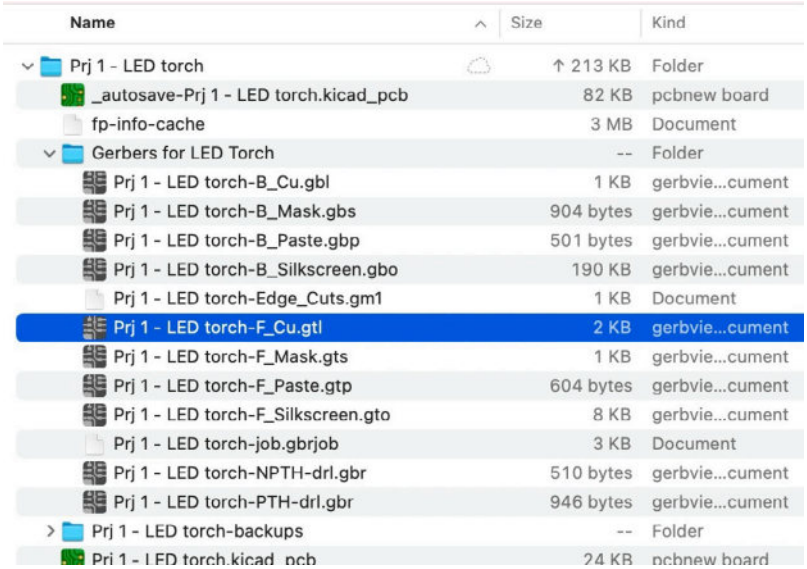


Figure 4.9.4: The exported Gerber files.

Most online manufacturers ask that you compress the Gerber files directory as a ZIP file and upload the ZIP file. Therefore, I will create the ZIP from the contents of the Gerbers output directory:

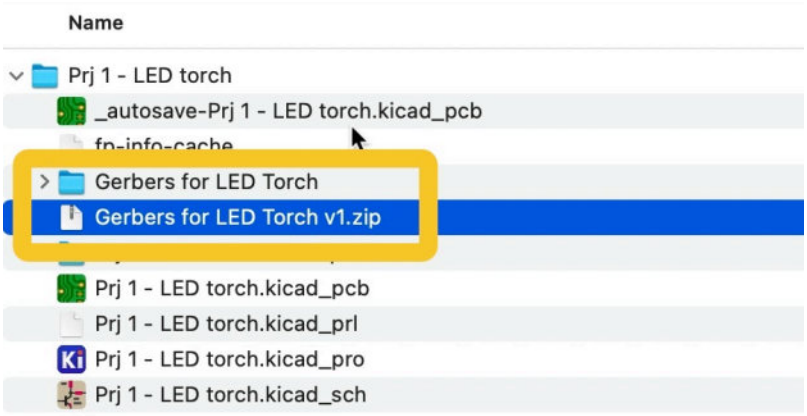


Figure 4.9.5: The compressed (ZIP) Gerbers output directory.

I will use KiCad's Gerber viewer app to open the Gerber files and inspect them. The Gerber viewer will help me notice problems with the Gerber files that could lead to defects to the manufactured PCB. The better online manufacturers may do a quick manufacturing check before accepting your order, but ultimately it is your responsibility to upload bug-free files.

From the KiCad main project window, click on the "Gerber Viewer" button. In Gerbview, go to File and click on "Open Gerber Plot File(s)…":

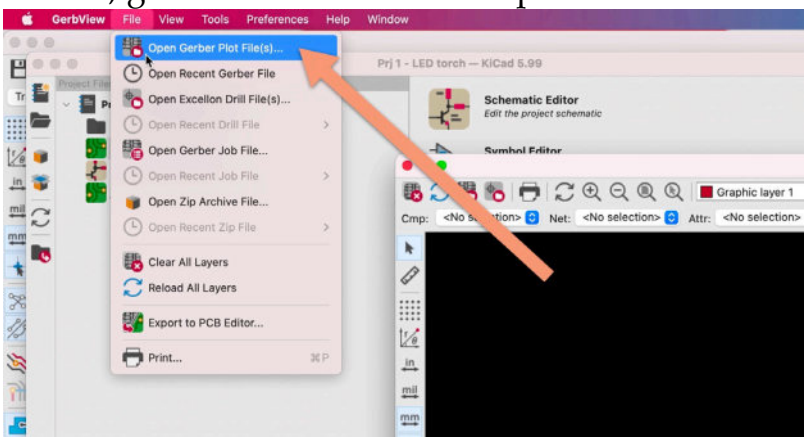


Figure 4.9.6: Open the Gerber files in Gerbview.

You can import one Gerber file at a time or all together (which is my preference). From the file chooser, under "File type", choose "All Files", and then multiple-select all files in the Gerber output directory except for the one with the extension "gbrjob".

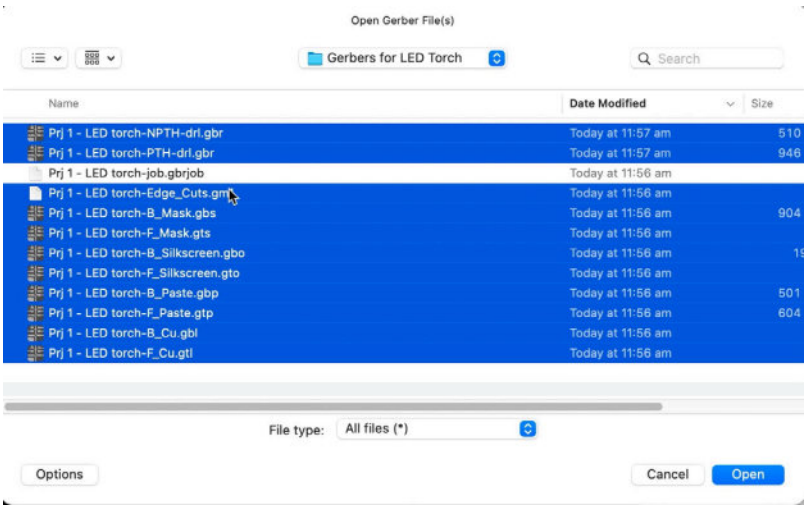


Figure 4.9.7: Import all files.

Gerbview will render the Gerber files so that you can visually inspect each one at a time (Figure 4.9.7). You can enable and disable layers from the Layers tab of the Layers Manager (right pane). Take the necessary time to review each layer to ensure that there are no omissions or errors.

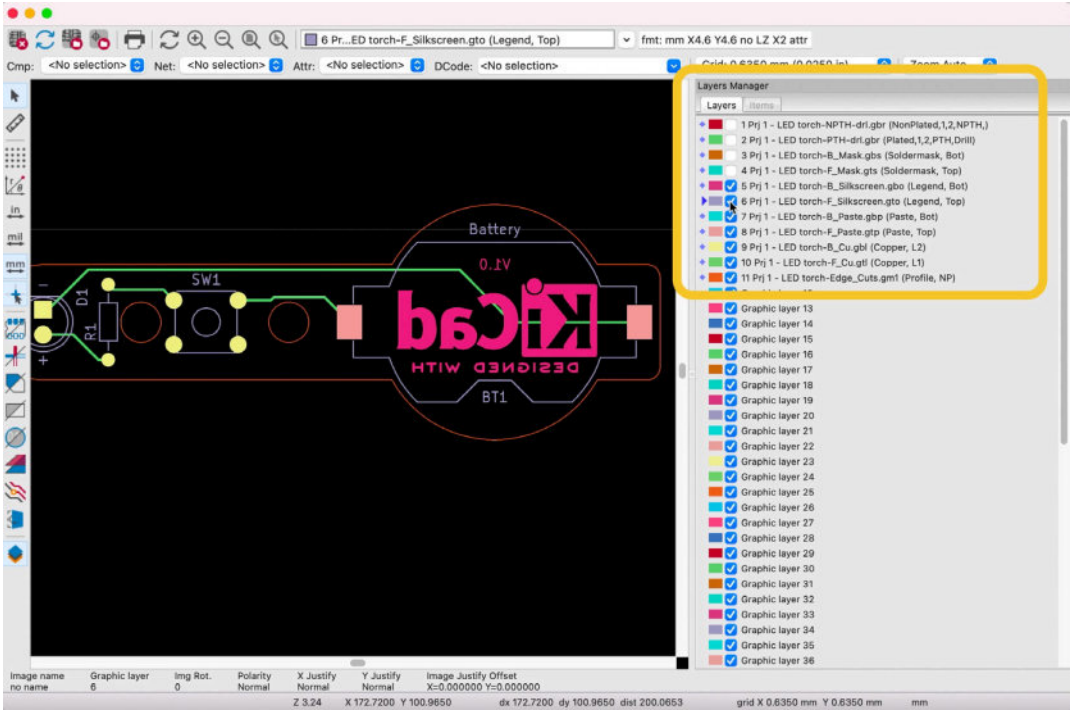


Figure 4.9.7: Gerbview showing my new PCB.

In addition to Gerbview, you can use online Gerber tools such as gerber-viewer.com.

Once you are confident that your Gerber files are correct, you can continue with the manufacturer's website process. Each online manufacturer has its process. Roughly the process involves uploading the Gerbers ZIP archive, confirming that the Gerbers are correct using the manufacturer's Gerber files

viewer, selecting the various finishing options, and paying for the manufacturing and shipping.

An essential bit of information that most manufacturers will need is the width and height of your board. Use the measuring tool in Pcbnew to find out this information. For my current project board, you can see its dimensions below:

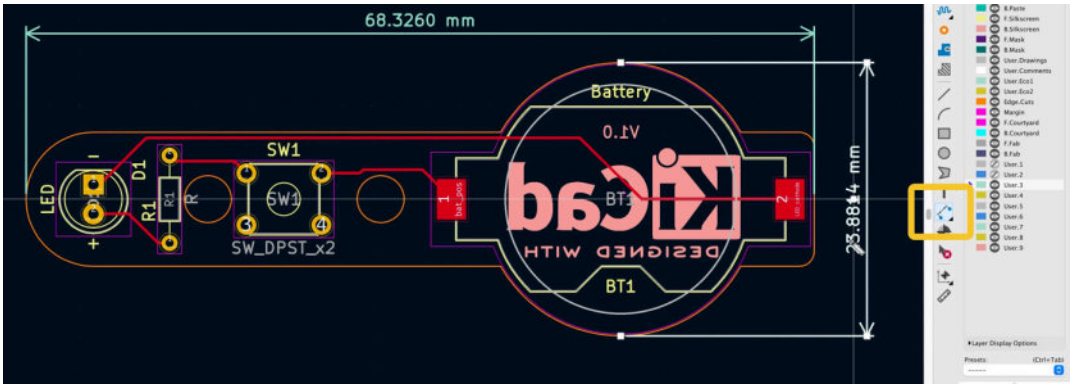


Figure 4.9.8: The width and height of this board.

A manufacturer like Oshpark.com is one of the easiest to use, though it lacks finish options. You can simply upload the Gerbers ZIP file, confirm that the Gerber files are correct, and choose one or more simple options:

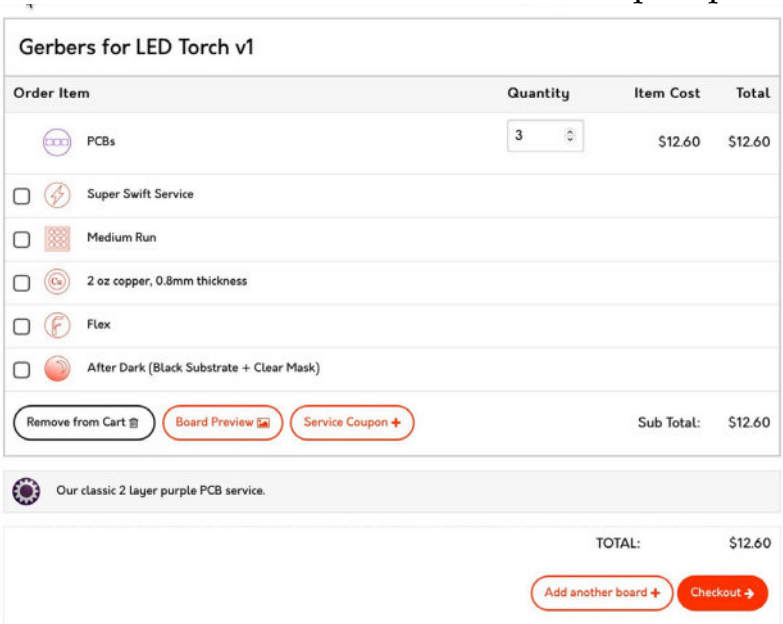


Figure 4.9.9: Oshpark's user interface is as simple as it gets.

With Oshpark, there's only a handful of options you can choose. If you want more control, consider a service like Pcbway.com.

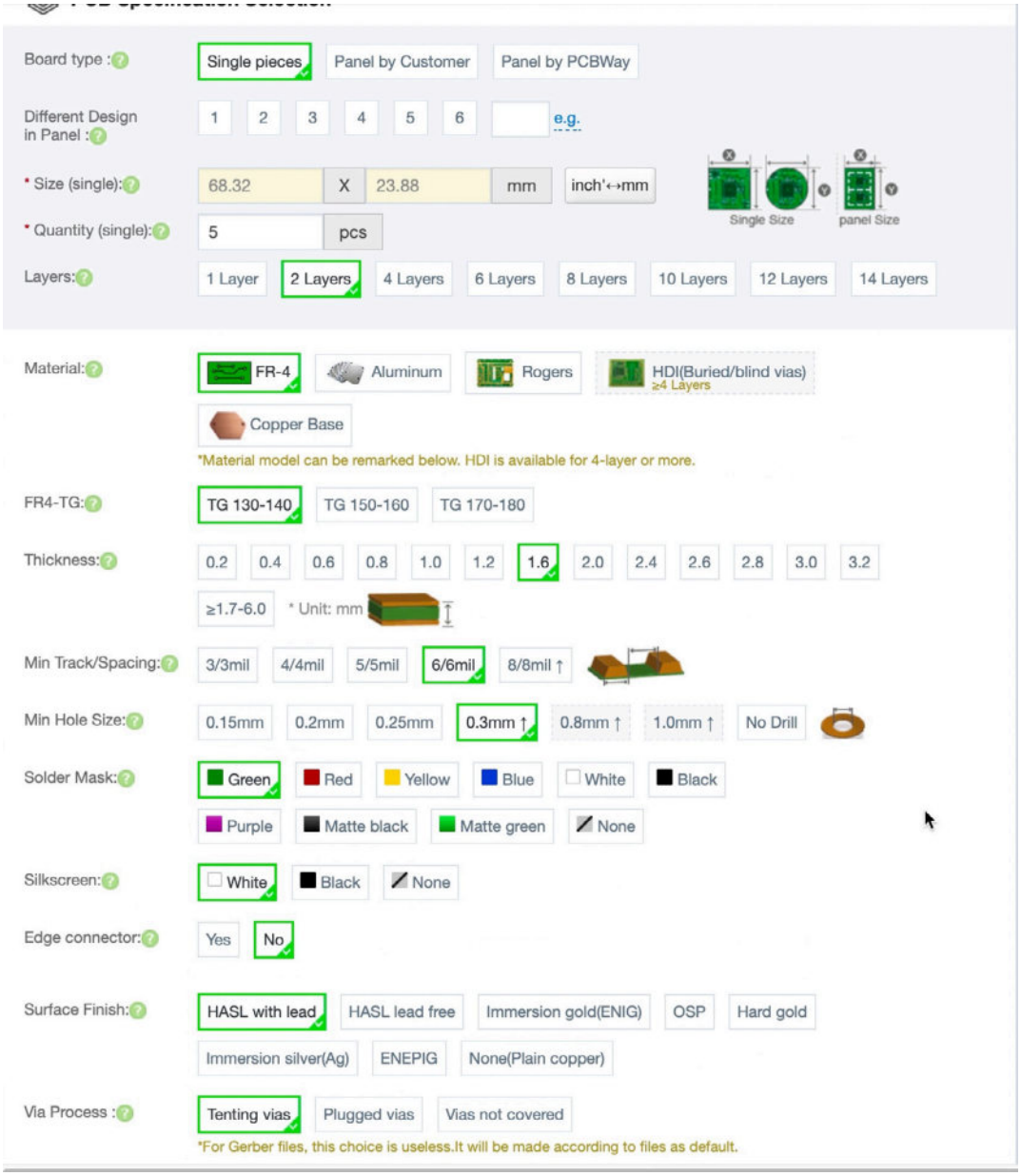


Figure 4.9.10: Pcbnew offers lots of options.

With Pcbnew and similar services, you can configure the manufacturing of your PCB with choices of substrate material, thickness, solder mask color, and much more. I keep the defaults in most cases, but I do customize the color (I like red PCBs).

Reputable online manufacturers offer fast service and high-quality results. A couple of weeks after my order, I received my new PCBs. Let's take a look at them in the next chapter.

10. The manufactured PCB

A couple of weeks after my order, I received the finished PCB in the mail. I placed my order on pcbway.com using the settings you can see in Figure 4.10.10.

In summary, these are the PCB specifications:

1. Copper traces on the top layer only.
2. Edge cuts with rounded segments and no ninety-degree corners.
3. Both surface-mounted and through-hole components.
4. A mounting hole.
5. Silkscreen text and graphics at front and back.

Here are a few photographs from the PCB (Figure 4.10.1):

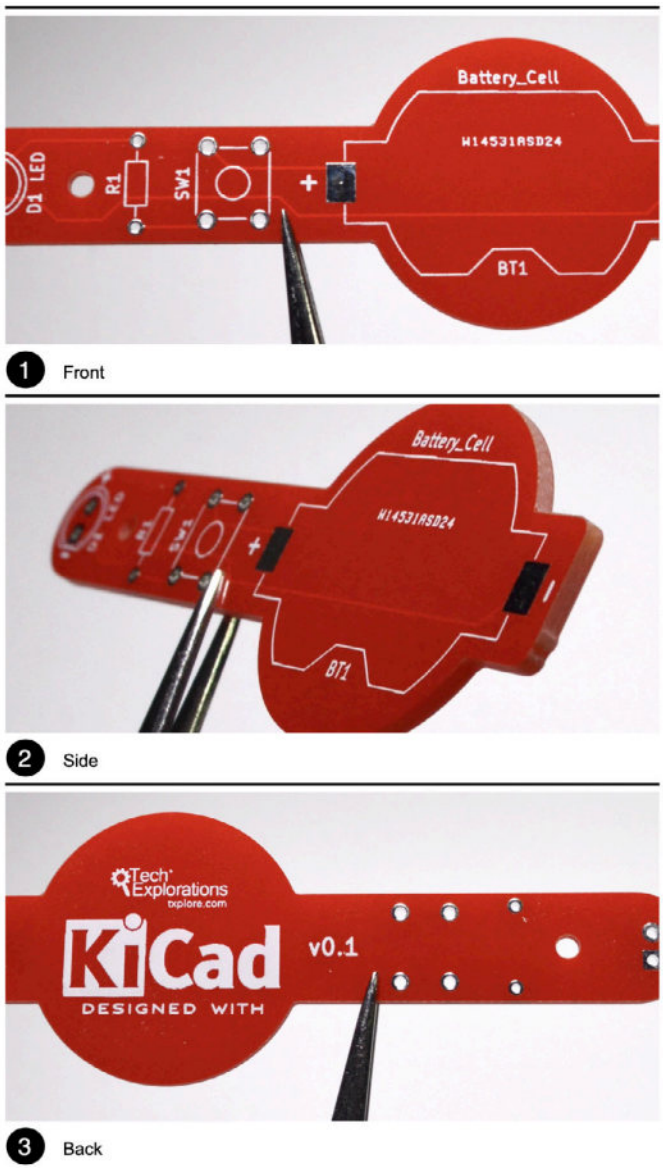


Figure 4.10.1: Photographs from the manufactured PCB of this project.

There are a couple of differences between the PCB in the photos above and the one in Figure 4.10.8. There is only one mounting hole and the Tech Explorations logo in the back. The reason for these differences is that I manufactured this PCB as part of a prototype run in preparation for writing this chapter. During the actual writing, I introduced a second mounting hole. Still, I did not include the Tech Explorations logo because creating it requires an additional step that I cover later in this course.

Also, notice an alpha-numeric code printed in the front side silkscreen (image 1, above). This is a tracking code that the manufacturer uses as part of their production process. You can request not to print this code; however, this will increase the PCB price because it affects the degree of automation of the manufacturing process.

Part 5: Design principles and PCB terms

1. Introduction

In Parts three and four of this book, you learned about some of KiCad's most commonly used features. You used Eeschema and Pcbnew to design the schematic diagram and the layout of a simple board. You also had your first experience with some of the decisions a PCB designer must make during a printed circuit board design process.

Before you continue your learning journey with more interesting PCB projects, it is appropriate to become familiar with concepts, conventions, and design patterns that will help you work and produce better-performing boards.

You will learn about the symbols and units that appear in the schematic diagrams and the layout diagrams.

You will also learn the terminology used to describe a printed circuit board's various components and characteristics.

Later in this book, in Part six, you will learn about the general processes of the schematic and layout steps of designing a PCB. These are processes that every designer will go through to one degree or another, regardless of which CAD application they are using.

2. Schematic symbols

Electronics and PCB design has their own symbolic language. We use this language to create schematic diagrams. In Figure 5.2.1 you can see an example[1] of a schematic that contains several symbols of this language.

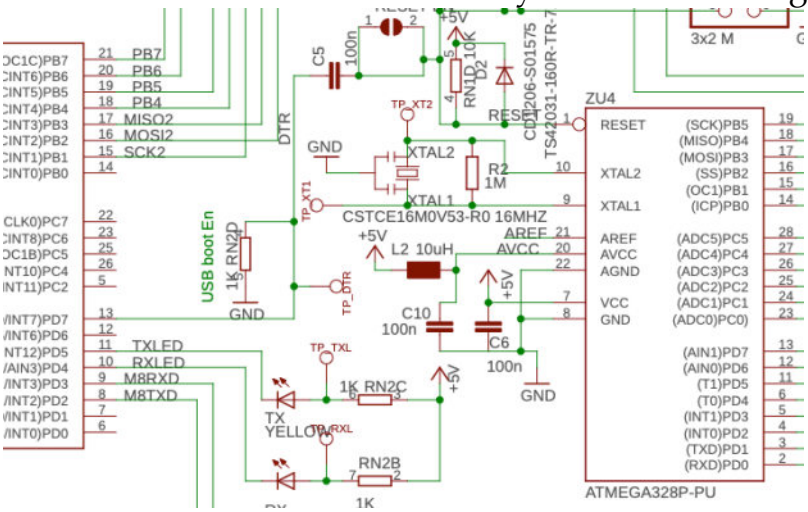


Figure 5.2.1: A segment of the Arduino Uno Rev3 schematic diagram.

This example shows the schematic symbols of several components that make up the Arduino Uno Rev3. The large rectangular symbol with the designator 'ZU4' is the ATMEGA328P-PU microcontroller chip. The symbol contains several pins that provide inputs and outputs, and each of them is named.

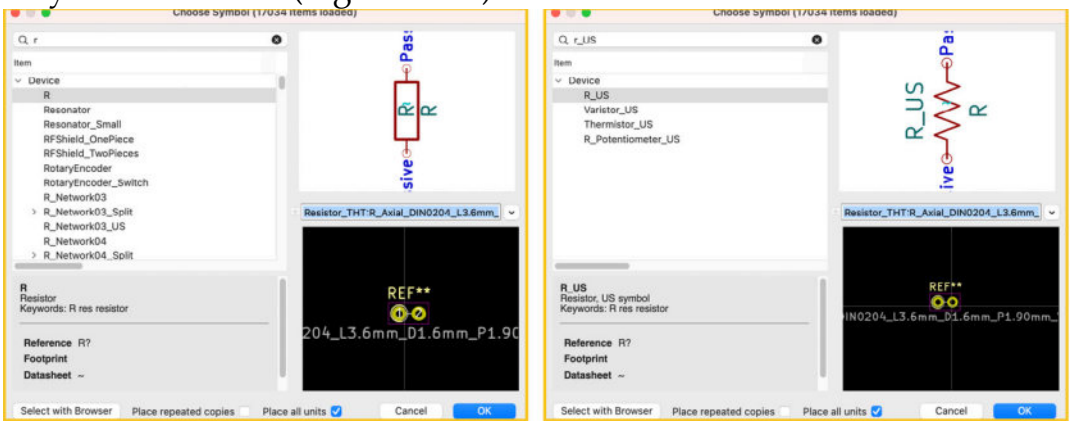
You can see a few capacitors, designated C5, C10, and C6, none of them polarized. There is one resistor, R2, and a few resistor networks (like RN2 and RN1, which are simple resistors in a network configuration). The capacitors and the resistor also have their values marked in the schematic. There are also symbols for an LED, a jumper connector, and power (Vcc, RAW, and GND).

All of these symbols follow a particular standard. . Several standards are available, but most notably, engineers worldwide tend to work with the American style ('IEEE') or the European ('IEC') style. The symbols in Figure 5.2.1 follow the IEC style.

[1] See source of this schematic at
https://content.arduino.cc/assets/UNO-TH_Rev3e_sch.pdf

In KiCad, you can choose to use either the American or the European style symbols. Whichever one you choose, be consistent. Do not mix American-style resistor symbols with European-style capacitor symbols in the same schematic.

The KiCad standard symbols library contains symbols of both styles, though the European symbols seem more plentiful. In most cases, American-style symbols will have the postfix 'US' in their name. In Figure 5.2.2, you can see an example of a resistor symbol, with the European style on the left and the American on the right. The name of the American-style symbol ends in 'US,' and you can take that into account when you are searching for a symbol in the Symbol Chooser (Figure 5.2.2).



5.2.2: A resistor, European-style (left) and American-style (right).

3. PCB key terms

Creating printed circuit boards is an engineering discipline. As such, it has its own 'language.' In this chapter, you will learn the most commonly used terms to understand the information found in places such as PCB fabrication websites and CAD tool documentation.

3.1. FR4

The most common material used to make printed circuit boards is FR4 (or FR-4). It is a glass-reinforced epoxy laminate composite material, or in simpler terms, fiberglass cloth bound using an epoxy resin.

Go over to the Wikipedia article[2] to read more about this material.

The 'FR' part of the name stands for 'Flame Retardant,' a desirable quality for a board that will hold together components that can potentially ignite when they fail.

Other valuable attributes of the FR4 substrate are:

- Very light and strong
- Does not absorb water
- It is an excellent isolator
- Maintains its quality in dry and humid environments

Other materials can be used in rigid or flexible printed circuit boards, apart from the standard FR4 and variants (like FR4 tracking resistant and halogen-free). Examples include G-11 for applications that must operate in high temperatures, FR-3 (cotton-paper impregnated with epoxy), and Polyimide[3] (high-performance yet expensive, appropriate for cryogenic applications).

3.2. Traces

Traces (also called 'tracks') are conductive paths. Most often, the material used to make traces is copper. Electrical signals and power use traces to travel throughout a circuit.

In Figure 5.3.2.1, you can see the traces in the front side of this PCB as thin purple lines that provide the connections between the golden pads where

[2] See FR4 entry: https://en.wikipedia.org/wiki/FR-4

[3] See Polyimide entry: https://en.wikipedia.org/wiki/Polyimide

the component terminals will eventually be. You will learn more about pads later.

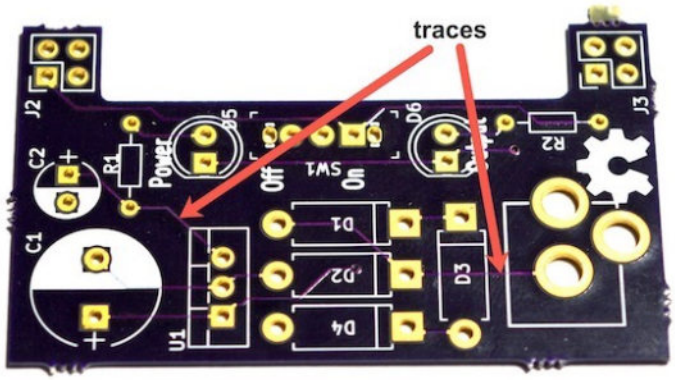


Figure 5.3.2.1: An example of traces.

As the designer of a PCB, you have total control over the characteristics of traces. You can control their width, height, and route, including the angles by which a trace changes direction. If you want a trace to accommodate a large current flowing through it with little resistance and temperature rise, you can design it wider and thicker. This is useful when a trace must feed power to the components of your board. Traces that convey low power-current signals (less than 20 mA) can be narrower using less copper.

Keeping the width of traces to around 0.3 mm (or even less, depending on your manufacturer's guidelines) makes it possible to draw traces closer together and reduce the final size of your PCB.

Figure 5.3.2.2: An example of wide traces.

In Figure 5.3.2.2, you can see an example of much wider traces than regular signal traces. These traces connect the terminals of a 240 Volt relay.

The traces in these examples are purple because of the solder mask chemical used to finish the manufacturing process. You will learn about the solder mask further down in this chapter.

3.3. Pads and holes

Pads and holes are the most prominent feature of a printed circuit board. Pads come in two varieties: TH (through-hole) and SMD (surface-mounted device). For each, there are several shapes.

In Figure 5.3.3.1, you can see an example of a board that contains TH pads exclusively, and in Figure 5.3.3.2, you can see a board with TH and SMD pads.

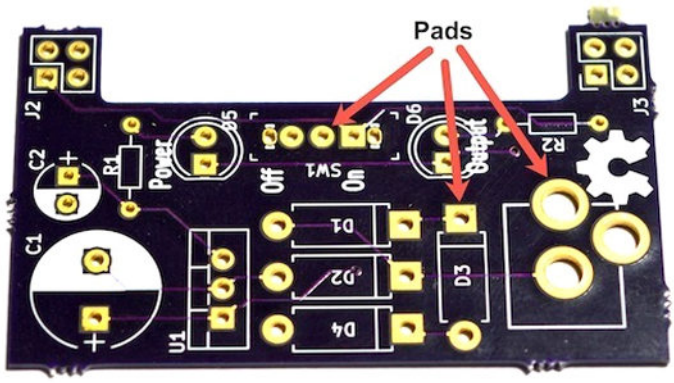


Figure 5.3.3.1: Through-hole pads.

Through-hole pads, unlike SMD pads, connect the front for the PCB with the back electrically. In the examples, you can see that the gold plating of the pad fills the inside of the hole. If you turn the PCB around, you will see that a matching pad exists in the back.

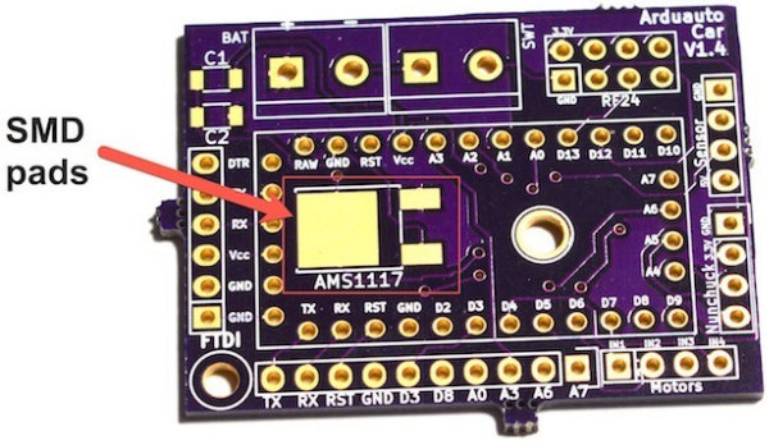


Figure 5.3.3.2: An example of SMD pads.

Boards with mostly TH pads are popular among hobbyists because through-hole components are easier to work with, at least initially. SMD components are smaller; hobbyists tend not to use them until they are more

comfortable with their soldering skills. With a bit of practice, SMD components are as easy to work with as their TH counterparts.

In the industry, on the other hand, the vast majority of PCBs are designed to contain SMD components. This is because SMD components can be populated automatically using pick and place machines and because their small size results in smaller PCBs.

Apart from the two varieties I described above, pads also come in several shapes. Most often, you will see round pads, but rectangular and oval shapes are also possible. Using KiCad, you can create such pads and control their geometry to the extent that your PCB manufacturer allows.

In Figure 5.3.3.3, you can see an illustration of a cross-section of a PCB showing the configuration of pads, and two types of holes, Plated-Through Hole (PTH) and Non-Plated Through Hole (NPTH).

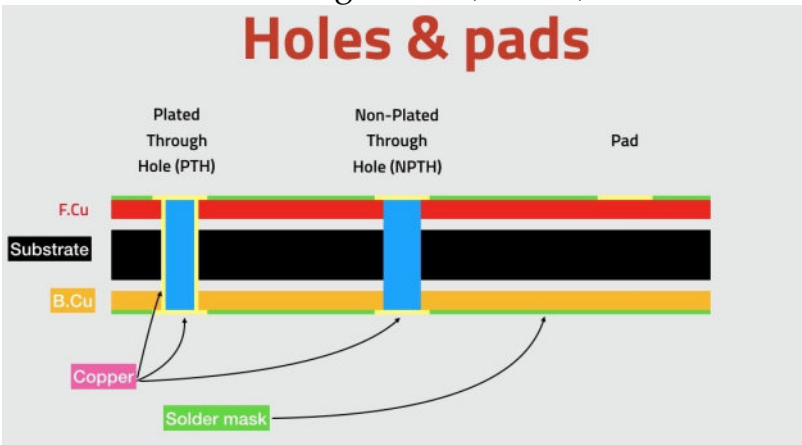


Figure 5.3.3.3: Pads and holes.

Plated-Through Holes is the more common variety and the default type of hole in more cases. We use a drill to create the hole and then copper to cover the hole's sides so that its two ends (at the front and back copper layers) are electrically connected. Vias are constructed the same way, except they have a smaller diameter, so it is impossible to accommodate component pins.

On the other hand, in a Non-Plated Through Hole, we use the same drill to create the hole, but no copper is used to cover the sides of the hole, so there is no electrical connection between its two ends.

Finally, pads without holes are useful for attaching surface-mounted components, as you learned earlier.

3.4. Via

You can create a via when you want to move a signal that travels across a trace from one side of a PCB to another (say, from front to back). A via is a

hole with its sides covered with copper or gold (or other conductive material) that allows a trace to continue its route across layers.

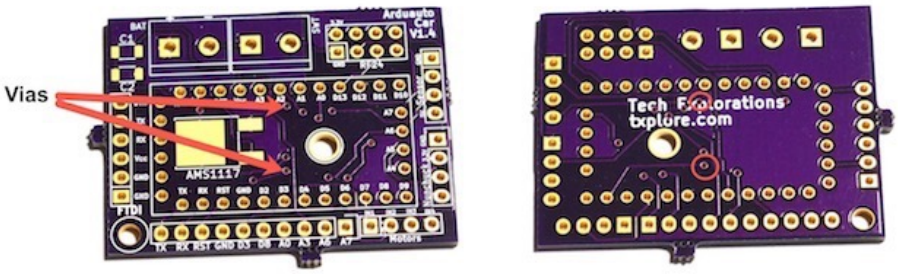


Figure 5.3.4.1: Vias allow a trace to continue between layers.

In Figure 5.3.4.1, you can see the two sides of the same PCB. On the left, the arrows point to two vias in the front of the PCB, and on the right, the circles indicate the same vias on the back of the PCB. Vias are similar to through-hole pads, except they don't have any exposed copper (the solder mask covers them), and they don't have a pad (so you can't solder a component).

In simple circuits with only a few components, it is possible to create all traces on one layer of the PCB. When a PCB gets busy with more components, it quickly becomes impossible to do the routing on a single layer. When multiple layers are needed, vias provide the simplest method of allowing a trace to use the available board real estate.

In Figure 5.3.4.2, you can see the types of interconnections between layers that are possible.

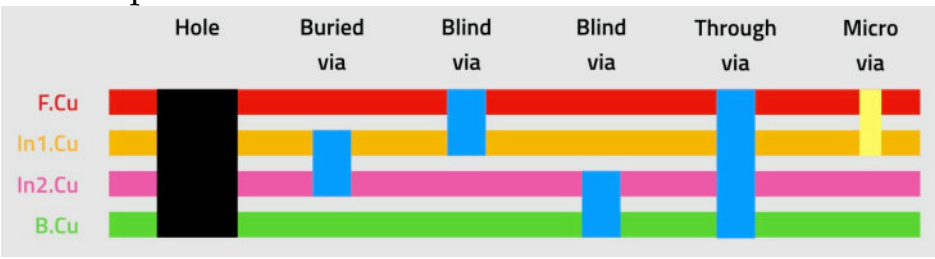


Figure 5.3.4.2: Types of interconnections between layers.

For through-hole components, you would design a hole that connects the top and bottom copper layers. We use a drill to create this hole. It is wide enough to allow for the pin of the component to go through it.

In vias, the diameter is smaller when compared to a regular platted hole. They are not wide enough for pins to go through them, but they are plated, like holes, and allow for electrical connection between layers.

A 'through via' is like a hole but narrower. It connects the top and bottom layers. A buried via is a via that connects any two internal layers. In the four-layer example of Figure 5.3.4.2, the buried via connects the In1.Cu and In2.Cu.

A 'blind via' also connects two layers but has one end exposed to the outside of the board, either top or bottom.

Another option for interconnecting layers in high-density boards is to use a 'micro via' ('uvia'). A micro via is made using a high-powered laser instead of a mechanical drill; the use of lasers makes it possible to reduce the diameter of the via dramatically .

3.5. Annular ring

The annular ring is a term that describes the area on a pad that surrounds a via. A primary metric of an annular ring is its width, defined as the minimum distance between the edge of the pad and the edge of the via or pad hole.

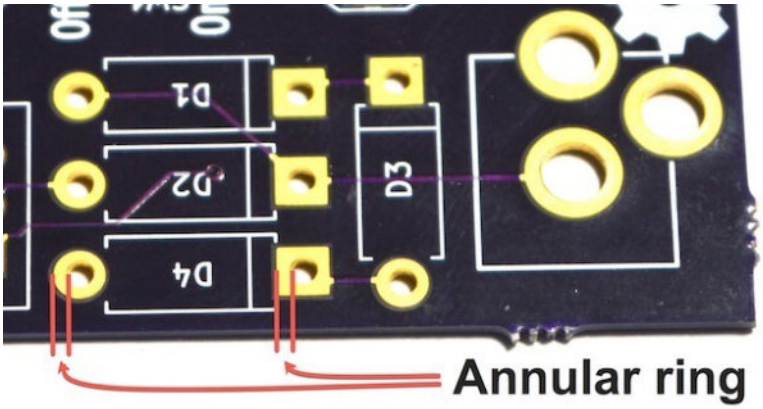


Figure 5.3.5.1: Annular rings and width.

In Figure 5.3.5.1, the width of two annular rings is marked with two red lines. Ideally, the drill hit (the location on the board where the drill lands and creates a hole) is in the middle of the pad. If the drill bit is not aligned correctly, the hole can be closer to one edge of the pad (a 'tangency'), or it could even miss the pad completely (a 'breakout').

3.6. Soldermask

As you know, traces are made of copper. Copper slowly reacts with oxygen in the air, resulting in oxidization. Oxidized copper produces a pale green outer layer. PCB manufacturers cover the exposed copper with a solder mask, a thin layer of polymer that insulates it from oxygen to prevent this from happening. As an additional benefit, the solder mask also prevents solder bridges from forming between pads.

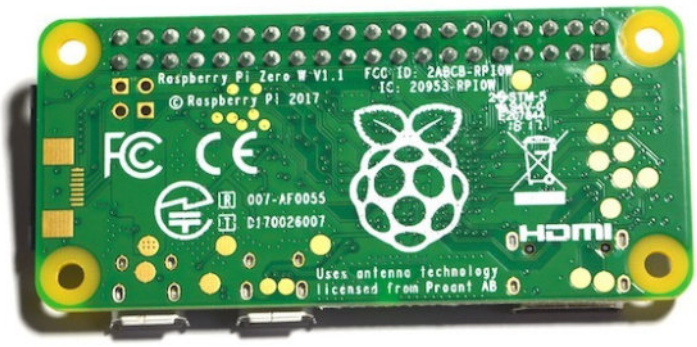


Figure 5.3.6.1: The rear of a Raspberry Pi Zero is protected by a thin layer of solder mask.

In Figure 5.3.6.1, you can see the back of a Raspberry Pi Zero. In this example, the copper is protected by a thin layer of green solder mask. Only the pads and the mounting holes are not covered by the solder mask.

Solder mask polymers are available in different colors, with green being the most common and cheaper. You can create fancy-looking PCBs with black, blue, red, purple, and many other colors.

3.7. Silkscreen

Printed circuit boards are not complete without text and artwork. The purpose of those elements is to convey useful information and add a touch of elegance. In Figure 5.3.7.1, you can see an example of such text and artwork on the back of a Raspberry Pi Zero. You can see the Raspberry Pi logo, logos of various certifications, and different text items that inform us about the model, etc. All this consists of the silkscreen.

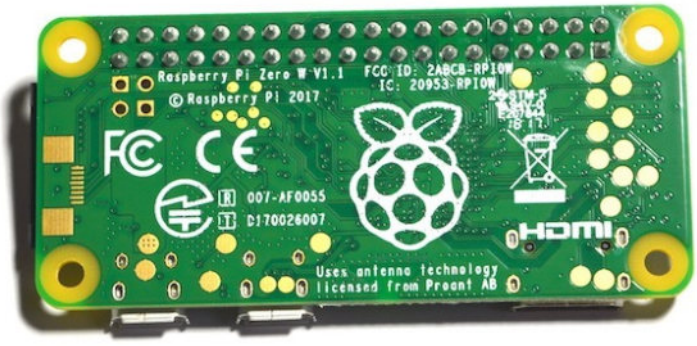


Figure 5.3.7.1: The

The name 'silkscreen' is somewhat misleading. Of course, no natural silk is used to produce the white elements on the PCB. The method used to print the silkscreen in large numbers is a relative of the traditional screen

printing process that you can use to print a graphic on a T-shirt. The silkscreen text and graphics are printed on the boards while they are still in their panels.

White is the most common color for the silkscreen, but black and yellow are also available.

In the projects that you will work through later in this book, you will spend a considerable amount of time creating the informational and decorative text and graphics in the silkscreen layer of the PCB.

3.8. Drill bit and drill hit

Drill bits are used to create holes and vias, but also cutouts. Drill bits are typically made of solid coated tungsten carbide material and come in many sizes, like 0.3 mm, 0.6 mm, and 1.2 mm. They look like the one in Figure 5.3.8.1. These drills are attached to computer-controlled drilling machines and are guided by a file that contains information about the coordinates and the drill size for each hole on the PCB.

Figure 5.3.8.1: A drill bit.

It is interesting to note that drill bits are replaced with lasers for tiny holes, like vias. These vias are often called 'micro vias. With laser drilling, it is also possible to create vias that connect in-between layers of the PCB.

The term 'drill hit' describes the location on the PCB where the drill bit comes in contact with the PCB and creates a hole.

3.9. Surface mounted devices

If your objective is to create a PCB that is easy to manufacture in large numbers, with a minimum size, you should design it to contain surface-mounted components instead of through-hole components.

In Figure 5.3.9.1, you can see an example of what is possible to do with SMD on PCB. A computer, on a tiny board, for a few dollars.

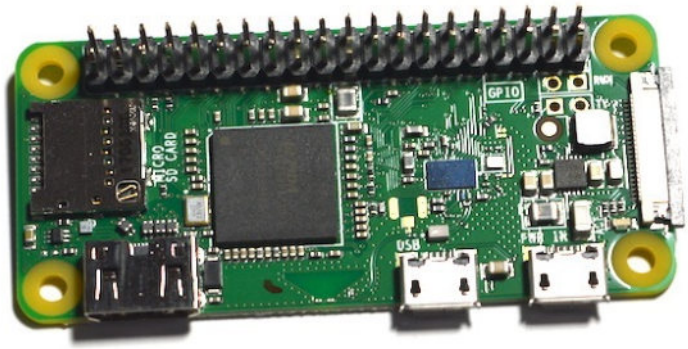


Figure 5.3.9.1: The Raspberry Pi Zero contains almost exclusively SMD components.

On this board are a highly integrated microprocessor, memory, communications, and connectors. Even the connectors are SMD. The only through-hole component is the pin header.

Creating something like this using through-hole components, if at all possible, would result in a board that was many times the size of the Raspberry Pi Zero and would cost many times more because most of the assembly would have to be done by hand.

While hobbyists prefer to work with TH components because they are easier to solder and repair, learning to work with SMD, at least the larger ones, is certainly possible.

In this book, you will learn how to create an SMD version of a PCB, in addition to the TH version.

3.10. Gold Fingers

Appropriately called 'Gold finders' are gold-plated connectors placed on the edge of a PCB. Gold fingers are useful for interconnecting one board to another. You can see an example in Figure 5.3.10.1; it shows the micro:bit educational single board computer.

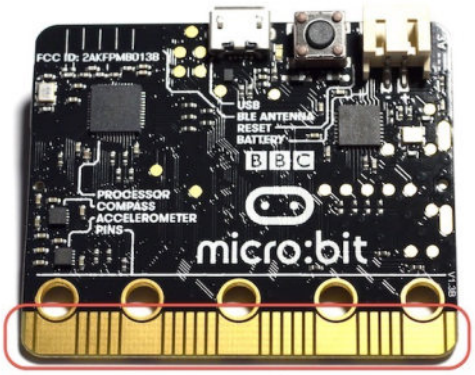


Figure 5.3.10.1: Gold fingers on a Micro:bit.

The micro:bit uses gold fingers to connect to other devices via a slot, like motor controllers and sensors. Gold fingers make it possible to attach and detach the PCB to a slot at least 1,000 times before they start to wear out.

3.11. Keep-out areas

A “keep out area” is what it sounds like: an area on the PCB that must be clear of components and perhaps even traces.

In Figure 5.3.11.1 I show three examples of devices that contain keep out areas.

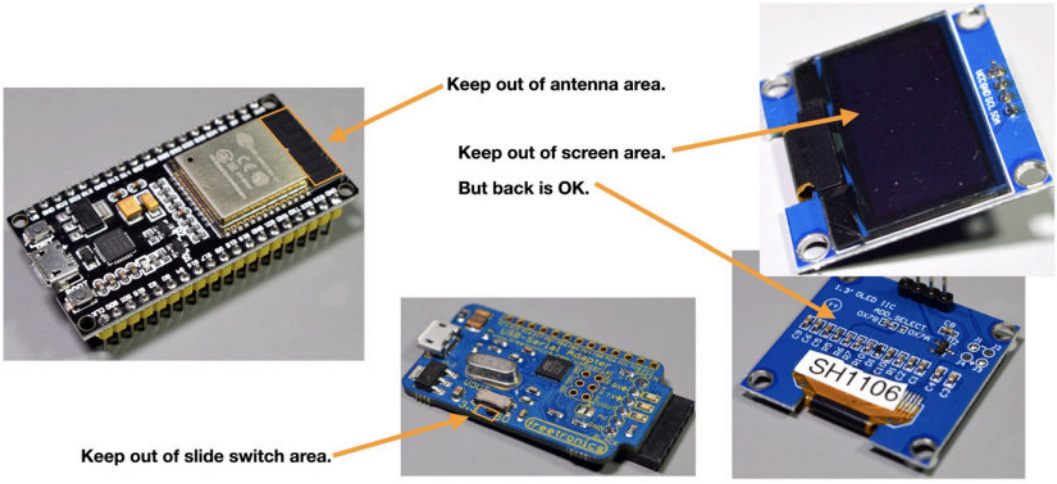


Figure 5.3.11.1: Examples of devices that contain keep out areas

CAD software, including KiCad, allows you to mark an area on the PCB as “keep-out.” In KiCad, you can also configure the keep-out area to prevent the user from adding specific or all types of elements, including footprints, tracks, vias, and cutouts. You can also tell KiCad to apply the keep-out rules to specific (or all) copper layers.

On the top left is an ESP32 development kit. The kit is based on an ESP32 module that contains an integrated antenna that requires a patch of PCB clear of components and other traces. You don’t want to add any other components in that area not to affect the antenna’s performance. Note that the keep-out area must include all copper layers, not just the front one where the antenna is.

On the right side of Figure 5.3.11.1, I show the back and front sides of a TFT screen that I use in my Arduino projects. The front side is where the TFT screen is placed. You can see that the screen’s ribbon connector is attached to a row of pads in the back of the PCB. We can mark a keep-out area in the front of the PCB only and allow for footprints and traces to be placed on the back of the PCB.

The last example, in the middle of Figure 5.3.11.1, is a UART interface with a voltage slide switch. The slide switch is oriented to its side. Even though the switch notch is tiny, it is still a good idea to mark the area below it as a keep-out area for footprints but allow traces. This way, there is no risk of placing a footprint by mistake and obstructing the travel path of the notch. However, we can still use that patch of PCB for tracks or vias.

With KiCad, you can create keep-out areas of any shape and configure them in almost any way needed.

3.12. Panel

To manufacture PCBs economically, manufacturers use machines that can work on large panels. Each panel can contain many copies of the same PCB. It is also possible to use clever algorithms that place different PCBs on the same panel so that the panel's capacity is fully utilized and that the individual cost of each PCB is reduced. This is how it is possible to have a single 'hobby' PCB manufactured for a few dollars. This panelization process is key to this reduction in costs.

In the example of Figure 5.3.12.1, a single panel contains four individual PCBs. The four PCBs are populated while still part of the panel using an automated pick and place machine. A pick and place machine is a robot that uses an arm to pick each component from a container and places it precisely on the pads. Once the components are on the board, the panel moves into the next step of the process, in which they are 'baked' and secured in place.

Figure 5.3.12.1: A panel with four individual PCBs.

Manufacturers utilize defined breakaway routes and points on the board to remove the PCBs from the panel to snap them off. In Figure 5.3.12.1, you can see the breakpoints along the edges of this PCB.

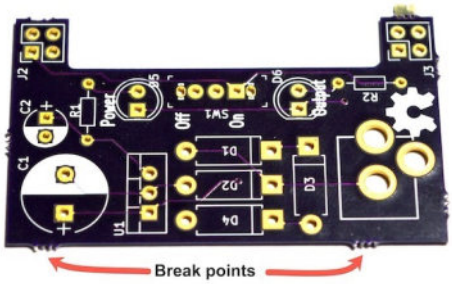


Figure 5.3.12.1: This PCB was part of a panel.

Using a drill, the manufacturer removed the substrate material in between the breakpoints. With a small amount of force, you can remove individual PCBs from the panel without damage.

3.13. Solder paste and paste stencil

Solder paste (or solder cream) is a soft and sticky material (at room temperature) applied on pads. The purpose of solder paste is to help attach an SMD component to the pad. Think of solder paste as ordinary solder. With solder, you will need a soldering iron to heat it, melt it, and apply it on a component pin already in place. With solder paste, you will first use a syringe (or one of the other application methods) to cover the pad, then place the component on the pad, and provide heat in the form of an oven to heat the paste and bond it with the pad and the component's plated area.

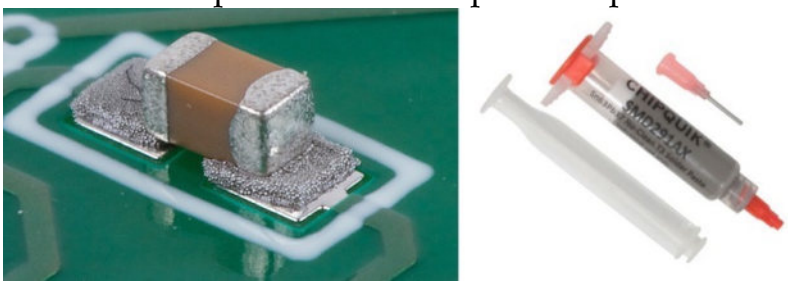


Figure 5.3.13.2: Solder paste in a syringe dispenser and an SMD component.

In Figure 5.3.13.2, you can see an example of a solder paste in a syringe dispenser that you can purchase from retailers like RS Components. Using the syringe equipped with a thin nozzle, you can manually deposit a small amount of solder paste on the pads. Using tweezers, you can place the component you want to attach on the solder paste. Because solder paste is

sticky (before it's baked), the component will adhere to it. After you have all the components you want on the board, you place the board in an oven to bake it. After the baking process is complete, the solder paste becomes solid. The SMD components will be mechanically secure and electrically connected to the pads.

Solder paste also comes in a tub, which is more appropriate for application to a board using a stencil (Figure 5.3.13.3). Stencils are helpful in large-scale productions.

Figure 5.3.13.3: Solder paste is a tub container.

A stencil, typically made of stainless steel, is cut to have openings of the exact size and the precise location of the board's pads. The technician will place the stencil over the board and then apply the paste to the openings. When the technician removes the stencil, the paste remains on the pads only.

Then, manually or using an automated pick and place machine, the components are placed on the pads and stick on them because of the paste. The last step is to bake the board in a reflow oven to solidify the paste.

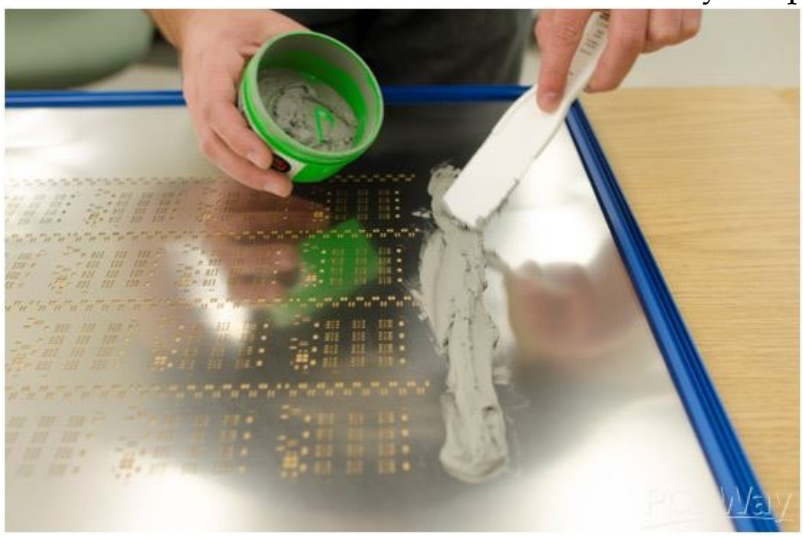


Figure 5.3.13.4: A stencil, with solder paste being applied using a squeegee (photo courtesy of Pcbnew.com).

A reflow oven is an industrial-sized machine used to complete attaching SMD components on a PCB. You can also purchase or make a reflow oven for use at home. People have even made reflow ovens for their projects using discarded toaster ovens. In either case, a reflow oven is designed to operate under a specific program that controls the amount of heat a board receives over time. This is important because the heat must be appropriate for converting the solder paste into good-quality electrical connections without causing damage to the board or the components on it.

3.14. Pick-and-place

Pick and place machines are robots that assemble the various components on the surface of a circuit board. When you contract a manufacturer to make your boards and populate them, they will be using a pick and place machine. You can see an example of a pick and place machine in Figure 5.3.14.1.

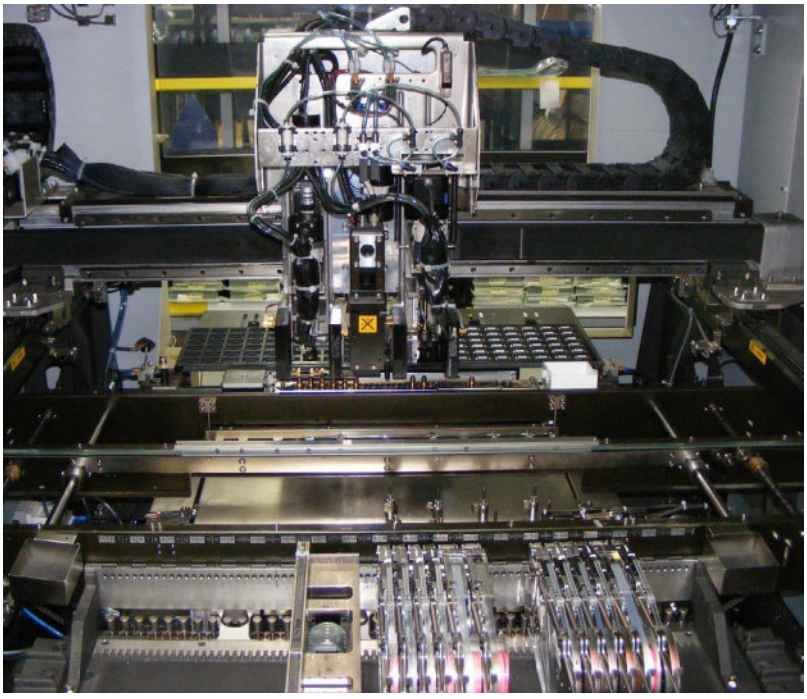


Figure 5.3.14.1: Figure 12.18: A large pick and place machine[4]

A typical pick and place machine, like the one in Figure 12.18, includes:

1. A repository of the various components that are to be placed on the board
2. A conveyor belt that brings in the boards.

[4] By Peripitus [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], from Wikimedia Commons

3. An inspection system composed of cameras that can optically recognize the board, components, and other guidance markings on the board.
4. A robotic arm that can pick a component from the repository and place it on the board (these arms are usually fitted with suction cups so they can pick and manipulate components).

Modern high-end machines are very versatile, optimized for short runs of complicated boards that employ artificial intelligence. These machines are designed to assemble and test boards autonomously, ensuring high levels of reliability.

Part 6: PCB design workflows

1. The KiCad Schematic Design Workflow

In the initial project earlier in this book, you learned about the PCB design workflow without getting into the details. It is time to take a closer look at the two parts of the process: the schematic design workflow and the layout workflow. In KiCad, Eeschema is the schematic design editor, and Pcbnew is the layout editor.

Let's learn about the workflow of creating a schematic in Eeschema. You can see it in Figure 6.1.1.

But before we get into it, I'd like to highlight this: I designed the workflow you see in Figure 6.1.1 to help you take your first few steps in PCB design. As you gain confidence and knowledge, you will develop your personal workflow. While you should follow the workflow in Figure 6.1.1, know that you are by no means' locked' in it. KiCad is very versatile and can accommodate your working style. It can also work in tandem with other tools in your toolchain, such as other CAD applications and autorouters.

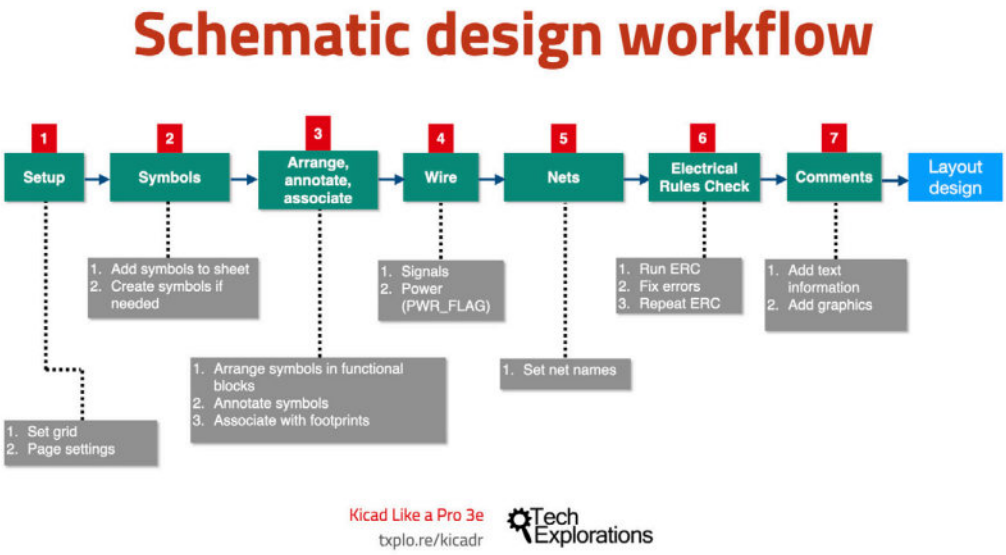


Figure 6.1.1: The schematic design workflow

In addition, engineering and design is a massively iterative process. Linear simply doesn't work. Yes, the workflow you see in Figure 6.1.1 is linear, but that's ok because you will abandon it in favor of an iterative model before you finish working with this book.

Let's have a look at the seven steps of the schematic design workflow.

1.1. Schematic Design Step 1: Setup

When you start to work on a new schematic in Eeschema, there are two things that you will want to do: configure the grid size and set up the page. The Setup in Eeschema is more straightforward in that it is in Pcbnew, as you will see.

To access the Preferences window on a Windows machine, click on Preferences, and the Preferences (Figure 6.1.1.1, left). On the Mac, click KiCad and Preferences (Figure 6.1.1.1, right). The Preferences window shows the Schematic editor options only when Eeschema is running, so go ahead and start Eeschema. Then, you will see the "Schematic Editor" group of tabs.

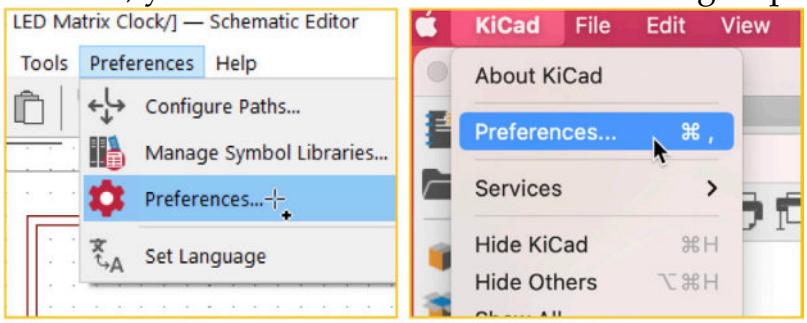


Figure 6.1.1.1: Open the Preferences window: Windows (left), MacOS (right).

You can set the grid options in the Display Options tab under "Schematic Editor", which you can find in the Preference menu (Figure 6.1.1.2).

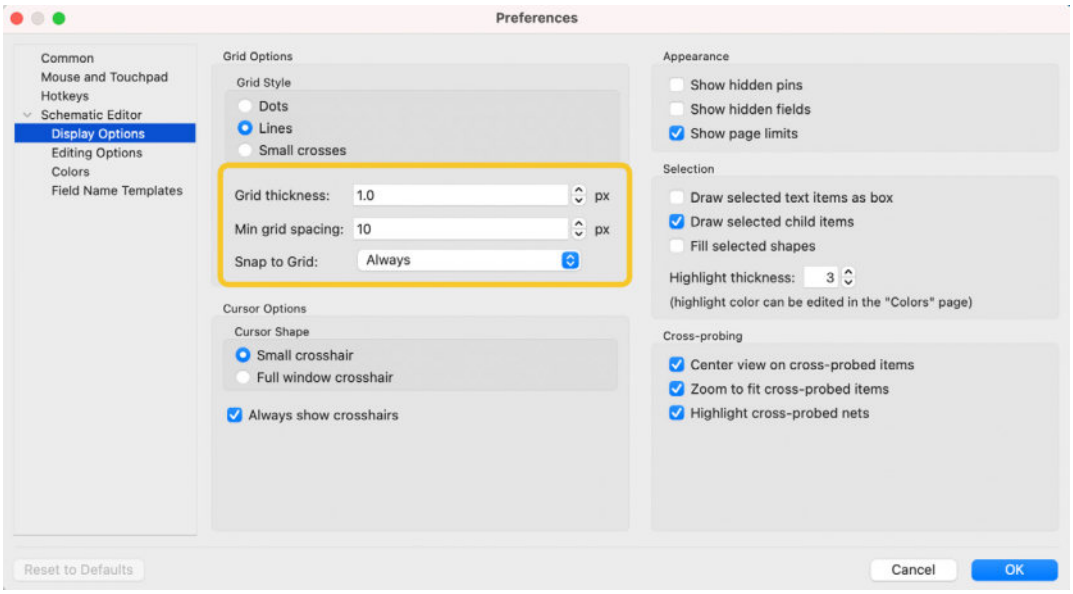


Figure 6.1.1.2: Grid options for the schematic editor.

You can set the grid thickness, minimum spacing, and snap to grid. The snap feature helps in the drawing process so that lines, pins and other elements align well. The default setting for the minimum grid spacing is 10 px, which I find suitable for most of my projects.

You can now close the Preferences page and take a few minutes to set up your page in the Page Settings window, available from the File menu (Figure 6.1.1.3).

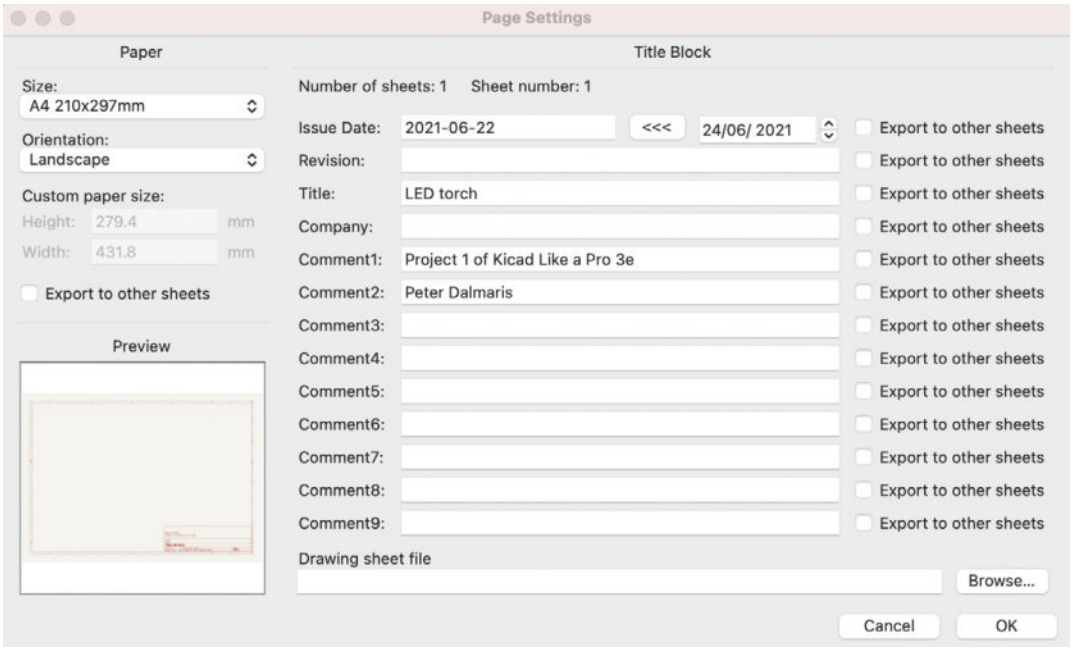


Figure 6.1.1.3: Page settings.

As in my example in (Figure 6.1.1.3), you should type in the details of your project, such as:

- The date (you can copy the current date into the field by clicking on the '<<< 'button).
- The revision number.
- A title.
- Comments that describe the purpose and functionality of the PCB.

When you print out your schematic (or export it as a PDF to share it with others), this information will also be included. In Pcbnew, you will find the same Page Settings window, as you will see in the next chapter.

1.2. Schematic Design Step 2: Symbols

The two most essential elements of schematic diagrams are the symbols and the electrical connections between the symbols. The symbols are graphical representations of real-life components. For example, Figure 6.1.2.1 shows examples of a resistor, led, and two switches in the schematic editor.

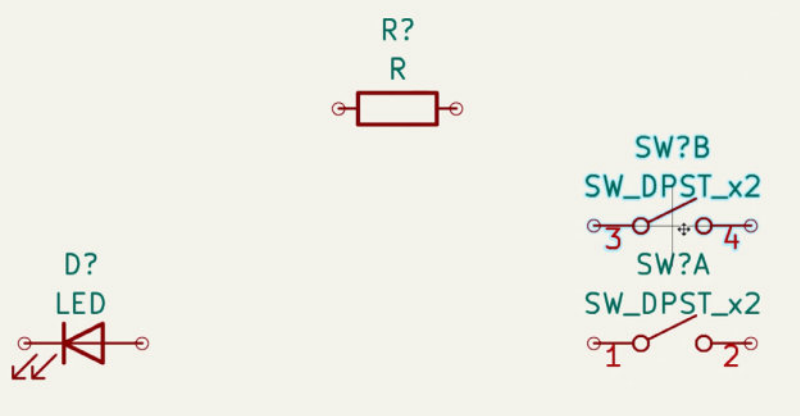


Figure 6.1.2.1: Example symbols in Eeschema.

Assuming you know what you are designing, the second step of the workflow involves getting all the needed symbols from the symbol chooser and adding them to the schematic sheet (Figure 6.1.2.2).

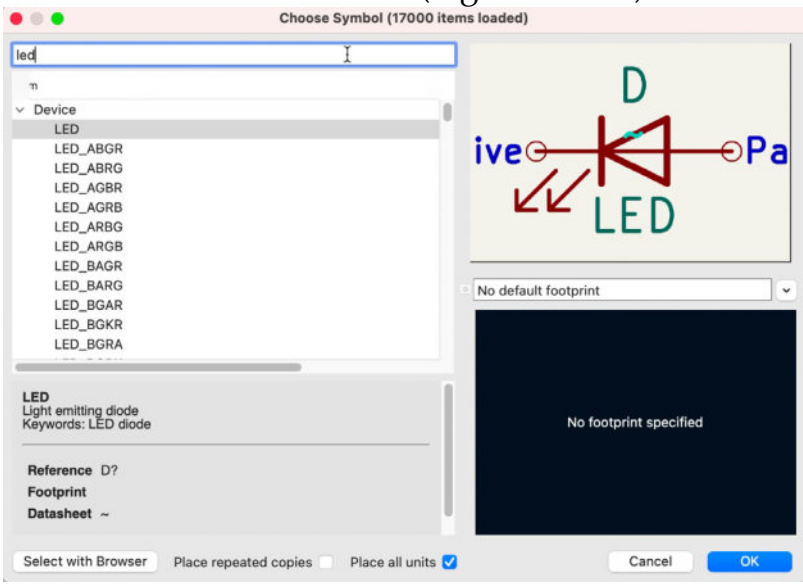


Figure 6.1.2.2: The Symbols chooser in Eeschema.

KiCad's schematic libraries contain a huge variety of symbols. On top of that, you can create your custom symbols and your libraries. Those custom libraries can include your custom symbols or symbols that you have collected from other sources.

In this step, simply get all the required symbols on the sheet. If any are missing, create them yourself (you will learn how to do that in this book), or find them by searching online. Once you have all of them, continue to step 3.

1.3. Schematic Design Step 3: AAA (Arrange, Annotate, Associate)

In step 3, you will move the symbols in place and prepare them for wiring. In Figure 6.1.3.1, I have placed the symbols in the approximate position in relation to the other symbols to wire them in the next step quickly.

I want to keep the wires short and prefer straight lines whenever possible to produce a more readable schematic.

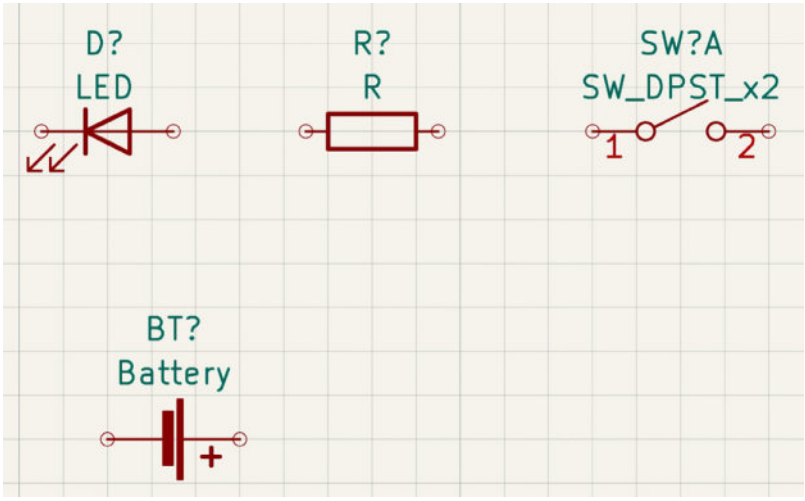


Figure 6.1.3.1: Symbols placed in the schematic editor, not annotated.

Before moving on to the wiring step (step 4), you should also annotate the symbols. Notice how the LED and the resistor have designators' D?' and 'R?' in Figure 6.1.3.1? The question marks indicate that these symbols have not been annotated. With annotation completed, the schematic will look like the example in Figure 6.1.3.2.

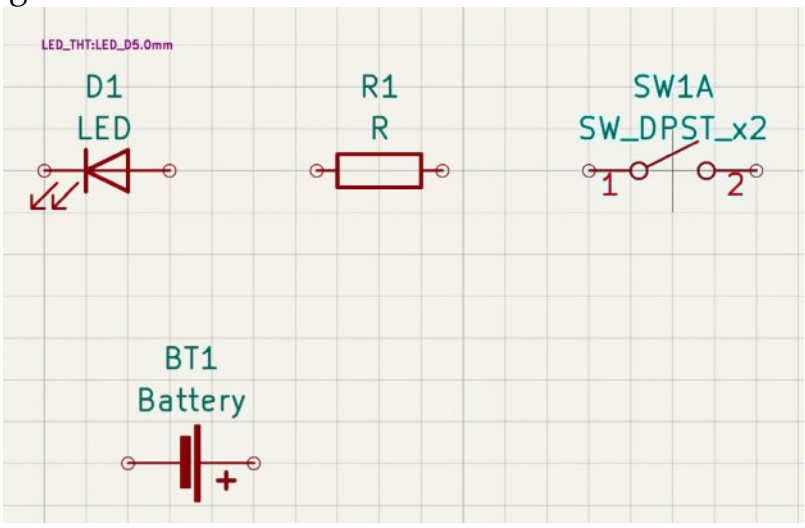


Figure 6.1.3.2: Symbols placed in the schematic editor, annotated.

KiCad can annotate all symbols with the click of a button, as you will see later.

To complete Step 4 of the workflow, you will associate each symbol with a footprint. In KiCad, you are free to associate a symbol with any footprint, even if the footprint has an incompatible PIN number and configuration. You can see a completed associations table below.

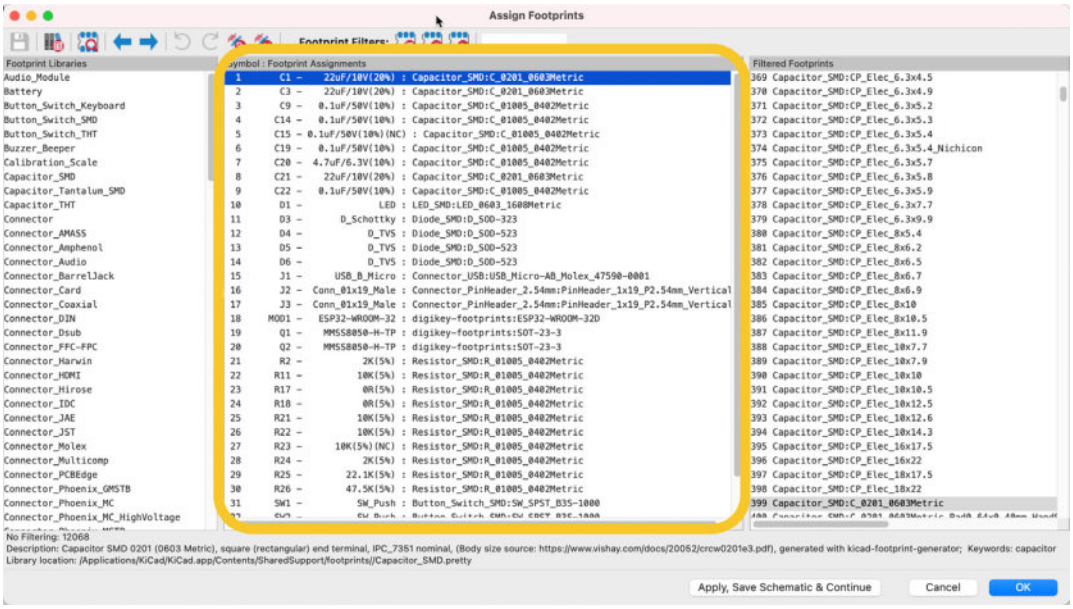


Figure 6.1.3.3: Symbols associated with footprints.

The example above comes from one of the projects in this book. In the association's table (middle pane highlighted by the orange box), the symbols appear in the left column, and their assigned footprints in the left. You will learn how to use the "Assign Footprints" window later in this course.

With your schematic symbols annotated, you can move on to Step 4, wiring.

1.4. Schematic Design Step 4: Wire

In step 4, you will connect the symbols using wires. Each wire is attached to a symbol pin and creates a net. In the example of Figure 6.1.4.1, the green wires connect the LED and the resistor to the 5V voltage source and the GND. The 5V and GND symbols are special 'power' symbols. This schematic contains three wires, so it includes three nets.

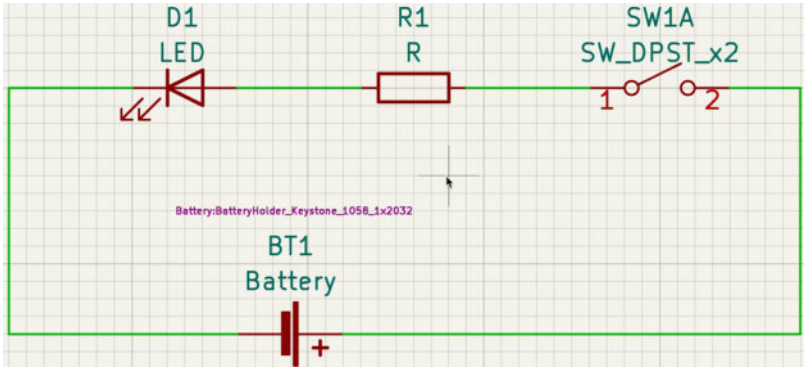


Figure 6.1.4.1: Symbol pins connected with wires.

Understanding nets is essential because of what you can do with them in Pcbnew. The examples here don't have a custom net name, but they have an automatically assigned name that Eeschema has generated. With or without a custom net name, this schematic is fully wired, so you can continue in the next

step of the workflow, where you will assign custom names for these nets, or at least the most important nets.

1.5. Schematic Design Step 5: Nets

In step 5, you will name your schematic nets, at least the most important ones. Important nets are, for example, those that connect to the GND and 5V symbols. Giving them a custom name to replace the generic one assigned by Eeschema will make it easier to work with it in Pcbnew. Typically, power nets are implemented with wider traces to allow them to accommodate higher currents. Other examples of 'important' nets are those that implement antennas and data or address buses. In either case, the geometry of those nets, once implemented as traces, is essential, and having a custom name makes their manipulation easier.

The analogy from the programming world is this: imagine you have a variable used as a counter to the number of users participating in a forum discussion. This variable could have any time you like, as long as it is valid. You could call it 'number_of_forum_participants' or 'var32'. Which one would you rather use?

In Figure 6.1.5.1, you can see two labels attached to the wires that connect the circuit to the GND and 5V symbols.

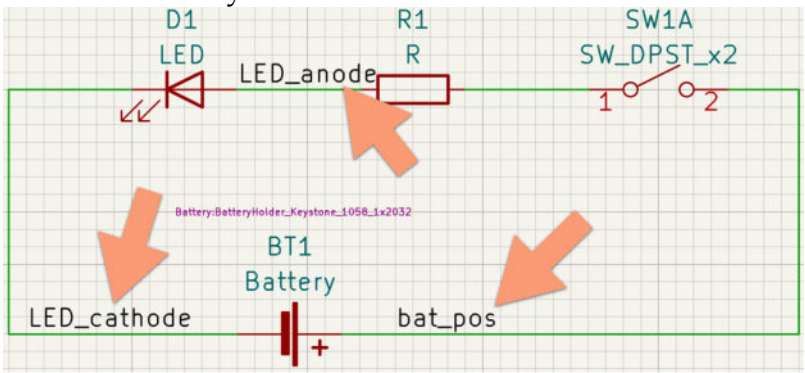


Figure 6.1.5.1: This schematic contains named nets.

1.6. Schematic Design Step 6: Electrical Rules Check

In more realistic circuits than the one with the LED and resistor symbols we have been using as an example, it is a good habit to check for errors regularly. In programming, the equivalent is to periodically run the compiler every time you complete a new code segment. The compiler will produce messages to draw your attention to defects. You will need to fix those defects before you continue to write new code.

In Eeschema, this kind of check is done by the Electrical Rules Check utility (ERC). The ERC will check for various error conditions, such as unconnected pins, power pins, problems connecting to incompatible pins, etc.

In Figure 6.1.6.1, the ERC revealed two problems with this schematic. This is one of the most common problems that you will come across. You learned about this in Part 2, in the section with the title 'Electrical rules check (ERC).'

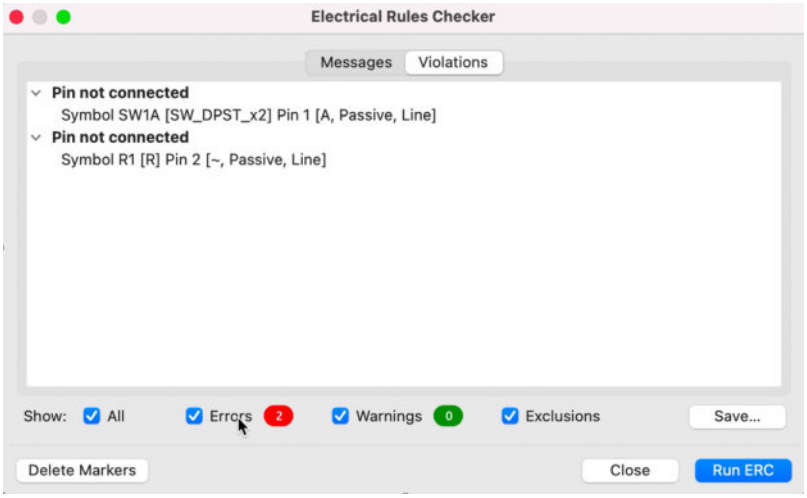


Figure 6.1.6.1: This ERC has revealed two violations.

The Electrical Rules Checker tool in Eeschema is customizable. You can adjust almost every aspect of its operation to fit your requirements. You can configure the ERC from the schematic setup window (Figure 6.1.6.2). You can learn how to do this in a dedicated chapter later in this book.

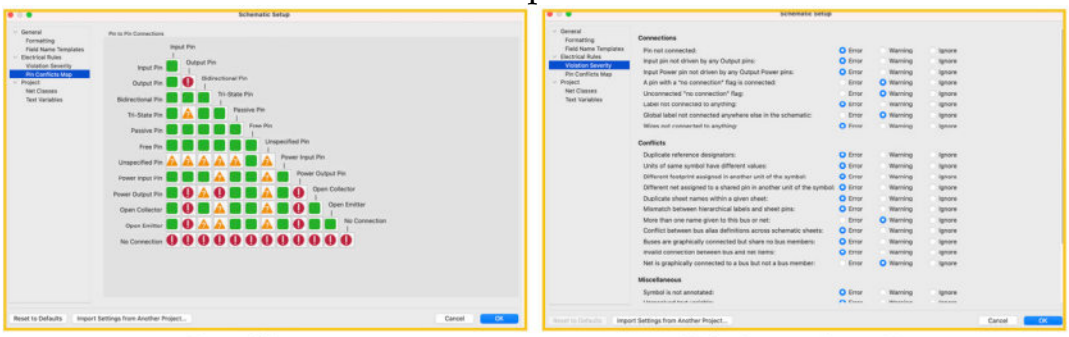


Figure 6.1.6.2: The Electrical Rules Checker configuration options.

1.7. Schematic Design Step 7: Comments and Graphics

Let's continue with the programming analogy. Just like in programming, comments are essential to increase the value of your code; similarly, in CAD design and in particular in schematic and layout design; you can use comments for the same purpose. I try to be liberal with

comments, especially when I work on schematics and layouts that I plan to share with other people.

Comments come in text labels that highlight schematic features and graphics, such as lines and boxes, that help group segments of the schematic into functional components.

You can see an example in 6.1.7.1. I may have gone a bit overboard with my comments. However, the result is that for the reader/learner, it is evident what each group of symbols is and how it relates to the other symbols.

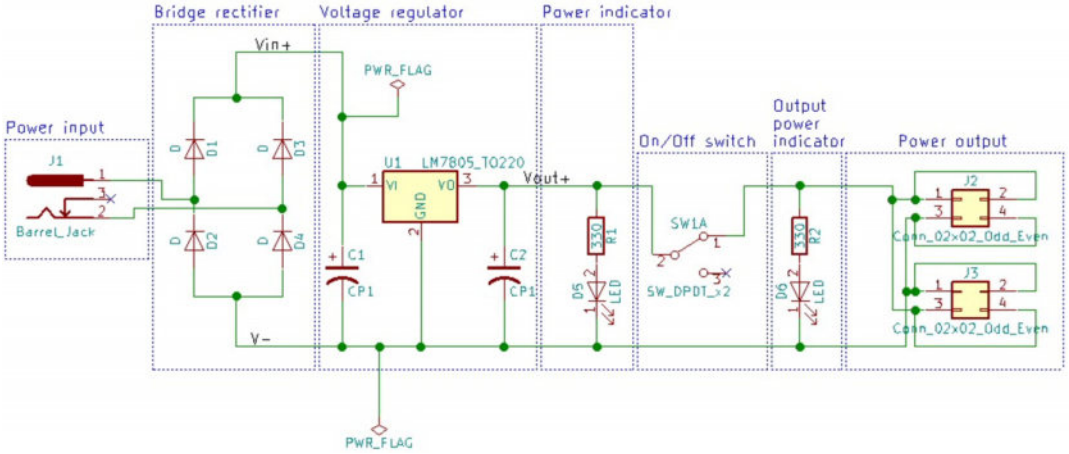


Figure 6.1.7.1: A schematic with several short comments.

2. The KiCad Layout Design Workflow

Earlier in this book, you saw a high-level view of the process of creating a PCB in KiCad. The process starts with the design of the schematic diagram that describes your circuit using Eeschema and concludes with the layout of the physical components of the PCB using Pcbnew. KiCad contains other tools that assist us in this process, like the footprint and component editors and Cvpcb, which allows us to associate components to footprints.

In this chapter, we'll focus on the PCB layout workflow. In KiCad, this is the step that involves Pcbnew. The layout step is the one that uniquely shapes the final 'real world' product: what it will look like, how well it will function, how 'manufacturable' it will be, how durable it will be. All this depends on the layout.

The process I describe in this chapter applies to any PCB layout tool. However, I will be using KiCad terminology to make it easier for the reader to associate the process with the functionalities inside Pcbnew.

You can see the process in Figure 6.2.1. For simplicity, as I mentioned in the chapter on the schematic design workflow, I have rendered it to look linear. In reality, the process is iterative. It is often necessary to move, for example, from step 3 back to step 1 to adjust the grid so that parts fit better in the available space. This diagram will help you understand the steps through which every PCB design has to go through. In this chapter, I will describe each of those steps. Once you apply this process to the projects in later in this book, it will become 'natural.'

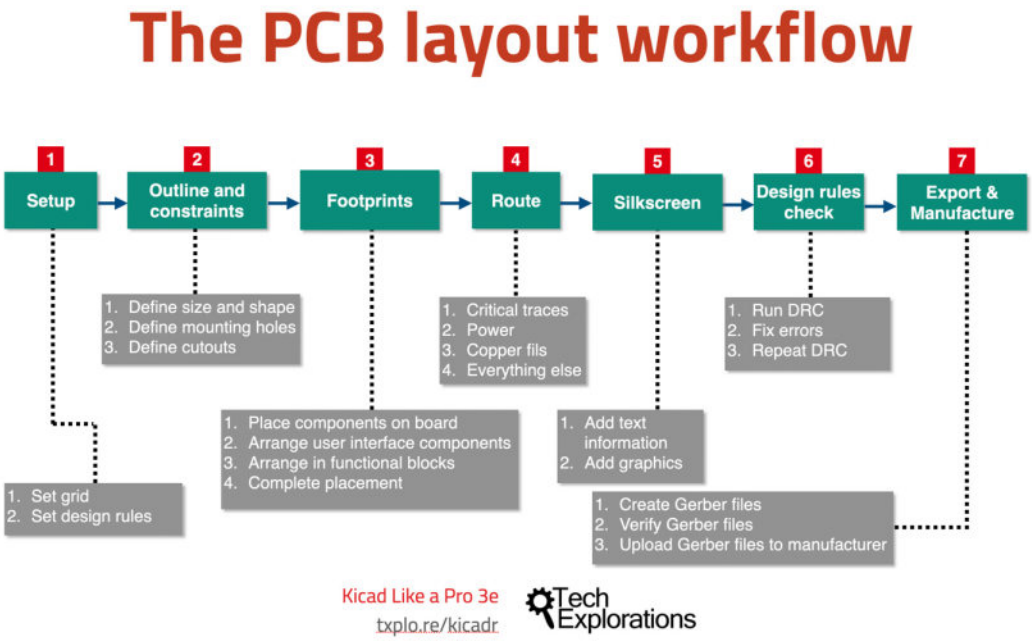


Figure 6.2.1: The layout design workflow

2.1. Layout Design Step 1: Setup

In the setup step, you configure the Pcbnew drawing grid size, set the required copper layers, and set the design rules for the layout. The grid size and number of layers depend on the dimensions and complexity of your PCB. The design rules are governed mainly by the constraints set by your PCB manufacturer but also by the requirements of your project. Let's start with the grid first.

The grid assists with the placement of footprints, drawing of traces, and the drawing of the cutout (boundary) of your PCB. Pcbnew provides multiple grid sizes, but you can also define your custom size.

When you start a new project, select the grid size that is most appropriate for the design of the board's outline and the placement of the footprints. Typically, we will choose a larger grid size for the board outline and a smaller one for the footprints. We may go for a third, smaller grid size to help draw the traces as this often needs to negotiate tiny details on the PCB, like going around pads and vias.

Larger grids produce more coarse drawings.

Let's say that you want to produce a rectangular PCB that measures 50 mm by 30 mm. On it, you want to place a few resistors, LEDs, and headers. A reasonable grid choice for the outline and the larger features is 1.27 mm, and half that (0.635 mm) for the components. When you start work on the traces

you can consider going one level down from 0.635 mm, to 0.508 mm or even 0.2540 mm if there are too many pads to connect.

The larger grid will allow you to draw the cutout of the board in precise 1.27 mm segments. With this grid, you will be able to draw the long side (50 mm) with 39 x 1.27 mm segments, for a total length of 49.53 mm, or with 40 x 1.27 mm segments for a total length of 50.8 mm. If you need to go for an exact 50 mm side, you can define a custom grid side, say 0.5 mm, using the Grid Setting dialog box (Figure 6.2.1.2), and select it from the Grid drop-down menu (Figure 6.2.1.1).

Once you are finished drawing the cutout, you can switch to the smaller Grid side and continue placing the footprints inside the board.

KiCad makes it easy to switch between two grid sizes by using the grid fast switching shortcuts. The default shortcuts are Alt-1 and Alt-2, although I have changed this to something else to avoid conflict with the Ubuntu operating system's Alt-[X] shortcut that allows fast switching between applications (where 'X' is the number of the application on the taskbar).

To define which grid size you want to work with, use the Grid drop-down menu (Figure 6.2.1.1).

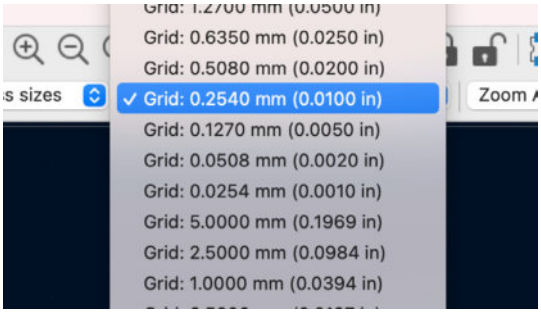


Figure 6.2.1.1: Grid size drop-down menu.

To define the two Grid Fast Switching sizes and to set your custom grid sizes, open the Grid Setting dialog box by clicking on View, Grid Properties (Figure 6.2.1.2).

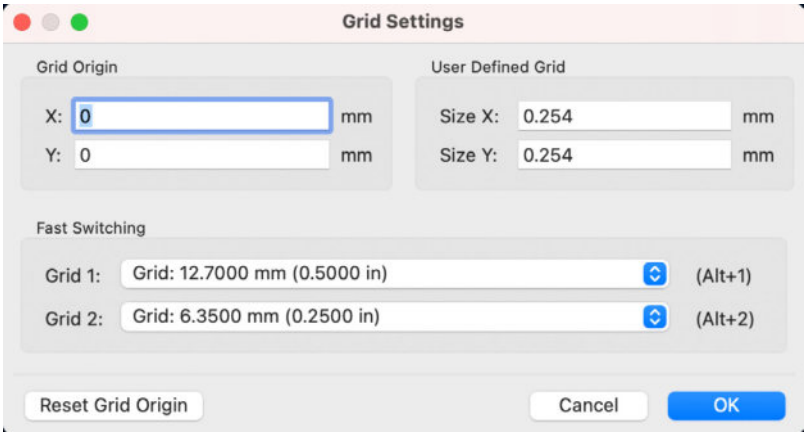


Figure 6.2.1.2: Grid size fast-switch settings.

After the grid size, you need to define and confirm the number of layers you want to use and the design rules. Regarding the number of layers, most hobbyist projects seem to be best implemented with two layers. The best online manufacturers have optimized their processes to work with two layers, both from a cost and quality perspective. Even if you create a single-layer PCB design, these manufacturers will use a two-layer process on it, and therefore the cost will be the same. If you make a more complicated PCB, you can set up Pcbnew for more than two layers. You can select the number of layers for your PCB from the Board Setup dialog box that you can reach from the File menu. Open the Board Setup dialog box and click on “Physical Stackup” under “Board Stackup” (Figure 6.2.1.3).

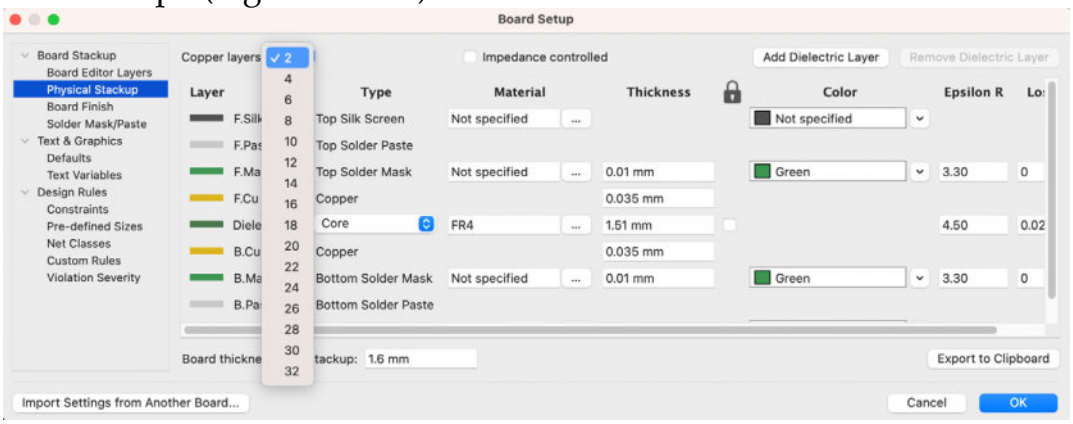


Figure 6.2.1.3: Copper layers drop-down menu.

The last item in the Setup to-do list is to define and confirm the Design Rules. Your project’s technical requirements and your preferred manufacturer’s capabilities and guidelines dictate the PCB design rules. To set up your project's design rules, open the Design Rules Editor from the Setup menu. In Figure 6.2.1.4, you can see the constraints tab in the Board Setup window with the default values for one of the projects in this book.

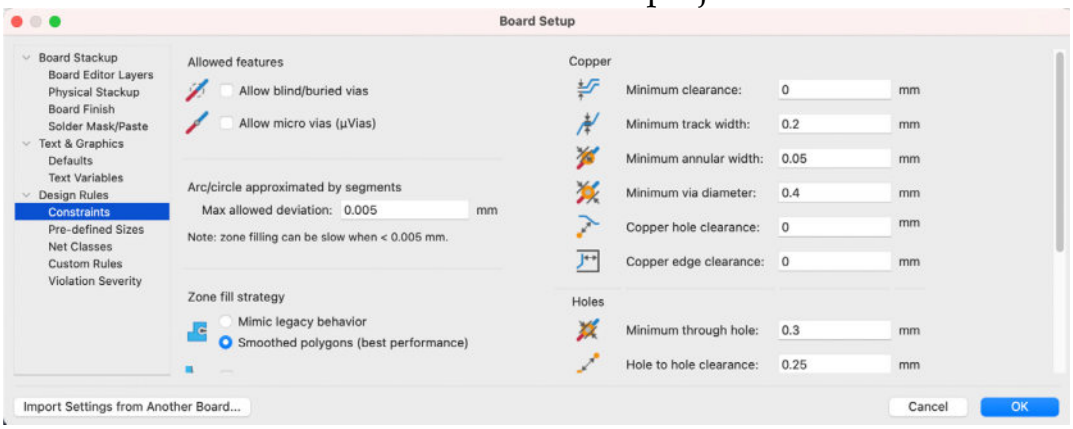


Figure 6.2.1.4: The Board Setup menu.

When starting a new project, it is a good habit to confirm that, at the very least, the design rules are compatible with the guidelines set by your

preferred manufacturer. Many online manufacturers publish these guidelines on their websites. For example, OSHPark has a web page[5] with design rule information specifically for KiCad, where we learn that the minimum track (trace) width is 0.006' and the minimum via diameter is 0.027'. PCBWay also publishes this information on a page on their website,[6] where we learn that their minimum track (trace) width is 0.1 mm and minimum drill size is 0.2 mm. Beware of the units that manufacturers report these numbers as they may be in imperial units (inches), metric (mm), or mil.

You should ensure that the values in the Design Rules editor, both in the Net Classes Editor tab and in the Global Design Rules tab, are equal or larger to those that your manufacturer specifies as a minimum. If you want to manufacture your boards with multiple manufacturers, you must ensure that your design rules comply with the largest minimum of their requirements.

The default values in the Design Rules Editor are larger than the minimum values of both OSHPark and PCBWay, so I usually don't make any changes to them. However, I add at least one new row in the Net Class Editor tab to define larger track widths and vias for power tracks as these convert more current than the default (signal) tracks. You will learn how to do this in the projects. There is also a recipe in Part 5 where you can learn how to do this quickly.

2.2. Layout Design Step 2: Outline and constraints

The next step in the process is the definition of the board outline. This outline defines the shape of your board. It is good practice to define the shape of your board before you add any components to it so that you can ensure that it will fit properly within the confines of a project box or other mechanical constraints.

Apart from defining the shape and the dimensions of your PCB, this is the step where you must also define fixed features such as mounting holes and cutouts. Again, these items must match your project box or other external mechanical constraints. If you don't deal with these issues now, you will likely have to relocate components and traces later, a much more tedious exercise. In Figure 6.2.2.1, you can see the board outline from one of the projects in this

[5] https://docs.oshpark.com/design-tools/kicad/kicad-design-rules/
Accessed November 19, 2018

[6] https://www.pcbway.com/capabilities.html
Accessed November 19, 2018

book. I have unchecked all layers except for Edge.Cuts to make it easier to see the yellow line of the board outline.

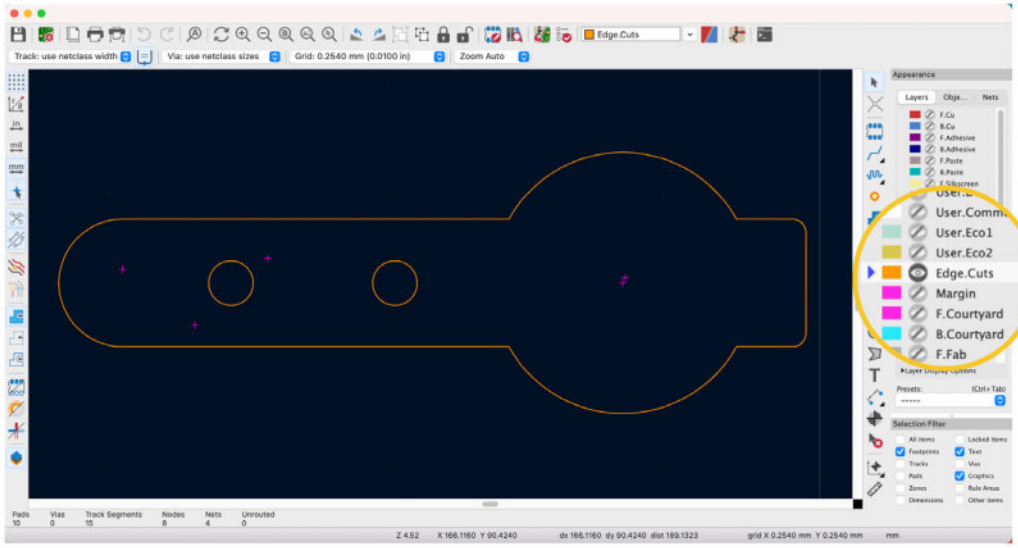


Figure 6.2.2.4: Drawing the board cutout in the Edge.Cuts layer.

To define the outline and the cutouts of a board, select the Edge.Cuts layer from the layers manager and use the graphics tools to draw it. As mentioned in the Setup section, I prefer to use a large grid size while outlining.

The easiest shape you can draw for your PCB is rectangular. However, using Pcbnew's Arc and Circle graphics tools, you can create elaborate non-rectangular shapes that will make your PCB stand out. In the projects in this book, we'll use these tools to create rounded corners and other interesting features.

To create mounting holes and cutouts, you can choose one of a couple of methods. You can make them in the Edge.Cuts layer or using pads during the component placement step. See an example in Figure 6.2.2.1.

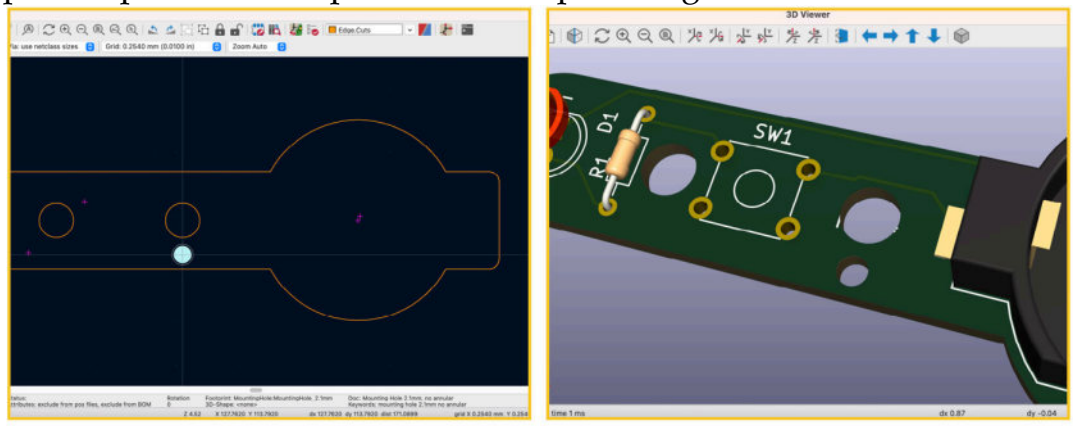


Figure 6.2.2.1: Cutout drawing in Pcbnew (left), and its 3D rendering (right).

In this example, I have created a simple rectangular PCB. I used the circle graphics tool to create two round openings (screw holes) and the

polygon tool to create a small rectangular opening. You can learn more about how to create openings in the projects later in this book.

2.3. Layout Design Step 3: Place footprints

After we have defined the mechanical characteristics of our PCB, we can proceed by placing the components (called 'footprints' in Pcbnew) on it. Like everything in engineering (and in life, in general), a good amount of thinking and planning here will pay off dividends later in the form of fewer errors and the need to move things around to fix those errors.

When you import the project footprints to PCBnew by reading the netlist file, all of the footprints are arranged in a matrix adjacent to each other.

Continue by placing in position the footprints that make up the user interface of your PCB. These are things like connectors, indicator LEDs, and buttons. You can see an example of this placement process in Figure 6.2.3.1.

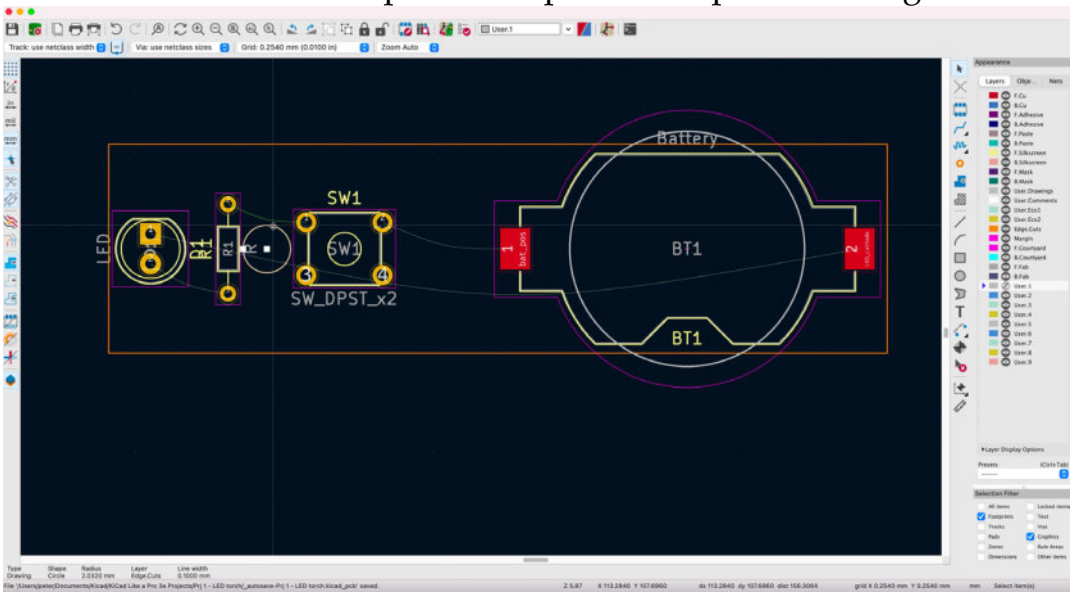


Figure 6.2.3.1: An example outcome of the component/footprint placement step.

In this example, I started the placement process by positioning the user interface components along the board's edges. You should think carefully about where to place those components. For example, at the top left of the board, you can see the barrel connector. A cable will plug into this component. Therefore there is a requirement here to ensure sufficient space around the board for the cable. I could have placed the connector facing in the opposite direction, but then the cable would interfere with my work on the breadboard. In Figure 6.2.3.2, you can see how the barrel connector in the top left corner of the PCB makes it easy to connect the power supply without obstructing the use of the breadboard.

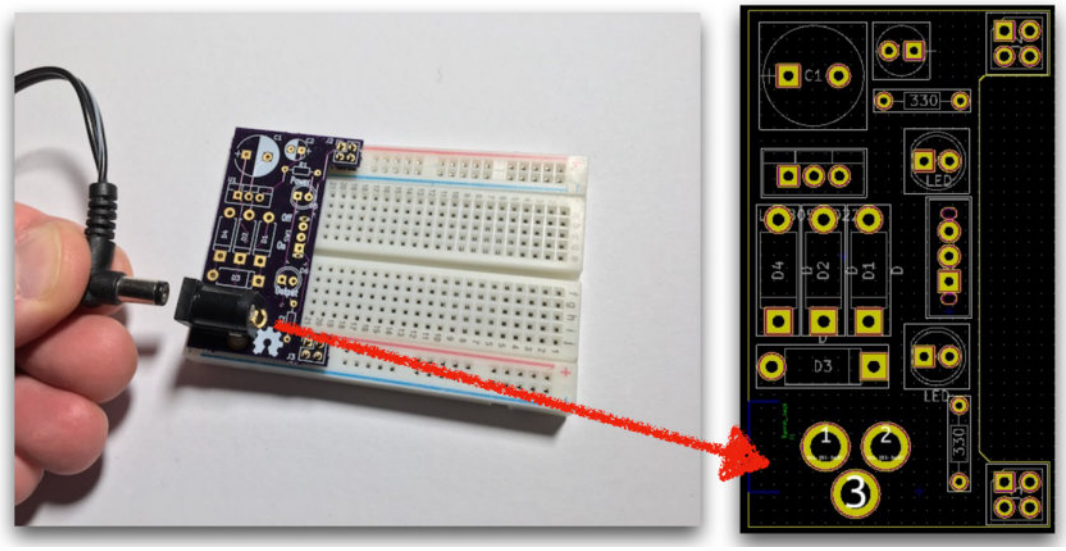


Figure 6.2.3.2: Consider the user interface requirements of your PCB carefully.

Other user interface parts are the mini slide switch, the two indicator LEDs, and the headers. The headers consist of the mean by which the power supply PCB plugs into the breadboard, so the geometry of the breadboard severely restricts their positions. I placed these components first and locked them in place.

I have more freedom to decide the position of the slide switch. My final decision was guided by the principle of placing UI components along the edges of the PCB for easier access and ergonomics. It would be easier to reach the switch if it was away from bulky components (like the large capacitor) and closer to the breadboard. I placed the two LEDs around the switch after I had locked the switch in place. Apart from better ergonomics, the LEDs are electrically connected to the switch, so it makes sense to place them close as this results in shorter traces.

Once I had finalized the positioning of the UI components, I continued with everything else. I followed these principles:

1. Components that are functionally related should stay close to each other.
2. Shorter traces are better.
3. Consider how the placement will affect assembly.
4. Consider component manufacturer specifications.

The four diodes belong to the same functional block (the bridge rectifier), so I placed them as closely as possible. The same applies to the two capacitors and the voltage regulator (the regulator stage). Finally, I positioned the current limiting resistors close to their LED for the same reason.

In this example, there is no particular manufacturer specification that I had to take into consideration other than providing sufficient space for each

component. In other cases, however, you may have to deal with components that need, for example, specific provisions for removing excess heat. This can be done by providing additional space for a heat sink or providing thermal vias for the same purpose. Thermal vias are vias that are not connected to a trace. They help move heat away from a component, such as an integrated circuit, through a die-attached paddle between the via and the component. If the manufacturer specifies this as a requirement, you should implement the heat vias or other provisions in this step.

With the components placed on the board, the next step of the process is routing the traces.

2.4. Layout Design Step 4a: Route

With the component placement step complete, you can move on to the next step, routing the traces. This step involves the drawing of the copper connections between the pads. To implement the traces between the pads, you can follow this process:

1. Start with any critical traces. This could include signal traces that have specific shape and length requirements, like an onboard antenna.
2. Continue with power traces.
3. Finish with the rest of the traces.

Let's have a look at an example of a critical trace. In Figure 6.2.4.1, you can see the front and back view of the Micro:bit board. The Micro:bit includes a trace Bluetooth antenna. You can see its trace in the top left corner of the front of the board (left image). Because the antenna has strict geometrical specifications for it to operate correctly (and legally), it is the first trace placed on the board. In the right image of Figure 6.2.4.1, you can see the back of the board. In the top right corner of the back of the board, you can see that the antenna area has nothing on it. No components and no traces. This is a 'keep out' zone, an area that we can define on a board to ensure that the autorouter or we cannot place anything there. Doing so would affect the way that the antenna works.

To learn how to create a keep-out area, you can refer to the relevant chapter.

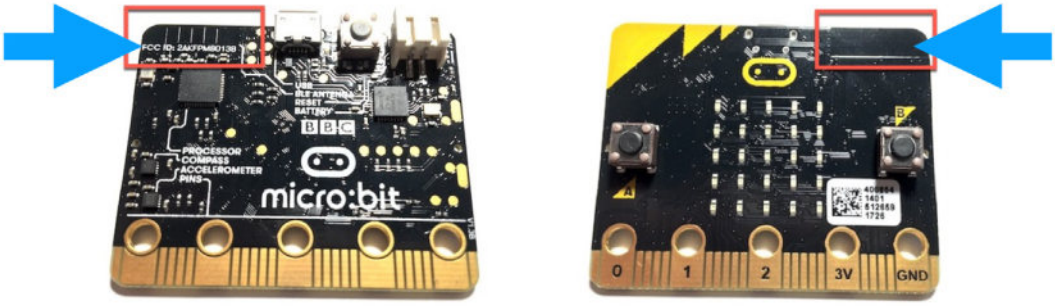


Figure 6.2.4.1: The integrated antenna in this Micro:bit is a critical trace.

Once you have taken care of critical traces, if any, continue with power traces. Power traces travel throughout your circuit to feed it with power, and as such, they convey a higher current than signal traces. For this reason, it is appropriate to design power traces (GND, 5V, 3.3V, etc.) so that they are wider than typical signal traces. Power traces should be around 0.30 mm to 0.40 mm in width for low voltage and low power consumption boards. The trace width depends on a few variables, and you can learn more about it in the Trace Width Calculator recipe. You can learn about using net design rules to automatically define the width of a trace in a dedicated chapter.

In Figure 6.2.4.2 you can see an example of a fully routed board.

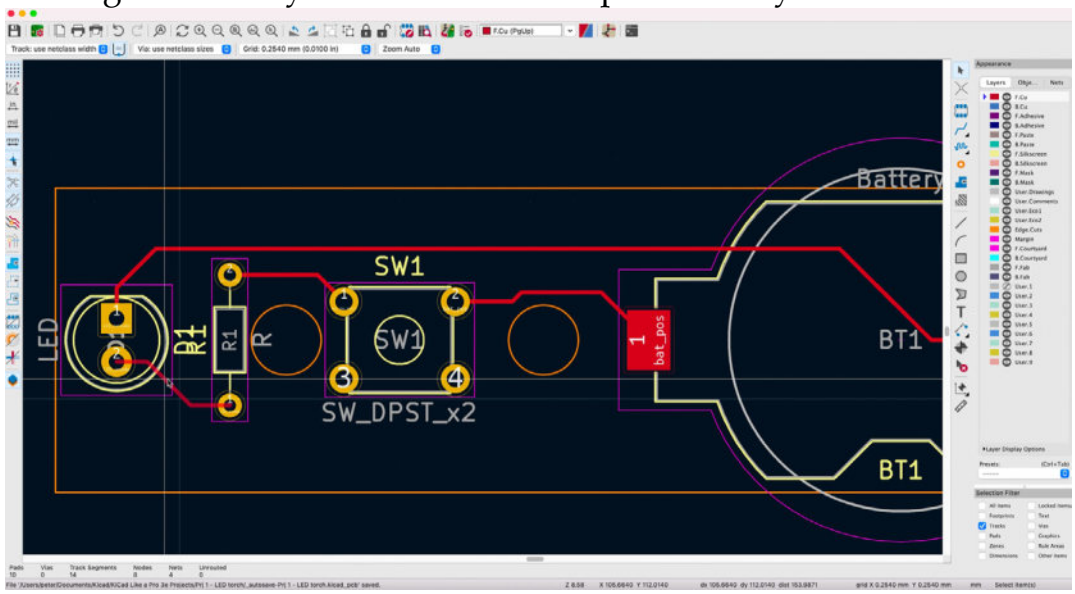


Figure 6.2.4.2: A fully routed board.

After you have completed the routing of the power traces, you can continue with the rest of the traces. For larger boards, you can set up an autorouter that can save you some time. For smaller boards, you can finish routing manually. You can learn about the autorouter in the relevant recipe.

Before continuing to the next step, run the Design Rules Check process to ensure no defects have been introduced.

2.5. Layout Design Step 4b: Copper fills

This step is not strictly necessary, and often designers decide to skip it. Copper fill is an area on the board that is fully covered with copper.

Typically, copper fills (also known as 'copper pours') create a ground plane, a contiguous mass of copper connected to electrical ground. Similar to a ground plane, you can create copper planes connected to a voltage level. If your PCB draws power from a battery, then the ground plane is connected to the negative electrode of the battery, and a voltage plane is connected to the positive electrode of the battery.

When a copper fill is created, pads can be connected to the fill using a small number of very short traces called 'thermal reliefs' (or 'thermals' for short). You can see an example of this in Figure 5.2.4.3.

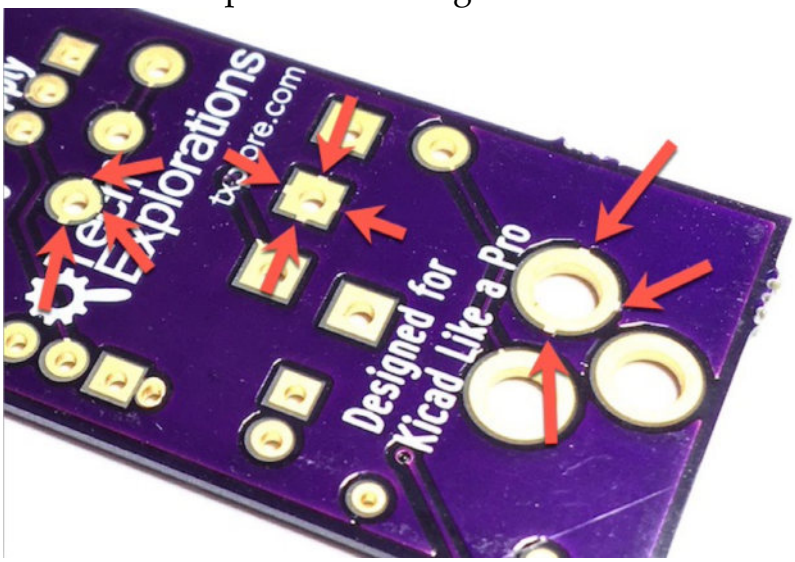


Figure 5.2.4.3: Arrows show where GND pads use thermals to connect to the GND plane.

In this example (the breadboard power supply board project from this book), you can see how multiple GND pads are connected to the ground plane in the back of the PCB using up to four short traces. The purpose of the thermals is to help with the soldering of the components on the pad. If the pads were connected to the plane with a full pour of copper, the soldering iron's heat would dissipate into the copper fill too fast, and the pad would not be able to reach a temperature suitable for the solder melt. To reduce the heat dissipation speed, the thermals ensure electrical conductivity while managing the heat from the soldering iron.

We refer to the distance between the copper fill and traces that belong to a different net as 'backoff' or 'standoff.'

Copper fills may be made to be solid or with a pattern like a 'cherry pie lattice.' Modern copper pours are almost always solid. In the past, cherry pie lattice or hatched patterns were used to prevent warping, but this does not

seem to be a problem anymore. I have never experienced warping in my boards using a solid copper fill.

The benefits of using ground copper planes are:

1. They offer a degree of protection against electromagnetic interference
2. They help to dissipate heat produced by the board components
3. They enforce the discipline of placing signal traces on the top layer and connecting all ground pads to the bottom ground copper plane using vias.

To learn how to create a copper fill, please refer to the relevant recipe in Part 5. As you will see, the process is very similar to creating a keep-out zone.

A copper fill is made once all routing is completed. In a typical 2-layer board, a ground plane is created in the bottom of the PCB, and often a V+ (say, 5 V) copper fill is created on the top layer. Suppose your board uses multiple layers and multiple voltages (like 3.3 V or 5 V). In that case, another option is to use the bottom layer for the ground plane, one of the middle layers for the positive voltages, and the top layer for the signals.

Before continuing to the next step, run the Design Rules Check process to ensure no defects have been introduced.

2.6. Layout Design Step 5: Silkscreen

Step five of the PCB design process is the silkscreen artwork. Silkscreen artwork can be placed on the top and bottom layers. KiCad offers two special layers dedicated to the silkscreen: 'F.SilkS' and 'B.SilkS.' In general, the silkscreen artwork involves the following elements:

1. Descriptions of pads (i.e., what is the role of each pad). This is done using text characters.
2. A name and version number of the board, also in text characters.
3. Your logo and other graphics you may want to include.
4. Other instructions that may assist the end-user include names, models, and values of components, input and output voltage levels, email addresses, or a website to look for more information.

Let's look at an example of these elements in the board you can see in Figure 6.2.5.1.

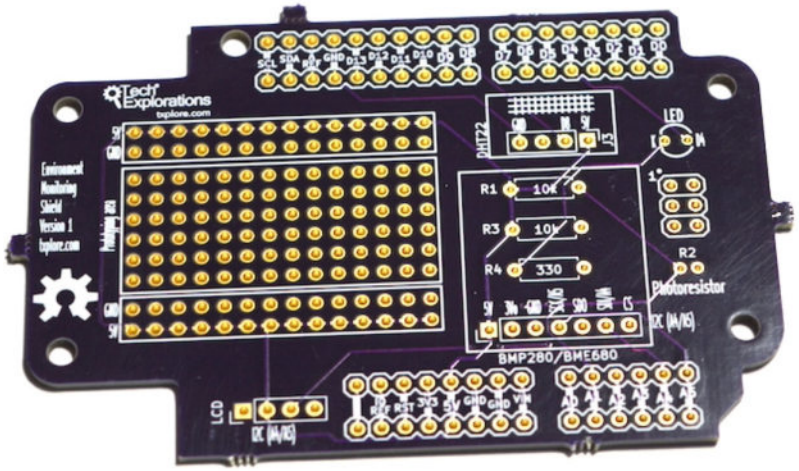


Figure 6.2.5.1: The silkscreen information on this board improves its usability.

The board in Figure 6.2.5.1 is one I designed for one of my Arduino courses. I created it as an Arduino shield. It contains a prototyping board and has provision for a DHT22 sensor, a BMP280 sensor, a photoresistor, and LED, and it exposes the I2C interface to connect an LCD screen. When fully assembled and stacked on an Arduino Uno and an Ethernet Shield, it looks like the example in Figure 6.2.5.2.

Figure 6.2.5.2: The Environment Shield in operation.

In Figure 6.2.5.1, you can see the contents of the top layer silkscreen in white ink. The Tech Explorations logo, with a URL, and the 'open-source hardware' logo appear on the left of the board. The name of the board and its version appear at the left edge of the board. Each pad has text that describes its purpose; the BMP280 and DHT22 pads are marked. There is also information on the values of the resistors, and the cathode of the LED is marked. All this information will help the end-user to assemble the board without the need for a reference document.

The prototyping area is also marked. The row of pins that convey the GND and 5V levels are marked. The bottom layer can also have a silkscreen where you can provide additional information. Because the bottom layer typically does not have components, there is more available real estate to use for this purpose.

Spending some time to design a beautiful and informative silkscreen adds significant value to your board, so it is worth the effort. Because there is no automated test, like the DRC, to ensure that the information in the silkscreen is correct, you should check manually and double-check that there are no errors. Much of the silkscreen text and graphics belong to the footprints themselves. If the footprints come from a quality source, like the KiCad repository, the risk for errors is low, but it is still prudent to check yourself. For example, in Figure 6.2.5.1, the LED footprint came with silkscreen graphics. The circle around the pads is part of the footprint. But to make the assembly process easier, I added the 'K' designator to indicate the cathode pad so that the end-user would know how to connect the LED. The shield pad markings ('A0', 'A1', 'SCL,' 'SDA,' etc.') are part of the Arduino shield footprints that I imported into the project. While I modified the footprint to add space for the prototyping area and the LCD screen, I left the silkscreen text in place.

2.7. Layout Design Step 6: Design rules check

The sixth step of the PCB design process is the Design Rules Check (DRC). While it is a good habit to run the DRC frequently, and at least every time you complete major trace routing or add a copper fill, you should always run it when you are satisfied that the design work is complete and before you send the board to your manufacturer.

To start the DRC, click on the DRC button located in the top toolbar (Figure 6.2.7.1).

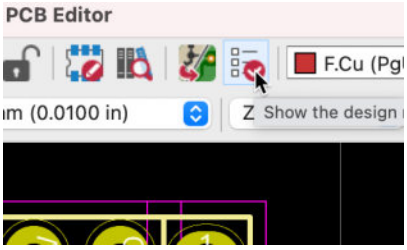


Figure 6.2.7.1: The DRC button in the top toolbar.

In the DRC window (“1” in Figure 6.2.7.2), click ‘Run DRC' to run the basic check. If it all goes well, the Problems and Unconnected tabs will remain empty.

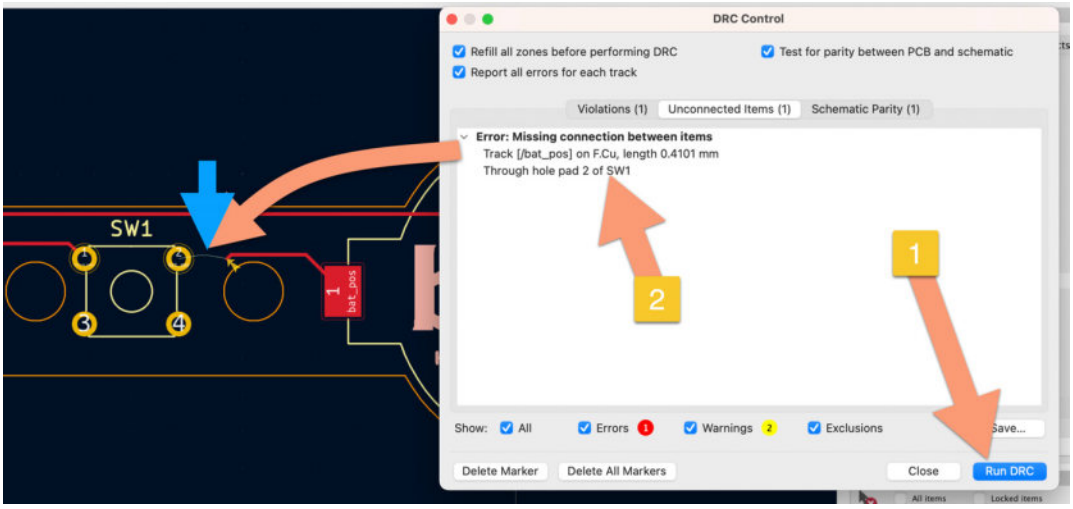


Figure 6.2.7.2: The DRC found three issues.

The DRC window allows several check options. You can get it to automatically refill zones before performing the DRC, to report all errors for tracks, and do a couple of courtyard checks, like courtyard overlap and footprints with missing overlaps. The KiCad documentation defines a footprint courtyard as the smallest area that provides a minimum electrical and mechanical clearance around the component. Footprint courtyard layers are F.CtrYd and B.CtrYd.

When I did the DRC again in my project board, with all options selected, as you can see in Figure 6.2.7.2, the DRC returned one "missing connection" problem. The information the the tool provides allows you to find and fix the problem.

2.8. Layout Design Step 7: Export & Manufacture

The goal of the PCB design process is to turn the PCB from a set of files on your computer into a physical object. There are a few ways by which you can do that. You can use a chemical etching process or a CNC machine to carve out a circuit on a copper board at home. I find the chemical etching process too messy, not to mention that you have to work with potentially toxic chemicals. If you already have a CNC machine, then you can certainly use it to make your boards. With a bit of patience and practice, you will create two or even four-layer boards. You will need to use a special process to create vias and holes in both cases, with copper-plated holes being more challenging.

I prefer the simplicity of using online manufacturers who can produce high-quality boards for a relatively small cost. You will need to plan because the lead time (manufacturing plus shipping) can take weeks.

The standard method for ordering a PCB from an online manufacturer is exporting Gerber files from KiCad and importing them to the

manufacturer's website. Companies like Oshpark have made it possible to upload Pcbnew's simply '. KiCad_pcb' file. I find this development very encouraging because it is so simple. Exporting Gerber files has been the source of many mistakes and wasted time in my life. You must be careful to export the correct files with the correct file name extensions and units and not forget the drill files. With the ability to upload the '.KiCad_pcb', you automatically eliminate a big risk factor.

This book will teach you how to manufacture a PCB with an online service by using both the traditional Gerber files and Pcbnew's '. KiCad_pcb' file.

Part 7: Fundamental Kicad how-to: Symbols and Eeschema

1. Introduction

In the chapters that follow, I will give you an overview of the schematic editor's user interface to know where to find the various tools and options.

I will also show you how to work with schematic symbols, including finding the right one from the symbol chooser, installing external symbol libraries, or creating custom symbols.

Finally, I'll show you how to work with frequently used schematic design elements, like labels and classes, and configure and use the Electrical Rules Check (ERC).

By the end of this part, you will know everything you need to help you work through the example projects in this course.

Keep in mind that KiCad, and Eeschema in particular, have many more features and capabilities than the ones I demonstrate in this section.

This section aims to show you only the most important and frequently used to help you learn what you need to follow and learn from the upcoming projects.

In the last part of this book, you will find several recipes with practical guides and examples of more advanced features.

In Part 8, I will show you the most important and frequently used features of the layout editor, Pcbnew.

2. Left toolbar overview

In this chapter, you will learn the functionalities available through the buttons in the left toolbar of Eeschema. You can see the left toolbar in Figure 7.2.1 below.

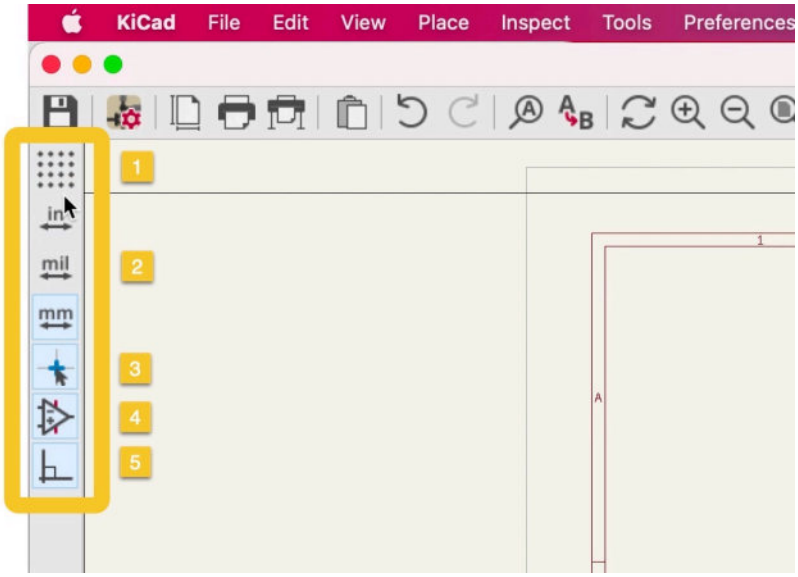


Figure 7.2.1: The right toolbar.

You will learn about the functionalities in the top and right toolbar in later chapters.

To demonstrate what you can do with the buttons in the left toolbar, I will use the schematic diagram from one of the projects of this book.

Grid lines

The first button in the left toolbar ("1") allows you to enable or disable the editor's grid lines. This is a toggle button: click to show the grid and click again to hide it. The schematic editor can display the grid in three styles: lines, dots and small crosses. You can learn more about the grid styles in chapter “5. Schematic editor preferences” (later in this part of the book). Use the grid show/hide button (“1”) to turn show or hide the grid. In Figure 7.2.2 (below) you can see an example of the “Lines” grid style in the schematic editor.

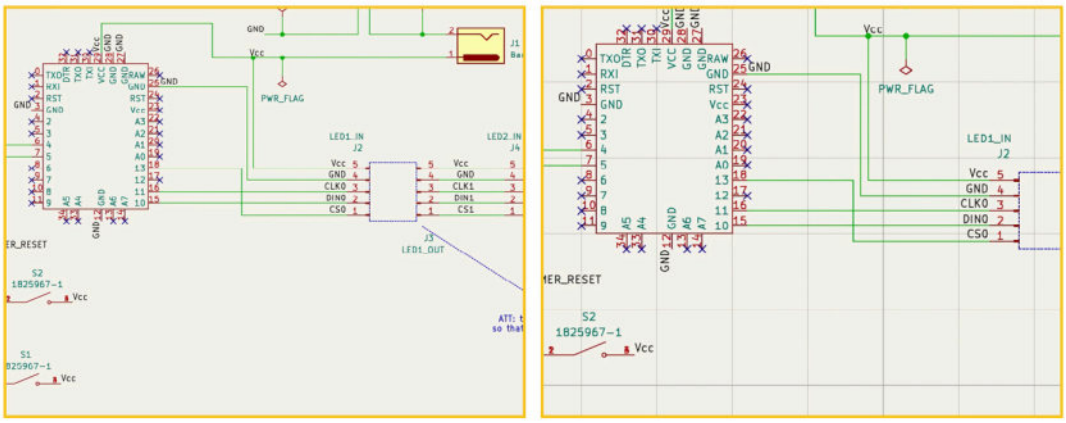


Figure 7.2.2: Grid lines.

As you zoom in and out using your mouse's scroll wheel, the grid automatically adjusts. You can also change the size of the grid size. To do this, you can right-click anywhere in the editor window to open the context menu, then select Grid to open the submenu, and then select one of the available grid sizes (Figure 7.2.3).

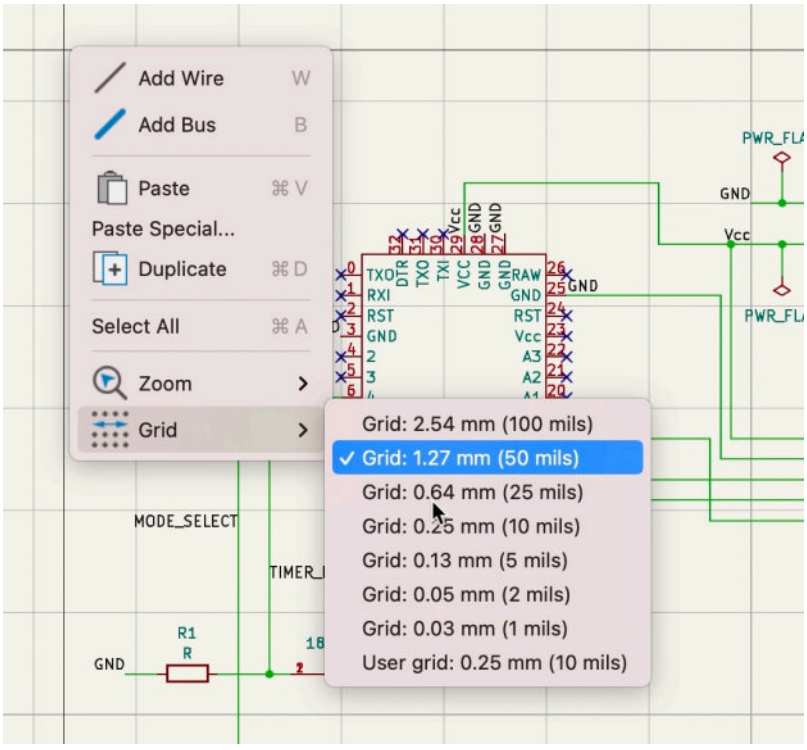


Figure 7.2.3: Change the grid size.

Generally, small grid sizes are better for busy schematics. You can always check the current grid side in the status bar at the bottom of the Eeschema window (Figure 7.2.4).

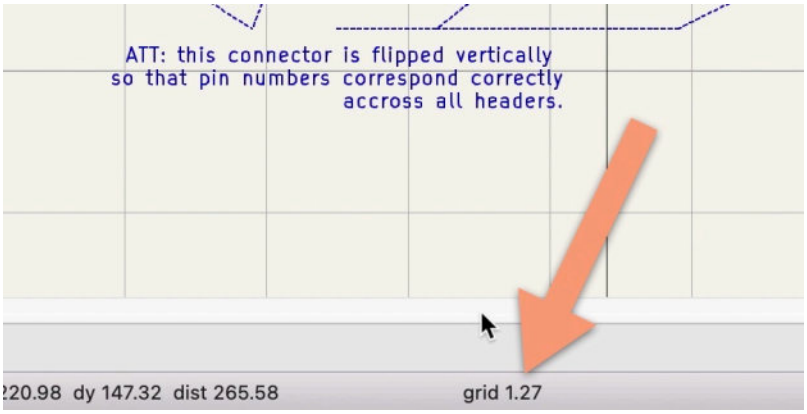


Figure 7.2.4: The current grid size.

Length units

The following three buttons (marked “2” in Figure 7.2.1) allow you to select your preferred unit of length. You can choose between inches, millimeters, and mils.

You can confirm the active unit on the right side of the status bar at the bottom of the Eeschema window (Figure 7.2.5).

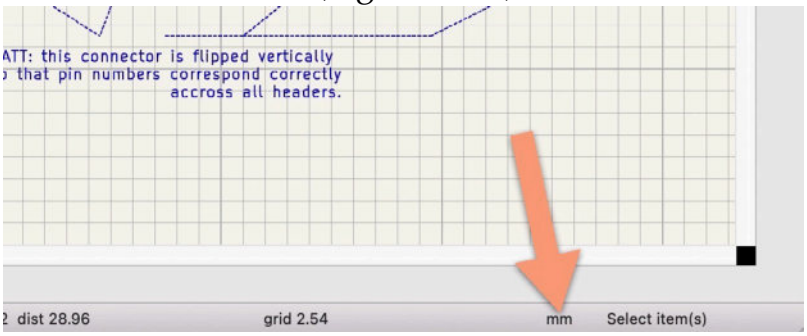


Figure 7.2.5: The current length unit setting.

Cursor

You can change the shape of the cursor by clicking button "3" in the left toolbar. You can choose between a small or full-window crosshair cursor (Figure 7.2.6).

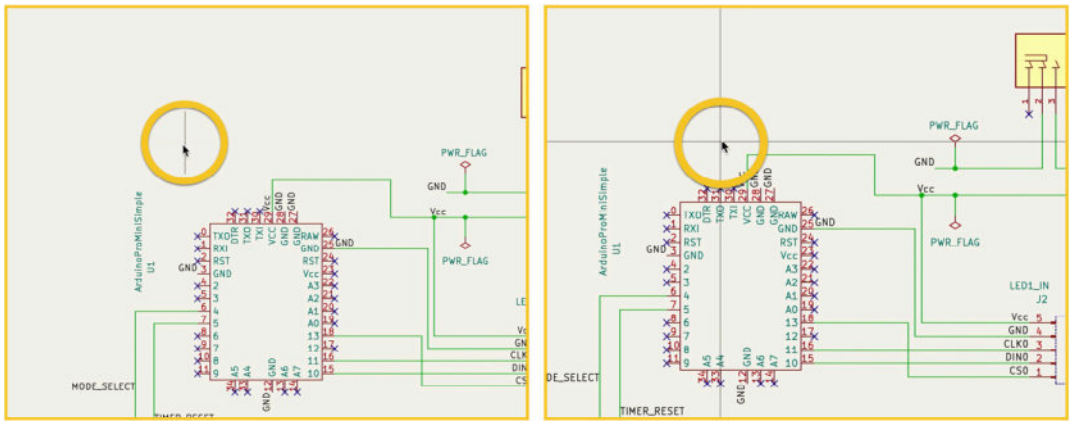


Figure 7.2.6: Eeschema offers two types of crosshairs.

The full-window crosshair variant makes it easier to compare the cursor's position against other objects in the designer window and therefore makes it easier to align objects.

Display hidden pins

Schematic symbols may contain hidden pins to reduce clutter in the editor. An example of a schematic symbol that has at least one hidden pin is the ATmega328PB-A. In this symbol, pin 21 is set as "hidden." To reveal it, you can click on button "4" of the left toolbar. Below (Figure 7.2.7) , you can see pin 21 when the hidden pins button is pressed (left). When the hidden pins button is not pressed, pin 21 is hidden (right).

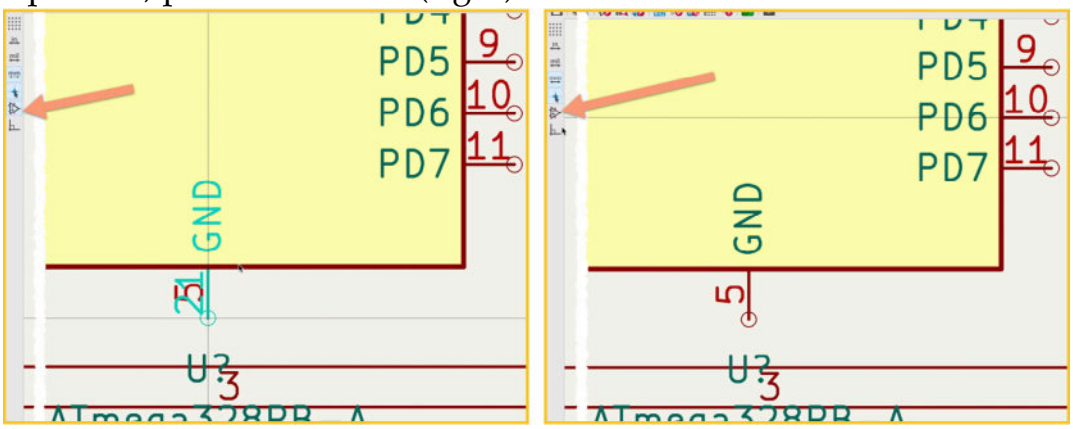


Figure 7.2.7: Showing hidden pins.

H & V lines

The wires and buses tools are available in the right toolbar, and you will learn about it in a later chapter. In short, these tools allow you to draw wires and busses (a bus is a collection of wires) that connect pins. When you click and enable the H & V lines tool (click on button “5” in Figure 7.2.1), the editor

restricts these lines to horizontal or vertical. If you disable the tool, you can draw these lines at any angle you want. You can see the effect of this tool in Figure 7.2.8 below. To draw the lines, I used the wire tool from the right toolbar.

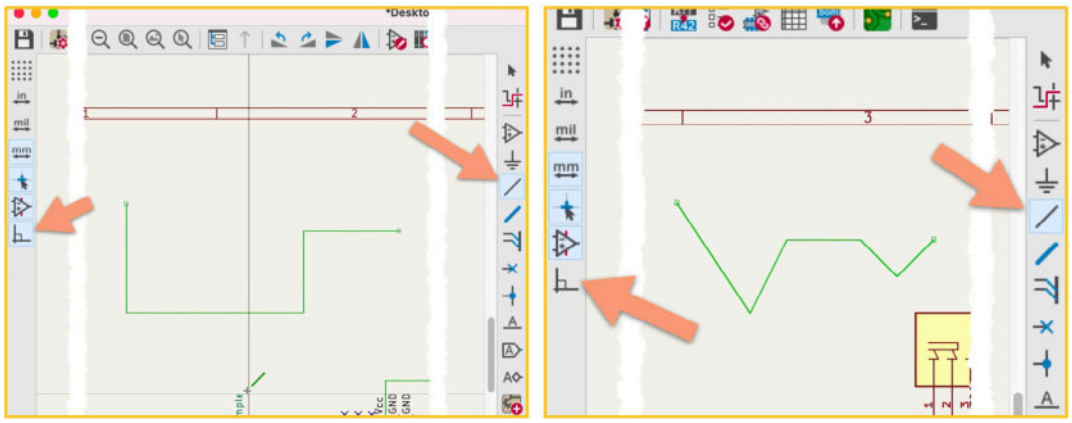


Figure 7.2.8: The effect of the H&V line tool.

Generally, by using horizontal and vertical wires and buses, your schematic will look neat and will be easier to read. I rarely (if ever) draw free-angle wires.

3. Top toolbar overview

This chapter will give you an overview of the buttons that appear in the top toolbar in Eeschema. Most of the functions of those buttons are also accessible via the top menus.

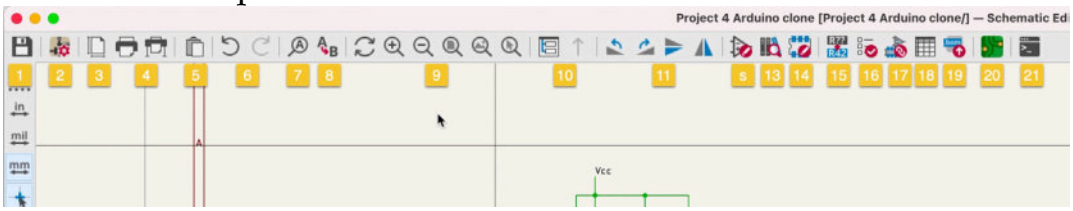


Figure 7.3.1: The buttons in the top toolbar.

For example, the Save button ("1" in the figure above) is accessible via the File menu.

Let's take a closer look at each button or button group.

Save

This button does what you think it does. Click Save to save your work on the disk. You can also use the Ctr-S or Cmd-S shortcuts or use the menu File —> Save.

A variation of Save is "Save as…" which allows you to change the name of the schematic file.

Schematic Setup

Button "2" in the top toolbar gives you access to the Schematic Setup window. In the Schematic Setup window, you can customize the schematic editor to match your work style and project requirements (Figure 7.3.2).

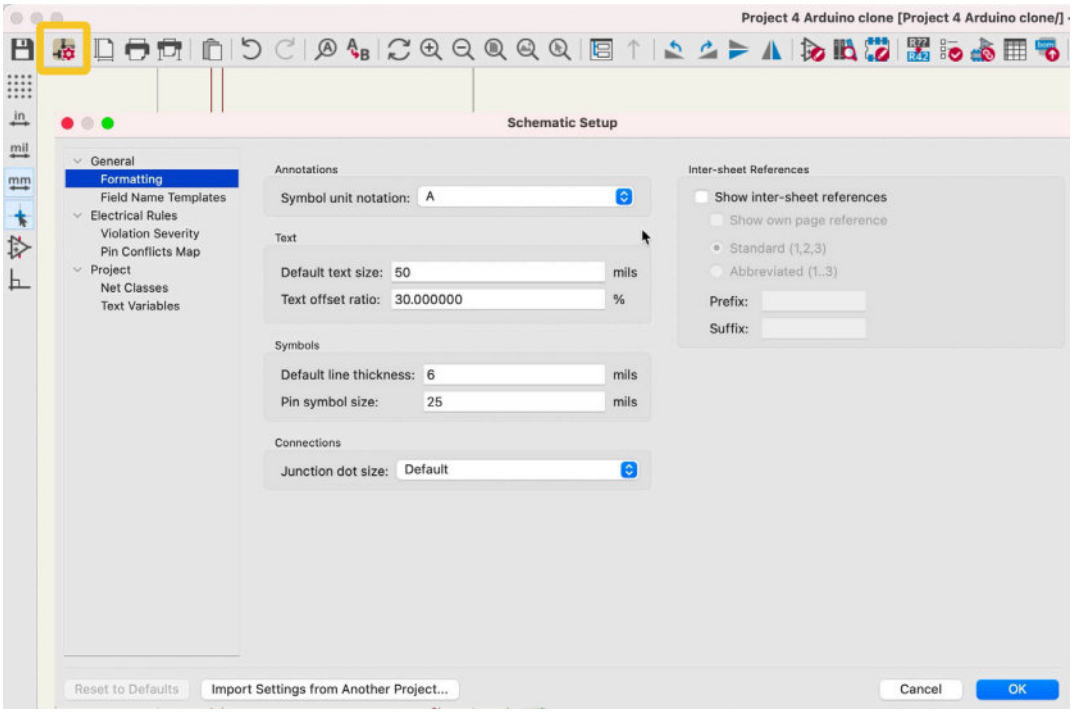


Figure 7.3.2: The Schematic Setup window.

For example, you can set the default text size for text labels, create field name templates, and customize how the electrical rules checker works. You can find dedicated chapters to learn more about the contents of this window later in this part of the book or Part 13 (Recipes).

Page Settings

The Page Settings button ("3") allows you to configure the schematic editor page. When you click on it, you will see the Page Settings Window (Figure 7.3.3).

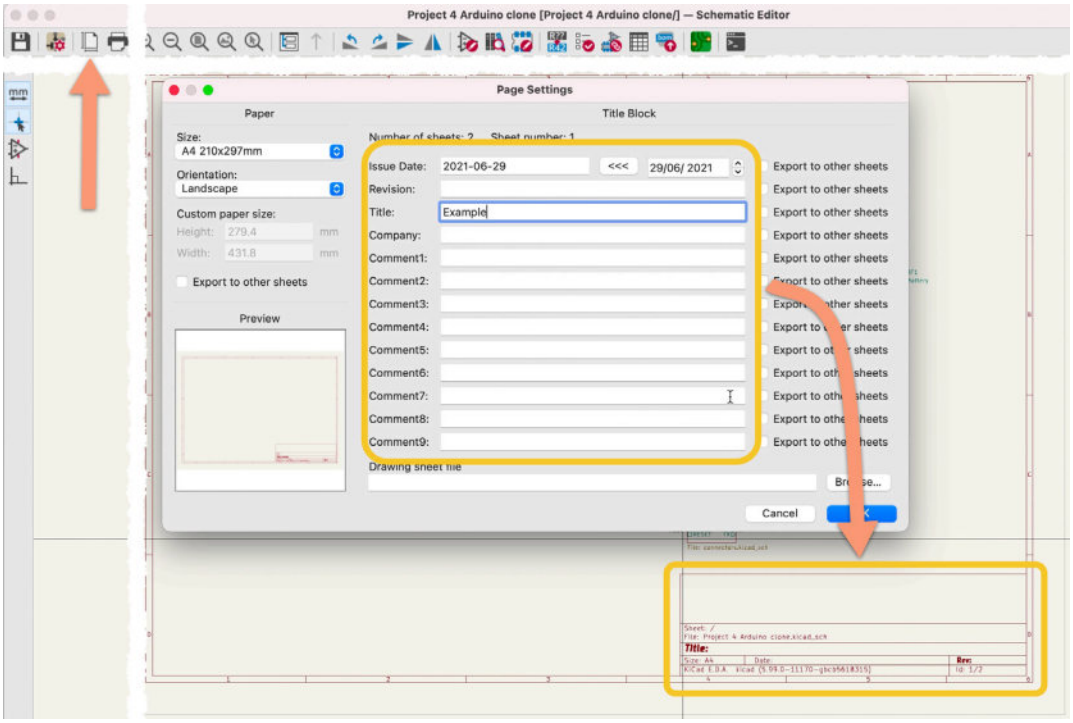


Figure 7.3.3: The Page Settings window.

In the Page Settings window, you can choose the size of the schematic editor page and then set the contents of the sheet label that appears in the bottom right corner of the page. You can access the Page Settings window from the File menu (File —> Page Settings).

Print and Plot

The following two buttons, marked as "4" in Figure 7.3.1, allow you to print or plot your schematic. The difference between the two options relates to where the printing will take place. The first option, Print, allows you to print the schematic to paper on a regular printer. The second option, Plot, allows you to export the schematic to a file using one of the available file formats: Postscript, PDF, SVG, DXF, and HPGL (Figure 7.3.3).

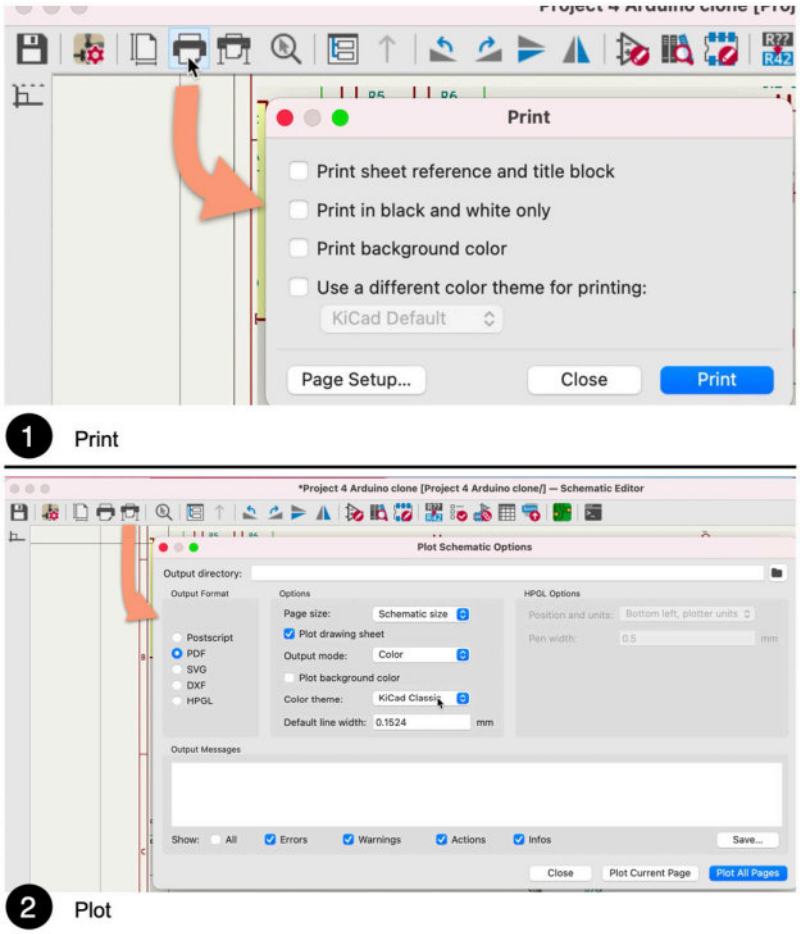


Figure 7.3.3: Print and Plot.

Paste from clipboard

Button "5" allows you to copy a selected item to the clipboard so that you can then paste it somewhere else in the editor.

To use it, first click on any element to select it (this could be a symbol, a text label, a wire, etc.), then press Ctr-C/Cmd-C on your keyboard. This will copy the item to the clipboard. To paste, click on the Paste button ("5") or press Ctr-V/Cmd-V on your keyboard.

Undo and Redo

The two buttons marked as "6" are "undo" and "redo." In KiCad, Undo and Redo work as they do in any other application. Use Undo to revert the last change you made, and Redo to redo it.

KiCad's clipboard has memory, so you can use Undo repeatedly to undo all recent changes.

Find

Click on the Find button to search for a string of text anywhere in the schematic editor. In the example below, I used Find to search for occurrences of the text “SCL” in my schematic. “Find” highlights any hits with a blue halo around the letters.

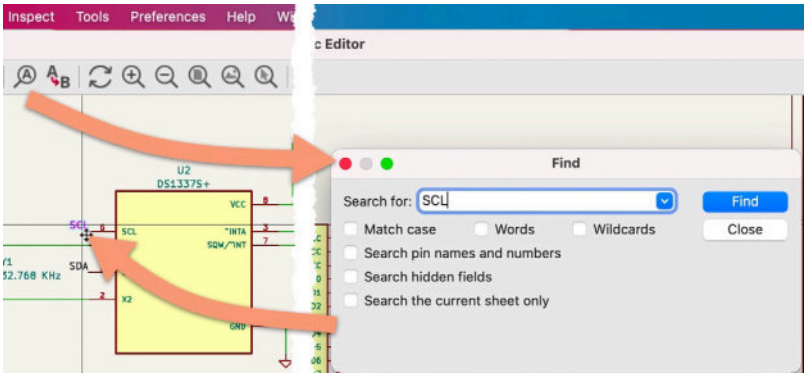


Figure 7.3.4: Find text.

Continue to click on the Find button to jump to the next occurrence of the search term.

Find and replace

The Find and Replace button allows you to change the matching text to a new text string. In the example below, I have used Find and Replace to change all occurrences of "SCL "to" SCK. "This schematic contains multiple instances of "SCL," and I was able to change them all to "SCK" with a single click.

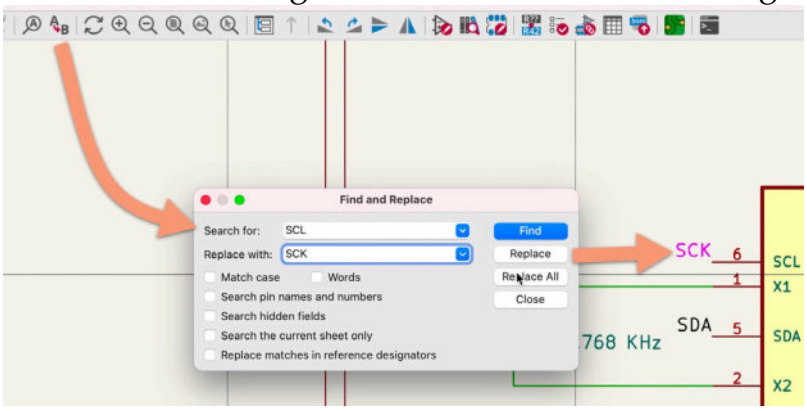


Figure 7.3.5: Find and replace text.

Navigation buttons

The next six buttons allow you to navigate the sheet or refresh the page. From left to right, you can see:

1. Refresh (Ctr-R/Cmd-R).

2. Zoom In.
3. Zoom Out.
4. Zoom to Fit (Ctr-0/Cmd-0).
5. Zoom to Objects (Ctr-Home/Cmd-Home).
6. Zoom to Selection (Ctr-F5/Cmd-F5).

You should experiment with these buttons to experience their effect. I will give you one example: Zoom to Selection.

This button allows you to select a rectangular area in the schematic and have the editor automatically zoom in. To use it, first, click on the Zoom to Selection button, and then use your mouse to draw a rectangle that contains the details at which you want to take a closer look. When you release the mouse button, Eeschema will zoom into the selected area.

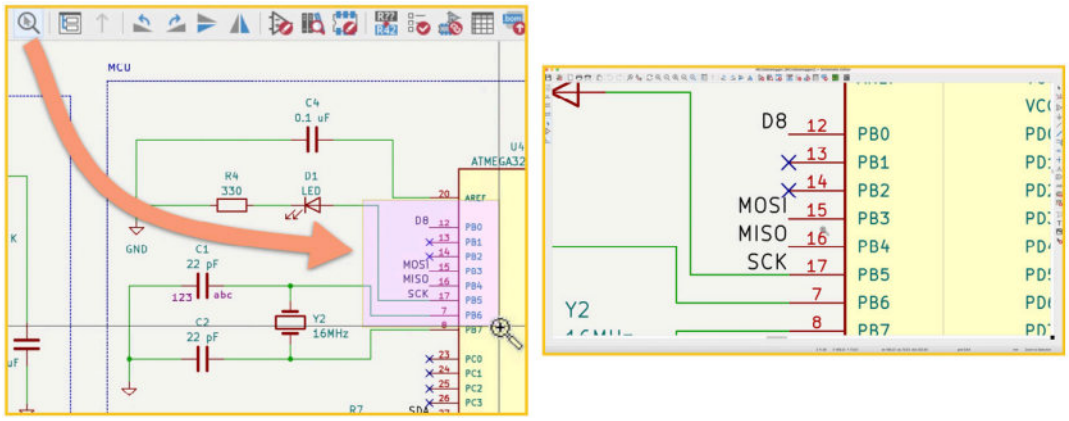


Figure 7.3.6: Zoom to Selection.

Sheet hierarchy

In KiCad, it is possible to split your schematic into multiple sheets. When you choose this approach, you create a hierarchy of sheets. The first sheet is the "root," and it may contain one or more sheets. Subsequent sheets may have sub sheets.

You can navigate the sheet hierarchy with the help of the sheet navigator window. To show the navigator window, click on the Sheet hierarchy map ("10"). You can see an example of the map below. This example comes from one of the example projects in this book, where I have used hierarchical sheets to split the schematic into multiple pages.

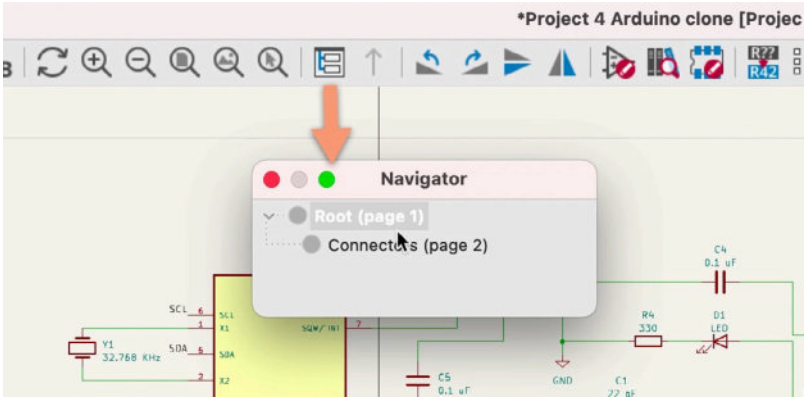


Figure 7.3.7: Hierarchical sheets button.

When you are in a sheet other than "root," you can click on the Up Arrow button (on the right side of the Hierarchical Sheet map button) to jump up to the previous sheet.

You can learn more about hierarchical sheets in a dedicated chapter later in this part of the book.

Rotate and flip

The next group of four buttons (marked "11") allow you to rotate or flip the currently selected element. With these buttons, you can:

1. Rotate by 90 degrees.
2. Rotate by -90 degrees.
3. Flip vertically.
4. Flip horizontally.

For example, I have applied the vertical flip operation to the 4020 symbol in the figure below. On the left is the original orientation of the symbol, and on the right is the flipped version.

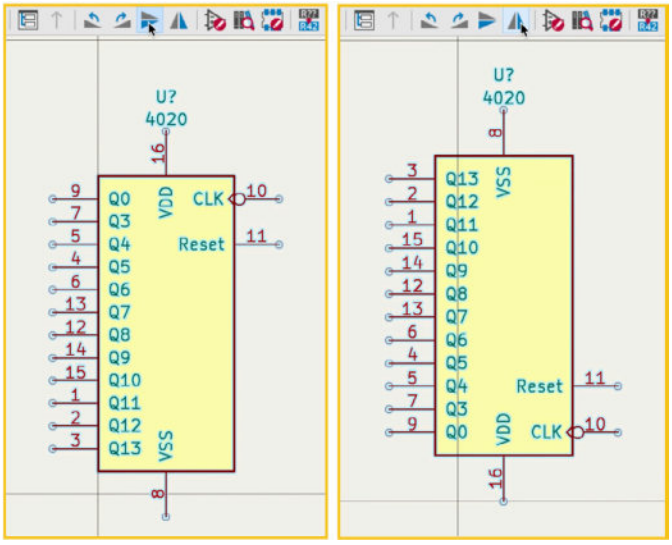


Figure 7.3.7: Vertical flip.

Use these tools to orient a symbol as required to make wiring more straightforward.

Symbol editor

The Symbol Editor is an essential app and component of KiCad. With the Symbol Editor, you can edit existing symbols or create new ones. To invoke the Symbol Editor click on its button in the top toolbar ("12").

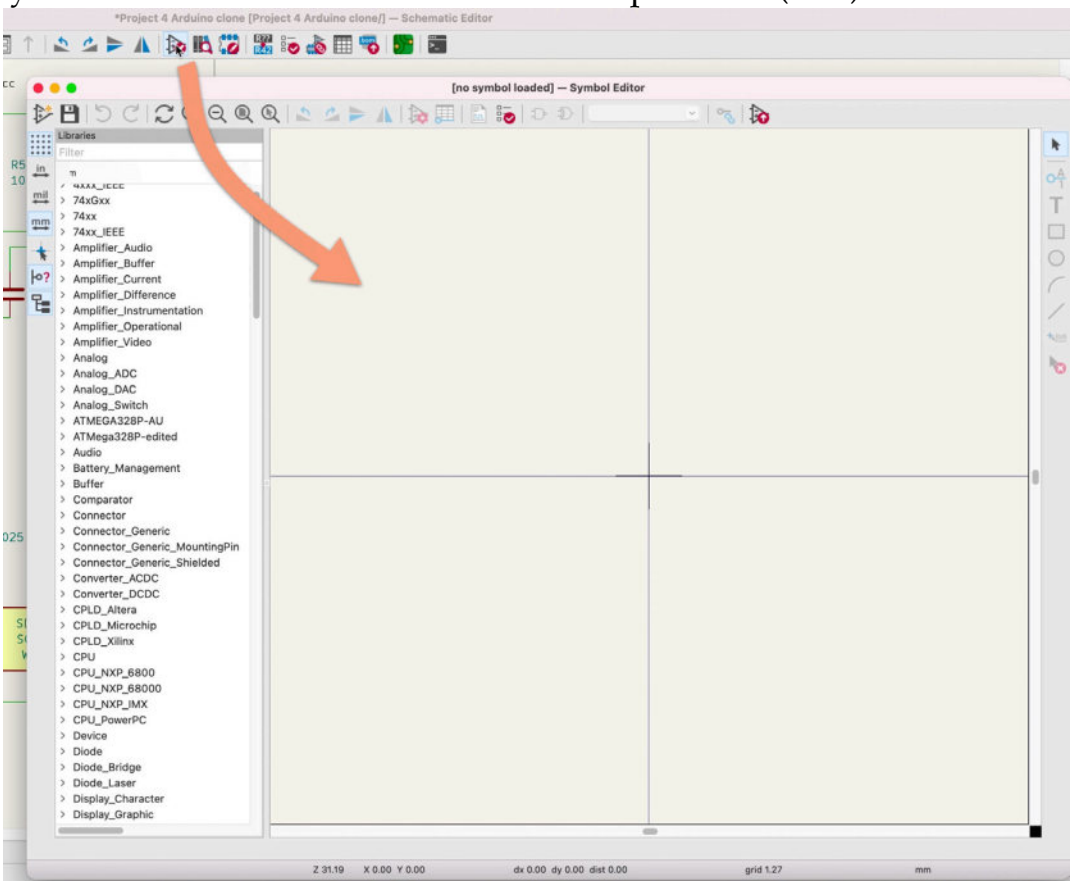


Figure 7.3.8: The Symbol Editor.

In the dedicated chapter later in this part of the book, you can learn more about the Symbol Editor, including creating a new symbol.

Symbol libraries browser

The Symbol libraries browser (sometimes I call it the "Symbol chooser") is a tool you will use during the schematic design workflow. This tool allows you to find a symbol and add it to the schematic editor sheet. Click on the symbol chooser button ("13") to bring up this window.

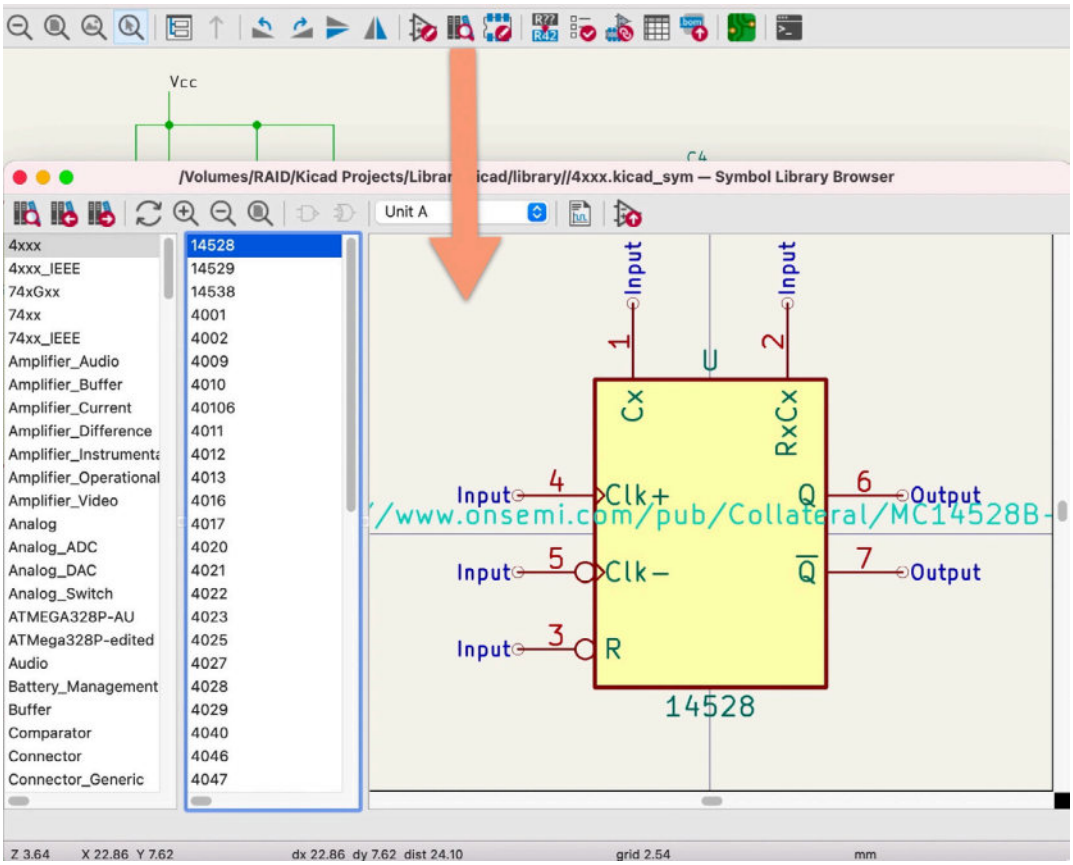


Figure 7.3.9: The Symbol Library Browser.

The browser allows you to select a library (either one that comes with KiCad or others that you can find and install). Once you select a library, its contained symbols appear in the middle pane of the window. Once you choose one of the symbols, you can see a preview in the right pane.

You can learn more about this tool in a dedicated chapter later in this part of the book.

Footprint Editor

The Footprint Editor is a tool that allows you to edit or create footprints. You can start the editor by clicking on its button in the top toolbar ("14").

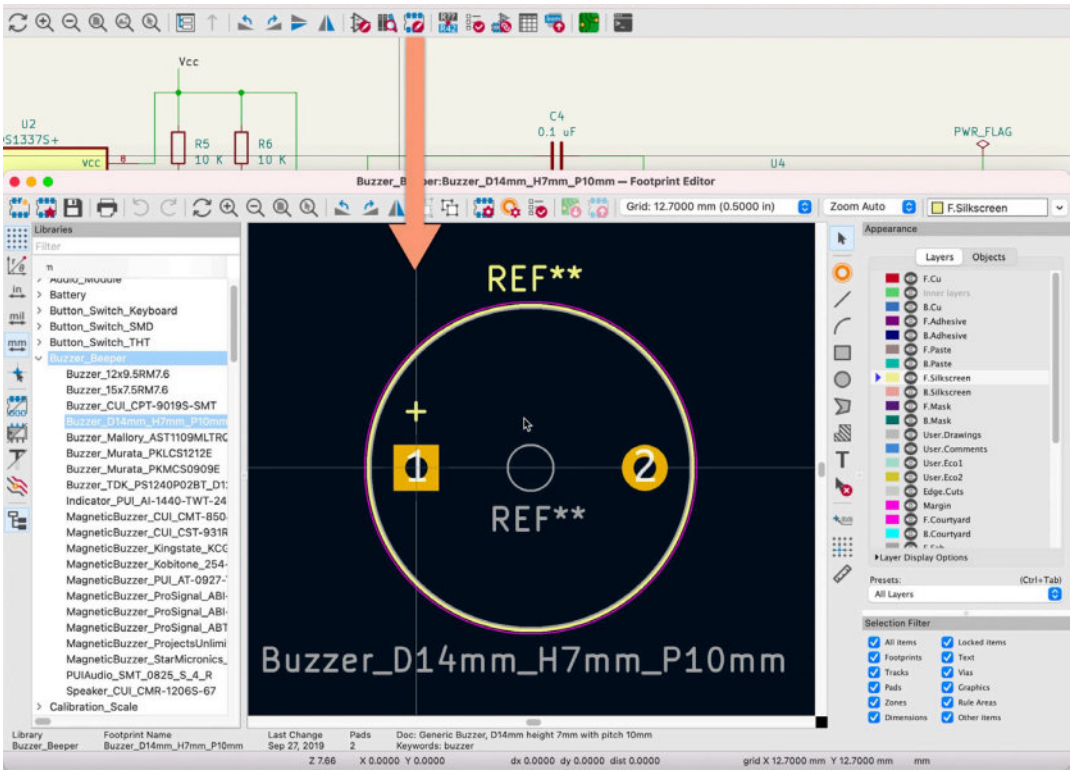


Figure 7.3.10: The Footprint Editor.

You can learn more about the footprint editor and how to create a new footprint in a dedicated chapter in Part 8 of this book.

Annotate Schematic

The annotator tool allows you to quickly create and assign reference designators to all symbols in the schematic. A reference designator is an identifier. As such, it must be unique for each symbol. You can set designators manually, but it is better to leave this task to the automated tool for all practical purposes.

In the example below, I have used the annotator to create designators for the symbols in my project. I have used a yellow circle to indicate some of the designators that the annotator created and assigned. To start the annotator, click on its button ("15") in the top toolbar.

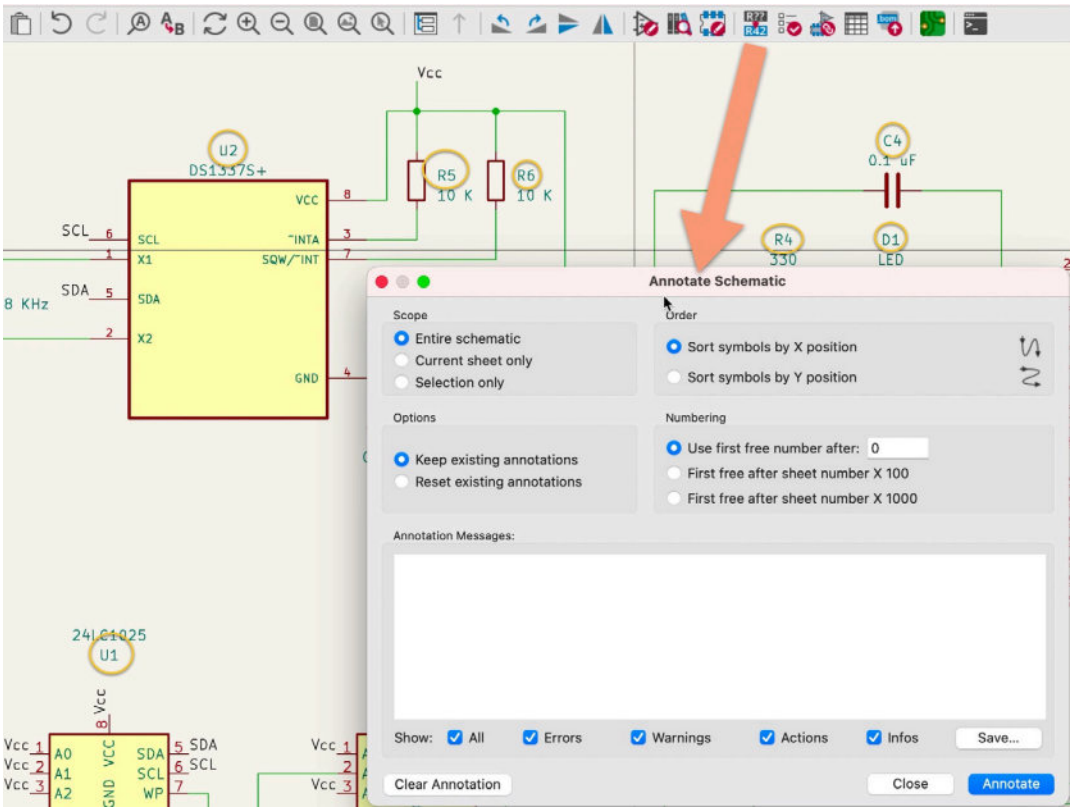


Figure 7.3.11: The schematic annotator.

Electrical Rules Checker (ERC)

The Electrical Rules Checker (ERC) is a tool that finds violations in the schematic. For example, it can find pins left unconnected or power pins not connected to a power source. To conduct an ERC, click on its button ("16") and then click on "Run ERC." In the example below, the ERC has detected one error and two warnings.

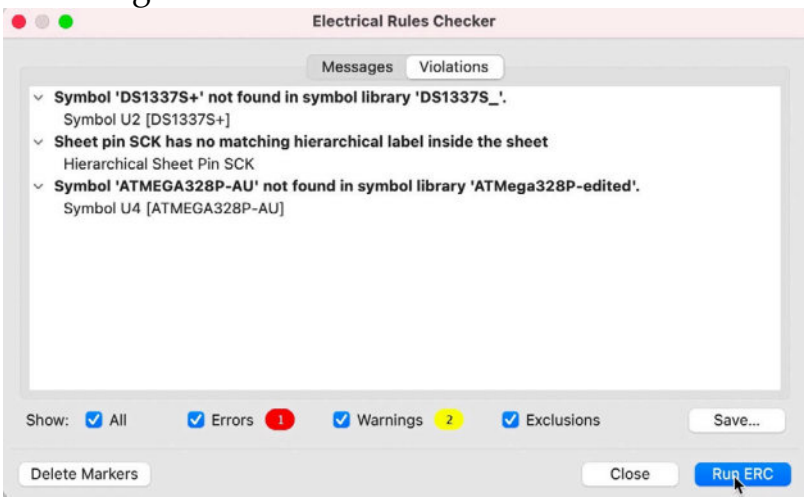


Figure 7.3.12: The Electrical Rules Checker.

It is possible to customize how the ERC tool works and fine-tune it to the requirements of your project. If you want to learn more about this, please read the dedicated chapter later in this part of the book.

Assign footprints

Each symbol in the schematic editor must be associated with footprints that will appear in the layout editor. Some symbols have default associations, but you will want to do the associations manually in most cases.

It is possible to associate a symbol with a footprint via the symbol's properties window. Doing it this way is tedious and time-consuming. A better way is to use the bulk associations tool available from the top toolbar ("17"). When you invoke this tool, the "Assign Footprints" window will appear:

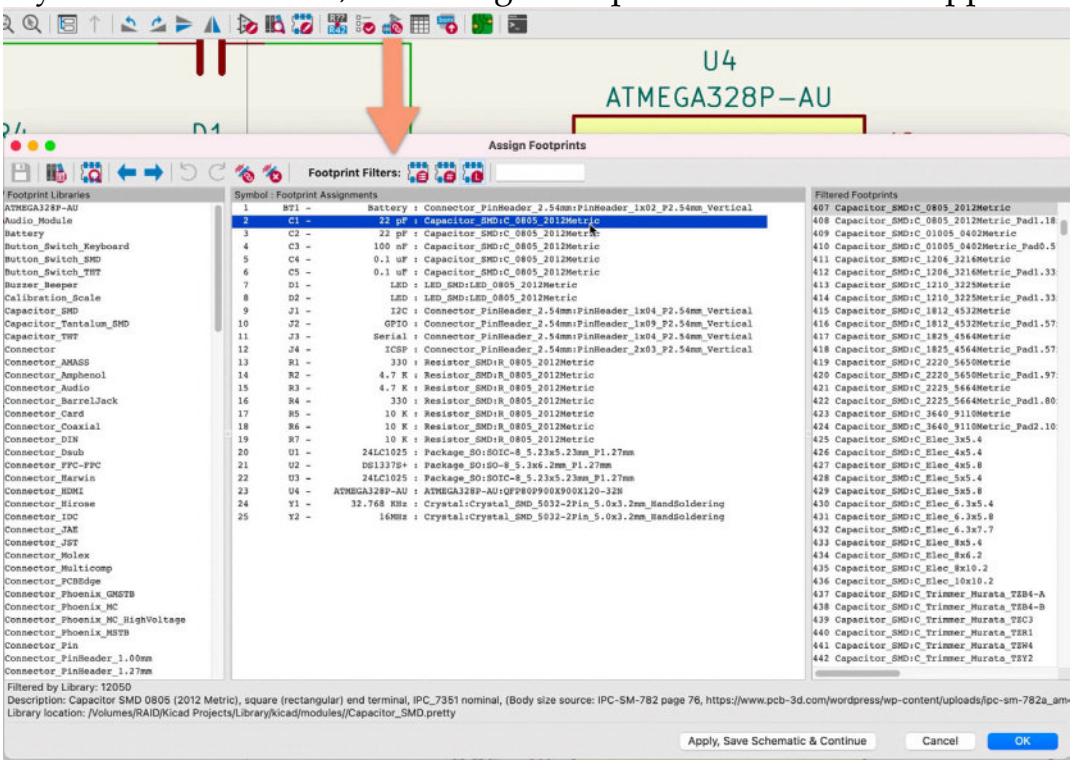


Figure 7.3.13: The “Assign Footprints” window.

The main area of this window is the middle pane, where you can see the existing and pending associations. The left pane contains the available footprint libraries, and the right pane contains footprints that match the search filters you have selected.

I have prepared a chapter dedicated to the symbol and footprints association tool where you can learn how to do the associations efficiently.

Bulk-edit symbol fields

A quick way to edit symbol properties across all schematic symbols (without going into each symbol's properties window) is to use the bulk symbol field editor.

Click on the bulk symbol property button ("18") to bring up its window (Figure 7.3.14).

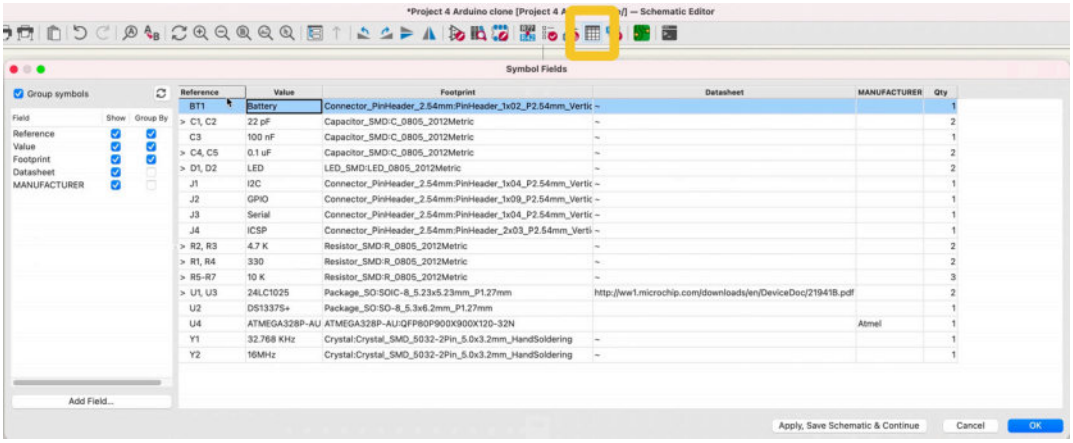


Figure 7.3.14: The bulk symbol field editor.

You can use this tool like a spreadsheet. To edit a field, click on it to select it and type a new value. To commit your changes and close the window, click on OK. To commit your changes but keep the window active, click on "Apply, Save Schematic & Continue."

This tool is also helpful if you want to create a BOM (Bill of Materials). I show how to do this in a chapter in the Recipes part of this book.

Bill of Materials

A Bill of Materials is a list of the components needed for your printed circuit board, including information like quantities, values, and sources. The BOM button ("19") gives you access to the BOM functionality in Eeschema.

You can learn more about BOM in both Eeschema and Pcbnew in the dedicated chapters in the Recipes part of this book.

Pcbnew

The Pcbnew button ("20") will open the layout editor. Pcbnew also has a button to allow you to jump back into Eeschema.

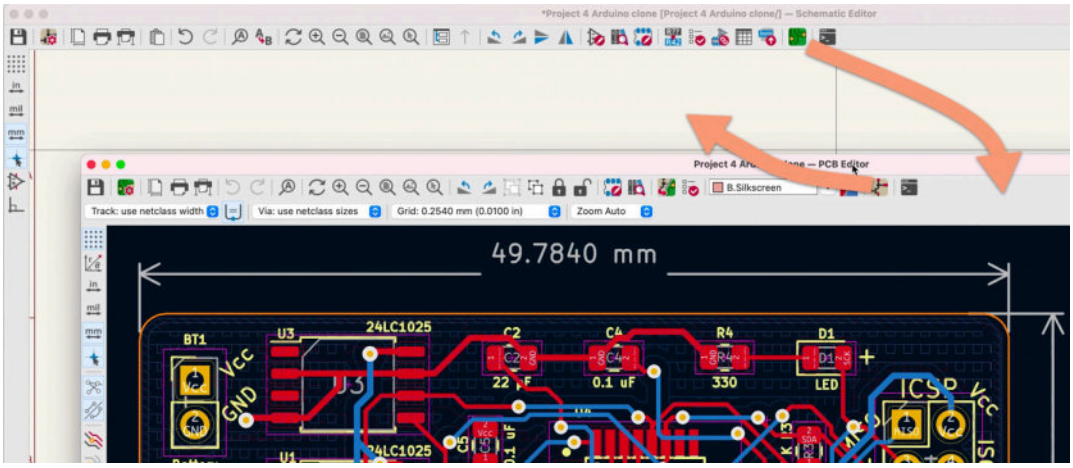


Figure 7.3.15: Pcbnew and Eeschema are well integrated.

You will learn to use Pcbnew and its most essential features in Part 8 of this book, so I will keep this segment short and continue with the Python shell.

KiCad Python shell

KiCad 6 contains a Python API that you can use to add functionality in Python modules or interact with KiCad directly using Python commands and KiCad's Python API.

You can open KiCad's Python shell by clicking on the shell button in the top toolbar ("21").

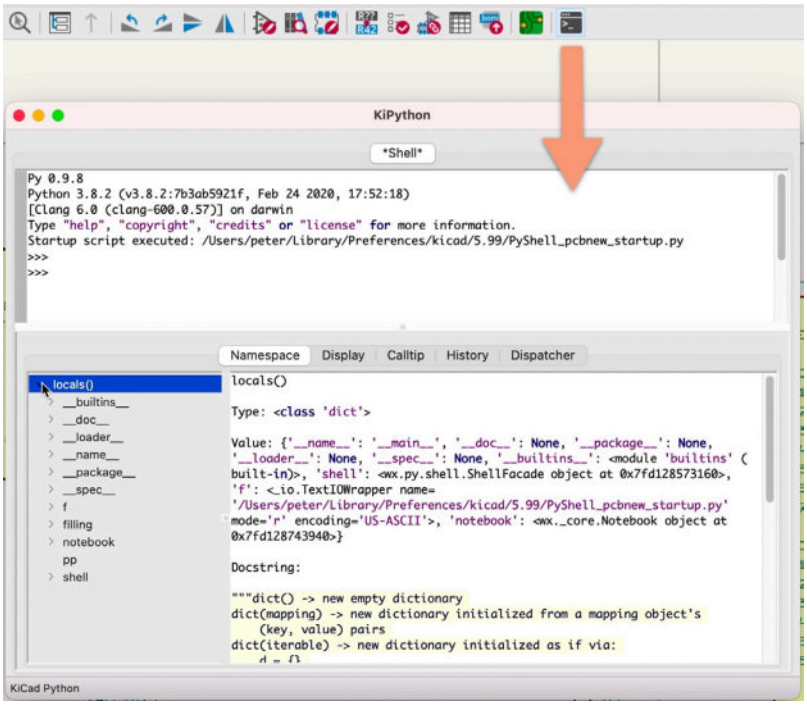


Figure 7.3.16: The KiCad Python shell.

The Python shell is a new feature in KiCad and was still "work in process" as I wrote these lines. For this reason, I have chosen not to cover it in the current edition of this book.

4. Right toolbar overview

This chapter will give you an overview of the buttons that appear in the right toolbar in Eeschema. You can see the right toolbar with its numbered buttons in Figure 7.4.1 below.

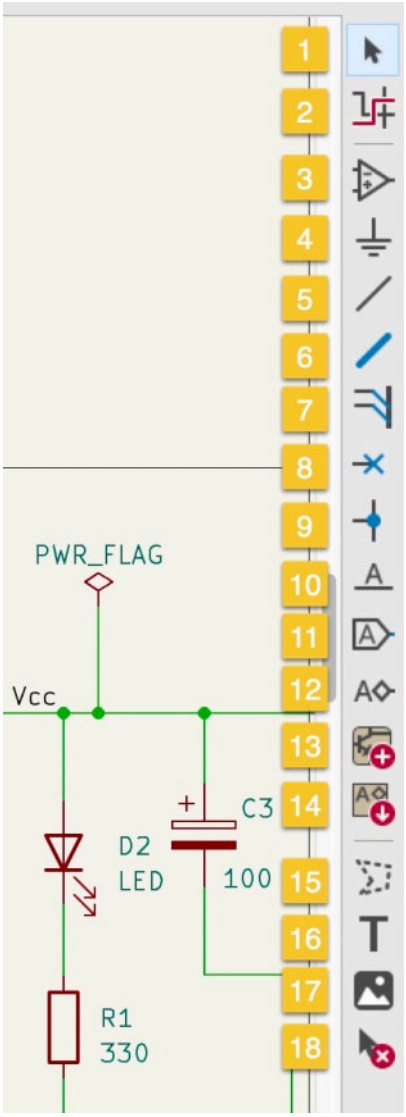


Figure 7.4.1: The buttons in the right toolbar.

Let's take a close look at each button or button group.

Pointer

When no tool is active (for example, the Wire or label tool), you are using the default pointer. If you have activated a tool, you can revert to the pointer by clicking on the arrow button ("1" in Figure 7.4.1) or hitting the escape key.

With the pointer active, you can left-click on any element in the designer to select it, and then you can use a hotkey, then right-click to show the context menu, or click and hold to move it.

Net and pin highlighter

The net and pin highlighter tool is a helper tool that makes it easy to see wires and pins that belong to the same net. To use it, click on the highlighter button ("2") to enable it, and then click on any wire or pin that belongs to the net that you want to highlight.

In the example below, I have clicked on a GND net member wire. The editor then highlights all GND member wires and nets with an alternate color.

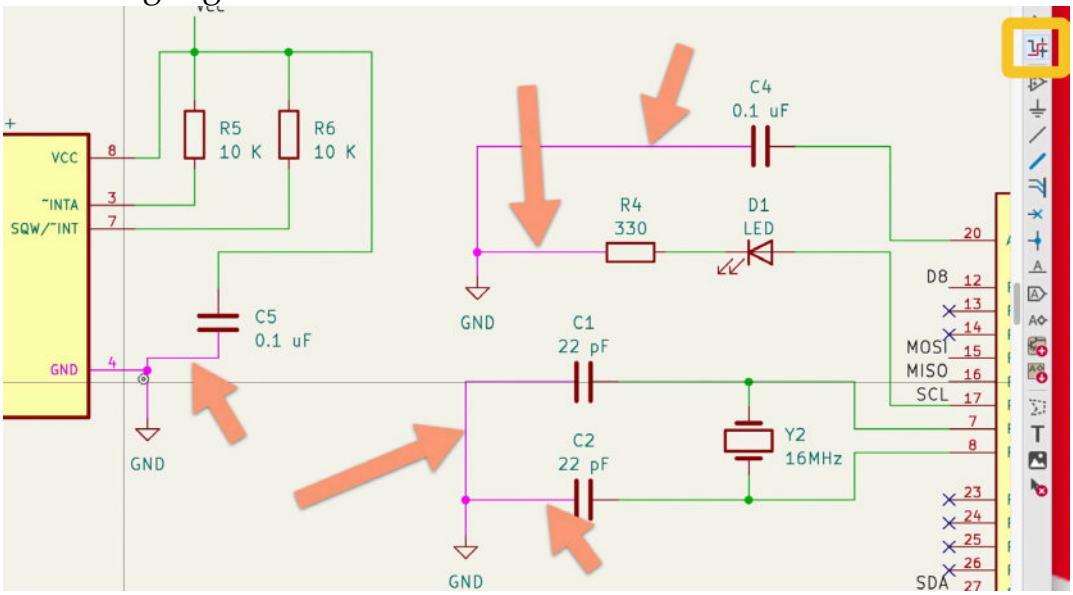


Figure 7.4.2: The Net and Pin Highlighter.

Symbol Chooser

Clicking the third button in the right toolbar brings up the symbol chooser window. You can browse the accessible schematic symbol libraries with the symbol chooser, find a symbol, and add it to the schematic editor sheet.

In the example below, I used the symbol chooser window to browse the available libraries. I have found a diode and selected it. On the right side of the window, you can see the symbol in the preview pane. This symbol also has an associated footprint, which you can also see in the right bottom pane. Notice that between the two preview panes, there is a footprint dropdown. Many symbols have more than one compatible footprint, and you can use this dropdown to select the most appropriate footprint for your project.

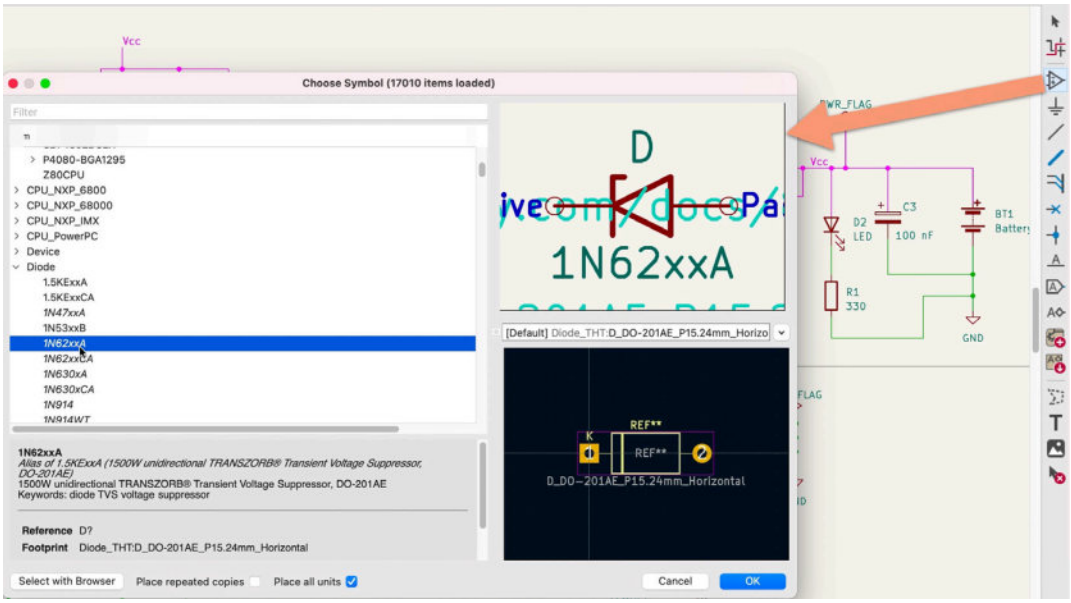


Figure 7.4.3: The symbol chooser.

To add the selected symbol to the editor, double-click on its row in the right pane or click OK. You can also bring up the symbol chooser window by using the "A" hotkey.

Power symbol chooser

The power symbol chooser is a specialized version of the symbol chooser. You can quickly find and add power-related symbols using the power symbol chooser, like ground and voltage source symbols. You can invoke the power symbol chooser window by clicking on the button "4".

In the example below, I have invoked the power symbol chooser and have selected the +3.3VA symbol. You can see the symbol in the preview pane on the right of this window.

Power symbols don't have associated footprints because they don't have a physical representation on the final PCB. KiCad uses power symbols during electrical rules checks to determine whether power pins are correctly wired. For example, imagine a symbol, such as a microcontroller that contains a pin configured as a power pin. If this pin is not connected to a compatible power symbol, the ERC will report this as a violation. You will learn about cases like this in the projects later in this book.

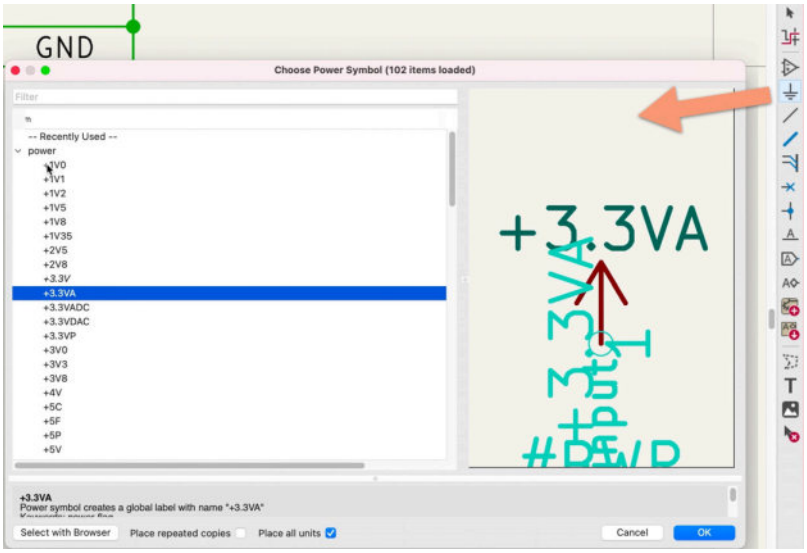


Figure 7.4.4: The power symbol chooser.

You can find the power symbols also in the regular symbol chooser, listed under the power library.

Wire

The wire button ("5") enables the wiring tool. With the wiring tool, you can draw wires that connect pins. To draw a wire, click on its button to enable the wire tool, and then left-click anywhere in the editor to start drawing. Click again to create an angle, and continue drawing. Double click to end drawing.

If you place the mouse pointer over the circle at the end of a pin, the wire tool is automatically enabled so that you can start drawing a new wire. During drawing, when you place the cursor over the pin circle and click, the drawing ends (and you don't have to double-click). In KiCad 6, wire drawing uses the technique I described above to make drawing more efficient.

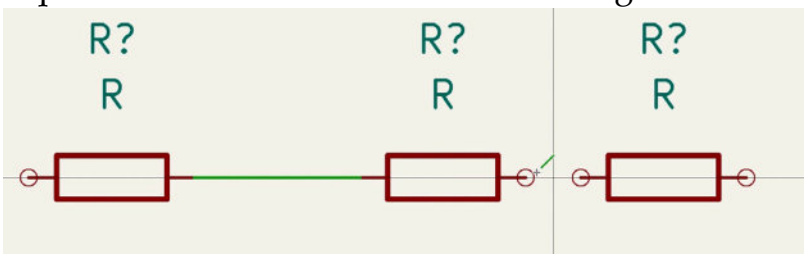


Figure 7.4.5: Connecting pins with wires.

Wires (and buses) are two out of three ways for connecting pins. The third way is to use labels. You will learn about labels later in this chapter and the book's projects.

Bus and bus entry

A bus represents a collection of wires as a single thick line in the schematic. Buses are useful for efficiently depicting related wires. For example, imagine a memory module with four pins for addressing and eight pins for its data. Instead of using 4 + 8 = 12 regular wires, you can use one bus line for the address pins and one for the data pins. The result is a cleaner-looking schematic.

You can learn how to use the bus and bus entry feature and its enabling buttons (in group “6” in Figure 7.4.1) in a dedicated chapter in the Recipes part of this book.

I show a simple example in the figure below where four regular wires are bundled into a single bus. To connect a wire to the bus, I use the bus entry symbol. This symbol looks like a regular wire, except it is at 45-degree against the bus line.

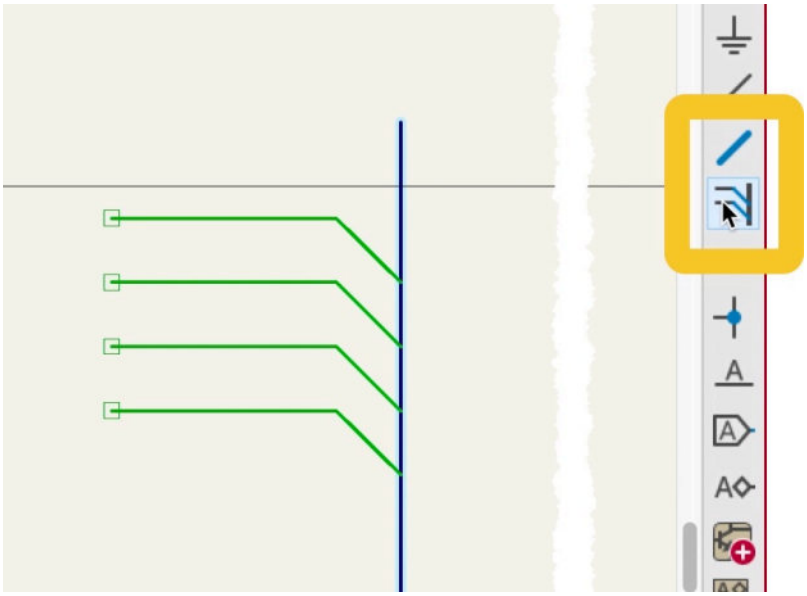


Figure 7.4.5: Four wires bundled in a bus using bus entry symbols.

No connect

The electrical rules checker (ERC) will specifically look for unconnected pins and list them as a violation if it finds any. However, it is common to leave a pin unconnected deliberately. To prevent the ERC from raising a fault flag for deliberately unconnected pins, you can attach the "no connect" symbol. Click on the "no connect" button on the right toolbar ("7") and then attach the

"x" symbol on any pin that you specifically want to leave unconnected to another pin.

You can see an example of several unconnected pins using the "no connect" symbol to prevent the ERC from flagging violations.

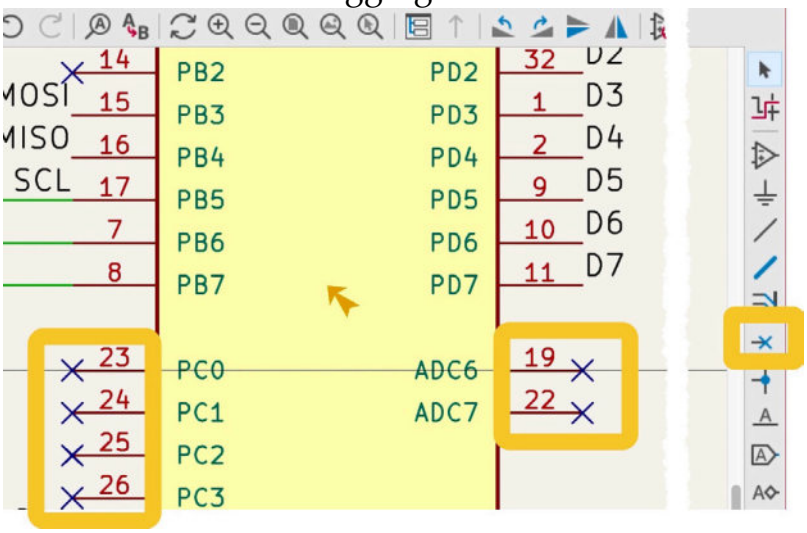


Figure 7.4.5: Unconnected pins marked with the “no connect” symbol.

Junction

With the junction tool ("8"), you can electrically connect two wires. Let's look at an example to understand how this works.

Consider the two wires that are electrically connected with a junction below:

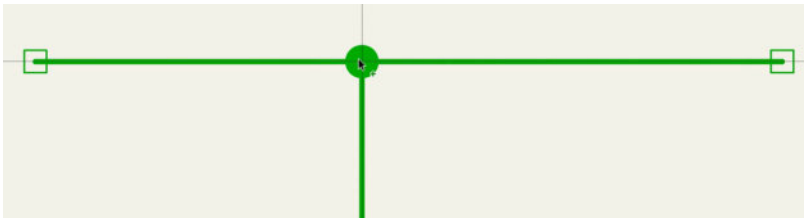


Figure 7.4.6: Two wires connected with a junction.

The green disc is the junction symbol, and it electrically connects those two wires. You can move the junction or delete it as you would with any other symbol. Click to select it and click-hold to move it, or type the delete key to remove it.

In the example below, I have moved the junction away from the two wires. The removal of the junction made the two wires electrically unconnected. You can move the junction back to the intersection between the two wires to restore the electrical connection. You can also use the junction button to create a new junction.

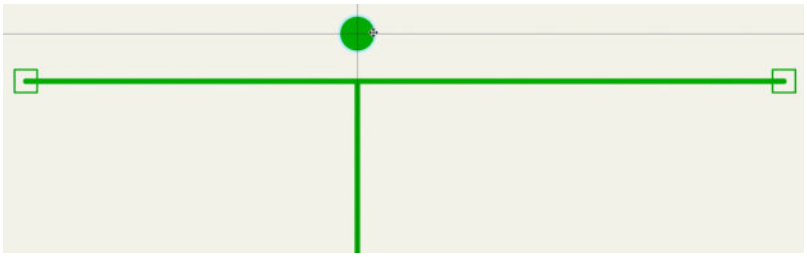


Figure 7.4.7: The two wires are not connected.

Labels

Alongside wires and buses, labels consist of another way to create connections between pins. Unlike wires and busses, labels use text to create named nets. Pins that share the same label are members of the same net. In addition, nets that connect pins that share the same net label also inherit the name contained in the label; hence they are "named nets."

Generally, net labels help to create cleaner-looking schematics because they don't use wires. In a schematic, you can achieve the best results by combining net labels with wires and buses.

Consider the example below:

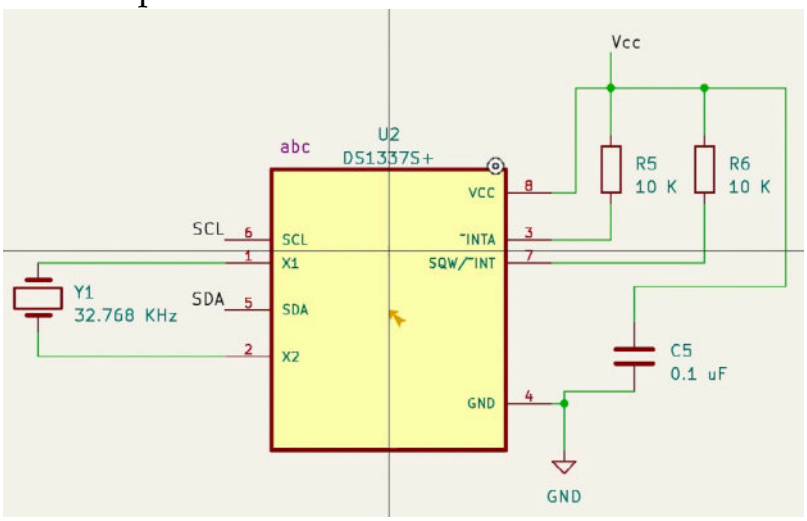


Figure 7.4.8: Using net labels and wires.

In this example, I have a symbol with several pins. I have used regular wires to connect pins to nearby symbols, like the capacitor and resistor on the right. However, two pins, 6 and 5, must connect to pins in a symbol that is elsewhere in the schematic. Instead of using long wires with multiple 90-degree angles, I have used the "SCL" and "SDA" net labels and logically completed these connections.

In addition, because I have attached net labels to pins 6 and 5, I have created nets with the names "SCL" and "SDA." This makes it easier for me to distinguish these important nets elsewhere in KiCad, especially in Pcbnew, where I will make decisions relating to various nets' geometrical characteristics based on their roles.

Eeschema has three types of labels:

1. Net labels (the first button from the top in the buttons marked as "9" in Figure 7.4.1). These labels work within a single schematic sheet.
2. Global labels (the second button from the top in “9”). These labels work across all schematic sheets in a project.
3. Hierarchical labels (the third button from the top in “9”). These labels work with hierarchical sheets.

To learn more about labels, please refer to the dedicated chapter later in this part of the book. There is also a chapter on hierarchical labels.

Hierarchical sheets

The two buttons marked as “10” in Figure 7.4.1 allow you to create and work with hierarchical sheets. Hierarchical sheets help break up a busy schematic sheet into two or simpler sheets.

You can see an example below. In this example, I have a root sheet that contains a symbol that represents a second sheet. This hierarchical symbol provides a way for the two sheets to communicate via means of hierarchical labels. You can double-click on the hierarchical symbol to jump to the sheet it represents or use the navigator from the top toolbar.

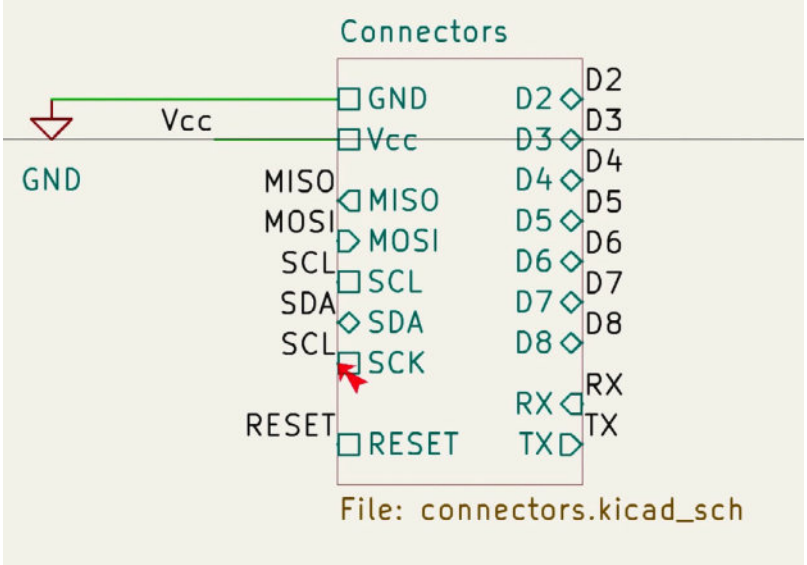


Figure 7.4.9: A hierarchical sheet symbol.

In the example above, notice the "file" reference below the rectangle. Eeschema uses a dedicated schematic file to store data in each hierarchical sheet.

You can learn more about hierarchical sheets in a dedicated chapter later in this part of the book. I am also using hierarchical sheets in the MCU Datalogger project.

Graphics, text, images

The three buttons in the group marked “11” in Figure 7.4.1 allow you to draw simple line graphics, add text labels, and insert images. You can use these tools to annotate your schematic.

You can see an example below where I have drawn a rectangle around a schematic segment, added a text label, and inserted a JPG image.

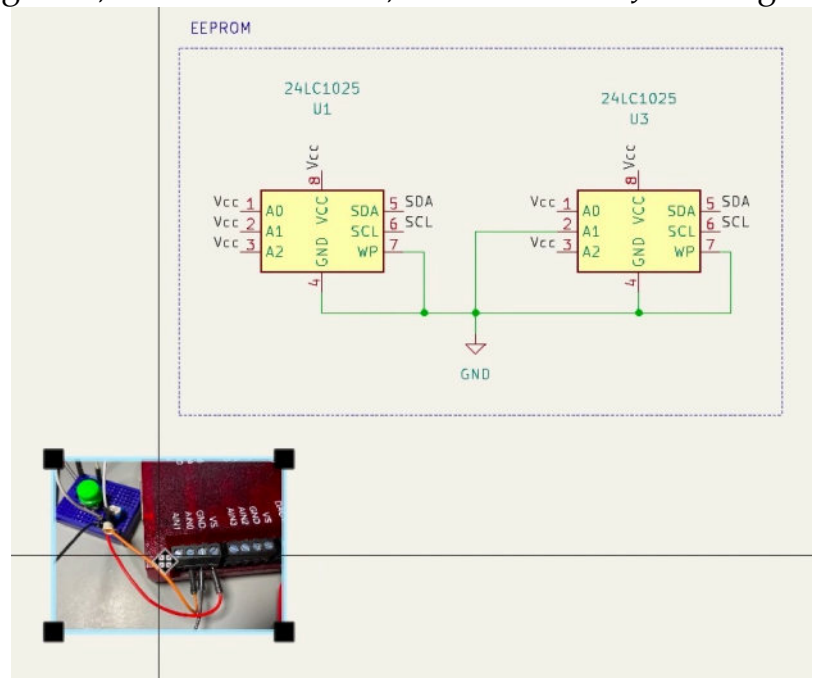


Figure 7.4.10: Example of use of the line, text, and image tools.

Interactive delete

With the interactive delete tool, you can click and delete any element in the schematic editor. This tool is "interactive" in the sense that it will highlight the element that you are about to delete, giving you visual feedback.

In the example below, I have enabled the interactive delete tool by clicking on its button (“12” in Figure 7.4.1). The cursor becomes a virtual eraser. As I

hover the cursor over an element, such as the graphics line, the editor highlights the element by changing its color to purple.

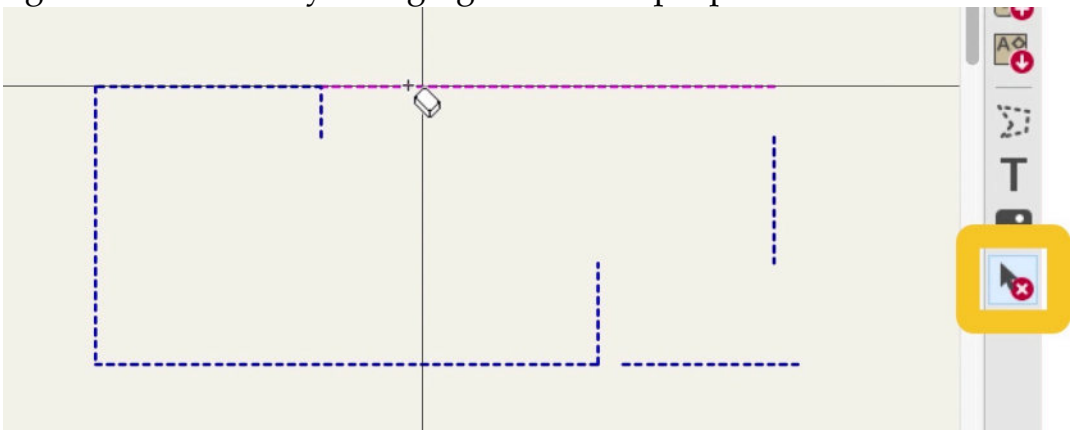


Figure 7.4.11: The interactive delete tool.

To delete anything that the cursor is hovering over, click.

5. Schematic editor preferences

KiCad 6 is very configurable. You can use its configuration options to set up KiCad to suit your work style and project requirements. You can configure much of KiCad in the Preferences window. The Schematic editor has several tabs there that allow you to customize how the editor looks and works.

Bring up the KiCad preferences window (on Windows and Linux, click Preferences —> Preferences, and on Mac, click on KiCad —> Preferences). Note that for the "Schematic Editor" group of tabs to appear, Eeschema must be running.

Below you can see the Preferences window, with the Schematic Editor group expanded.

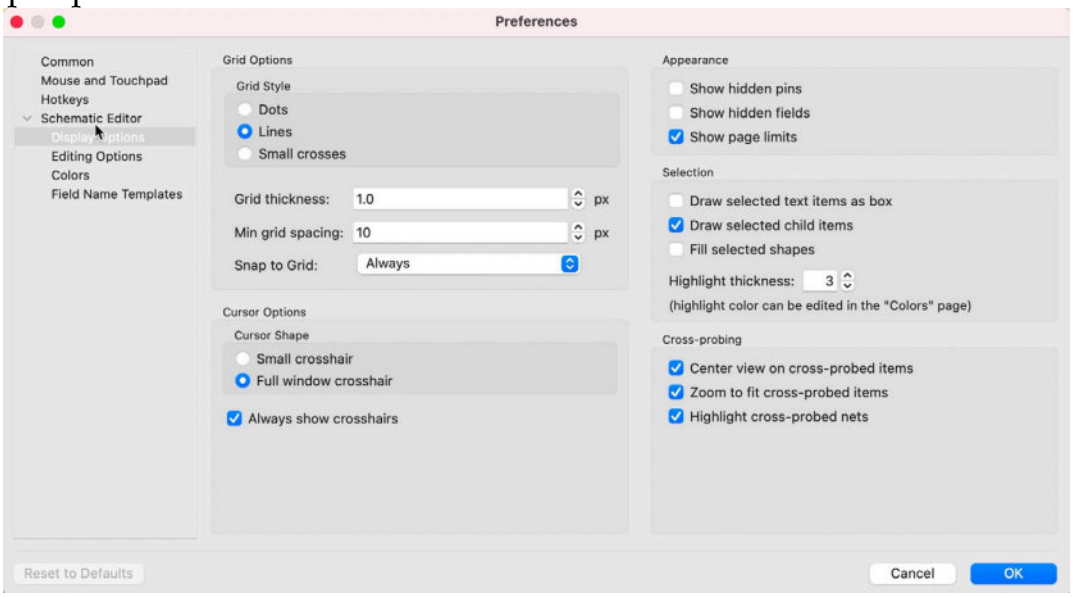


Figure 7.5.1: The Preferences window, with the Schematic Editor group expanded.

In this chapter, I will discuss each item in the Schematic Editor group.

Display options

In the Display Options tab, you can control:

- How the grid looks and works.
- The shape of the cursor.
- The appearance of hidden pins and fields.
- How selection works.
- How cross-probing works.

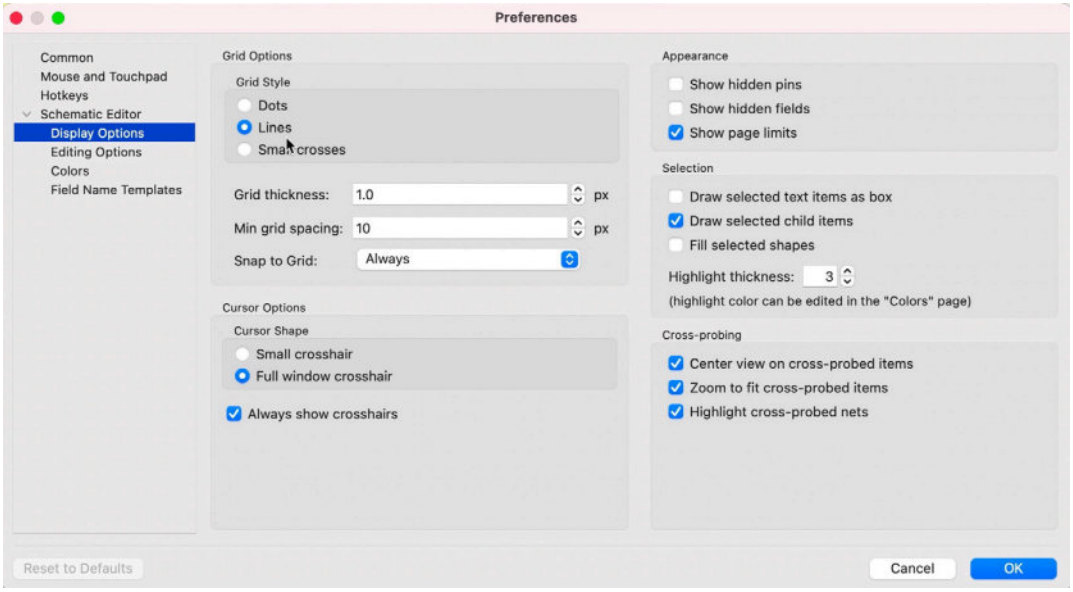


Figure 7.5.2: The Display Options tab.

Grid Options

In the Grid options group, you can set the grid style (dots, lines, crosses), thickness, minimum spacing, and enable/disable snap to grid.

In general, I keep snap-to-grid always enabled to help me position the various elements in the editor and draw the wires.

Cursor Options

The Cursor Options group allows you to switch between small and full window crosshairs. You can also choose the cursor shape from the left toolbar.

Appearance

The Appearance group allows you to choose default settings for the hidden pins and fields. You can also toggle showing the hidden pins from the left toolbar.

Below you can see an example of a hidden pin showing (left) and hiding (right).

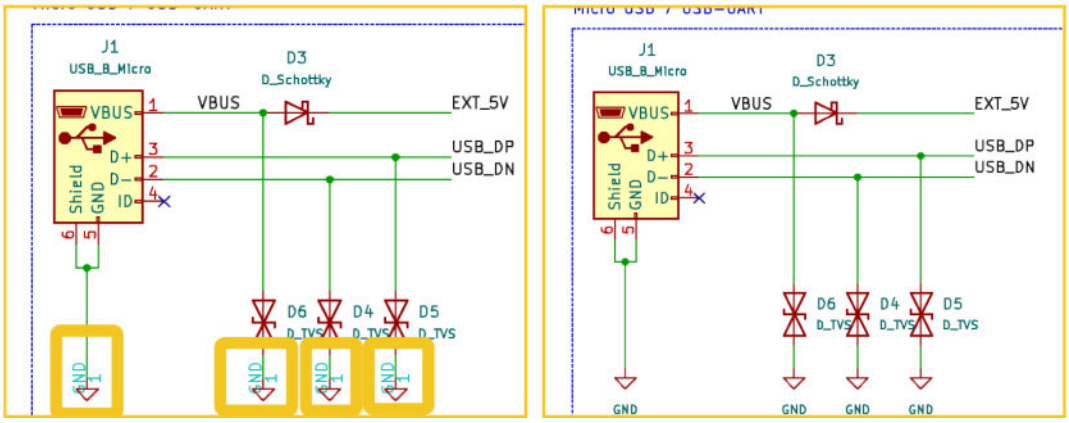


Figure 7.5.3: Hidden pin showing (left), and hiding (right).

The "show hidden fields "option, when checked, will show symbol hidden fields in their editor. This results in a very busy schematic that you probably want to avoid. See an example below:

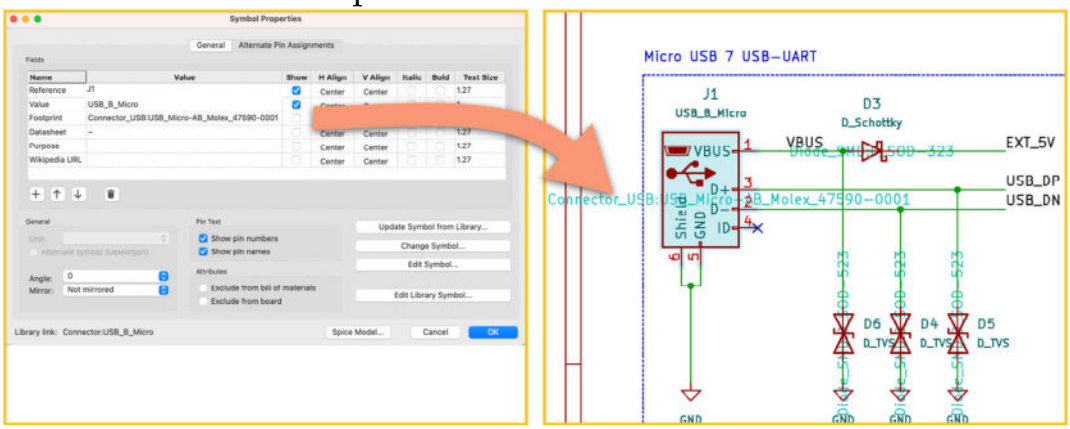


Figure 7.5.4: Hidden fields made visible.

In this example, I have set the footprint field of symbol J1 to be hidden (the "Show" attribute is unchecked). With "show hidden fields" selected, this hidden field is visible in the editor.

Selection

In the selection group, you can set the appearance of a single or a group of elements selected with the mouse. Consider the option "fill selected shapes," and look at the example below.

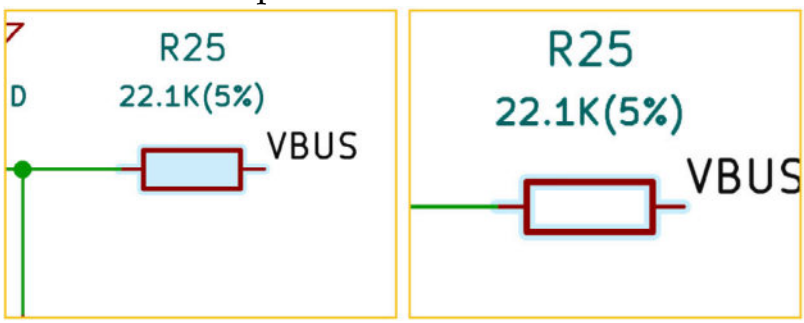


Figure 7.5.5: Fill selected option.

With Fill Selected checked, the schematic editor will fill the resistor's symbol with the highlight color (left). With Fill Selected unchecked, the border of the resistor's symbol becomes highlighted when you select it.

Feel free to experiment with the other two options in this group to learn how they work.

Cross-probing

Cross-probing is a feature that links the schematic symbols to their layout counterparts. You can arrange Eeschema and Pcbnew next to each other. If

any cross-probing options are enabled, clicking on a symbol will pan the layout editor to bring the associated footprint in view. In the example below, I have placed the two editor windows side by side. When I click on a footprint, such as "D1", the schematic editor will pan and zoom on the symbolic counterpart.

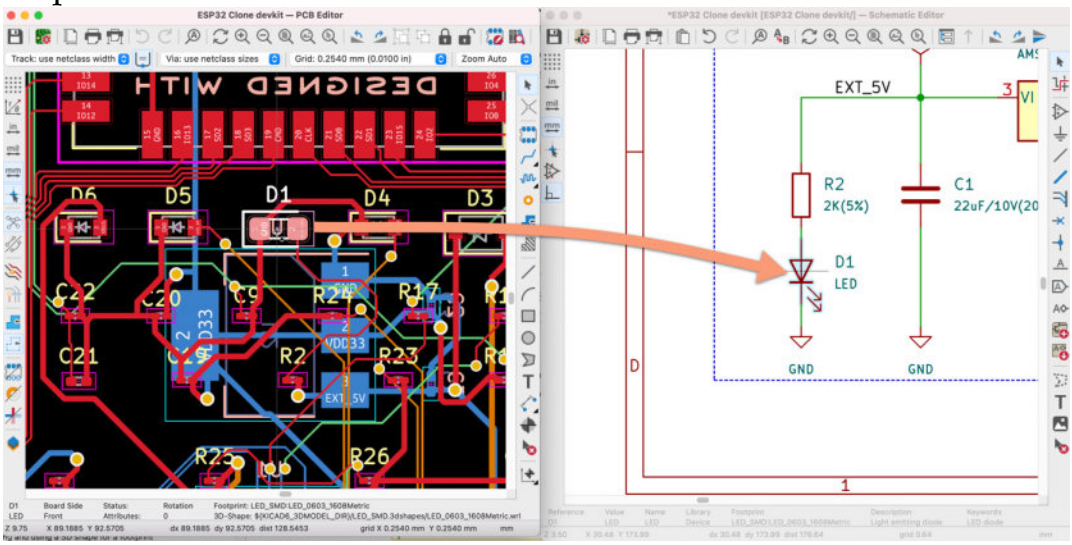


Figure 7.5.6: Cross-probing.

My preference is to keep all three options in the Cross-probing group checked so that when I click on an element in Pcbnew, Eeschema will:

1. Pan the view to center on the associated symbol.
2. Zoom in.
3. Highlight the symbol.

Editing options

Editing options allow you to configure the behavior of the schematic editor. You can see these options below:

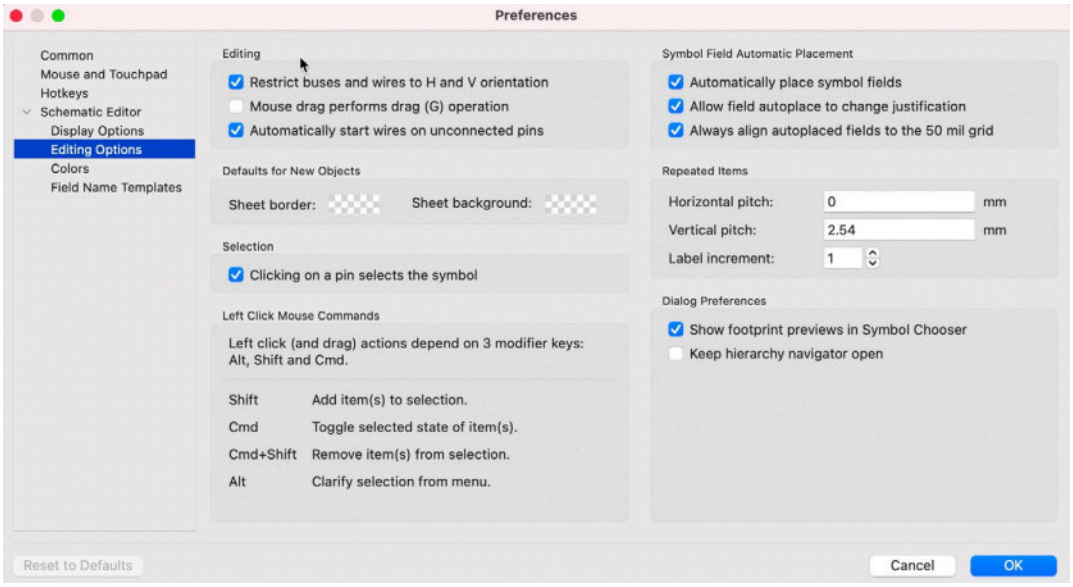


Figure 7.5.7: The Editing Options.

Most of these options are self-explanatory, especially in the Editing, Defaults for New Objects, and Selection groups.

The Repeated Items group controls a very useful and time-saving behavior; I will demonstrate. Say that you have four pins or wires. You want to use net labels to connect them to other pins. One option is to create four labels and attach them to the wires manually. Another option is to use the "repeated items" feature. With this feature, you can create the first label, say "net_1", and attach it to the first wire. Here is the starting setup:

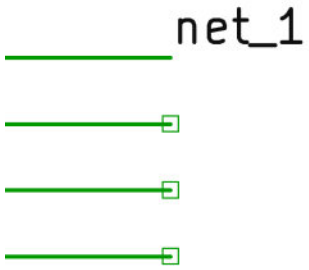


Figure 7.5.8: Setup for repeated items.

Now you can repeat the last action (creating the net label) and have the editor create and attach the remaining labels automatically. To do this, use the "repeat last action" hotkey. I have set my hotkey to be "Shift-L" (it may be different in your case, so check your setup in the "Hotkeys" tab of the Preferences window).

I type "Shift-L" four times and see the final result of this effort:

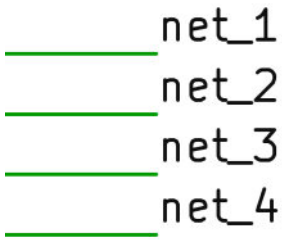


Figure 7.5.9: The last three labels are auto-created.

The last three of the labels were created and placed by the schematic editor. For this to work, the labels must finish with a number so that the auto-numbering can work, and the placement of the wires (or pins) must be equal to the vertical pitch specified in the "Related items" group. You can change these settings, but keep in mind that most symbols with multiple pins tend to observe the 2.54-inch pitch convention.

The Symbol Field Automatic Placement group options are important as they facilitate the position of fields when you rotate or flip a symbol, so I recommend that you keep them checked.

Colors

In KiCad 6, you can customize the color scheme in both editors. Apart from several built-in themes, you can create custom themes.

To create a new theme, go to the Colors tab in the Schematic Editor group, and select "New Theme" from the drop-down menu (see below).

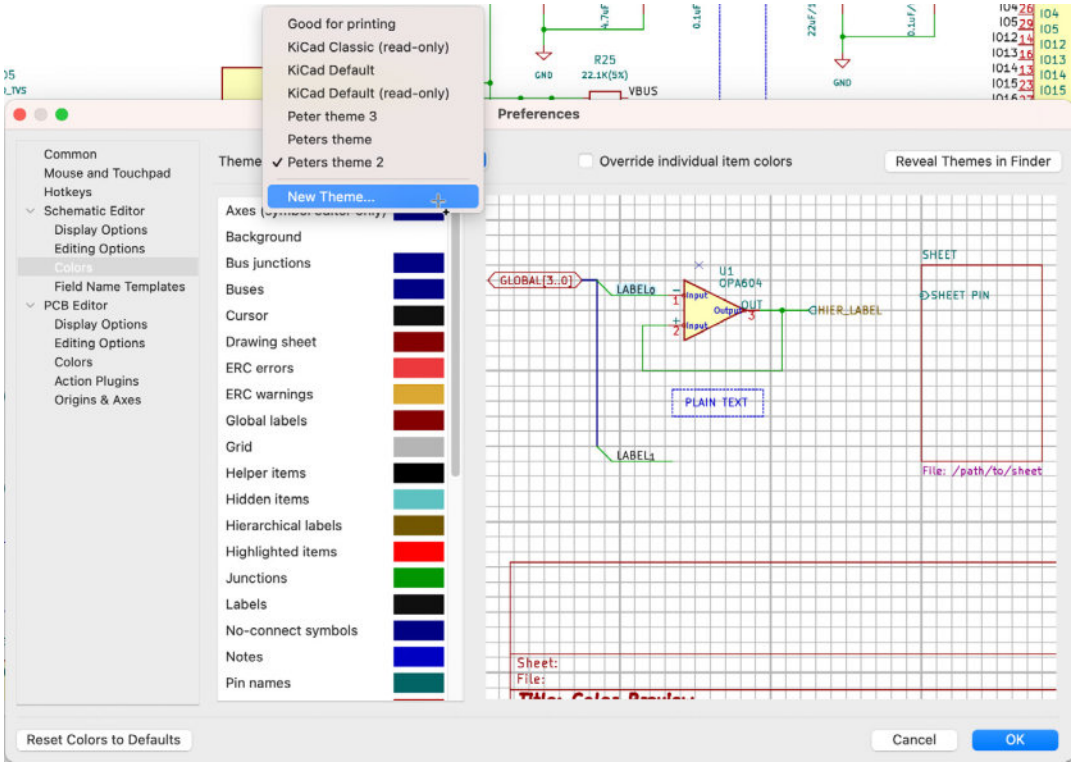


Figure 7.5.10: Create a new color theme.

Give your theme a name, and then click on the color box of the element that you want to change. This will reveal the color-chooser. Select a color and click OK (see below).

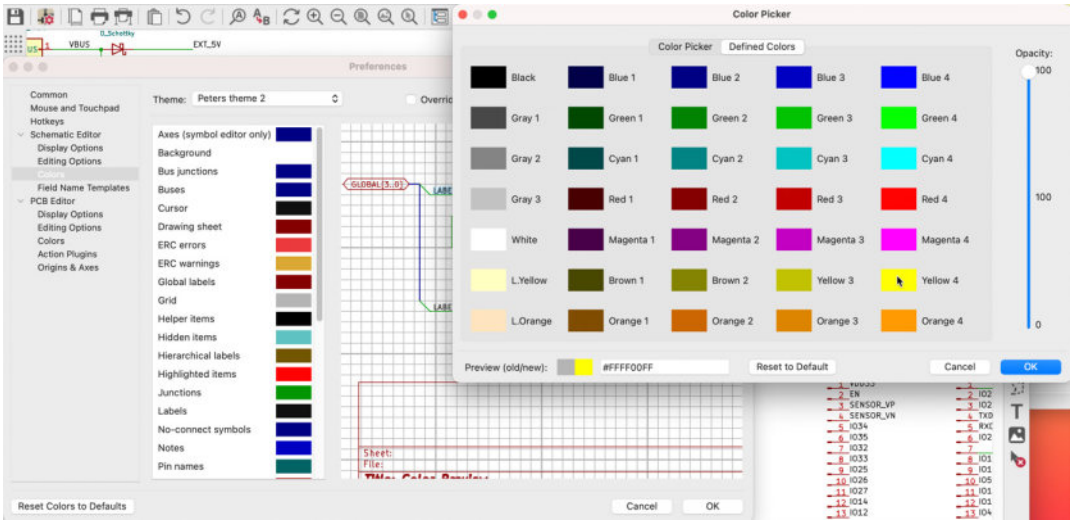


Figure 7.5.11: Select a color from the color chooser.

Click OK to dismiss the Preferences window. In the example below, I have changed the color of the grid to yellow:

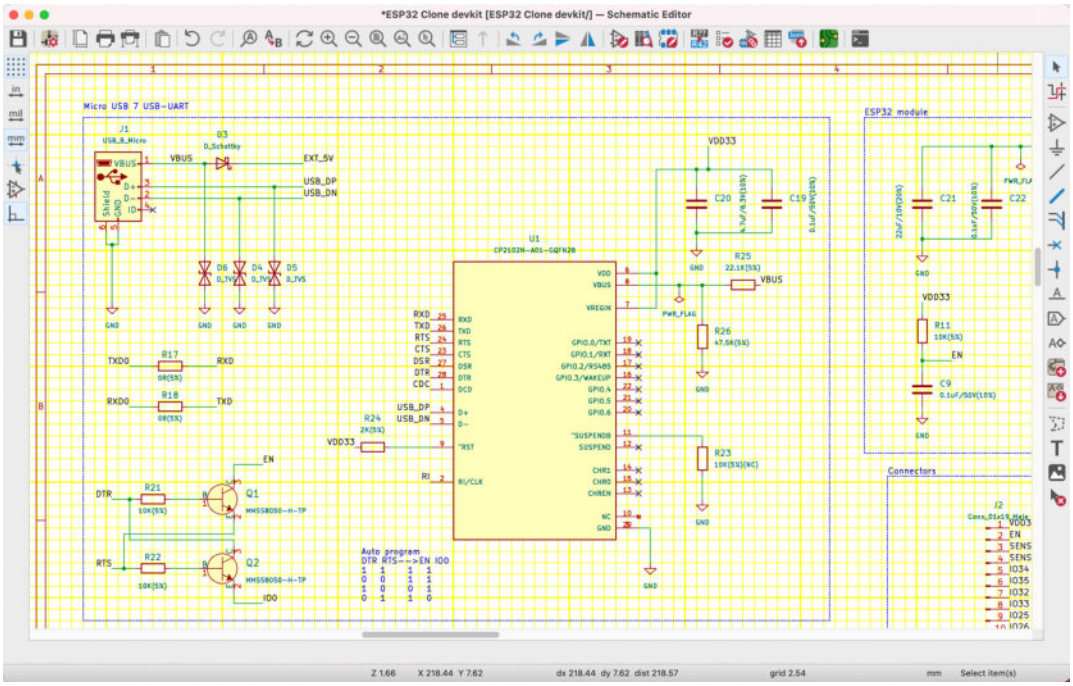


Figure 7.5.12: The grid, in yellow.

Field name templates

"Field name templates" is a new feature in KiCad 6. With this feature, you can add custom fields to the list of symbol field properties and complement the pre-defined ones.

You can learn how to use this feature in a dedicated chapter in the recipes Part.

6. How to find a symbol with the Chooser

In this chapter, you will learn how to find a schematic symbol using the Symbol Chooser tool. Once you find a symbol, you can add it to your schematic design and integrate it into your project. You have already used the Symbol Chooser in the introductory project earlier in this book. In this chapter, I will formalize this knowledge.

To invoke the Symbol Chooser window, start Eeschema, and then click on the third button from the top in the right toolbar:

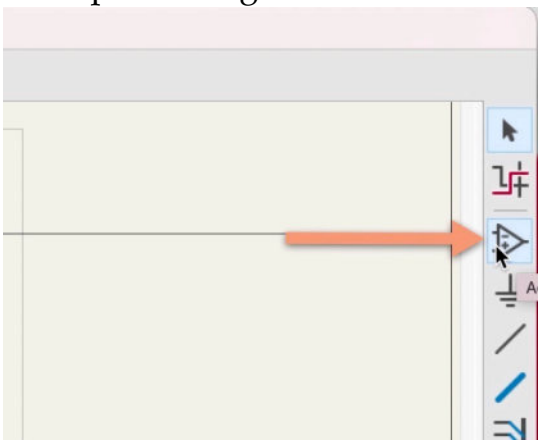


Figure 7.6.1: The Symbol Chooser button.

You can also invoke the Symbol Chooser by using the "A" hotkey. The Symbol Chooser window will appear:

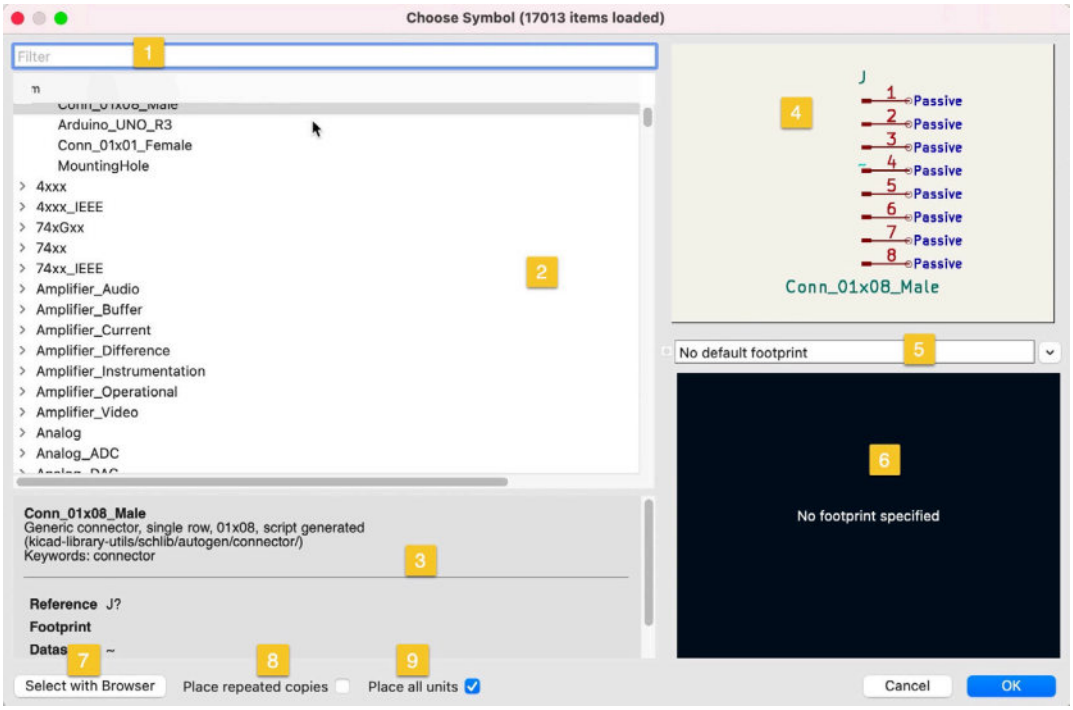


Figure 7.6.2: The Symbol Chooser window.

The window contains several widgets:

1. The search filter. Type in text to match symbols based on their name or keywords. If you know what you are looking for, this is the fastest way to find it.
2. The library browser. This tool contains a hierarchy of symbol libraries. Click on the greater-than symbol (“>”) to expand it and reveal the symbols contained within it.
3. When you select a symbol in the library browser, the symbol information will appear in the information pane. You will see its name and keywords (helpful in identifying the symbol), as well as its reference designator. If the symbol has an associated footprint or datasheet, you will also see that here.
4. Schematic symbol preview.
5. List of compatible footprints. Many symbols have a list of footprints with which they are compatible. You can associate the symbol with one of those footprints by using this dropdown menu.
6. Footprint preview. If you select a footprint using the compatible footprints dropdown, you will see the footprint’s preview here.
7. Show the Symbol Library Browser window. This window provides an alternative method to find a symbol. In Figure 7.6.3, you can see a comparison of the two windows showing the same symbol. On the left is the Symbol Library Browser, and on the right is the Symbol Chooser.

8. Place repeated copies. With this checkbox selected, you can add multiple copies of the same symbol to the sheet. After you find and choose the symbol you want to add, click on the "Place repeated copies" checkbox to enable it, then click OK to dismiss the Chooser window. Then, repeatedly click on the schematic editor sheet to place multiple copies of the same symbol.

9. Place all units. If a symbol contains more than one unit, checking the "Place all units" checkbox will add all units to the schematic sheet. An example of a symbol with more than one unit is the SW_DPST_x2 symbol.

In the figure below, you can see the Symbol Library Browser. It works in the same way as the Symbol Chooser. The main differences between the two is that:

1. The Browser does not have a filter field, so you have to navigate using the library and symbol panes.

2. Symbols that contain more than one unit are accessible via a dropdown menu in the Browser and the library and symbol hierarchy pane in the chooser. You can see an example with the SW_DPST (below.)

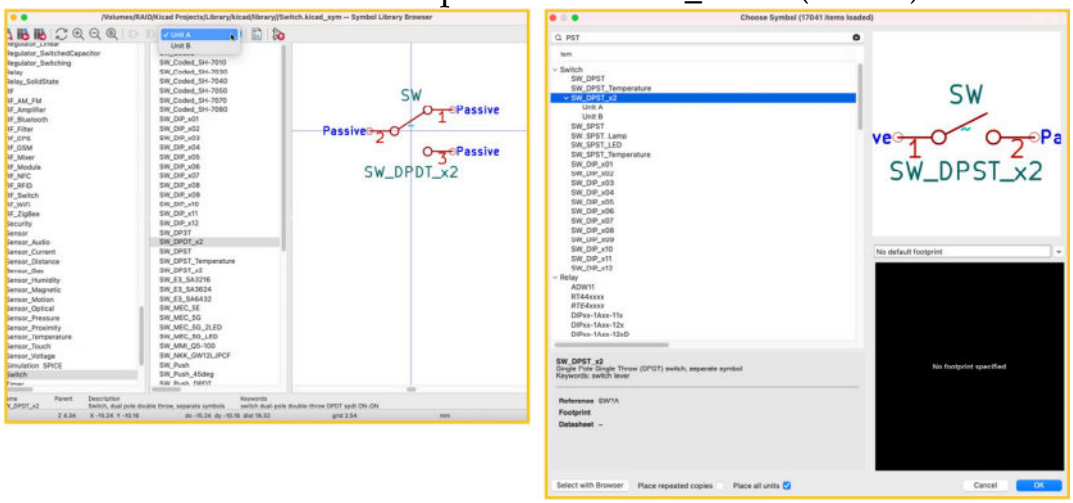


Figure 7.6.3: The Library Browser and the Symbol Chooser.

You can also invoke the Library Browser from the top menu (see Figure below):

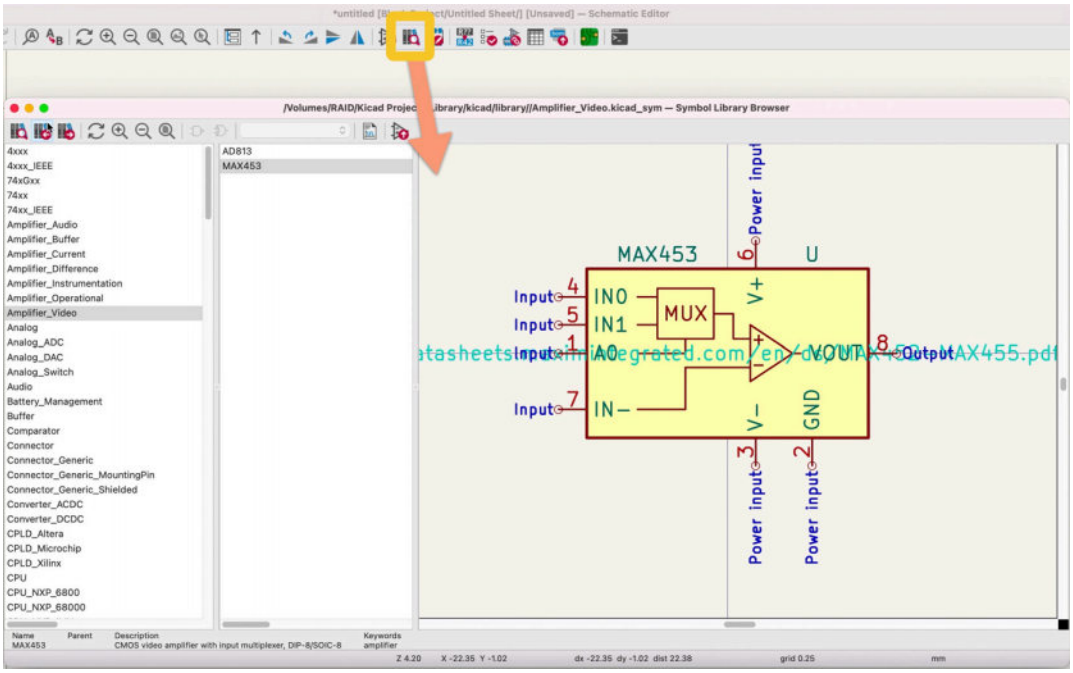


Figure 7.6.4: Invoke the Library Browser via the top menu.

Let's look at an example. Say that you need a NA555D timer IC for your schematic. Using the Symbol Chooser, search for "na555". Two options are available, as you can see below:

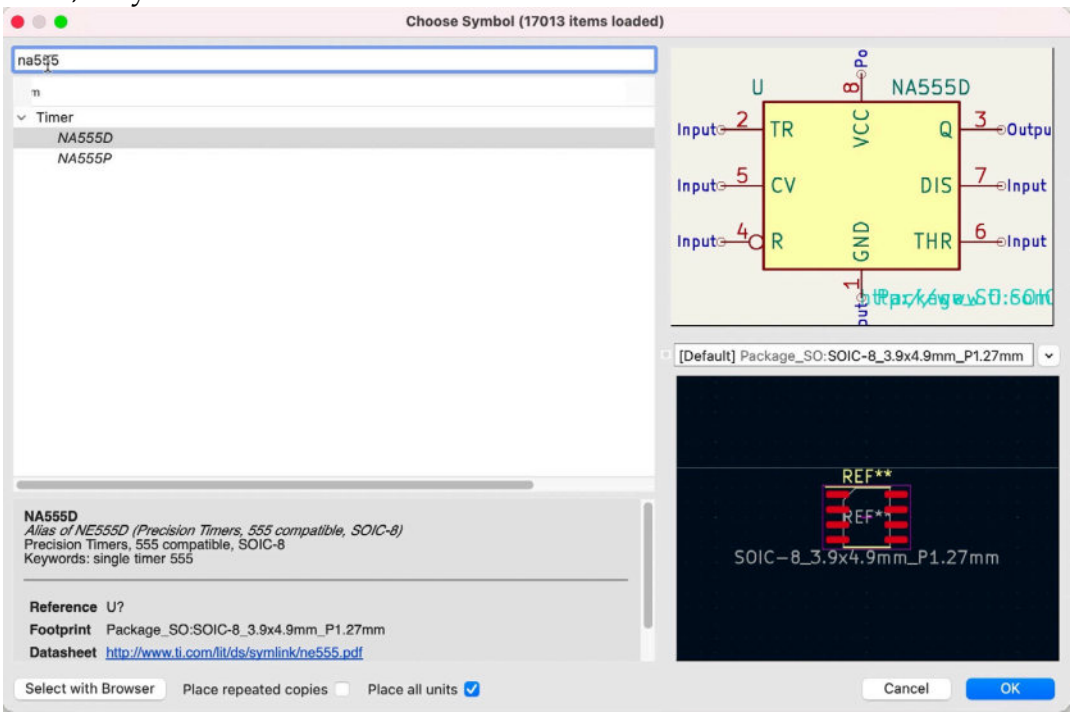


Figure 7.6.5: Using the Symbol Library Chooser.

There is a DIP and an SMD option. Both have existing associations with footprints. I have selected the "D" variant. To add it to the sheet, you can double-click on the symbol row in the chooser, or (with the symbol selected) click on OK.

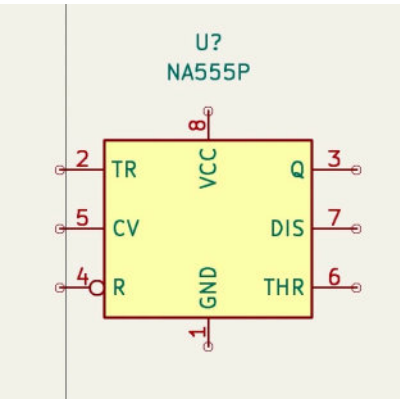


Figure 7.6.6: The selected symbol in the editor sheet.

KiCad comes with an extensive collection of symbol libraries, but it is also possible to add new libraries or create custom symbols. You can learn how to find symbols on the Internet in the next chapter in this book. You can also learn how to install external symbol libraries, and create custom symbols.

7. How to find schematic symbols on the Internet

KiCad comes with an extensive collection of symbol libraries, but it is also possible to find new symbols on the Internet and add them to your KiCad instance. You can then use the new symbol for your current or future projects. In this chapter, you will learn how to to find a symbol on the Internet.

One way to look for a schematic symbol (or a footprint or a 3D shape) is to use a search engine like Google. You can use a search string line "kicad symbol for na555". A search like this will inundate you with advertisements and loosely related content. A page with an actual symbol download may appear towards the bottom of the first page of the results. In my experience, such a search is not very helpful.

My preferred method is to look for symbols in a small number of reputable libraries repositories. This is my shortlist, in order of personal preference:

1. KiCad's symbol library repository at https://kicad.github.io/symbols/. KiCad contributors add new symbols frequently. It is possible that in the time elapsed since I downloaded my copy of the libraries, the symbol that I am looking for was added to the repository.
2. Snapeda at https://www.snapeda.com/. Snapeda is a repository with millions of parts for all major CAD software applications. Rarely, a part I am looking for does not exist in Snapeda. In most cases, Snapeda will provide everything you need for a part: symbol, footprint, and very often the 3D shape.
3. Octopart at https://octopart.com/. I find Octopart to be as good as Snapeda in terms of finding symbol and footprint libraries. You can use Octopart as an alternative or a complement to Snapeda.
4. Ultralibrarian at https://www.ultralibrarian.com/. Similar top Snapeda and Octopart.

Several libraries are worth downloading and installing because they contain parts that most people tend to use frequently. These are:

1. Digikey's KiCad library at https://github.com/Digi-Key/digikey-kicad-library.

2. Sparkfun's KiCad library at https://github.com/sparkfun/SparkFun-KiCad-Libraries.
3. Freetronics's KiCad library at https://github.com/freetronics/freetronics_kicad_library.

You can learn how to install these libraries in the next chapter.
For now, let's return to finding on using Ultralibrarian as an example. To follow along, go to the Ultralibrarian website, create a free account, and log in. Go to the home page and type the search string in the search box. I am searching for a NA555 timer IC part:

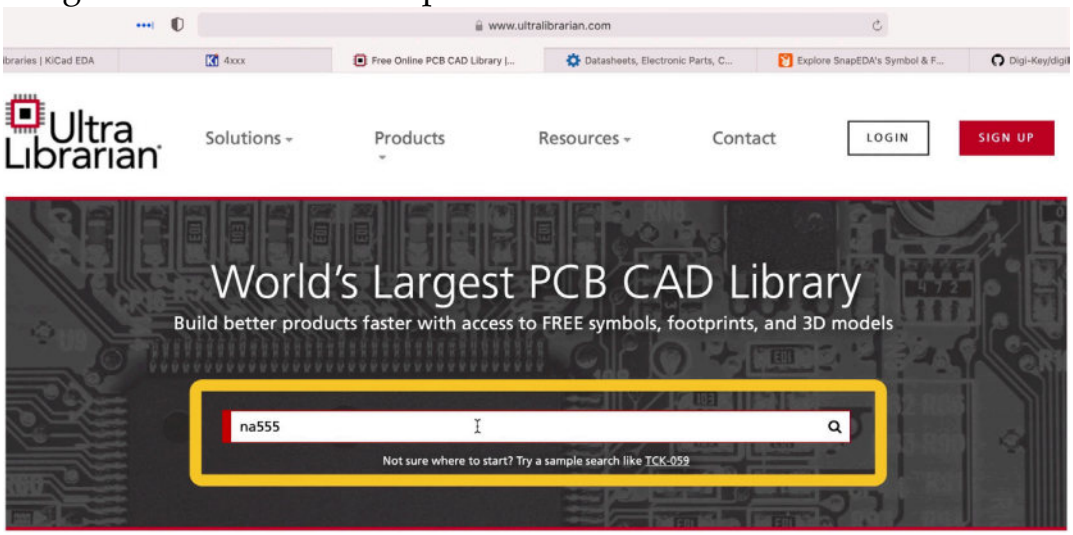


Figure 7.7.1: Search for the NA555 timer IC in Ultralibrarian.

The results will appear. Each row provides information about the part: availability from a supplier, price per unit, and the available models (symbol, footprint, 3D).

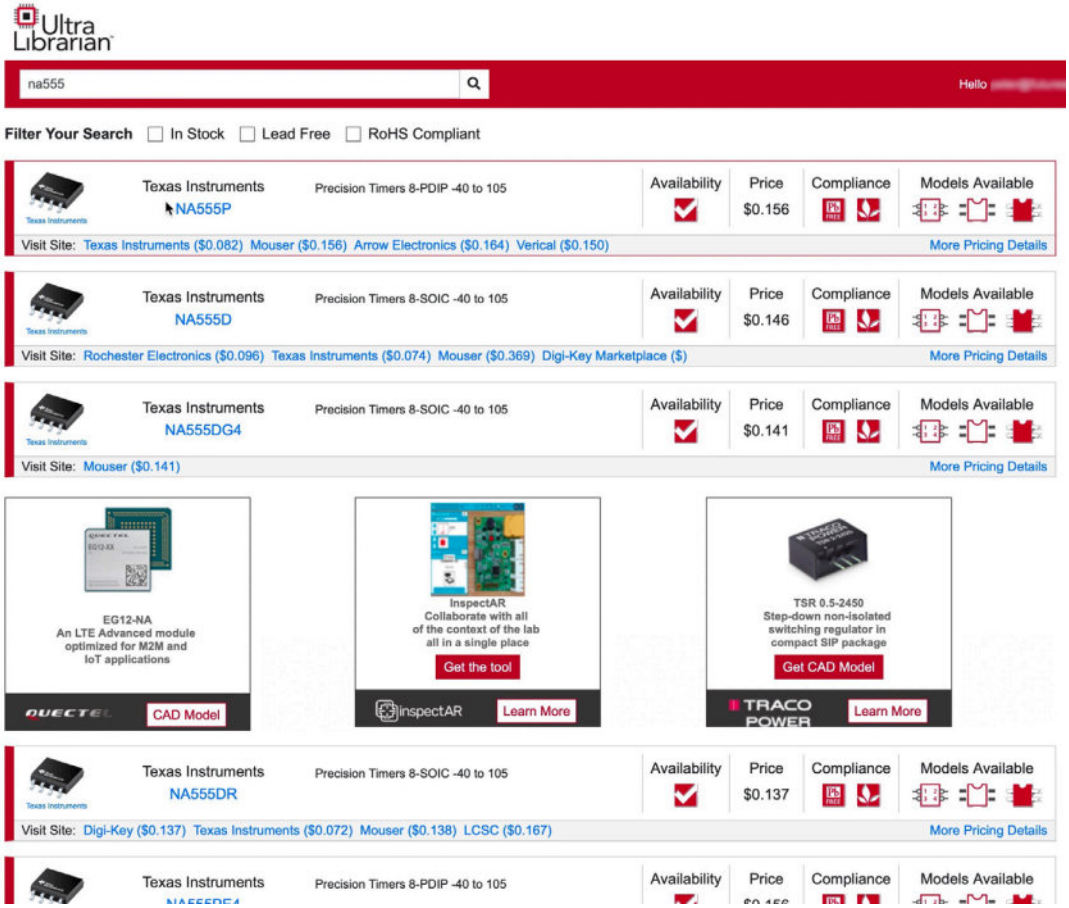


Figure 7.7.2: Search results for “na555”.

You can browse through the results to find the one that best matches your requirements. I am looking for a variant of the timer IC that uses a DIP package and found it on the second page of the results. Click on the part to get to the part details and download page:

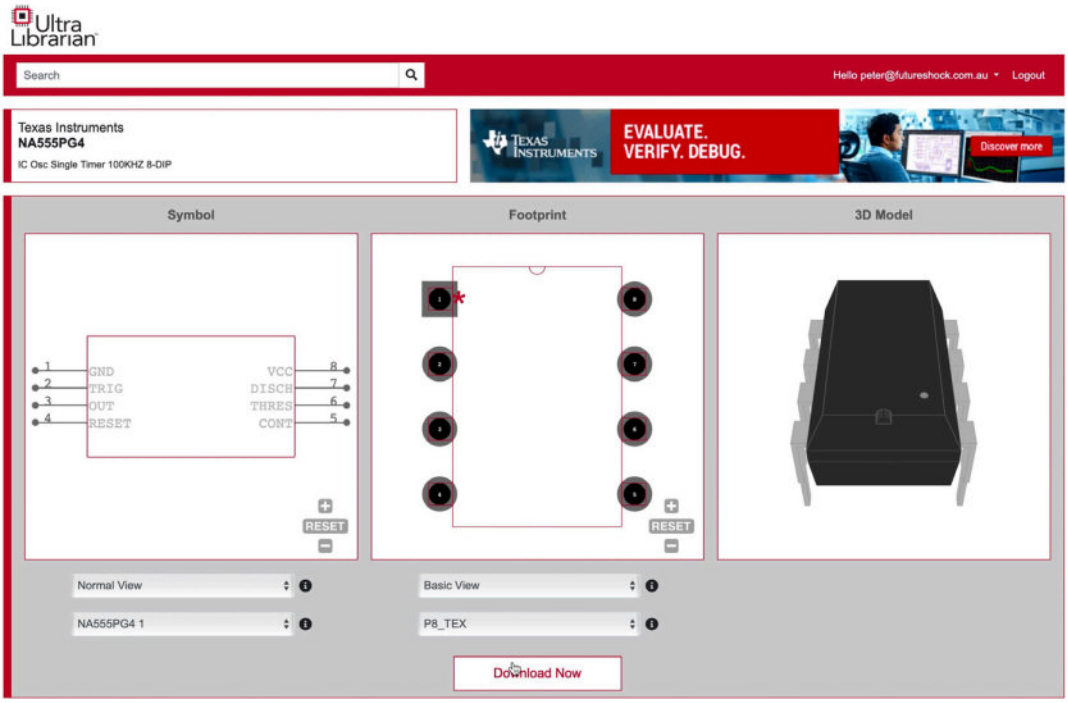


Figure 7.7.3: Details and download page for “NA555PG4”.

On the details and download page, you can see a preview of the available models. You can use dropdown widgets to switch to different views, such as Basic or Detailed. To download the models, click on the ”Download Now” button. This will reveal the Download options, where you can select KiCAD as the CAD format. Notice that for KiCad, Ultralibrarian offers the symbol and footprint, but not the 3D shape.

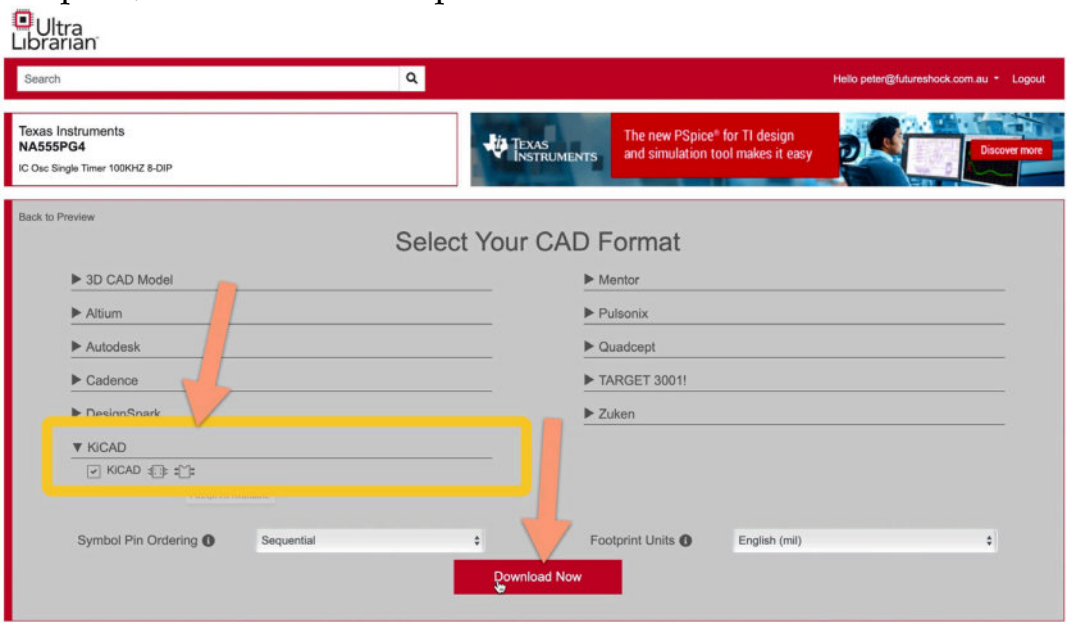


Figure 7.7.4: Select your download CAD format.

Click on “Download Now“ again to start the download. Expand the downloaded ZIP file to see its contents. The contents look like this:

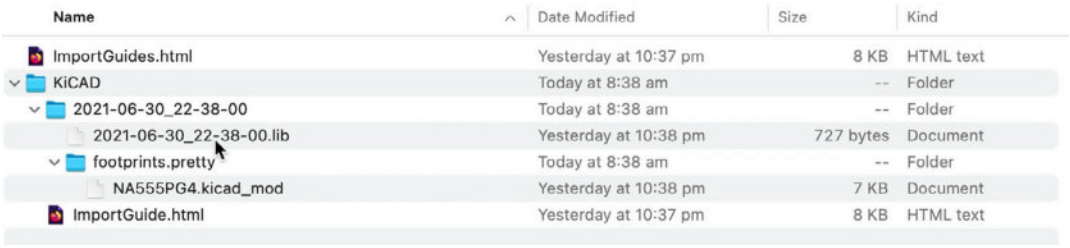


Figure 7.7.5: The download models.

The ZIP archive of this example contains ".lib" (symbol) and ".kicad_mod" (footprint) files. I will concentrate on the symbol file and install it before I can use it in the editor.

First, change the name of the ".lib" file to make it easier for you to recognize it. As you can see in the figure above, the current name is a date. I prefer to change this to the model of the part that the symbol represents, like this:

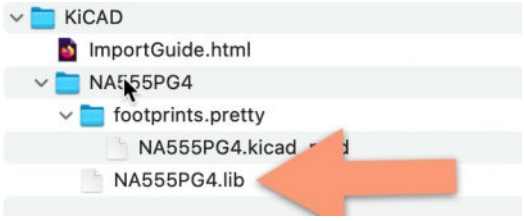


Figure 7.7.5: Renamed ".lib" file.

Next, copy the folder that contains the part (named "NA555PG4 in the example above) to a folder where you keep your KiCad libraries. Below you can see mine:

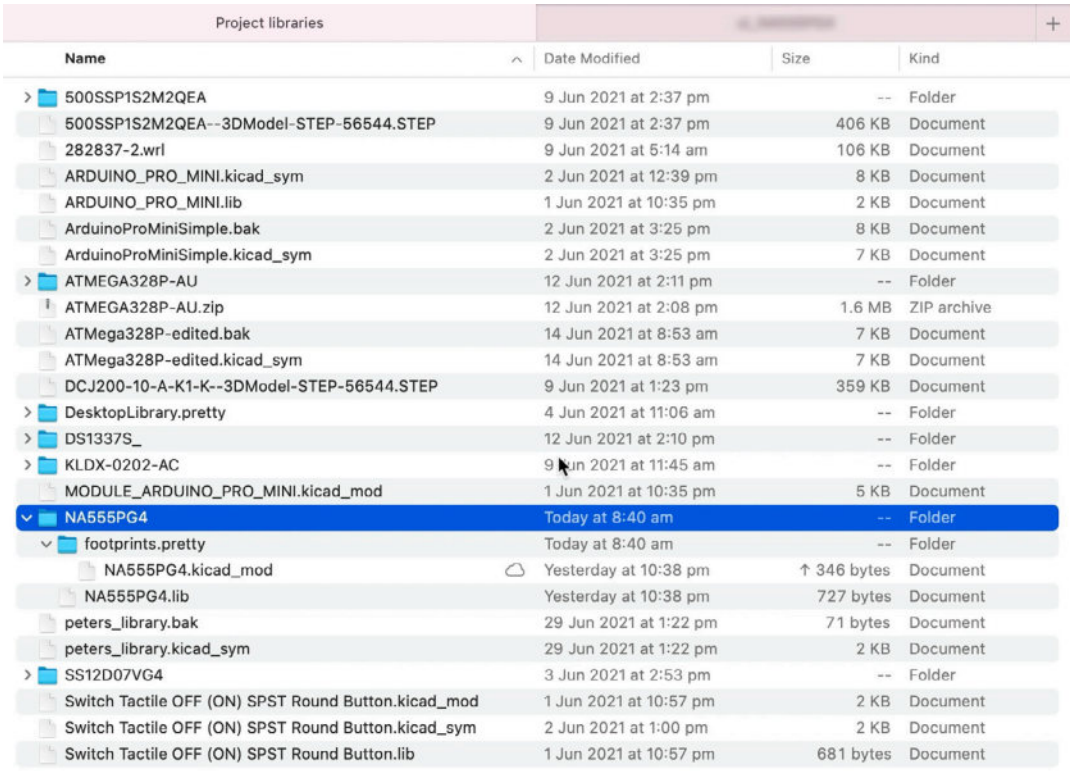


Figure 7.7.6: My project libraries folder.

The new library is now in place for KiCad. Go to Eeschema, and select "Manage Symbol Libraries" from the Preferences menu. In the Symbol Libraries window that appears, click on the "Project Specific Libraries" tab ("1" in Figure 7.7.7 below). I will install this library for my current project only, but you can follow the same process to install it for all projects by doing this work in the "Global Libraries" tab.

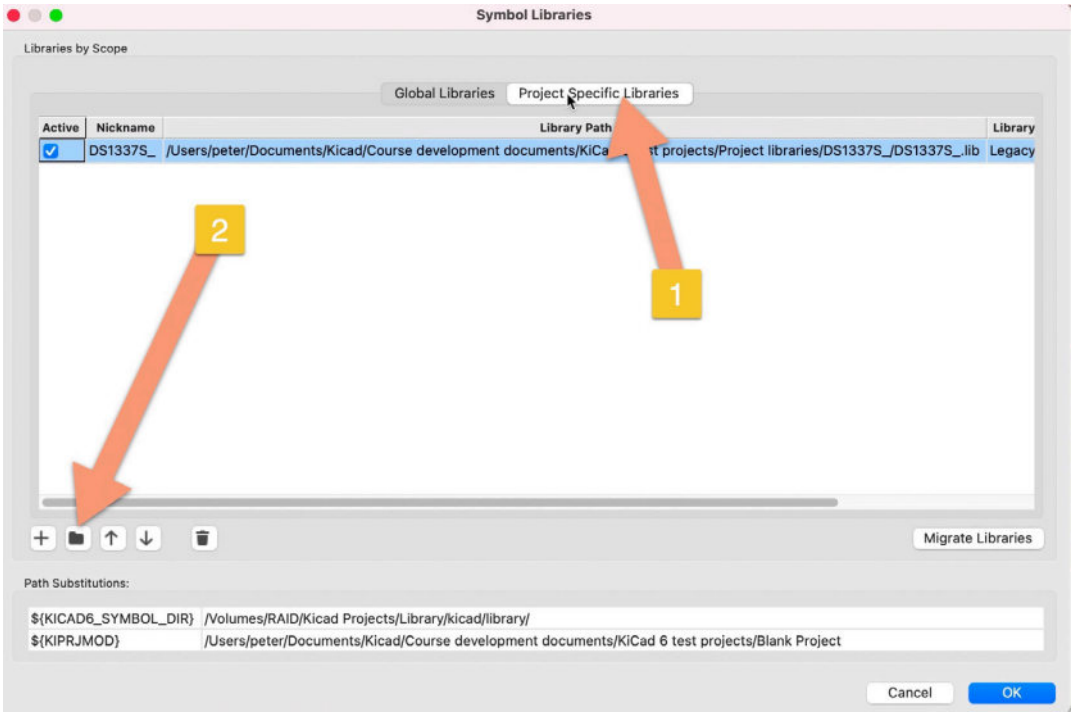


Figure 7.7.7: Installing a new library.

Click on the folder button ("2") to bring up the file browser, and navigate to the location where you save the new ".lib" file. Select it ("1" in Figure 7.7.8 below), and click Open ("2").

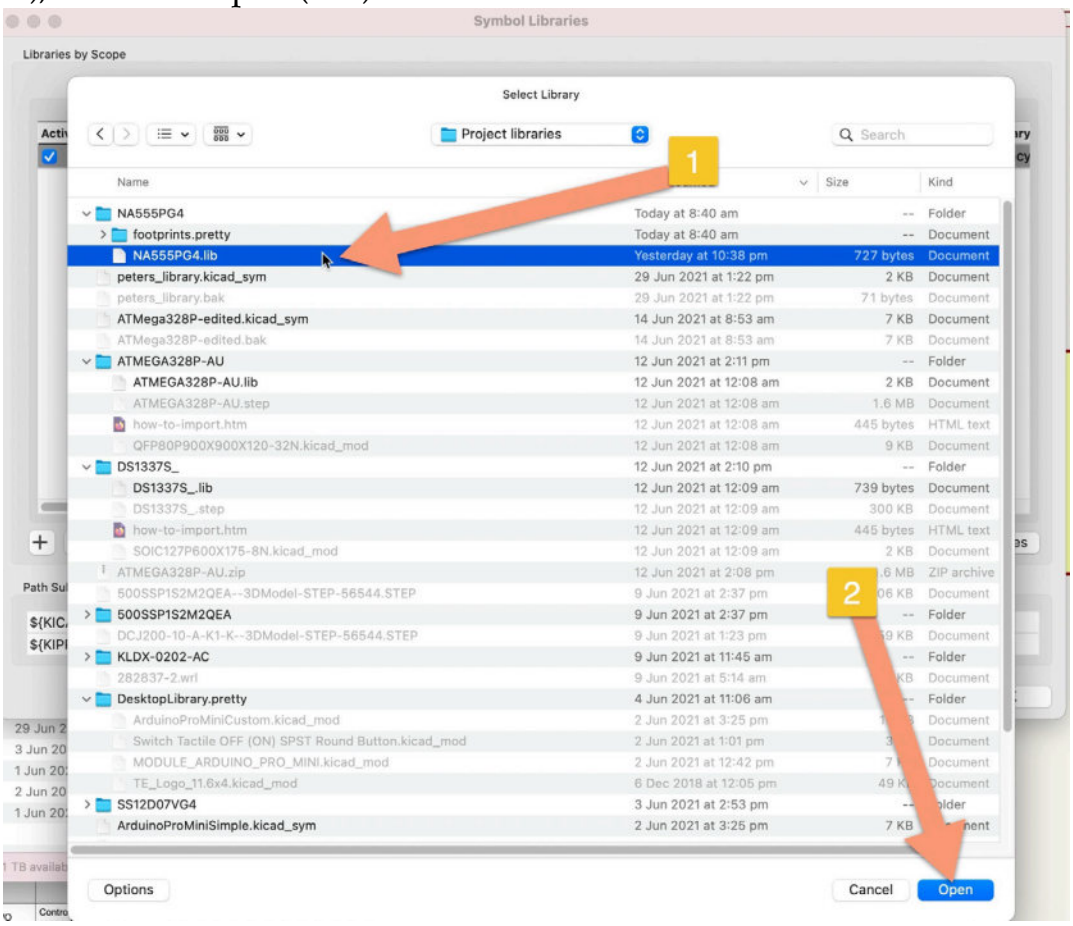


Figure 7.7.8: Navigate to the new ".lib" file.

The new ".lib" file now appears in a new row in the Symbol Library window, and it is automatically active:

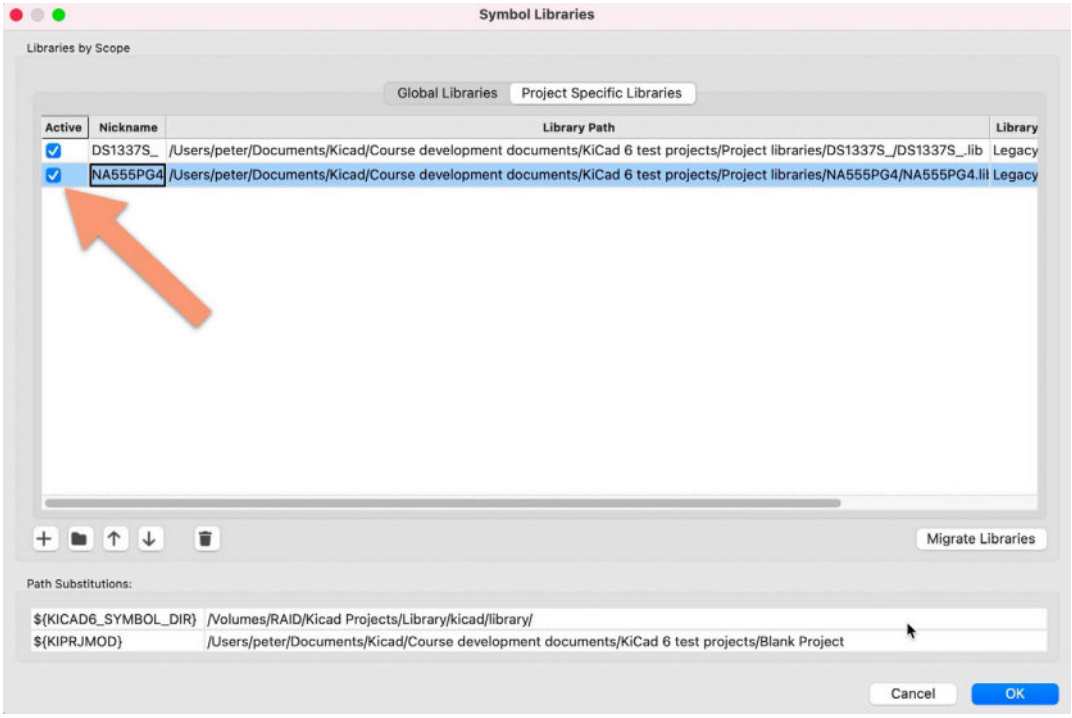


Figure 7.7.9: The new symbol is active and ready to use.

The new symbol is now active and ready to use. Click OK to dismiss the libraries window, and bring up the symbol chooser to search for the new symbol:

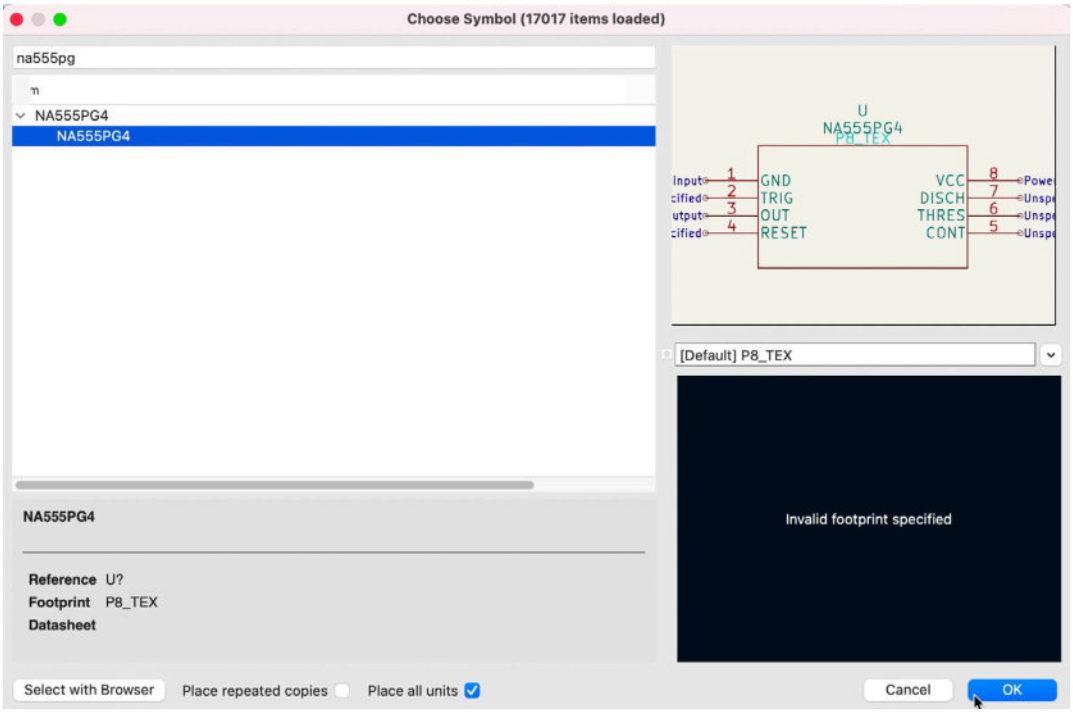


Figure 7.7.10: The newly imported symbol appears in the Symbol Chooser.

You can double-click to add this symbol to the editor sheet and continue your work in your schematic. You can download symbols from Snapeda and Octopart, and use the same method to import them into KiCad so you can use them.

8. How to install symbol libraries in bulk

In this chapter, you will learn how to install schematic symbols in bulk. I remind you that in the previous chapter, you learned how to install a single schematic symbol to KiCad. The process for installing multiple symbols is similar.

You will want to install multiple symbols when you download a collection of symbols from a symbol repository, such as those from Digikey or Sparkfun. In this chapter, I will show you how to do this using Digikey's symbol collection.

Start by using your Brower to access Digikey's KiCad repository at https://github.com/Digi-Key/digikey-kicad-library . It looks like this:

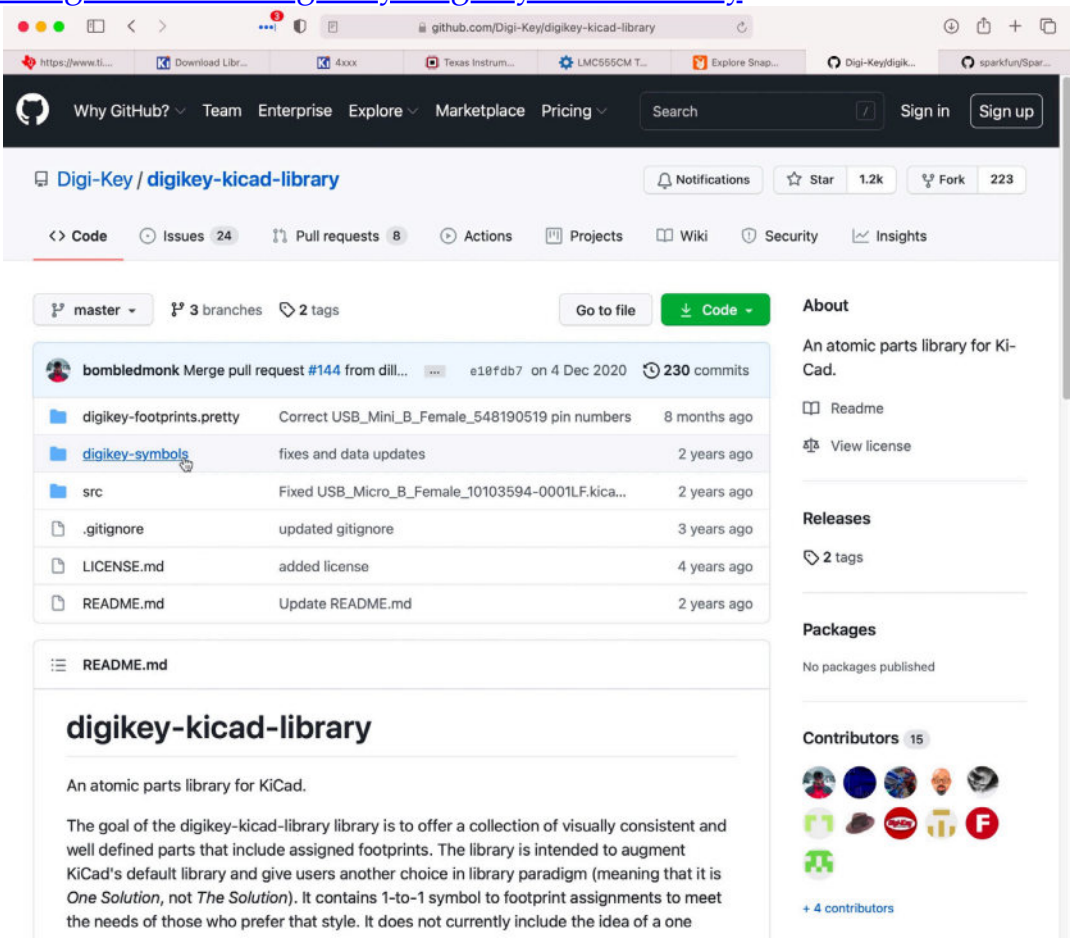


Figure 7.8.1: Digikey's KiCad repository.

The repository contains footprints (in the folder "digikey-footprints.pretty") and symbols (in the folder "digikey-symbols"). Click on the symbols directory to see inside.

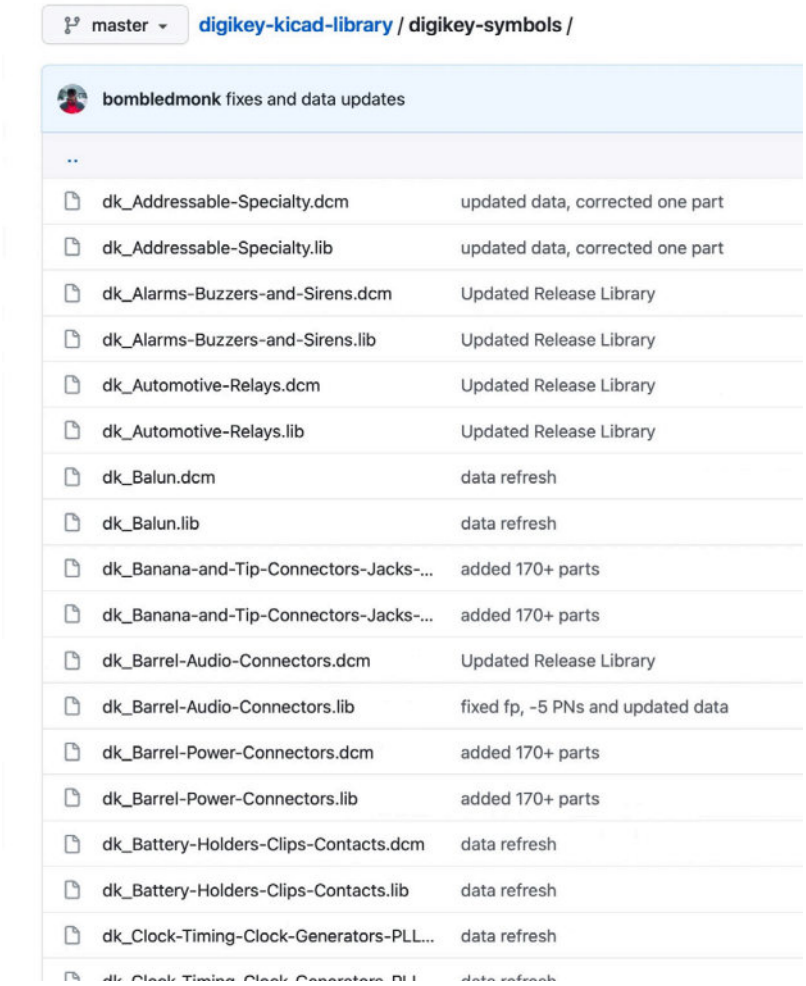


Figure 7.8.2: Digikey's symbols collection.

This folder contains a collection of symbols (with the ".lib" extension) and their metadata (with the ".dcm" extension). Let's import all of these symbols to KiCad.

Download the entire repository by clicking on the green "Code" button and selecting "Download ZIP." You must be at the root of the repository to see the green button:

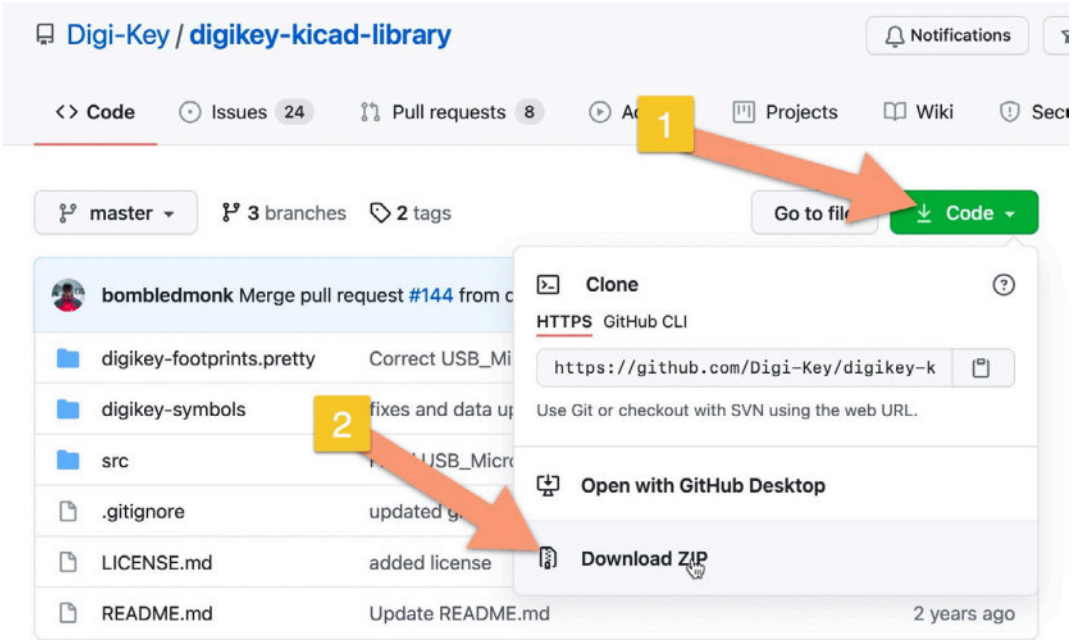


Figure 7.8.3: Download the Digikey repository.

Once the download is complete, expand the ZIP file and copy the resulting folder to your preferred location. I have a central folder where I keep all my third-party libraries. Below you can see the freshly downloaded Digikey folder in my project libraries folder (notice I have also copied the ".pretty" folder that contains the Digikey footprints collection):

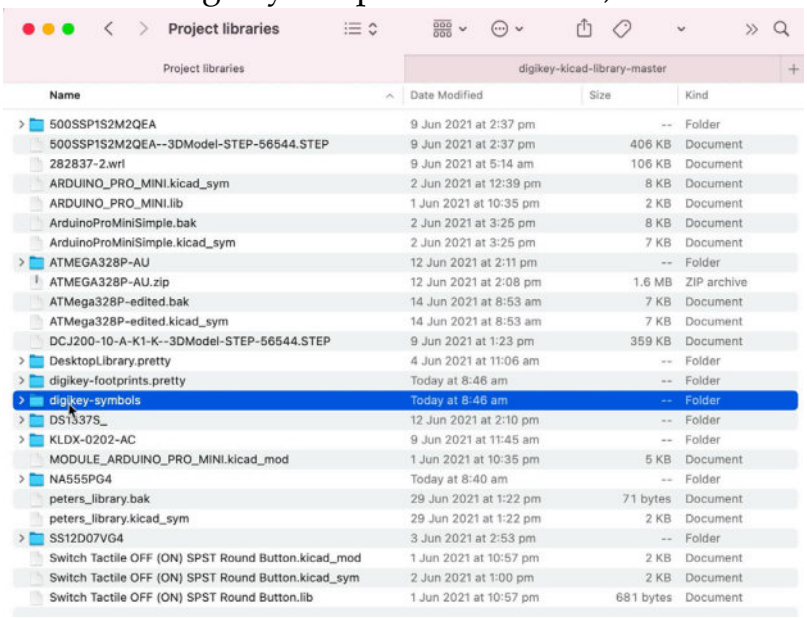


Figure 7.8.4: The new Digikey folder in my KiCad project libraries folder.

To import the new symbol collection in KiCad, open Eeschema. Bring up the symbols library manager (under the Preferences menu). I will import the Digikey symbol collection into the Global Libraries list so that all projects can use the new symbols.

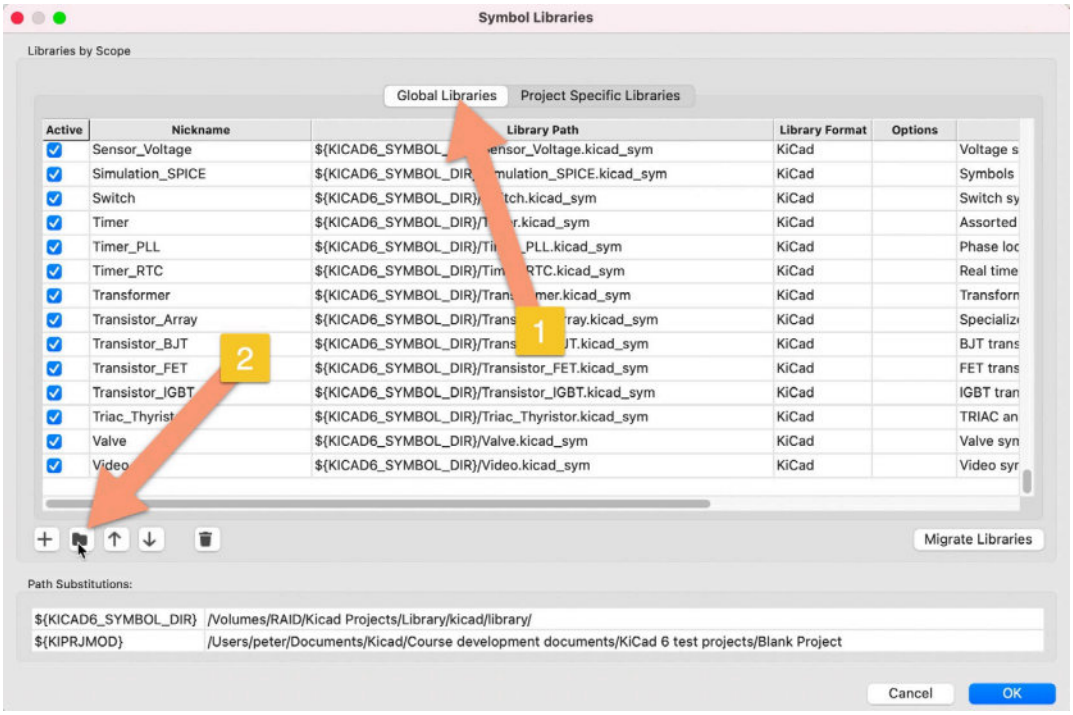


Figure 7.8.5: The library manager in Eeschema.

Click on the Global Libraries tab and then on the folder button to bring up the file browser. Navigate to the location where you have stored the Digikey symbol files (".lib"). Select all symbols in the folder (or only the ones that you want). In my example below, I have selected all of the ".lib" files in the folder:

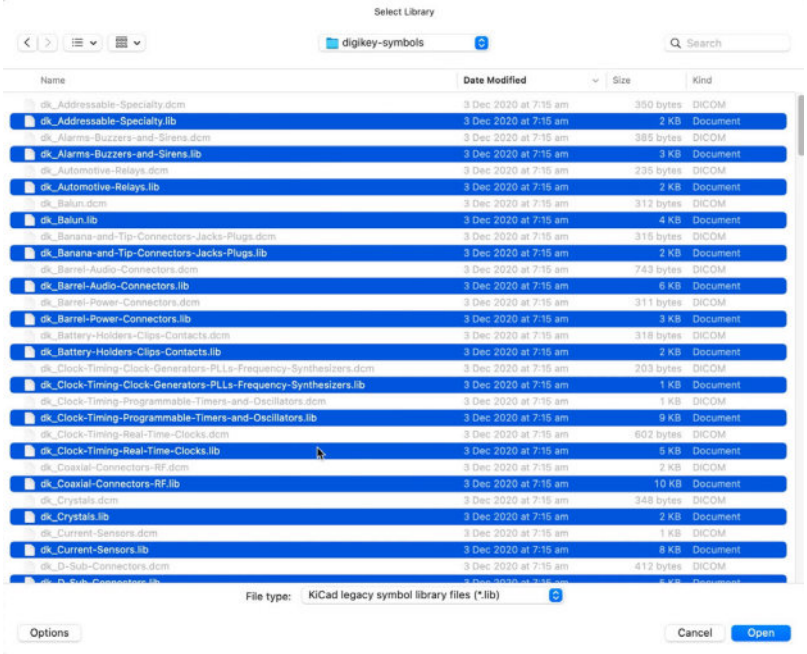


Figure 7.8.6: Importing all symbols in the collection.

Click Open, and inspect the resulting new rows in the symbol libraries list. The symbols that originate from Digikey have a nickname with the prefix "dk_":

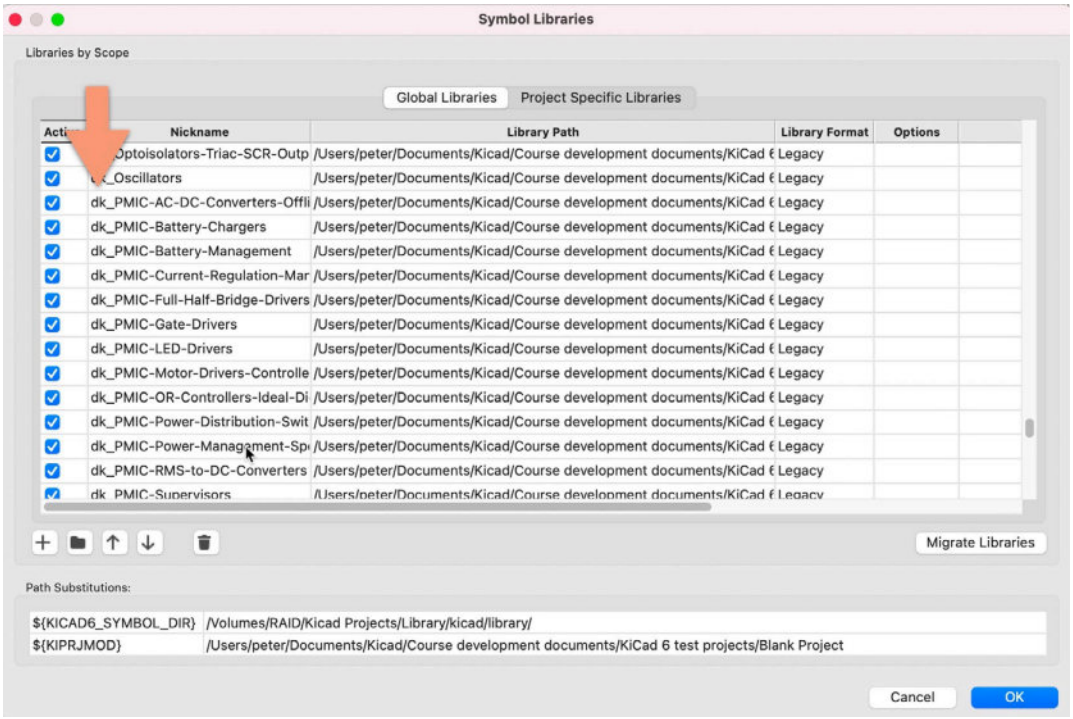


Figure 7.8.7: The Digikey symbols are ready to use.

The new symbols are ready to use. Click OK to dismiss the symbol libraries window, and bring up the symbol chooser for a quick test. In my example below, I am browsing the symbols in the Digikey collection:

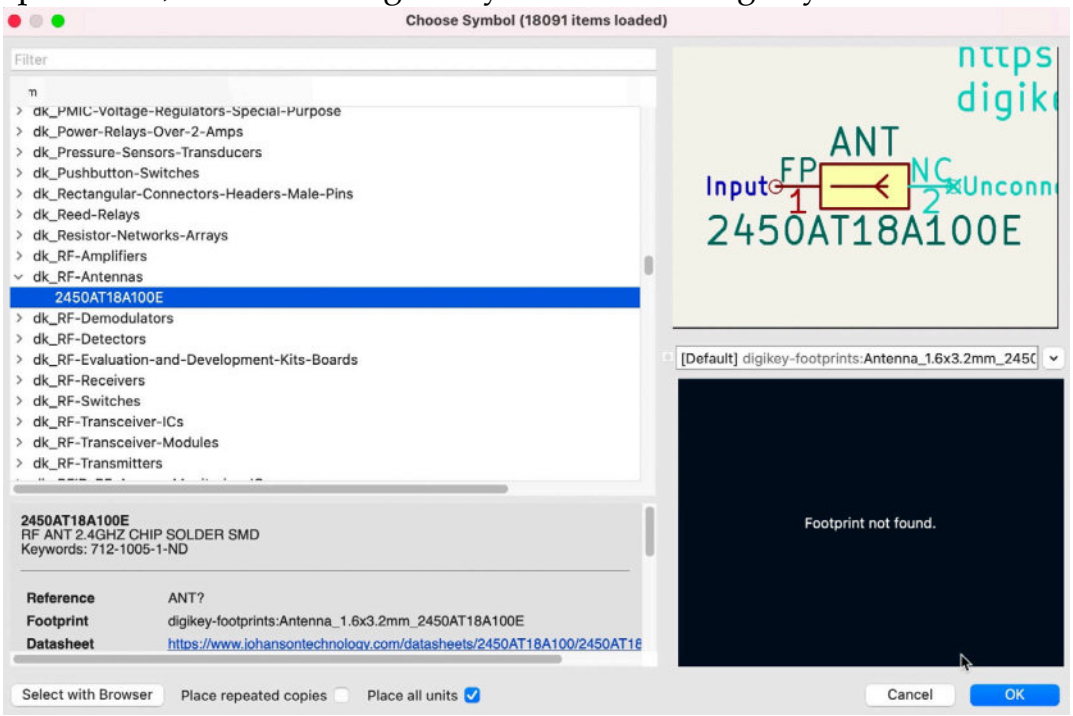


Figure 7.8.8: Browsing the Digikey symbols.

You can double-click a Digikey symbol to add to your schematic as you would with any other symbol. In the example above, notice that a footprint association is available in the footprints dropdown; however, it is

unselectable. The footprint previewer is also empty. This is because, at this point, I have not yet imported the footprints that come with the Digikey collection. You can learn how to do this in the relevant chapter in the next part of this book.

9. How to create a custom symbol

Imagine that you are looking for a symbol that you can't find. You have already searched through KiCad's libraries, online library repositories, and you have even searched for it on Google. There is only one option left: create this symbol using Kicad's schematic editor.

In this chapter, you will learn how to create a custom symbol with the help of an example. By the end of this chapter, you will have made a symbol for the NA555P timer integrated circuit, like the one below:

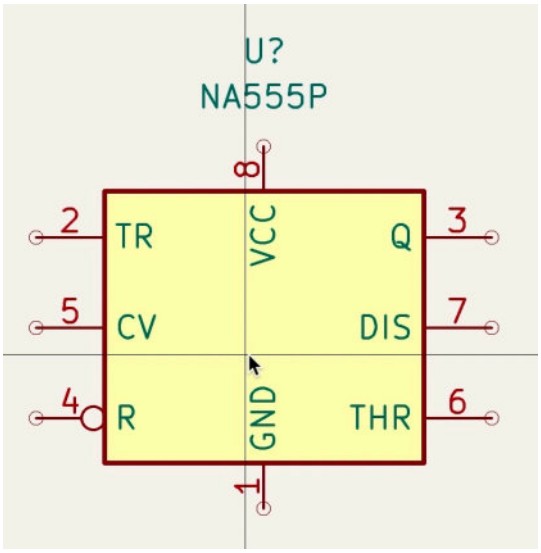


Figure 7.9.1: A custom symbol for the NA555P timer.

To understand the process of creating a custom symbol, you must first understand the elements that make up a symbol. Looking at Figure 7.9.1 above, those elements are:

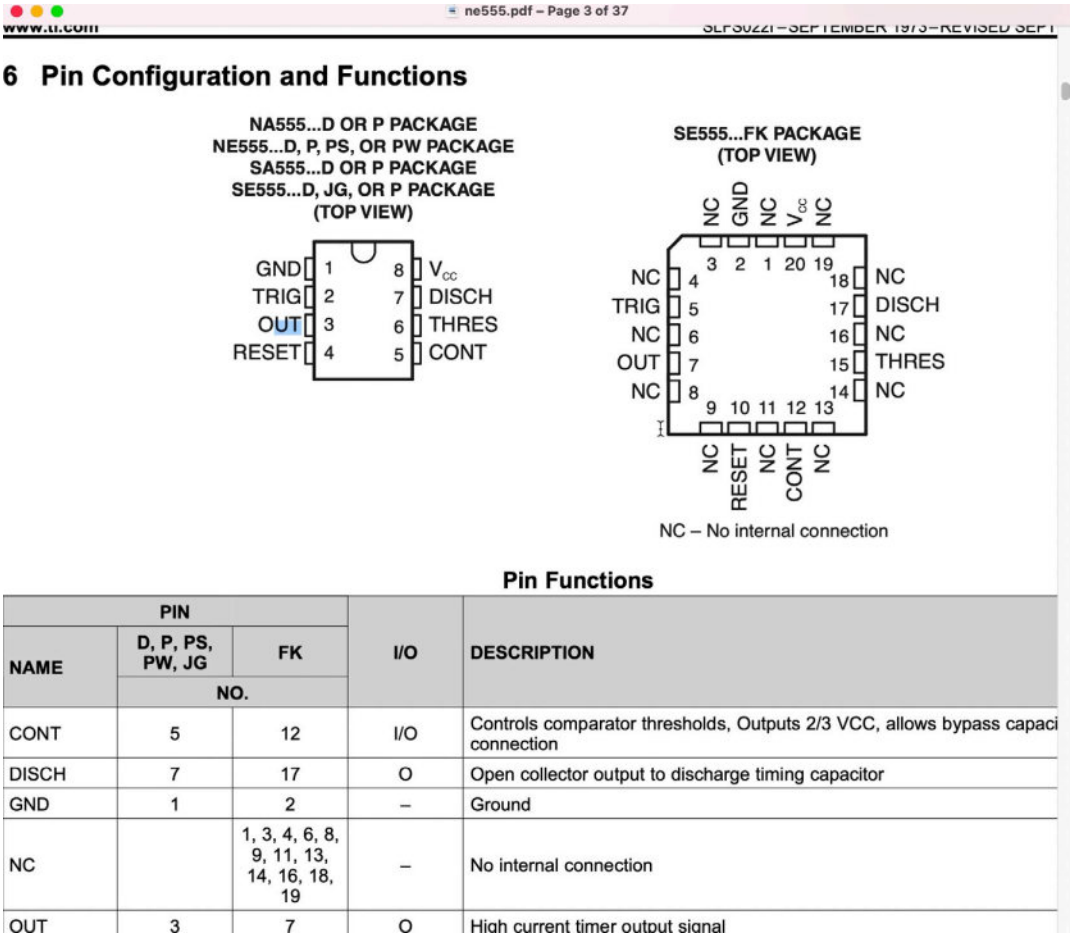
1. A rectangle or other closed shape that represents the body of the symbol. Usually, the shape is filled with a color (yellow in the example above) and has pins attached to its outline.
2. One or more pins. The pins have two sides: the one that attaches to the symbol's outline and the side that connects to wires or labels. The latter is marked with a small circle. A pin may also have a type. The type of a pin may be communicated with a symbol, such as a large circle in the example of pin 4 (see above). Each pin also has a number (that appears over the pin line) and a name (that appears across the pin inside the

symbol's box. Typically, the Vcc pin is placed on the top side of the symbol's rectangle, GND on the bottom, and the functional pins on the left and right sides. Whenever possible, group pins of the same kind together. For example, you can place all inputs on the left side and all outputs on the right.

3. A designator. In the example above, the designator is “U”. You can consult this page for more information on reference designators.
4. A name for the symbol. In the example above, the name is "NA555P". The name of the symbol appears in the "Value" field of its properties window.

Apart from the properties, I listed above, a custom symbol in KiCad may have properties stored in custom fields. You can learn about custom fields in a later chapter in this book. To keep things simple in this example, I will only address the build-in symbol fields.

Before you start work on a custom symbol (or footprint), it is best to gather as much information about it as you can. The component's datasheet is the best source of information that is useful for creating custom symbols and footprints. For the example here, I will be drawing information from the component's datasheet:



ne555.pdf – Page 3 of 37

6 Pin Configuration and Functions

NA555...D OR P PACKAGE
NE555...D, P, PS, OR PW PACKAGE
SA555...D OR P PACKAGE
SE555...D, JG, OR P PACKAGE
(TOP VIEW)

SE555...FK PACKAGE
(TOP VIEW)

NC – No internal connection

Pin Functions

NAME	PIN D, P, PS, PW, JG NO.	PIN FK NO.	I/O	DESCRIPTION
CONT	5	12	I/O	Controls comparator thresholds, Outputs 2/3 VCC, allows bypass capaci 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

Figure 7.9.2: The component's datasheet contains valuable information for creating custom symbols and footprints.

To start the symbol editor, click on "Symbol Editor" in the main KiCad project window or the button from the top toolbar in Eeschema:

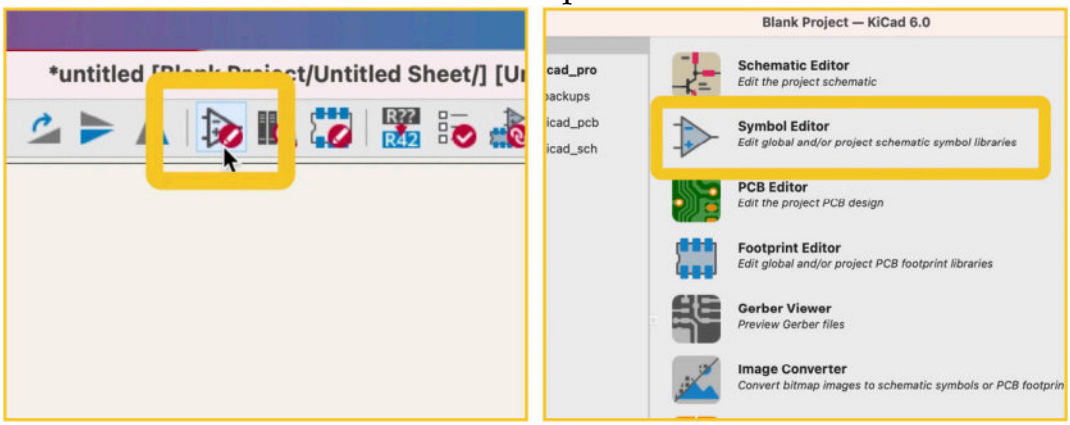


Figure 7.9.3: Starting the symbol editor from Eeschema (left) and the main project window (right).

Either way, you will see the symbol editor:

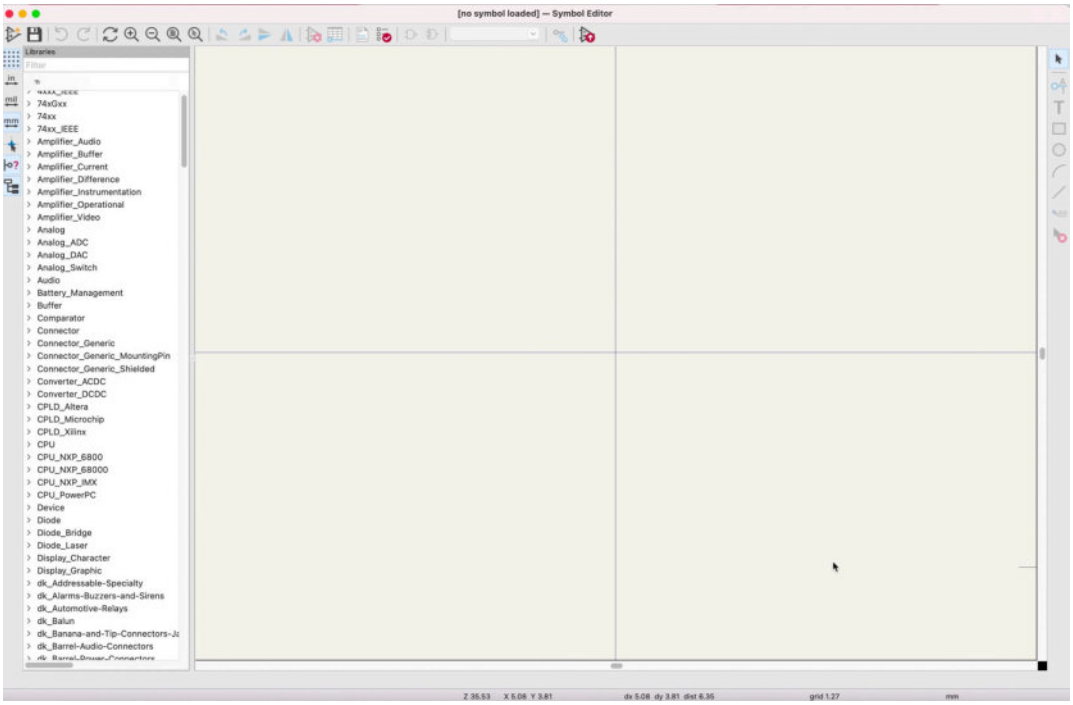


Figure 7.9.4: The symbol editor.

Let's start the process of creating a new symbol. I want to save the new symbol inside a new symbol library. With the symbol editor open, click on File and then "New Library." I will make the new library available to my current project only, so I select "Project" in the "Add to Library Table" and click OK. The symbol editor will add the new library automatically to my project, so I will not need to do this manually.

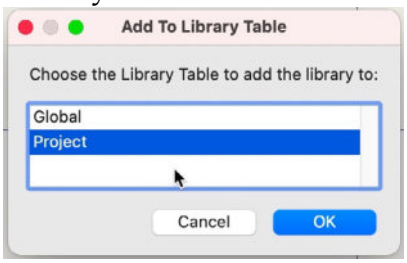


Figure 7.9.5: The symbol editor can add the new symbol to the symbol library table.

Click OK to dismiss the "Add To Library Table." The symbol editor will ask you to choose a location for the new library. Select a suitable location, and click Save. I have saved mine in a folder that contains other libraries that I use in my projects.

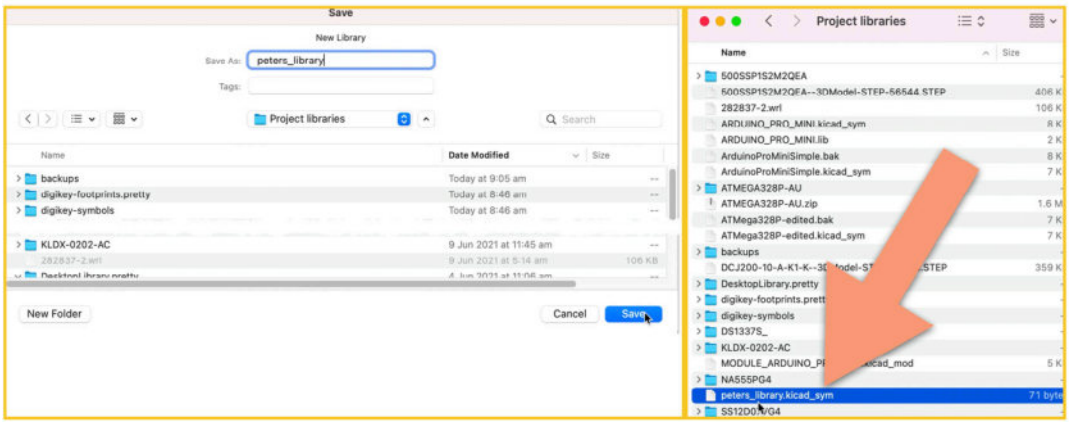


Figure 7.9.6: Saving the new library.

You will see a new file with the ".kicad_sym" extension appear in the same location that you chose.

Now that you have a new library, you must first select it before saving a new symbol in it. In the figure below, notice the text "no symbol in the symbol editor's library filter, type the first few letters of your new library to find it (see "1" in the figure below), and then click on it to select it ("2"). Next, click on the New Symbol button from the top toolbar ("3").

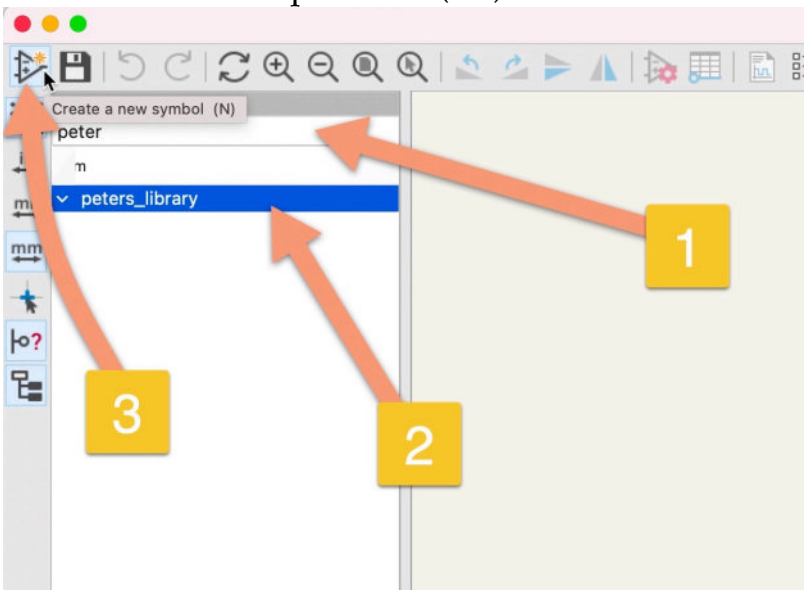


Figure 7.9.7: Select the new library and create a new symbol.

This will bring up the New Symbol window. Type in the symbol name ("NA555_PD") and confirm the reference designator. I have used the suffix "_PD" to denote symbols that I have created. You can leave the rest of the properties as they are. See my example below:

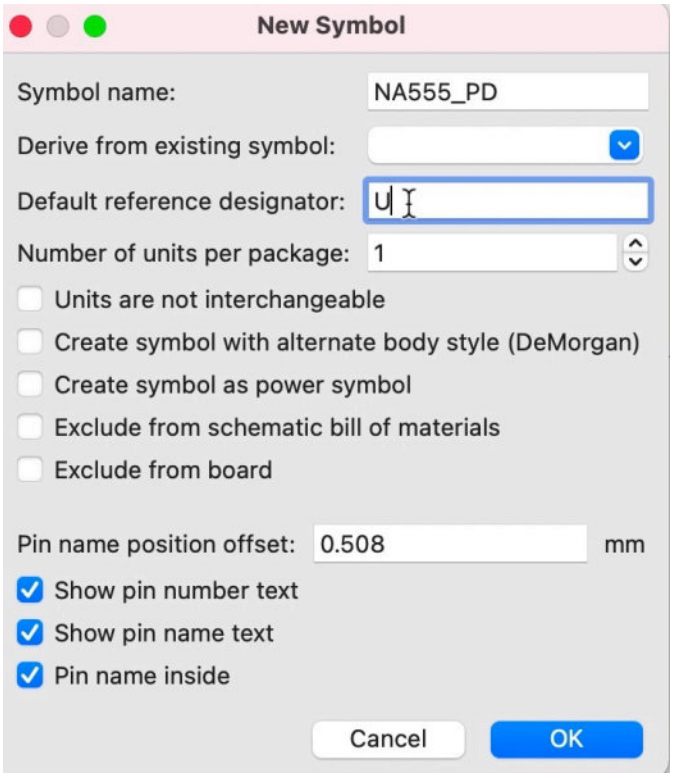


Figure 7.9.8: The properties of the new symbol.

Click OK to close the window. You will see the symbol name and designator in the editor. Drag them towards the top of the editor window so that the center of the editor sheet is empty, as you can see "1" below:

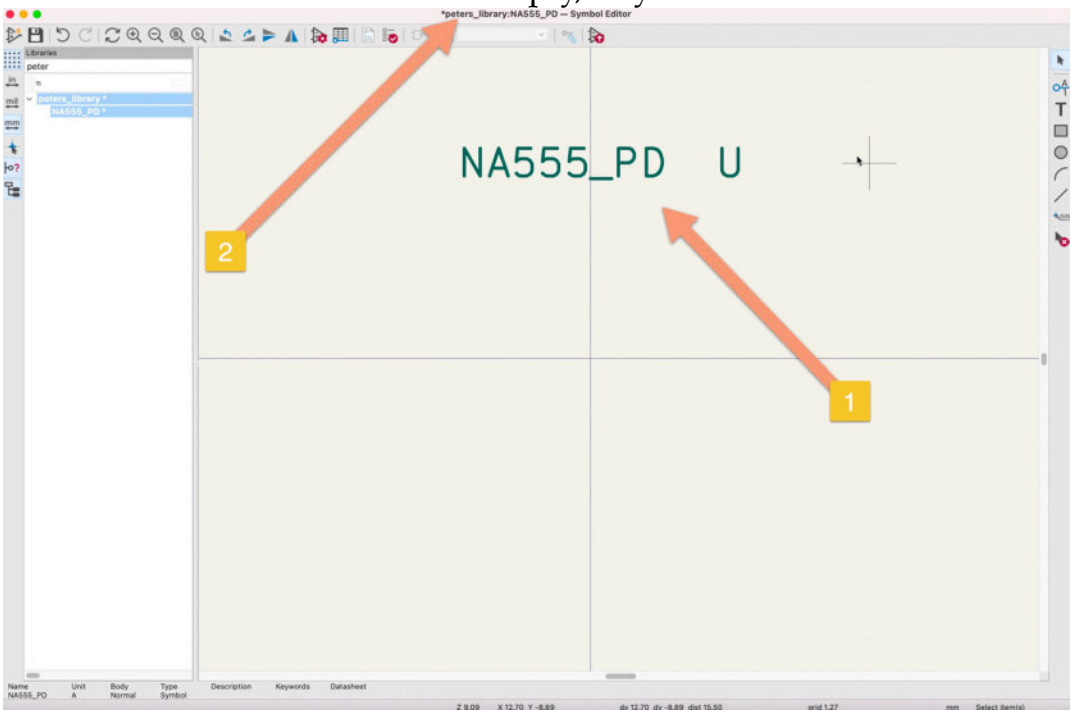


Figure 7.9.9: Getting started to draw a new symbol.

Still, with reference to the figure above, notice that the new symbol is a member of my new library. This information is displayed at the top of the editor window ("2").

Continue to draw a rectangle that represents the outline or body of the symbol. Click on the box button from the right toolbar, and draw the rectangle around the center point of the editor's crosshairs. See the example below:

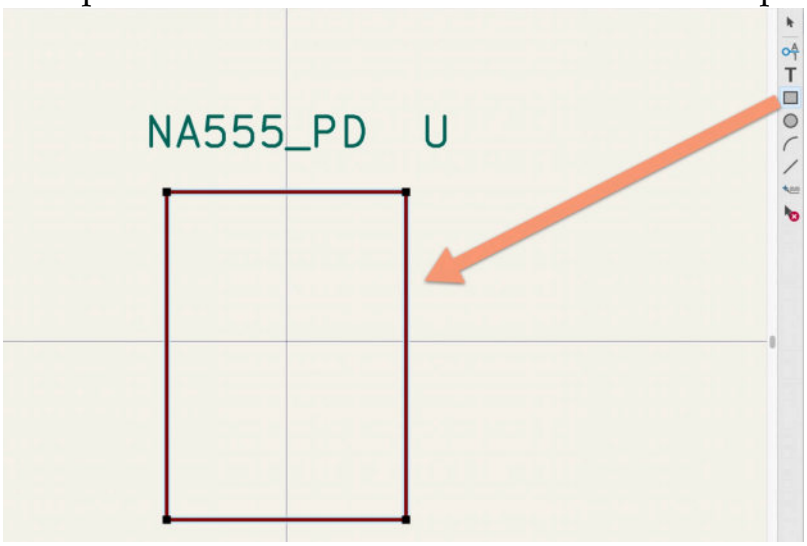


Figure 7.9.10: Creating the outline of the symbol.

Now I can draw the pins. I will arrange the pins according to the layout I see in the datasheet of Figure 7.9.2 in the left diagram the represents the DIP option of the timer chip. The only variation is placing the Vcc and GND pins at the top and bottom of the symbol to keep with convention.

To create a pin, click on the pin button in the right toolbar. In the Pin Properties window that appears, type in the pin name ("Vcc"), the pin number ("8"), and choose an electrical type and style. See my example below:

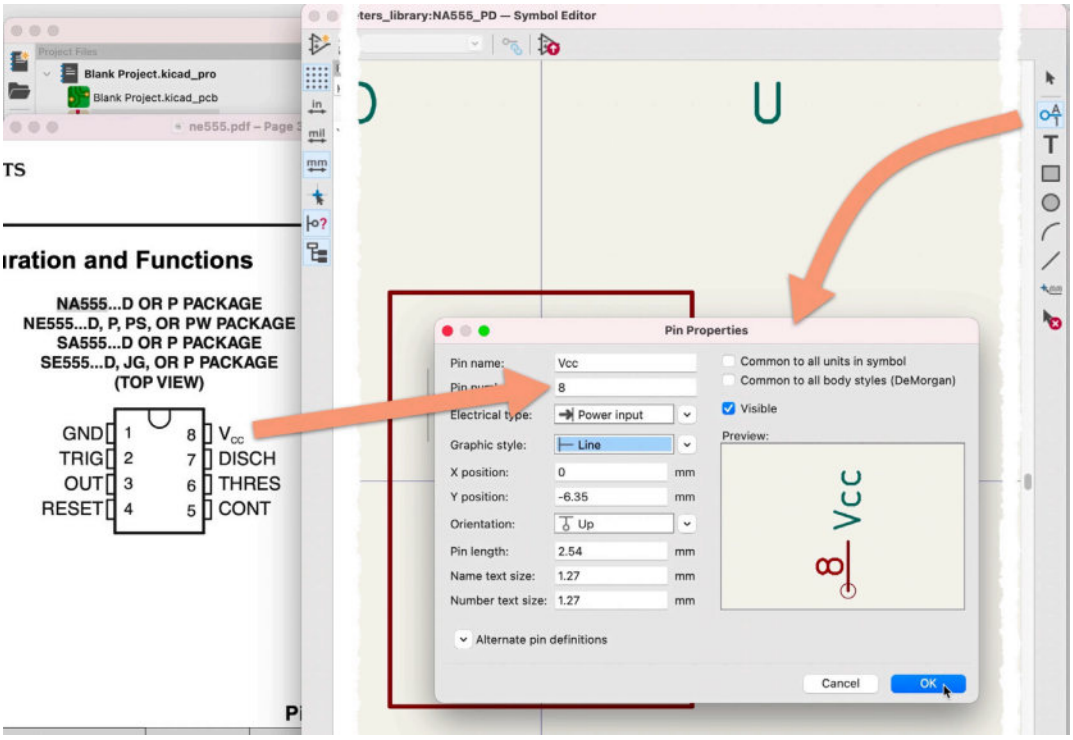


Figure 7.9.11: Creating the first pin.

Using the component datasheet, I learn that the number for the Vcc pin is "8". This Vcc draws power from a power supply, so it is a "power input." I chose a simple line for the graphic style. The electrical type is essential for the correct operation of the ERC. However, the graphic style is purely visual.

Click on OK to close the window, and place the first pin at the top edge of the symbol's outline as in the example below:

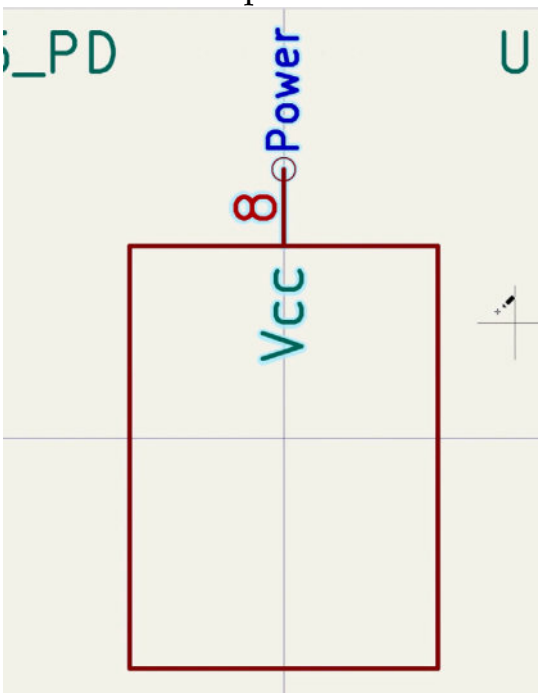


Figure 7.9.12: The first pin is complete.

You may need to resize the outline to contain the pins and their names fully as you add more pins. Continue to add the GND pin at the bottom of the symbol outline. Here is the new version of the symbol:

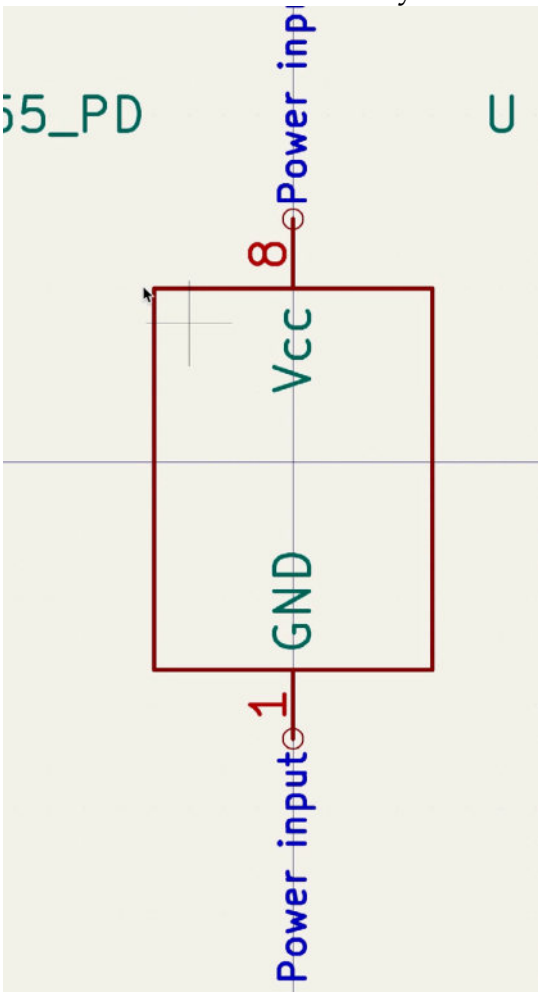


Figure 7.9.13: Added the GND pin.

I have also marked the GND pin as a "power input" and its pin number as "1," following the information in the datasheet. Continue work with the pins on the left side of the symbol:

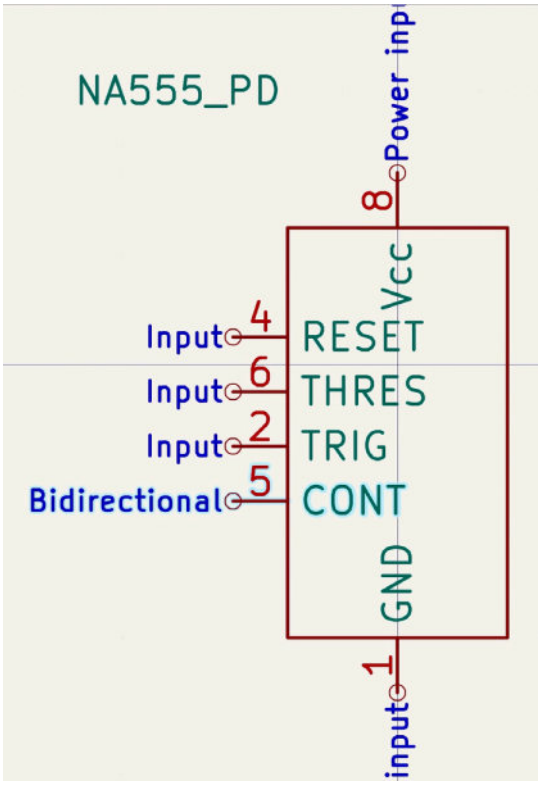


Figure 7.9.14: Added the pins in the left side.

Pins 4, 6, and 2 are inputs, and pin 5 is bidirectional. Notice that their numbers are not in order, as I have not simply copied the layout from the datasheet. This is the difference between creating a symbol and creating a footprint. If I were making a footprint, I would accurately copy the physical characteristics of the component package as described in the datasheet. I would also examine an actual physical component if I had one. But as I am working on a symbol, I place the pins according to their function and type. In the case of this example, I have placed the inputs on the left side, and I am about to place the outputs on the right side.

Continue with the pins on the right side now. Here is the result of this work:

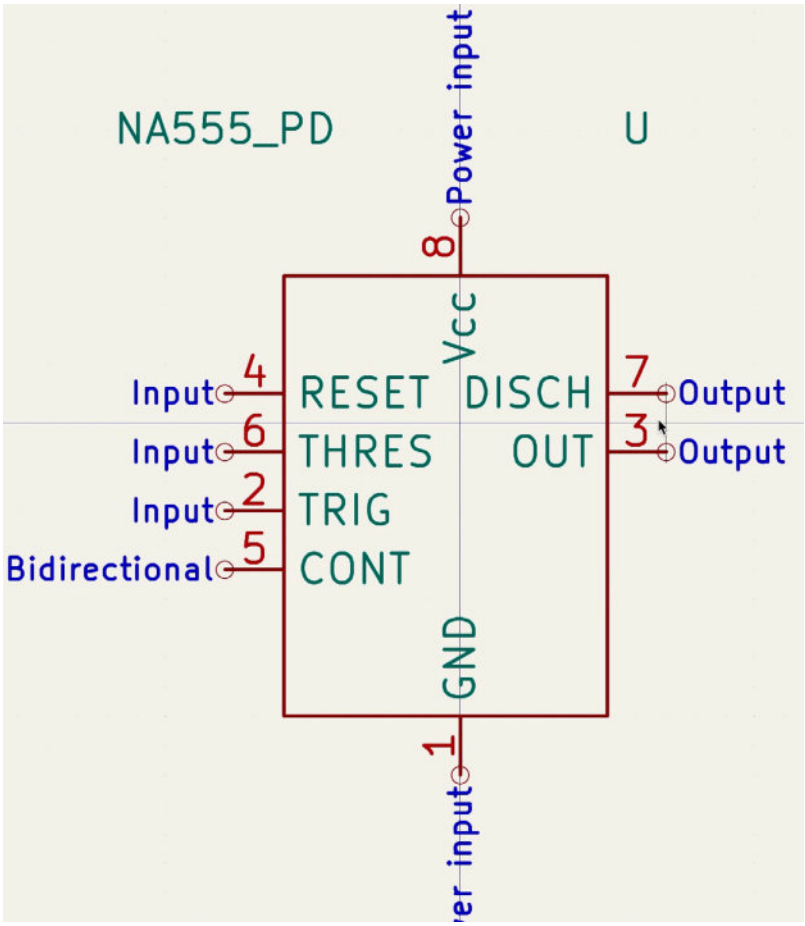


Figure 7.9.14: Added the output pins in the right side.

In the example above, I have added the two output pins on the right side of the symbol.

The symbol is now complete, but there are a few properties that you can set.

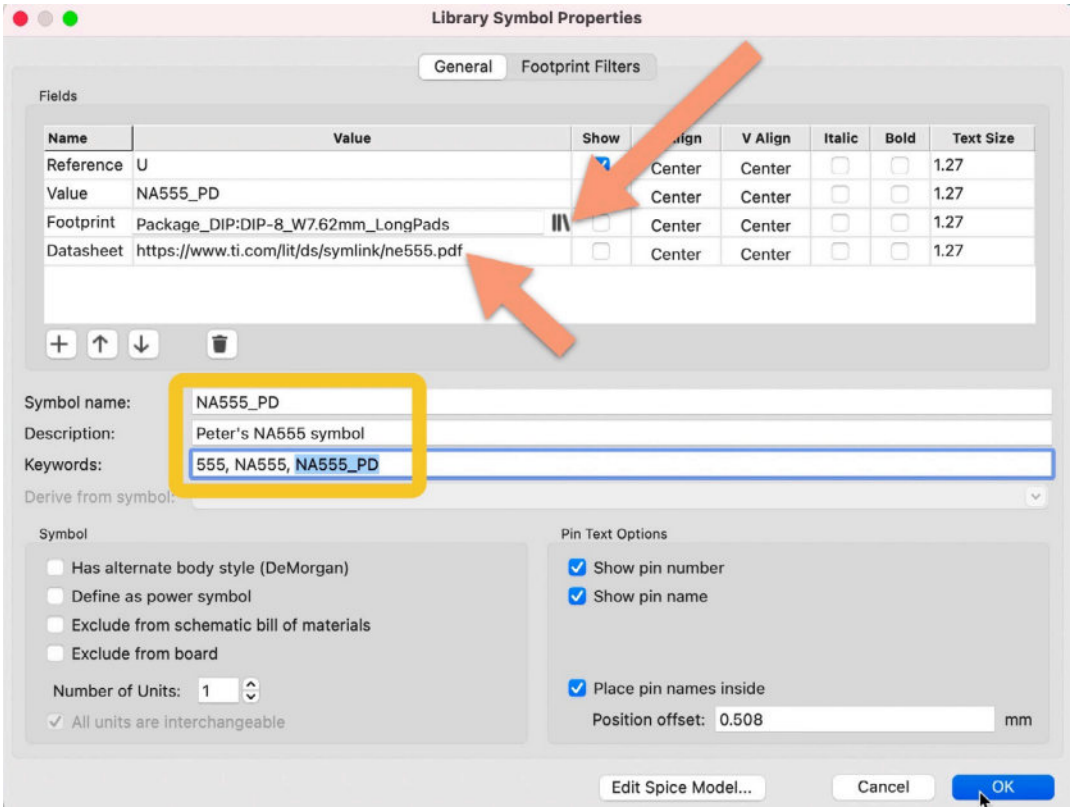


Figure 7.9.15: The new symbol properties window.

Click on the Properties button from the top toolbar (see image above). In the properties window, you can choose a default footprint. To do this, click in the footprint field and then the library button, and use the footprint chooser to find a matching footprint such as the PDIP8 package.

You can also add a URL to the component's datasheet, a description, and keywords.

Click OK to close the properties window.

Save the changes in the symbol editor, and switch to the schematic editor to use the new symbol.

In Eeschema, type "A" to bring up the symbol chooser and search for the symbol you just created. I am searching for "NA555_PD":

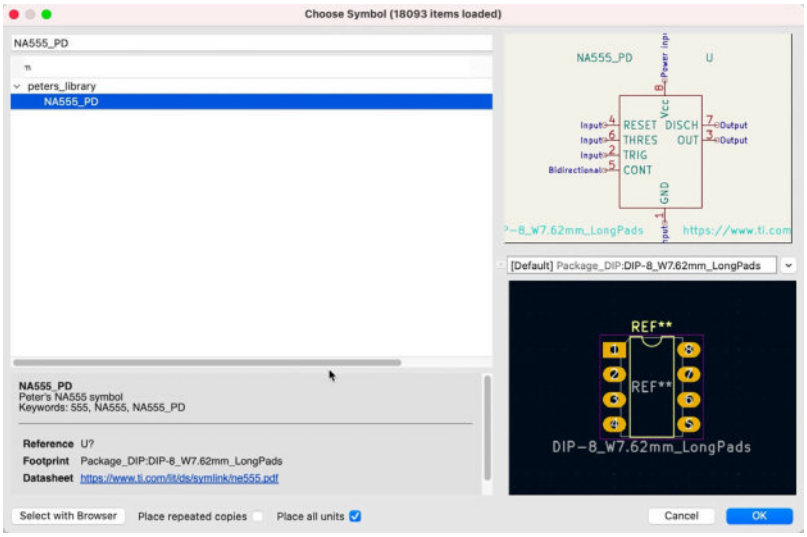


Figure 7.9.16: I made this!

As with any symbol, double-click to select it and close the symbol chooser window. In the schematic editor, you will see the new symbol, ready to wire to other symbols:

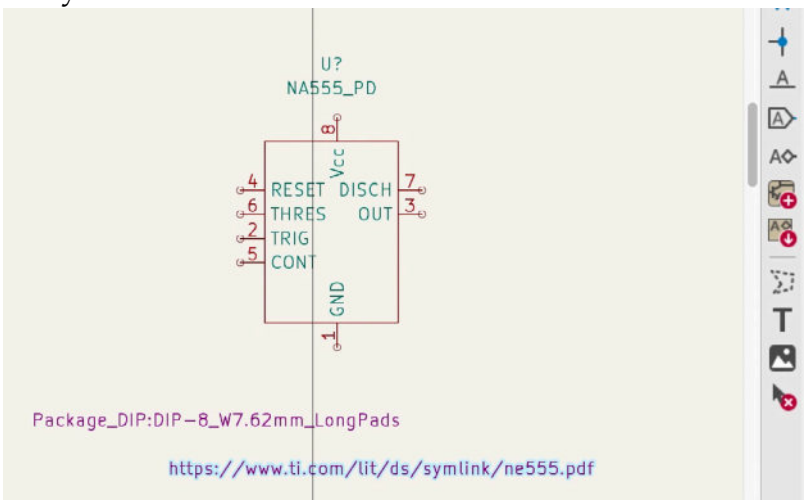


Figure 7.9.17: The new symbol in the schematic editor.

The figure above shows the new symbol with its visible properties, such as the URL to the datasheet and the default footprint.

Knowing how to create a custom symbol is important in the rare case where you can't find what you need. Knowing how to use the symbol editor is also helpful to modify an existing symbol; this is a more typical case scenario.

10. How to associate a symbol with a footprint

In this chapter, you will learn how to associate a symbol with a compatible footprint. In KiCad, symbols and footprints are independent entities. You can match them in any way you want, including in incorrect ways. Consider the example in Figure 7.10.1 below:

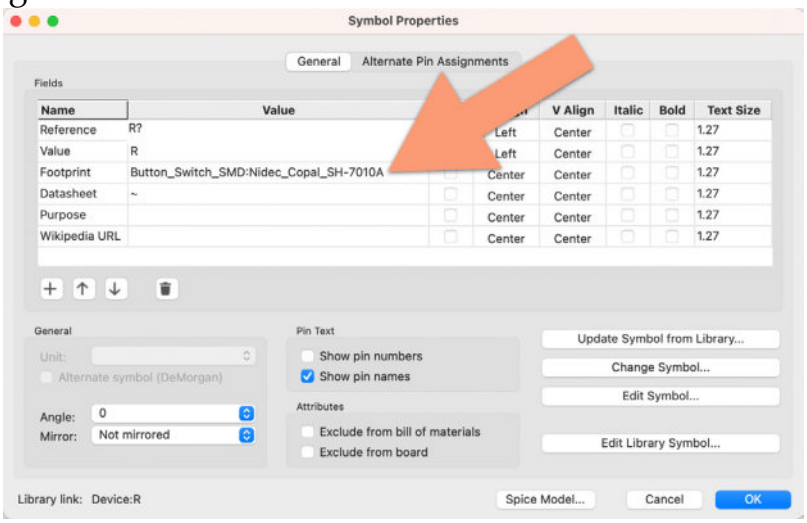


Figure 7.10.1: The properties of a resistor symbol.

The symbol properties window in the figure above represents a resistor. Notice the content of the Footprint field; I have associated this resistor with an SMD button switch footprint. A resistor symbol is not electrically compatible with a switch. Nevertheless, KiCad accepted the association. Even the ERC will not complain about the mismatch.

You have total and unlimited freedom to match a symbol with a footprint. It also means that you have to be careful to associate symbols with compatible footprints. You can assess compatibility by evaluating the number and roles of symbol and footprint pins, the layout of the pins, and the shape of the footprints. You should also take into account manufacturer information, ideally coming from datasheets.

In this chapter, I will explain the process of associating a symbol with a footprint using a simple example. Start by adding a resistor to the editor sheet. Annotate it using the symbol annotator tool from the top toolbar. Your schematic will now look like this:

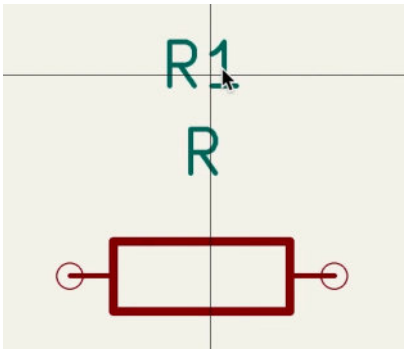


Figure 7.10.2: A resistor.

At this point, you have a symbol that is not associated with a footprint. If you switch across to Pcbnew and try to import the schematic from Eeschema, you will see an error, like the one in Figure 7.10.3 below:

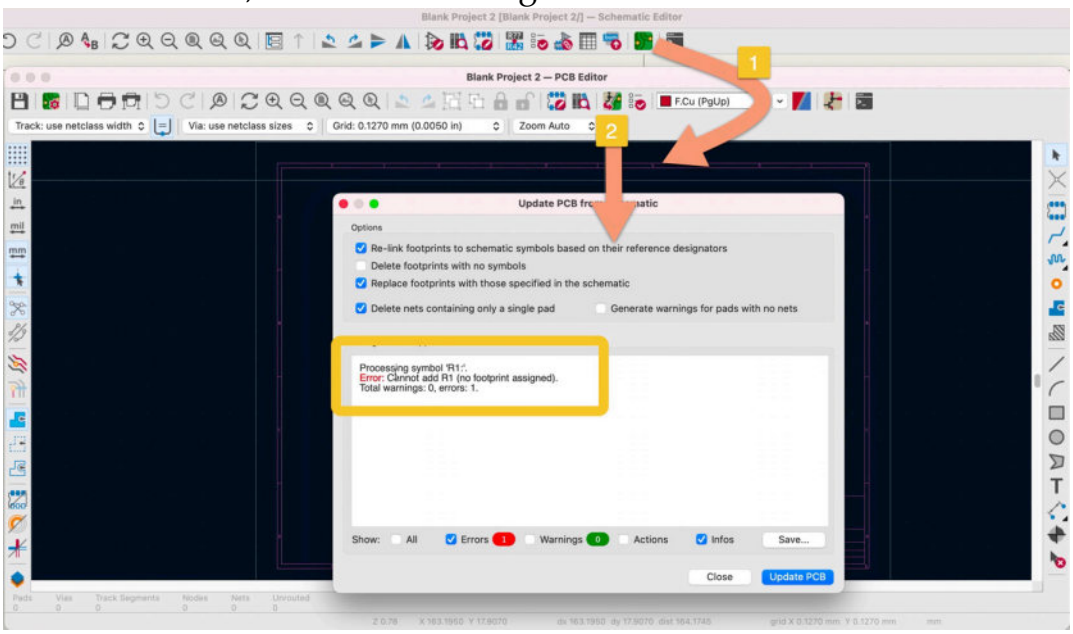


Figure 7.10.3: No footprint assigned to R1.

Return to Eeschema to correct this error.

In Eeschema, there are two ways to associate a symbol with a footprint:

1. Individually, using the symbol's properties window.
2. In bulk, using the footprint assignment tool.

Let's look at each one.

Symbol properties window

To set an association using the symbol's property window, double-click on the symbol to open its properties window ("1", in the figure below).

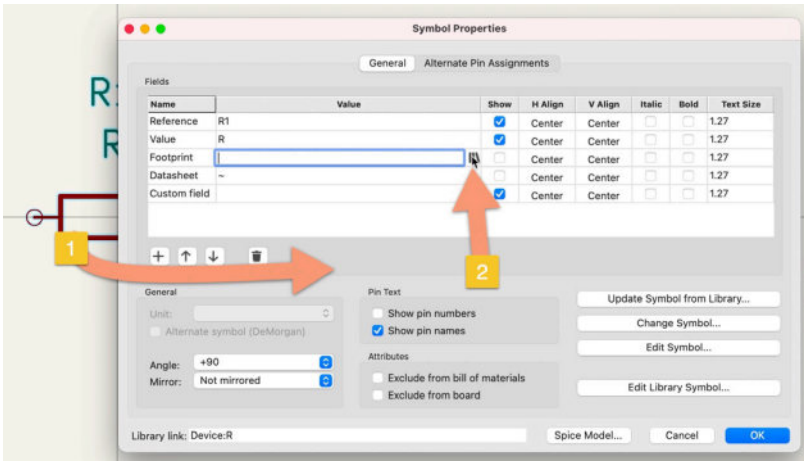


Figure 7.10.4: The symbol properties window.

Click inside the footprint field to reveal its library button, and click on the button ("2", above). This will show the footprint library browser window (see below). Use the filter to help you find the required footprint. I am looking for a resistor footprint, so I have typed "res" in the field ("1", below).

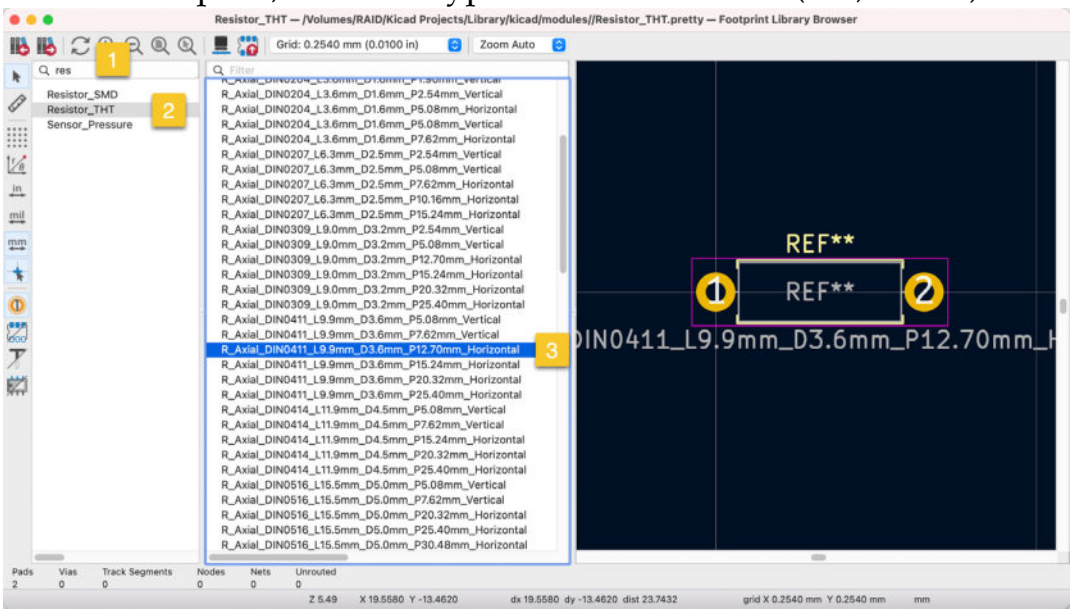


Figure 7.10.5: The footprint library browser window.

Browse through the libraries ("2", above) and the footprints ("3") until you find a compatible footprint. When you find it, double-click on it to associate it with the symbol. In the symbol's properties window, confirm that the correct footprint name appears:

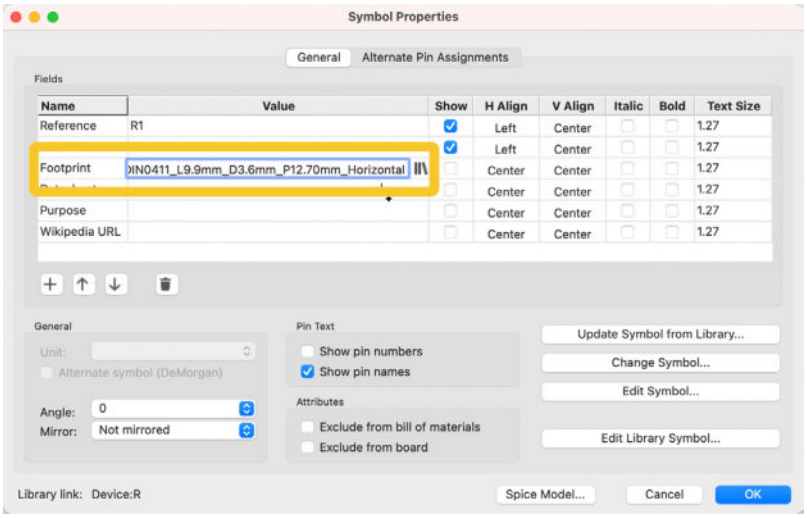


Figure 7.10.6: The associated footprint appears in the symbol properties window.

Your symbol is now associated with a footprint, and you can continue work in Pcbnew. This method of setting an association is sufficient for a small number of symbols. But if you have more than a few unassociated symbols, a better way is to use the associations tool.

The Footprint assignment tool

The footprint assignment tool allows you to set symbol-footprint associations in bulk. Click on the footprint assignment tool in the top toolbar to bring up the tool's window ("1" in Figure 7.10.7 below).

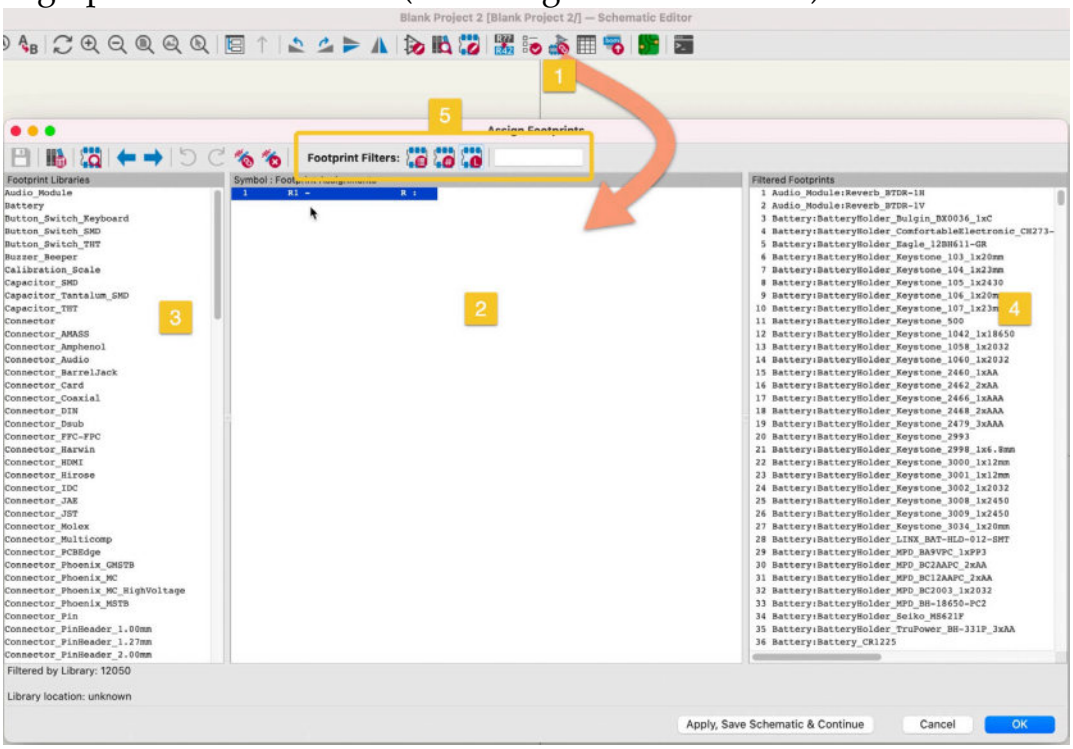


Figure 7.10.7: The footprint assignment tool.

In the window, the symbols are listed in the middle pane (“2”). The left pane (“3”) contains a list of libraries, and the right pane (“4”) includes a list of footprints.

You can control the libraries and footprints listed in the two side panes using the filters (“5”). You can type text in the text field to match footprint names, descriptions, and keywords. You can also filter by pin count, footprint description, and footprint library. For example, suppose I want to see only footprints from a specific library with several pins equal to the number of pins in my selected symbol. In that case, I will click and enable the second (“#”) and third (“L”) buttons and then click on my preferred library in the left pane).

In this example, I am looking for a THT resistor footprint. In the assignment tool (see Figure 7.10.8 below), I will select the “L” filter and keep the text field blank so that the left pane contains all of the available libraries (“1”). I scroll down to find the Resistor_THT library and click to select it (“2”).

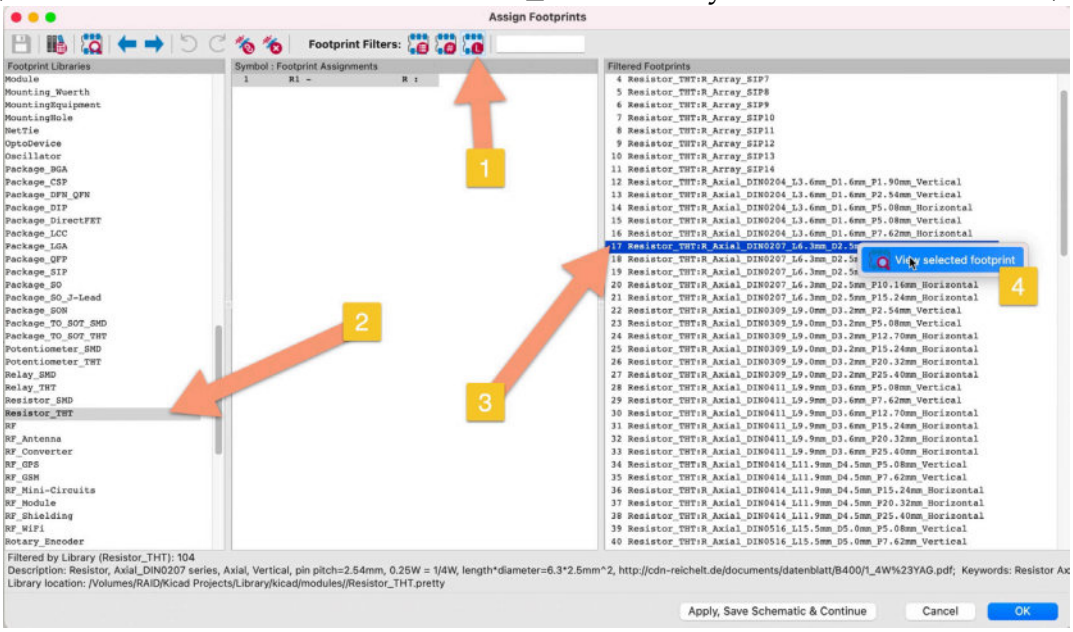


Figure 7.10.8: Found a footprint for the resistor symbol.

In the right pane, a list of all footprints that belong to the selected library will appear. I scroll through this list and choose the footprint that I want to use. To check that my selection is correct, I can right-click on a row and click on “View selected footprint.” This will bring up the footprint viewer where I can examine the selected footprint for compatibility. I can check its dimensions, pad shape, and type, pitch, etc.

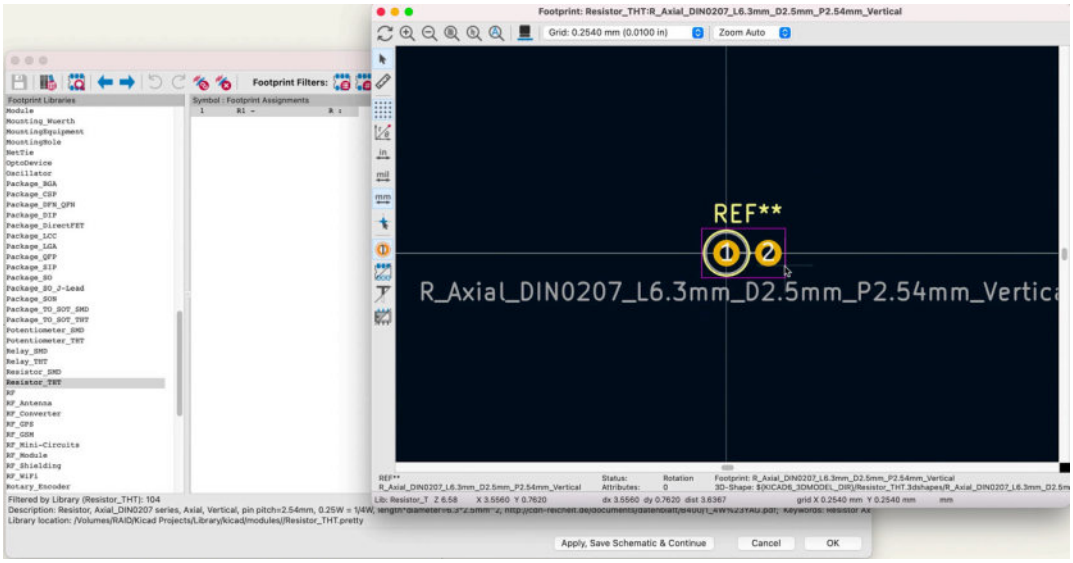


Figure 7.10.9: Examining the footprint in the footprint viewer.

Once I am satisfied that this is the correct matching footprint, I will close the viewer and double-click on the footprint row to assign it to the symbol. In the Assign Footprints tool window, you can confirm that the association between the symbol and its new footprint is completed in the middle pane (see Figure 7.10.10 below).

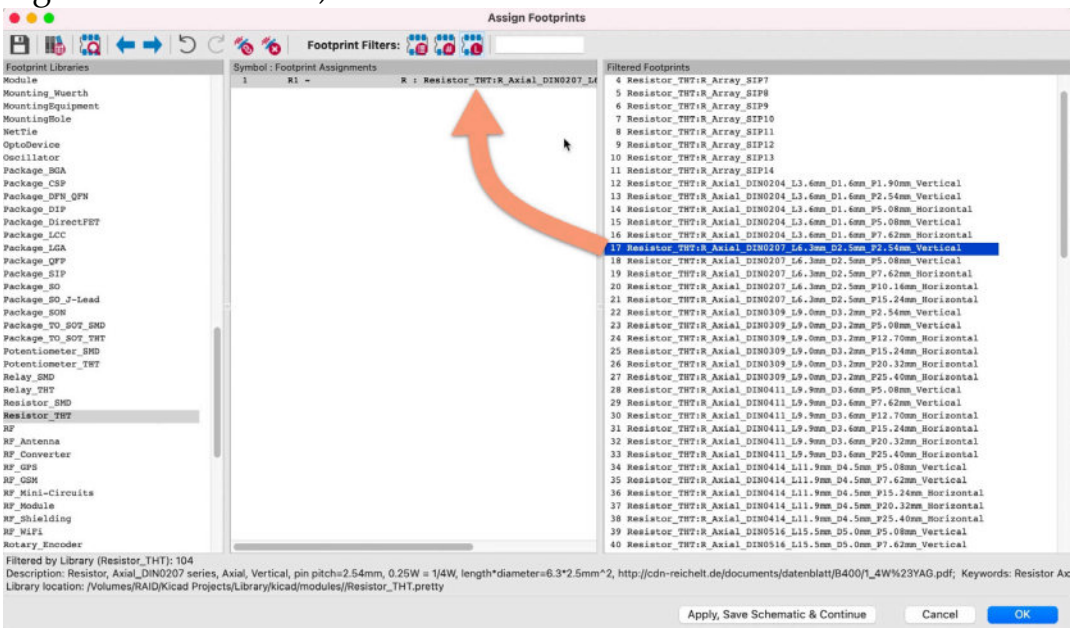


Figure 7.10.10: Association is complete.

In this example, I only included a single symbol. However, you can use the same process for any number of symbols. Below is an example where I have set associations for a much larger number of symbols (this is from one of the projects in this book):

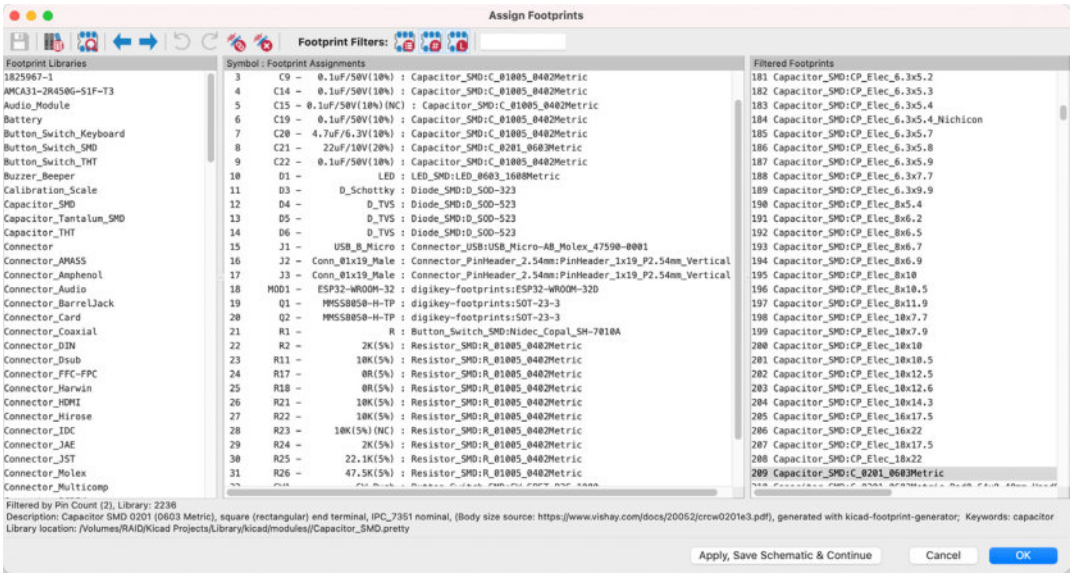


Figure 7.10.11: An example with a large number of associations.

11. Net labels

In this chapter, you will learn how to use net labels in your schematics. With net labels, you can achieve two outcomes:

1. Avoid using wire and bus lines, resulting in a less cluttered and better-looking schematic.
2. Create nets with custom names that you can recognize in the schematic and layout editors.

To demonstrate the use of net labels, I will use a simple schematic consisting of a few connectors (“Conn_01x01_Female”) and wires. You can see this schematic below:

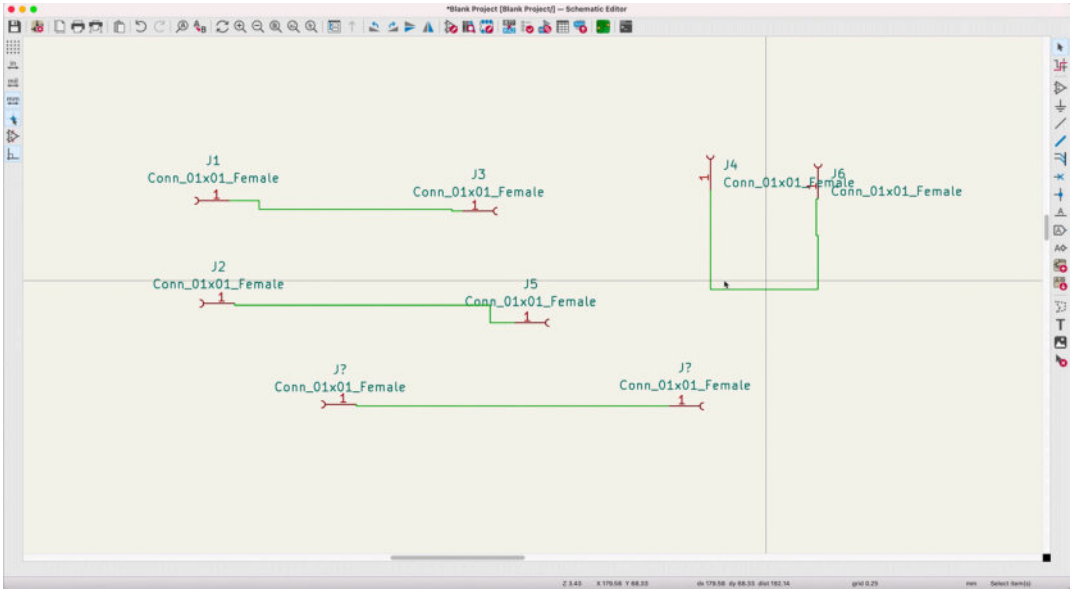


Figure 7.11.10: A simple example to demonstrate the use of net labels.

I have already set the symbol-footprint associations. If you want to follow along, associate each symbol (“Conn_01x01_Female”) with a single-pad footprint such as the ” Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vertical”. Once you have completed the associations, continue with the net labels.

You can attach a net label to a wire or directly onto a pin. Either way, the result is the creation of a named net. Before I create and attach the first net label, let’s look at the current nets and how they appear in Pcbnew.

Open Pcbnew and import the schematic from Eeschema. You can see the result below (I have separated the footprints to make it easier to distinguish them):

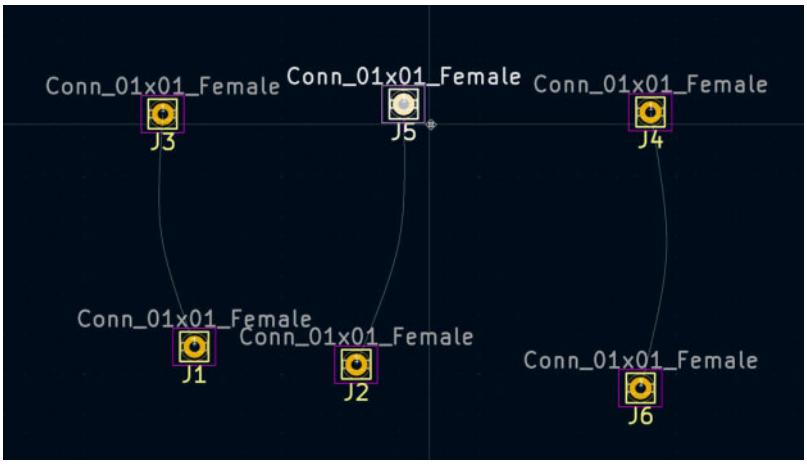


Figure 7.11.1: The layout editor showing the connections between pins using ratnests.

Before drawing the tracks, the layout editor depicts the nets with thin ratnest lines. If you zoom into one of the pads, you will see the automatically assigned net name.

Let's create a net label and attach it to one of the wires in the schematic editor. Click on the Net Label button in the right toolbar, and type a name in the Label field (see below). The consequence of this is to have a newly named net.

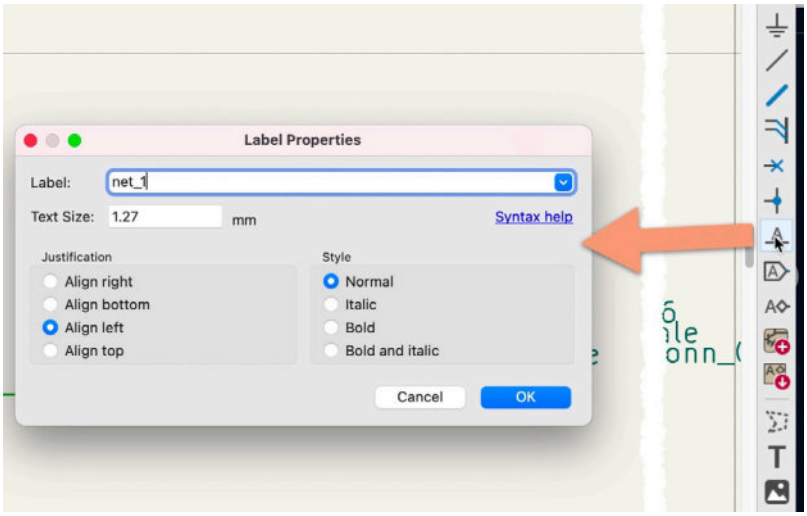


Figure 7.11.2: Create a new net label.

In the example above, I am creating a new net label with the name "net_1". Click OK. Attach the new label to the wire that connects J1 to J3:

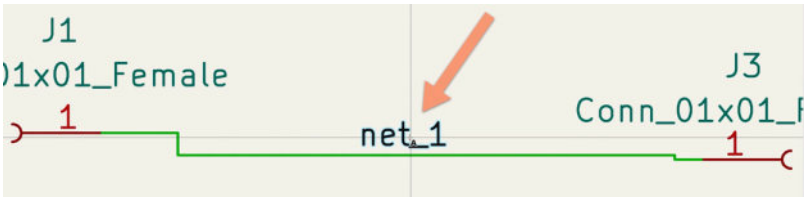


Figure 7.11.3: The net label is attached to the wire.

Beware that the label will not be attached to a wire or pin until the small box that appears in its lower-left corner disappears. In the figure above, there is no box. Therefore the attachment is correct. Next, duplicate the "net_1" label, and attach it to the second wire (the one that connects J2 to J5). The schematic now looks like this:

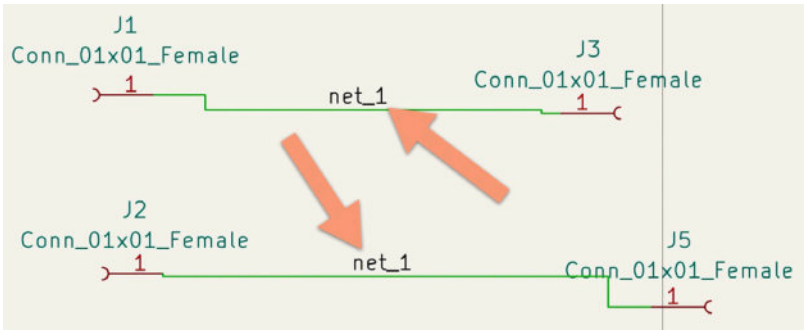


Figure 7.11.4: Two wires belong to the same net because they have the same net label attached.

Because both wires have the same net label attached, they belong to the same net label. As a result, these wires are electrically connected, even though they look separate in the schematic editor. "Seeing is believing," so to confirm that these two wires are indeed electrically connected, switch over to Pcbnew, and import the changes from Eeschema (click on the "Update PCB" button). The result is below:

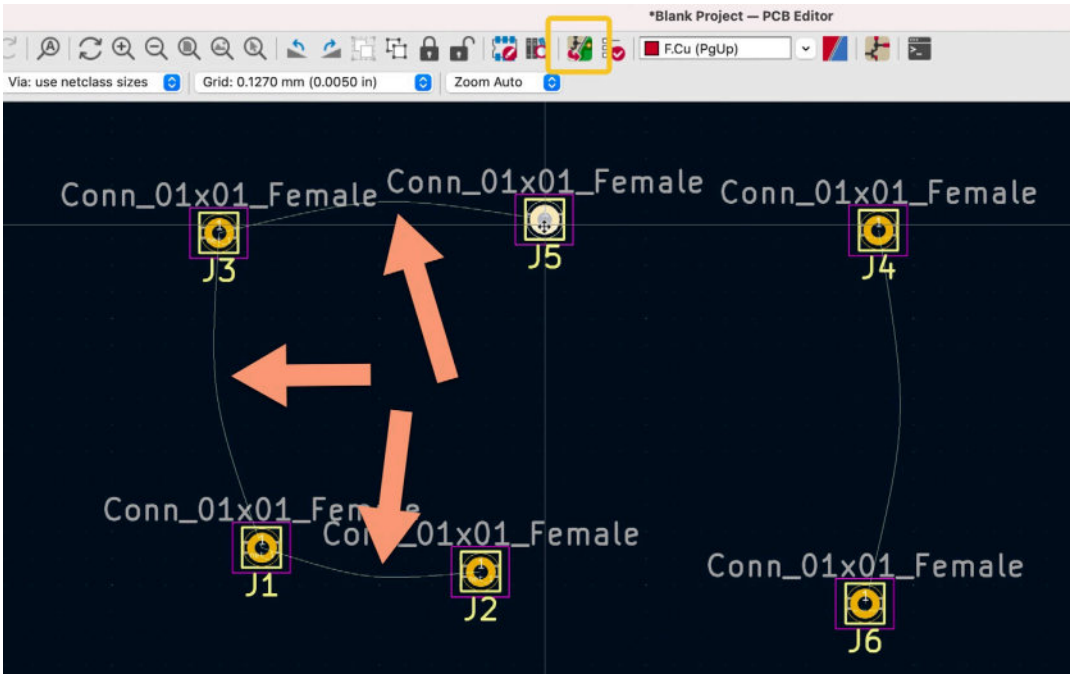


Figure 7.11.5: J5, J3, J1, J2 are connected.

As you can see, the pads of connectors J5, J3, J1, and J2 are electrically connected because they are part of the same net. You can zoom into one of those pads to see the name of the net where it belongs:

Figure 7.11.5: Pad 1 of J3 belongs to net_1.

As you can see, pad 1 of J3 belongs to net_1 (as do pads of J5, J1, and J2).

Let's create a second named net. Create a new net label with the name "net_2", duplicate it, and attach one each to the wires that connect the pins of connectors J4, J6, J7 and J8. Here is the result:

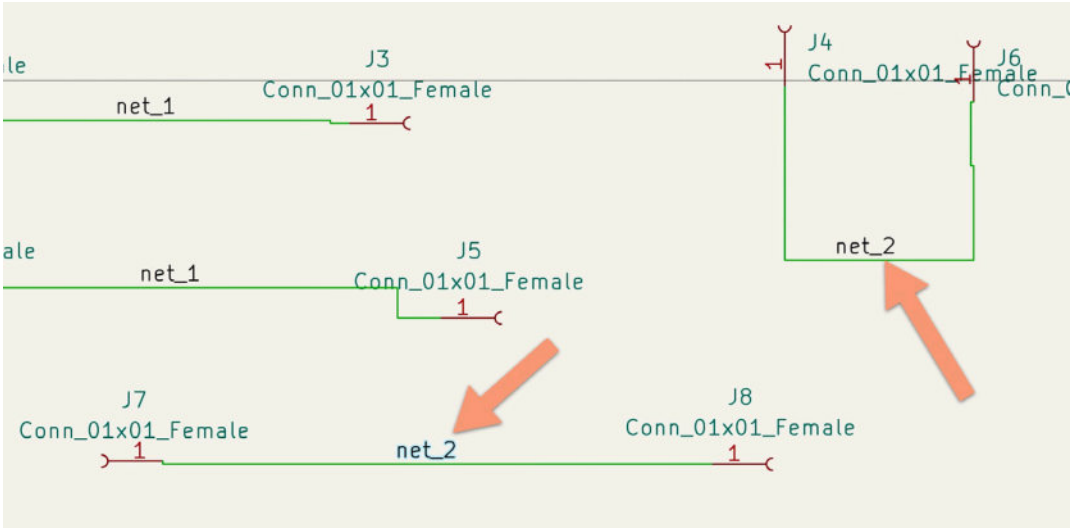


Figure 7.11.6: The “net_2” named net.

Return to Pcbnew and import the changes from Eeschema. The layout editor now looks like this:

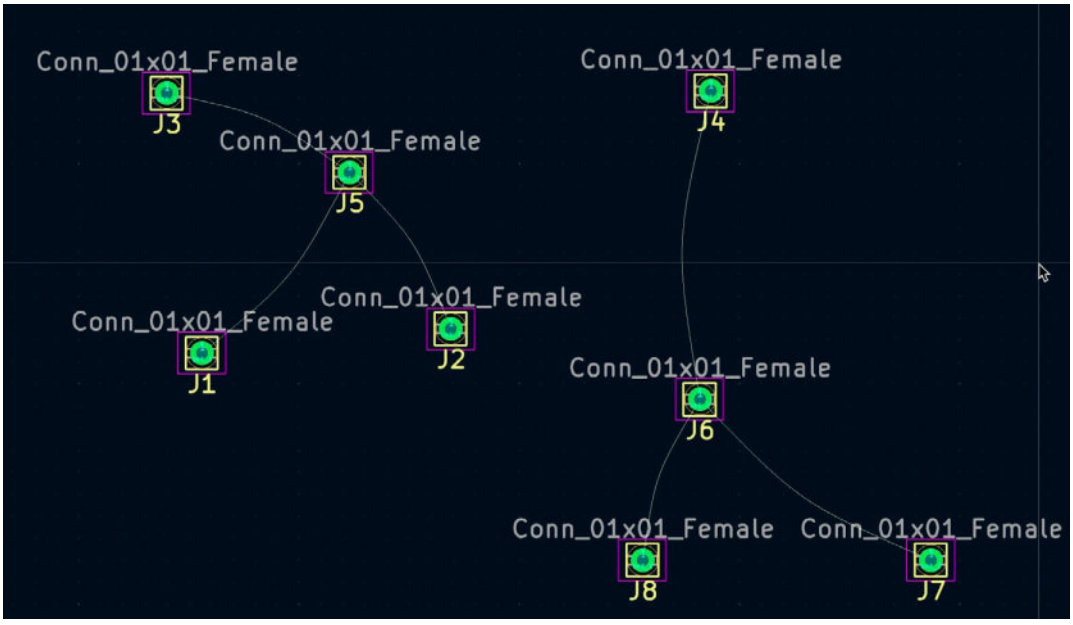


Figure 7.11.7:Pads for J4, J6, J7 and J8 belong to the net_2 net.

As you saw above, you can use net labels to assign a name to wires. However, you can use net labels without wires. You can attach a net label directly to a symbol pin. For example, see the schematic (segment) in Figure 7.11.8 below (this comes from one of the projects in this course):

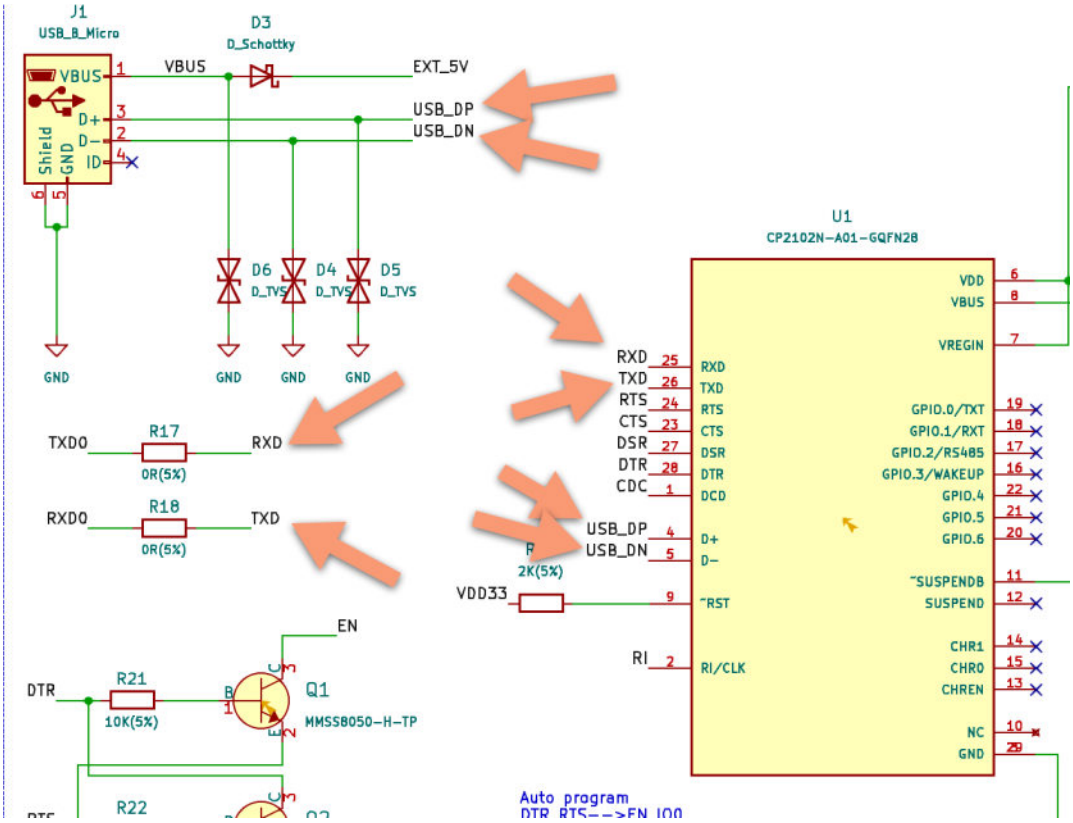


Figure 7.11.8:Use of net labels without wires.

In this example, I have used net labels to electrically connect several pins of the U1 symbol with pins elsewhere in the schematic. Notice the “RXD” label attached to pin 2 of R17 and the same label attached to one of the pins of U1. These two pins are now connected, albeit without a visible wire line. In the layout editor, you will still see the net ratnest. In most cases, you will want to use net labels attached directly onto pins that you wish to connect rather than using graphical wires electrically.

Apart from the advantages of net labels that I described above, net labels make it easier to manage the physical characteristics of traces based on their net membership and the use of net classes. You can learn about this capability in the next chapter.

12. Net classes

In this chapter, I will continue where we left off in the previous chapter to explain the use of net classes. At the end of the last chapter, our schematic diagram contains several connectors, and their pins belong to two nets: "net_1" and "net_2". In this chapter, I will show you how you can use nets (and in particular named nets) to control some of the parameters of the copper wires that connect the connector pads in the layout editor.

Schematic editor

In the schematic editor, click on File, then "Schematic Setup." In the Schematic Setup window, click on "Net Classes" under Project. The Net Classes tab looks like this:

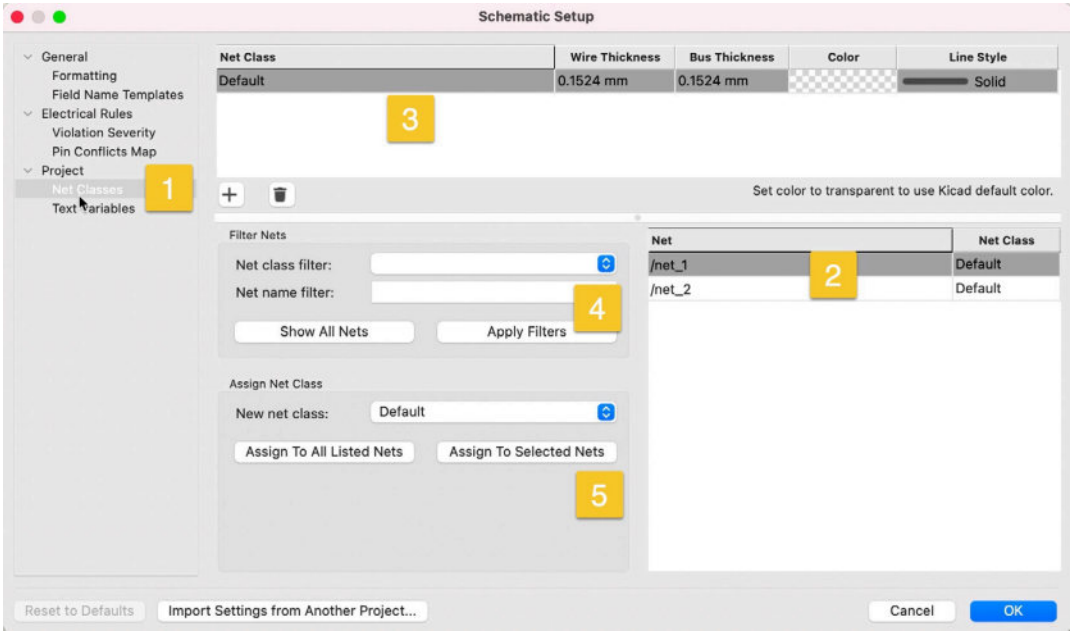


Figure 7.12.1:Net Classes in the Schematic Setup window.

In the figure above, under Net Classes ("1"), you will see the Nets and Net Class membership pane ("2") and the Net Class list pane ("3").

In the Net Class list pane, you can configure the thickness, color, and style of a wire or bus () based on the net class membership of the wire or bus. You can also create new net classes or edit and delete existing classes.

In the Nets and Net Class membership pane, you can see the net class each name belongs to.

In busy schematics with many nets and net classes, you can use the Filter Nets group of widgets ("4") to narrow down the nets listed in the membership pane. Finally, in the Assign Net Class pane ("5"), you can set a net class for the nets you have selected in the membership pane.

Pcbnew has a similar configuration tab for Net Classes that you can use to set the physical characteristics of tracks that belong to a particular net class. You will learn more about this later in this chapter.

By default, all nets belong to the Default net class. Go ahead and create two new classes: "net_1_class" and "net_2_class". Set the wire thickness, color, and line style as per my example below:

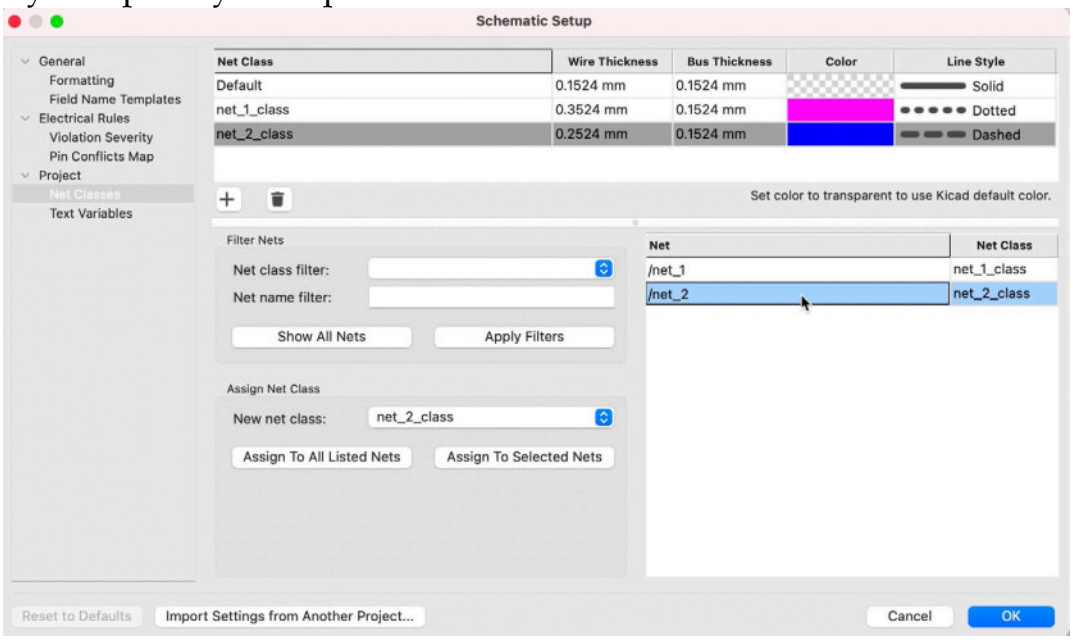


Figure 7.12.2: Added new net classes.

To assign a net to a net class:

1. Click on the net row on the membership pane to select it (you can also choose multiple nets).
2. Select the target net class from the drop-down menu in the Assign Net Class pane and click "Assign To Selected Nets."
3. Click on the "+" button of the Net Class list pane to add a new net class and the field in each row to make changes.."

Click OK to exit the Schematic Setup window. The schematic now looks like this:

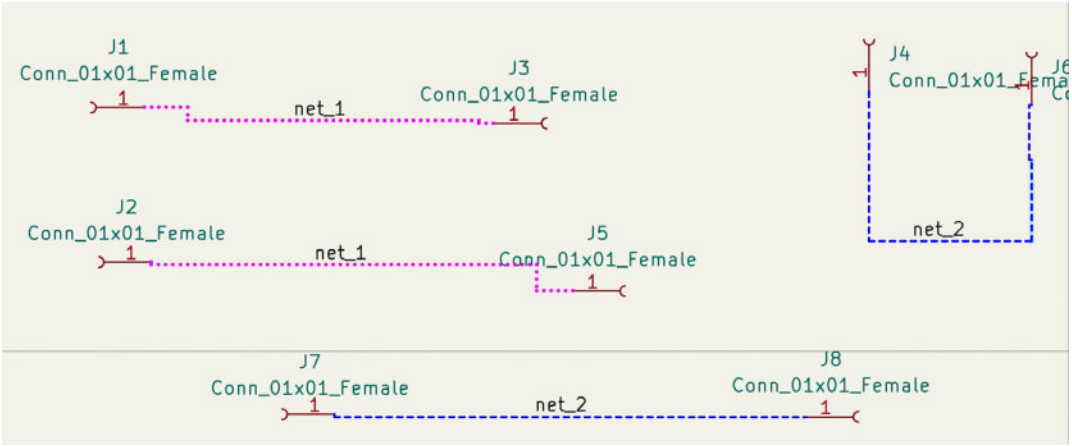


Figure 7.12.3: Net lines drawn as per their net class visual characteristics.

Layout editor

Let's continue with the Layout Editor. Open Pcbnew, open the Board Setup window and select the Net Classes tab (under Design Rules). It looks like this:

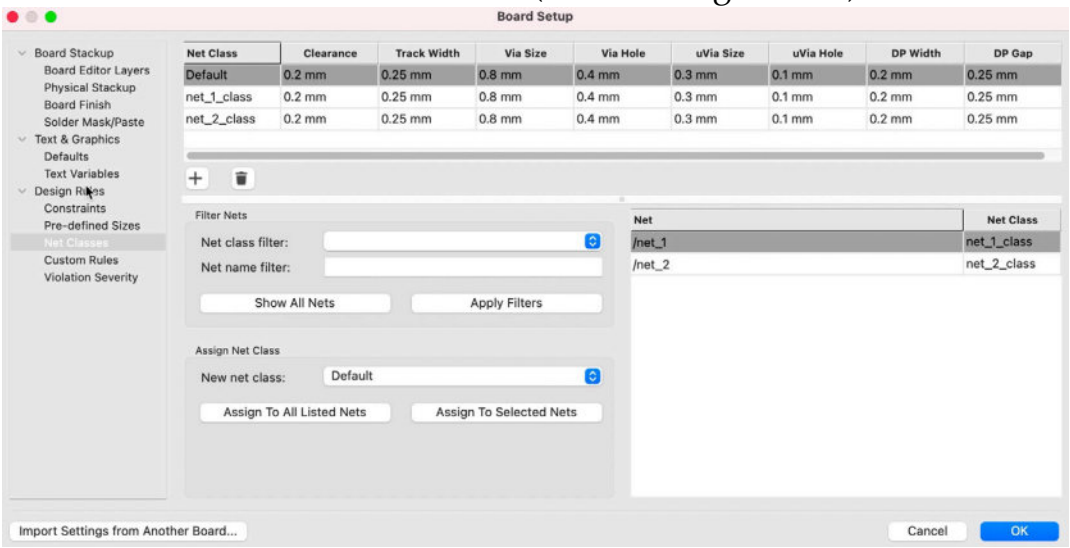


Figure 7.12.4: Net classes in Pcbnew.

As you can see, the layout editor has inherited the net classes and memberships I set in the schematic editor. You can still create new net classes or edit and delete existing net classes. You can also change memberships.

In the layout editor, you can specify the track characteristics of tracks that belong to specific net classes: clearance, track width, via size and hole diameter, etc. In the example below, I have changed the values in the track width fields for the new custom net classes. Copy these values to your project and click OK.

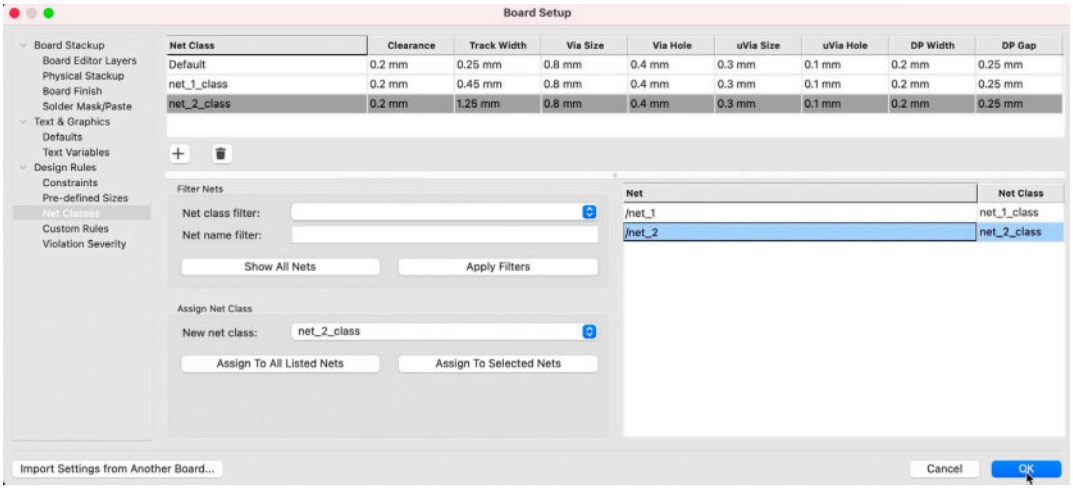


Figure 7.12.5: Net classes with edited track and via values.

In Pcbnew, select the wire tool from the right toolbar, and start drawing some wires in the copper layer. In the figure below, you can see two tracks. One belongs to net_1_class, and the other to net_2_class:

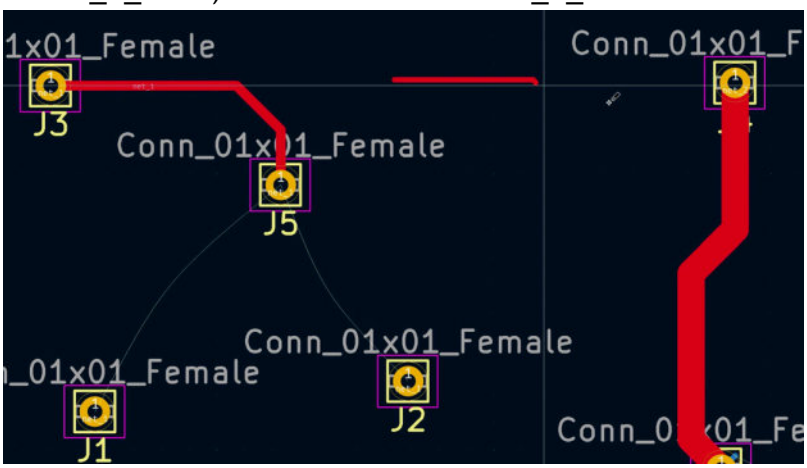


Figure 7.12.6: Track characteristics depend on their net class configuration.

In the middle of the figure above, I have drawn a wire that does not connect to any pads. This wire automatically belongs to the "Default" net class. You can see the difference in its width compared to that of the other two tracks.

For "stray" tracks like this, it is possible to make them members of an existing net class and therefore electrically connect them to that net. To do that, right-click on the track and click on Properties:

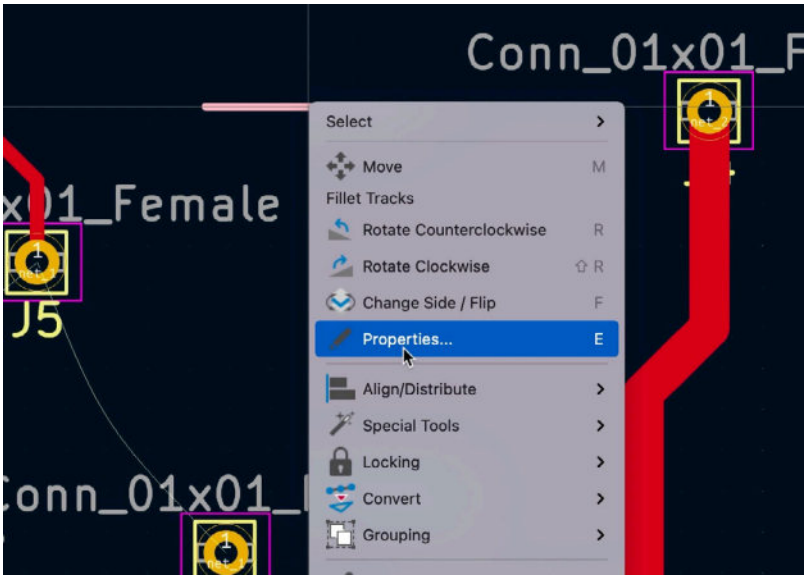


Figure 7.12.7: Show the track properties window.

In the properties window, select the required net, and check the "Use net class widths" checkbox:

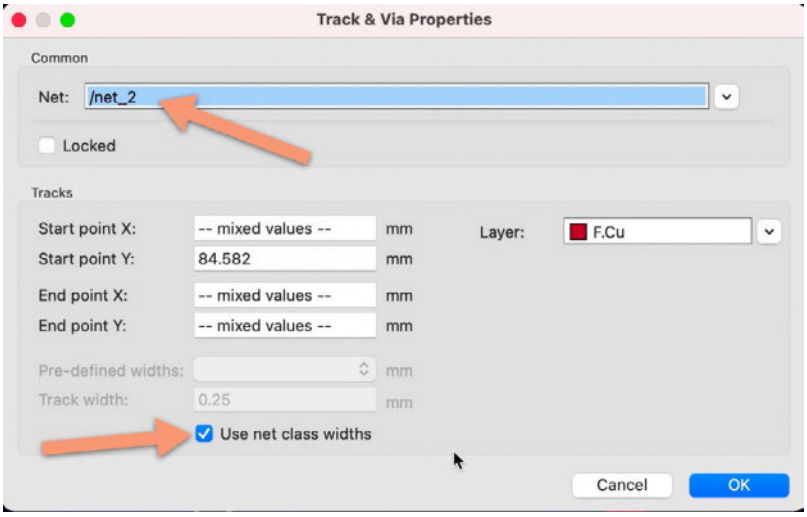


Figure 7.12.8: Select a net.

Click OK to commit the changes. Notice that the layout editor now shows a ratnest line that connects the original "stray" track segment with the existing net_2 track:

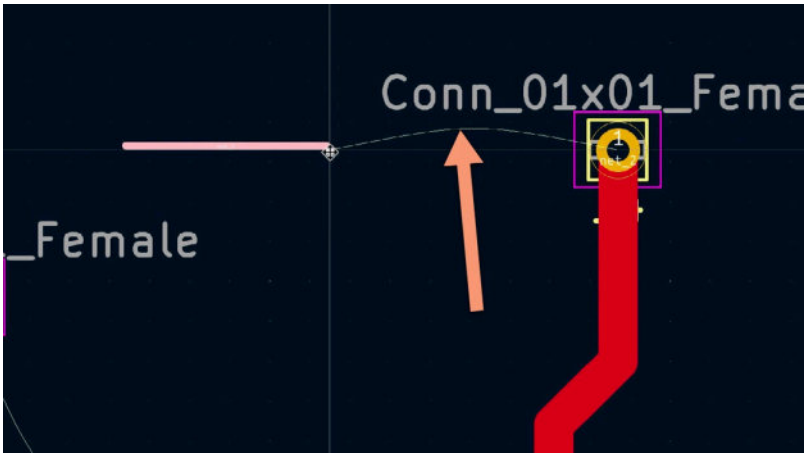


Figure 7.12.8: The original “stray” track segment is now part of net_2.

You can use the Track tool to draw a new track segment that replaces the ratnest line. The new track will inherit the track characteristics of the net class to which net_2 belongs:

Figure 7.12.9: Completed drawing the new track.

Net labels, combined with net classes, provide a powerful way to control your design's visual and electrical characteristics. Try to take the time to understand them fully; they will save you a lot of time in the long run.

13. Hierarchical sheets

In this chapter, you will learn how to split your project's schematic across more than one sheet using hierarchical sheets. Hierarchical Sheets is a feature that is particularly useful in complex schematics.

When you create a new project, Kicad will generate a single schematic sheet. To create a second sheet, use the Hierarchical sheet tool from the right toolbar ("1" in Figure 7.13.1 below):

Figure 7.13.1: Creating a hierarchical sheet.

With the hierarchical sheets tool selected, draw a rectangle, as per the example above. The size of the rectangle does not matter. If you expect that the new sheet will contain many inputs and outputs, you can draw a larger rectangle. You can always resize this rectangle later, as needed.

Click to start drawing the rectangle, and click again to finish. After the second click, the Sheet Properties window will appear, where you can set a sheet name and filename. Here is my example:

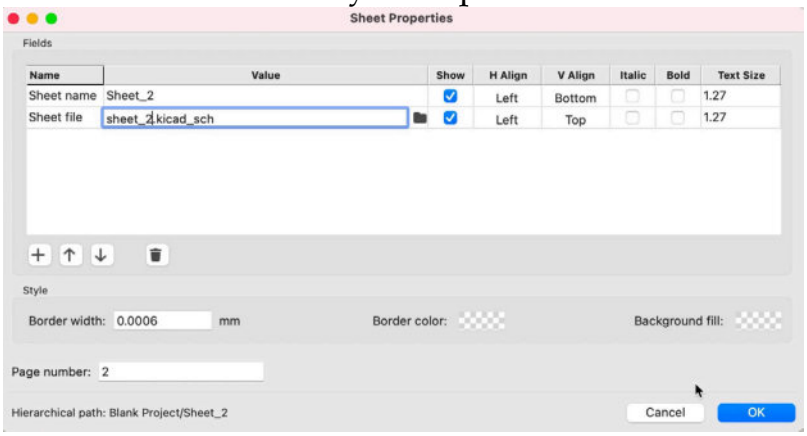


Figure 7.13.2: Hierarchical sheet properties.

Click OK to dismiss the window and save the schematic. Use a file browser to inspect the project folder on your computer's drive. Notice that there is a new schematic file with the filename you specified for the new hierarchical sheet:

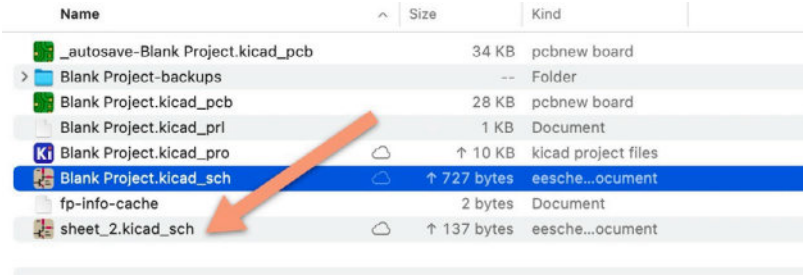


Figure 7.13.3: Each sheet has its own ".kicad_sch" file.

With the new sheet created, you can add content. In sheet 1 (root), double click on the box that represents Sheet_2 to bring the new sheet in the editor. You can add symbols, wires, and other elements as usual.

In the example below, I have copied a couple of elements from sheet 1. Notice the name of the current sheet that appears in the window's top.

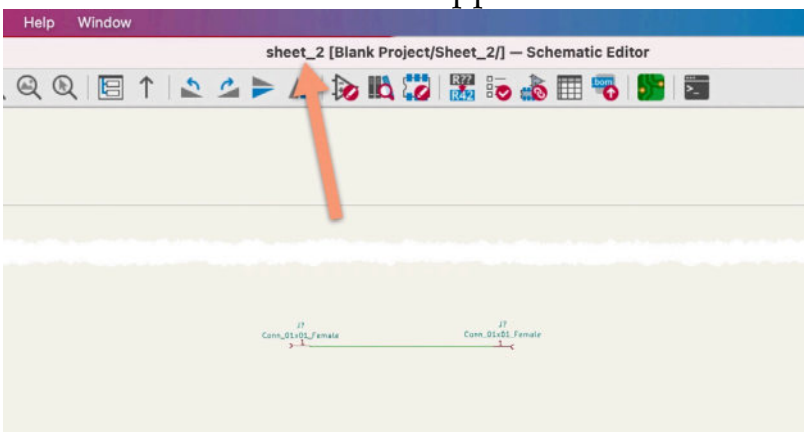


Figure 7.13.4: Sheet 2.

You can navigate between sheets using the Sheet Navigator. To reveal this window, click on the Navigator button in the top toolbar:

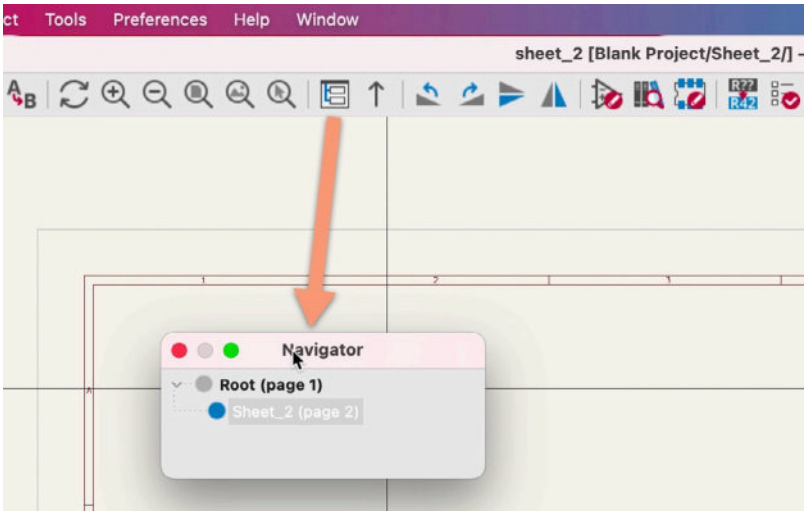


Figure 7.13.5: The Navigator.

If you want to keep the Navigator always open, check the "Keep hierarchy navigator open" in the Schematic Editor's Editing Options in the KiCad Preferences window.

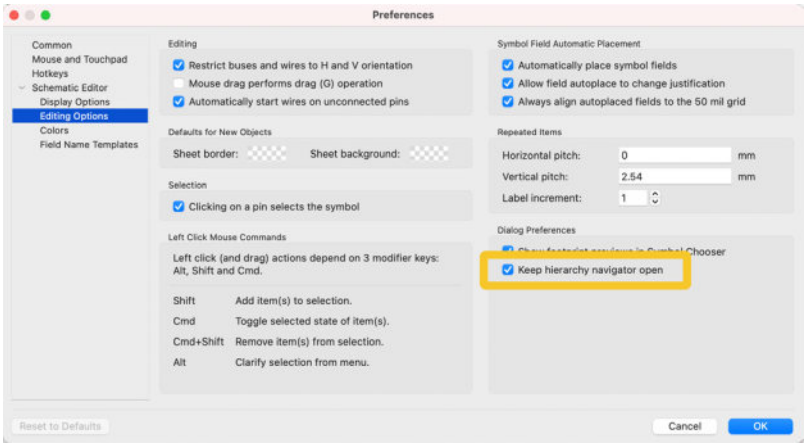


Figure 7.13.6: Keep Navigator always open.

In the Navigator, click on a sheet to load it in the editor. You can also use the up arrow button (right side of the Navigator button in the top toolbar) to go to the parent of the current sheet.

You can create any hierarchy you wish between sheets. The only restriction is that a sheet must have one parent only, except for the root sheet.

In the simple example above, I created a child sheet with no electrical connection with the root sheet. To make electrical connections, you will use either global labels or hierarchical labels. You can learn how to use these labels in the following two chapters.

14. Global labels

In the previous chapter, you learned how to create hierarchical sheets to distribute your schematic diagram across multiple sheets. In the schematic that I used there, I did not make any electrical connections between the symbols across the two schematic sheets. Without such connections, the hierarchical sheets feature would not be very useful.

In this chapter, you will learn how to use Global labels. Global labels is one way by which you can create electrical connection among sheets. The second way is using hierarchical labels, which you will learn about in the next chapter.

To create a global label, click on the enclosed "A" button in the right toolbar. To help you remember the difference between the regular net label and the global label buttons, think of the global label button as the one to click if you want to create labels that work across sheets, and a sheet is enclosed in a border. Thus, the global label button has a border.

Using the schematic from the previous chapter, go to sheet two and click on the global label button. This will bring up the Global Label Properties window, where you can type a name for the new label and choose its visual styling options. I named mine "Global_1", as in the figure below:

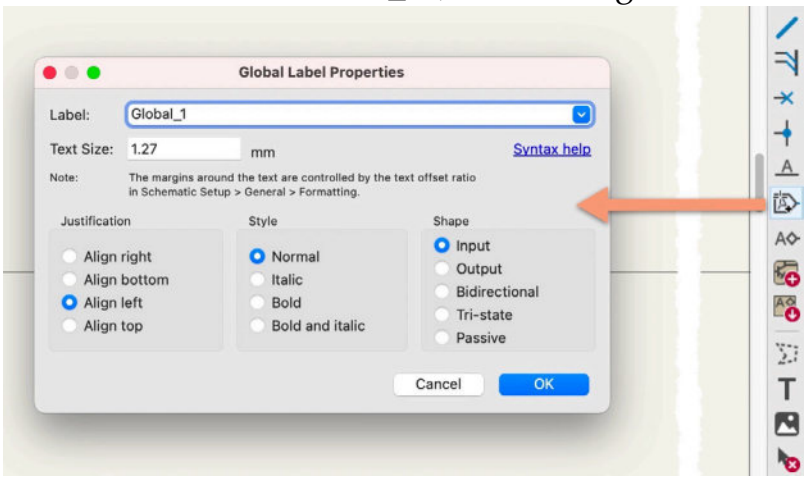


Figure 7.14.1: A new global label.

Click OK to dismiss the window, and click somewhere above the only wire in the schematic sheet to drop the new global label. Then, use a wire to

connect the pointy end of the global label symbol to the wire. The schematic should look like this:

Figure 7.14.2: A global label connected to a wire in Sheet 2.

The next step is to go to the root sheet, create a new global label with the same name, "Global_1", and wire the new label to another part of the circuit. You can also copy the label in sheet 2 (right-click and select "Copy," or use Ctr-C/Cmd-C) and then past it in sheet 1 (right-click and choose "Paste" or Ctr-V/Cmd-V). Use a wire to connect the label to the wire that connects J7 and J8. Sheet 1 should look like this:

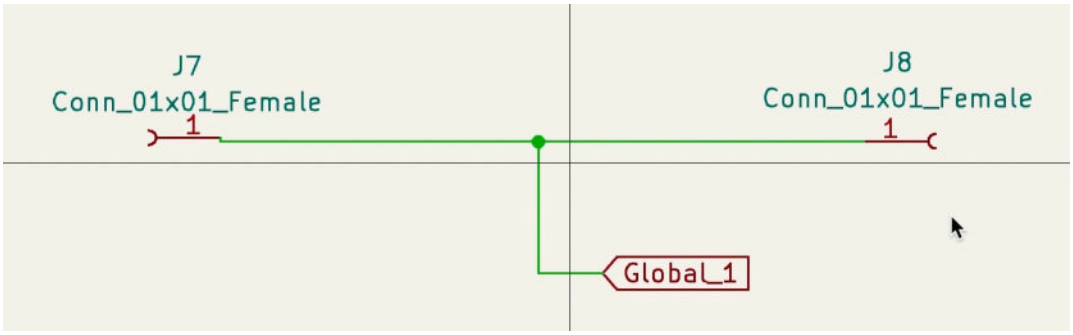


Figure 7.14.3: A global label connected to a wire in Sheet 1.

Save the schematic and continue in Pcbnew. Update the layout from the schematic editor (click on the "Update PCB" button in the top toolbar.

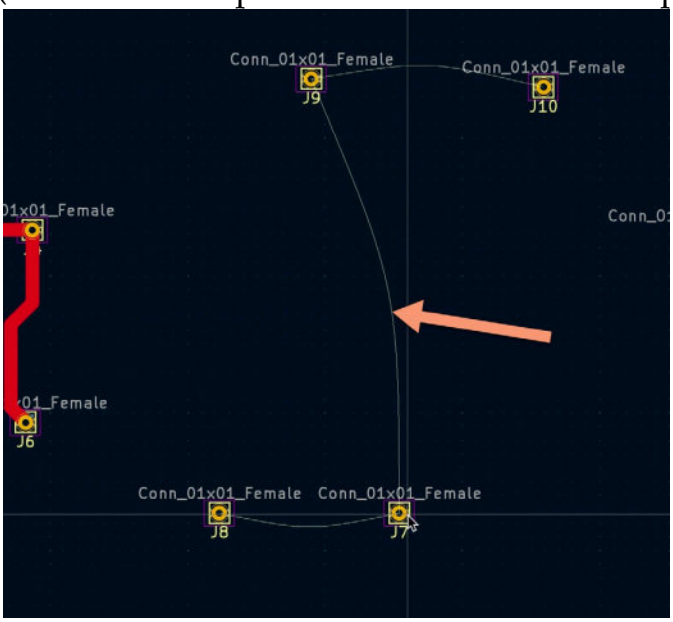


Figure 7.14.4: A ratnest line connects J10, J9, J8, J7.

As you can see, a new ratnest line connects the pads of J10, J9, J8, and J7. This line uses a global net label to connect the wires that connect the pins of the same symbols. In other words, you have created an electrical connection between pins and wires that are drawn in different sheets of the same schematic.

Another interesting change to notice is that the global net label I have created has generated a new net. Open the schematic setup window, and click "Net Classes." You will see a new row in the Net list: "Global_1" (see below).

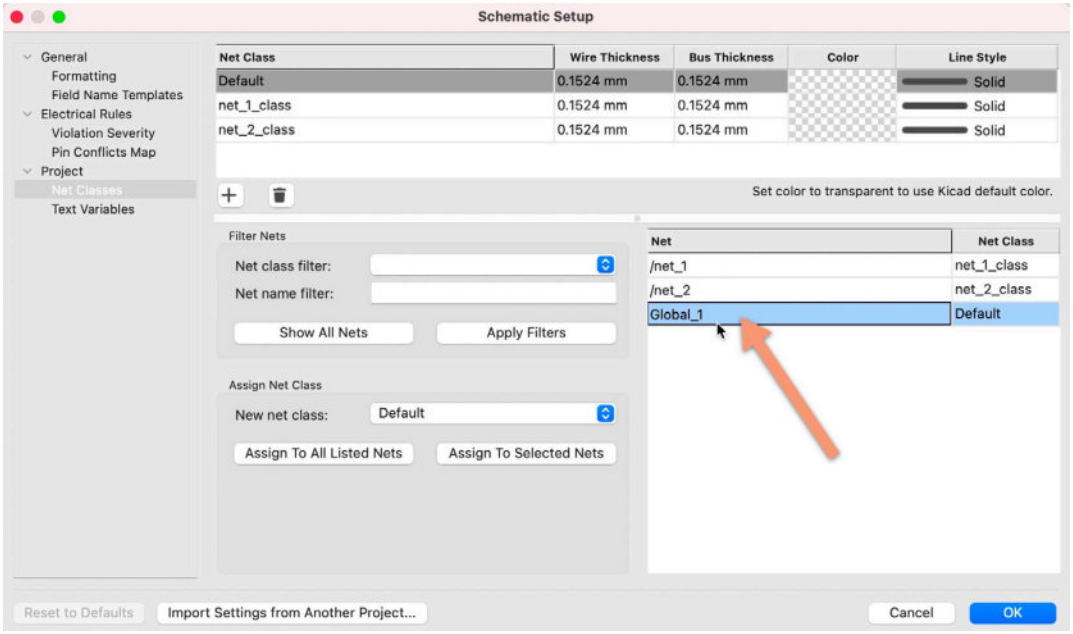


Figure 7.14.5: A new net with the name "Global_1".

You can treat this new net as any other net and assign it to a net class.

As I mentioned earlier, the second way to electrically connect elements that span across multiple schematic sheets is to use hierarchical labels. You can learn how to use hierarchical labels in the next chapter.

15. Hierarchical labels and import sheet pin

You learned how to create electrical connections between elements in multiple hierarchical sheets using global labels in the previous chapter. This is the equivalent of using a global variable in a programming language. While global variables work, they are typically frowned upon in the programming community. Their perceived "advantage" of convenience has disadvantages, such as their impact on performance, the risk of conflicting global variable names, and rendering the code less readable and easier to break. The same applies to global variables in KiCad. Use them with care, and first consider using hierarchical labels.

To continue with the programming example, think of hierarchical labels as a parameter that you can pass to a function in a programming language. This chapter will show you how to create hierarchical labels and their matching import sheet pins by continuing the simple example that I started in the earlier chapter, 13. Hierarchical sheets.

In Eeschema, go to sheet 2, and click on the hierarchical label button in the right toolbar (see Figure 7.15.1 below).

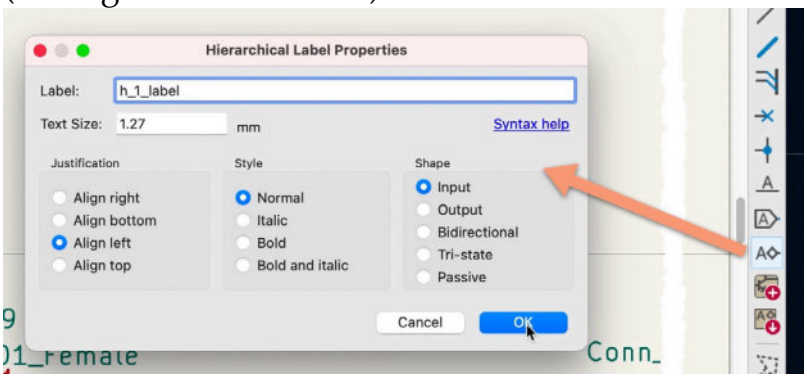


Figure 7.15.1: Creating a hierarchical label.

In the properties window that appears, type in a name. I use "h_1_label". Click OK to dismiss the window. Click again to set the label just above the wire, and use the wire tool to connect the label to the wire below it. The schematic in sheet two now looks like this:

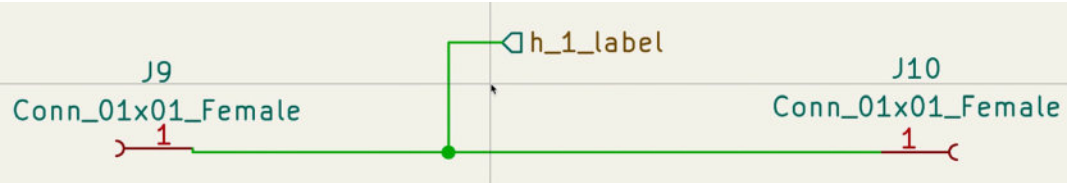


Figure 7.15.2: Attach the hierarchical label to a wire.

The hierarchical label is ready to connect to a matching hierarchical pin in the parent of this sheet (which happens to be the root sheet). To show you how convenient hierarchical pins are, I will create two more copies of the schematic in sheet two and edit their hierarchical labels to "h_2_label" and "h_3_label". Remember to annotate the new symbols. Below you can see the updated sheet 2:

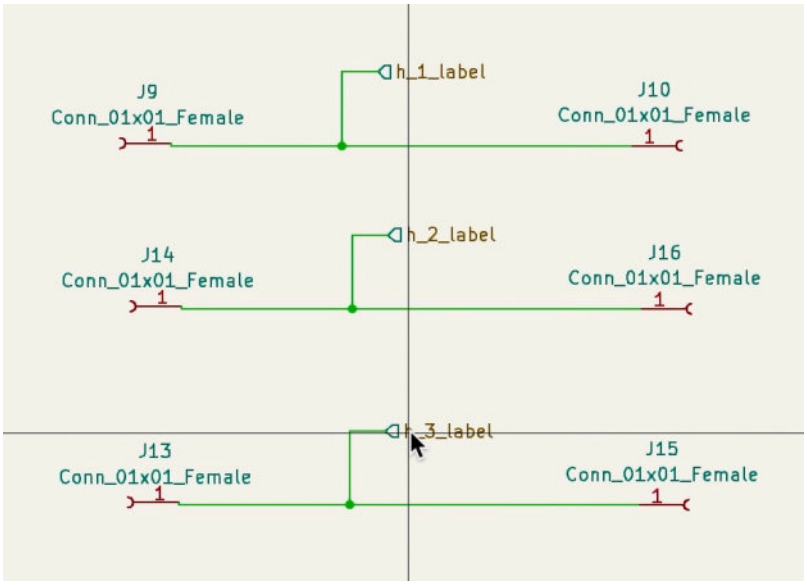


Figure 7.15.3: This sheet contains three hierarchical labels.

Continue to work on sheet 1. Click on the "import a hierarchical sheet pin" button on the right toolbar, then place your mouse pointer inside the hierarchical sheet box, and left-click. You will see a copy of one of the hierarchical pins. Place the label somewhere on the left edge of the box using the mouse, and click again. You will see a second hierarchal pin. Again, click somewhere in the box (close to the left edge) to place it. The third click will yield the last pin, which you can place next to the other pins with a final left click. The hierarchical sheet box in sheet one now looks like this:

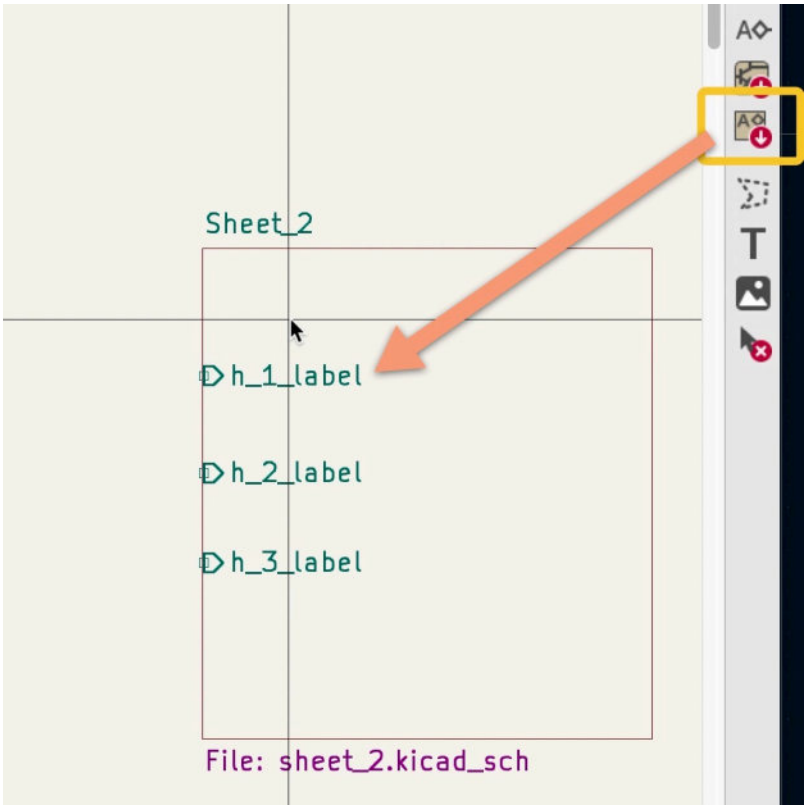


Figure 7.15.4: Three hierarchical pins imported in Sheet 1.

You have just imported the three hierarchical pins from sheet 2 to sheet 1. These are regular pins, so you can go ahead and connect wires, local net labels, or even global labels to them. For example, below, I have connected a net label to the top and bottom hierarchical pins and used a wire to connect the middle hierarchical pin to another wire in the schematic:

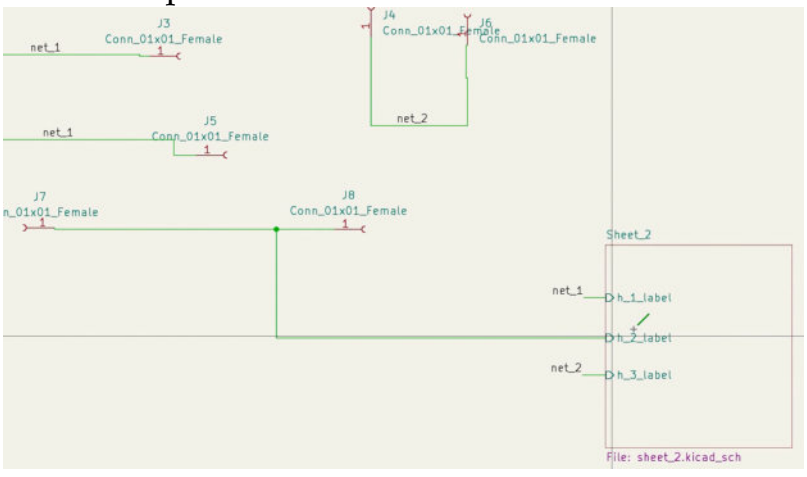


Figure 7.15.5: A net label connected to a hierarchical pin.

The symbols and pins in sheets 1 and 2 are now fully electrically connected using hierarchical labels and pins. To confirm, switch to the layout editor and update the changes from the schematic. Below you can see the new

ratnests that depict the electrical connections between footprints that are associated with symbols in the schematic editor's sheet one and sheet 2:

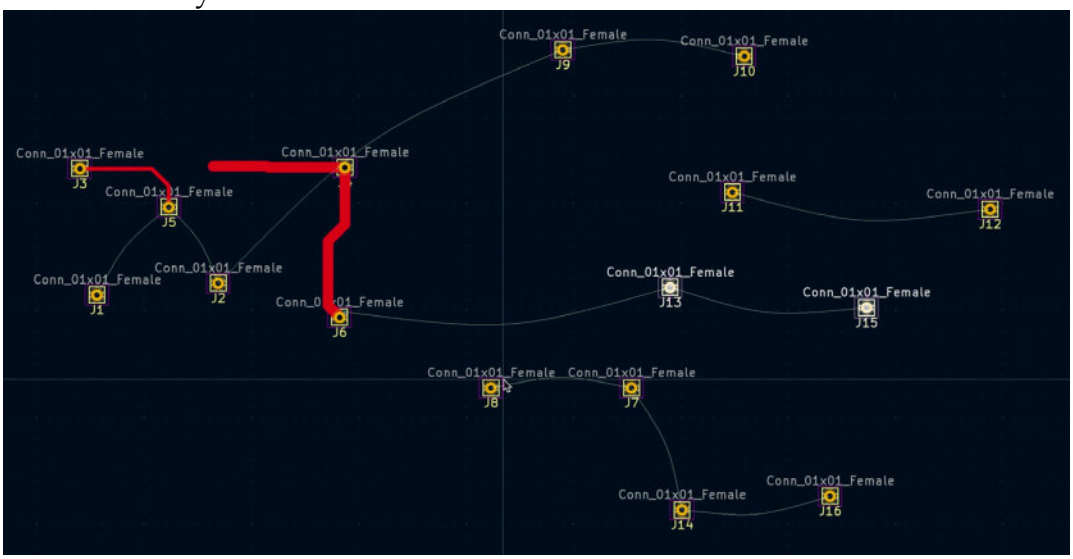


Figure 7.15.6: The hierarchical labels and pins created new ratnest lines in the layout editor.

As you can see above, ratnest lines connect pads of connectors J9, J10, J14, J16, J13, and J15 that exist in Sheet 2 to pads J8, J7, J4, J6, J3, and J1 that exist in Sheet 1.

Hierarchical labels and pins are the preferred methods of creating electrical connections between symbols and pins in different hierarchical sheets.

16. Electrical rules and customization

As you are working on your schematic, adding symbols, wiring, and making changes, you will frequently use the Electrical Rules Checker (ERC) tool. The ERC will help you find and fix violations, such as leaving pins unconnected or incorrect wirings of power pins. You certainly should run an ERC and correct any violations that it reports before you continue work in the layout editor.

Run the Electrical Rules Checker

To invoke the ERC tool window, click on the relevant button in the top toolbar that I have marked as "1" in Figure 7.16.1 below:

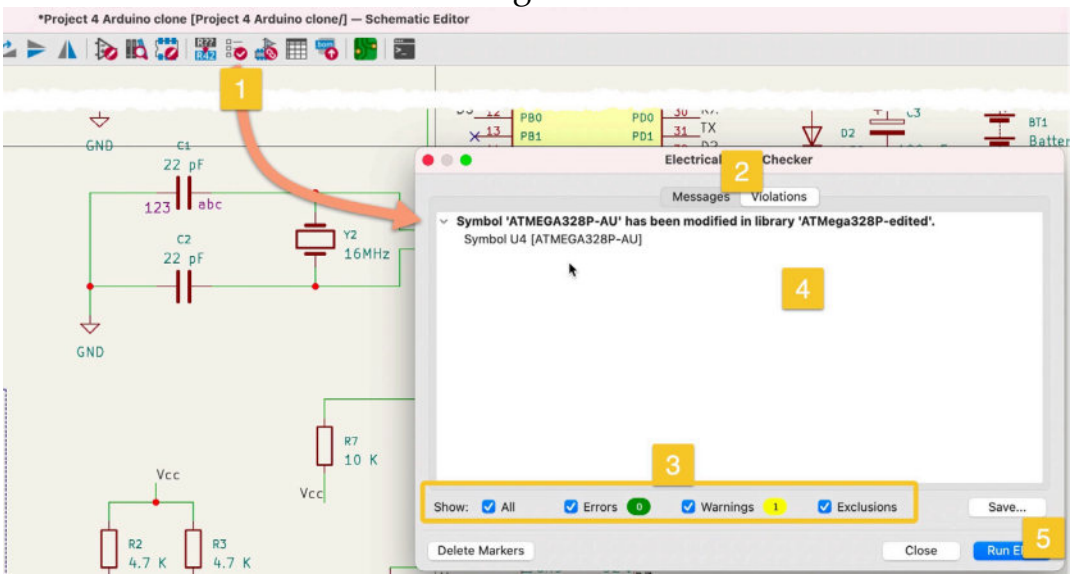


Figure 7.16.1: An Electrical Rules Check.

The ERC window contains two tabs: Messages and Violations ("2"). In Messages, you will see helpful information that does not represent errors that need immediate attention. In Violations, however, you will see a list of violations that you must correct before continuing.

At the bottom of the window ("3") are checkboxes that allow you to filter the types of messages that appear in the issues listing ("4"). To run the ERC, click on "Run ERC" ("5").

In the example above, the ERC lists one issue relating to a modification that I made to a symbol in the schematic. I modified it to only exist in my

schematic and did not save it in the library. I did this intentionally as a "once-off" modification, so in this case, I chose to ignore this issue and close the ERC window.

To demonstrate an issue that consists of a violation that I would have to fix, I have deleted a segment from the wire that comes out of pin 8 of symbol U2, like this:

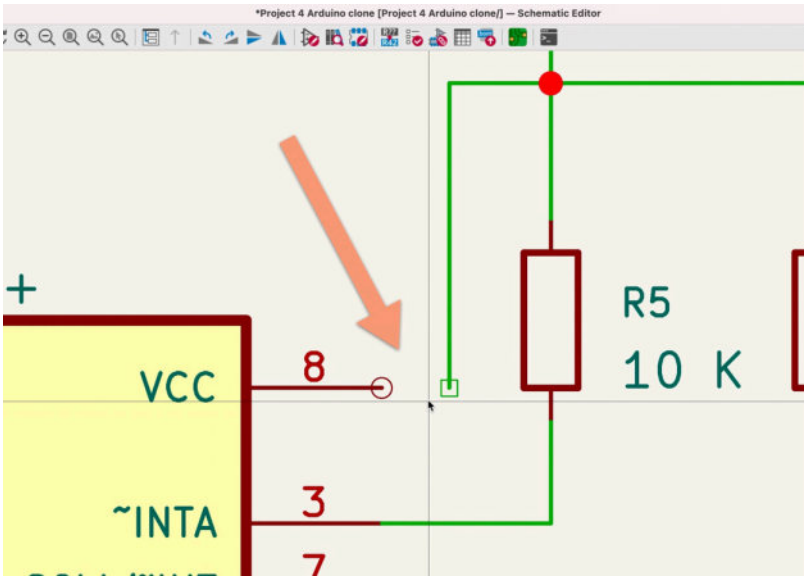


Figure 7.16.2: I have made an error.

Repeat the ERC. The tool adds a new issue to the violations list:

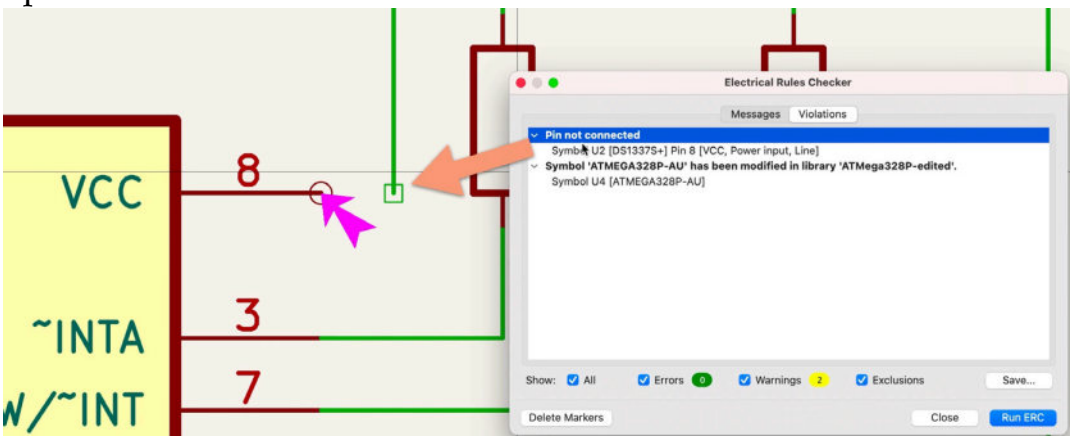


Figure 7.16.3: The ERC has detected the error.

The ERC has detected the error and listed it as a violation of type "Pin not connected." You can click on an issue, and the layout editor will pan the sheet to display the location of the violation. Notice the arrow that points to the pin that I forgot to connect.

Notice that in Figure 7.16.3 (above), the two issues listed in the Violations tab are classified as "warnings" (see figure "2" in the Warnings checkbox at the bottom of the window). Normally, violations of type "Pin not connected" are

listed as Errors. However, I have changed the ERC configuration to classify this violation as a "Warning." The ERC in KiCad 6 is configurable and customizes the tool to fit your project requirements.

Customize the Electrical Rules Checker

To customize the ERC, bring up the Schematic Setup window (under File). There are two tabs under the Electrical Rules group: "Violation Severity" and "Pin Conflicts Map."

In "Violation Severity," you can change the classification of a range of violations to "Error," "Warning," and "Ignore." In the figure below, you can see that I have changed the classification of the "Pin not connected" violation to "Warning." The default is "Error."

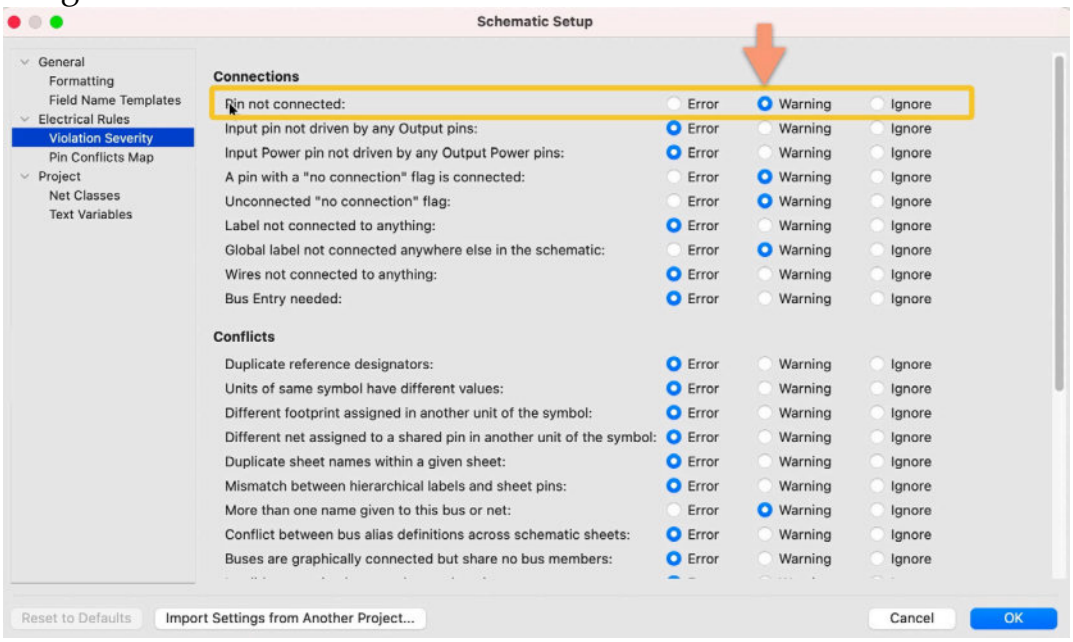


Figure 7.16.4: The ERC' Violation Severity tab.

I will change this setting back to the default "Error":

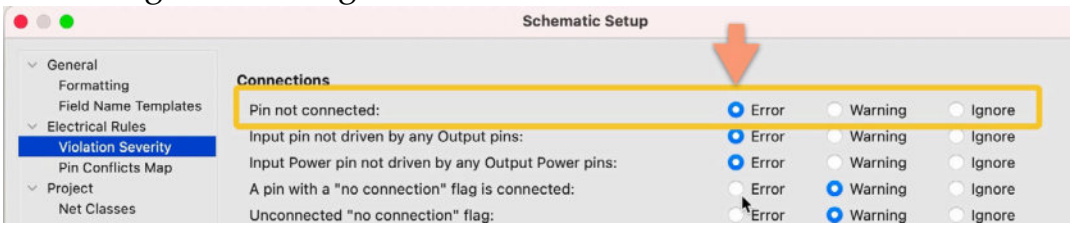


Figure 7.16.5:Changed the "Pin not connected" severity.

And then repeat the ERC. The "Pin not connected" violation is now classified as an Error:

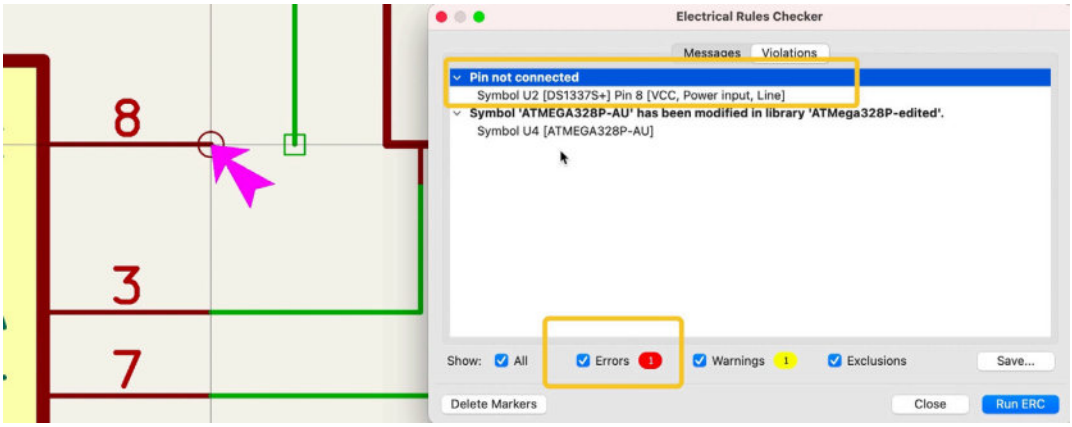


Figure 7.16.6: The “Pin not connected” violation is now an Error.

I will now fix this error before I continue and restore the missing wire.

The "Pin Conflicts Map" allows you to set the error conditions for connections between pins of the same or different roles. You can see the Pin Conflicts map in Figure 7.16.7 below.

Figure 7.16.7: The Pin Conflicts Map.

In the example above, you can see three possible pin-to-pin connection states:

- A green indicator designates an allowable connection.
- A red indicator designates a violation.
- An orange indicator is uncertain.

For example, a connection between two output pins (marked "1" above) is a violation, and you have to fix it before continuing. A connection between a tri-state pin and an output pin may or may not be correct, so you should check it. A connection between two passive pins ("3") is allowed.

You can change these states by clicking on a box to cycle through to the classification you want.

Let's look at an example. Below you can see a part of my test schematic:

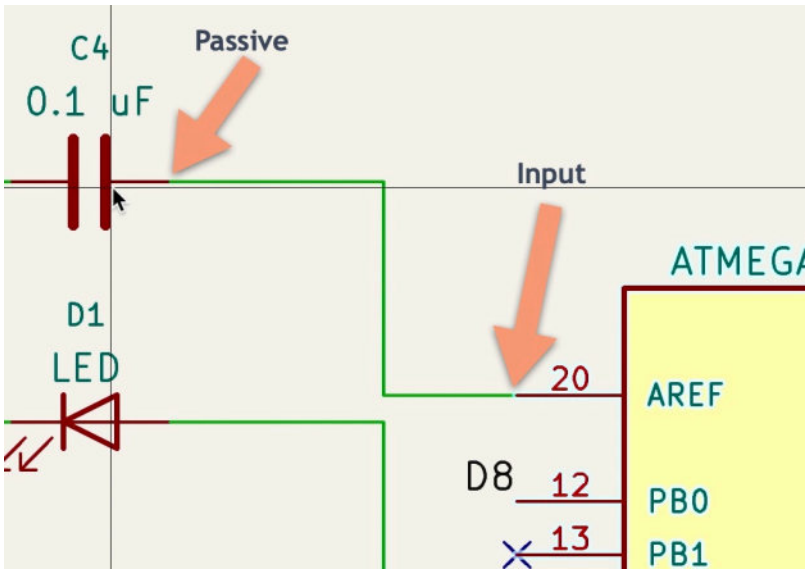


Figure 7.16.8: Connection between a passive pin and an input pin.

In this example, a passive pin is connected to an input pin. You can find out the pin type by using the symbol editor and looking at the pin's configuration. To bring up the symbol editor, right-click on the symbol and click on "Edit with Symbol Editor" in the context menu. Then, double-click on the pin in question to bring up its properties window. Below is the properties window for the capacitor pin 1:

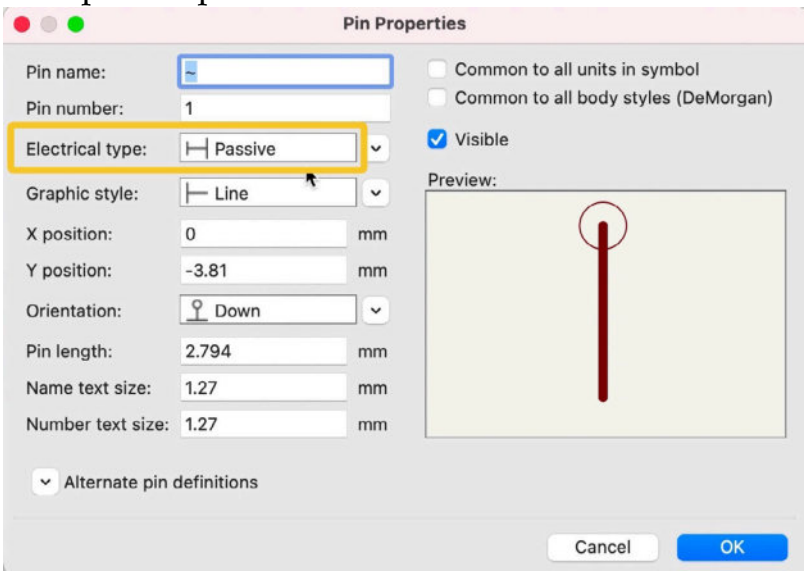


Figure 7.16.9: Electrical type of pin 1 of the capacitor.

In the default conflicts map (see Figure 7.16.7 above), a connection between a passive pin and an output pin is allowed. Let's change this to an error. The pin to pin connections map now looks like this:

Figure 7.16.10: Passive to input pin connection is now an error.

Click OK and repeat the ERC. You can see the result below:

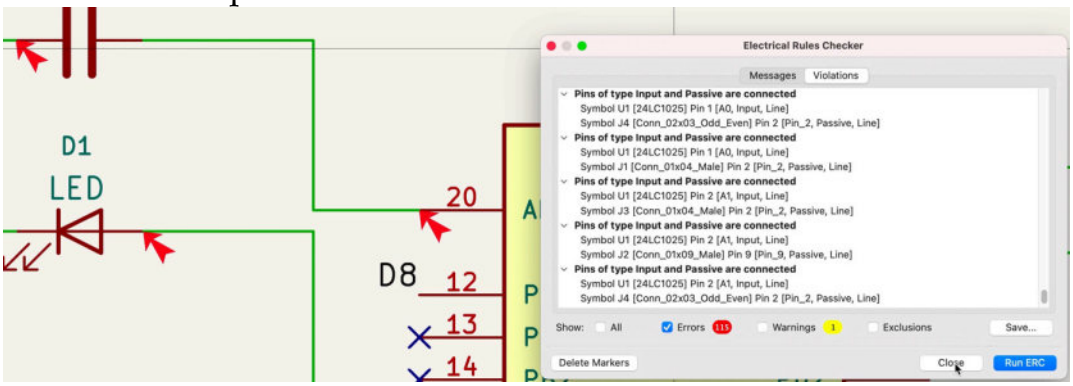


Figure 7.16.11: Unconnected pins are now OK.

The ERC has found 115 errors, indicating that there are a lot of "erroneous" passive to input pin connections. Of course, this happened because I tweaked the pin conflicts map as an example. The default setting is more sensible, so I will change the passive to input pin connect back to the original green label.

In summary, while the default ERC settings of the schematic editor are appropriate for most projects, you may want to make suitable changes for your specific project circumstances. You can customize the way that the ERC works and its various violation classifications using the Violation Severity and Pin Conflicts Map tabs in the Schematic Setup window.

17. Bulk editing of schematic elements

In KiCad 6, you can edit many of the properties of the elements in a schematic diagram in bulk. Bulk editing is an important productivity feature that can speed up your development significantly. In the schematic editor, you can use bulk editing to do things such as apply a new style to all text labels that contain symbol reference designators, or change the symbol of all resistors from the US (IEEE) version to the European (IEC) version.

I will demonstrate a bulk editing scenario with an example. Consider Figure 7.17.1 below:

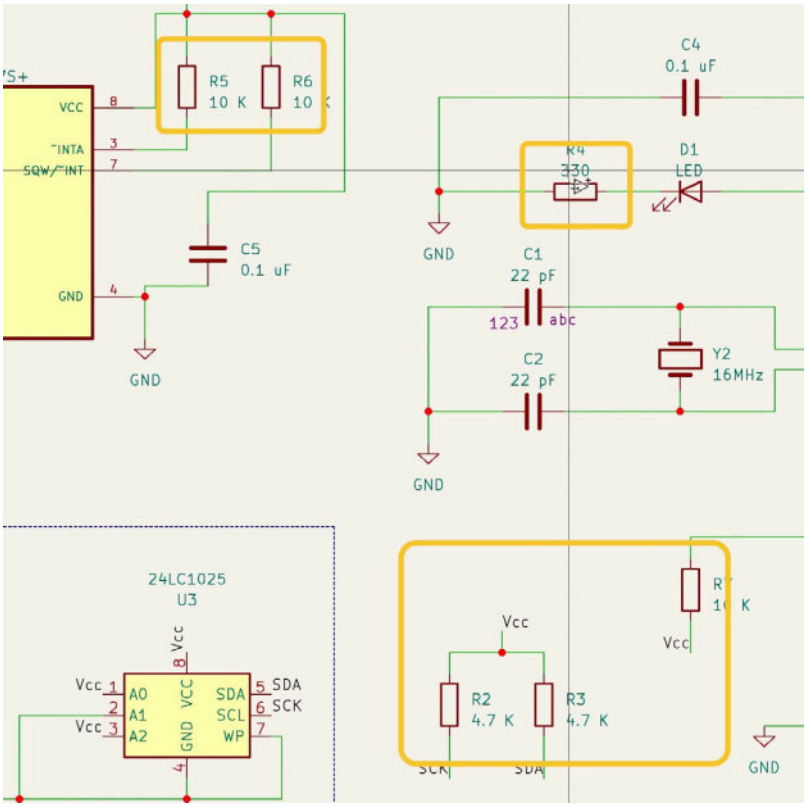


Figure 7.17.1: I will change the resistor symbols in bulk.

Suppose you want to change all resistor symbols to use the US notation instead of the European. One way to do this is to go to each resistor symbol properties and change its symbol. Then, repeat the process for each resistor. See the process below (Figure 7.17.2).

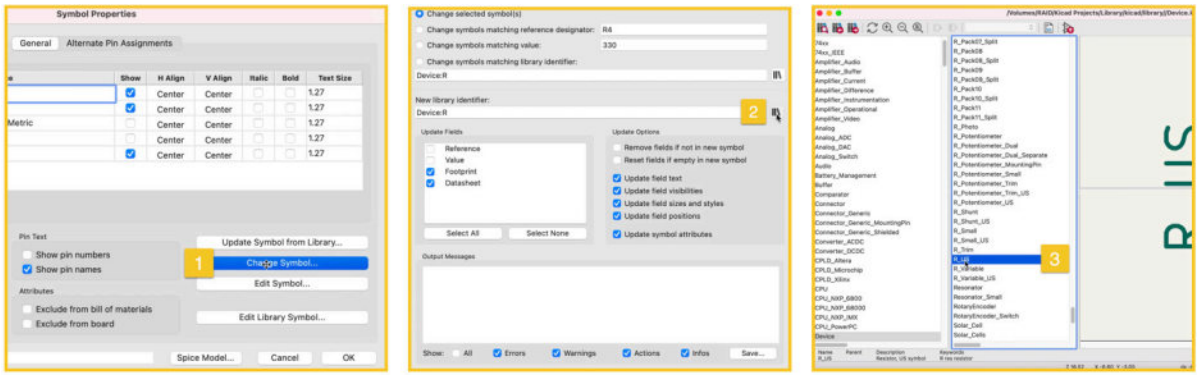


Figure 7.17.2: Change a symbol, one at a time.

Double click on the symbol to bring up its properties window. Click on "Change Symbol" ("1"), and then click on the library button to bring up the symbol library browser ("2"). In the browser, find the new symbol and double-click it to select it ("3"). Exit the open windows and return to the editor. The resistor now has a new symbol:

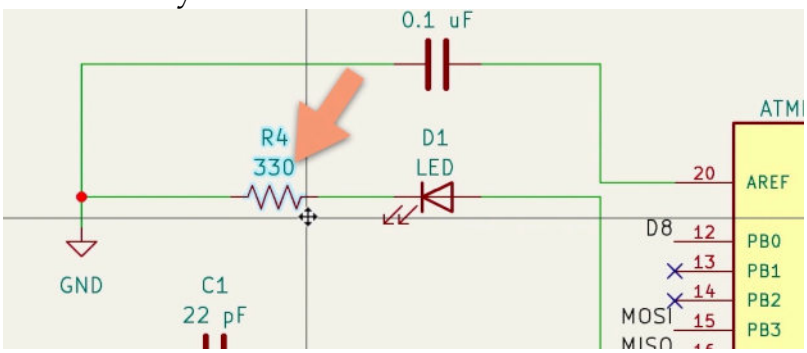


Figure 7.17.3: The resistor has a new symbol.

For a couple of symbols, this method is sufficient, but for more, it is not. Let's look at the bulk-editing method.

From the Edit menu, choose "Edit Symbol." The Change Symbols window appears:

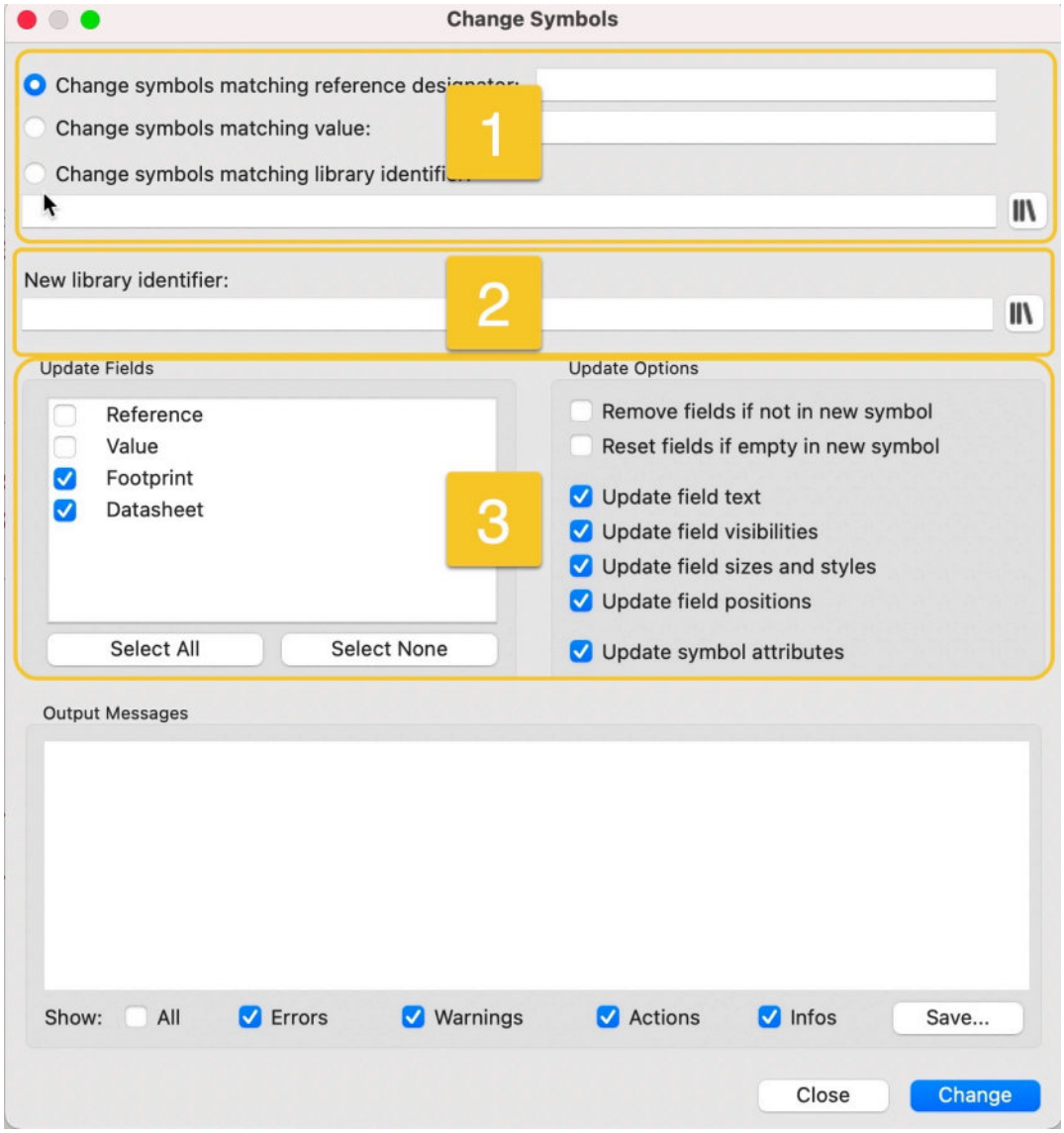


Figure 7.17.4: The Change Symbols window.

The Change Symbols window contains three main groups of widgets:

1. "Symbols to change" filter. Use these widgets to specify which symbol or symbols you would like to change. You can target symbols by their reference designator (i.e., all symbols with designator starting with "U"), their value (i.e., all symbols with value "330"), or their library identifier (i.e., all symbols with library identifier "R").

2. Specify the new library identifier. You can type this identifier in the field or use the browser to find it.

3. Update fields. You can choose which of the original symbol fields you would like to change.

To continue with my example, I will use the Change Symbols window to change the symbol of all resistors in my schematic to use the US version. The

common attribute of all resistors in my schematic is their library identifier. You can find this identifier by going into the properties window of one of the resistors:

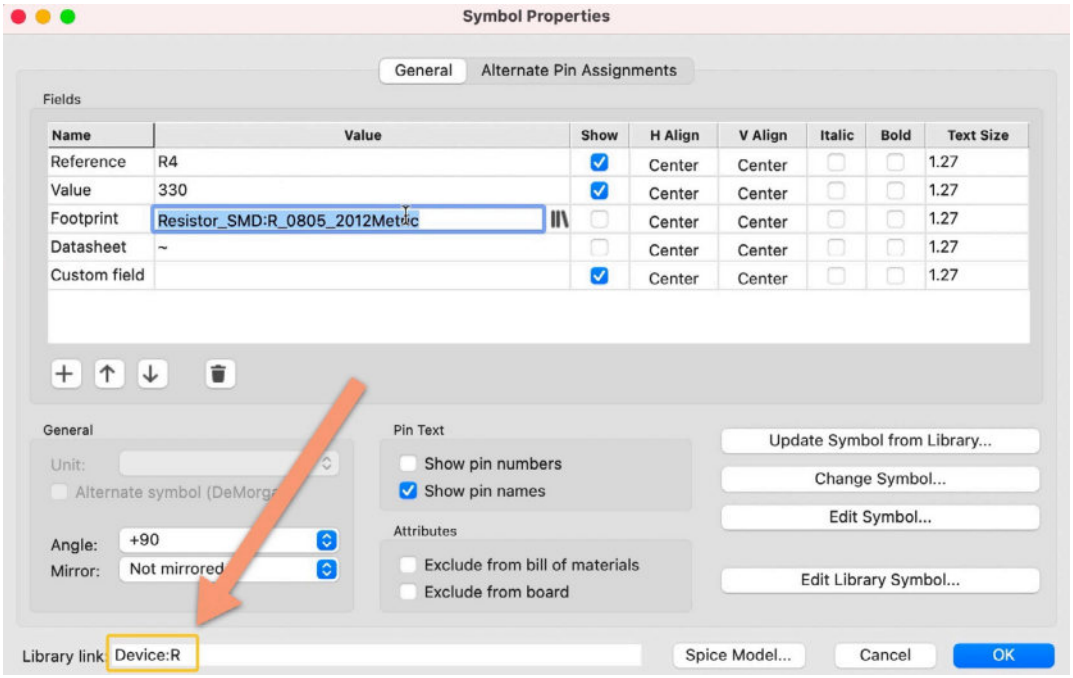


Figure 7.17.4: The library identifier for this symbol is in its properties window.

The library identifier for this resistor is "Device:R."

Continue in the Change Symbols window. Copy the library identifier into the "Change symbols matching library identifier" field. Then, type the library identifier of the new symbol in the "New library identifier" field (or click on the library button to use the browser). The Change Symbols window now looks like this:

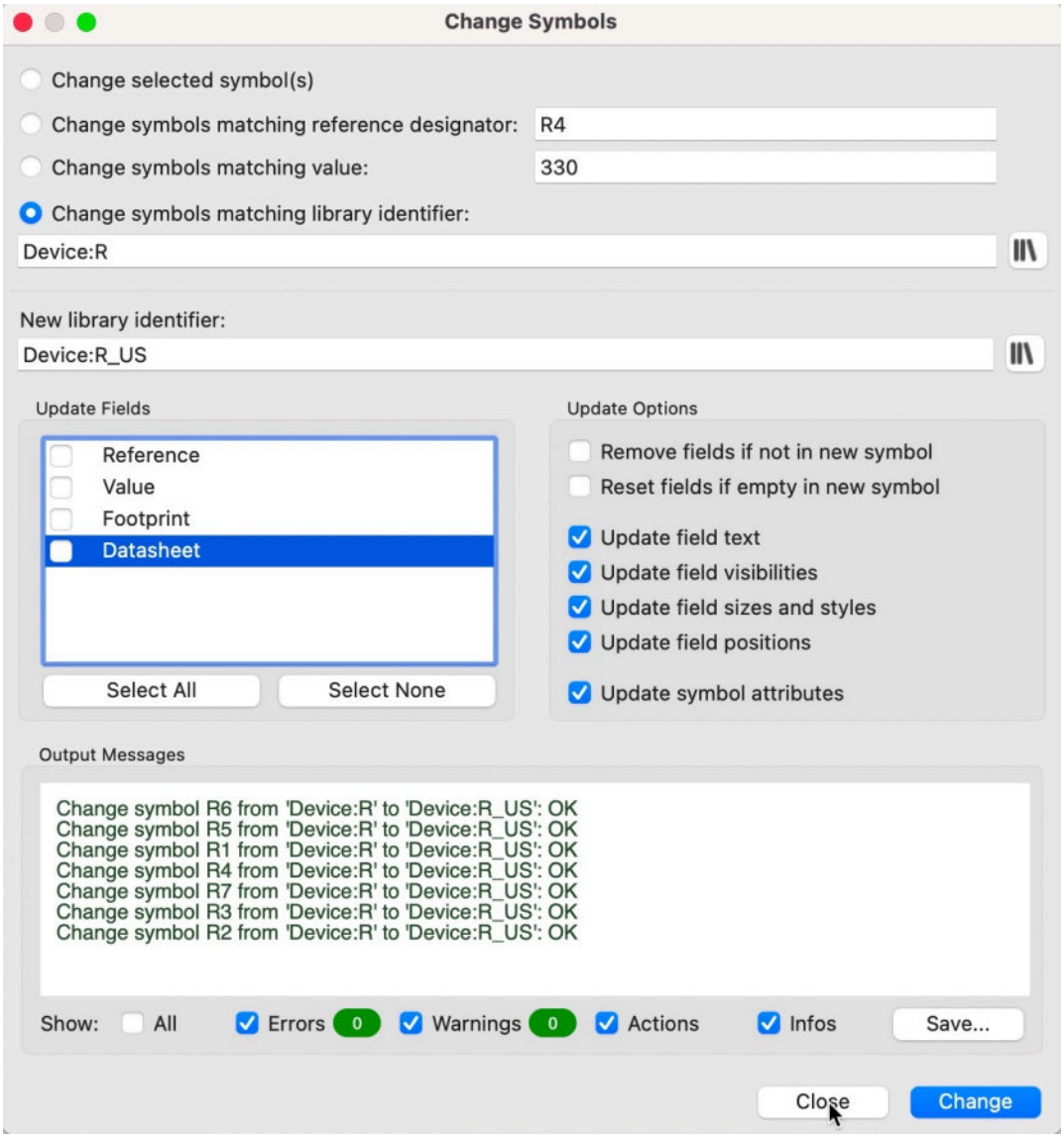


Figure 7.17.5: Changing the symbol for all resistors matching “Device_R”.

Click "Change" and notice the output messages confirming that the various resistors have a new symbol. Click Close, and confirm the new symbols in the editor sheet:

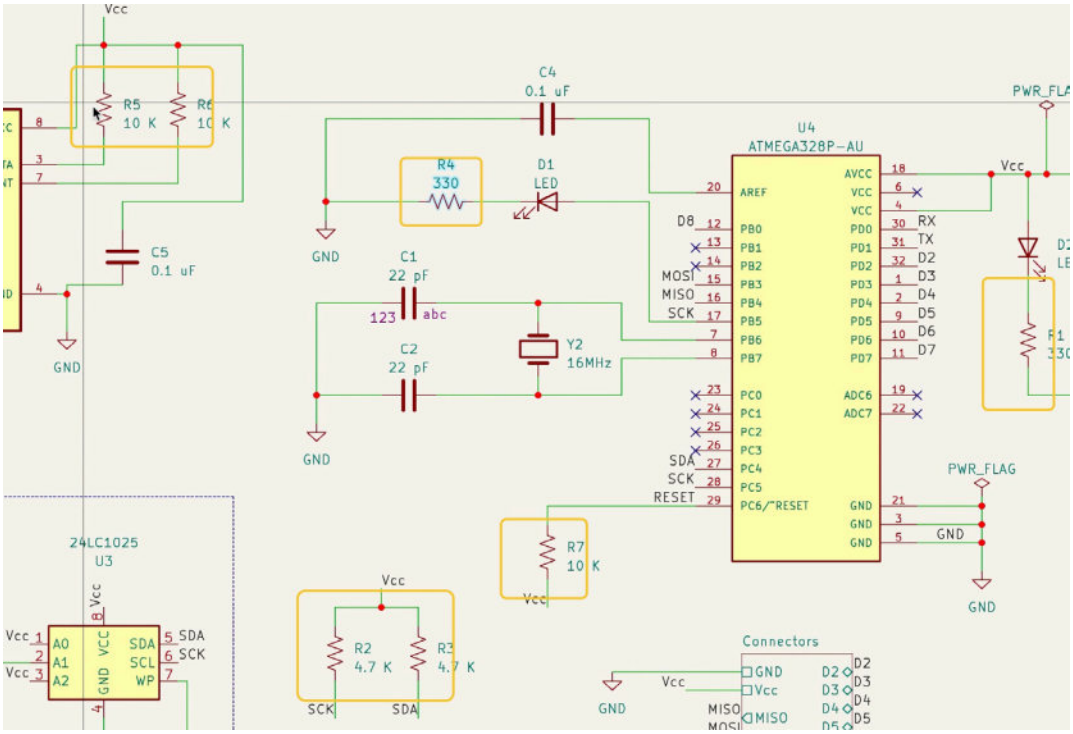


Figure 7.17.6: Changed the symbols for all resistors.

I prefer to use the EIC notation, so I will use Cmd-Z to undo those changes.

You can use a similar bulk-editing tool to change text and graphic elements' attributes and replace any text with new text. Both tools are available in the Edit menu: "Edit Text & Graphics Properties" and "Find and Replace." For example, to change all text "RESET" to "RST," I can use the Find and Replace window like this:

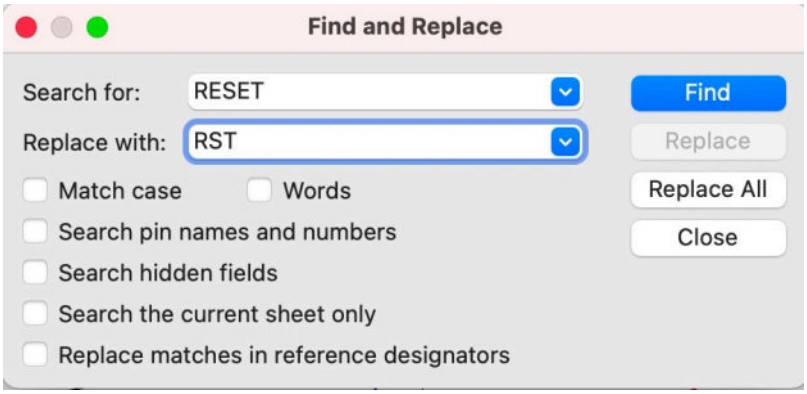


Figure 7.17.7: Changing “RESET” to “RST”.

This change will occur in all sheets of the schematic.

If I want to change the color and thickness of all wires in my schematic, I will use the "Edit Text and Graphic Properties" window, like this:

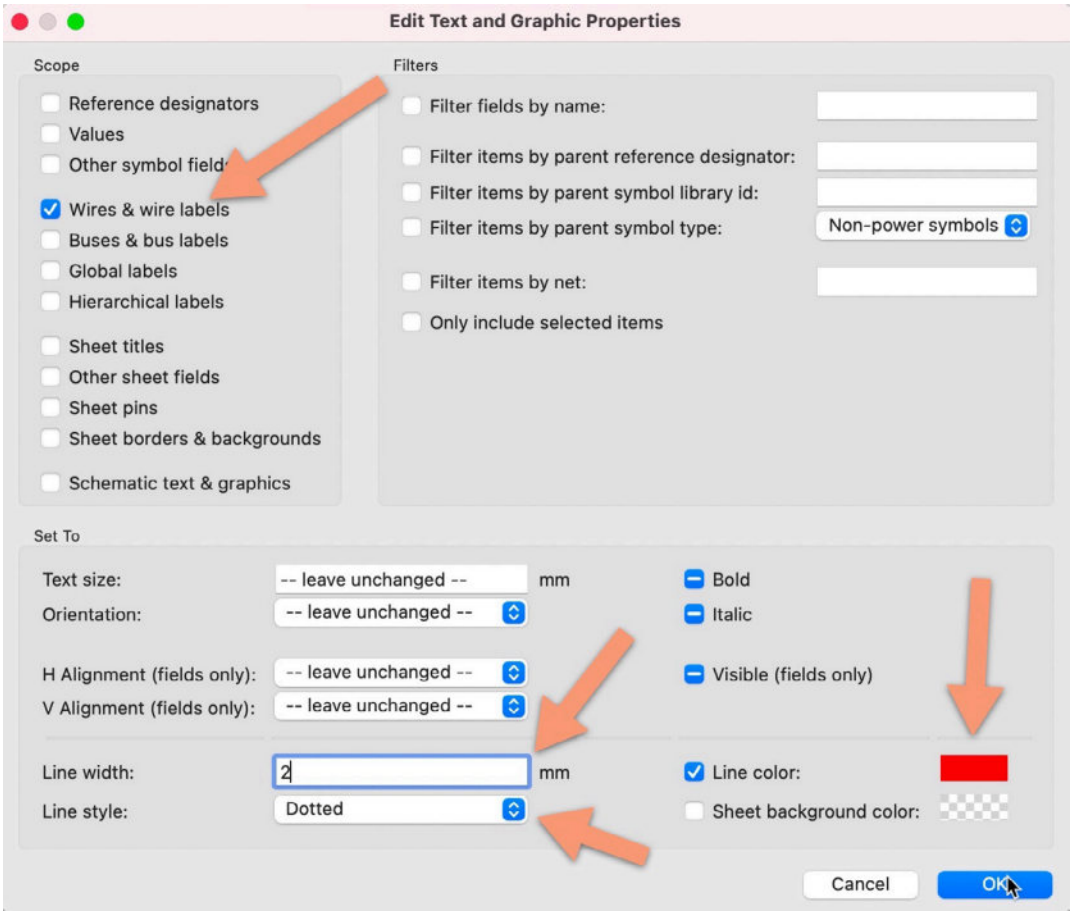


Figure 7.17.8: Changing the way that wires look.

Above, I use arrows to point to the selected scope and new properties I apply to the wires. Below you can see the result of this bulk change:

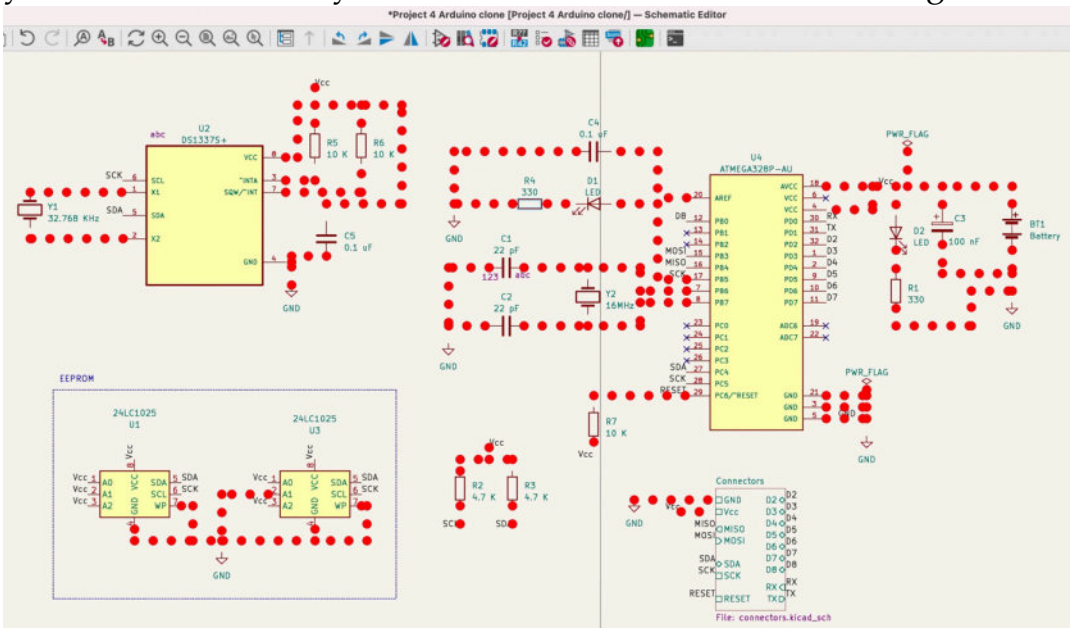


Figure 7.17.9: Wires with new styling.

Of course, the new style of the wires is arguably objectionable, and you can always revert to the original by undoing the changes (Ctr-Z/Cmd-Z).

Part 8: Fundamental Kicad how-to: Footprints and Pcbnew

1. Introduction

In the following chapters, I will give you an overview of the layout editor's user interface to find the various tools and options.

I will also show you how to work with footprints, including finding the right one from the footprint chooser, installing external footprint libraries, or creating custom footprints.

Finally, I'll show you how to work with frequently used layout design elements, like filled zone and measurement tools, and configure and use the Design Rules Checker (DRC).

By the end of this part, you will know everything you need to help you work through the example projects in this course.

Keep in mind that KiCad, and Pcbnew in particular, have many more features and capabilities than the ones I demonstrate in this part of the book.

This part of the book aims to teach you only what you need to know to make the most of the upcoming projects.

In the last part of this book, you will find several recipes with practical guides and examples of more advanced features.

There are two ways to start Pcbnew. First, if you are using the KiCad main project window, click on Tools and then PCB Editor, or click on the PCB Editor button in the right pane of the project window.

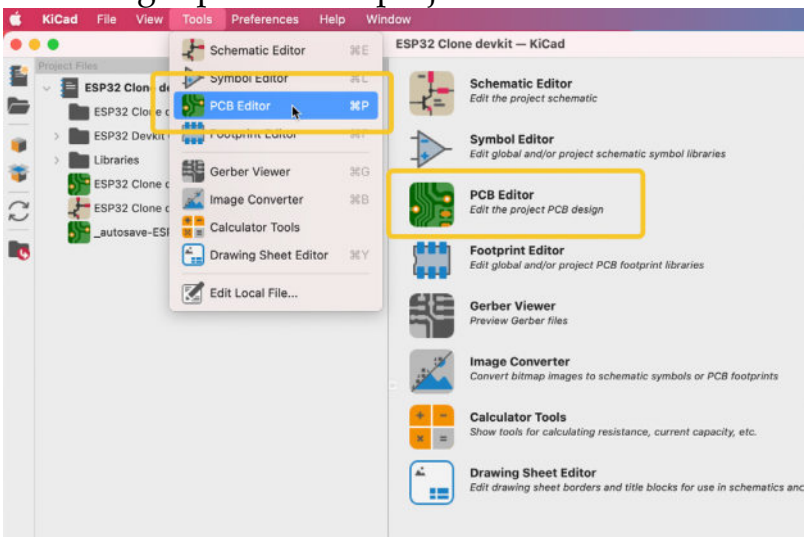


Figure 8.1.1: Starting the PCB editor from the KiCad project window.

Second, if you are working in the schematic editor, click on the PCB editor button in the top toolbar:

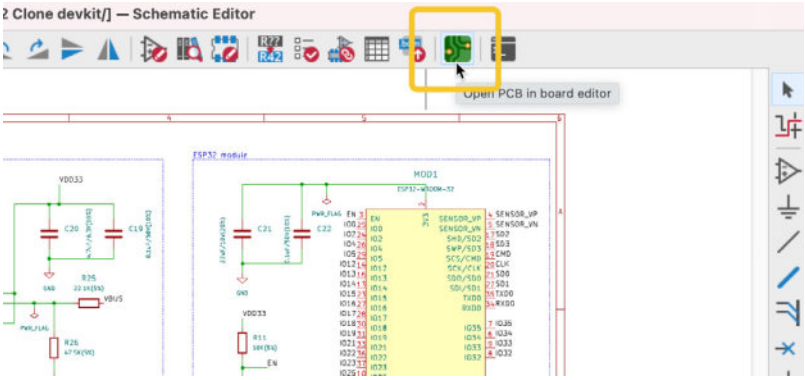


Figure 8.1.2: Starting the PCB editor from the schematic editor window.

Once you are in the PCB editor (which I also refer to as the “layout editor” or “Pcbnew”), you will be able to start designing your PCB layout. The layout editor looks like this:

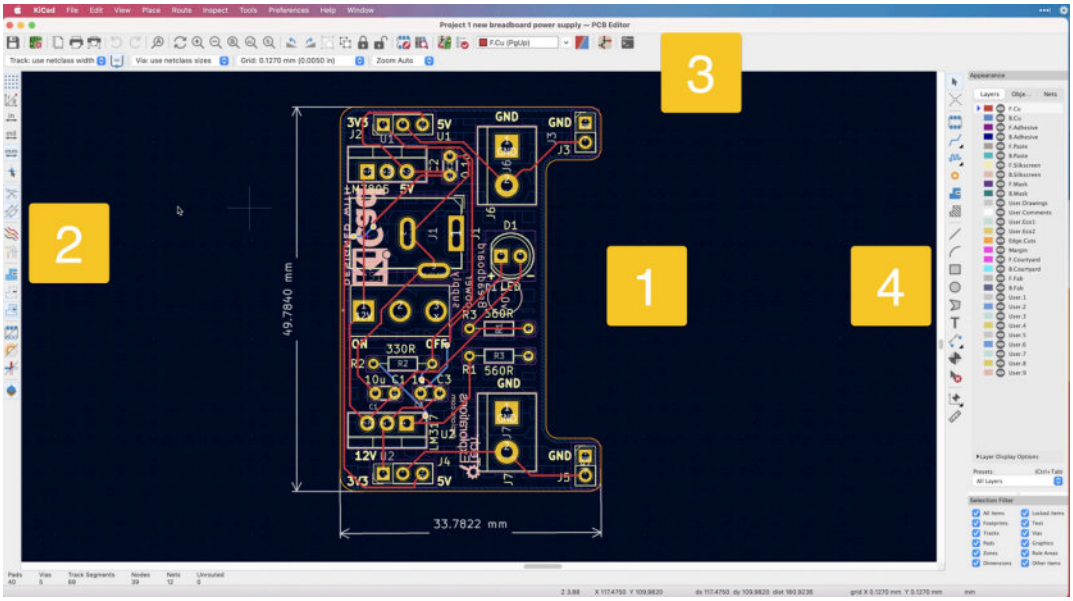


Figure 8.1.3: The layout editor window.

The main area consists of the design editor, where your PCB exists (“1”). To design the PCB, you will use the tools that are available in the three toolbars: left (“2”), top (“3”), and right (“3”). There are also many tools and configuration options available via the top menu bar.

You will learn about Pcbnew's functions in this part of the book and the Recipes at the end of the book.

2. Left toolbar

In this chapter, you will learn about the various buttons and functions available from Pcbnew's left toolbar. You can see the left toolbar below:

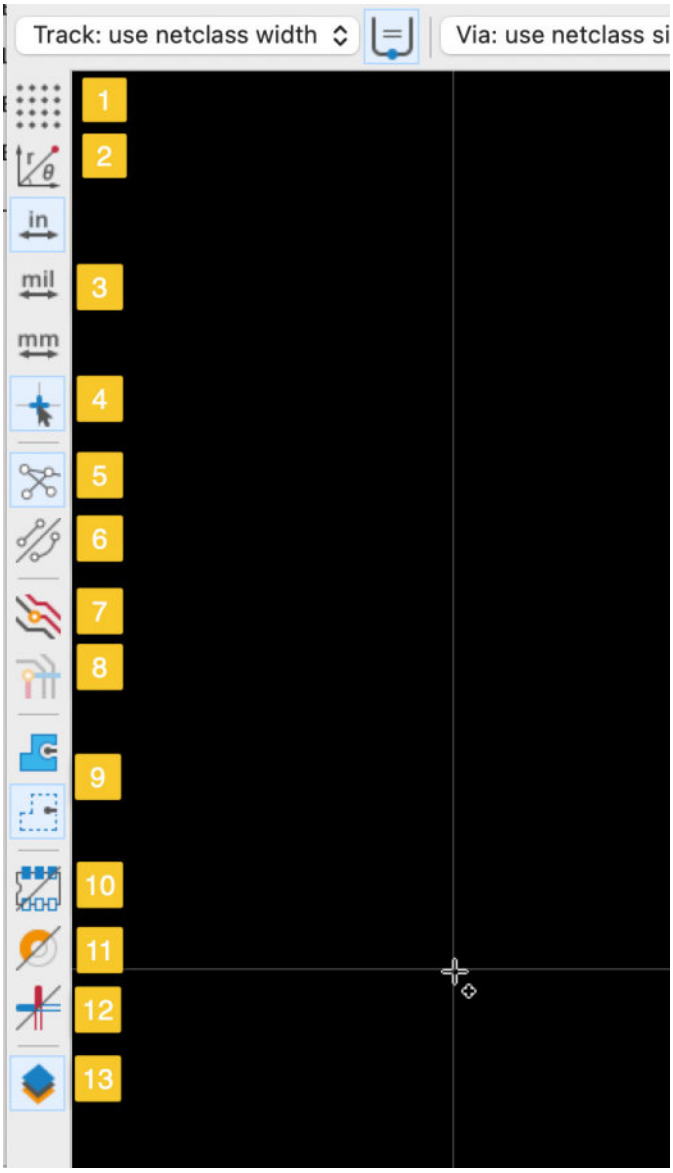


Figure 8.2.1: The left toolbar.

Let's dive into the buttons in this toolbar.

Display grid

Click the Display Grid button ("1") to toggle the grid lines on and off. You can see an editor segment with the grid lines off (left) and on (right) in the example below.

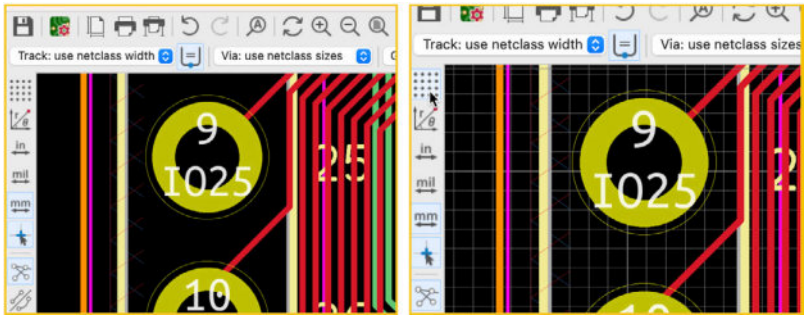


Figure 8.2.2: Grid lines.

The grid lines are helpful when placing footprints or drawing copper tracks because they provide a way to compare relative positions between objects visually. Alongside the snap-to-grid option (you can enable this from the Preferences window, under PCB Editor Display Options), the grid lines are an indispensable tool that I usually turn off only when I have completed my design work.

Coordinate system

You can use this button ("2") to toggle between the polar and the cartesian coordinate systems. Depending on which system you have selected, the location of the mouse course is shown in the status bar. Below, you can see the cursor position values in the cartesian coordinate system (left) and polar system (right).

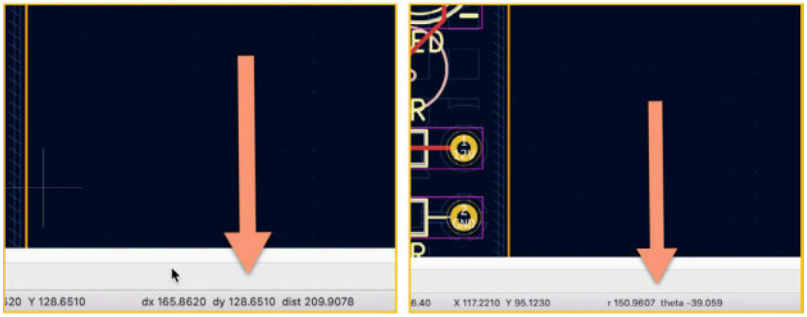


Figure 8.2.2: Coordinate systems: cartesian (left), polar (right).

Length unit

In the layout editor, you can measure length using inches, millimeters, or mils using the buttons in the units group ("3"). Depending on your choice, the respective unit is displayed in the status bar. Below you can see the status bar indicating inches (left), millimeters (middle), and mils (right).

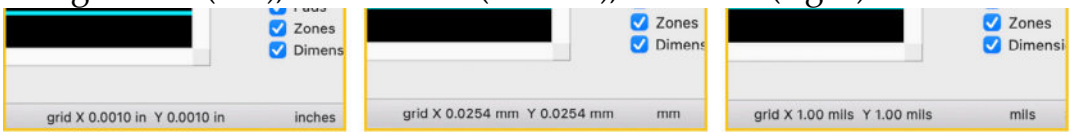


Figure 8.2.3: Length measurement units.

Cursor shape

You can select the shape of the mouse cursor. There are two shapes available: regular crosshairs or full-window crosshairs, and you can toggle between them using the cursor shape button (“4”). You can see the two types below.

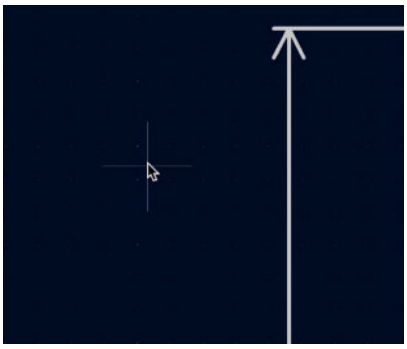


Figure 8.2.4: Small crosshair cursor.

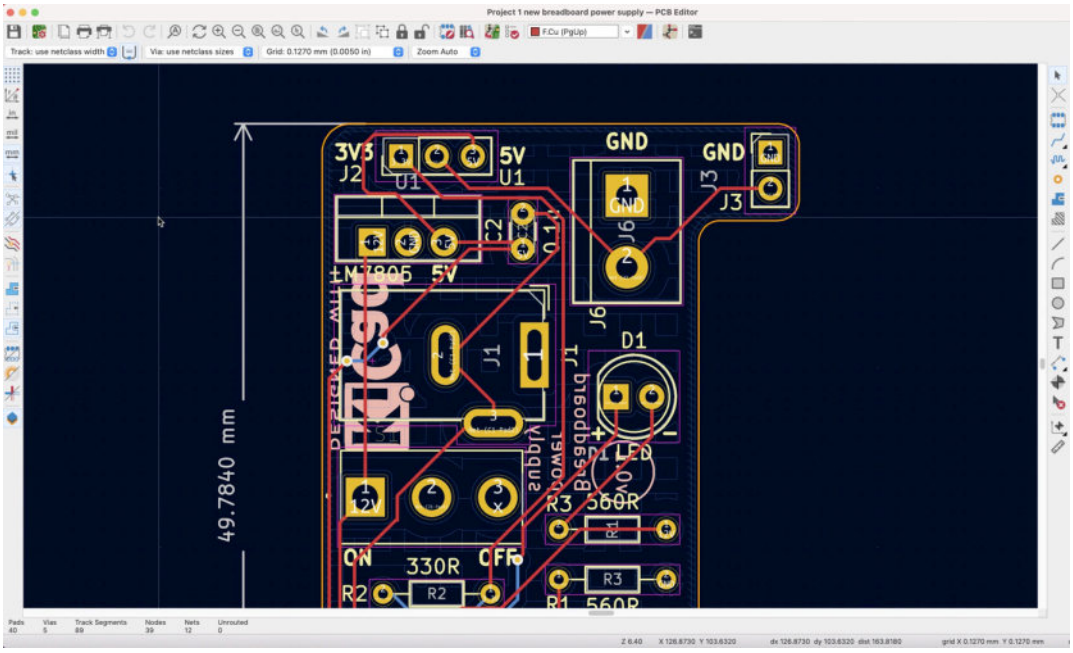


Figure 8.2.5: Full window crosshair cursor.

I use the full-window crosshair to help me determine relative positions between layout objects, especially when those objects are far from each other.

Show/hide ratsnest lines

You can use the ratsnest lines button (“5”) to show or hide the guidelines between unconnected pins. In the example below, you can see a ratsnest

indicating the unconnected segment between a pad and a copper track in the left image. In the right image, I have hidden the ratsnest line.

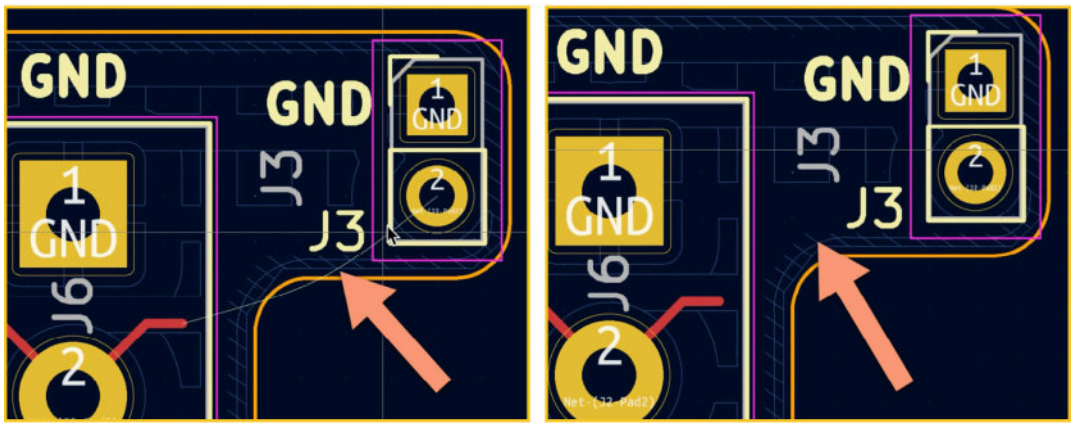


Figure 8.2.6: A ratsnest line showing (left), hiding (right).

Ratsnest line style

The layout editor can draw straight or curved ratsnest lines. You can toggle between the two using the button “6”. You can see the difference between the two styles below:

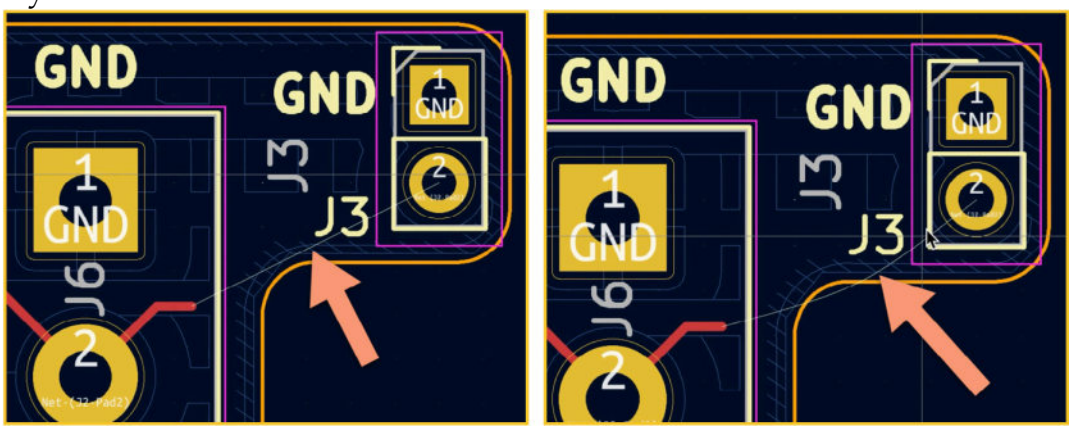


Figure 8.2.7: A ratsnest line straight (left), curved (right).

Inactive layers

This button (“7”) allows you to highlight the active layer by dimming any element that belongs to an inactive layer. These elements include things such as footprints, copper tracks, and text.

In the example below, I have enabled the front copper layer from the right toolbar (“1”) and then dimmed inactive layers (“2”). The result is that the elements of the front copper layer are pronounced compared to everything else.

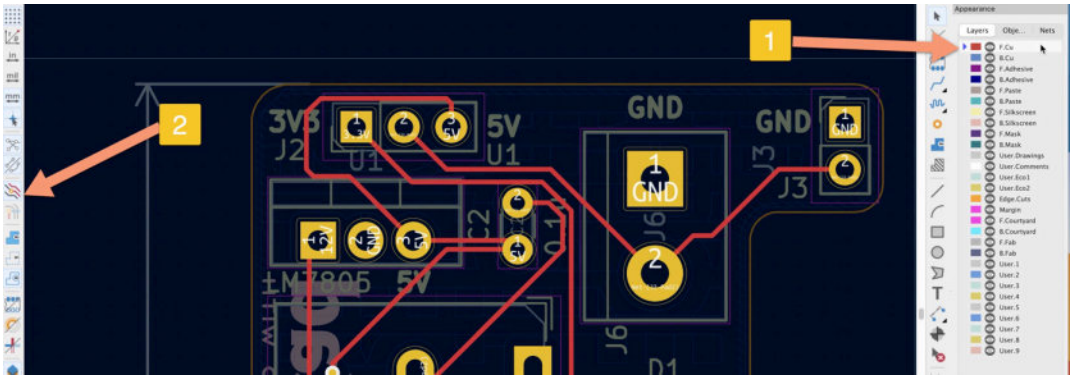


Figure 8.2.8: Dimmed inactive layers.

Net highlighting

At the time I am writing this chapter, this button does not seem to be working. This paragraph is a placeholder for a future update.

Copper fill style

Copper fills are areas on the PCB that are fully covered with copper. You can use the two copper fill style buttons (“9”) to choose how to depict copper fills in the editor. The first button will show the copper fill with high fidelity, appearing in the final manufactured PCB. The second button will only show the boundaries of the copper fills. The advantage of the first style is that you can see precisely what the copper pour will look like in the manufactured PCB. The advantage of the second style is that it produces a less cluttered layout.

You can see examples of the two styles below:

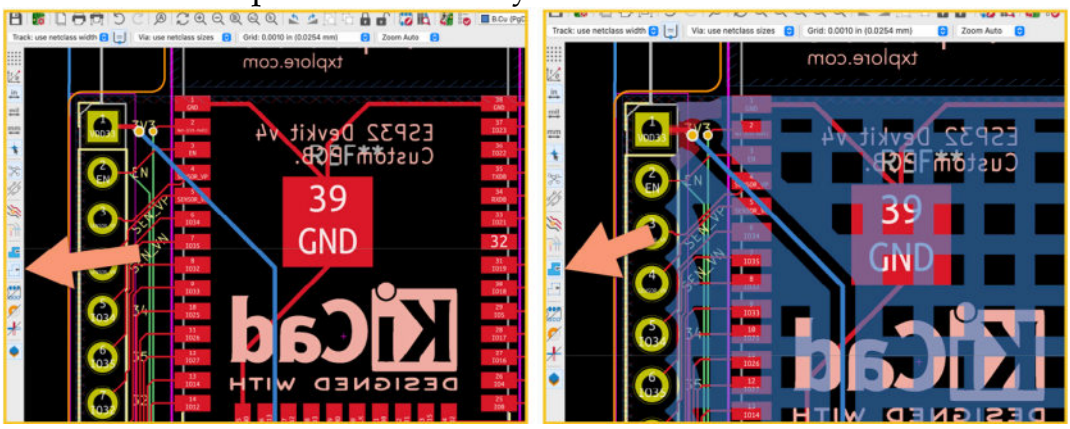


Figure 8.2.9: Filled areas showing boundary only (left) and filled (right).

Pad outlines

With the Pad Outlines button (“10”), you can change the look of pads between normal and outline. In the example below, you can see SMD and THT pads in outline mode (left) and normal mode (right):

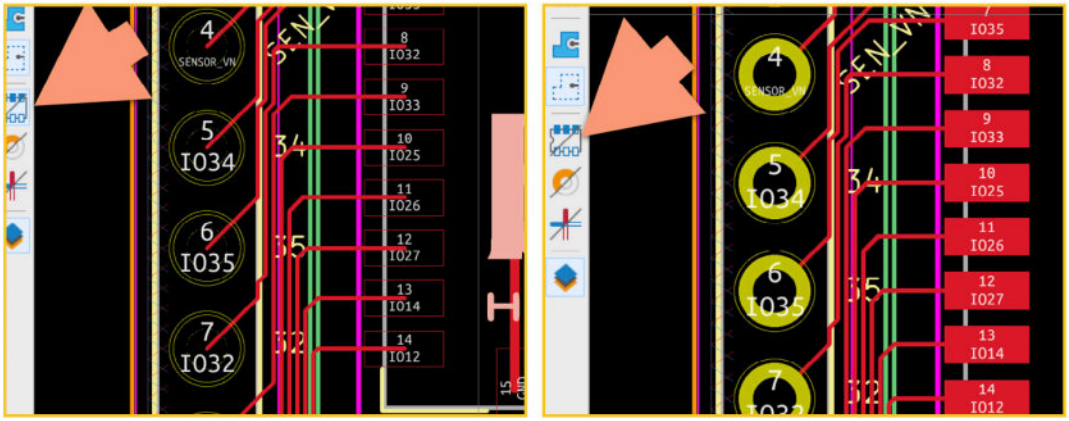


Figure 8.2.10: Pad modes, outline (left), normal (right).

Via outlines

Similar to the two modes for depicting pads, there are two modes for representing vias: outline and normal. You can toggle between them using the button “11”. You can see an example of the two modes below:

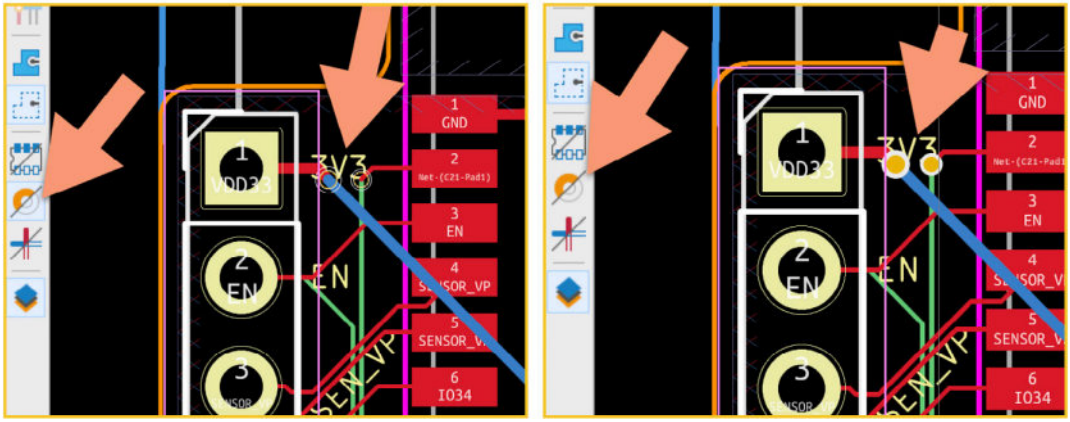


Figure 8.2.11: Via modes, outline (left), normal (right).

Track outlines

You can also toggle the copper track style between normal and outline. You can do this by clicking the button “12”. You can see an example of the two styles below:

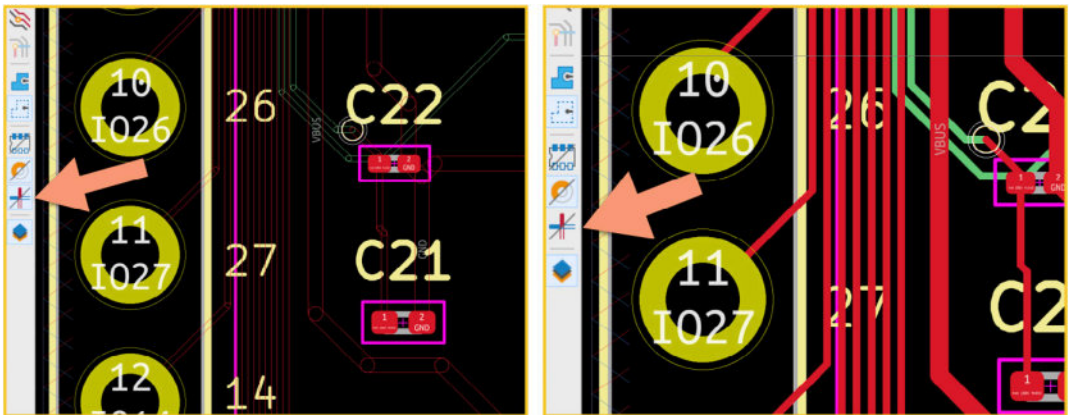


Figure 8.2.12: Track modes, outline (left), normal (right).

Appearance manager

The Appearance Manager is a pane that appears on the right side of the right toolbar. Because this pane takes up a lot of space in the editor, you can choose to hide it by clicking on the button “13”. You will learn about the Appearance Manager in a later chapter.

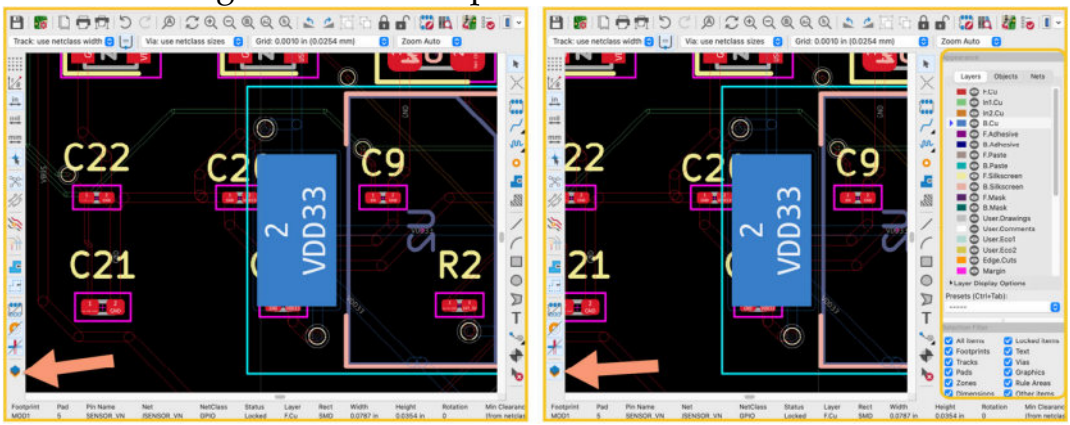


Figure 8.2.13: Appearance Manager hidden (left), showing (right).

3. Top toolbar

In this chapter, you will learn about the buttons and functionalities available in the top menubar. Beware that some of those functions are also available from the top menus and the right toolbar.

In Figures 8.3.1 and 8.3.2 (below), you can see the top toolbar. I have annotated it with numbers to make it easier to refer to them during the walkthrough below. The top menu bar contains two rows with widgets, so I have split it accordingly in this figure.

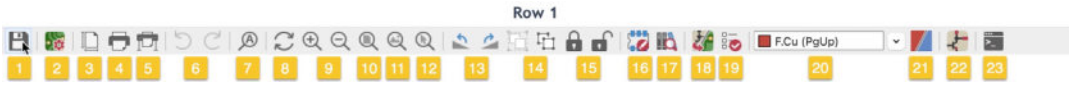


Figure 8.3.1: Top tool bar, row 1.

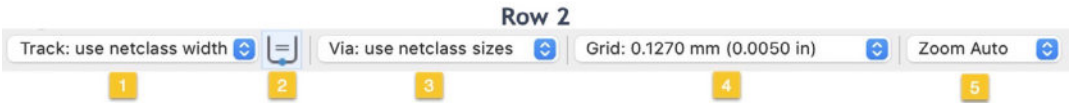


Figure 8.3.2: Top tool bar, row 2.

Let's examine the two rows of the top toolbar.

3.1. Top toolbar Row 1

Please refer to the numbers in Figure 8.3.1 in the walkthrough that follows below.

1: Save

The regular Save button will save your latest work to a file. You can also save your work by clicking on File, Save, or type the Ctr-S/Cmd-S shortcut.

KiCad has an autosave feature. You can configure it via the Preferences window, under Common. In the Session group, you can set the auto-save period and file history size. I have set my KiCad instance to save my work every five minutes.

When the layout editor (or the schematic editor) contains changes that have not been saved to disk, it will display a "*" on the left side of the project name.

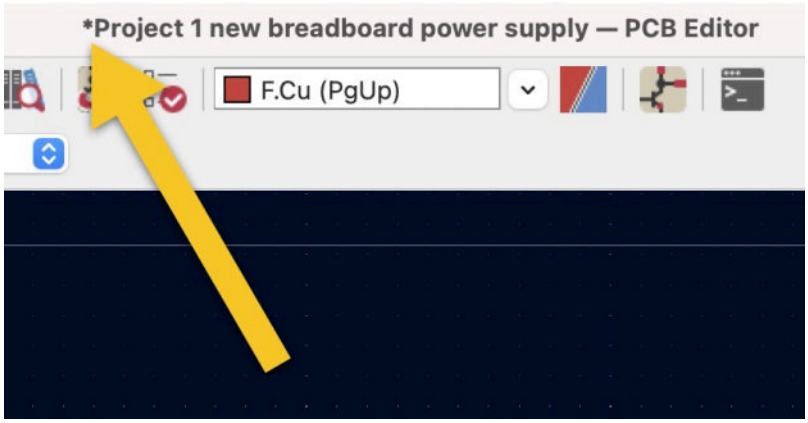


Figure 8.3.1.3: Contains un-saved changes.

Frequent saves are crucial if you are using a cutting-edge nightly build of KiCad. Nightly builds are work-in-progress and may crash at the most inconvenient times. Frequently saving your work, manually or automatically, will protect you from lost work.

2: Board Setup

Click this button to bring up the Board Setup window. You can also access it via File —> Board Setup. You can learn about the Board Setup window in a dedicated chapter later in this part of the book.

3: Page Settings

Click this button to bring up the Page Settings window. You can also access it via File —> Page Settings. In the Page Settings window, you can set the size of the layout editor page and the content of the information tab that appears in the bottom right corner of the layout page. This information is useful when you export the layout as PDF or print it on paper.

Below you can see the Page Settings window and an example of the information tab.

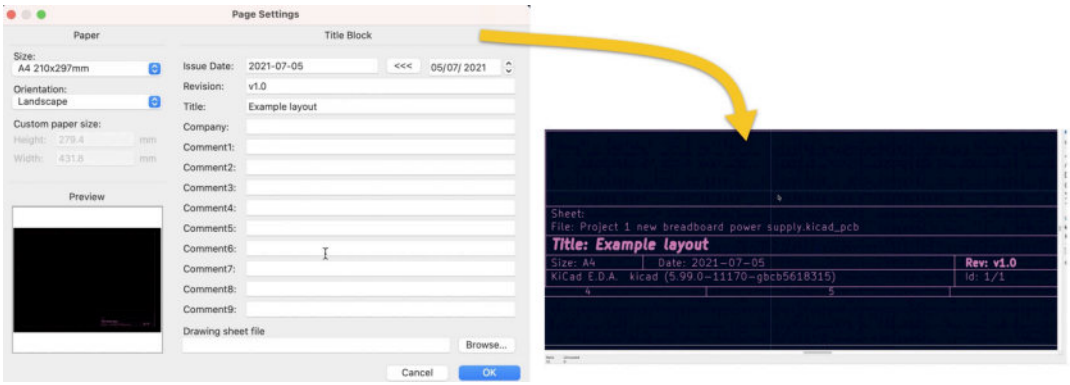


Figure 8.3.1.4: The Page Settings window and the information tab.

4: Print

Click on the Print button to send the layout to a paper printer or export it as a PDF. You can also bring up the Print window via File —> Print.

You can see the Print window below:

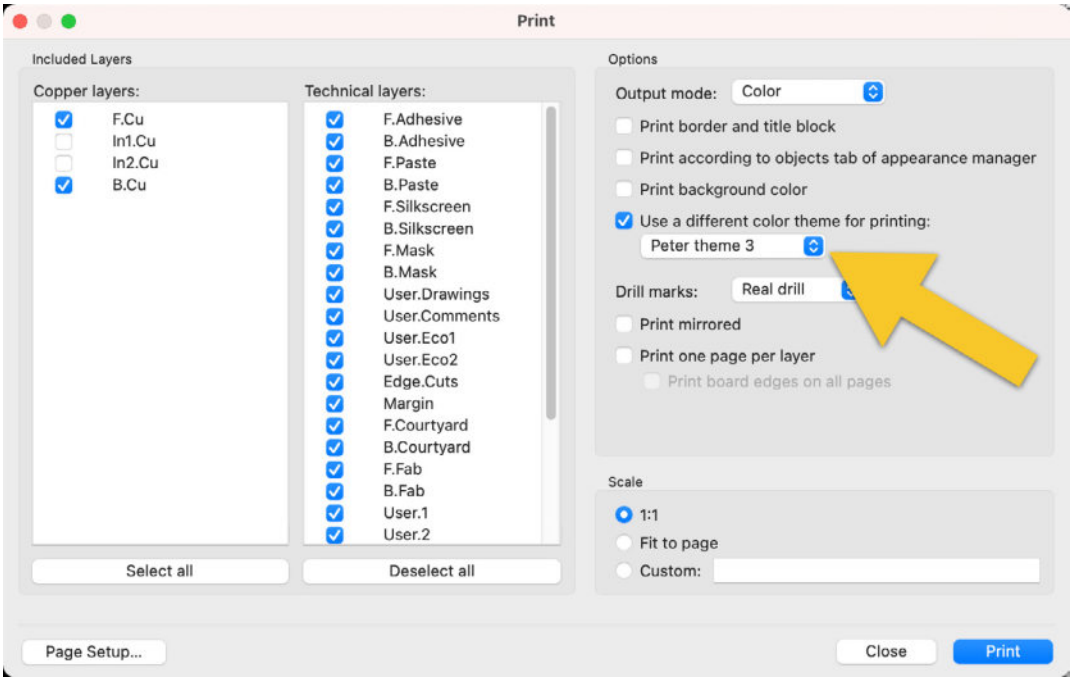


Figure 8.3.1.5: The Print window.

In the Print window, you can select the copper and technical layers that you want to include in the printout. You can configure the printout parameters via the widgets in the Options group. The quickest and easiest way to produce a clean and readable printout is to choose the "Black and White" output mode. The default is "Color" which will preserve the assigned layer colors, but change the background color to white (the screen default is black).

Another option that is available to you (which is my preferred option) is to use a custom "printer-friendly" color theme. I have created this theme specifically for printing. The main difference between this theme and the default theme are the color of the background (white, instead of black), and using darker colors for various text and graphics elements.

5: Plot

Plot is similar to Print but used to export the layout design to a format suitable for manufacturing. You can invoke the Plot window via File —> Plot.

In a dedicated chapter later in this part of the book, you can learn how to use the Plot function to export your PCB as a set of Gerber files suitable for manufacturing.

6: Undo and Redo

KiCad has an unlimited undo buffer. As you make changes to the schematic or layout, KiCad will remember those changes. If you need to undo them, click on the Undo button (the one that rotates anti-clockwise), type Ctr-Z/Cmd-Z, or click Edit —> Undo.

Redo is the opposite of Undo. You may want to use Undo to go back in time and see your layout before the latest change, but then decide that the change is acceptable. Use Redo to return to the newest change, type Shift-Ctr-Z/Shift-Cmd-Z, or click Edit —> Redo.

7: Find

Click on this button to search for text in the layout. You can also invoke this tool with Ctr-F/Cmd-F or via Edit —> Find. In the example below, I have used Find to search for instances of the string "GND." I can iterate through the found instances by clicking on Find Next or Find Previous.

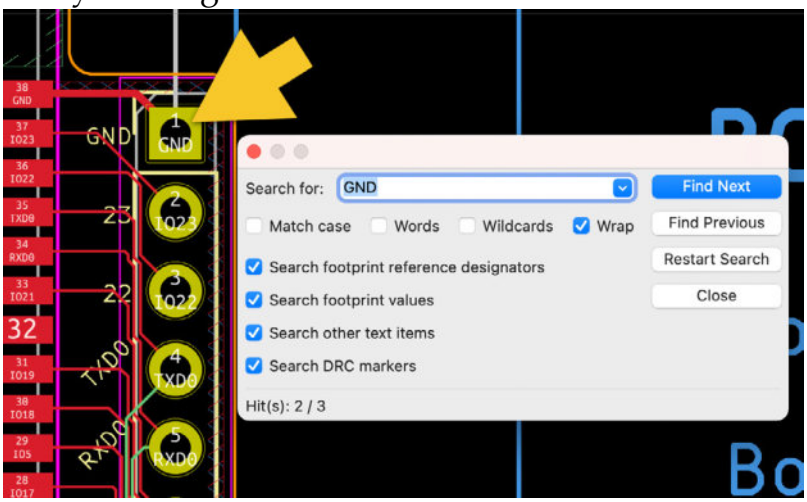


Figure 8.3.1.6: The Search for Text window.

You can narrow the search to specific text types, such as within the group of reference designators or footprint values.

8: Redraw

Click this button to redraw the layout editor window. This can be useful if you notice artifacts leftover by the graphics engine. In the time I have been

using KiCad 6, I have not witnessed such artifacts, so I have not had to use the Redraw function. In earlier versions of KiCad, especially in KiCad 4, the graphics engine would regularly leave "garbage" behind, and the Redraw button provided a way to a clean-up.

9: Zoom

Use the Zoom buttons to zoom in or out. A more convenient way to zoom is to use your mouse's scroll wheel.

The Zoom buttons will zoom in/out in the center portion of the window.

If you use the mouse scroll wheel, zoom in/out will occur at the mouse pointer's position.

10: Zoom to fit

When you click the Zoom to Fit button, the layout editor will zoom appropriately to make all design contents visible within the frame. This includes content in all layers, including the user layers. See the example below, and then contrast it with the Zoom to object function (see section 11 below):

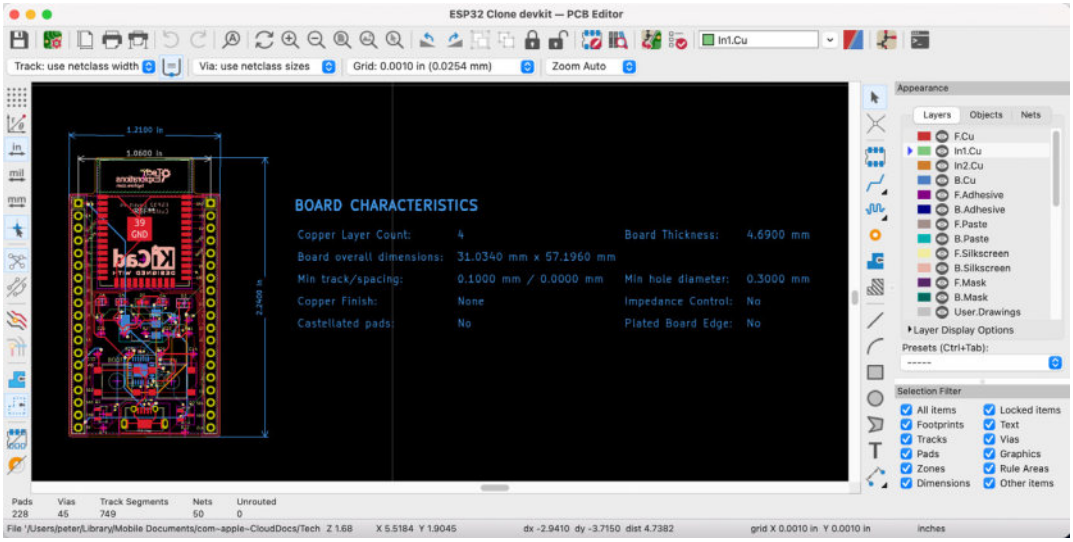


Figure 8.3.1.7: Zoom to fit.

11: Zoom to object

Zoom to Object is similar to Zoom to Fit, except that Zoom to Object will zoom into the content of the manufacturing layers. See the example below of the same PCB, but this time I have clicked on Zoom to Object:

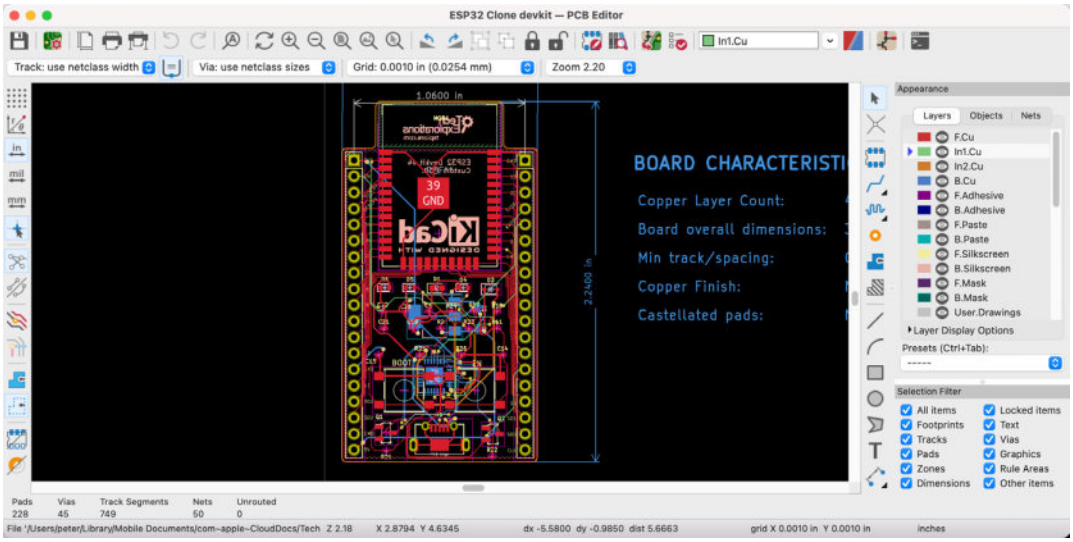


Figure 8.3.1.8: Zoom to Object.

Zoom to Object has zoomed to the appropriate level to ensure the contents in layers such as Edge.Cuts, F.Cu, and F.Silkscreen are entirely within the window frame.

12: Zoom to Selection

In the example below, I have used the "Zoom to Selection" tool to draw a rectangle around a detail of the PCB. With Zoom to Selection, you can use your mouse to draw a rectangular region, which Pcbnew will then zoom in. When I released the mouse button, Pcbnew used up the entire window frame to zoom into the detail I marked with the rectangle.

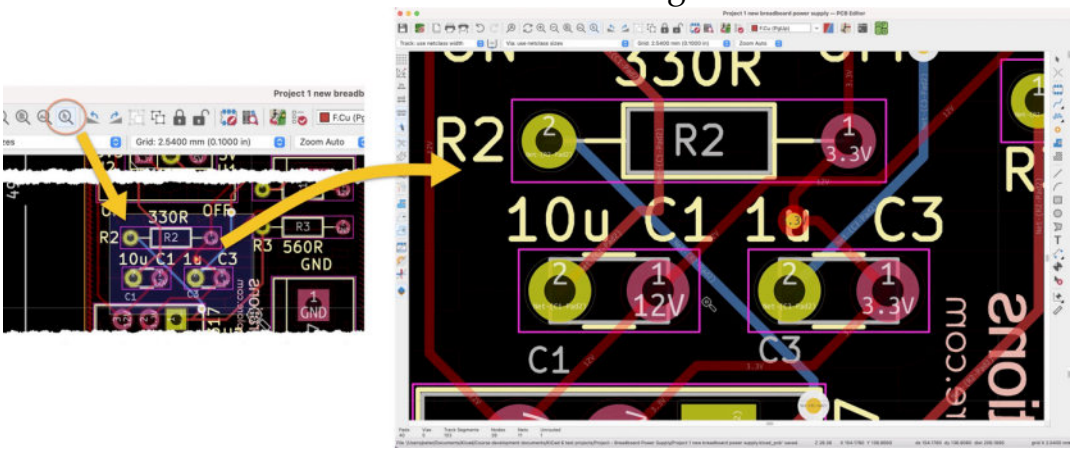


Figure 8.3.1.9: Zoom to Selection.

13: Rotate

Use the rotate buttons to make any selected object rotate clockwise or anti-clockwise at 45-degree steps. You can also use keyboard shortcuts: anti-clockwise is R and clockwise is Shift-R.

The use of keyboard shortcuts is the preferred method of rotating objects in the layout editor. Rotation is one of the most commonly used editing functions as you are positioning footprints in the PCB, so you should make an effort to commit these two shortcuts in your muscle memory.

14: Group and ungroup

You can group any number of elements so that they behave as one element. For example, I can create a group containing the two screw terminals from my power supply PCB (see figure below). Once I have created the group, I can move both footprints simultaneously, maintaining their relative positions.

To create a group, multiple-select (hold down the Shift key and click) the items you want to include (footprints, silkscreen text or graphics, etc.). Then right-click to show the context menu and select Grouping —> Group. Alternatively, use the Grouping button from the top toolbar.

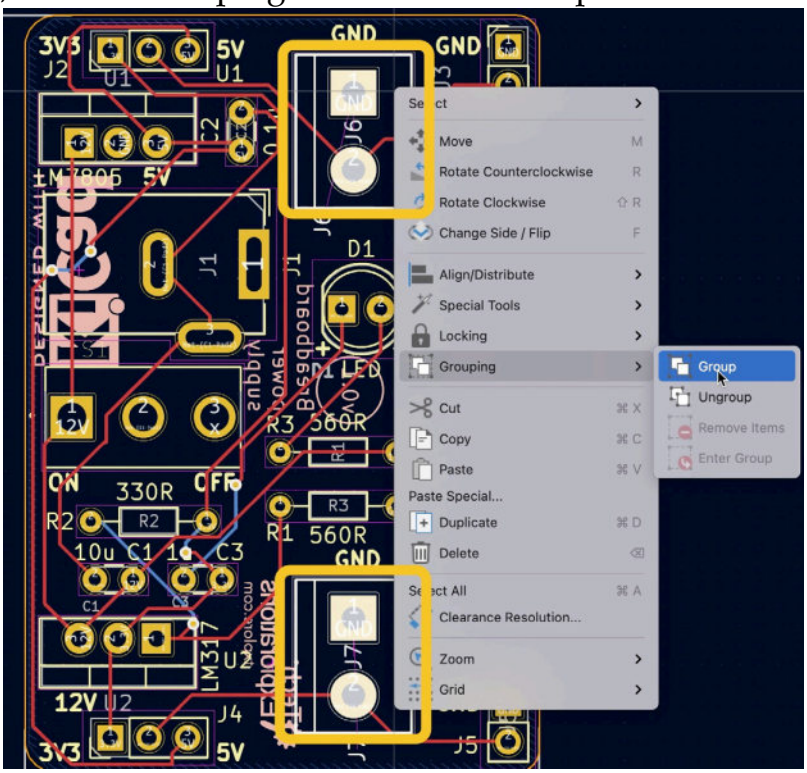


Figure 8.3.1.10: Creating a group.

A group is depicted with a rectangle around the group members. You can move the group to a new location by clicking on any group member and typing "M" (the Move hotkey):

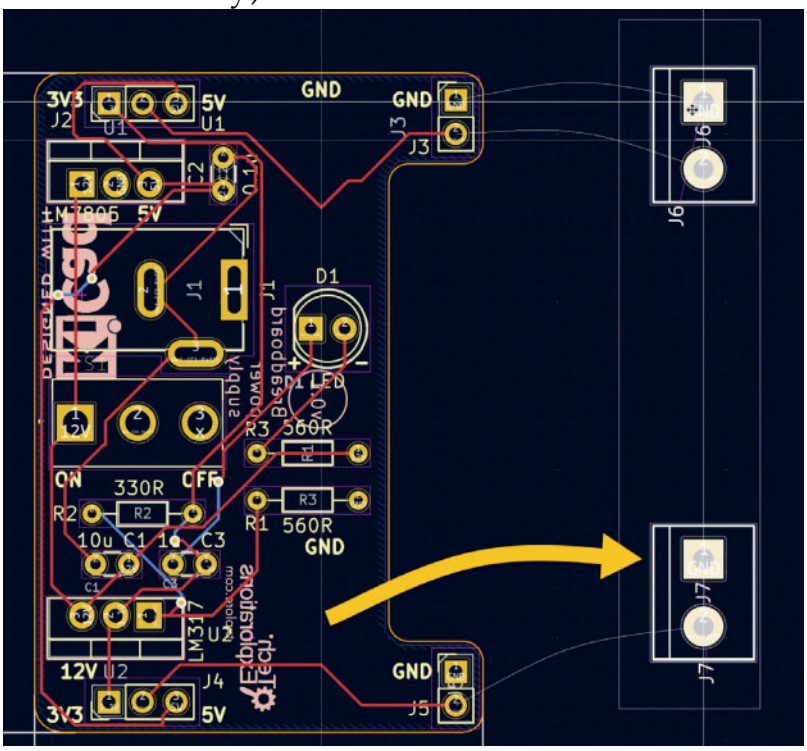


Figure 8.3.1.11: Moving a group.

You can then move the entire group as if it is a single element.

15: Lock and Unlock

You can use these two buttons to lock an element in place. You can use the unlock button to reverse locking. Conveniently, the keyboard shortcut for locking and unlocking is L (toggle). You can also right-click on an item to bring up its context menu and select Lock/Unlock from the Locking submenu.

You can consider locking footprints with an essential position on the board, such as connectors or mounting holes. These elements may have a position that should not change to fit with external components, such as a power supply or projections from an enclosure.

To lock/unlock an element, select it with the mouse and type L (or use the context menu). A locked element will show its "locked" status in the status bar:

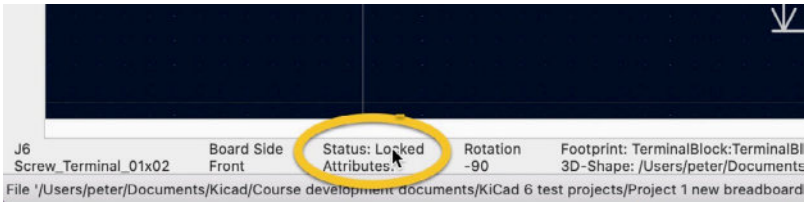


Figure 8.3.1.12: This footprint is locked.

If you try to move a locked element, you will see a pop-up warning window. This window will give you the option to override the lock and continue to move the element:

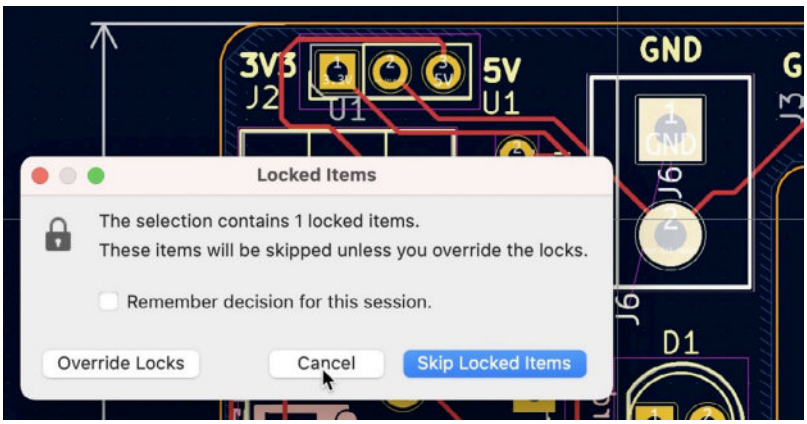


Figure 8.3.1.13: Trying to move a locked footprint.

16: Footprint editor

You can use the footprint editor to edit and create footprints. This is an essential tool in the KiCad toolset, and you can learn how to use it to make footprints in the dedicated chapter later in this part of the book.

17: Footprint Library Browser

Use the footprint library browser to browse the installed footprints, find what you need, and add a footprint to the editor. You can see the footprint library browser below:

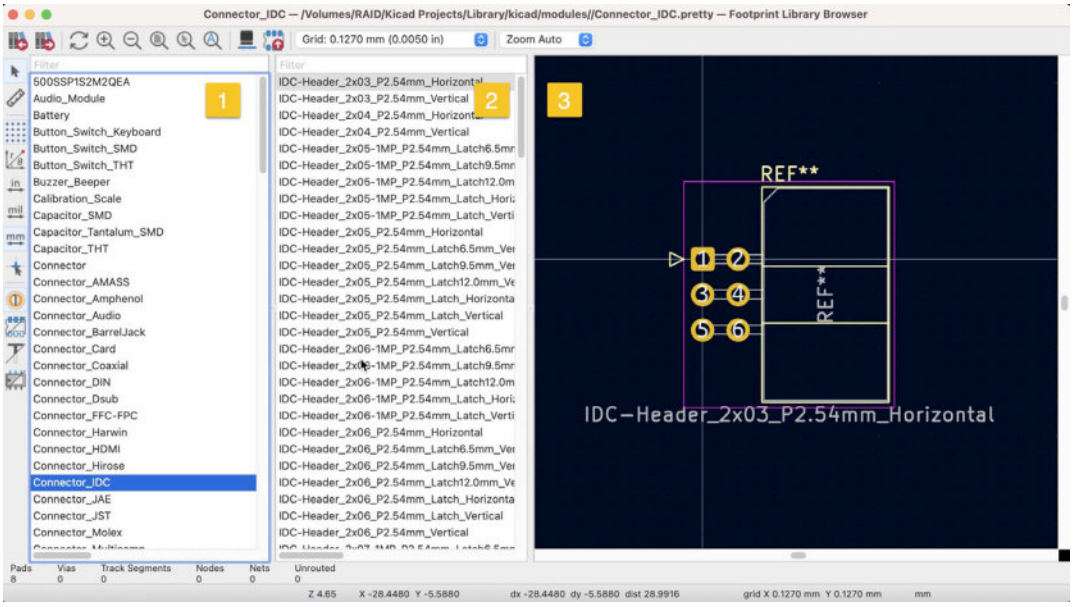


Figure 8.3.1.14: The Footprint Library Browser.

The Footprint Library Browser is a scaled-up version of the Footprint Chooser that you can access from the right toolbar. The Footprint Library Browser has three main panes, in addition to the top and left toolbar:

1. List of libraries. Select one to show its contained footprints in pane 2.
2. List of footprints. It lists the footprints contained in the selected library in pane 1.
3. Selected footprint preview.

To speed up your search, you can use the filter text box at the top of panes 1 and 2.

The top and left toolbar buttons work as their counterparts in Pcbnew.

18: Update PCB with changes made to schematic

PCB design is an iterative process. For all but most trivial designs, you will find yourself switching between Pcbnew and Eeschema. When you change the schematic in Eeschema, you can use the Update PCB button to import those changes into Pcbnew.

In the example below, I have changed the value of a footprint in Eeschema and saved the change. Then, in Pcbnew, I click on the Update PCB button to bring up the import tool window:

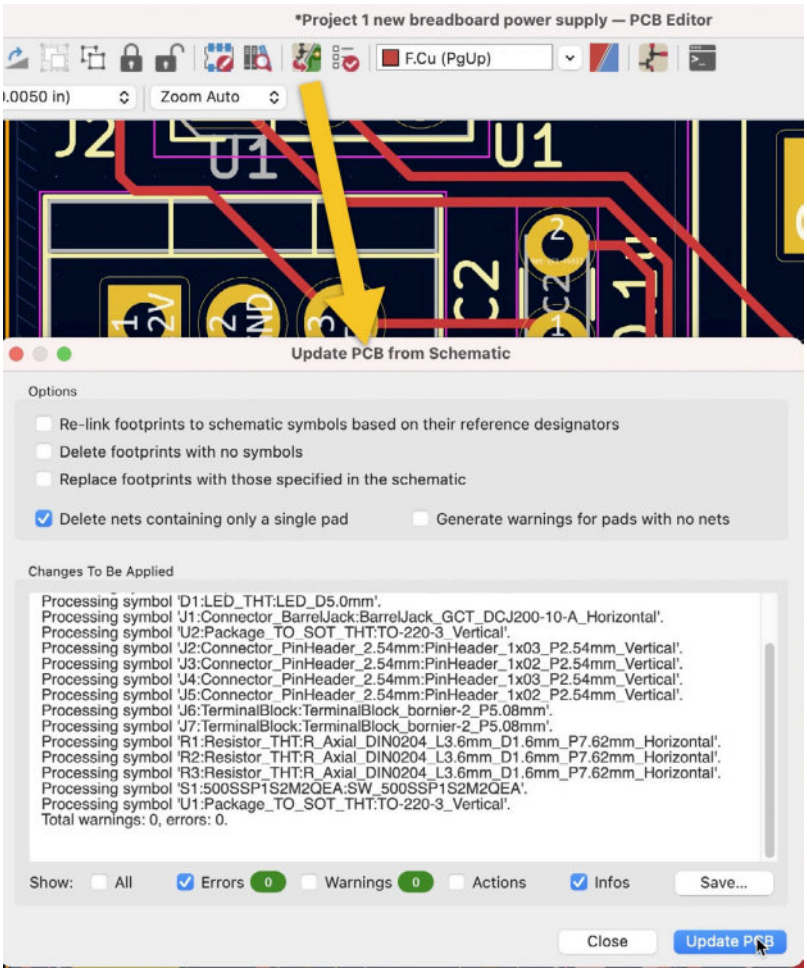


Figure 8.3.1.15: Importing schematic changes to the layout editor.

In the Update PCB window, you can enable the options you wish from the Options group. The change I made only involved a value text field, so I have chosen not to evaluate re-linking, deleting, or replacing footprints. Click on "Update PCB" to import the changes from the schematic editor. The changes are listed in the text area to know what is about to happen to your layout.

19: Design Rules Checker

The Design Rules Checker (DRC) is a tool that checks your PCB for violations (errors). Click on the DRC button to bring up the DRC window, and click on "Run DRC." When the DRC is complete, the window looks like this:

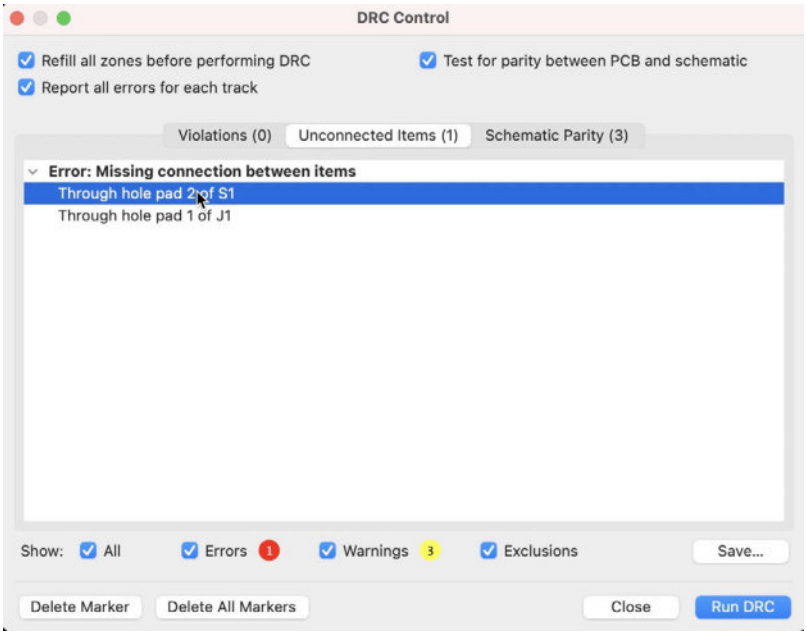


Figure 8.3.1.16: The DRC showing one error.

The DRC found one error in my example above. The DRC is configurable to choose how you would like it to classify and report violations. You can learn the details in a dedicated chapter later in this part of the book.

20: Layers chooser

The layers chooser provides a subset of the functionality of the Layers tab in the right toolbar. It allows you to enable a layer in the PCB layers stack.

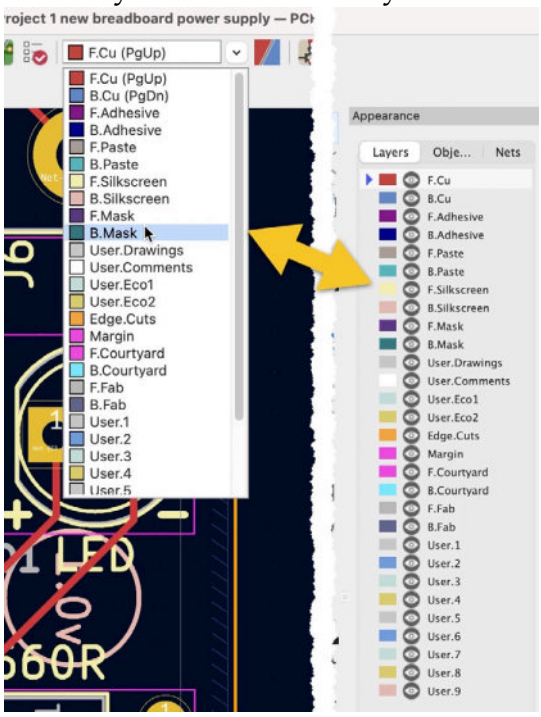


Figure 8.3.1.17: Two places to choose a working layer.

The layer chooser in the top toolbar is convenient when I have removed the Appearance pane from the right toolbar to increase my working space in the editor.

21: Change active layer pair for routing

This button allows you to choose two layers that you can switch to quickly using the V hotkey during routing. To set up your copper layer pair, click on the button to bring up the pair window and choose the two layers. If you are working on a two copper layer board, you will not have a choice, and the pair would be F.CU and B.CU. You can choose any combination of layer pairs for a board with four layers or more (like in the example below). Once you have configured the pair, you can use the tracking tool to draw a copper track and use the V hotkey to switch between the layers in the pair.

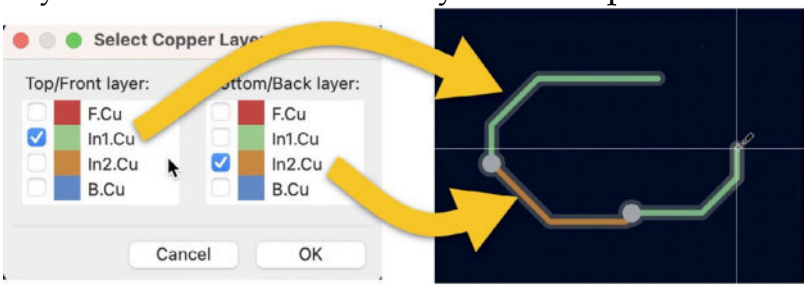


Figure 8.3.1.18: Copper layer pair.

Once you have selected the copper layers in the pair, the pair button in the top toolbar will use the colors of the two layers to see the members of the pair without having to open the pairs window.

22. Eeschema shortcut

You can switch to the schematic editor from Pcbnew by clicking on this button.

23: Python scripting console

KiCad 6 has a Python API that allows end-users to extend its functionality using the Python programming language. When you click on the Python scripting console button, you will see the KiPython window that will enable you to interact with the API.

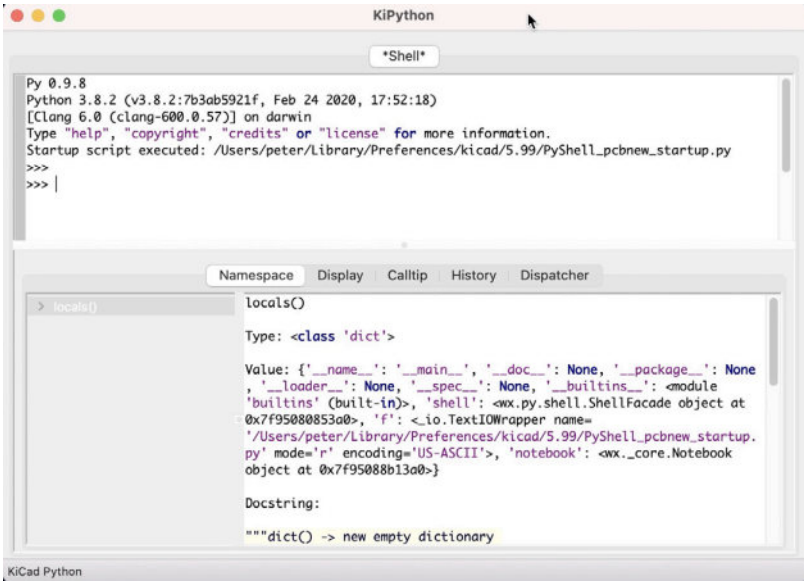


Figure 8.3.1.19: The KiCad Python scripting console.

When I write these lines, the KiCad Python API is still under active development and mostly undocumented. I will update this book with more information once this feature becomes stable.

3.2. Top toolbar Row 2

In this section, you will learn about the widget and functions in the second row of the top toolbar. Please refer to the annotated Figure 8.3.2 for the numbers I use below to mark each widget.

1: Track width

Use this dropdown to select the copper track width. You can set the width to be controlled by the net class settings (learn more about this in a later chapter), or choose a pre-set width and ignore the net class settings.

You can add custom width sizes by clicking on the "Edit Pre-defined Sizes" option. This will open the Board Setup window at the Pre-defined Sizes tab, where you can add custom sizes in the Tracks column.

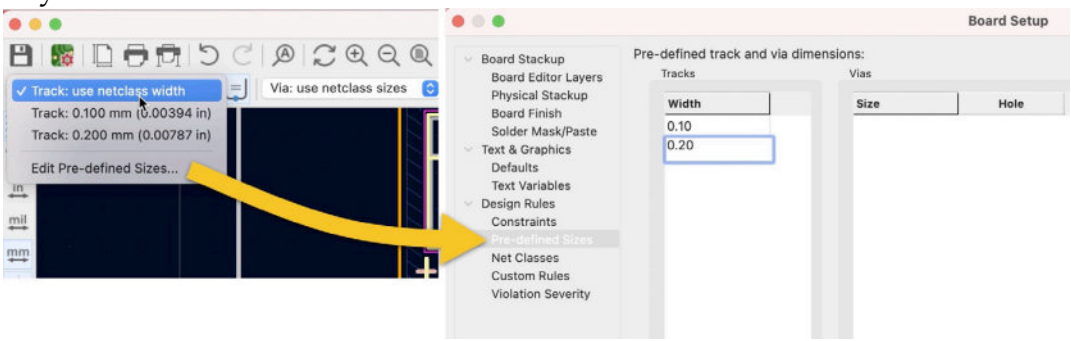


Figure 8.3.2.20: Copper track width dropdown.

2: Existing track width

The existing track width button ensures that as you draw a copper track and add new segments to it (i.e., by switching to a different layer with a via or adding a new segment at a later time), the width of the track remains equal to the width of the original segment.

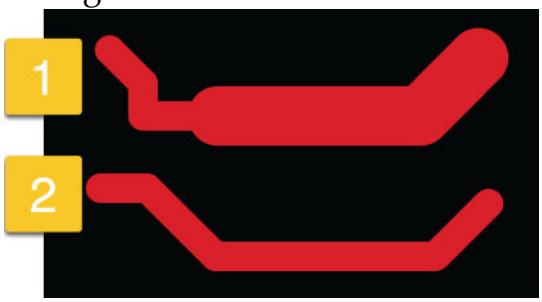


Figure 8.3.2.21: Keeping existing track width in “2”.

Consider the example of the two tracks in the figure above.

In "1", I have turned off the Existing Track Width button. I started drawing a track with a width of 0.1 mm. Then, I ended the drawing (double-click) and changed the track width to 0.2 mm. I continue the drawing to create a new segment. Because the existing track width option was not active, the second segment of the same track has a different width to the first segment.

In "2", and followed the same process as I described in the paragraph above, except that I enabled the Existing Track Width option. Even though I selected 0.2 mm width for the second segment, the existing track width option ensured that the second segment width was equal to the first segment.

3: Via Size

This dropdown works similarly to the track width dropdown menu and controls vias. You can choose a custom via size or allow automatic selection based on the net class to which the via belongs. You can set custom via sizes in the Board Setup window:

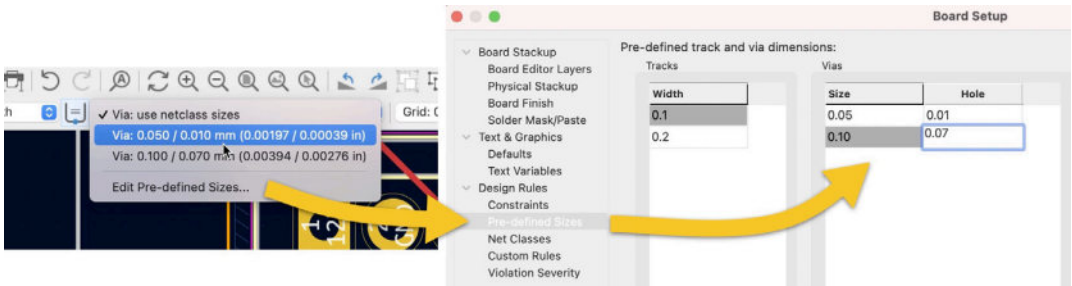


Figure 8.3.2.22: Via size dropdown.

4: Grid Size

Use the Grid Size dropdown to select a grid size. To be able to see the grid, ensure that the Grid is enabled. You can enable and disable the grid using the grid button in the right toolbar.

5: Zoom

You can select a zoom level using the zoom dropdown button. In most cases, you will control the zoom level using the scroll wheel of your mouse. When you do that, you engage the Zoom Auto mode. Side note: I have never needed to use any of the other options.

4. Right toolbar

In this chapter, you will learn about the functionalities of the right toolbar. This toolbar consists of the main button toolbar, the Appearance pane, and the Selection filter. Because the Appearance pane and the Selection Filter take up considerable space, you can hide them using the button at the bottom of the left toolbar.

To help identify the various widgets in all areas of the right toolbar in the discussion that follows, please see the annotated figures below.

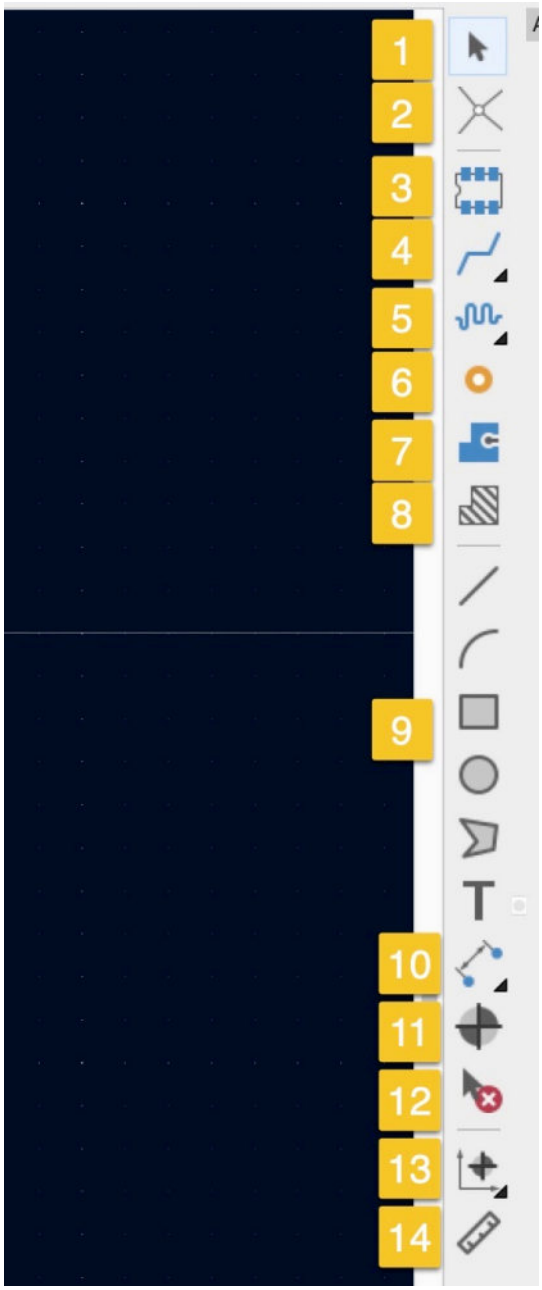


Figure 8.4.1: The main part of the right toolbar.

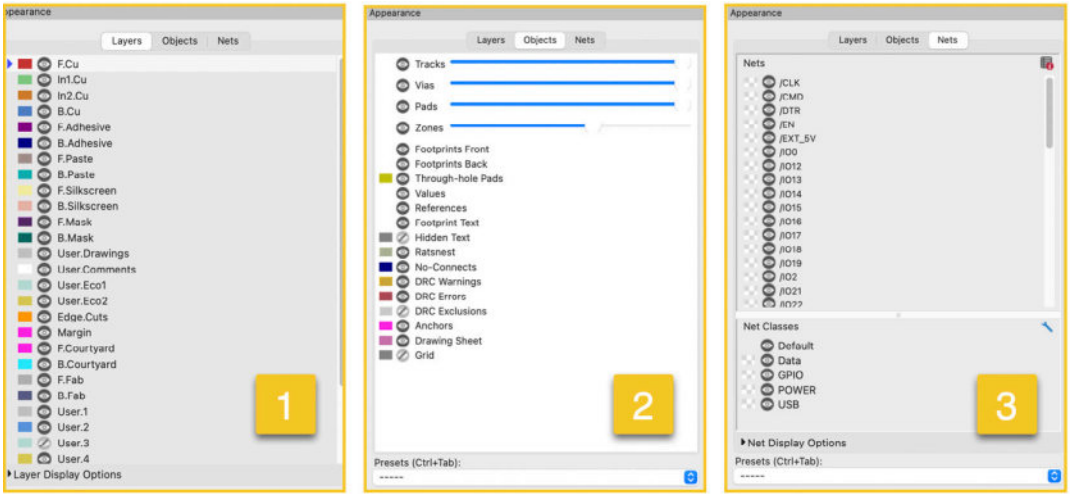


Figure 8.4.2: The Appearance pane.

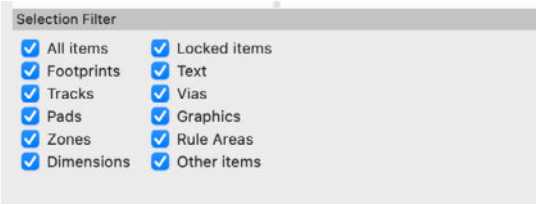


Figure 8.4.3: The Selection Filter.

Let's look at these three parts of the right toolbar.

4.1. Right toolbar main buttons

Please refer to the numbers in Figure 8.4.1 in the walkthrough that follows below.

1: The pointer

The pointer is the default tool selected when no other tool is active. The pointer allows you to click and choose an element in the editor.

2: Toggle ratsnest lines

Click this button to toggle show/hide the ratsnest lines. Those lines show the electrically connected pads but don't yet have a copper track drawn and are not connected via a copper fill.

3: Add a footprint

Click this button to bring up the footprint chooser. Use the footprint chooser to find a footprint and add it to the editor quickly. You can see the footprint chooser window below:

Figure 8.4.1.4: The footprint chooser window.

The footprint chooser is a simplified version of the footprint library browser that you learned about earlier (available through the top toolbar of Pcbnew).

4: Track and differential track

This is a multi-button. Click and hold to reveal the two buttons that are bundled together.

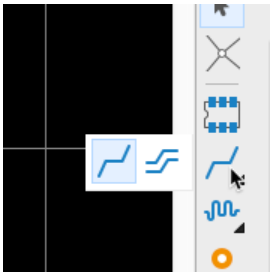


Figure 8.4.1.5: The track and differential track buttons.

The single-line tool allows you to draw a single track. The dual-line tool will enable you to draw a differential pair track.

To learn about differential pairs, please read the dedicated chapter in the Recipes part of the book.

5: Length tuner

This tool allows you to tune the length of a single or differential pair track. You can use it to set the track length to a specific length of your choice. As with the track button, the length tuner button contains a set of buttons. Click

and hold to reveal the contents of the multi-button, and click again to select the one you want to use.

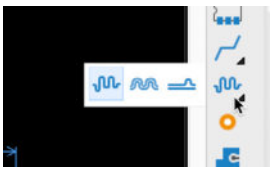


Figure 8.4.1.6: The track length tuner buttons.

To learn more about track length tuning, please read the dedicated chapter in the Recipes part of the book.

6: Add free-standing vias

This button allows you to create free-standing vias. These are vias that are not connected to a track when you create them. You can route tracks to them later and configure their type (through, micro, or blind) and sizes via their properties window.

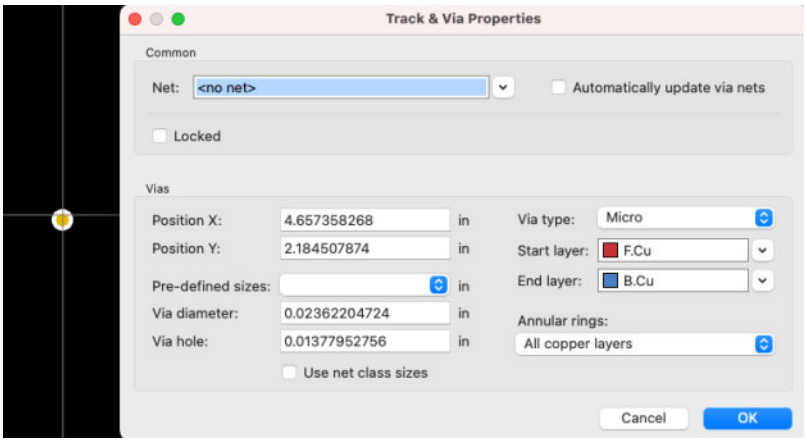


Figure 8.4.1.7: A free-standing via and its properties window.

7: Add a filled zone

Use this button to create a zone filled with copper in one or more copper layers. You can learn more about filled zones in a later chapter in this part of the book.

8: Keep out zones

Use this button to create a keep-out zone in one or more copper layers. You can learn more about keep-out zones in a later chapter in this part of the book.

9: Graphics and text

Use these buttons to create graphics and add text in any layer. These graphics and text can be necessary for the manufacturing of the PCB, such as those that exist in the Edge.Cuts layer. They may also be informational and decorative, such as those existing in the copper or silkscreen layers.

In the example below, I have used the widgets in the Appearance Objects tab to tone down all elements except the graphics and text. This example showcases the value that graphics and text can add to a PCB.

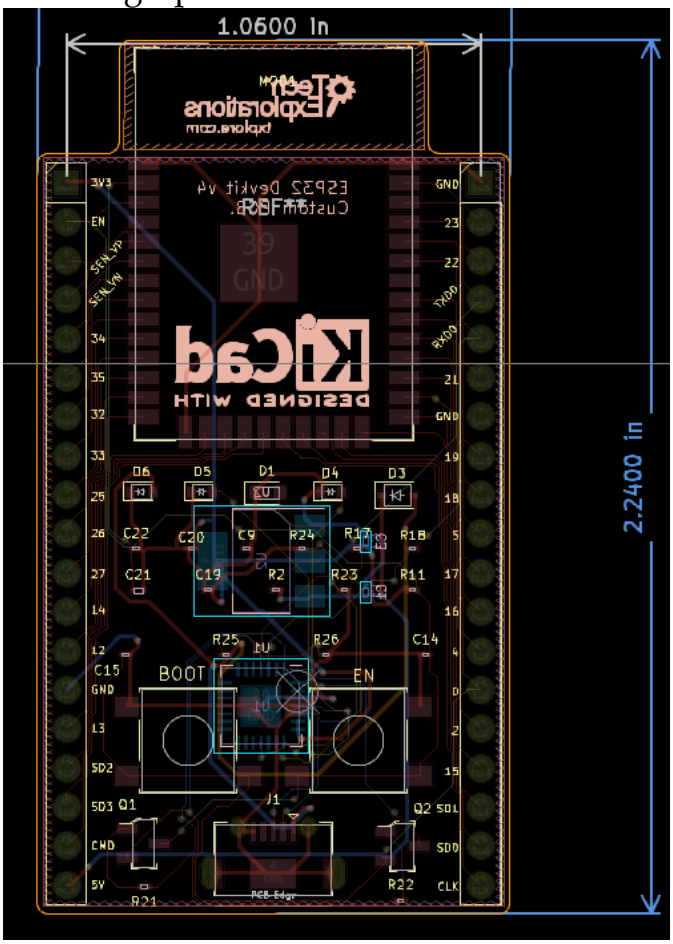


Figure 8.4.1.8: Highlighting graphics and text.

10: Measuring tools

This is a measuring multi-tool. Use it to measure distances between any two points in the layout editor.

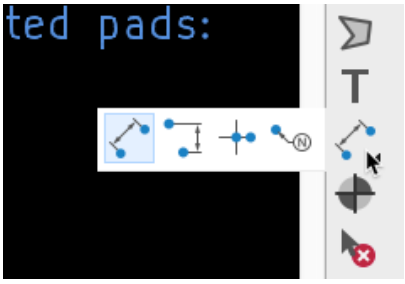


Figure 8.4.1.9: Length measuring tools.

You can learn more about the measuring tools in a later chapter in this part of the book.

11: Layer alignment target

A layer alignment target is an object that exists across all layers and helps the manufacturer to precisely align the layers. In the example below, I have added an alignment target near my PCB.

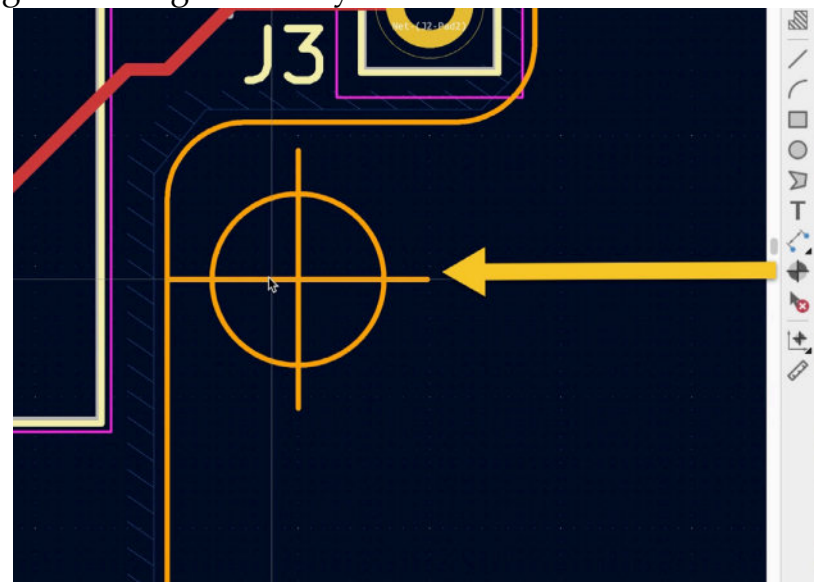


Figure 8.4.1.10: An alignment target.

Even though this object is available, I never had to use it with an online manufacturer in any of my recent (last ten years) orders. Modern manufacturers can align your PCB layers using the information in the Gerber files.

Still, your manufacturer may require that an alignment target is present in your design, so take care to ask them if in doubt.

12: Delete clicked item

The interactive delete tool allows you to delete an element in the editor with a click. Select the tool and place it over the element you wish to delete. The element (such as a track or a footprint) will change color to indicate that it will be deleted if you click.

To delete a highlighted element, right-click.

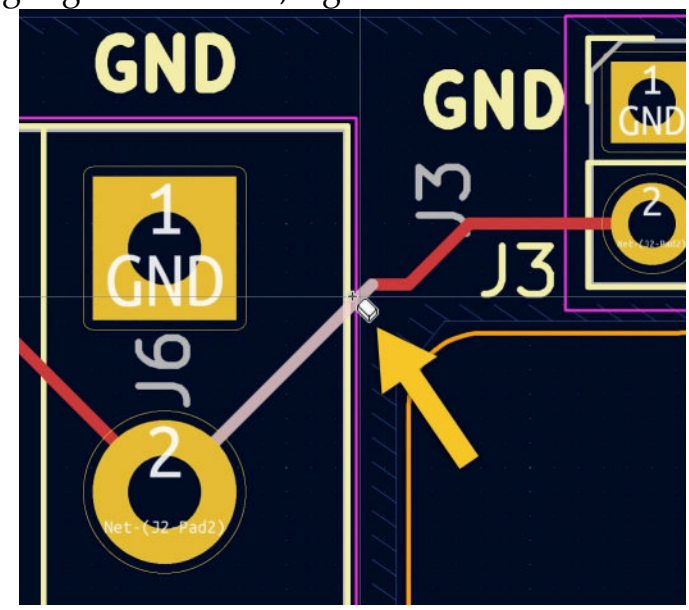


Figure 8.4.1.11: About to delete this track segment.

Made a mistake? Use the Undo tool.

13: Set origins

Pcbnew allows you to configure the coordinate system of the editor. One of the configuration options is to change the coordinate system origin. This button will enable you to set the grid origin point anywhere in the editor ("2", below). It also allows you to set a special origin point for the drill files ("1", below).

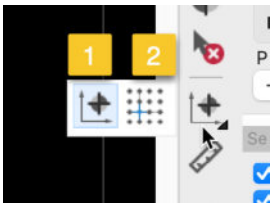


Figure 8.4.1.12: Coordinate origins.

You can learn more about the origins in a later chapter in the Recipes part of the book.

14: Interactive ruler

You can use the interactive ruler to make quick distance measurements between any two points of the editor space. You can see an example below:

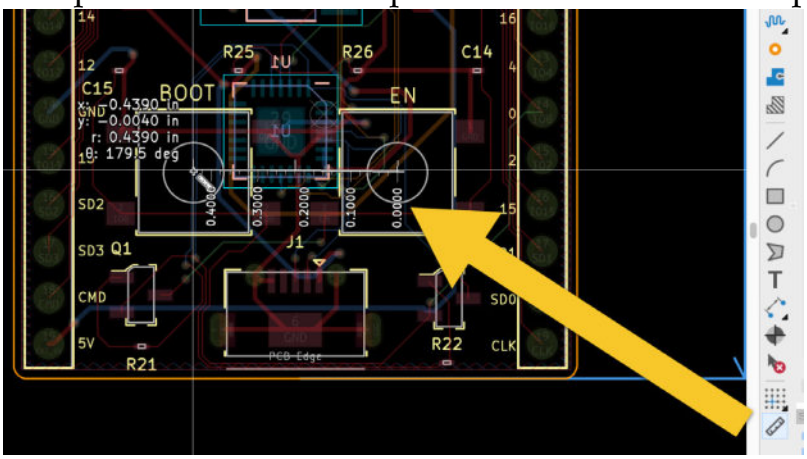


Figure 8.4.1.13: The interactive rule.

You can learn more about measuring distances in the layout editor in a dedicated chapter later in this part of the book.

4.2. Right toolbar - Appearance

Let's dive into the Appearance pane. The Appearance pane consists of three tabs at the top, Layers, Objects, and Nets, and the Presets and Selection Filter at the bottom.

Layers

The Layers pane allows you to select the active layer, the layer visibility, and the color. At the bottom part of the tab is the Layer Display Options box that controls the visibility of inactive layers.

An active layer is a layer that you are currently working on. A new track, or a new text element, for example, will be drawn in the active layer. A triangular indicator indicates an active layer.

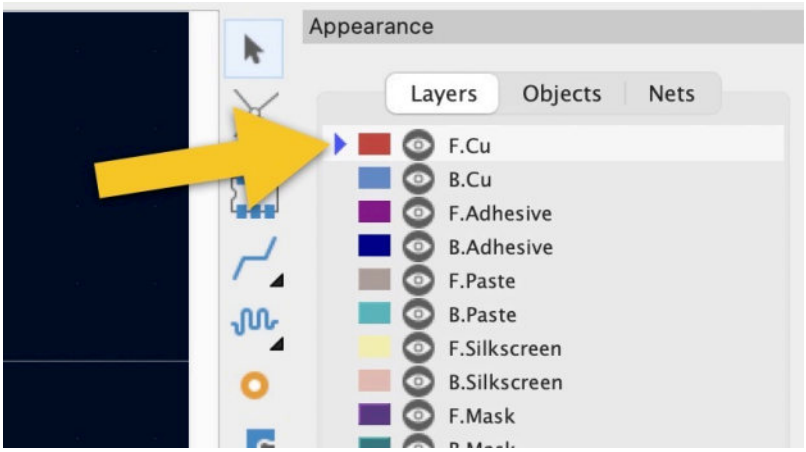


Figure 8.4.2.14: It's active.

You can hide a layer by clicking on the eye button next to the layer. This is a toggle button, so you can click it again to unhide a layer.

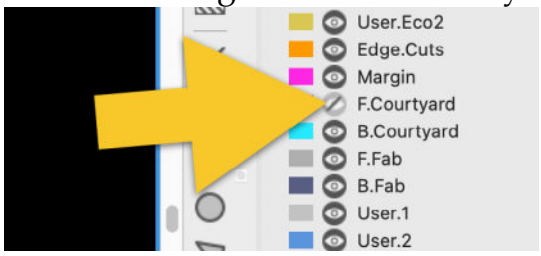


Figure 8.4.2.15: The F.Courtyard layer is hidden.

KiCad's main color theme is read-only, however you can create custom themes where you can freely pick and choose colors. In the example below, I am editing the color of the In1.Cu layer by double-clicking on its color box to reveal the color chooser window:

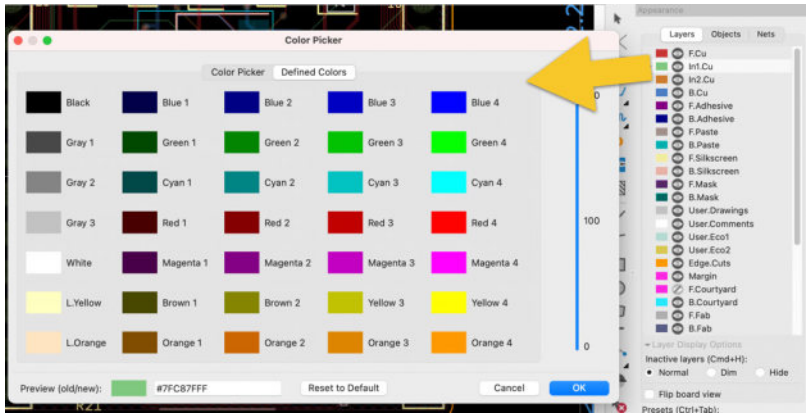


Figure 8.4.2.16: Changing the color to the In1.Cu layer.

For more information on how to create a custom color theme, see "Colors" in the next chapter ("5. Layout editor preferences").

Layer Display Options

At the bottom of the Layers tab in the Layer Display Options "thingy." This gives you three options for how to display inactive layers. You can either have the editor ignore inactive layers status (and display their contents normally), dim them, or hide them. You can see what these three options look like below:

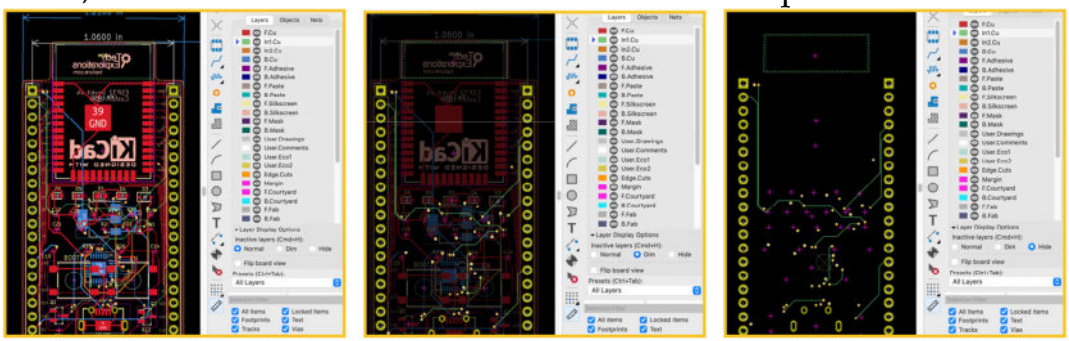


Figure 8.4.2.17: Inactive layer display, normal (left), dimmed (middle), hide (right).

In the example above, I have enabled In1.Cu.

Objects

In the Object tab, you can further control how tracks, vias, pads, zones, and other elements appear in the editor. You can use the scrollers at the top to control the opacity of tracks, vias, pads, and zones. You can also show/hide elements that may appear across several layers, such as values and references.

In the example below, I have reduced the opacity of vias:

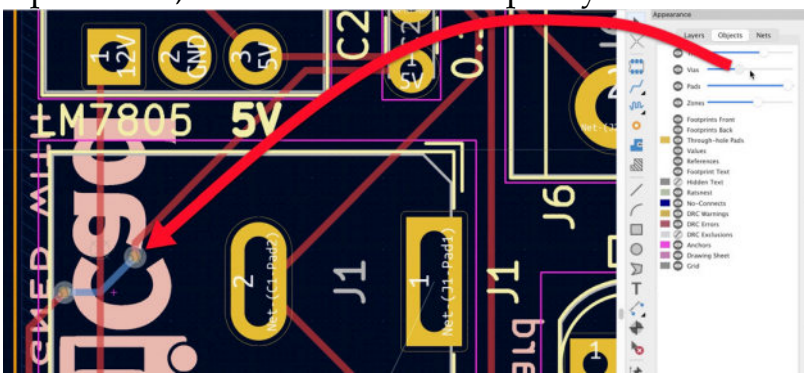


Figure 8.4.2.18: Reduced opacity for vias.

Nets

In this tab, you can control the color and visibility of nets and net classes.

In the layout editor, nets are visualized before your draw the copper tracks as ratsnest lines.

You can set colors for the ratsnest lines in the Nets tab or toggle them visible/hidden.

In the example below, I have deleted one of the tracks in this PCB to see its ratsnest line. I have customized the color (purple) of the line for the specific net to which this line belongs ("12V") so that its stands out from the rest:

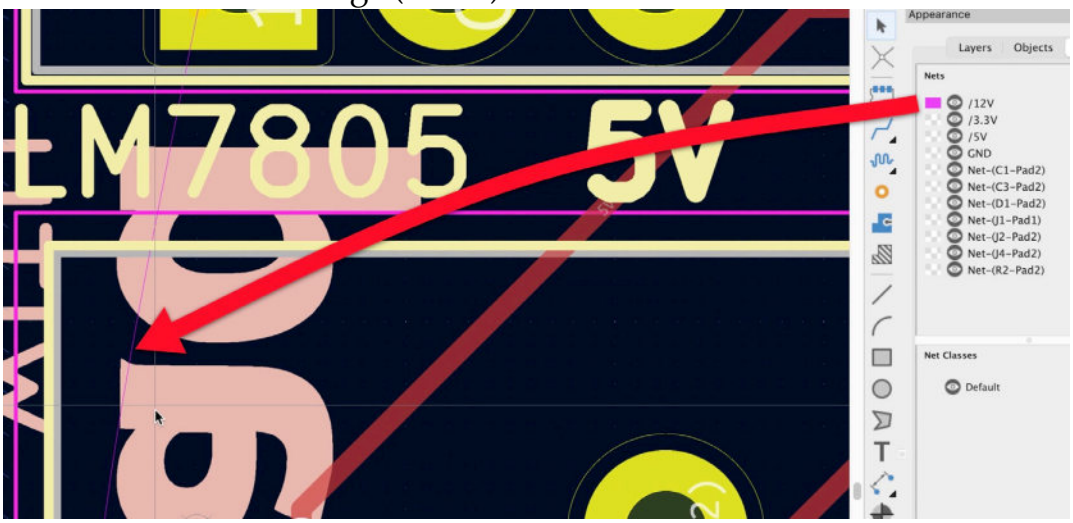


Figure 8.4.2.19: Set the color of the 12V net to purple.

The options in the Net tab are particularly useful when you are starting the layout of a new PCB as it can help you work with a dense network of ratsnest lines.

Presets

You can save your settings in the Appearance pane as a preset. Click on the drop-down menu to expand it, and select "Save preset…".

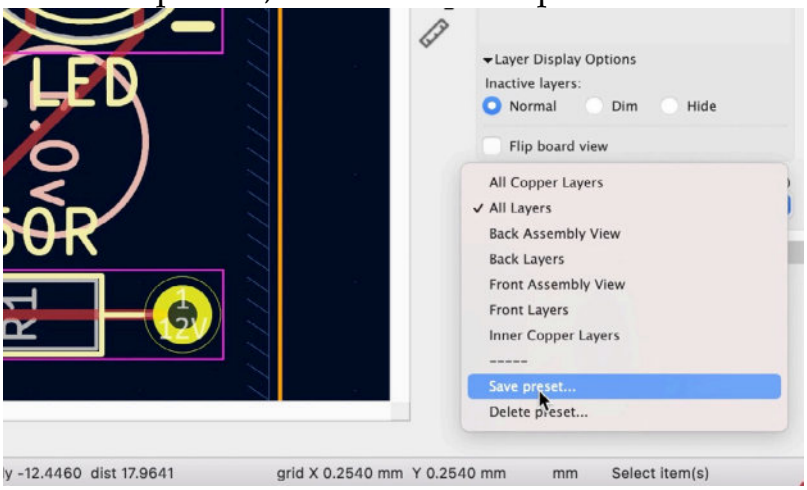


Figure 8.4.2.20: Create an Appearance preset.

You can then invoke your presets via the same drop-down menu.

Selection Filter

The Selection Filter allows you to choose which elements in the editor can be selected when you click. This is useful in busy layouts with elements overlapping other elements making it difficult to choose the one you want.

For example, if you want to work specifically with tracks, you can uncheck all items in the Selection Filter except for "Tracks". This way, when you click on a location where a track and a footprint or graphic overlap, only the track will be selected.

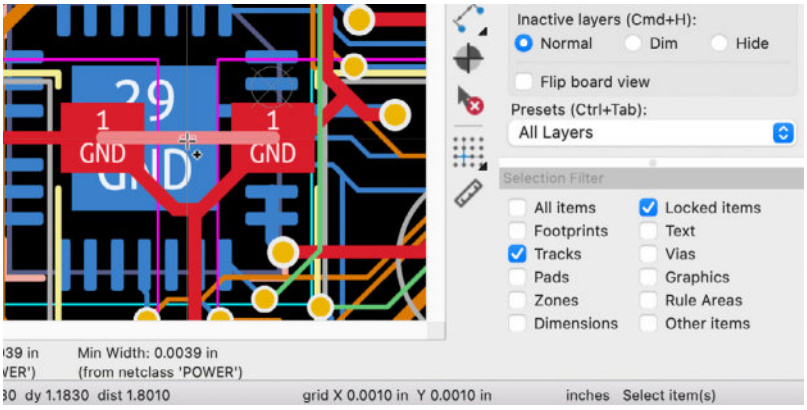


Figure 8.4.2.21: Tracks only.

In the example above, I have checked the Tracks options in the selection filter. As a result, I selected a track segment that overlaps with a footprint with a single click and no risk of choosing the footprint.

5. Layout editor preferences

In this chapter, you will learn about the configuration options available in the PCB Editor group of the KiCad Preferences window. There are tabs in this group:

1. Display Options.
2. Editing Options
3. Colors.
4. Action Plugins.
5. Origins & Axes.

Let's look at the most important options for each one.

1. Display options

Below you can see the Display Options tab:

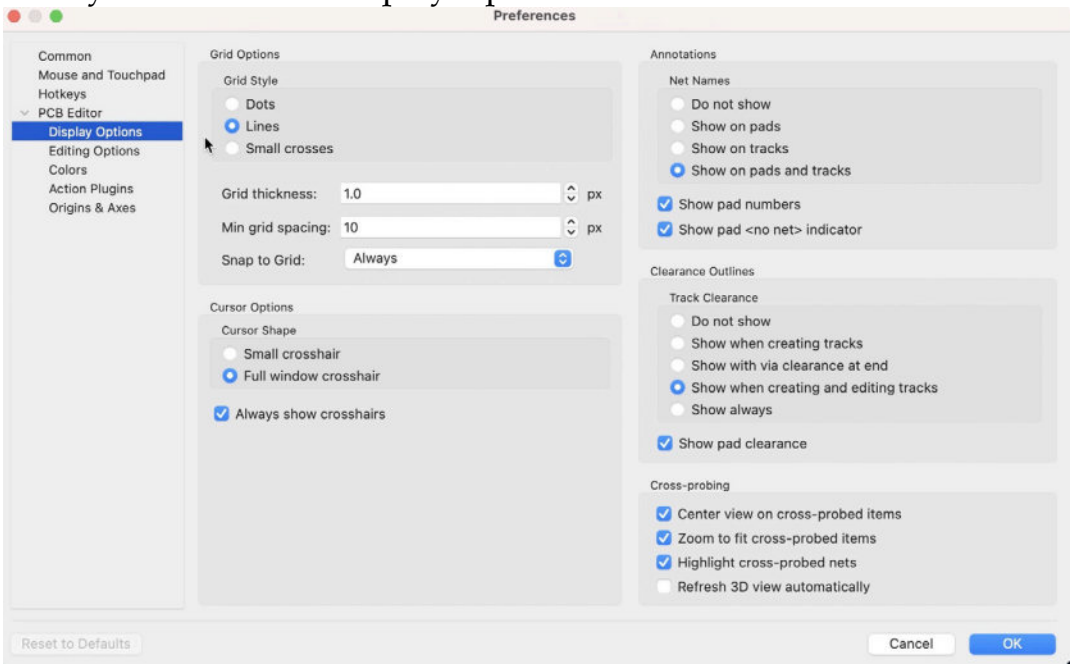


Figure 8.5.1: Pcbnew Display Options.

There are three grid style options: Dots, lines, and small crosses. I find that lines at 1px thickness look best.

The "snap to grid" option can help you pace elements and draw tracks accurately for the points on the grid where horizontal and vertical lines meet. I keep this option always enabled.

The cursor style can change between small crosshairs and full window crosshairs. I find that full window crosshairs help align elements that are not close to each other, as the long crosshair lines help to judge relative positions better.

The options on the right side of the Display Options tab are a matter of personal preference, and I don't have any suggestions to make. I typically leave them in their default settings unless I am testing them.

For information on the options under the Cross-probing group, please go to the earlier chapter on Eeschema editor preferences. The cross-probing feature works across the layout and schematic editors.

2. Editing Options

In the editing options tab, you can configure the behavior of the left mouse click, how snapping works, how ratsnest are drawn, and more. You can see the editing options tab below:

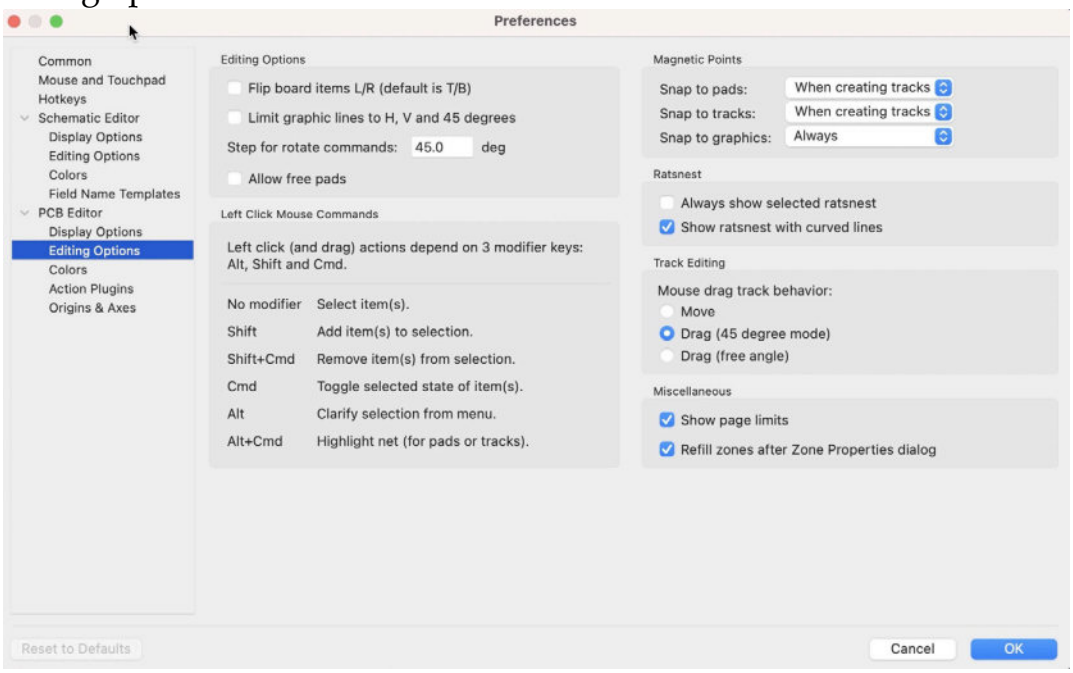


Figure 8.5.2: Pcbnew Editing Options.

I'll focus on some of the more frequently used options.

Editing Options

Under Editing Options, you will find a text field where you can set the rotate command step. By default, it is 45-degrees, but you can set it a custom value if you need more granular rotation control.

Magnetic Points

Under Magnetic Points, you can set how you would like your drawing to snap onto a compatible object as the mouse pointer gets closer to that object. Without snap to pads, tracks, and graphics, it will be more difficult to precisely join a track to another track or pad. It will also be more difficult to, for example, create a closed polygon which is essential when you draw the perimeter of a PCB in the Edge.Cuts layer. With snaps on, the layout editor will show a small circle around the point that is a candidate for a snap action and close the drawing if you move the mouse close enough.

In the example below, I have turned the three snap options to "always." My mouse pointer ("1") is close to the snap point ("2"). The editor has detected the pointer and snap-point proximity and marked the snap point with a circle. It also made the connection.

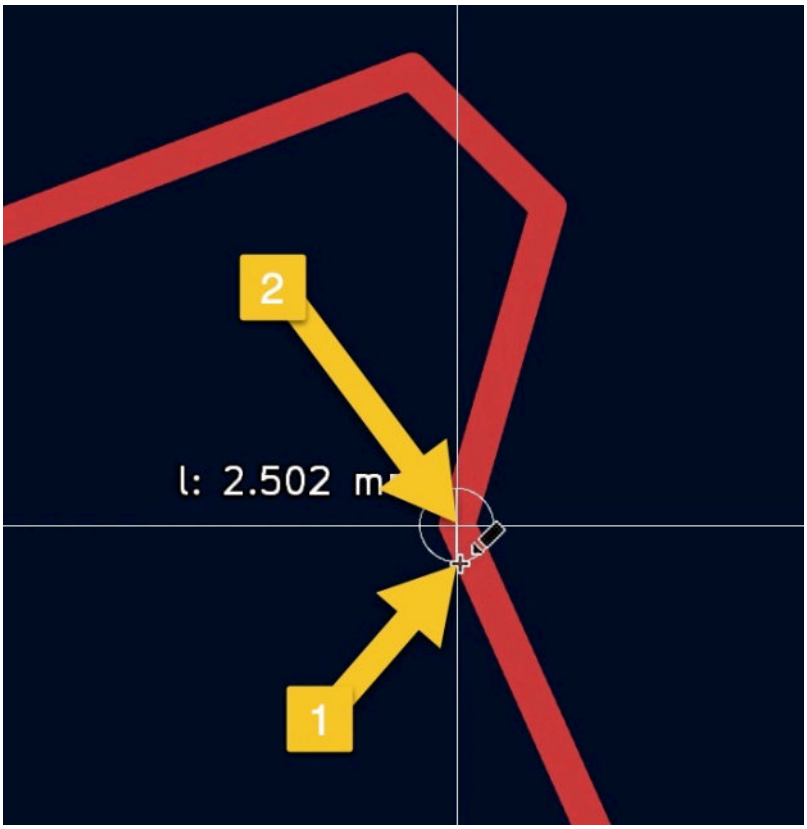


Figure 8.5.3: Snap in action.

3: Colors

In Pcbnew, you can create custom color themes. You can do this in the Colors tab, as you can see below:

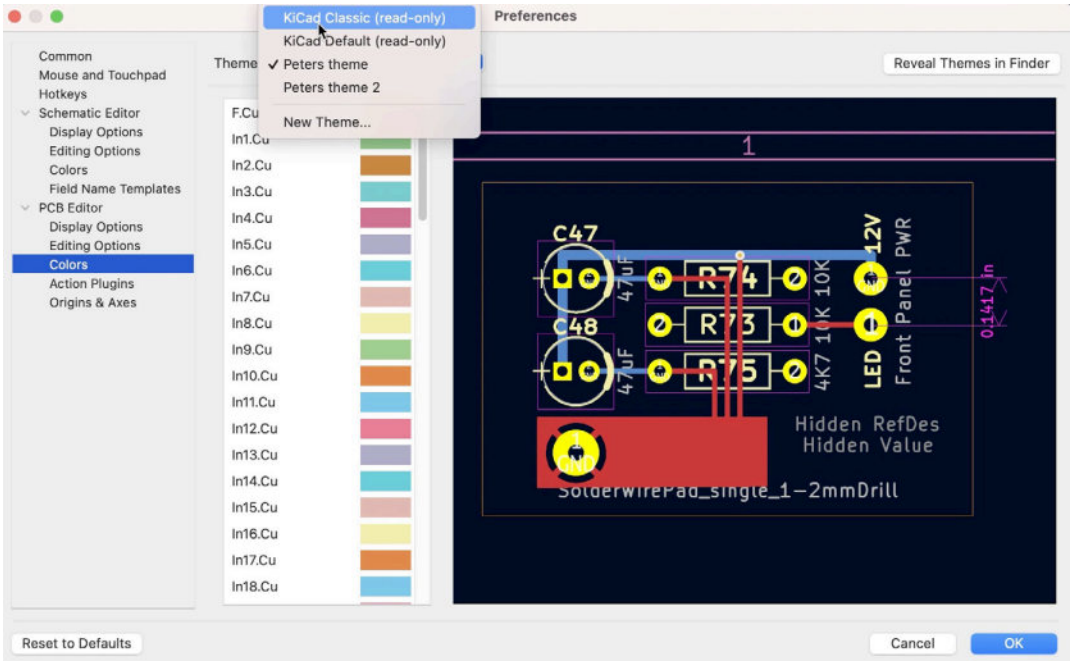


Figure 8.5.4: Color themes.

You can modify any of the KiCad (read-only) themes and save them as custom themes. You can then enable a theme using the drop-down menu at the top of the window.

To change the color of an item in the theme, click on the item's color box, and select the color from the color chooser window that appears.

4: Action Plugins

KiCad is extensible through its new Python API and plugin system. When I am writing these lines, both components are under development, and their documentation is not ready. As a result, I was not able to do any testing.

I will be updating this section as soon as the Action Plugins feature is released.

5: Origins & Axes

This pane allows you to change the origin and orientation of the coordinate system of the layout editor. The coordinate system is essential in any CAD application, so I have written a dedicated chapter in the Recipes part of this book.

6. Board Setup

In this chapter, you will learn about the configuration options in the Board Setup window. This window contains three groups of options:

- Board stackup controls the physical characteristics of your board, such as the total number of copper layers and the copper finish materials.
- Text & Graphics, which controls default attributes of silkscreen text and allows you to set text variables.
- Design Rules, which control the behavior of the Design Rules Checker, and constraints, among other things.

Let's begin with a look into the Board Stackup group of options.

6.1. Board Setup - Board Stackup

In this chapter, you will learn about the options available in the Board Stackup group of the Board Setup window. In the Board Stackup group, you will find four tabs:

1. Board Editor Layers.
2. Physical Stackup.
3. Board Finish.
4. Solder Mask/Paste.

To access the Board Setup window, select Board Setup from the file menu in Pcbnew, or click on the second-left button from the top toolbar (next to the Save button).

Physical Stackup

Perhaps the most important decision you have to make when you start the layout design of a board is to choose the number of copper layers the board will contain. The number of required copper layers depends on the complexity (number of components and pins that must be connected) and size of your board. In general, given an equal number of components and pins, you can reduce the size of the board by increasing the number of copper layers. More copper layers allow you to design a more compact board with fewer vias and jumpers but at a higher cost. A typical four-layer board can also have dedicated copper layers for ground and power, leaving two layers

for signal tracks. This setup is also well suited for high-speed PCBs and PCBs that contain radio components.

Select the Physical Stackup tab under the Board Stackup group in the Board Setup window to set the number of copper layers for a board. You can use the Copper layers dropdown to select anywhere from 2 to 32 copper layers (see Figure 8.6.1.1 below).

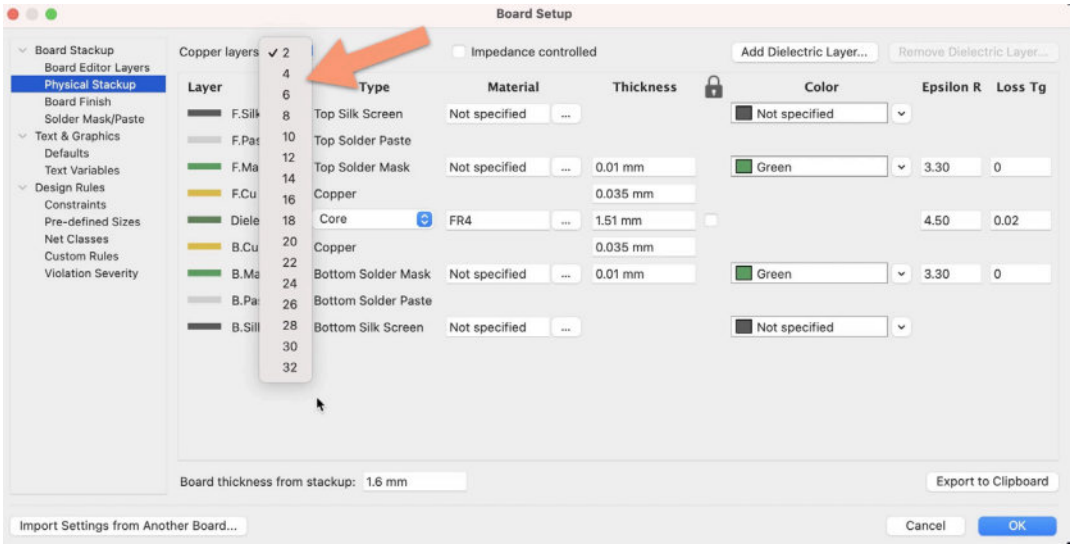


Figure 8.6.1.1: Setting the number of copper layers.

Once you select the number of copper layers for your board, the pane contents will reflect your choice. In the example below, I have set a four-layer board, and the layers listing contains a total of four configurable layers:

- Front copper: F.CU.
- Inner copper 1: In1.Cu.
- Inner copper 2: In2.Cu.
- Back copper: B.Cu.

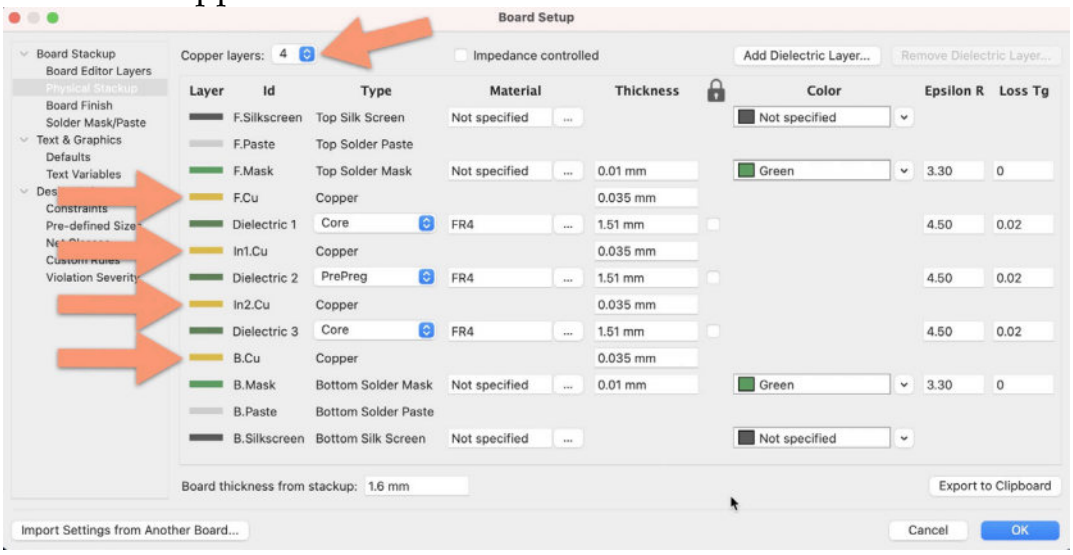


Figure 8.6.1.2: Four copper layers.

Click OK to commit the change, and notice that the Layers tab under Appearance in the right toolbar and the layers dropdown in the top toolbar are also updated to reflect the new board setup:

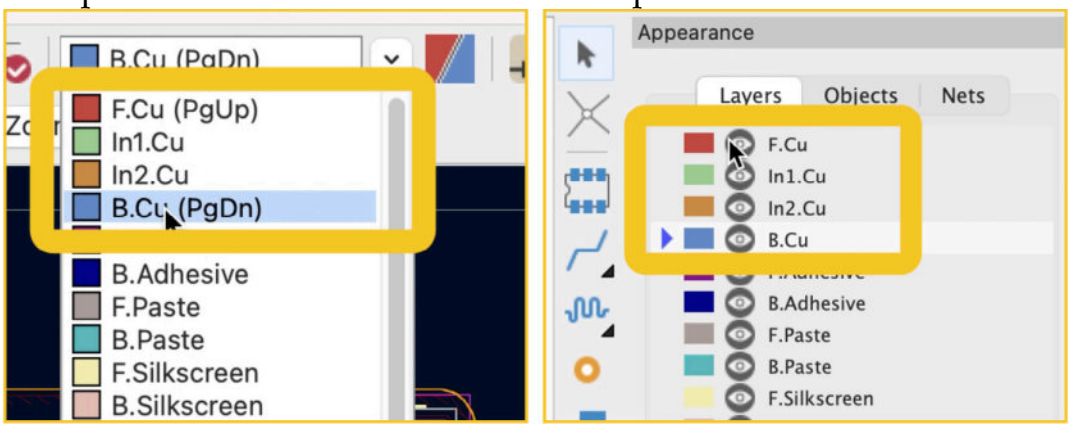


Figure 8.6.1.3: Four copper layers PCB.

Apart from the number of copper layers, the Physical Stackup tab also allows you to configure other aspects of your board, such as the type of dielectric material and thickness used between the copper layers and the material used in the silkscreen layers, and much more.

Keep in mind that although you can make these selections in KiCad, it is up to the manufacturer to apply them in manufacturing your board. Suppose you change any of the default settings here. In that case, you should communicate with the manufacturer to ensure that they can and that they will implement your customizations before you order your PCB.

Board Editor Layers

In the Board Editor Layers tab, you can change the name of each layer, enable or disable a layer, and select the role of copper layers.

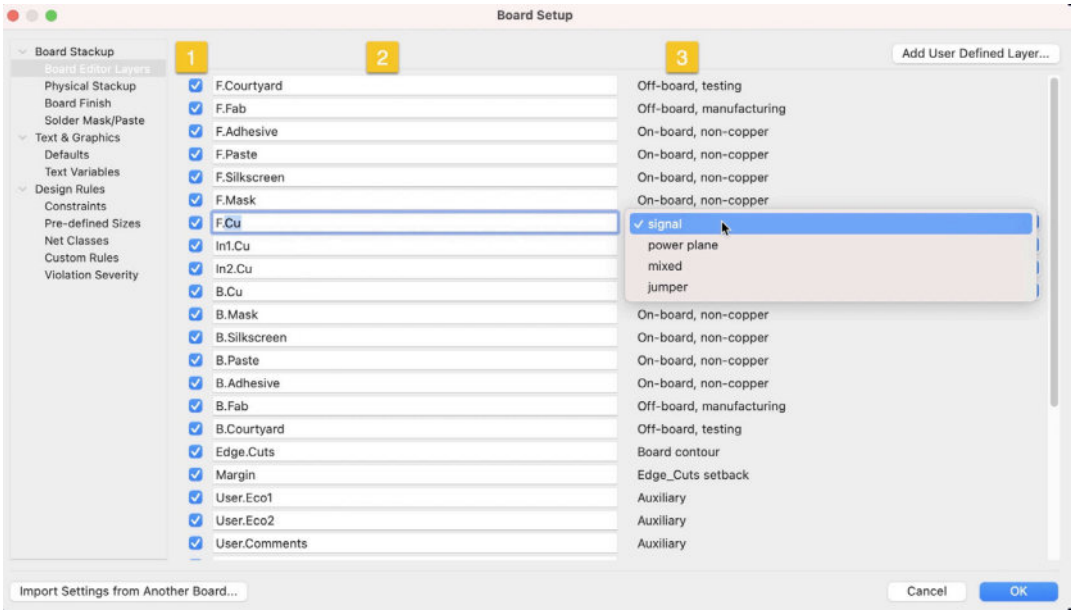


Figure 8.6.1.4: The Board Editor Layers tab.

In Figure 8.6.1.4 (above), you can see the options available in the Board Editor Layers tab.

1. To enable a layer and make it visible in the top toolbar layer dropdown or the layers tab of the Appearance pane, check its checkbox.
2. To customize the name of a layer, click in its name text field and type in a new name.
3. For copper layers only, you can customize the layer role. The options are signal, power plane, mixed, and jumper.

Board Finish

In the Board Finish tab, you can select various finish options for your board.

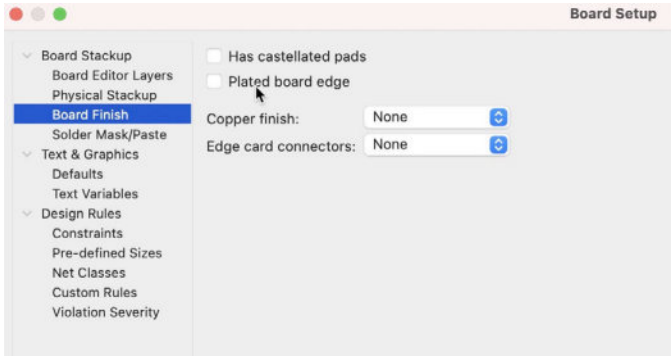


Figure 8.6.1.5: The Board Finish tab.

Castellated pads are semi-plated holes arranged on the edge of the board. These pads are suitable for soldering a board on another board without the

need for pins. In Figure 8.6.1.4 below, you can see the ESP-WROOM-32 module, which uses castellated pads along its three edges.

Figure 8.6.1.6: An example of a board using castellated pads.

A plated board edge is an option where the entire perimeter of a board is coated with copper. Plated board edging offers some benefits, such as the ability to solder the edge of a board within a container or improve the current carry capability of a PCB. You can see an example of a board with plated board edges in Figure 8.6.1.7 below. You can find the original, as well as more information on board edging, on the Pcbgogo website.

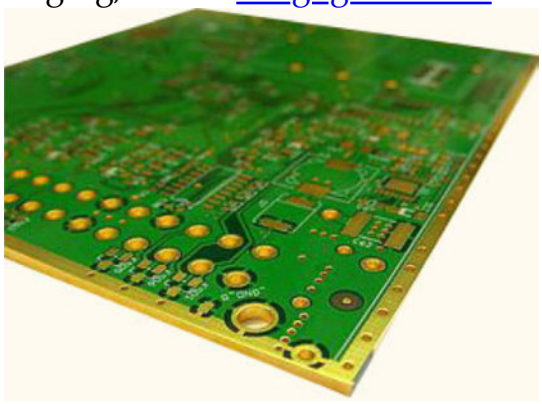


Figure 8.6.1.7: An example of a board using plated board edges.

In the same pane, you can also select your preferred copper finish material. There are several options, as you can see below:

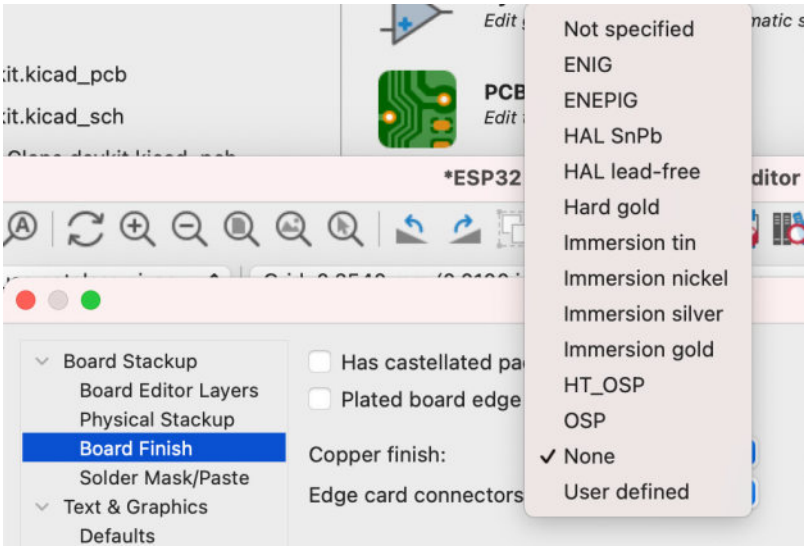


Figure 8.6.1.8: Copper finish options.

Pcbway has an excellent article on their website with information about many copper finish options.

Solder Mask/Past

This tab allows you to define solder mask and paste settings.

For example, you can set the solder mask and paste clearances. You can get the appropriate values for these fields from your manufacturer's website.

In most cases, it is safe to leave the default values (all fields to zero).

6.2. Board Setup - Text & Graphics

Let's continue with the options in the Text and Graphics group. Within this group are the Defaults and Text Variables panes.

Defaults

This is where you can set up defaults for the various text and graphics parameters. You can set parameters such as the line thickness for the edge cuts layer (where you design the board's perimeter) or the thickness of the text that appears in the silkscreen. It is possible to select different text appearance parameters for text labels in silkscreen and copper layers.

In the screenshot below, you can see the available options in this pane:

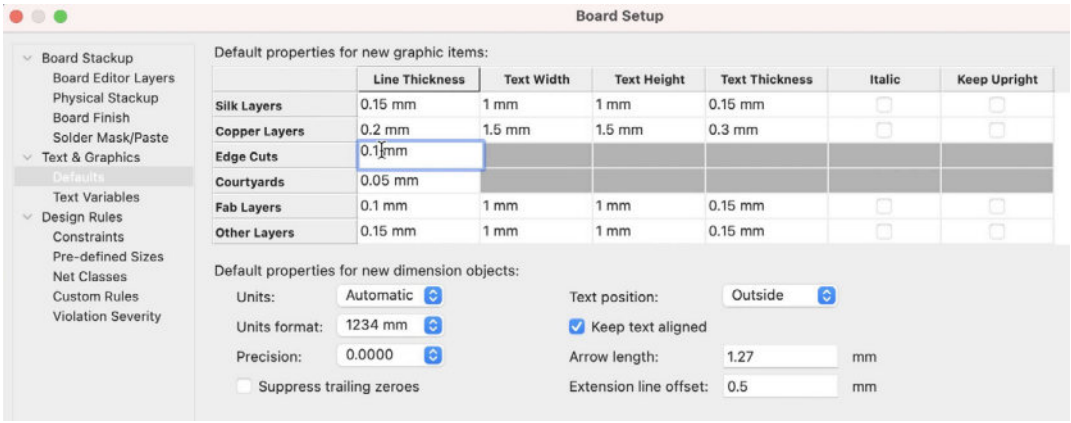


Figure 8.6.2.9: The Defaults pane in the Text & Graphics group.

Here, you can also set the preferences unit of measurement (inches, millimeters, mils), the format, and precision.

Text Variables

Text Variables is a new feature in KiCad 6. You can create a text variable, apply it in any element that contains text in the layout editor (such as a text label), and the layout editor will substitute it for the value you have specified in the Text Variables table.

I will demonstrate using an example. See Figure 8.6.2.10 below:

Figure 8.6.2.10: Text Variables table.

Bring up the Board Setup window, and click on Text Variables, under Text & Graphics. Click the "+" button to add a new row in the table ("1"). Type in a name ("2"), and a value ("3"). Click OK to close the window ("4").

Click on the text label button from the right toolbar, and type some text in the text field. Then, create a text label. Use the "${Variable_name}" pattern to use a text variable. In my example, I use "${example_variable_1}" (see below):

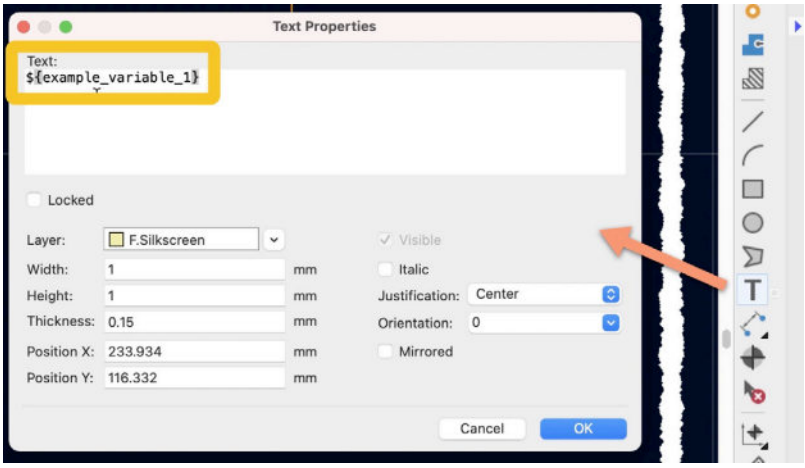


Figure 8.6.2.11: Creating a new text element that calls a text variable.

Click OK to close the window and place the new text label on the sheet. As you can see, the editor has already replaced the variable name with its value:

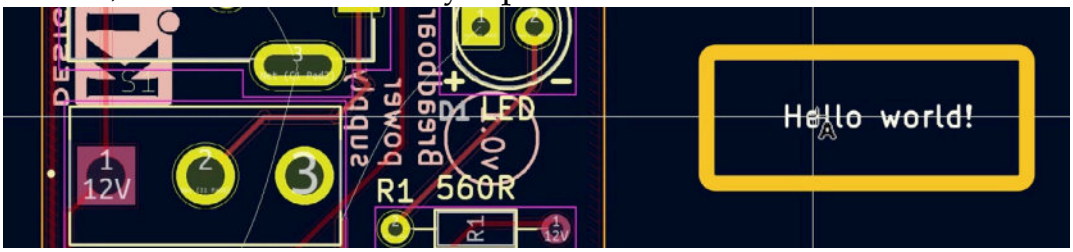


Figure 8.6.2.12: Text variable substituted with its value.

The text variable substitutions happen dynamically. As soon as you change the value of a variable, the change will be reflected in the editor sheet. For example, go back to the Text Variable pane, and make a change to the value for "example_variable_1".

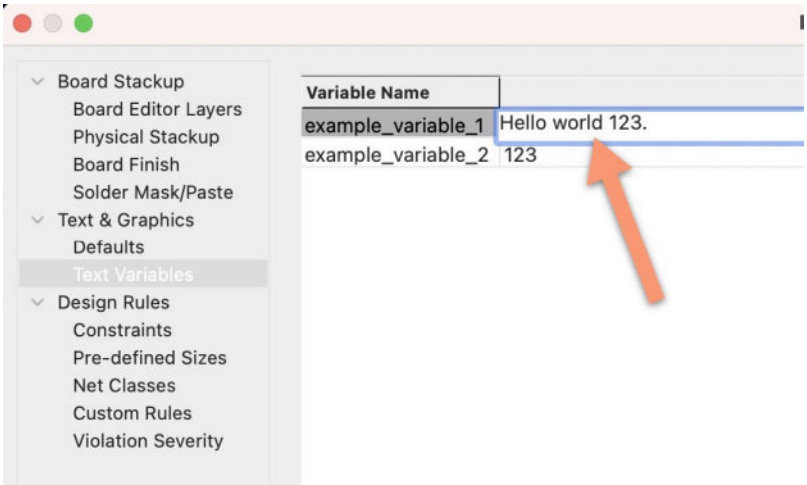


Figure 8.6.2.13: Changed the value of a variable.

Click OK to close the window. The text that appears in the label will change to reflect the variable's new value (see below).

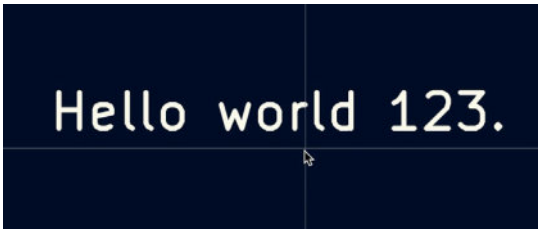


Figure 8.6.2.14: Text that contains text variables are always up-to-date in the sheet.

Of course, it is possible to mix text variables with fixed text anywhere you can use text.

You can also use text variables within footprint properties. For example, you can use text variables inside a footprint's Value field, like in the example below:

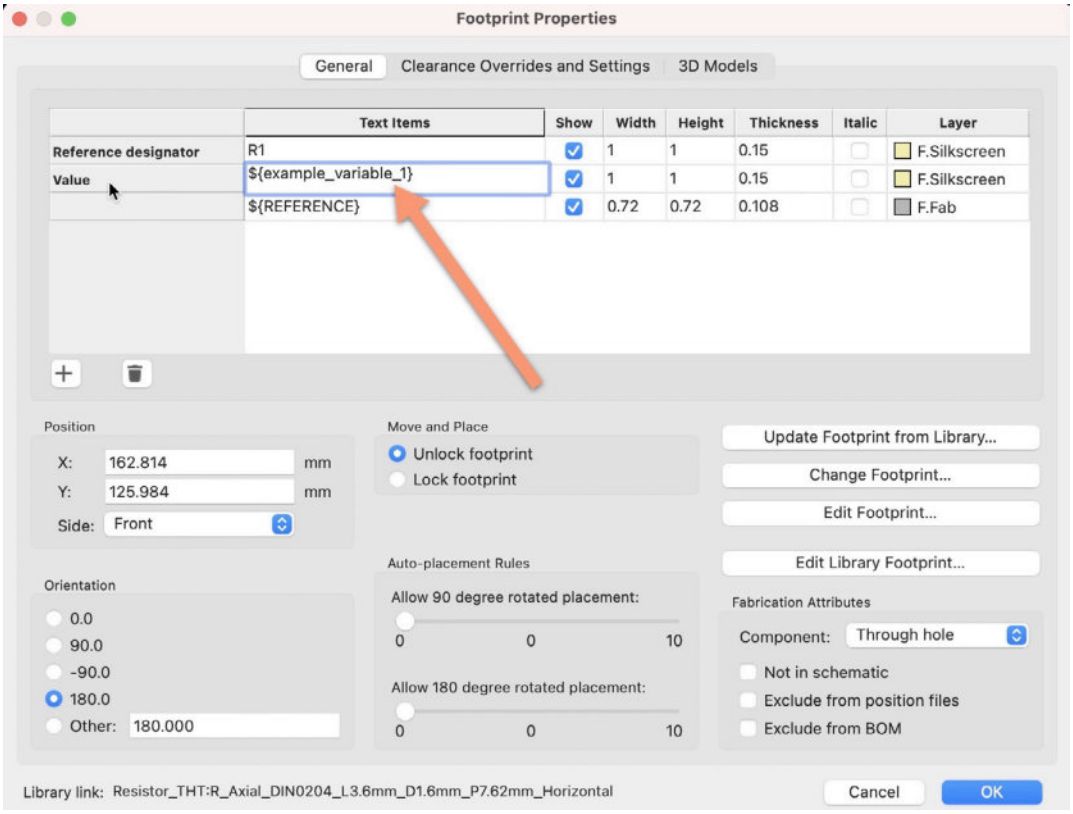


Figure 8.6.2.15: Text Variables can be used anywhere there is text.

Click OK, and notice that the value of the resistor as it appears in the front silkscreen now contains the value of the text variable:

Figure 8.6.2.16: Text Variable replaced with its value.

6.3. Board Setup - Design Rules and net classes

In the Design Rules group, you can configure settings relating to the Design Rules Checker and set up Net Classes. I have divided the discussion of the contents of the Design Rules group into two parts to make it more manageable.

In this first part, you will learn about Constraints, Pre-defined Sizes, and Net Classes.

In the second part (next), you will learn about Custom Rules and Violation Severity.

Constrains

This is where you can set up global constraints for your layout design. See the Constraints tab in Figure 8.6.3.17 below:

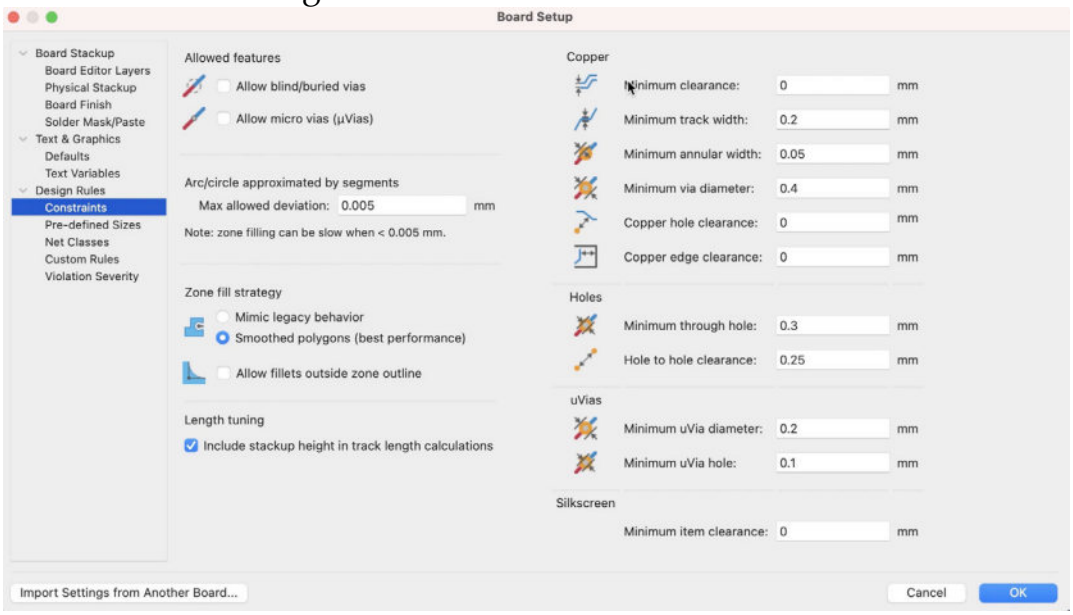


Figure 8.6.3.17: Design Rules Constraints.

Here, you can set up constraints such as the minimum acceptable clearance between copper tracks and the minimum through-hole diameter. The DRC (Design Rules Checker) will use these values to detect errors in your design. You should consult your PCB manufacturer's website for the minimum values they support before you make any changes to these fields.

For example, if I change the minimum clearance of a copper track to 0.50 mm, the layout editor will depict this change in the sheet using thin lines to indicate the required clearance around the track:

Figure 8.6.3.18: These thin lines mark the minimum clearance for copper tracks.

The values in the fields in the Constraints tab apply globally. Often, you want more granularity in the way that the DRC works. It is possible to apply minimum values for properties like copper track widths and clearances using the Net Classes pane, which you will learn about shortly.

Pre-defined Sizes

In this pane, you can set track widths via sizes and differential pair sizes that you can choose via the top toolbar dropdown or the context menu.

Here is an example:

Figure 8.6.3.19: The Predefined Sizes tab.

Below you can see the pre-defined sizes in the top toolbar dropdowns:

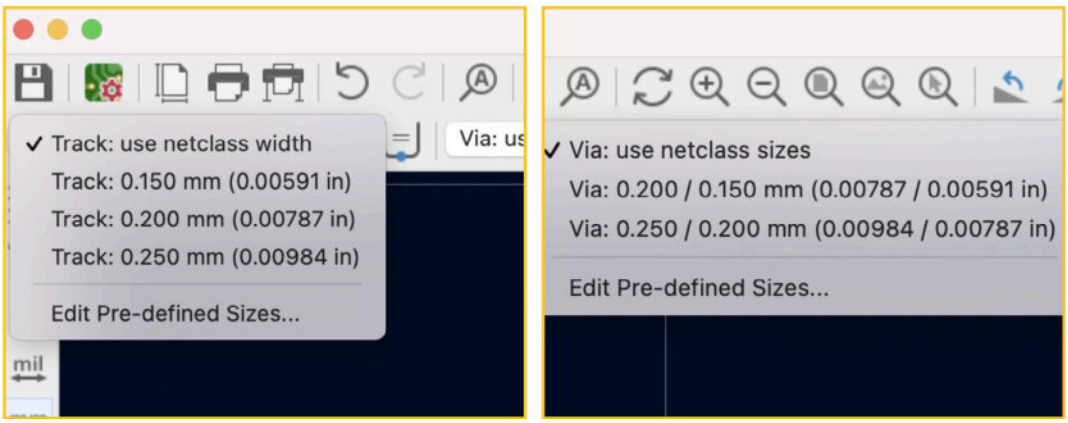


Figure 8.6.3.20: The Predefined Sizes in the top toolbar dropdown..

Net Classes

In the Net Classes pane, you can set minimums that apply to nets that belong to specific net classes. Those minimums must be equal or larger to the minimums specified in the Constraints tab.

Remember that you have seen Net Classes before in the schematic editor. Net Classes in the layout editor work similarly to their namesake in the schematic editor. If you have specified net classes in the schematic editor, you will see them in the layout editor and will be able to set their layout-specific properties. If you did not create any net classes in the schematic editor, you can do so in the layout editor.

Below you can see the Net Classes pane, containing two net classes. The default net class initially has all nets. You can create custom net classes and move nets into them.

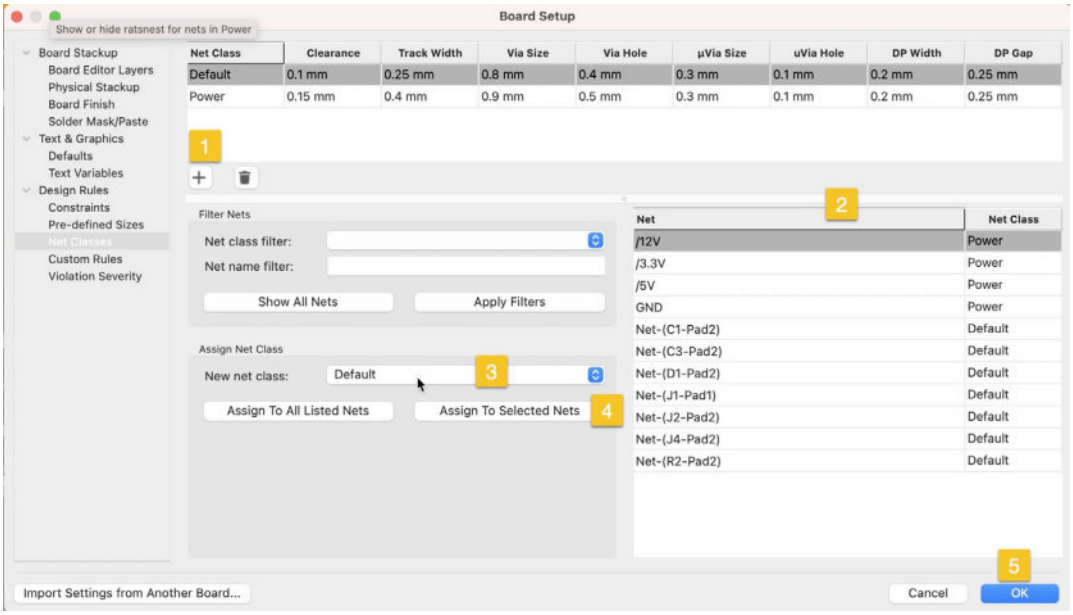


Figure 8.6.3.21: The Net Classes tab.

In the example above, I have created the Power custom net class. To create a net class, click the "+" button and give it a name. In the net class field, click inside the various parameter fields to change their values. In the net table ("2"), you can see all nets and their net class membership. Click one net to select, or select multiple nets that you want to change their net class membership. With the net(s) selected, use the net class dropdown menu to choose the target net class ("3") and then click "Assign To Selected Nets" ("4") to finish the allocation. Click OK to close the window ("5").

In the example above, I have set the minimums for the copper tracks in the Power net class to be slightly larger than the minimums in the Default class. This is because copper tracks that belong to the Power net class convey larger currents than signal tracks.

6.4. Board Setup - Design Rules - Custom Rules and violation severity

Here you will learn about the Custom Rules and Violation Severity tabs in the Design Rules group for the Board Setup window.

Custom rules

Custom Rules is a new feature in KiCad 6. With custom rules, you can set design rules programmatically. Below is an example of a simple custom rule:

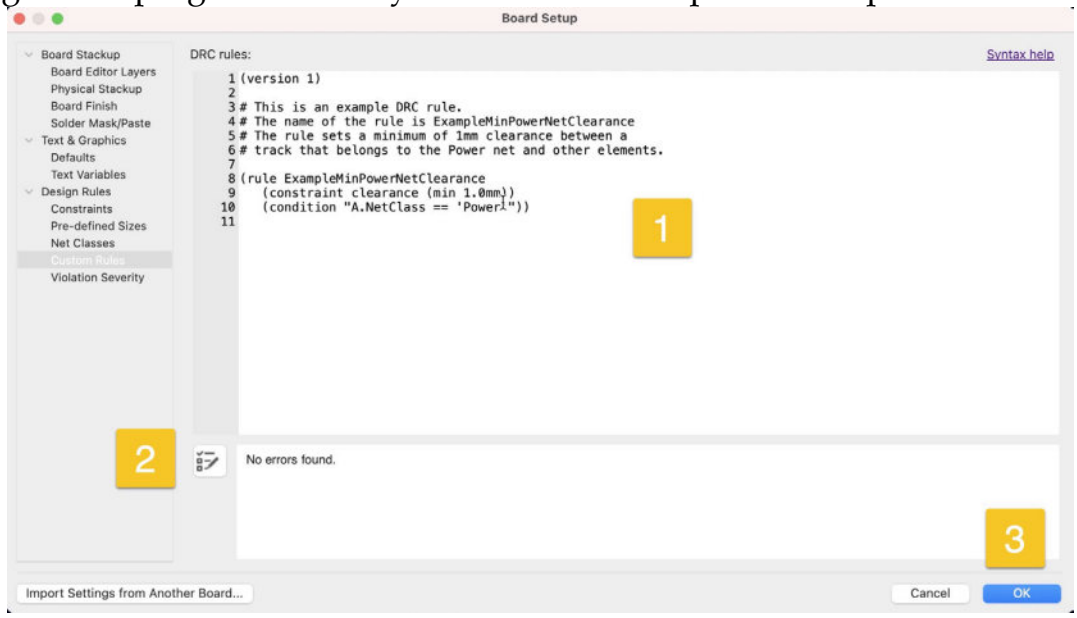


Figure 8.6.4.22: A simple custom rule.

In the rule in the example above, you can see a single custom rule with four lines of comments. The rule is versioned (this is "version 1"). This rule has:

- A name: “ExampleMinPowerNetClearance”.
- A constraint: “clearance (min 1.0mm)).
- A condition: “A.NetClass == ‘Power’”.

This rule enforces a minimum 1 mm clearance for copper tracks or pads that belong to the Power net class.

After you type the rule ("1"), click on the Checker button to look for bugs in the code ("2"), and then OK to deploy the rule ("3"). Do a DRC to test the new rule. In my case, because the clearance minimum is so small (1mm), I am getting a lot of errors:

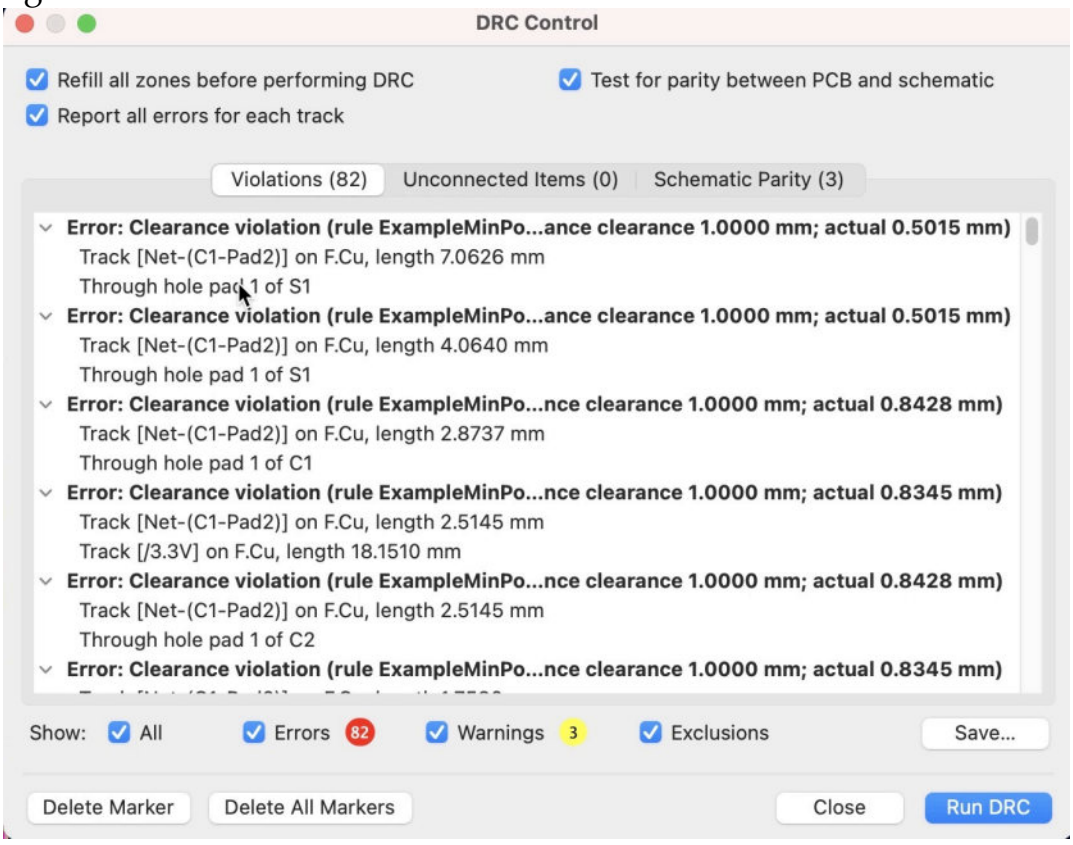


Figure 8.6.4.23: Custom rule errors appears in the DRC window.

As you can see, the name of the custom rule appears in the error messages so that you can know about the source of the error.

You can follow the same pattern to create multiple custom rules for your project.

Click on the Documentation link (top right corner of the Custom Rules pane) to see the rules syntax documentation with many examples to help you get started:

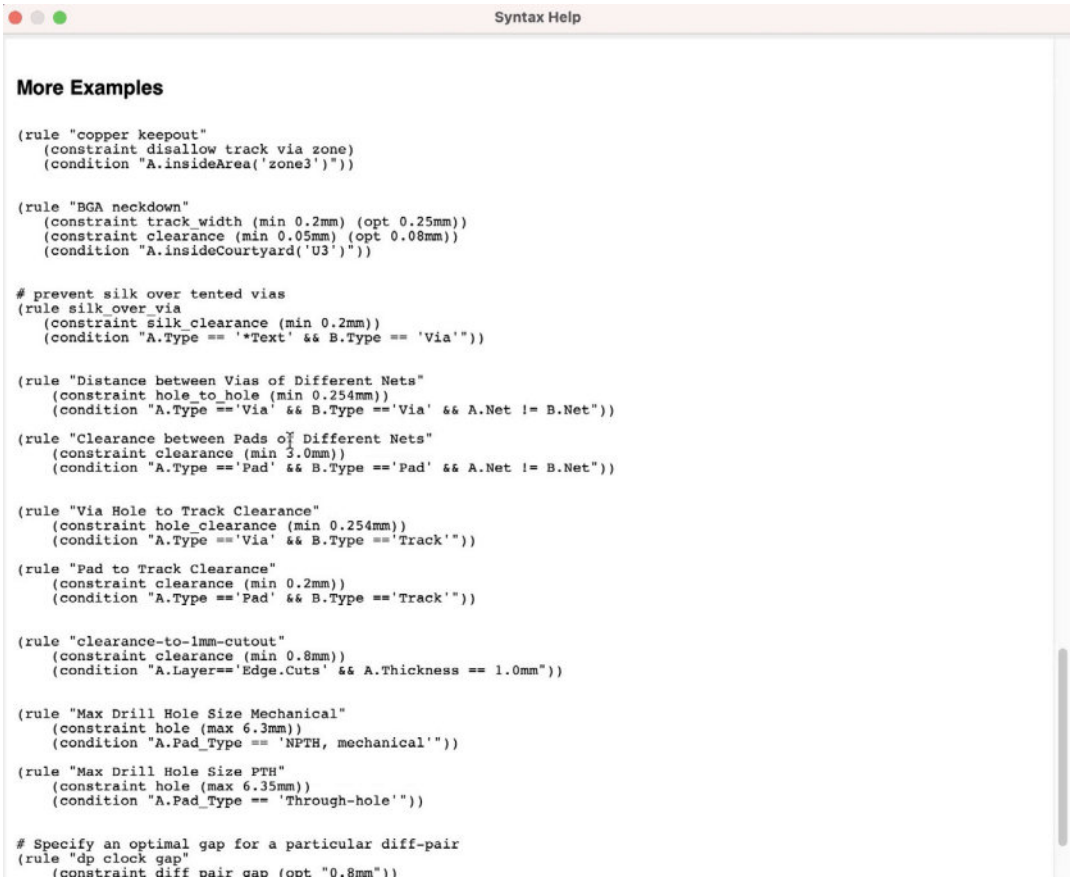


Figure 8.6.4.24: Custom Rules documentation

To disable a custom rule, you can either delete it from the editor or comment out its lines of code.

Violation severity

The Violation Severity tab settings work similarly to its namesake in the schematic design editor settings. You can see the Violation Severity pane in the Design rules group below:

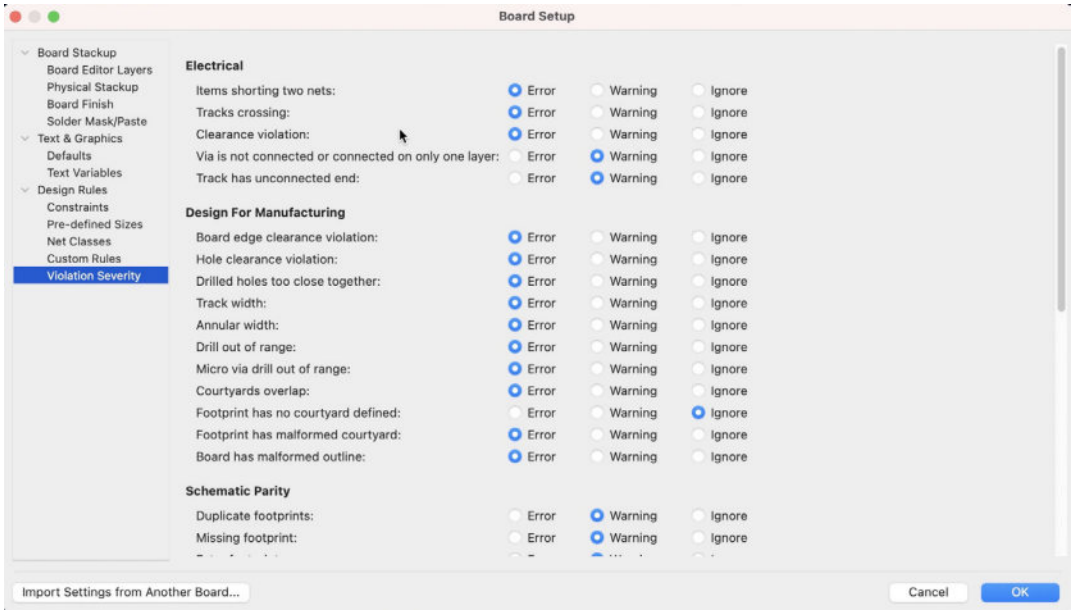


Figure 8.6.4.25: Violation Severity settings.

Use the radio buttons next to each violation to set its classification: Error, Warning, and Ignore.

The default settings are reasonable and appropriate for most projects, so think carefully before making any changes.

Here is an example. Below, I have changed the severity of the "Extra footprint" violation to "Error." The original is "Warning."

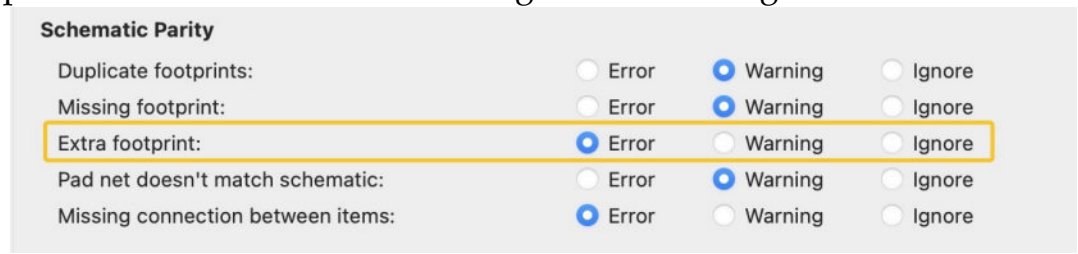


Figure 8.6.4.26: Changed severity for “Extra footprint”.

When I ran a DRC, the check revealed one "extra footprint" violation (a logo graphic that only exists in the layout but not in the schematic). Normally, this violation would appear as a warning, but because of my change, it now appears as an error:

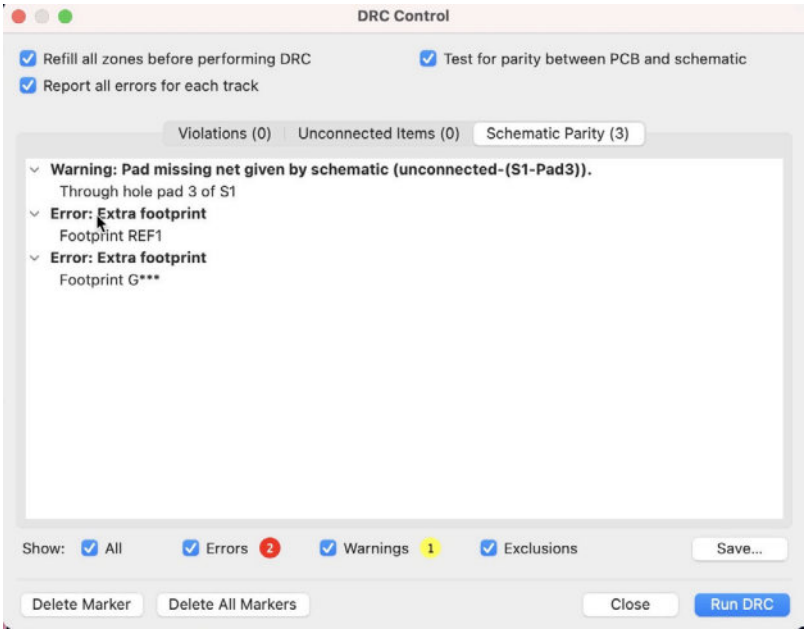


Figure 8.6.4.27: Extra footprint violation is classified as “error”.

7. How to find and use a footprint

In this chapter, you will learn how to find a footprint in the footprint browser and then use it in the layout editor. First, you should be familiar with the location of the footprint libraries on your computer's filesystem. This information is available in Pcbnew —> Preferences —> Manage Footprint Libraries.

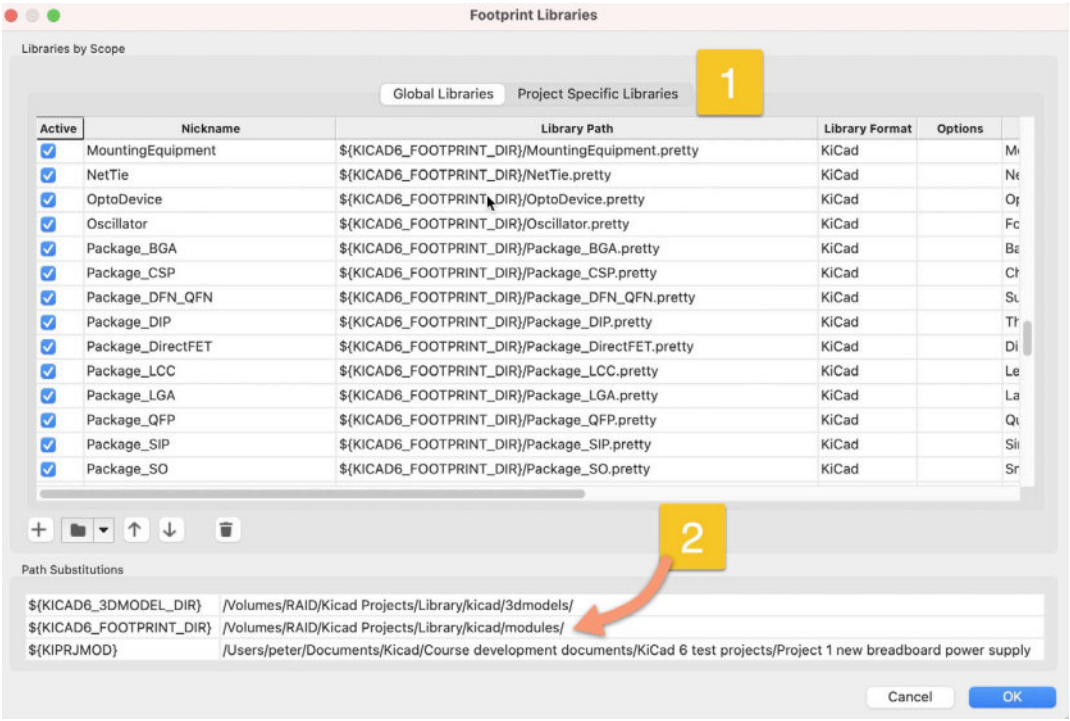


Figure 8.7.1: The location of the footprints libraries ("2") on my computer.

In the Footprint Libraries window, you will find the Global and Project libraries tabs ("1"). At the bottom is the path substitutions. The second row contains the footprint libraries path in the example above, which points to an external RAID drive. Because KiCad libraries can occupy a lot of disc space, I have chosen to store them in an external drive.

The footprints found in the specified path are listed in the Global Libraries tab at the top of the Footprints Libraries window, and I will be able to find them in the library chooser (see next). The Project Specific Libraries tab contains arbitrary libraries that are visible within their specific project. I tend to store these libraries within the library project folder.

It would be best if you also remembered that you would not need to look for footprints in the layout editor in most cases. This is because, by the time you start work in the layout editor, you will already have completed the footprint and symbol association in the schematic editor. A case where you would be searching for a footprint in the layout editor is if you want to add a graphic (like a logo) or perhaps auxiliary footprints like mounting holes. Even for those footprints, though, it is better to treat them as any other footprint and associate them with a symbol in the footprint editor.

Say that you wish to add a new footprint in the layout editor. You can bring up the footprint chooser window by clicking on the footprints button in the right toolbar. Then, browse the library hierarchy, or use the filter to find the footprint you need (Figure 8.7.2 below).

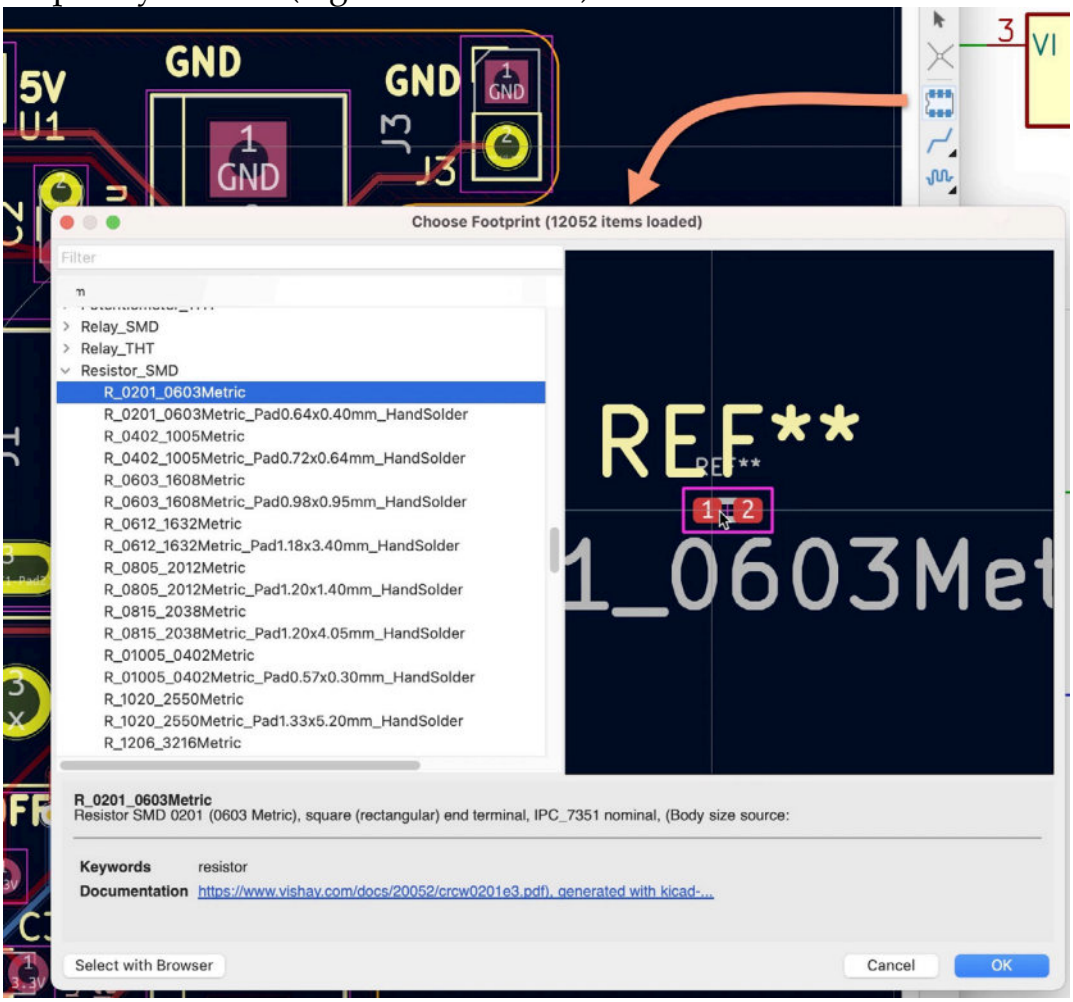


Figure 8.7.2: Finding a footprint in the footprint chooser.

Once you find and select a footprint, click OK to close the chooser window, and then place the footprint in the editor (Figure 8.7.3 below):

Figure 8.7.3: Placed the footprint in the editor.

Because the new footprint does not exist in the schematic editor, there are no ratnest lines to guide the wiring. You can still draw copper tracks, though. A way to do this without the interference of the interactive router is to use the "highlight collisions" router mode and check "Allow DRC violations" (see below). You can access the Interactive Router Settings window from the route menu in Pcbnew.

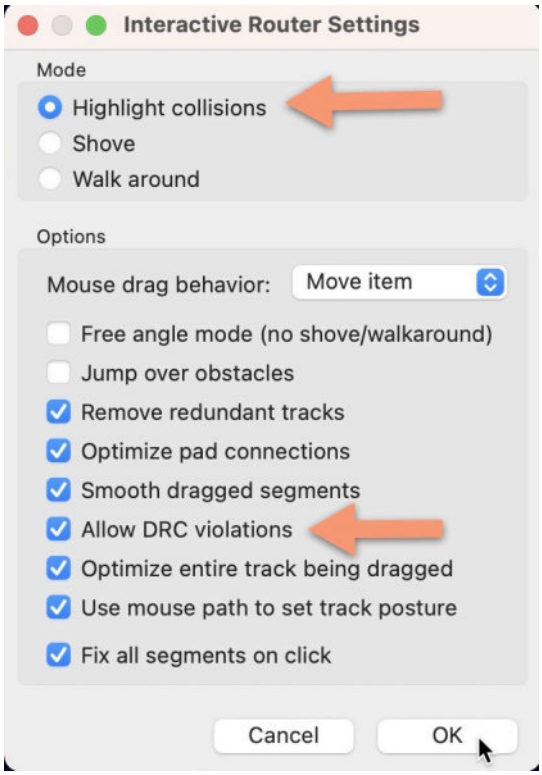


Figure 8.7.4: Enable "Allow DRC violations".

With these settings, you can choose the track tool from the right toolbar and draw a new track from the resistor to another tab (see below).

Figure 8.7.5: Drawing a track independent of the schematic design.

The “Highlight collisions” mode is the only one that permits the “Allow DRC violations” option to be enabled.

8. Footprint sources on the Internet

In this section, you will learn about sources on the Internet to find footprints or footprint libraries for your KiCad projects. In the chapter on Internet sources for schematic symbols, in Part 7 of this book, you learned about my four recommended sources. The same sources apply for footprints (as well as 3D shapes).

These sources are:

1. KiCad's footprint library repository at https://kicad.github.io/footprints/. KiCad contributors add new footprints frequently. It is possible that in the time elapsed since I downloaded my copy of the libraries, the footprint that I am looking for was added to the repository.
2. Snapeda at https://www.snapeda.com/. Snapeda is a repository with millions of parts for all major CAD software applications. Rarely, a part I am looking for does not exist in Snapeda. In most cases, Snapeda will provide everything you need for a part: symbol, footprint, and very often the 3D shape.
3. Octopart at https://octopart.com/. I find Octopart to be as good as Snapeda in terms of finding symbol and footprint libraries. You can use Octopart as an alternative or a complement to Snapeda.
4. Ultralibrarian at https://www.ultralibrarian.com/. Similar top Snapeda and Octopart.

In addition to the above repositories, there are several footprint collections that I also recommend you install in your KiCad instance.

These are:

1. Digikey's KiCad library at https://github.com/Digi-Key/digikey-kicad-library.
2. Sparkfun's KiCad library at https://github.com/sparkfun/SparkFun-KiCad-Libraries.
3. Freetronics's KiCad library at https://github.com/freetronics/freetronics_kicad_library.

I will not show you how to download an individual footprint file or a collection of footprints to avoid duplication. The process is identical to the one I describe in the chapter from Part 7, so I invite you to refer to that chapter for the details. In the next chapter I will describe installing single or multiple footprint files so you can use them in Pcbnew.

9. How to install footprint libraries

In this chapter, you will learn how to install both a single footprint file and a collection of footprint files in KiCad so that you can use them in your layout designs.

Single footprint file installation

We will use an example from Snapeda: an SMD antenna module. In this example, I have downloaded both the symbol and the footprint from the component page on Snapeda (for KiCad v4 or later):

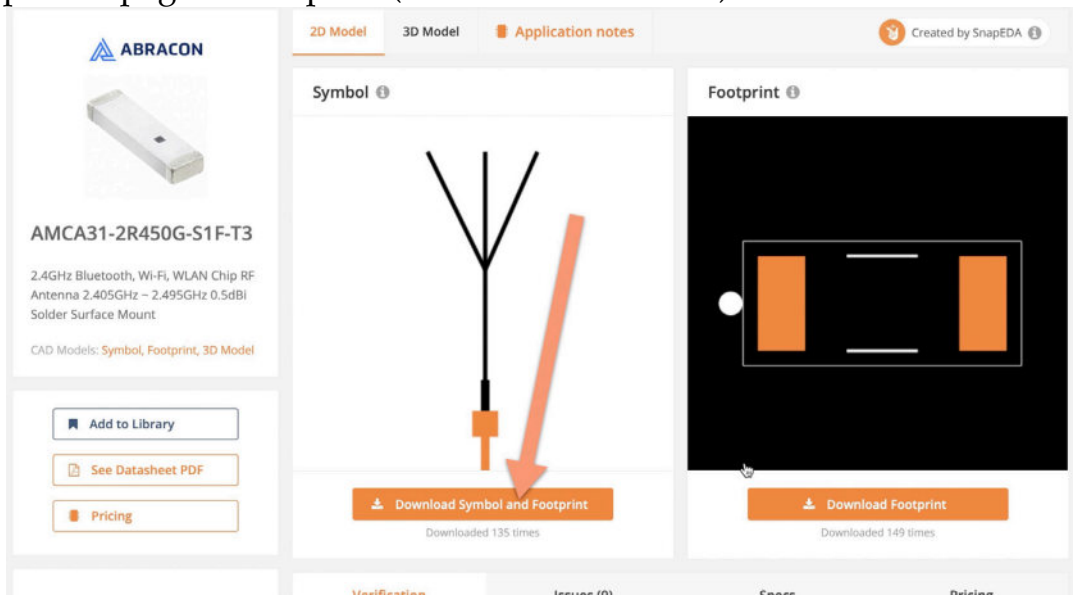


Figure 8.9.1: Downloading the symbol and footprint files for this component.

For this particular component, the downloaded ZIP file contains three main files: ".lib" for the symbol, ".step" for the 3D shape, and ".kicad_mod" for the footprint (see below):

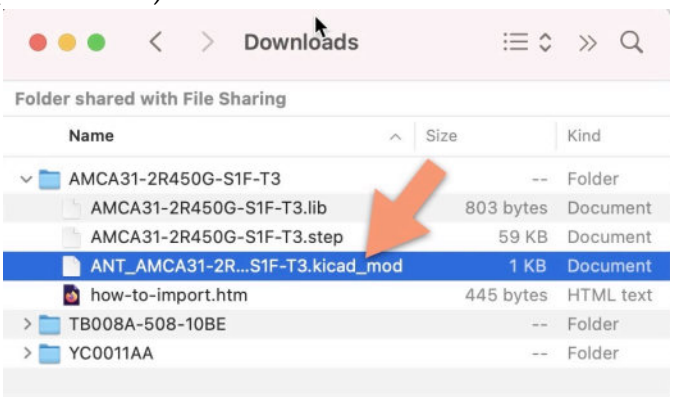


Figure 8.9.2: The contents of the downloaded ZIP file.

Let's install the footprint file so you can use it in Pcbnew. Go to Pcbnew, and click on "Manage Footprint Libraries" under Preferences. I want to install this file to be used by all my projects, so I select the "Global Libraries" tab ("1" in the figure below).

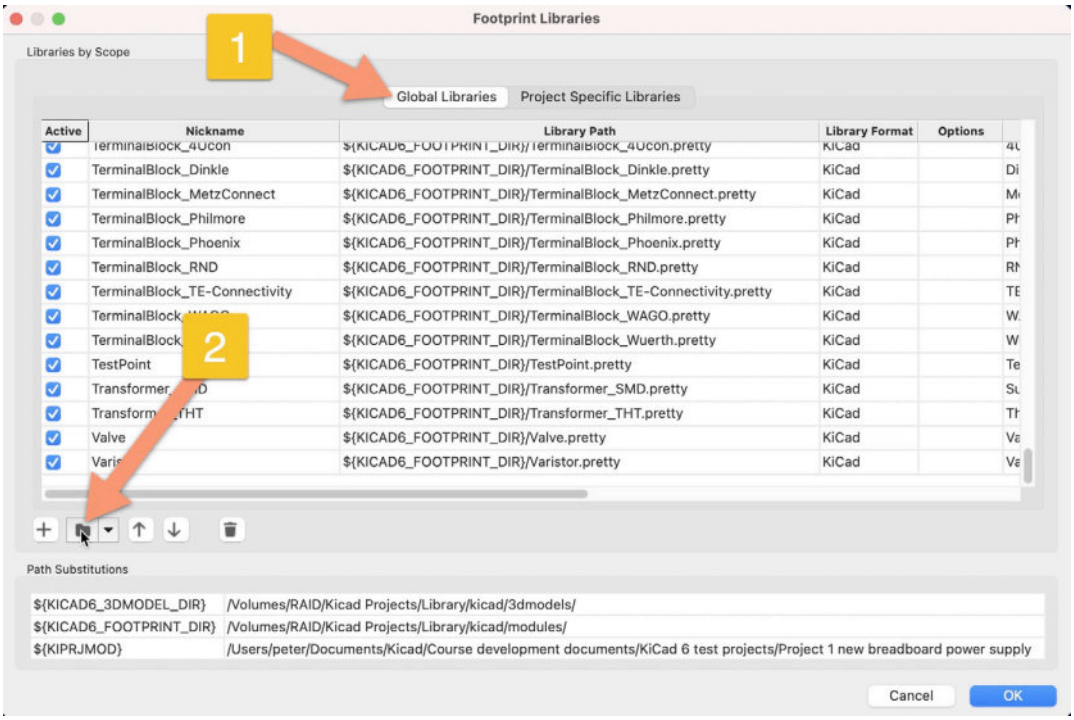


Figure 8.9.3: The footprint libraries window.

Click on the folder button ("2") to bring up the file browser so you can find the footprint file to import. Use the file browser to navigate your file system to the folder that contains the footprint file.

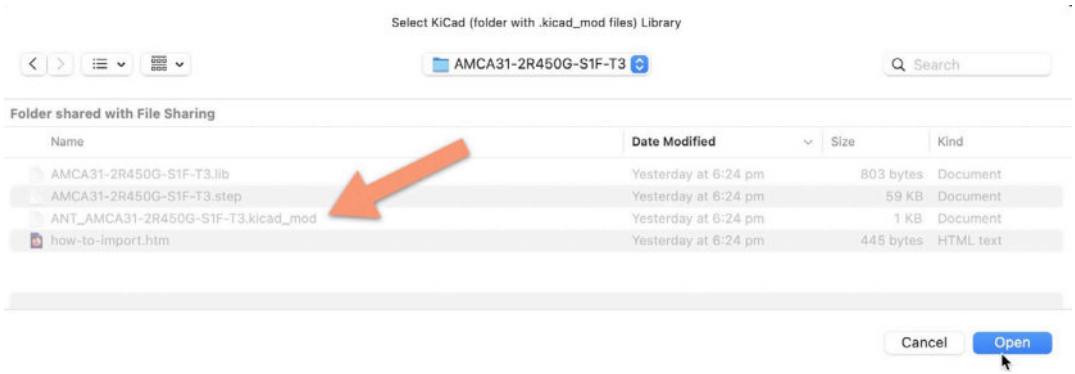


Figure 8.9.4: The folder that contains the footprint file.

The KiCad will look inside this folder, find any compatible file with the ".kicad_mod" extension, and import it. Click Open. At the bottom of the footprints list, you will see a new row that points to the imported footprint:

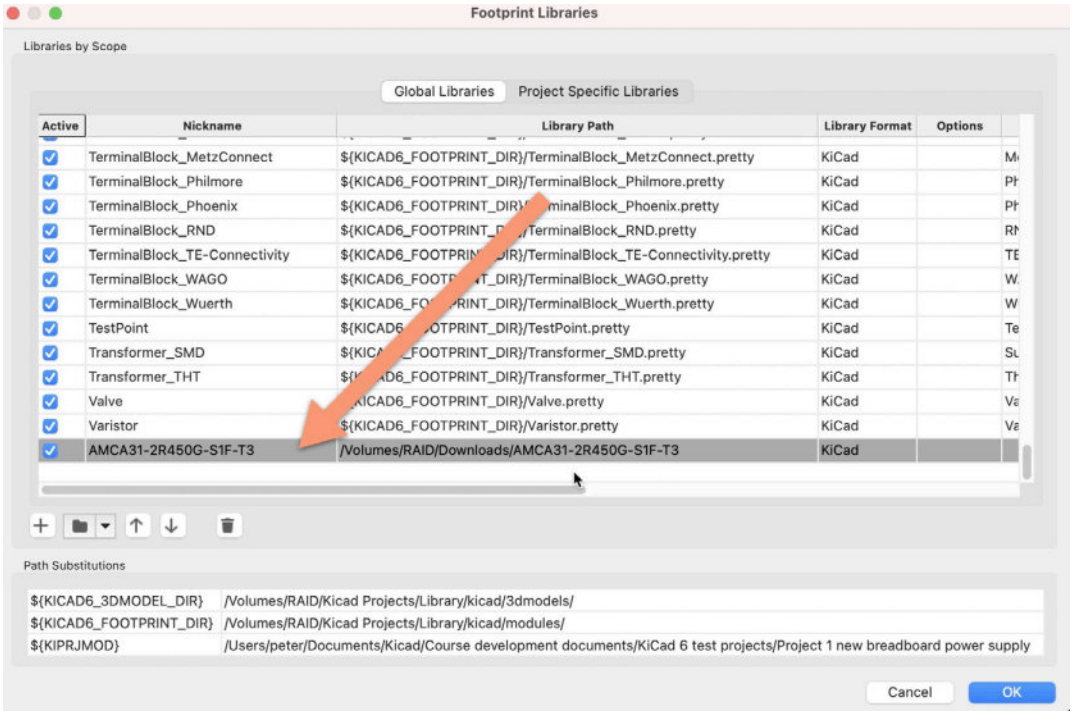


Figure 8.9.5: The new footprint is imported and ready to use.

Click OK to dismiss the footprint library window, open the footprints library browser (use the "O" hotkey), and search for the new footprint (I used the first three letters of its name). It will show up as in the example below:

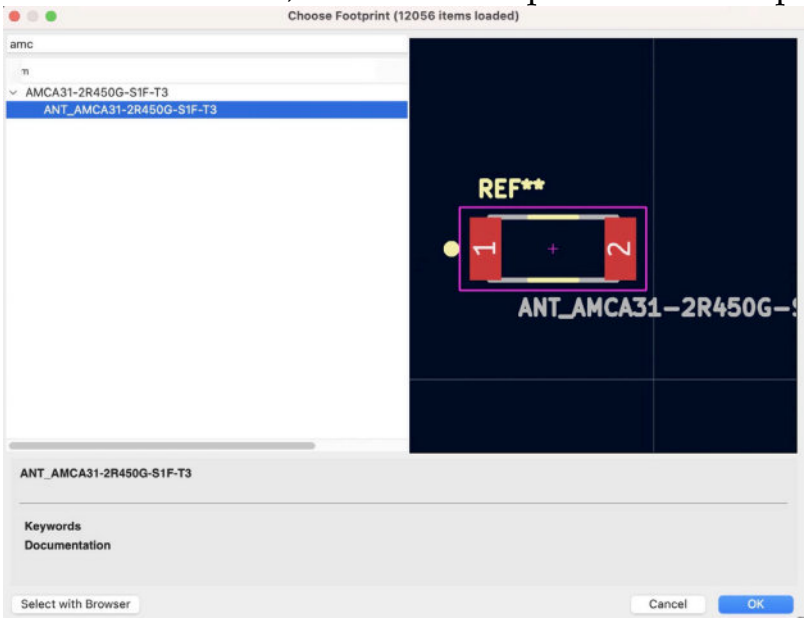


Figure 8.9.6: The new footprint in the footprint chooser.

Click OK and position the footprint in place (I had to move a few other elements around to make room in the PCB).

Multiple footprint file installation

Another common scenario is when you need to install multiple footprint files into KiCad. You could use the single installation method I showed above, but this is not optimal.

To demonstrate, I will use the footprints collection from Digikey (see below):

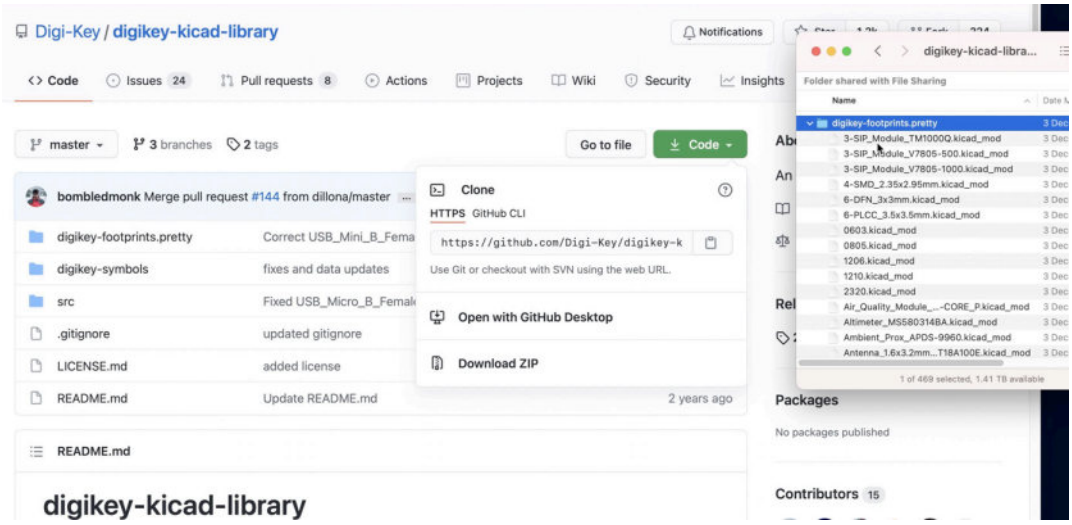


Figure 8.9.7: The Digikey footprint collection on my computer.

I have downloaded the ZIP archive from the Github repository and extracted it on my computer in the example above. The contents of the "digikey-footprints.pretty" folder are multiple footprint files.

Go to Pcbnew, and click on "Manage Footprint Libraries" under Preferences. Again, I will install this collection in the Global Libraries tab.

Click on the folder button to bring up the file browser, and navigate to the folder that contains the footprints (see below).

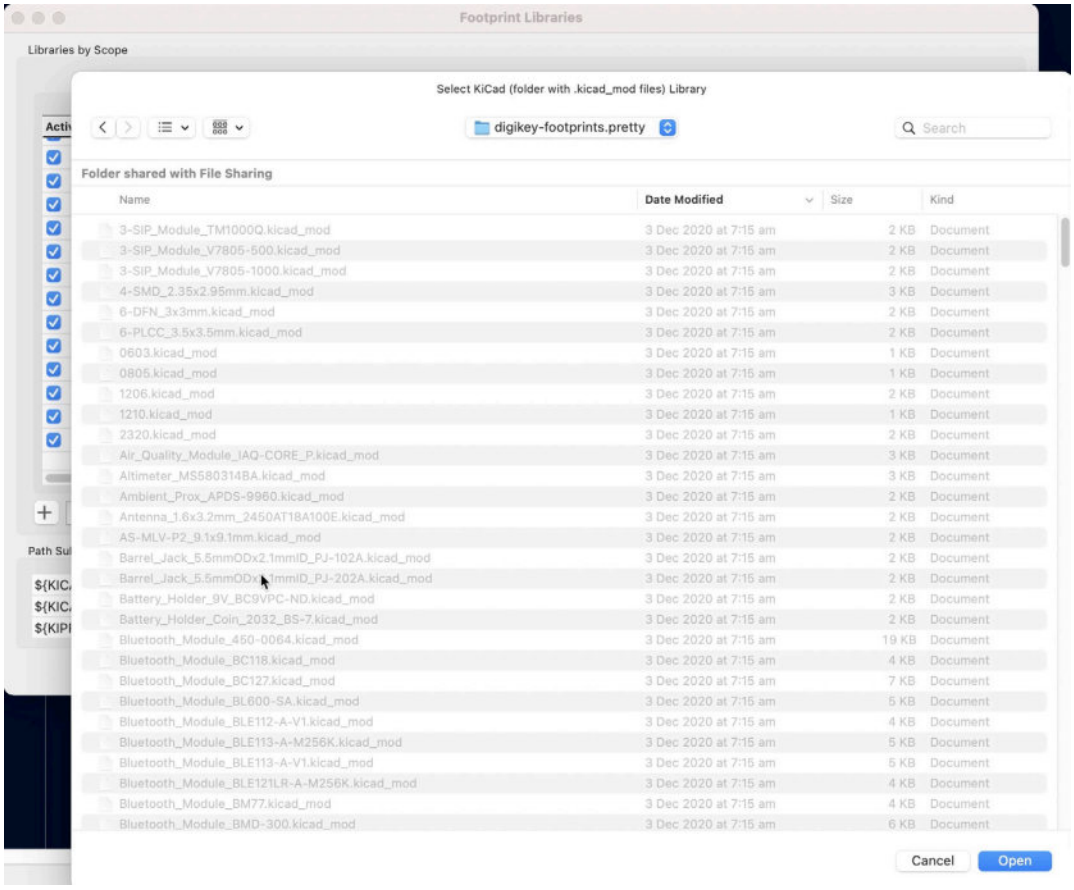


Figure 8.9.8: Importing the Digikey footprint collection.

Click Open. KiCad will look for all valid footprint files and import them.

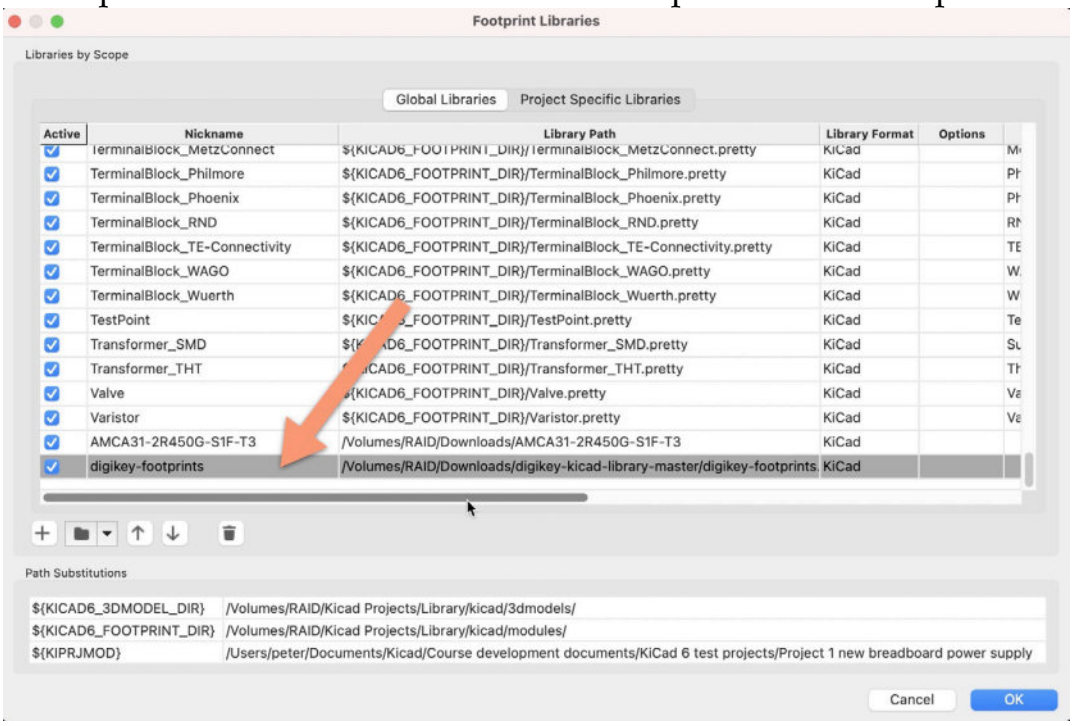


Figure 8.9.9: The Digikey footprint collection is ready to use.

The footprints in this collection are now ready to use. The pointer to the new library appears at the bottom of the list (see image above).

Let's look for one of the footprints in the Digikey collection, the BME680. Use the "O" hotkey to bring up the footprint chooser and search for "bme680". The footprint and its preview will appear:

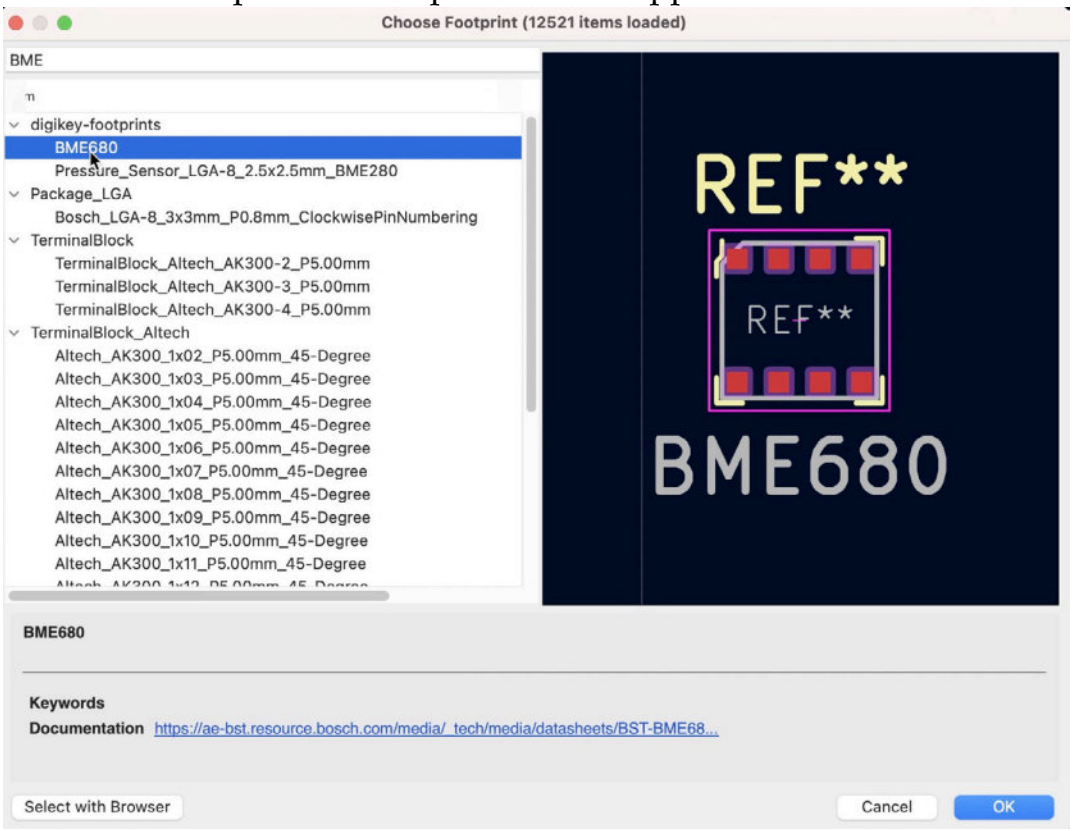


Figure 8.9.10: The BME680 footprint from the Digikey footprint collection.

Click OK to close the window, and place the footprint in the editor:

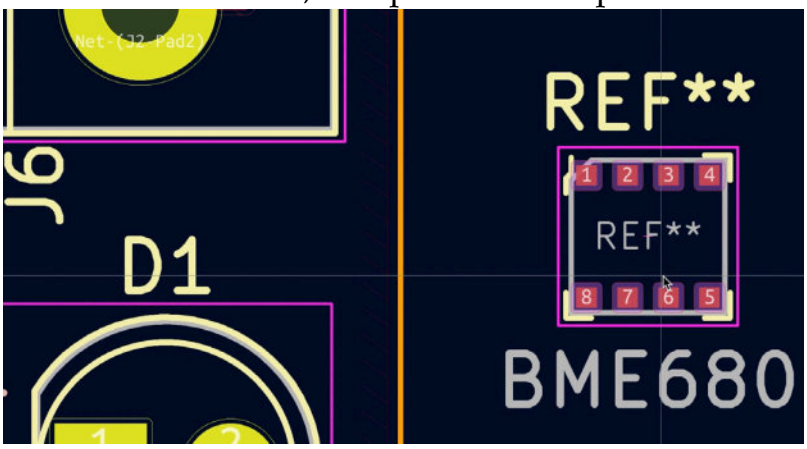


Figure 8.9.11: The BME680 in the editor.

The new footprints are also accessible from Eeschema. For example, below, you can see the footprint associations window. I have selected the LED

symbol (“1”) and the digikey-footprints library (“2”). I associate the LED symbol with the 0805 footprint (“3”) from the Digikey library.

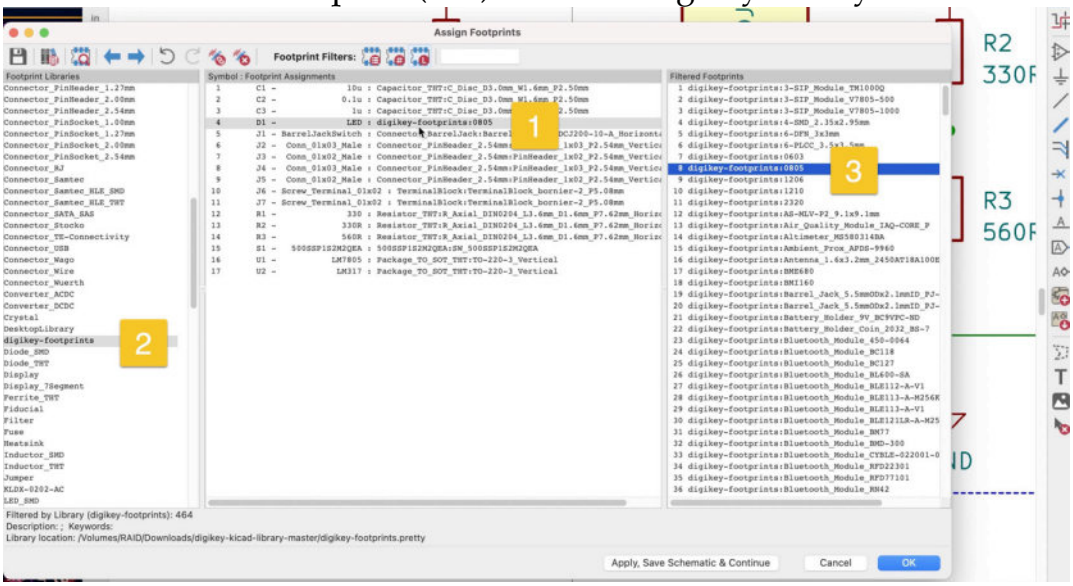


Figure 8.9.12: Associating the LED symbol with an SMD footprint from the Digikey library.

Return to Pcbnew and click on the "update from schematic" button in the top toolbar. Check the re-link footprints and replace footprints checkboxes so that the editor will remove the old THT footprint and replace it with the new SMD one (see below).

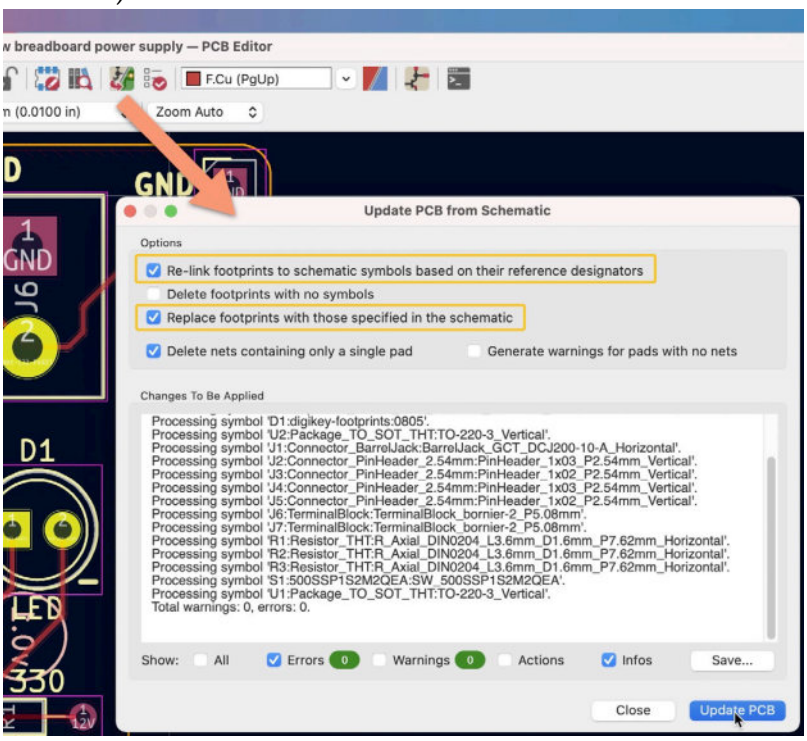


Figure 8.9.13: Updating the PCB from the schematic.

Click on Update PCB, and place the new footprint in position:

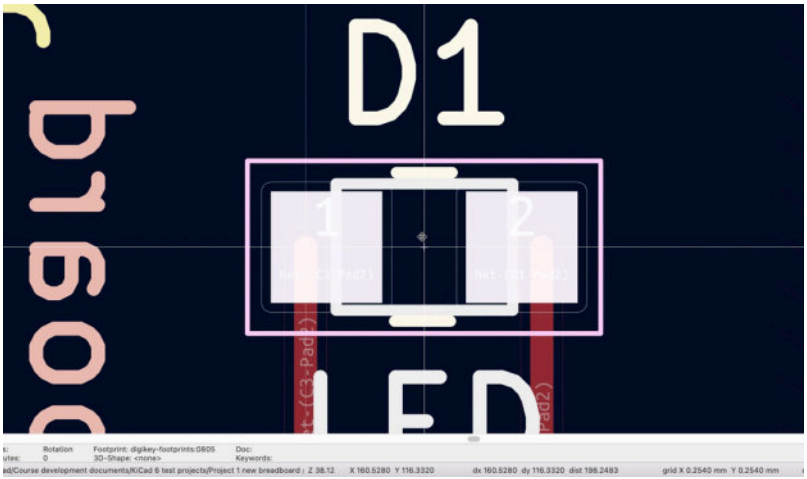


Figure 8.9.14: Changed the THT footprint to SMD.

As you can see, KiCad is very flexible: it is easy to import footprints and symbols and easy to associate and re-associate them.

10. Filled zones

In this chapter, you will learn how to add one or more copper zones to your PCB. A copper zone is an area of a PCB filled with copper, either in a continuous pour or a pattern. Using the process you will learn in this chapter, you can create multiple copper zones in any of the available copper layers.

To demonstrate the process, I will use a PCB from one of the projects in this book. I will show you how to create a copper zone in the back copper layer and will connect this zone to the GND net.

To create a new copper filled zone, click on the copper fill button in the right toolbar:

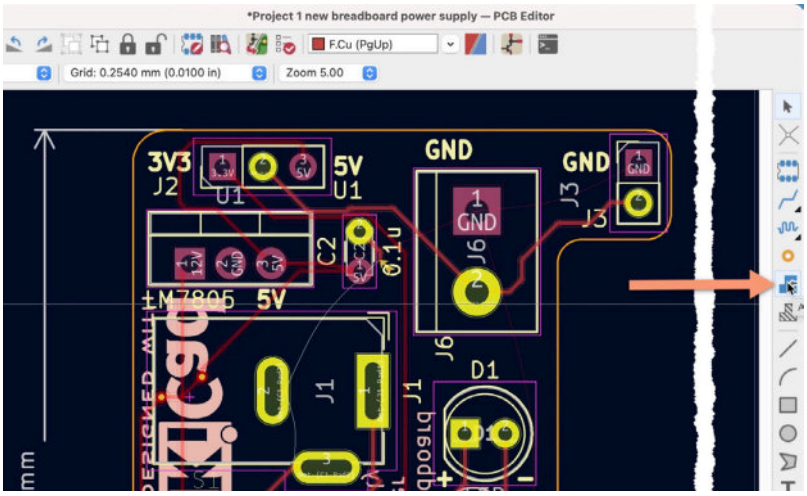


Figure 8.10.1: Creating a new copper filled zone.

This tool will change the cursor to a pen. Click on the location of the board where you want to start drawing the copper zone. In this example, I plan to create a copper zone that covers the entire back copper layer, so I will start drawing from the top right corner of the board, just below the top boundary:

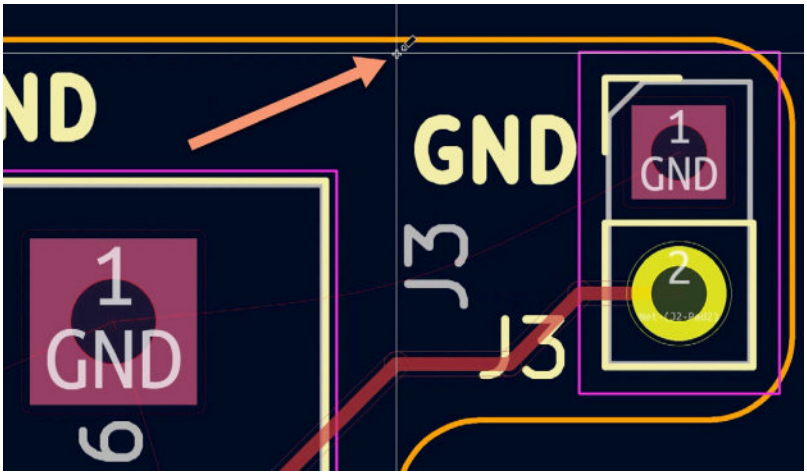


Figure 8.10.2: Start drawing the new copper zone.

Click to start drawing. The editor will bring up the copper zone properties window where you can configure the new zone:

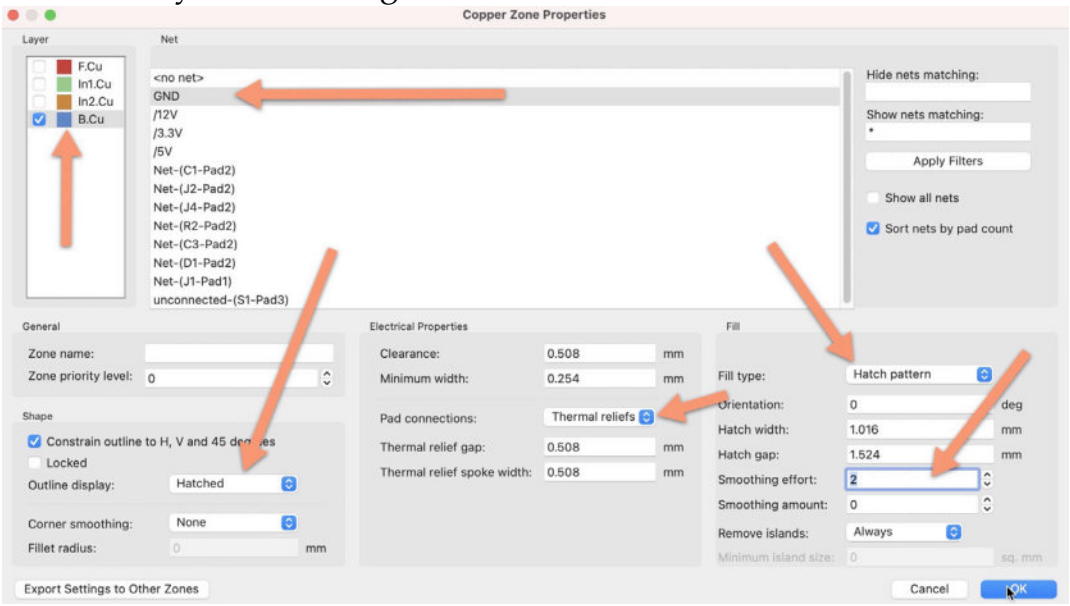


Figure 8.10.3: Configuring the new copper zone.

In the example above, I use pointers to indicate the settings I have changed. Since I want to create a copper zone in the back copper layer, contact to the GND net, I have selected “B.CU” and “GND” from the Layer and Net panes. You can explore the rest of the options and copy my settings from the example above.

Click OK to close this window and continue the drawing of the copper zone. Each click adds a new point in the zone’s polygon. I am drawing the zone as close as I can to the PCB’s boundary. To close the polygon, I finish the drawing with a double-click on the point I started. In the example below, you can see my drawing process at several points, from start to finish. Use your

mouse middle button to pan and the scroll wheel to zoom during the drawing process.

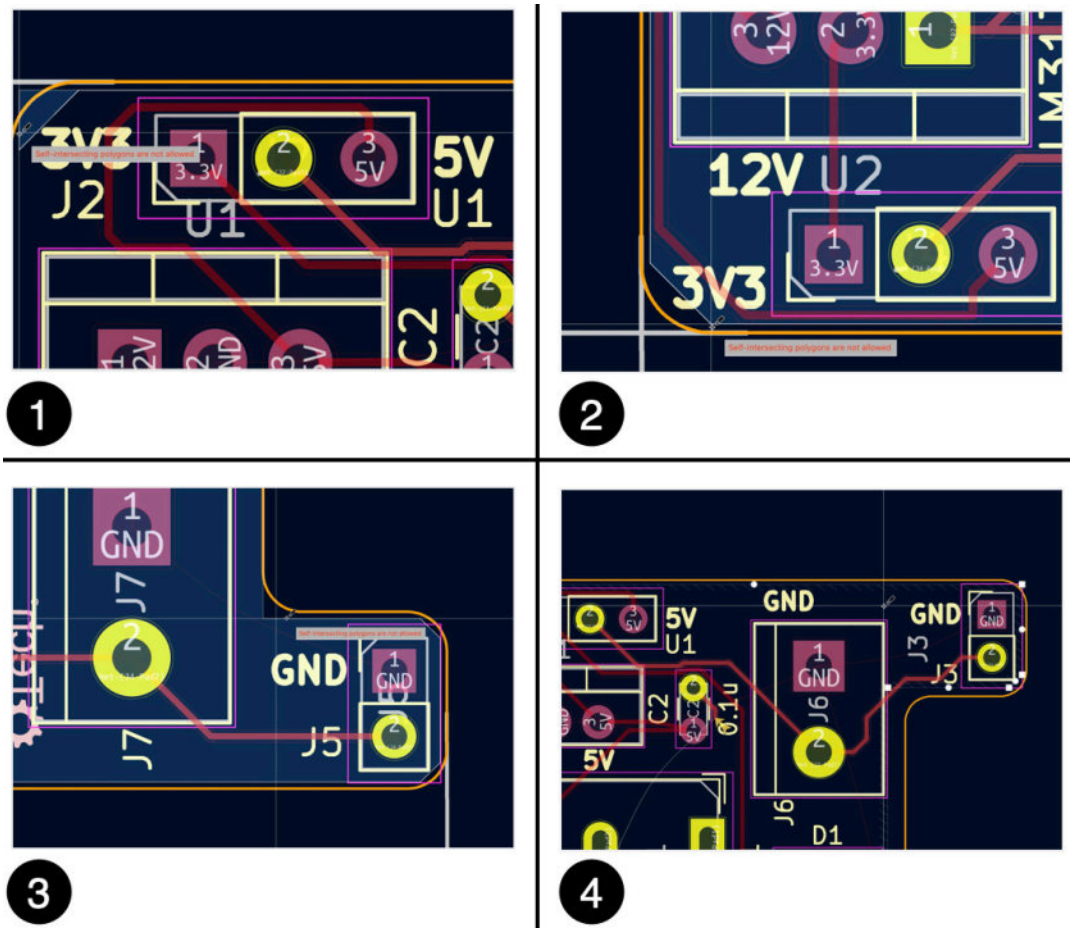


Figure 8.10.4: Drawing the new copper zone.

Once you have completed the drawing, the new zone is ready. However, it is depicted as an outline and not yet filled with copper. In the figure below, notice the hatched outline of the zone:

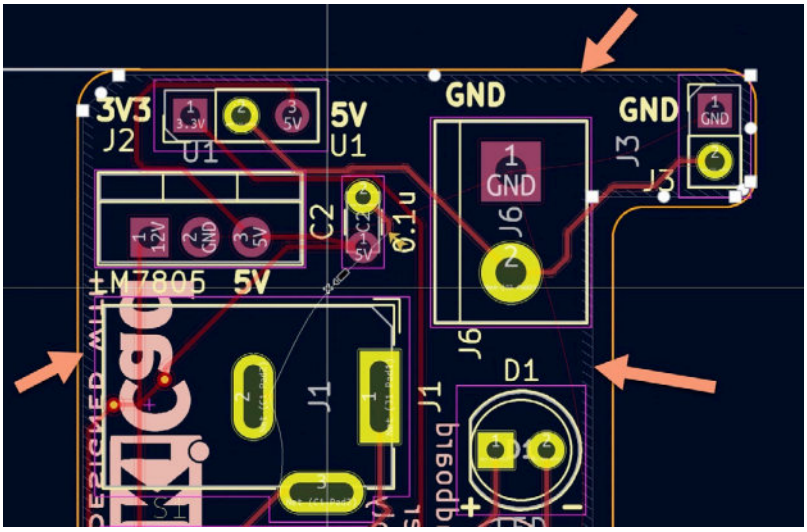


Figure 8.10.5: The new copper zone outline.

To fill the new zone with copper, right-click on the zone outline, then select Zones and Fill:

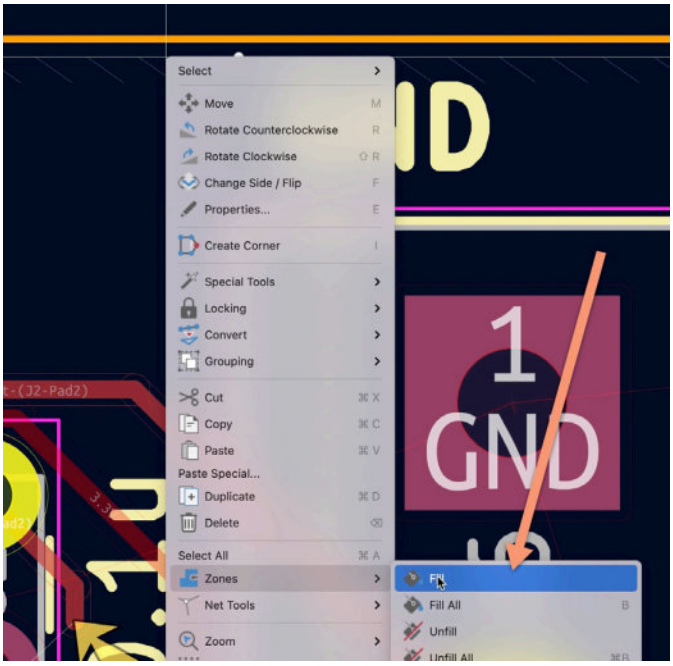


Figure 8.10.6: Fill the new zone with copper.

The new zone is now filled with copper:

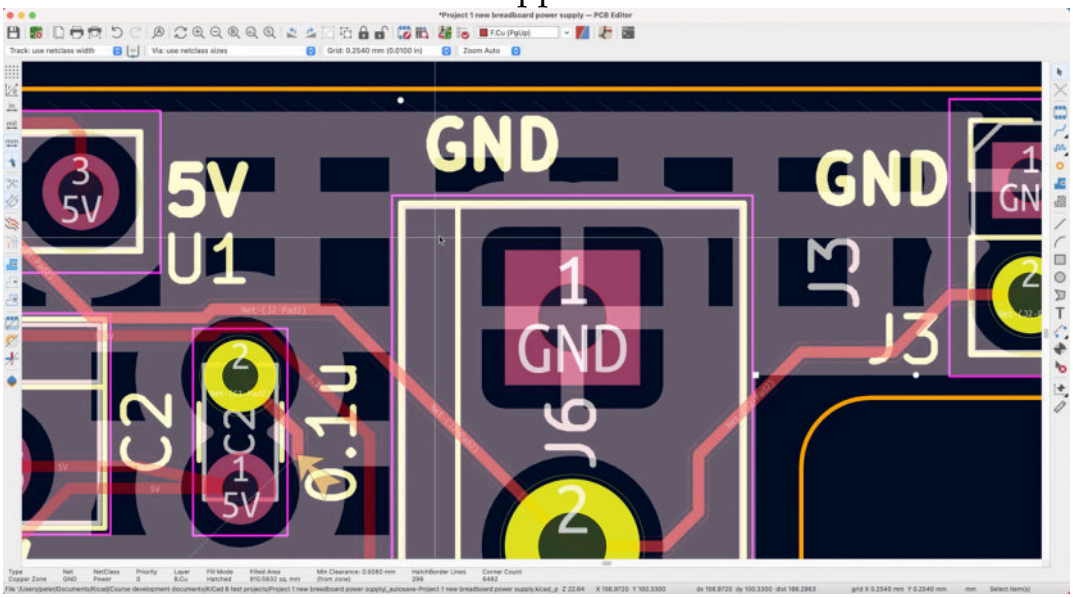


Figure 8.10.7: The new filled zone, completed.

To see the copper zone as it appears in the image above (i.e. with the hatched pattern visible), ensure that the button “Show filled areas of zones” is clicked and enabled. You will find this button in the left toolbar. In newer releases of KiCad 6, it seems that the default view mode is to show only the zone boundaries.

The copper fill that you see in the figure above uses the hatch pattern that I selected when I configured the fill (see Figure 8.10.3). You can make changes to the properties of an existing zone by double-clicking on its border to bring up the properties window. For example, I have changed the fill type to "solid fill," and the zone now looks like this:

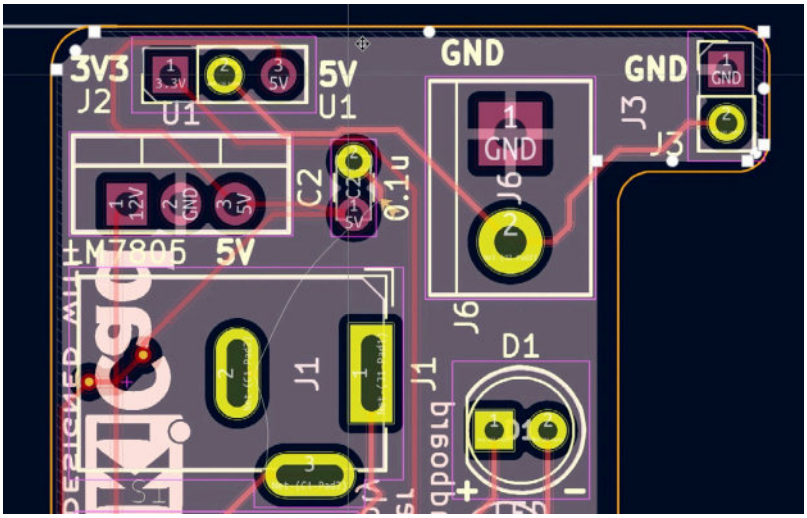


Figure 8.10.8: The new filled zone, with solid fill.

You can create independent filled zones in other copper layers or on the layer where another zone already exists (as long as there is available space). Just repeat the process I outlined above.

An extra benefit of using filled zones when connected to a net is that the filled zone will automatically connect and pads that belong to the same net. In the example above, I connected the filled zone to the GND net. The filled zone, then, will context any pads that belong to the GND net. This means that I don't have to draw copper tracks between GND pads; the GND-filled zone will take care of those connections.

Apart from filled zones, the layout editor has a tool for creating keep-out zones. A keep-out zone is a zone where footprints, vias, holes, and tracks are excluded. You can learn how to create a keep-out zone in the next chapter.

11. Keep-out zones

A keep-out zone is similar to a filled zone with one crucial difference: the purpose of a keep-out zone is to prevent elements such as footprints, vias, and copper tracks from being placed within its boundaries. Learn more about keep-out zones in an earlier dedicated chapter in this book.

I will create a keep-out area in the PCB of one of the projects in this book. To start, click on the keep-out area button from the right toolbar:

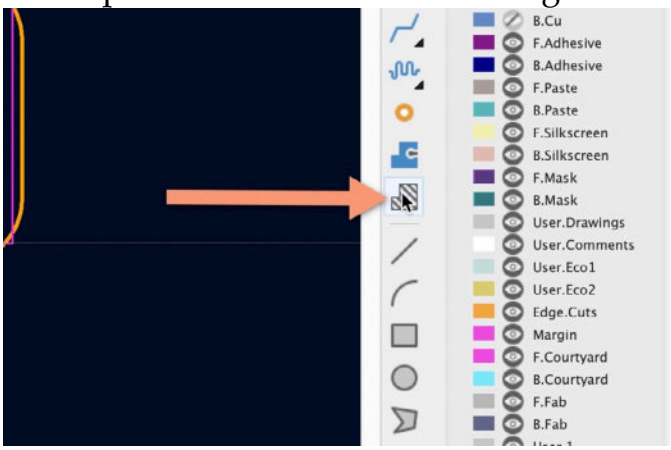


Figure 8.11.1: Creating a new keep-out zone.

The mouse cursor will change into a pen, just like it did when you created a filled zone. Click at the starting point of the keep-out area to bring up the zone properties window:

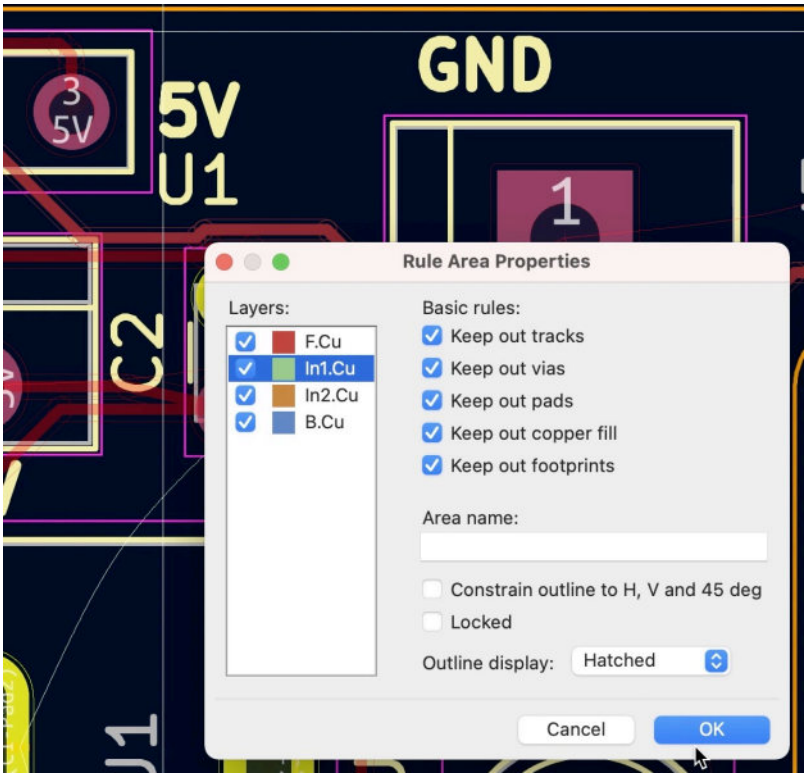


Figure 8.11.2: The new keep-out zone properties.

For this keep-out zone, I'd like to apply it in all copper layers and keep everything out (tracks, vias, pads, copper fills, and footprints). Set these preferences in the properties window, and click OK.

Like the drawing process for the filled zones, click to draw the zone's perimeter and double click to finish drawing a closed polygon.

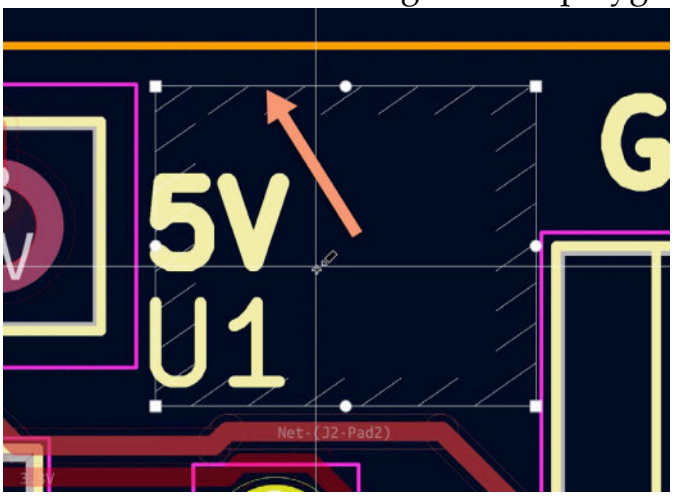


Figure 8.11.3: The new keep-out zone.

In Figure 8.11.3 (above), the new keep-out area is depicted with a hatched outline. This area is already active and keeping illegal objects out.

For example, say I try to drag an existing copper track into the keep-out area. I use the "D" hotkey to drag a track. In the example below, notice that

even though my cursor is within the keep-out area ("1"), the track I am dragging upwards is blocked at the boundary ("2").

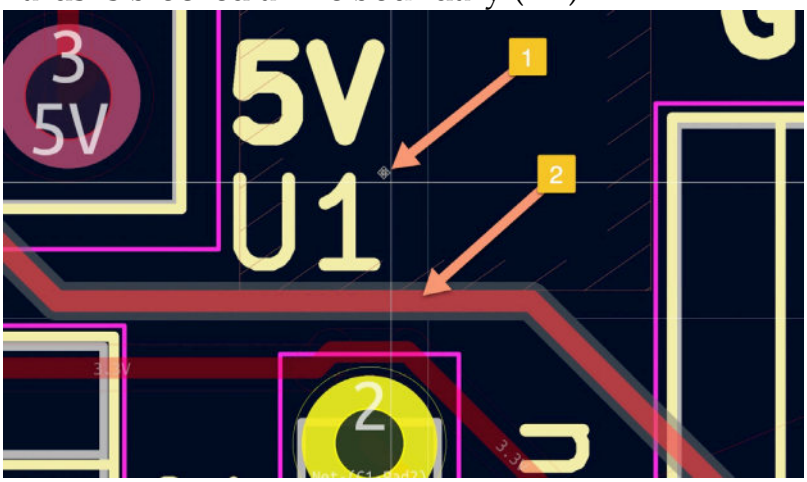


Figure 8.11.4: The new keep-out zone keeping tracks out.

You can repeat this experiment with footprints, vias, etc., to confirm that the keep-out area will prevent you from placing any such object within its boundaries.

12. Interactive router

The interactive router is the tool that helps you draw copper tracks. You use the interactive router every time you use the track or differential pairs tool. In this chapter, you will learn the fundamentals of the interactive router. See the example below:

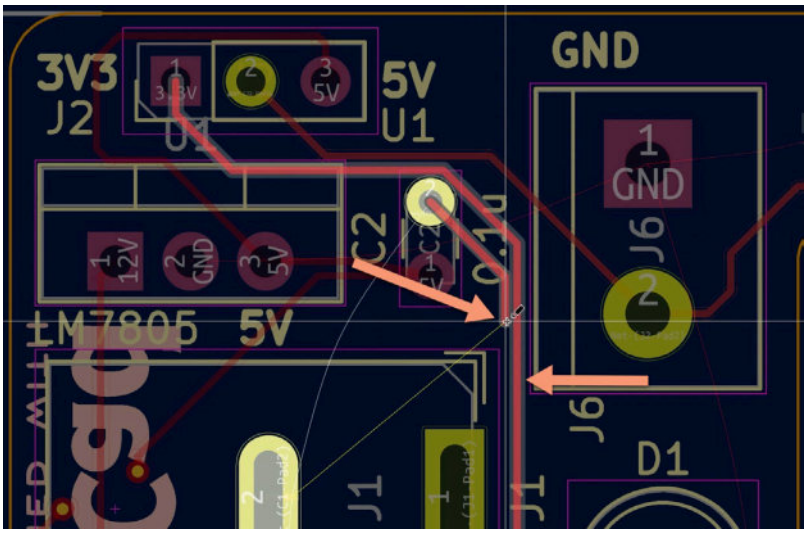


Figure 8.12.1: The interactive router in action.

In this example, I am drawing a new copper track from pad 2 of C2. I have drawn a segment of the new track against an existing track. Because of the interactive track mode, I have selected, the interactive router will "push" the existing track to make room for the new track. It will do this interactively in the sense that the router will reposition the existing track in relation to how I move the mouse as I am drawing the new track.

The interactive router is not perfect. The router will try to accommodate if you use your mouse to create a non-sensible path for the new track. However, the resulting track may often be full of unnecessary twists and turns. Over time, as you become familiar with how the interactive router works, you will find that it is a powerful tool for routing your boards.

The interactive router setup brings up the "Interactive Router Settings" window from the Route menu. You can see this window below:

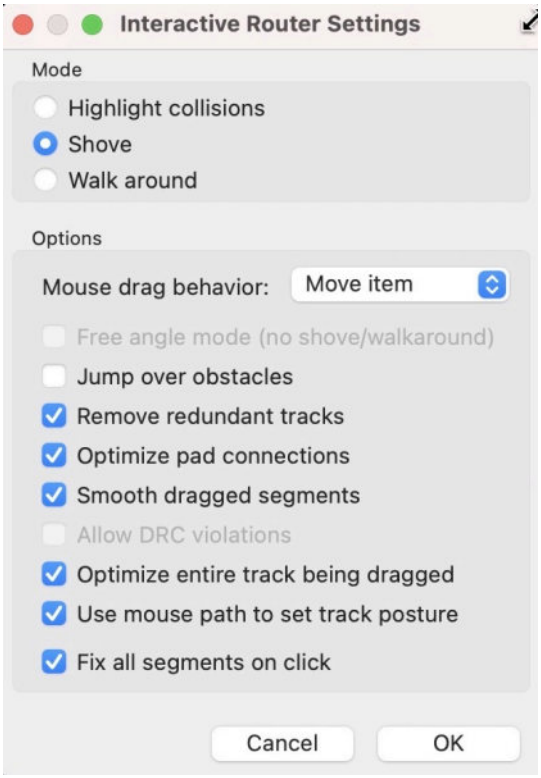


Figure 8.12.2: The interactive router settings.

You can choose between three modes:

1. Highlight collisions: this is the mode that gives most freedom but presents most risks. It allows violations (if you enable the "allow DRC violations" option) and highlights them. You have been warned.
2. Shove: this mode will move tracks and vias to make room for new tracks. It will not violate any design rules.
3. Walk around: this mode will not make any changes to the layout. It will find a route for the new track by going around existing elements. It will not break any design rules.

The options that are available in the "Options" group depend on the mode you have selected.

Let's look at examples of each mode.

Highlight collisions

Select the "Highlight collisions" mode in the interactive router settings, and click OK. I have also enabled "Allow DRC violations" for the sake of this example (I would not do this under normal circumstances").

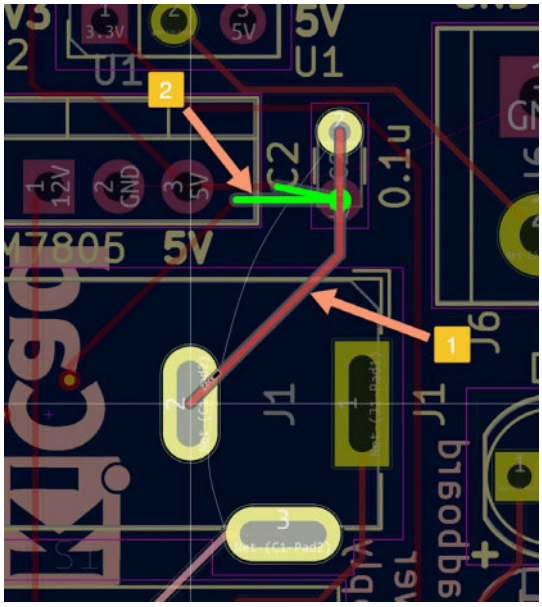


Figure 8.12.3: Highlight Collisions mode with Allow DRC Violations.

In the figure above, I have created a new track from pad 2 of C2 to pad 2 of J1. I drew this track over pad 1 of C2. This is a violation. The interactive router allowed me to do this because I enabled the "Allow DRC violations" option but used a bright green highlight to alert me of this violation.

I rarely use this mode and find the Shove or Walk Around more practical (and safe).

Shove

Select the "Shove" mode in the interactive router settings, and click OK. In the example below, I am drawing a new track from pad 2 of C2. I clicked on the pad to start drawing (left), then moved the mouse pointer towards the two existing tracks on the right side of pad 2, C2 (right). As you can see in the image, the interactive router changed the two existing tracks to help accommodate the new track as I drew it.

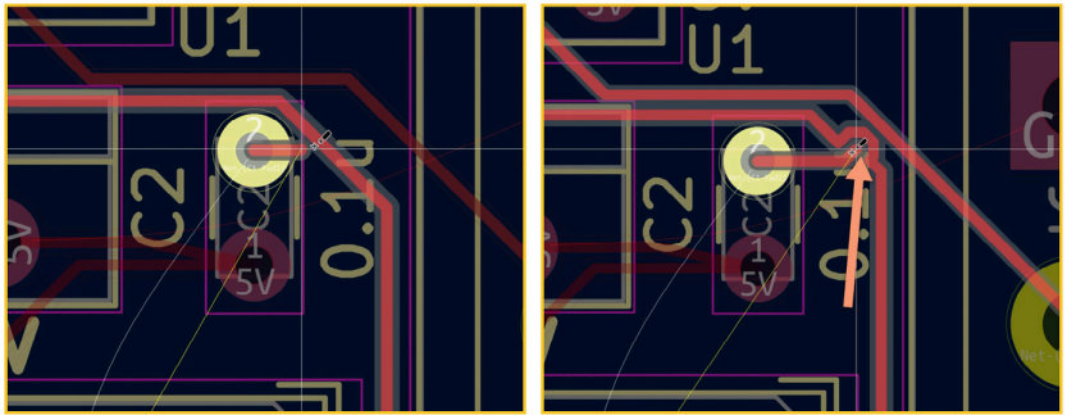


Figure 8.12.4: Shove mode.

If I move the mouse away from the two existing tracks, the interactive router will restore them to their original shapes and locations.

Walk around

Select the "Walk around" mode in the interactive router settings, and click OK. This mode will preserve the shape and locations of existing tracks, vias, etc., and look for viable paths (i.e., paths that do not violate the design rules). The interactive router will use input from the path that you are moving your mouse to infer that path that it will use to draw the new track. You don't have to click to provide input to the router. Simply moving the mouse after the first click that triggers the start of the drawing is sufficient. You can click to commit the path that the router has found and then continue with the same process to draw the next track segment.

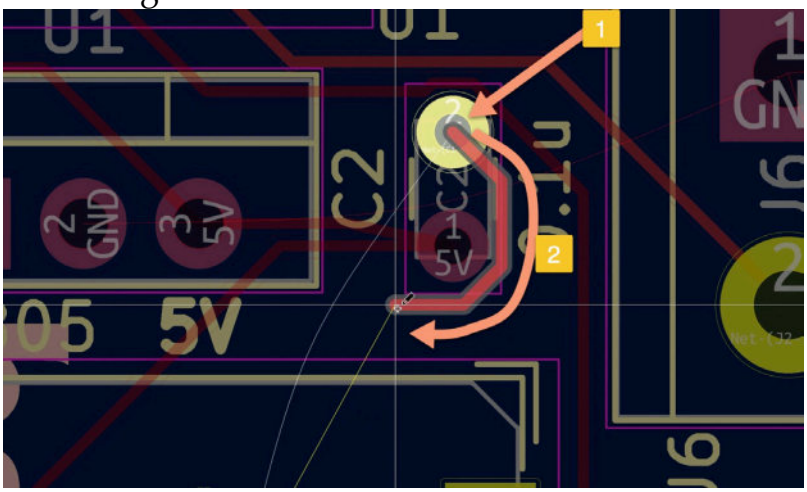


Figure 8.12.5: Walk around.

In the example above, I selected the walk-around mode for the router and then clicked on pad 2 of C2 to start drawing a new track. Without clicking, I moved my mouse around pad 1 to provide input to the interactive router (see path "2"). The interactive router used this input to draw the red track segment that you can see in the figure above.

In practice, you should use the Shove and Walk Around modes for most of your work with the interactive router. I have written a supplemental chapter on the interactive router as a recipe.

13. Length measuring tools

The layout editor provides two tools for making accurate length measurements. These tools will help you with the precise placement of footprint or other PCB elements on your board and precise board dimensions.

Figure 8.13.1: Measuring length in Pcbnew.

In the layout editor, you can access both tools from the right toolbar. With reference to the figure above, press button “1” to select one of four available length measuring tools that will add the measured dimension to the layout, or “2” to make an interactive length measurement between any two points in the layout.

Adding a length measurement to the layout

You can add a length measurement to the layout so that it is always visible. For this, you can select one of the User layers (i.e., “User.1”, “User.2,” etc.) and then select the appropriate measurement tool for your objective.

There are four tools to choose from:

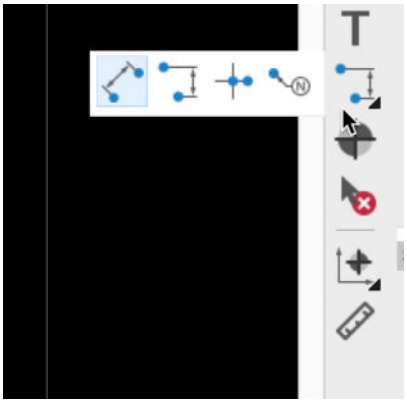


Figure 8.13.2: The four measuring tools.

From left to right:

1. Aligned linear dimension.
2. Orthogonal dimension.
3. Center dimension.
4. Leader dimension.

Below you can see an example of two measurements for the length between two points. One is an orthogonal dimension measurement that measures the orthogonal distance between two points in the layout. The other measurement returns a linear distance between the same points.

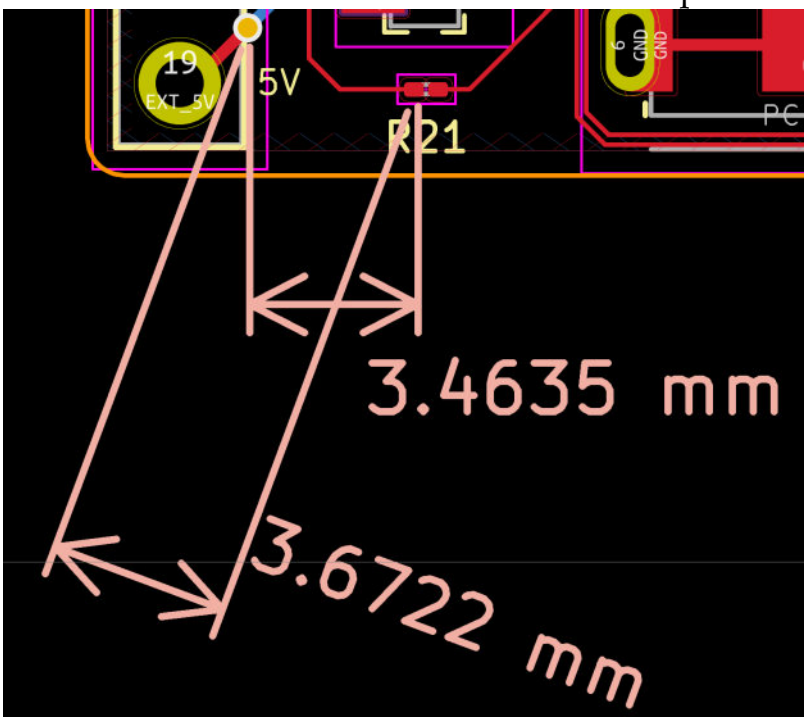


Figure 8.13.3: Orthogonal and linear distances between two points.

In the example below, I have placed linear measurements for the two sides of this PCB, and the pitch between the two rows of pins, in the User.2 and User.1 layers:

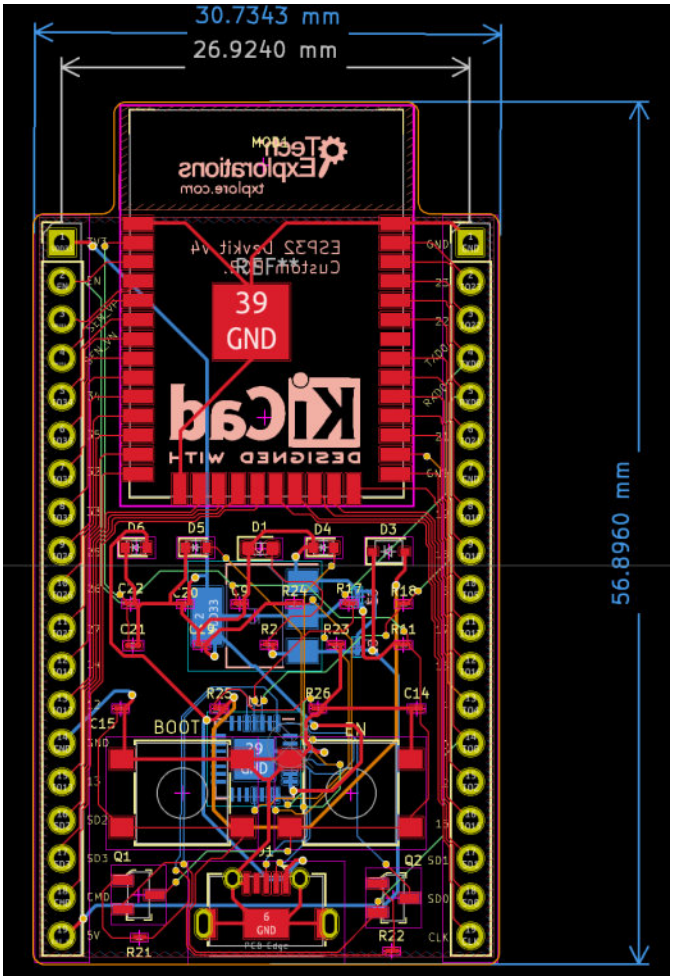


Figure 8.13.4: PCB dimensions measurements.

Interactive ruler

You can use the interactive ruler to make quick measurements that you don't need to remain in the editor. Select the tool from the right toolbar ("2" in Figure 8.13.1 above), then click anywhere to start measuring. As you move your mouse, the distance values change. You can click again to stop measuring. The last set of figures will remain until the next click triggers a new measurement.

You can see an example of the interactive ruler in operation below:

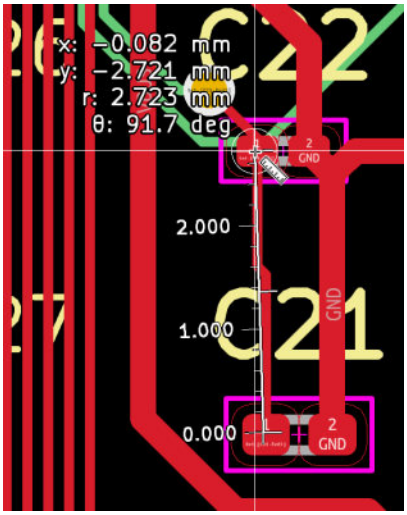


Figure 8.13.5: The interactive ruler in operation.

14. Bulk editing

The layout provides helpful tools for making bulk changes to your design. For example, you can use these tools to change the size of all silkscreen text or the thickness of all graphic lines in the User.1 layer. I will review those tools in this chapter. You can find these tools under the Edit menu.

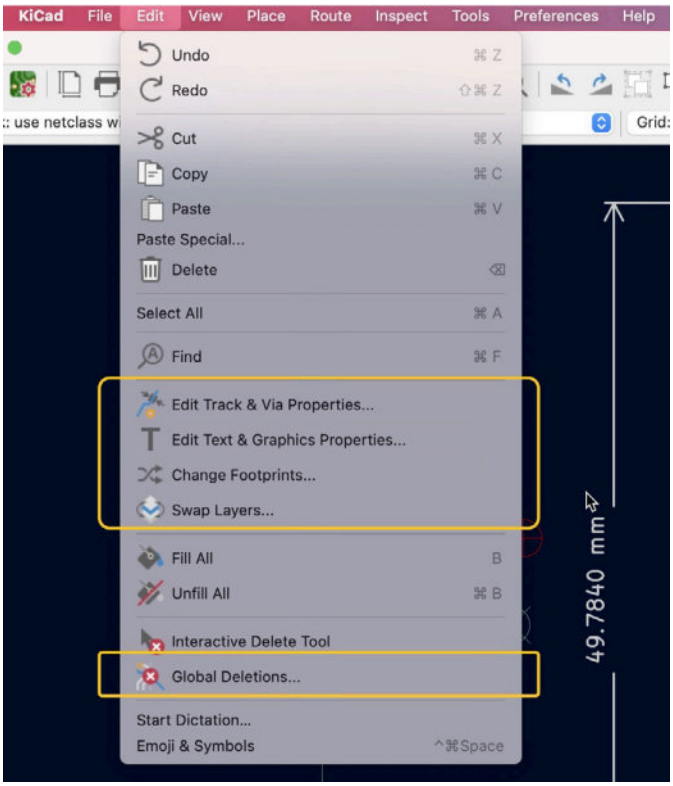


Figure 8.14.1: Bulk editing tools in Pcbnew.

The bulk editing tools to keep in mind are:

1. Edit Track & Via Properties.
2. Edit Text & Graphics Properties.
3. Change Footprints.
4. Swap Layers.
5. Global Deletions.

Edit Track & Via Properties

I have covered this tool in a dedicated chapter later in this book.

Edit Text & Graphics Properties

I have covered this tool in a dedicated chapter later in this book.

Change Footprints

I have covered this tool in a dedicated chapter later in this book.

Swap Layers

With the swap layers tool, you can take footprints from one layer and move them to another layer. Consider the red tracks in the board below:

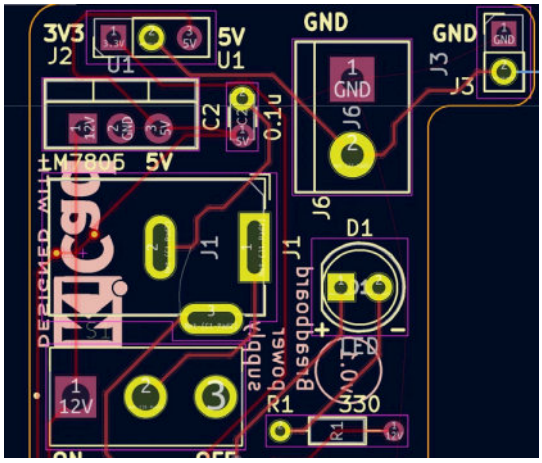


Figure 8.14.2: Moving the tracks from F.Cu to In1.Cu.

I want to move the red tracks (belonging to the F.Cu layer) to the In1.Cu. Bring up the Swap Layers window (under the Edit menu). In this window, use the dropdown menus in the right column to select the target layer.

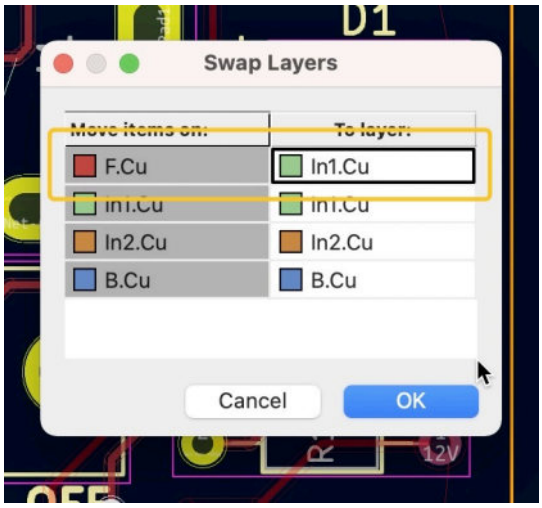


Figure 8.14.3: Moving the tracks from F.Cu to In1.Cu.

In this example, I want to move the tracks from the front copper layer to the first inner copper layer (In1.Cu). So, I have selected “In1.Cu” from the dropdown menu in the first cell of the second column (see above).

Click OK to commit the change. The editor will make the change and use green to depict the track that now exists in In1.Cu:

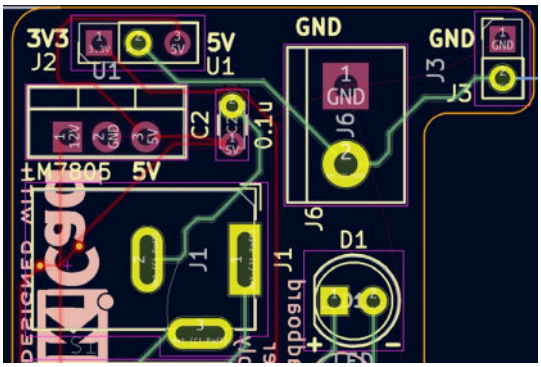


Figure 8.14.4: New tracks in In1.Cu (moved from F.Cu).

Changes like this are very easy using the Swap Layers tool.

Global Deletions

With the Global Deletions tool, you can quickly remove all elements of the same kind. Let’s look at an example. Consider this PCB:

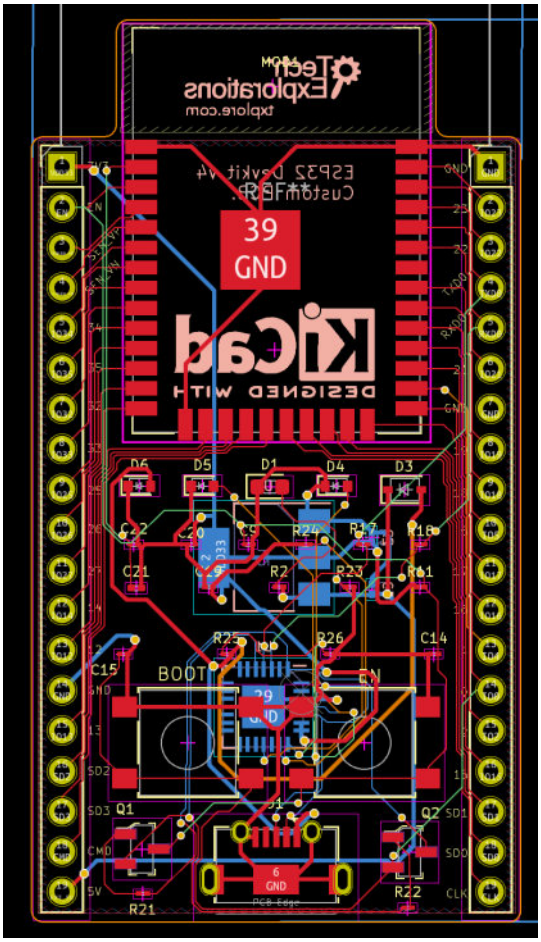


Figure 8.14.5: I will delete all tracks and vias.

I want to delete all tracks and vias as part of a redesign of the board. To do this quickly, I will use the Global Deletions tool. Invoke the tool from the Edit menu. You can see the tool window below with my settings.

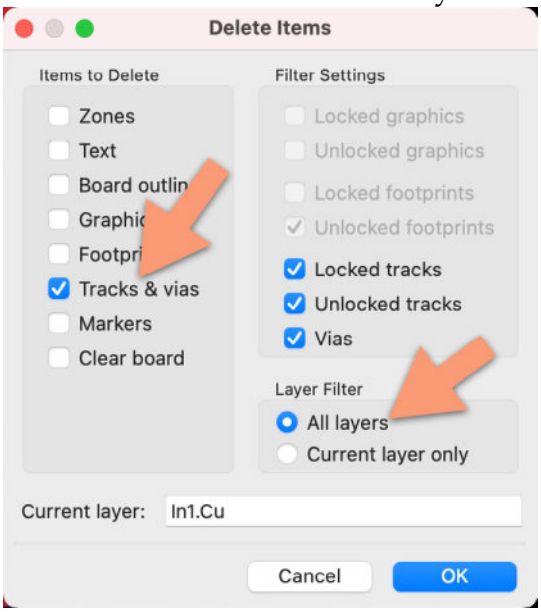


Figure 8.14.6: The Delete Items window.

I want to delete all tracks and vias, regardless of their layer, so I have selected “All layers” under “Layer Filter.” Click OK to dismiss the window, and again OK to dismiss the warning. The result is below:

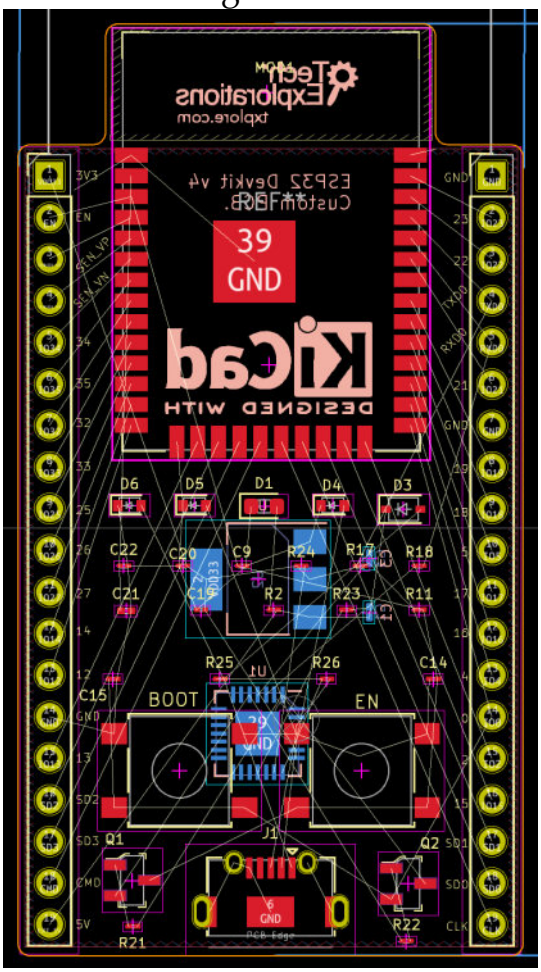


Figure 8.14.7: I have deleted all tracks and vias.

All vias and tracks are deleted. To bring them back, you can use Ctr-Z/ Cmd-Z (undo).

The Global Deletions tool is handy in a range of situations. In my experience, I found that I make frequent use of this tool when I need to redesign an aspect of the design, such as removing zones or changing the position of major components, which then require drawing new tracks.

15. Create a custom footprint, introduction

In previous chapters in this part of the book, you learned how to find a required footprint in KiCad's own libraries or on the Internet, and how to use it in your PCB. But, what if you can't find what you need? In that case, you can create a custom footprint.

In this chapter, you will learn how to create a footprint using KiCad's footprint editor app. You can access the footprint editor from KiCad's main project app, Pcbnew, and Eeschema (see below):

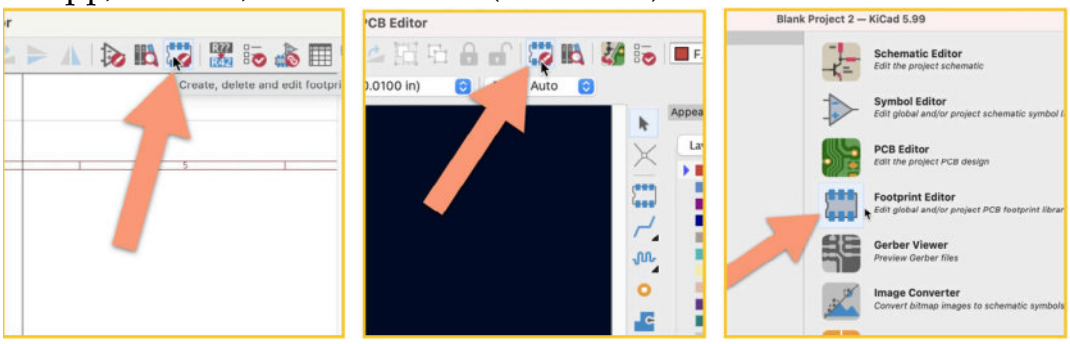


Figure 8.15.1: Starting the Footprint Editor.

Use any of those options to start the Footprint Editor. You can see the footprint editor below. The footprint editor looks very similar to the layout editor, so you should already be familiar with it:

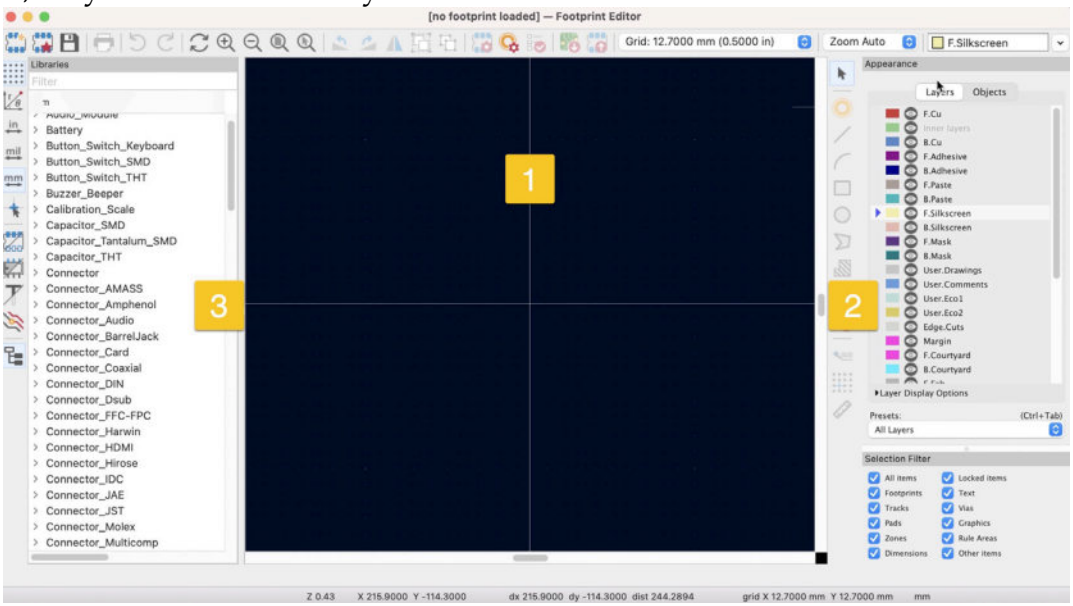


Figure 8.15.2: The Footprint Editor.

In the figure above, I have annotated the three main areas of the footprint editor window. In the middle ("1") is design area where you will be drawing a new footprint. On the right side ("2") is a composite toolbar that contains the drawing tools, the Appearance widgets, and the selection filter. This toolbar is almost identical to its counterpart in Pcbnew. The main difference is that the drawing buttons in the footprint editor contain a subset of those in Pcbnew. On the left side, you will find the Libraries browser ("3"). Use this browser to find an existing footprint and then modify it into the design editor.

With the footprint editor, you can design any footprint. As long as it has a perimeter and pads, you can design it. Some footprint designs are relatively common, such as BGA, QFP, DIP, and QFN components. To expedite the creation of standard footprints such as those mentioned above, the footprint editor includes a footprint generator tool (also known as the "footprint wizard"). This tool can help you quickly generate a footprint before continuing in the footprint editor to do further customization. You can learn how to use the footprint generator in a dedicated chapter in the Recipes part of this book.

In this chapter, you will learn how to create a new footprint from scratch. In an earlier chapter in this book, you learned how to make a new symbol. Specifically, you created a symbol for the NE555 timer integrated circuit. In this chapter, you will continue this work and create the corresponding footprint for the symbol. You will need information about the electrical and mechanical characteristics of the footprint. The best source for this information is the component's datasheet. Below you can see part of a page from this datasheet that contains the mechanical design information we need for the footprint.

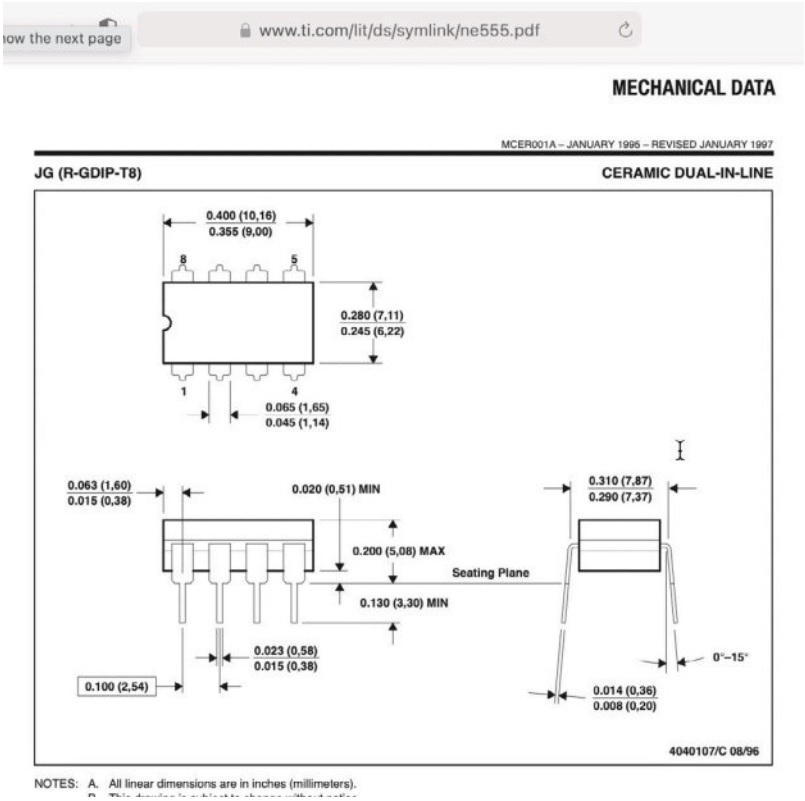


Figure 8.15.3: Mechanical data for the DIP package from the datasheet.

Keep this datasheet readily available as you will need it during the drawing process.

The process of creating a new footprint contains four steps:

1. Create a new blank footprint project in the footprint editor.
2. Draw the outline of the footprint in the fabrication layer.
3. Add the pads.
4. Draw the footprint outline in the courtyard layer.
5. Finish the process by drawing graphics and text in the silkscreen layer.
6. Save the new footprint, and use it.

Let's begin this process now.

15.1. Create a new library and footprint

Let's start the process of creating a new footprint. Because I'll be drawing the new footprint from scratch, I don't need the library browser in the left pane. I will remove it and increase the available space in the editor. To remove the library browser, click "Show footprint tree" from the View menu. This is a toggle option, so it will make the browser disappear if it is visible.

You are now working with the footprint editor window that looks like this:

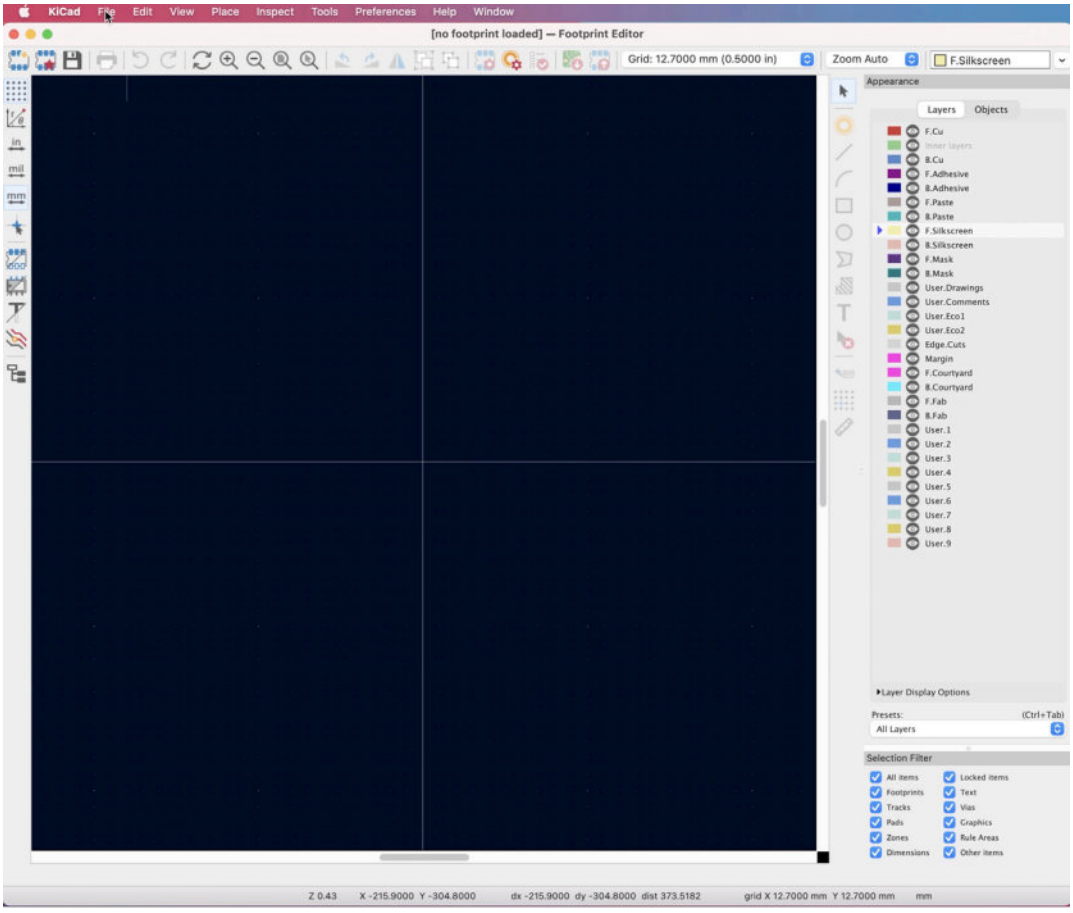


Figure 8.15.1.4: The blank footprint editor window.

I will demonstrate the entire process that involves creating a new library to store the new footprint. To create a new library, choose "New Library" from the File menu:

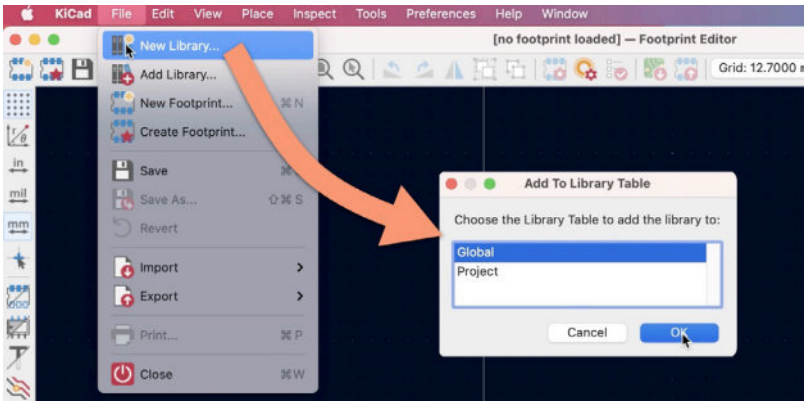


Figure 8.15.1.5: Creating a new library.

I will allow Global access to the new library, so select "Global" in the window that appears and click OK. Select the location for the new library a click Save to close the file browser window.

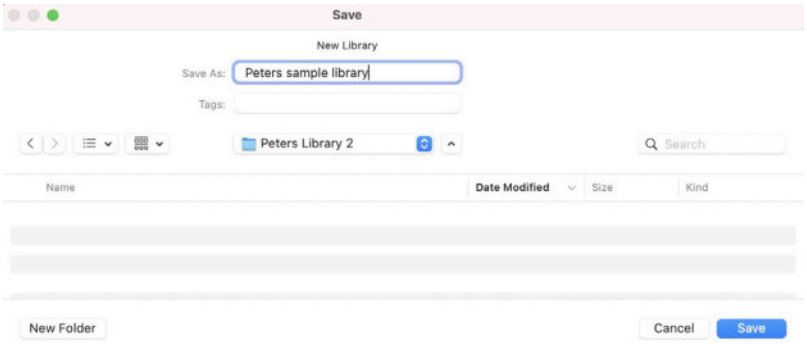


Figure 8.15.1.6: The location of the new library.

I have created a new library but have not yet selected it to contain the new footprint. To do so, enable the footprints tree pane from View, Show Footprint Tree, and use the filter to find the new library by name. In the example below, I searched for "Peter" in the filter ("1") and then selected the library I recently created ("3"):

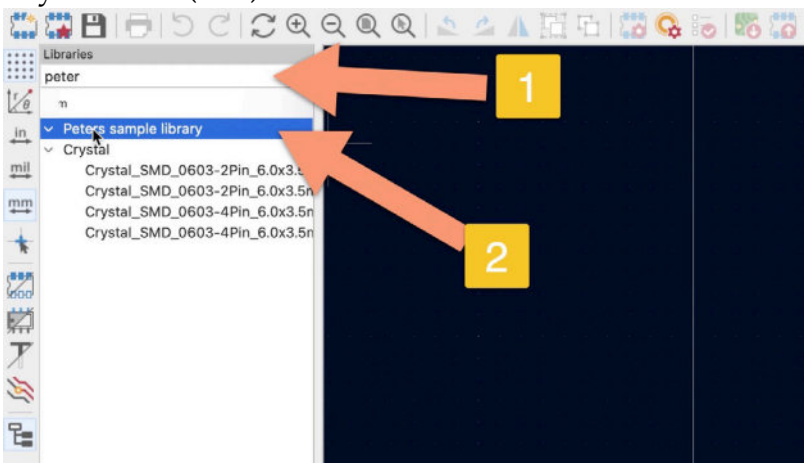


Figure 8.15.1.7: Selected the new library to contain the new footprint.

Time to create the new footprint. Right-click on the library row, and select "New footprint":

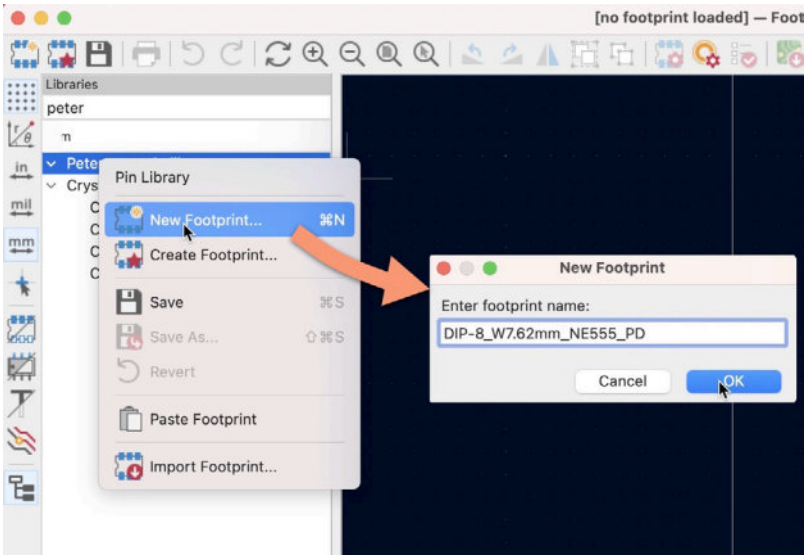


Figure 8.15.1.8: Creating a new footprint in the new library.

Give the new footprint a descriptive name. This will make it easier to find it among thousands of other footprints later. In the name, include:

- The type of package (i.e., "DIP").
- The number of pins (i.e., "8").
- The dimensions (i.e., "W7.62mm").
- The model of the component (i.e., "NE555").

I also like to add my initials "PD" to make it easier to distinguish my custom-made footprint among others.

Click OK, and confirm that you can see the new footprint under the library in the footprint tree, and text with the footprint name in the editor pane:

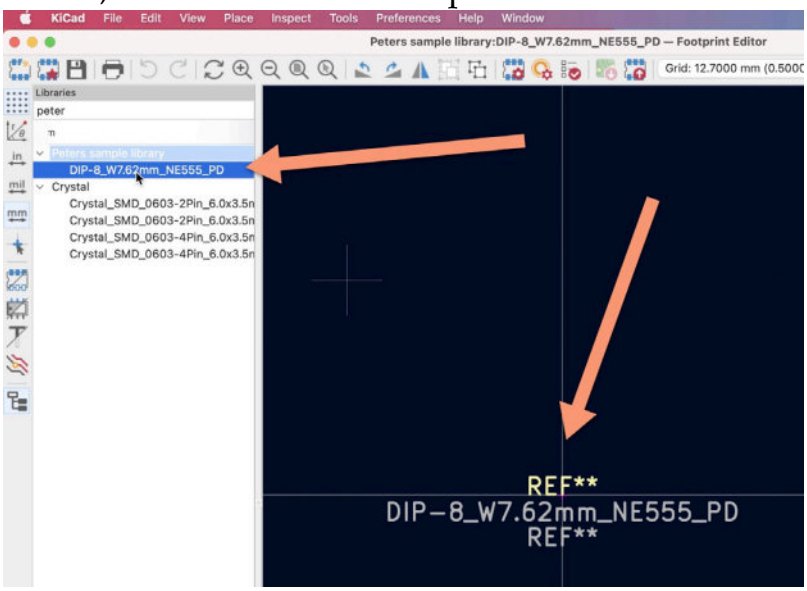


Figure 8.15.1.9: New footprint created.

Now that I have created the new footprint and stored it in a library, I no longer need the library tree pane to (again) remove it and reclaim additional space in the designer.

I will continue with the drawing process in the next section and work on the fabrication layer.

15.2. Create a footprint, 1, Fabrication layer

The first step is to draw the outline of the footprint in the front fabrication layer (F.Fab). Essentially, I will be drawing the component's border as I see it in the datasheet (see below).

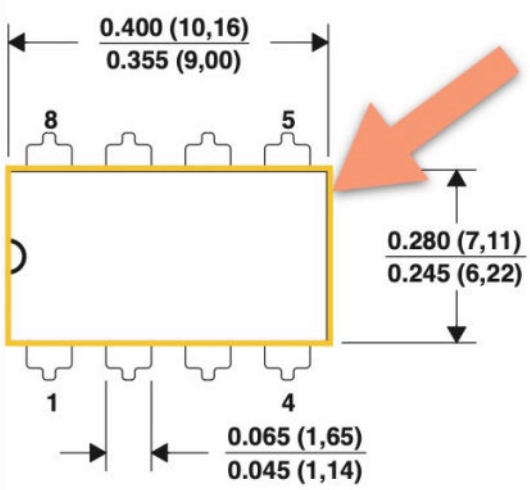


Figure 8.15.2.10: The outline of the footprint will go in the F.Fab layer.

I will use the rectangle tool from the right toolbar to draw the outline of the footprint. With reference to the figure below, select an appropriate grid to help you draw a rectangle close to the mechanical dimensions you see in the datasheet. The datasheet shows that the package is 10,16mm by 7,11mm, so I have chosen a grid of 0.127mm ("1").

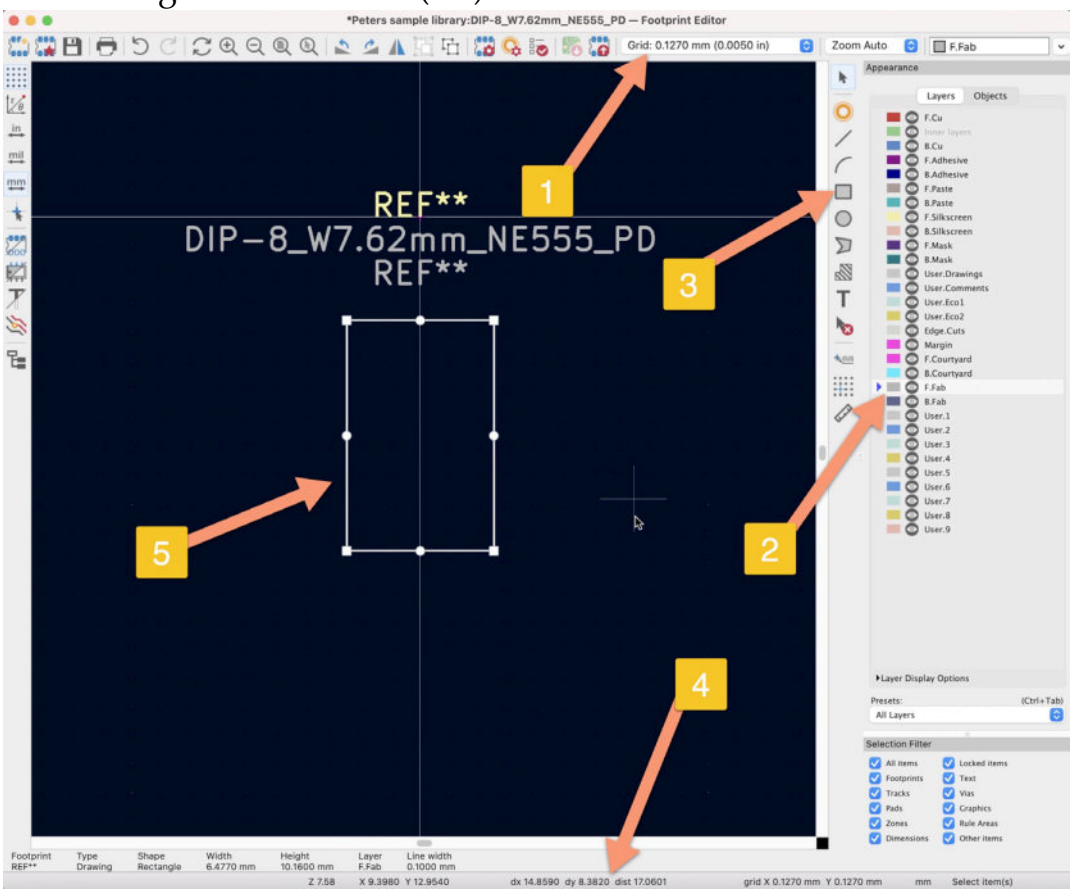


Figure 8.15.2.11: The outline of the footprint in f.Fab is complete.

Next, select the "F.Fab" layer from the Layers tab (under Appearance, on the right side, "2"), and choose the rectangle tool from the toolbar ("3").

Use the "dx" and "dy" values in the status bar (bottom of the window, "4"), and press the space bar to reset them to zero when needed. Start drawing the rectangle so that the final outline looks like the one in the example above. I use the mid-point crosshairs of the editor to help me place the rectangle in the middle along the Y-axis. You can use the handles in the corners and center of the rectangle to resize it. You can also move it (select the rectangle and click-hold to move it).

Once you have a rectangle approximately equal to the size indicated in the datasheet, the work is complete. You can continue in the next segment where you will add the pads.

15.3. Create a footprint, 2, Pads

Let's continue with the pads. The pads don't exist in a specific layer; they are entities that span across all layers. Therefore, it doesn't matter which layer is active in the Layers tab. However, what is very important is to determine the geometrical characteristics of the pads: their shape, position relative to the footprint perimeter in the fabrication layer, and relative to other pads. Also important are the pin numbers and names.

As always, all of this information is available from the datasheet. Below, I have circled the dimensions that are relevant to the work I'm about to do:

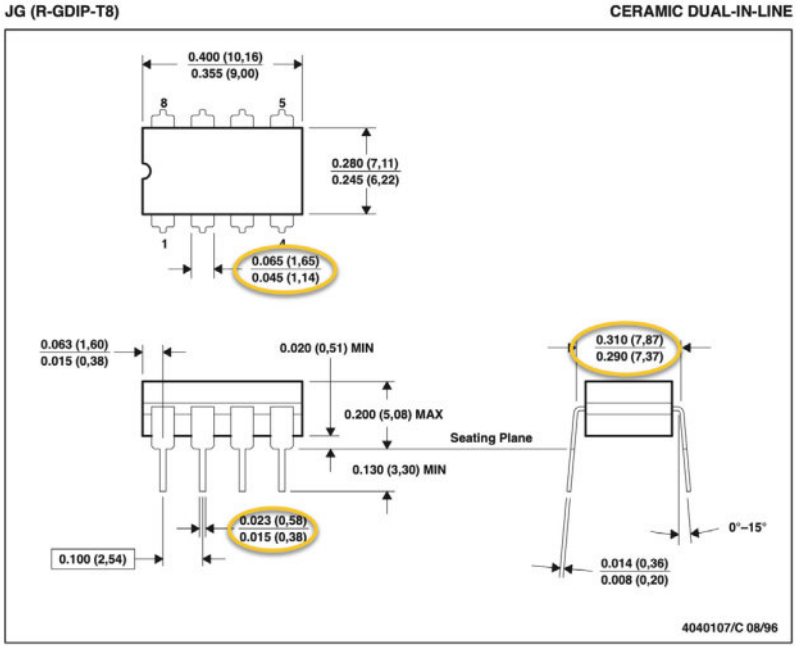


Figure 8.15.3.12: Important data relating to pins.

Choose a reasonable grid size that makes it easy to work with the pin sizes. I find that a 1.27mm grid size works well. Select the pad tool from the right toolbar (see figure below), and place the first pad (pad 1) close to the top left corner of the perimeter. After you click to add the first pad, press the space bar to reset the dx and dy values in the status bar ("2"). Move the mouse down until dy shows 2.54 mm, and click again. You will see a second pad added, marked as "pad 2". The distance between pads 1 and 2 is 2.54mm, equal to the pad pitch as indicated in the datasheet.

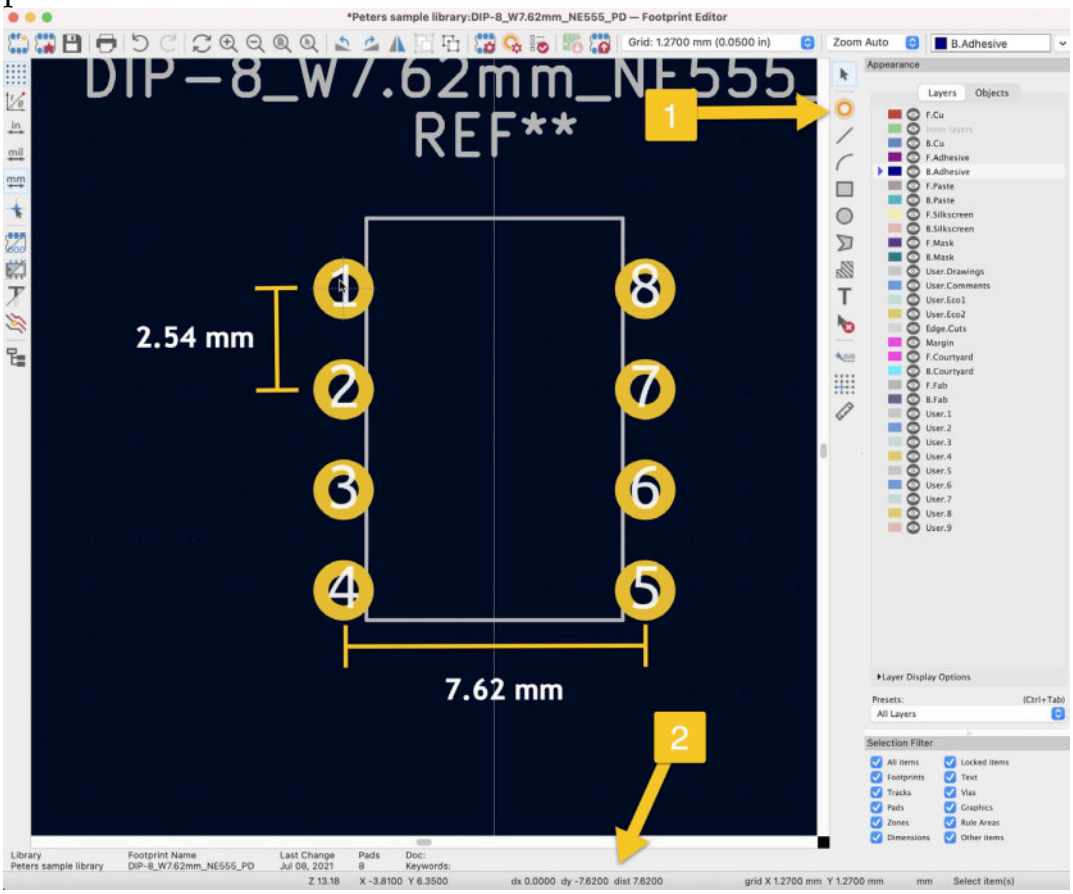


Figure 8.15.3.13: Added pads.

Continue in the same fashion with pads 3 and 4. Each time, reset the dx and dy counters to ensure the correct distances.

You now need to create pad five on the opposite side. According to the datasheet, the distance between pads 4 and 5 is between 7.37 mm and 7.87 mm. After creating pad 4:

1. Reset the dx and dy counters, and move the mouse towards the right.
2. When the dx value reaches 7.62 mm (and dy remains 0mm), click again to add pad 5.

3. Continue upwards to add pins 6, 7, and 8, always maintaining a distance of 2.54 mm.

The footprint now looks like the example in Figure 8.15.3.13. Compare the pin numbers and positions in the footprint against the datasheet and make sure they match.

I can see an improvement here because the pads are not appropriately entered against the perimeter; they are slightly towards the bottom end. To fix this, I will move the rectangle in the F.Fab layer downwards. To give me more positioning control, I have changed the grid size to 0.635mm. You can see the footprint after I move the rectangle in the figure below:

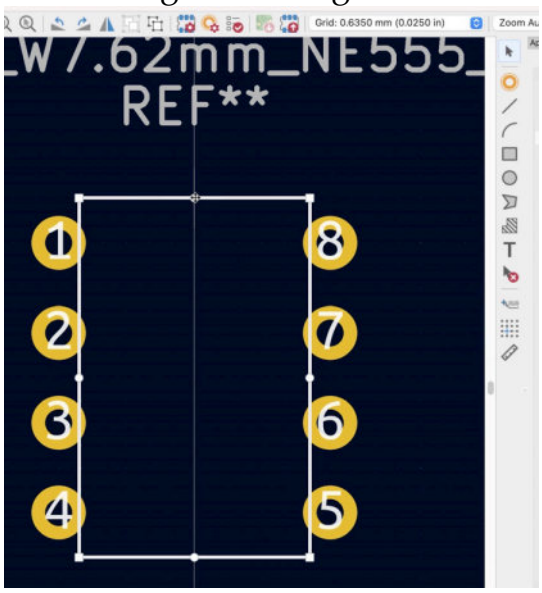


Figure 8.15.3.14: Improved the position of the rectangle relative to the pads.

I want to change the shape of pad one so that it stands out against the rest. This can help me determine pin one without needing silkscreen markings, but only using the shape of the pad as a guide. To do this, double-click on pad one to bring up its properties. In the example below, I have selected "rectangular" for the "Pad shape" property:

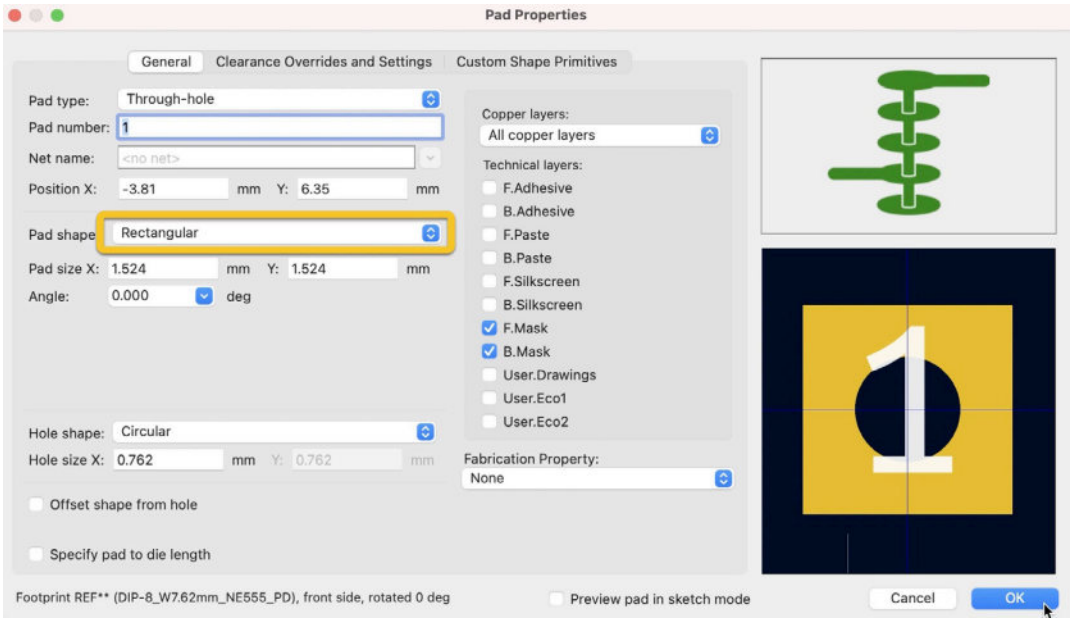


Figure 8.15.3.15: Pad 1 is rectangular.

You can edit other pad properties, such as the pad size, hole shape, and even add an offset of the pad against the hole. You can also set a net name for this pad to match the net of a pin from a corresponding symbol.

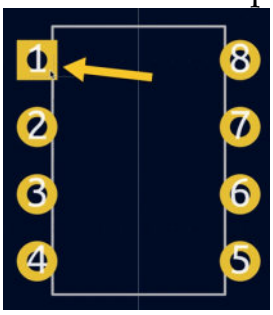


Figure 8.15.3.16: The footprint with a rectangular Pad 1.

Let's continue the process with work in the courtyard layer.

15.4. Create a footprint, 3, Courtyard layer

The core of the new footprint is ready, but there are a couple of elements that a good footprint should also have. The first one is a border in the courtyard layer that marks the external boundary of the footprint. KiCad's DRC uses this boundary to detect when another element (such as a track or part of another footprint) encroaches within the reserved space of the footprint.

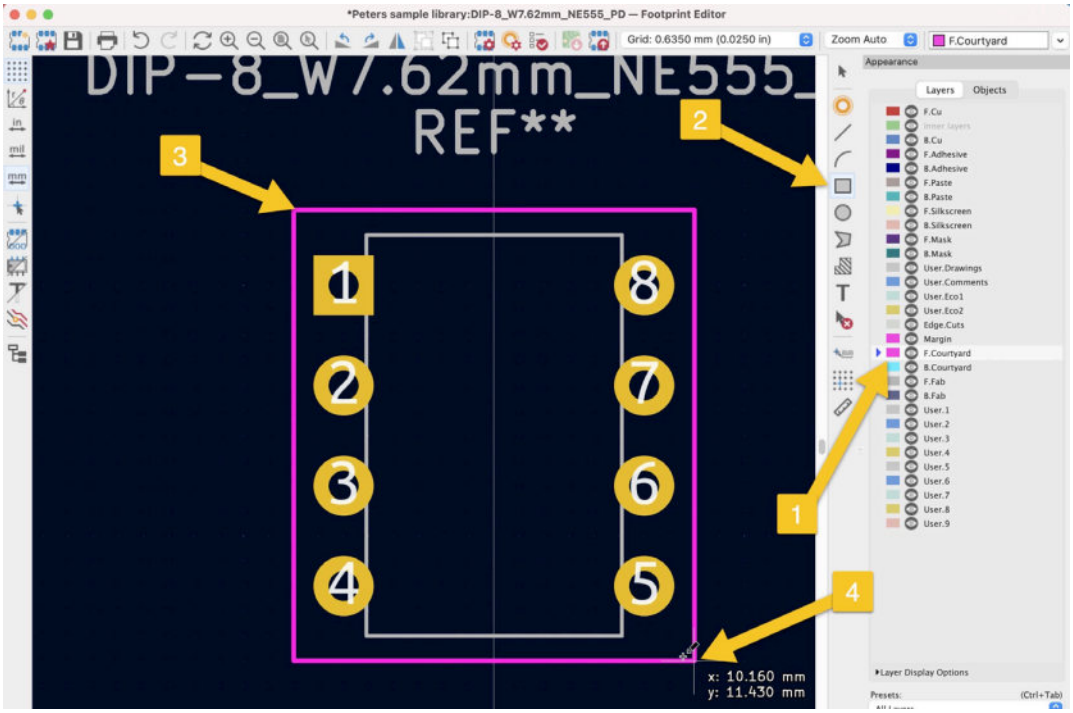


Figure 8.15.4.17: Drawing a rectangle in the F.Courtyard layer.

Save the footprint and continue in the following (and final) step, where you will add the finishing text and graphs to the silkscreen layer.

15.5. Create a footprint, 4, Silkscreen layer

The final step of the footprint creation process is to add text and graphics in the silkscreen layer. I will add a few simple graphical elements in the front silkscreen layer ("F.Silkscreen"). These elements will appear in your PCB when you import the footprint.

Enable the F.Silkscreen layer from the Layers tab, and change the grid size to 0.254 mm. I will add a few lines along with the corners of the footprint's fabrication layer perimeter and a small circle next to pad 1. From the right toolbar, I will use the line and circle tools for these graphics.

You can see the result below:

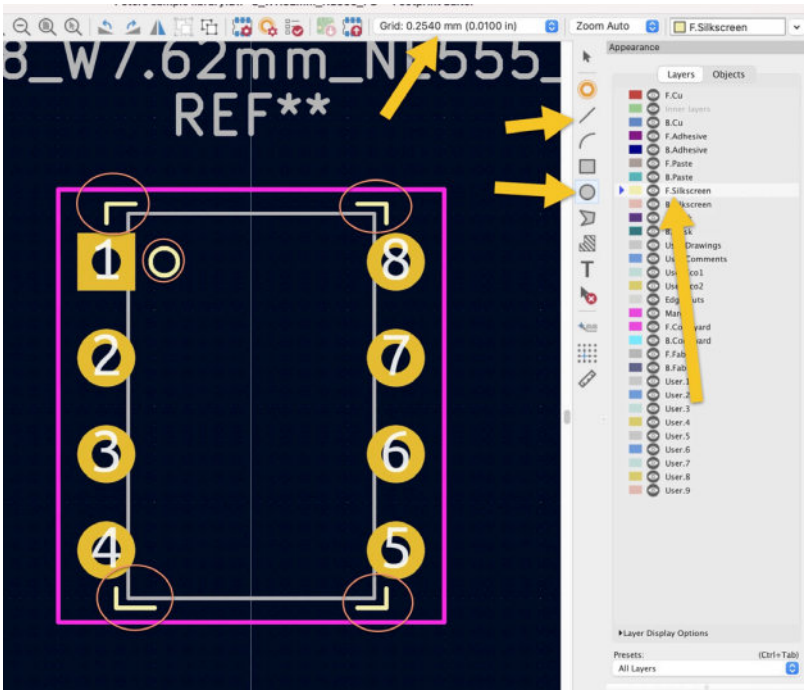


Figure 8.15.5.18: The completed footprint, including the silkscreen graphics.

The footprint editor also has a 3D viewer. You can find it under the View menu. Below I have used it to render my new footprint in 3D:

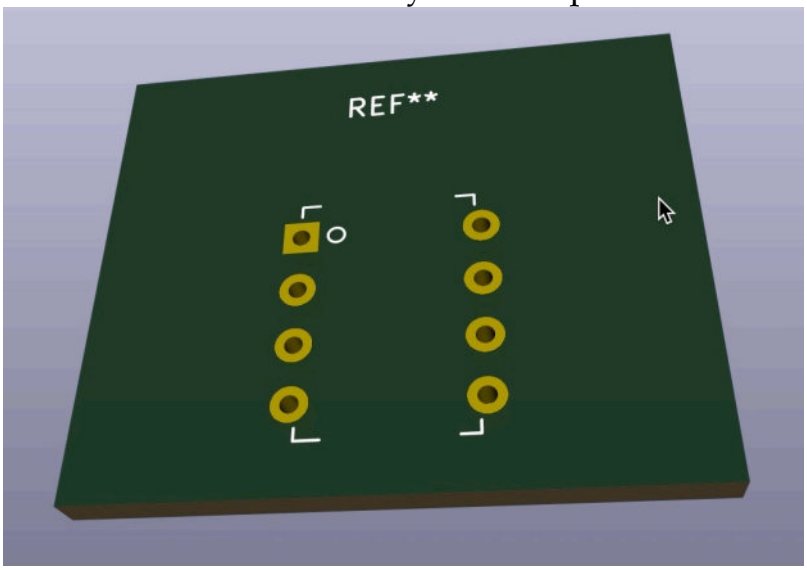


Figure 8.15.5.19: My new footprint, in 3D.

Exit the 3D viewer and save the footprint. In the next section, I'll use the new footprint in the layout editor.

15.6. Use the new footprint

Time to use the new footprint. If you are still in the footprint editor window, enable the footprint tree and confirm that you can find your new library and footprint. Here is mine:

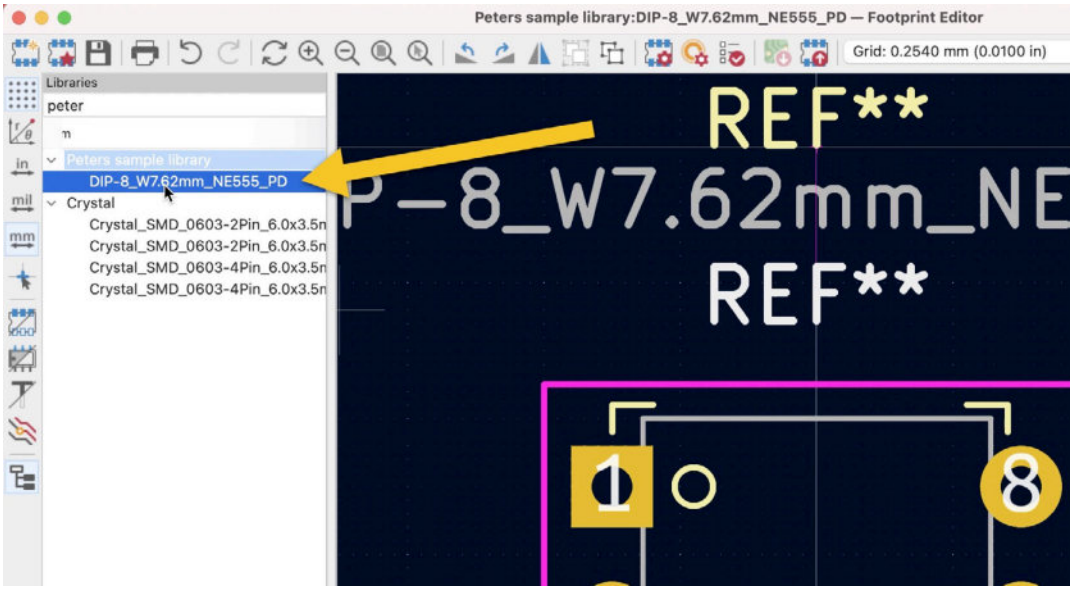


Figure 8.15.6.20: I can find my new library and footprint in the footprint tree.

Close the footprint editor, and return to the layout editor. To confirm that the new library is ready to use, open the Footprint Libraries window (under Preferences), scroll to the bottom of the libraries table, and confirm that your new library is already listed there. The footprint editor did take care of this. Close the footprint library window.

Type the "O" hotkey to bring up the footprint browser, and use the filter to search for the new library. I have typed in the first part of the library name, as you can see below:

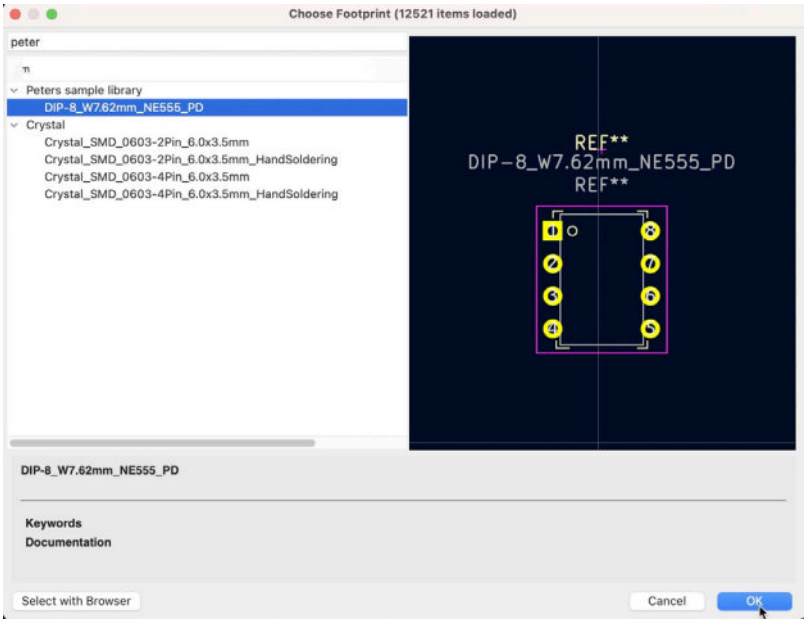


Figure 8.15.6.21: Searching for my new library in the footprint browser.

Double-click on the only footprint in the new library, and add it to the editor:

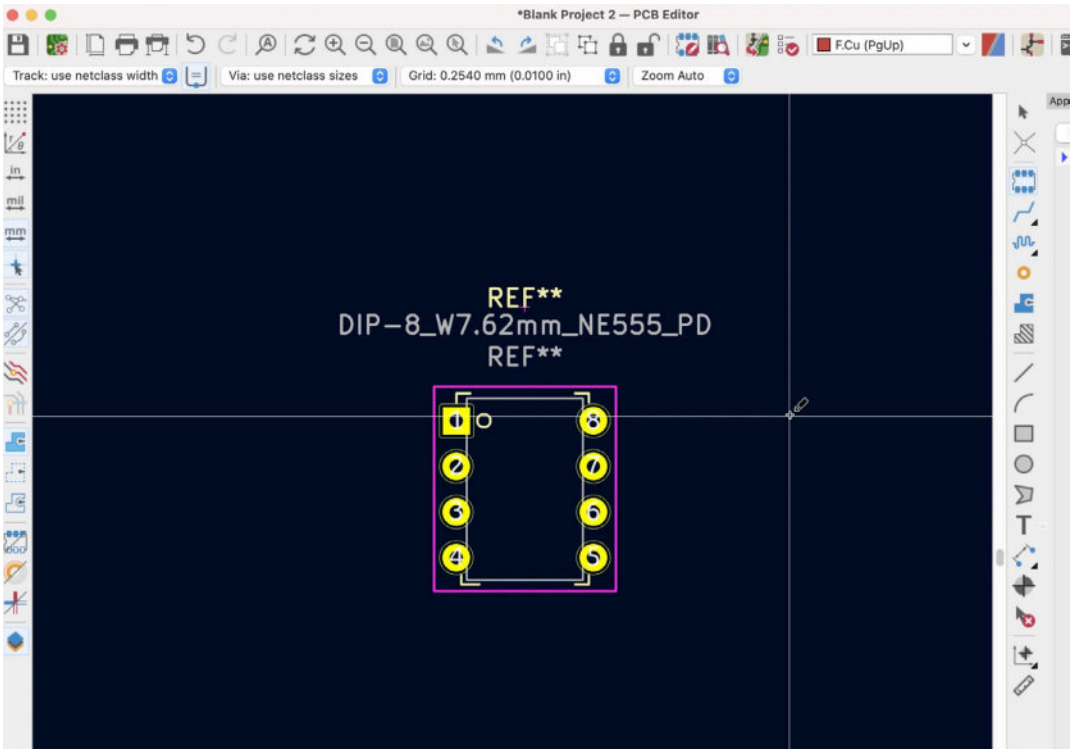


Figure 8.15.6.22: My new footprint in the layout editor.

My new footprint is now in the layout editor, and I can use it in my project as any other footprint.

16. Finding and using a 3D shape for a footprint

The layout editor in KiCad contains a 3D viewer that can render your design in 3D. Below is an example:

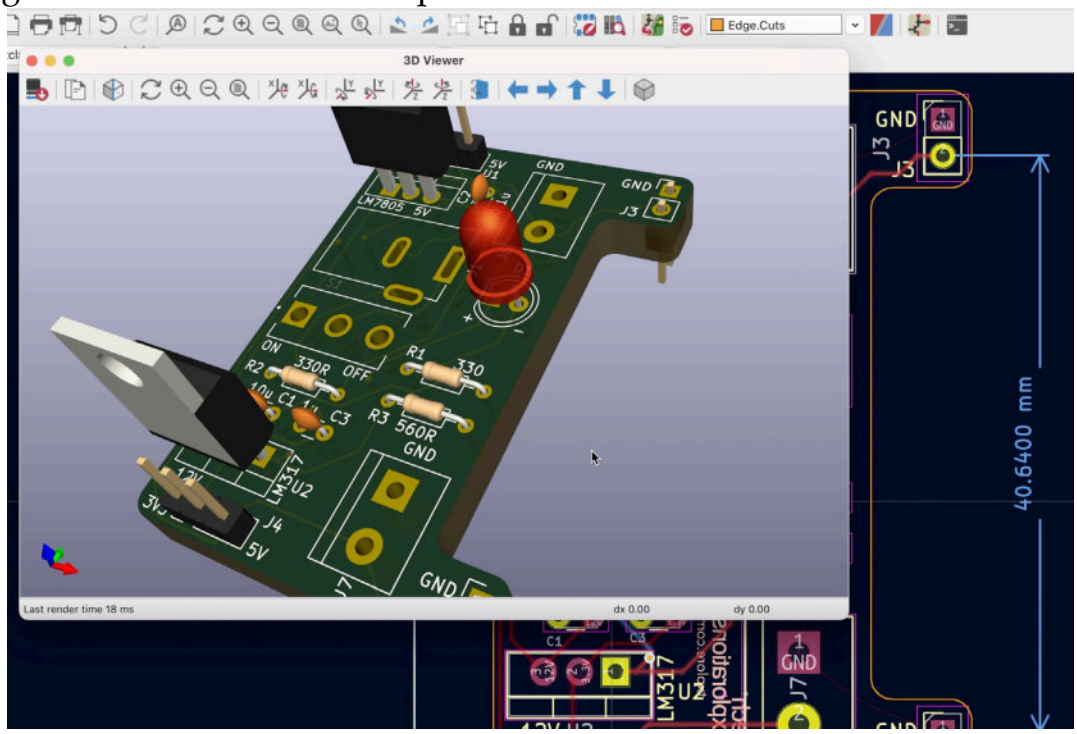


Figure 8.16.1: The 3D viewer showing a rendering of a PCB.

You can use your mouse to change the viewpoint of the 3D rendered board, rotate, pan and zoom. The components on the PCB are 3D shapes that you can find in the 3D shape library and associate with the footprints on the board. In some cases, the associations already exist, but in most cases, you will need to find the correct 3D shape, associate it with the footprint, and edit the properties of the shape to fit it correctly in place. In this chapter, you will learn how to do this.

In the example above, some of the footprints already have an associated 3D shape. Notice the LED, for example. But other footprints do not, such as the barrel connector and the screw terminals.

I will show you how to find and use a 3D shape for the screw terminal.

Double-click on the screw terminal footprint to bring up its properties window (see below).

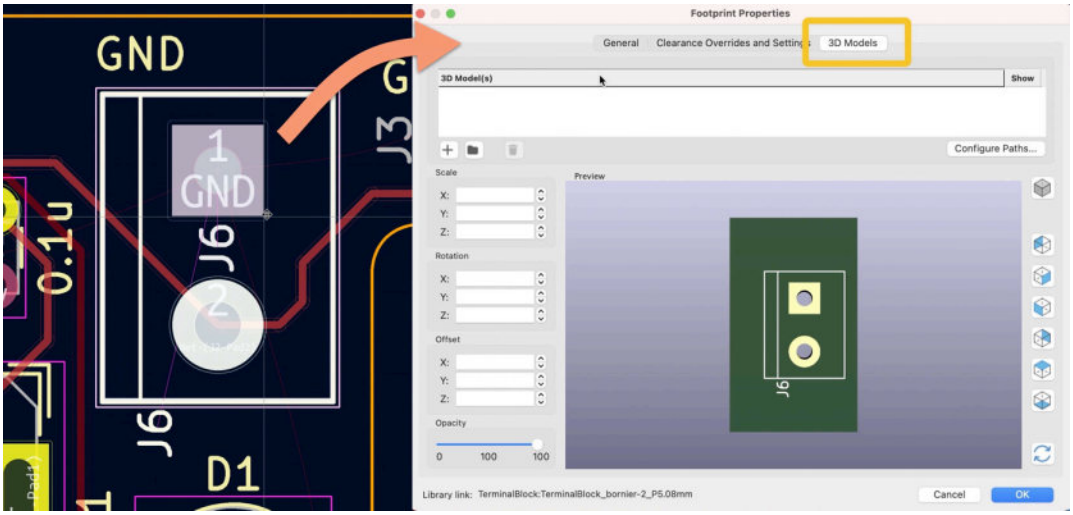


Figure 8.16.2: The properties window of the screw terminal footprint.

Click on the 3D Models tab, and notice that the 3D model's list is empty. To associate this footprint with a 3D shape, you must add a model to the list.

Try this now: click on the library button at the bottom of the 3D model's list. Using the 3D model selector, browse through the library and find a shape that you like. For now, it doesn't matter which one you found; anyone will do. In my example below, I have found a random connector.

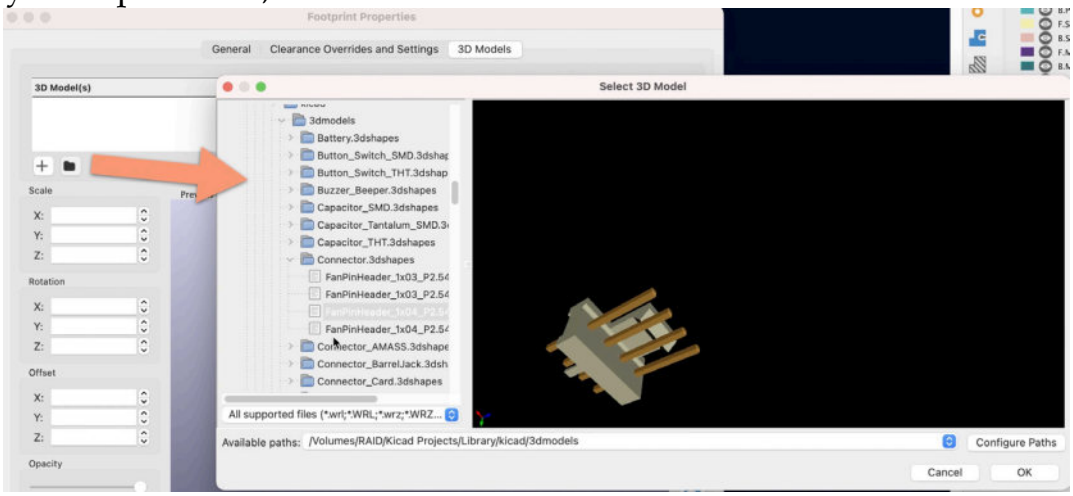


Figure 8.16.3: Finding a 3D shape.

Click OK to close the 3D model selector window. Back in the footprint properties window, 3D Models tab, you will see that the selected 3D shape is now included in the preview rendering of the footprint.

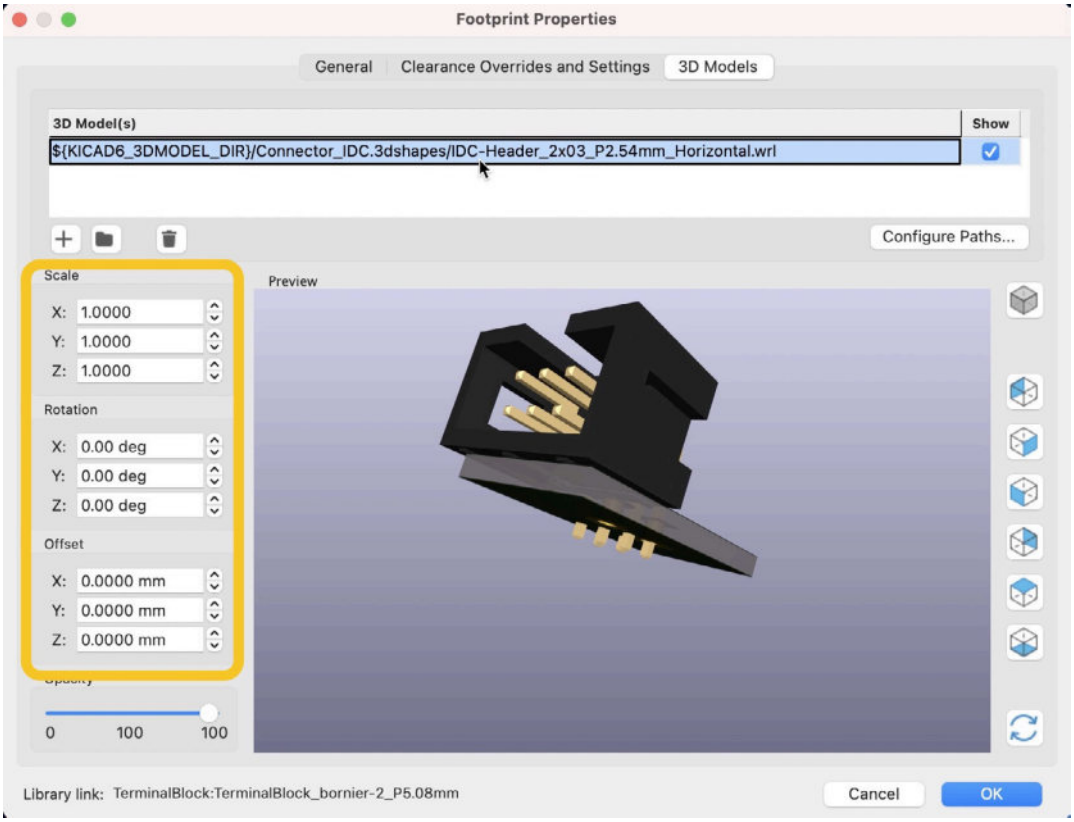


Figure 8.16.4: The selected 3D shape is included in the 3D rendering of the footprint.

The 3D shape may not be oriented appropriately or sized against the footprint. You can use the scale, rotation, and offset widgets to adjust as needed.

3D shapes do not have any electrical characteristics and do not change your layout design in any way. The only use of 3D shapes is to help you visualize how your PCB will look once manufactured and populated with its components. As a result, you can associate incorrect footprints and 3D shapes. Apart from a weird-looking 3D render, there will be no other consequence. The 3D viewer will happily render the incorrect shape:

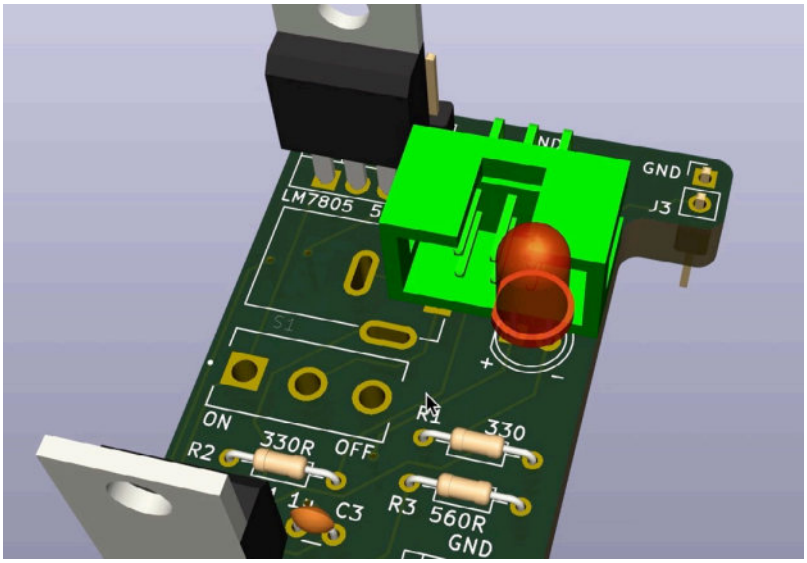


Figure 8.16.5: This is wrong.

I purposefully associated a random and incorrect 3D shape with the screw terminal footprint in the example above. Let's try again, but this time find the right shape. Start by deleting the incorrect 3D model (select the row, and click on the rubbish bin button).

I will assume that I cannot find the needed 3D shape in KiCad's 3D model libraries. So, I will have to look for the 3D shape elsewhere. You can follow the instructions in a previous chapter to learn where to look and how to download symbols, footprints, and 3D shapes.

For this example, I have found the appropriate shape in Snapeda, and download the file on my computer.

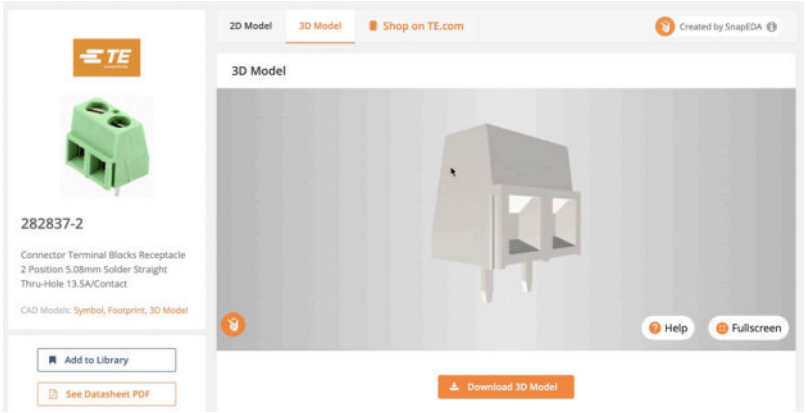


Figure 8.16.6: Found the correct one in Snapeda.

Download the shape on your computer and extract it from the ZIP archive. Below you can see the ".step" file in my project libraries folder.

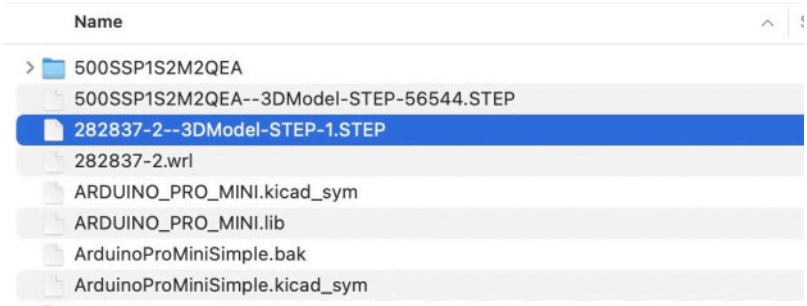


Figure 8.16.7: This is the correct 3D shape file for the footprint.

The 3D viewer in KiCad can use files with the ".step" or ".wrl" extension.

To associate the 3D shape file with the footprint, go to the footprint's properties, and in the 3D Models, tab click on the "+" button. I use the "+" button because I will browse my file system for the shape file. The folder button is more convenient for browsing the installed libraries.

Then click on the "+" button to add a new row to the 3D model's list. Click on the folder button in the right of the row to bring up the file browser, and browse to the location of the ".step" (or ".wrl") file.

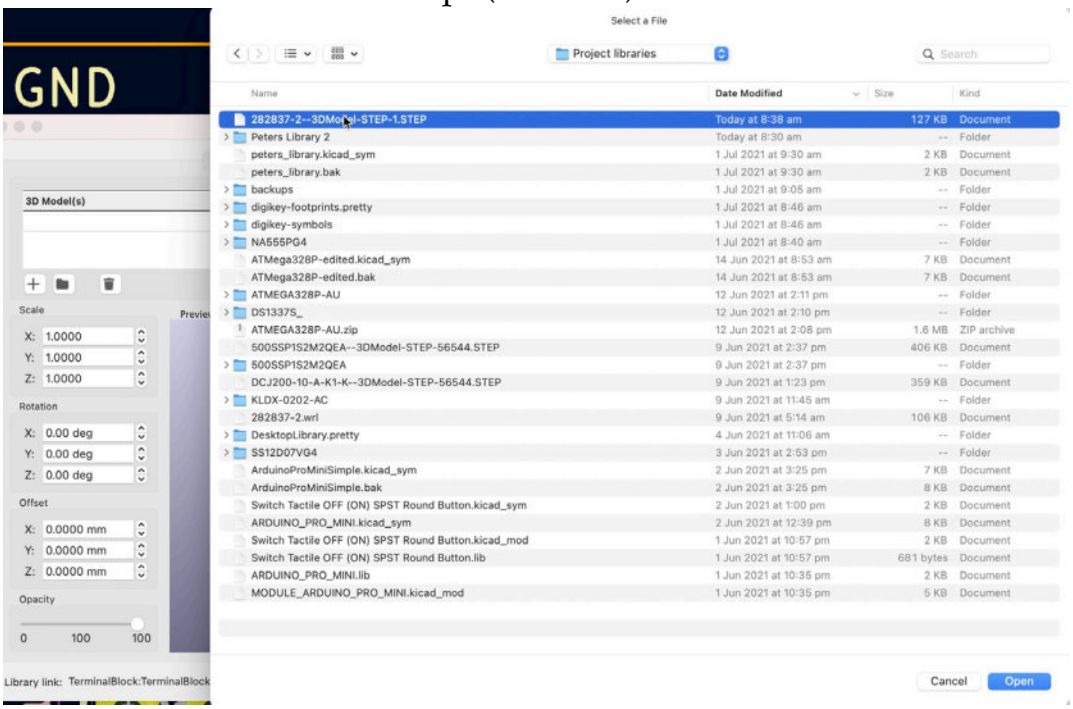


Figure 8.16.8: Find the 3D shape file.

Click "Open." As you can see below, the default position and scale settings render the 3D shape in the wrong position against the footprint.

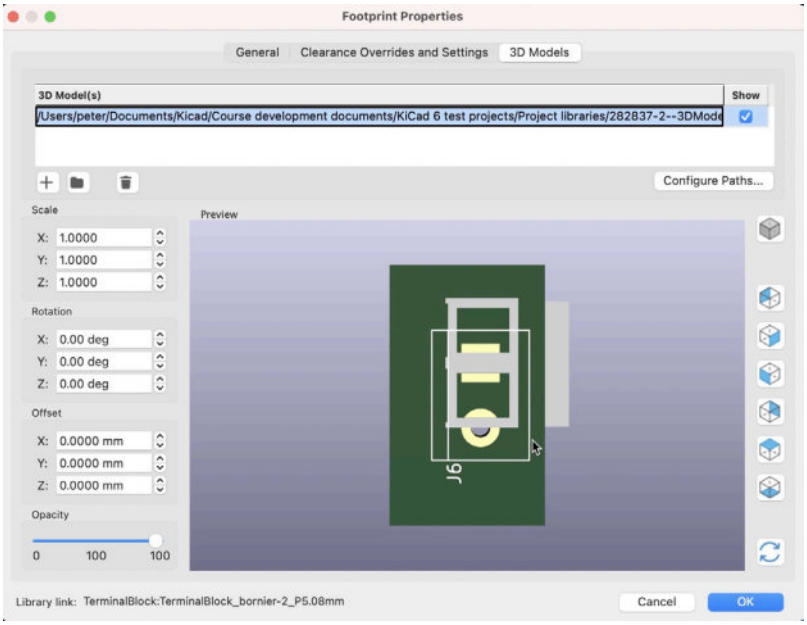


Figure 8.16.9: Associated but misplaced.

I will use the scale, rotation, and offset widgets to position the shape precisely on the footprint. You can see the final position and the positioning settings below:

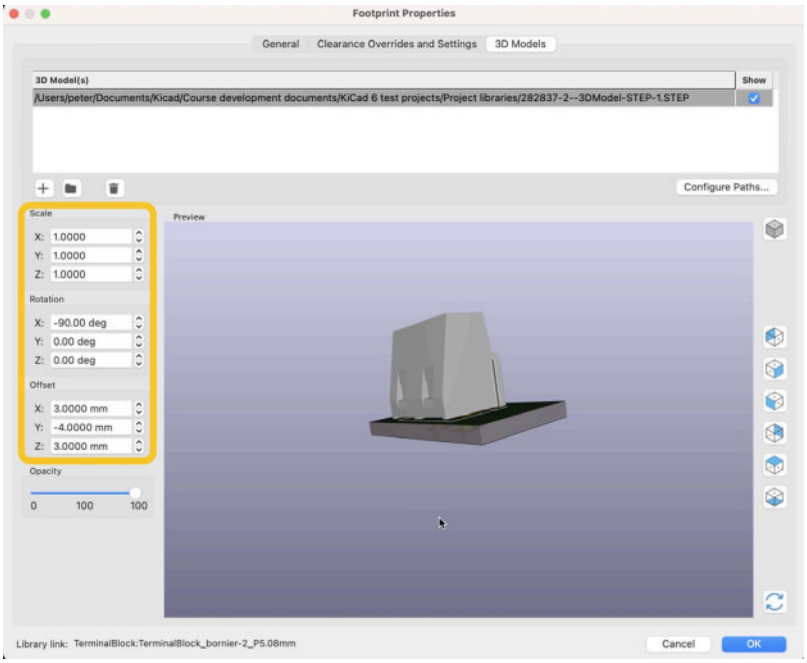


Figure 8.16.10: Final position (maybe?).

This is a good position for the 3D shape. Click OK to close the window and open the 3D viewer to see the new 3D shape on the PCB.

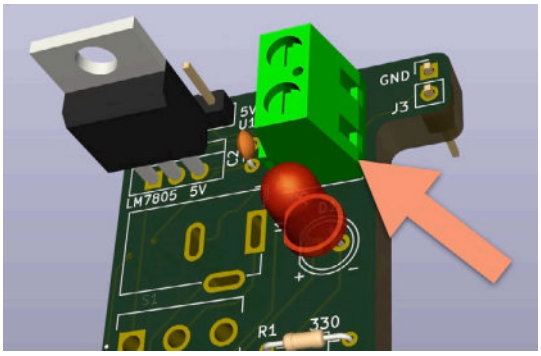


Figure 8.16.11: The terminals are pointing the wrong way.

No, the 3D shape is pointing the wrong way, as its terminals should be posting towards the right. Go back in the footprint's properties window and make a final adjustment of the position:

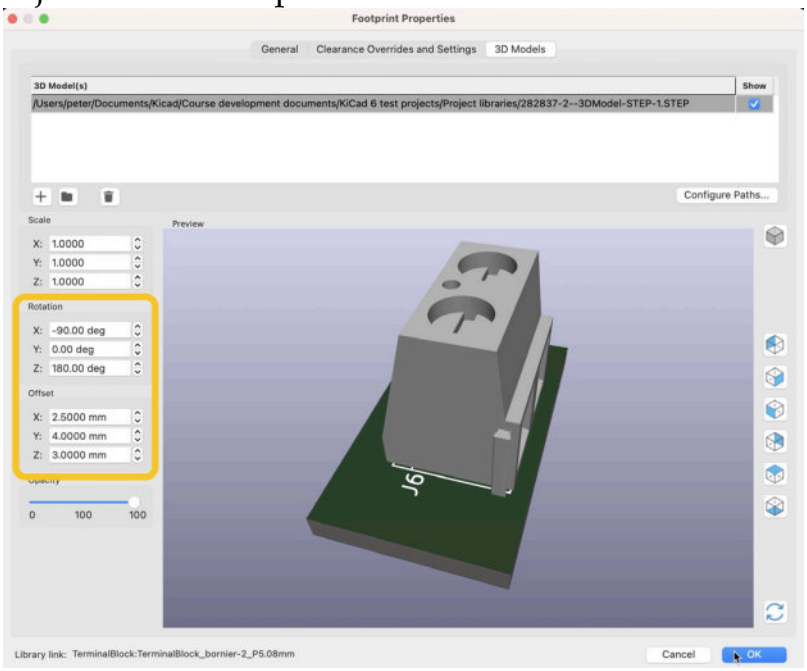


Figure 8.16.12: Corrected rotation and offset.

Let's confirm that this position is correct in the 3D viewer:

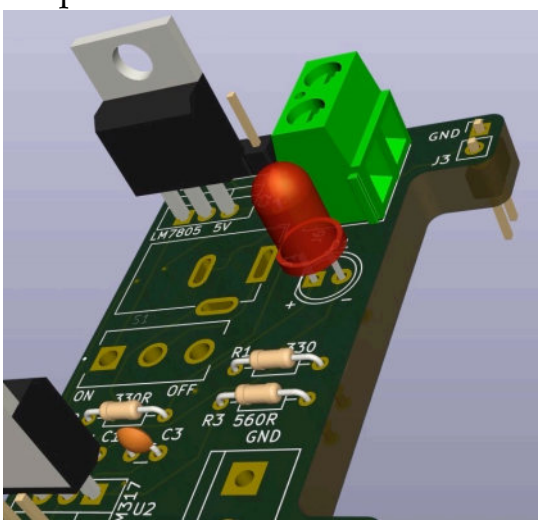


Figure 8.16.13: This position is correct.

The new position is correct. I will finish this work by associating the same 3D shape with the same position settings to the second screw terminal. The final result, in 3D, is below:

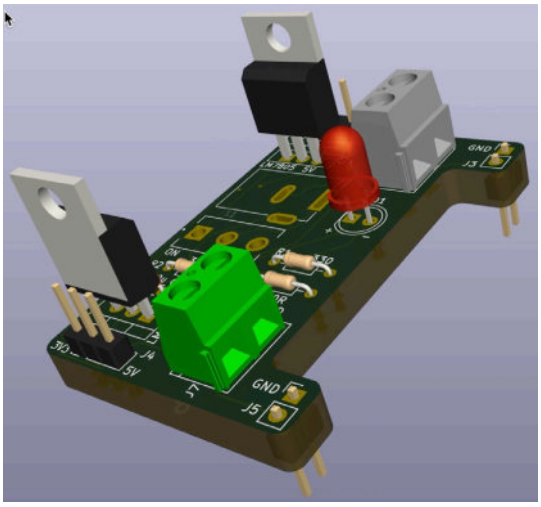


Figure 8.16.14: Applied the same 3D shape to the second terminal.

Taking the time to find and use appropriate 3D shapes to your footprints will help you produce realistic 3D renderings of your board. If you are planning to share your design with other people, a realistic 3D rendering will help communicate the features and characteristics of your board.

17. How to export and test Gerber files

In this chapter, you will learn how to export your finished PCB layout into a set of Gerber files and review them to ensure error-free. Once you have the Gerber files, you will upload them to your preferred manufacturer's website.

To demonstrate the two steps (export and evaluate), I will use the PCB from one of the projects in this book:

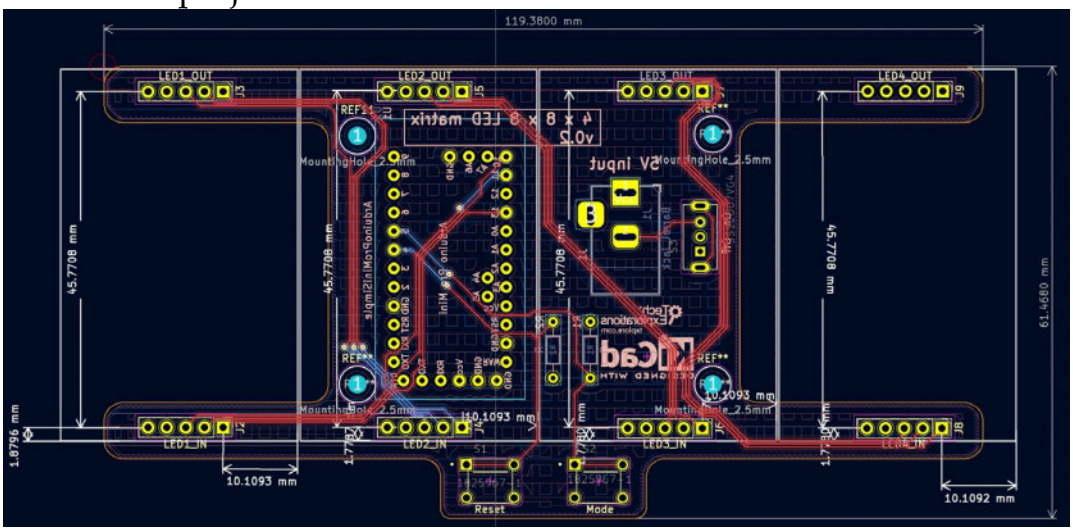


Figure 8.17.1: I will export and test the Gerber files for this PCB.

Before you export the Gerber files for your PCB, always run a final DRC. Ensure that the DRC does not show any errors. Also, check the warnings and ensure that none require action.

Once you are satisfied that your PCB is complete and ready to manufacture, it is time to export the collection of Gerber files. There is a Gerber file for each manufacturable layer, plus one or two files for the drills. To export the Gerber files, you will use the Gerbers export tool. Bring up this tool from the File menu, then "Fabrication Outputs" and finally "Gerbers (.gbr)."

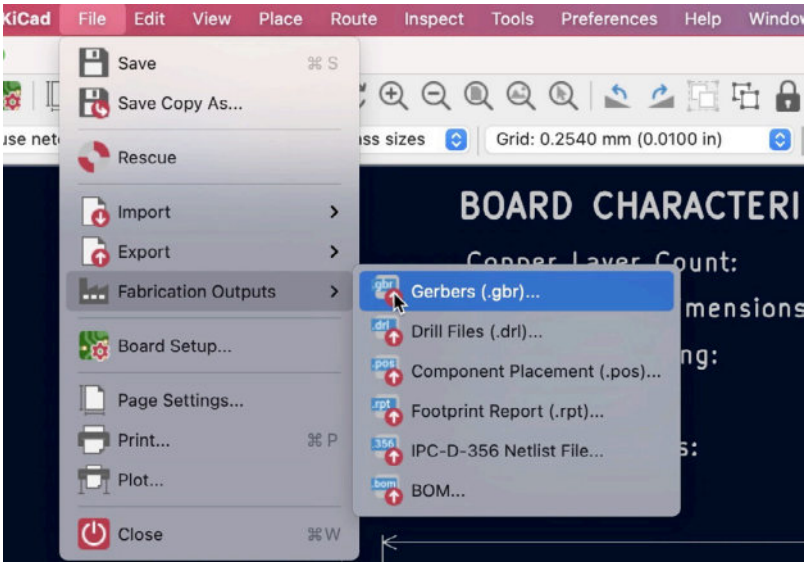


Figure 8.17.2: Invoke the Gerbers export tool.

You can also open this tool via the Plot button in the top toolbar (next to the printer button). You can see the export window below, with my recommended settings:

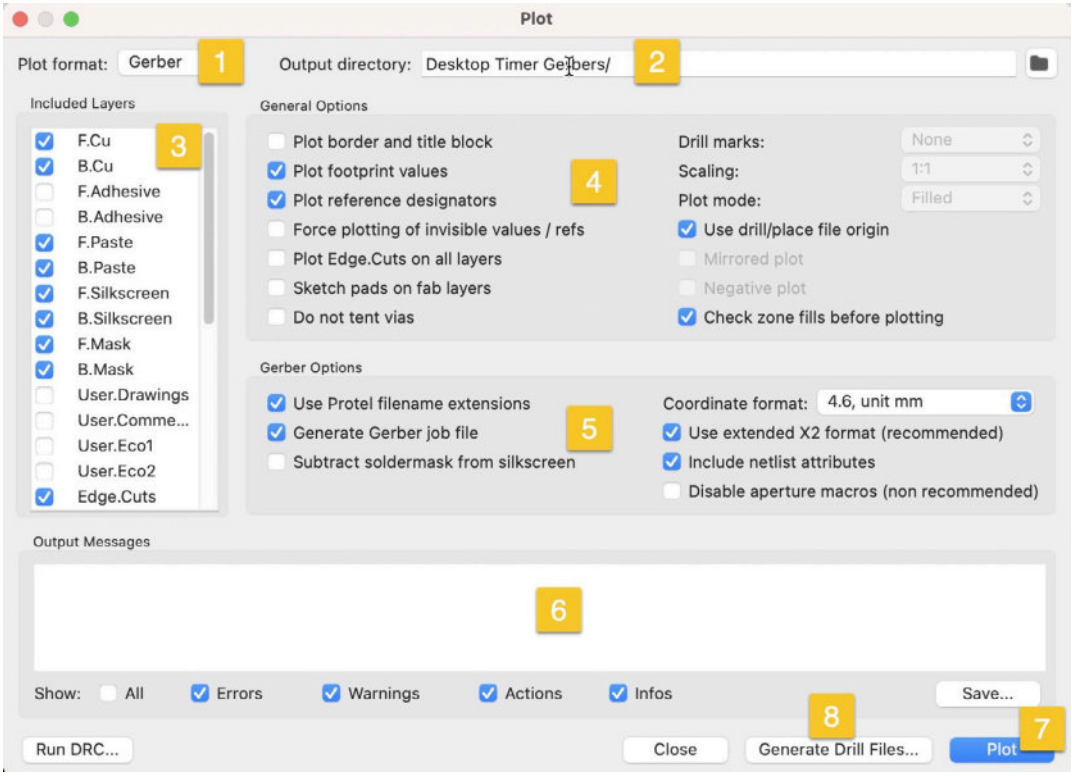


Figure 8.17.3: The Gerber export window and settings.

In the Gerber export window, I have tested the settings with various manufacturers (such as Nextpcb.com, JLCPCB.com, Oshpark.com, and Pcbway.com) and worked without any issues. Below are details and remarks:

1. Plot format: The export tool can export in various formats. Select "Gerber" from the list.
2. Set a directory to hold the exported Gerber files. A good location for this directory is within the project directory.
3. Included layers: Only included the layers that can be manufactured. Your preferred manufacturer may provide a listing of those layers. In my experiments, I include these layers:
   - F.Cu
   - B.Cu
   - F.Paste
   - B.Paste
   - F.Silkscreen
   - B.Silkscreen
   - F.Mask
   - B.Mask
   - Edge.Cuts
4. General options: Enable these options:
   - Plot footprint values.
   - Plot reference designators.
   - Use drill/place file origin.
   - Check zone fills before plotting.
5. Gerber options: Enable these options:
   - Use Protel filename extensions.
   - Generate Gerber job file (you can choose to omit this).
   - Use extended X2 format (recommended).
   - Include Netlist attributes (you can choose to omit this).
6. Output messages: the exporter will provide feedback once you click the Plot button.
7. Plot: click this button to export the files.
8. Generate Drill Files: click this button to open the drill files window (see below).

Click on the Plot button to generate the files (excluding the file for the drills, which we'll do next). You will see new content in the output messages text box confirming the work completed. You can confirm that the Gerber files are saved in the file system:

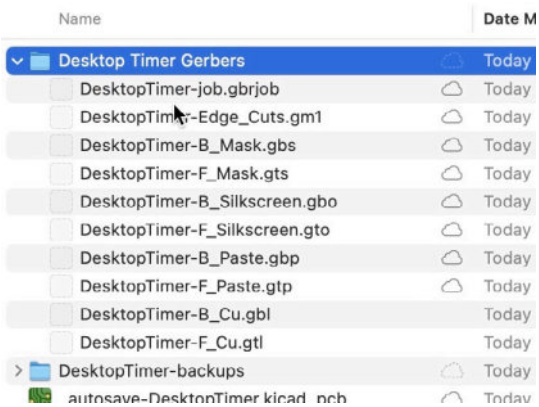


Figure 8.17.4: The new Gerber files.

In the export directory, you will find a Gerber file for each selected layer. The drill files are still missing, so let's generate them next. Still working with the export window (see Figure 8.17.3), click Generate Drill Files ("8").

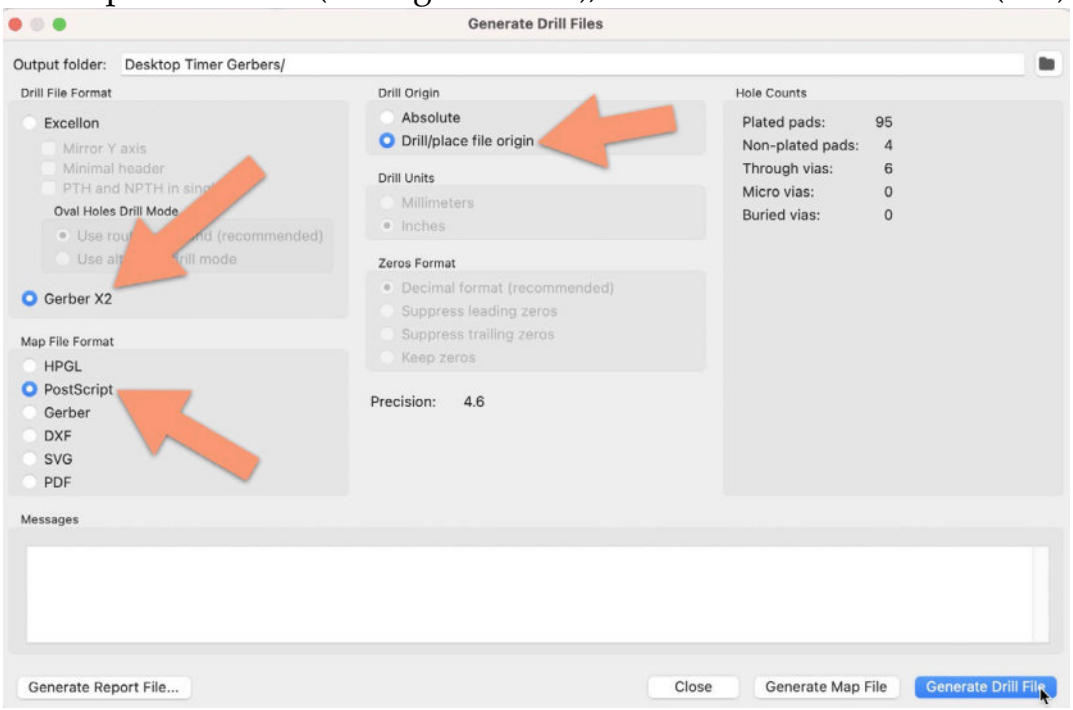


Figure 8.17.5: The Drill Files generator.

In the "Generate Drill Files" window, notice that the output folder is the same as the Gerbers output folder you selected in the previous step. For the various options, copy the settings as in the figure above. Yes: for the Map File Format, select "PostScript." Click "Generate Drill File" and look in the messages for information on the two files that were created (a PTH file and an NPTH file).

You can see the complete set of Gerber files below:

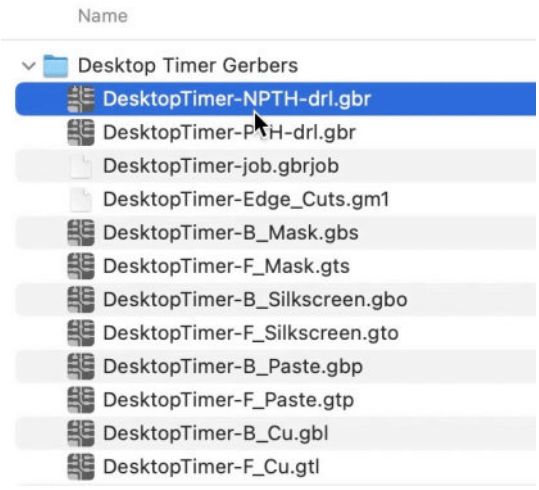


Figure 8.17.6: The complete set of Gerber files for this project.

Click Close twice to close the two windows and return to the layout editor. The Gerber files for the project are ready to test and then upload to the manufacturer's website.

KiCad contains a Gerber viewer app. You can start Gerber Viewer from KiCad's main project window:

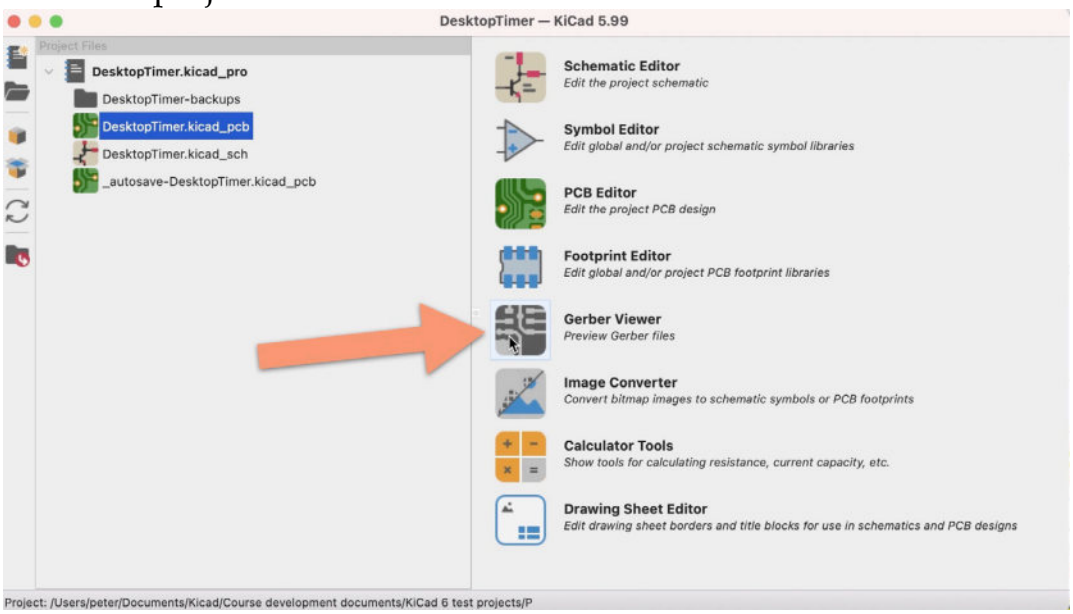


Figure 8.17.7: Starting KiCad's Gerber Viewer.

With Gerber Viewer open, go to File, and click on "Open Gerber Plot File(s)." Navigate to the Gerber files directory, and select all Gerber files in it (don't include the ".dbrjob" file), as in the example below:

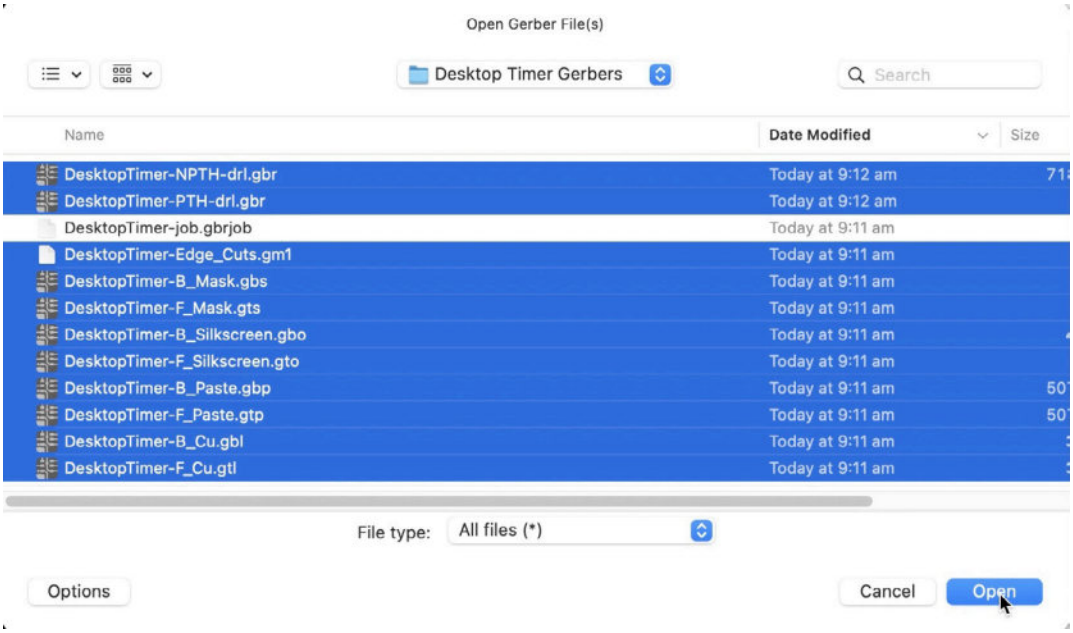


Figure 8.17.8: Select the Gerber files.

The Gerber Viewer will load the selected files and render them. You can enable and disable individual layers by checking them in the layers manager pane (see below):

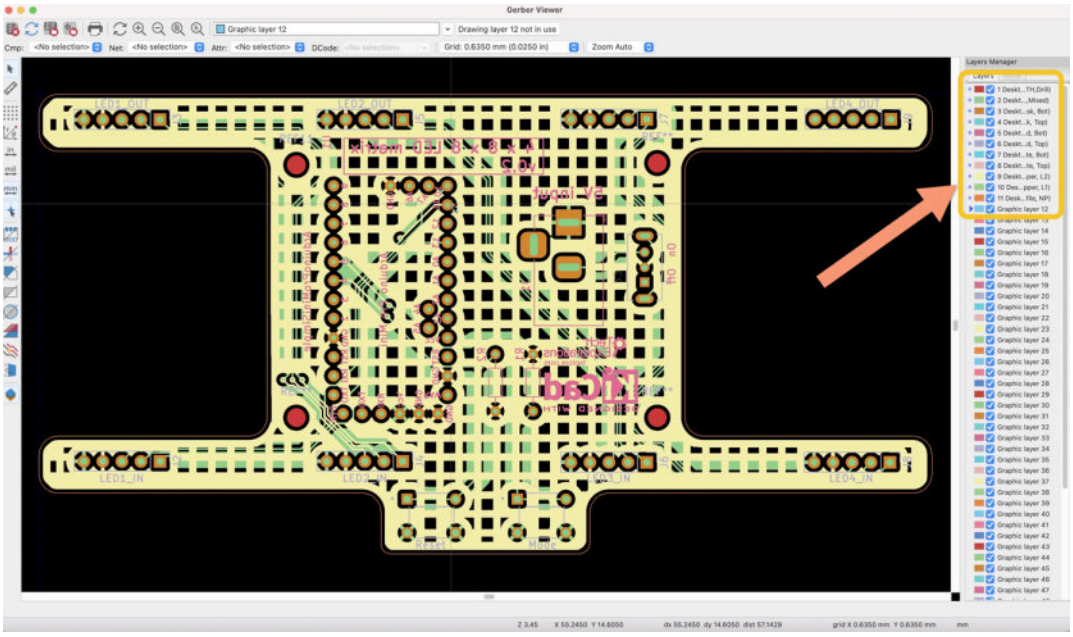


Figure 8.17.9: Gerber Viewer showing my PCB.

At this point, you should take a few minutes to inspect the individual layers. Look for problems with the silkscreen, the copper tracks and fills, vias and holes. Make sure that the edge cut that marks the perimeter of your board is good. Any problem you notice here requires you to return to the layout editor and fix it, then repeat the Gerber file export and inspection process.

I always make a check to ensure that there are no typos in my silkscreen text or graphics missing. In the example below, I have disabled all layers except for the edge cuts and back silkscreen. This removes clutter and makes it easier to find problems.

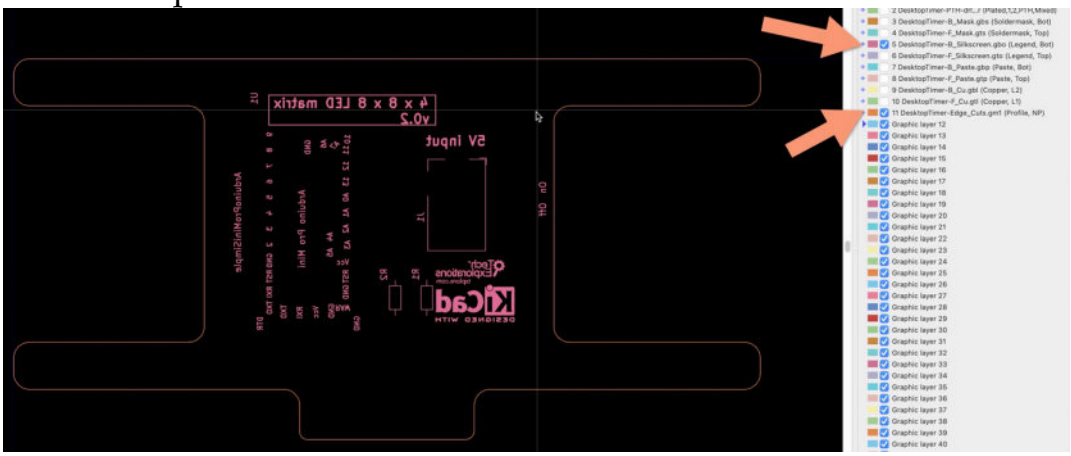


Figure 8.17.10: Gerber Viewer showing the Edge.Cuts and B.Silkscreen layers.

Aside from KiCad's Gerber Viewer, many online manufacturers offer their viewers. These viewers are tuned and made to ensure that customers can confirm that the manufacturer will read the uploaded Gerber files. If your preferred manufacturer offers an online Gerber viewer, you should always use it before ordering (in addition to KiCad's viewer).

There are also third-party viewers that I find helpful. In the example below, I am using the one at www.gerber-viewer.com/viewer to examine my PCB. In most cases, you will need to create a ZIP archive of the folder containing your Gerber files before uploading them to an online Gerber viewer.

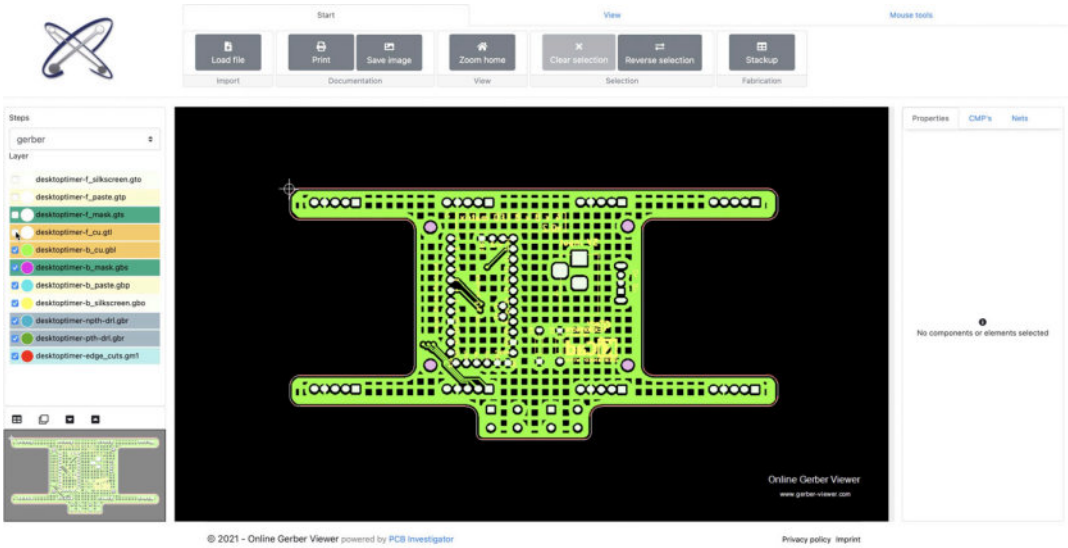


Figure 8.17.11: Online Gerber Viewer showing my PCB.

It is crucial not to rush the Gerber files evaluation process. Take the time you need to scrutinize the files using the KiCad Gerber Viewer app and at least one more online viewer, such as the one at gerber-viewer.com or www.gerblook.org. Most reputable online manufacturers also offer a Gerber viewer; if they do, you should use it to confirm that they will be able to read your files before you upload them.

To learn how to order your PCBs, please read the relevant chapter at the end of the first project in this book. Each online manufacturer may have a slightly different ordering process, but this example will help you with your very first order.

Part 9: Project - Design a simple breadboard power supply PCB

1. Introduction

Welcome to Part 9 of this book! In the following chapters, you will learn how to design a simple yet practical PCB. This PCB is a component of a breadboard power supply. You can use this power supply to provide power to circuits implemented on a mini breadboard, which is a core part of electronics prototyping.

This project is an opportunity to use the knowledge you acquired in the last part of this book to create a non-trivial PCB. To design this PCB, you will be using the majority of the capabilities of KiCad's schematic and layout editors. You will also practice the PCB development workflow that you learned in Part 6 of the book.

The inspiration for the design of this PCB came from my work at creating small electronics circuits for my Arduino and ESP32 courses. When the circuit I was building on the breadboard needed more power than the MCU could provide, I would search through a range of possible options that usually included one of my bench-top power supplies and wires. The problem is that the bench-top power supplies are noisy (they have a large cooling fan), need some setup (select voltage, current), and their wires get in the way. In addition, I have drawers full of wall power supplies that I could be using. They are plug-and-play and silent.

For my breadboard power supply, I needed something that:

1. Plugs directly on the breadboard; therefore, there are no wires.
2. Have an on/off switch.
3. Can provide 5V and 3.3V power.
4. Can draw power from a range of wall power supplies, from 6V to 12V.

After some deliberation, I settled for a design like the one in the image below:

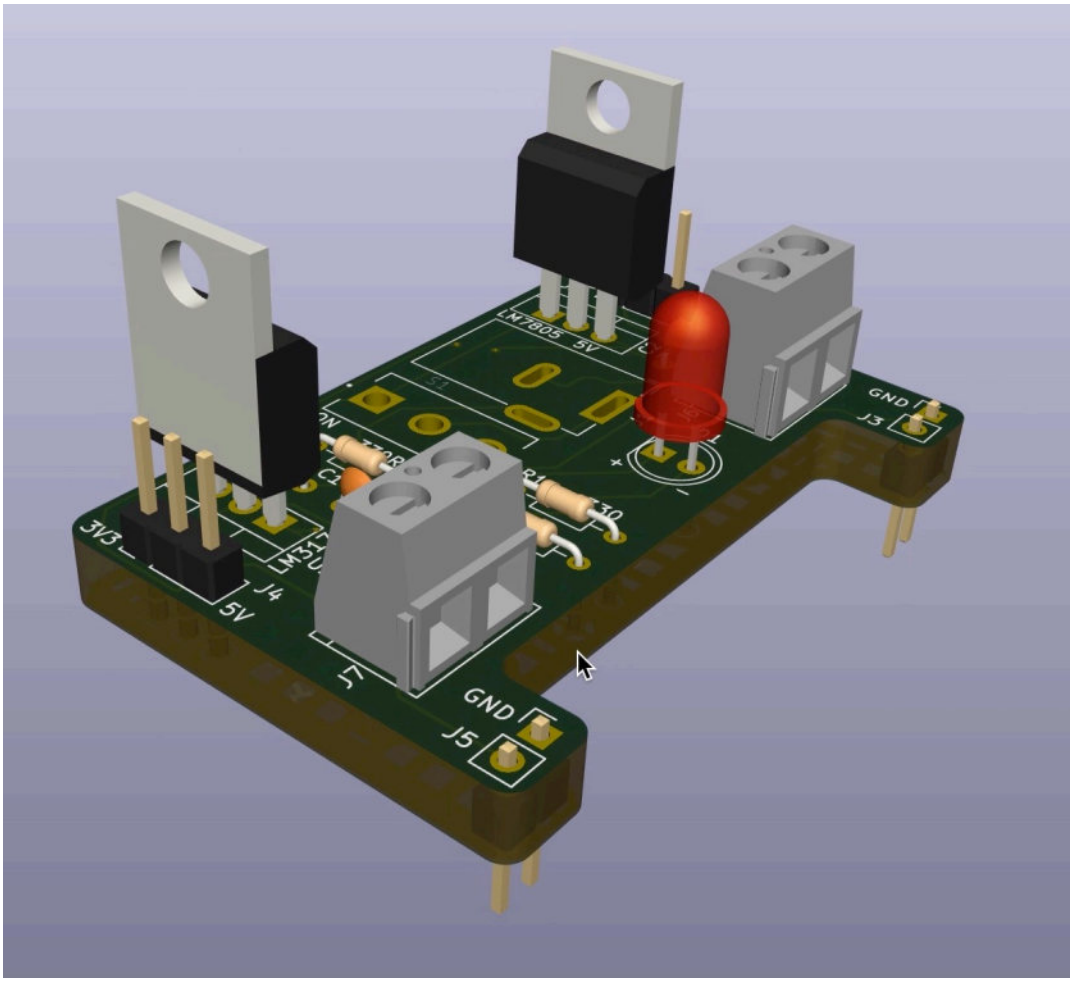


Figure 9.1.1: A 3D view of the breadboard power supply PCB.

The PCB's dimensions and shape are constrained by the dimensions of power row locations of the mini breadboard on which the PCB will connect. The connection between the breadboard and the PCB is made via two sets of pin headers. I have added two double screw terminals to provide an additional way to output power via jumper wires instead of the pins.

Below you can see a photo that shows the PCB against a mini breadboard:

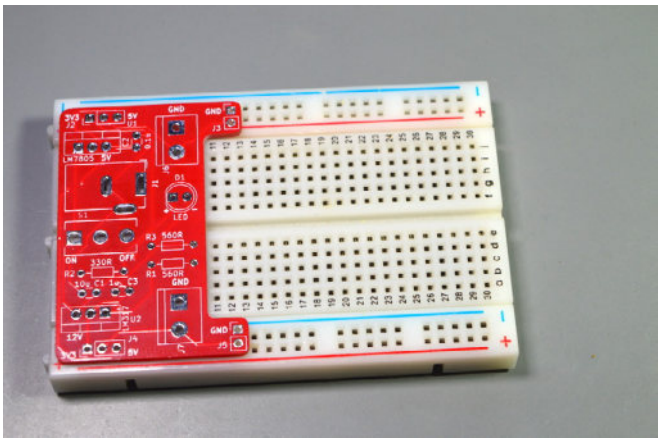


Figure 9.1.2: The power supply PCB over a mini breadboard.

When the PCB is attached to the left side of the breadboard, almost the entire right side is available for the prototyping circuit. The indentation between the pin headers also allows access to the first couple of columns in the breadboard that otherwise would have been covered.

To keep the power supply cable away from the prototyping area, I have placed the barrel connector on the left side of the PCB. The voltage selector switches are on the top and bottom of the board to make it easy to access.

Below you can see the final schematic design:

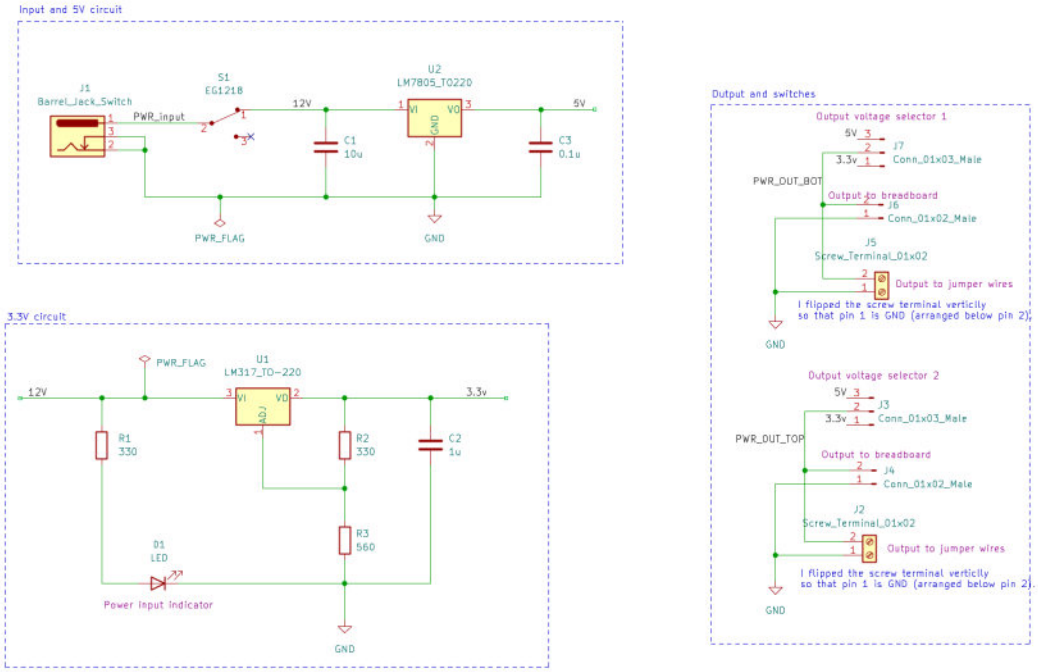


Figure 9.1.3: The project schematic design (final).

In the schematic above, you can see the power supply components arranged in three functional groups. You can see the two major components, the voltage regulators, inputs, outputs, and switches. You will work on the schematic design in the next chapter.

We'll do the schematic design in a single sheet. Most of the symbols needed come with KiCad's libraries, but one is available in the Digikey library.

Below, you can see the final layout design:

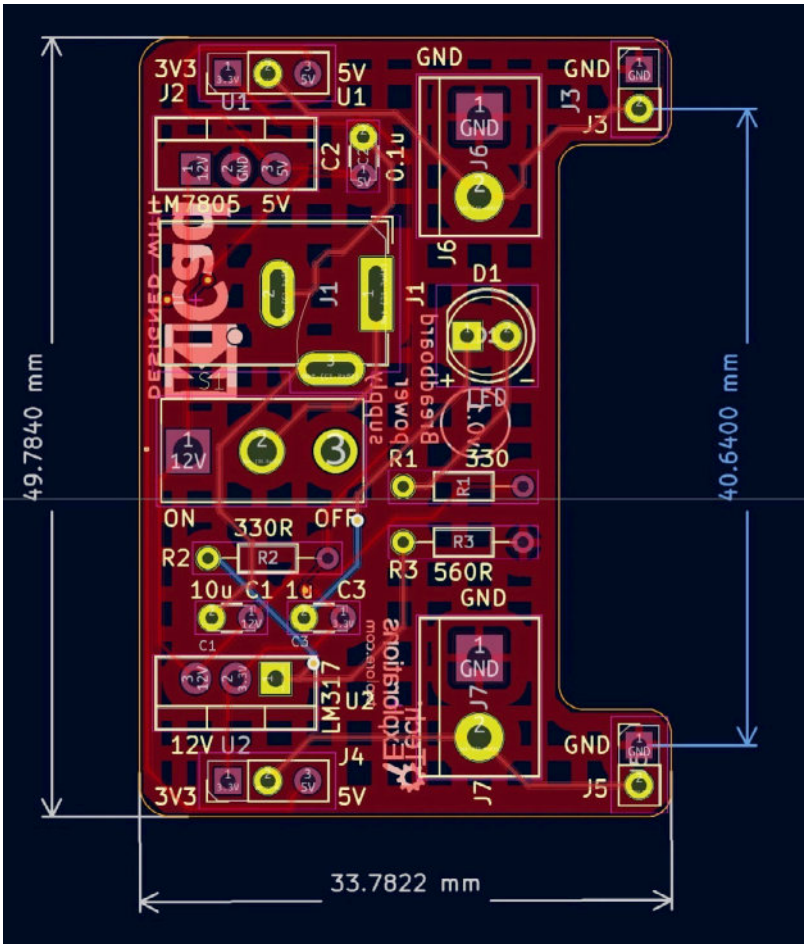


Figure 9.1.4: The project layout design (final).

The layout has several interesting features, including a composite shape with rounded corners, copper fills, all THT components to make it easy to assemble, a complete set of top and bottom silkscreen text and graphics, and is manually routed.

Perhaps the most challenging aspect of the layout design is its dimensions. The PCB's pin headers have to match precisely with the mini breadboard's power row pins. To achieve a good match, you will need to make accurate measurements on the breadboard and then use those measurements to precisely position the two double pin headers. Then you will design the board around those fixed footprints.

Reference	Value	Footprint
C1	10u	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C2	1u	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
C3	0.1u	Capacitor_THT:C_Disc_D3.0mm_W1.6mm_P2.50mm
D1	LED	LED_THT:LED_D5.0mm
J1	Barrel_Jack_Switch	Connector_BarrelJack:BarrelJack_Horizontal
J2, J5	Screw_Terminal_01x02	TerminalBlock:TerminalBlock_bornier-2_P5.08mm
J4, J6	Conn_01x02_Male	Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
J3, J7	Conn_01x03_Male	Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
R2	330	Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal
R1, R3	560	Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal
S1	EG1218	digikey-footprints:Switch_Slide_11.6x4mm_EG1218
U1	LM317_TO-220	Package_TO_SOT_THT:TO-220-3_Vertical
U2	LM7805_TO220	Package_TO_SOT_THT:TO-220-3_Vertical

Table 9.1.1: The Bill of Materials for this project.

IMPORTANT NOTICE

The first iteration of this PCB project contained a design defect. I learned about this error after Wayne, a reader of the beta version of this book

informed me by submitting errata reports. I decided that instead of rewriting the chapters in this part of the book, I should take the opportunity to document the process of correcting this defect. Therefore, I have maintained the defect in chapters two and three of this project.

I have added a new chapter, titled "4. Finding and correcting a design defect" where I detail the defect, and show how to correct it. The fix is fairly comprehensive as it requires significant changes to the schematic and to the layout.

2. Schematic design editing

In this chapter, you will complete the schematic design of this PCB by following the schematic design workflow (see below). You learned about this workflow in Part 6 of the book.

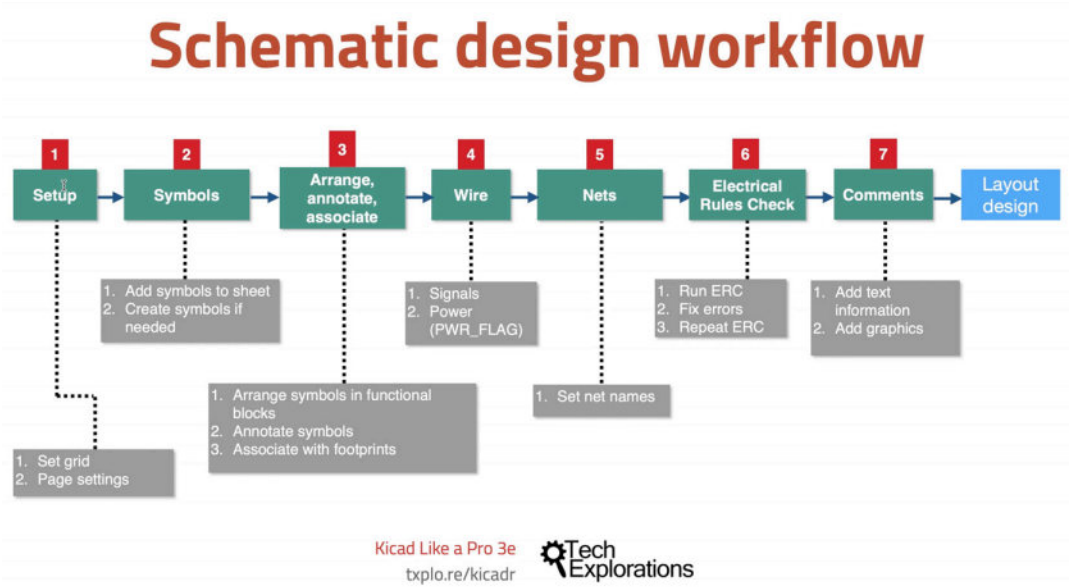


Figure 9.2.1: The schematic design workflow.

Unlike the simple project you completed in Part 3 of the book, in this project, you will use the model workflow more realistically. Instead of using it linearly, there will be cases where you will need to return to a previous step, fix or improve something, and then continue.

Time to start.

2.1. 1 - Setup

Open KiCad and create a new project. Give the project a name. I have called mine “Breadboard Power Supply” and saved it in my projects folder.

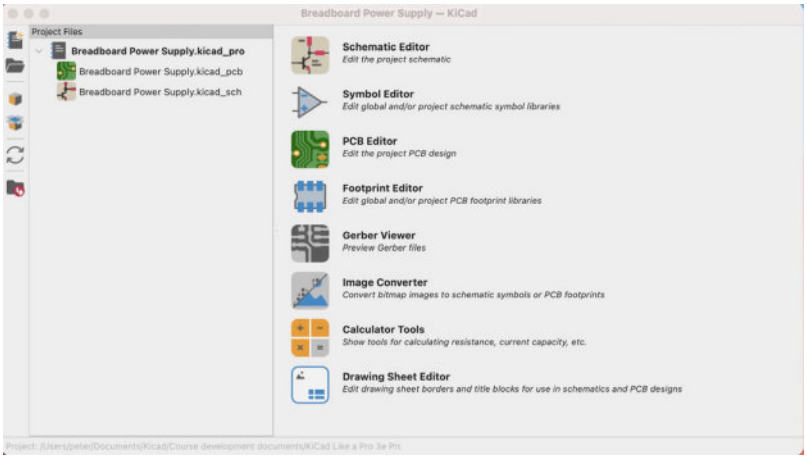


Figure 9.2.1.2: The new KiCad project.

In the project folder, you will see three files:

- The project file: “Breadboard Power Supply.kicad_pro”.
- The layout file: “Breadboard Power Supply.kicad_pcb”.
- The schematic file: “Breadboard Power Supply.kicad_sch”.

In the project window, click on the Schematic Editor to start Eeschema. The schematic sheet is empty. Let’s do a basic setup for the project.

Bring up the Schematic Setup window (File —> Schematic Setup, or click the Setup button from the top toolbar). Review the schematic settings. I will be leaving these settings as per their defaults. I will be adding Net Classes later in the project.

Similarly, in the KiCad Preferences window, I will be using the default settings. In the Display Options tab (under Preferences —> Schematic Editor), I have set “Snap to Grid” to “Always“and “Cursor Shape” to “Full window crosshair.”

I have checked “Constrain buses and wires to H and V.”

In the Colors tab, I have changed the background color of my theme to white so that the screenshots on these pages look better (the default background color is light gray).

The last setup item in my list is to enter the project details in the Page Settings window. Bring up this window (File —> Page Settings or click on the Settings button from the top toolbar). Fill in the text fields with the information you’d like to show in the sheet’s information corner. Also, select a sheet size and enter the issue date. You can see my Page Settings window below:

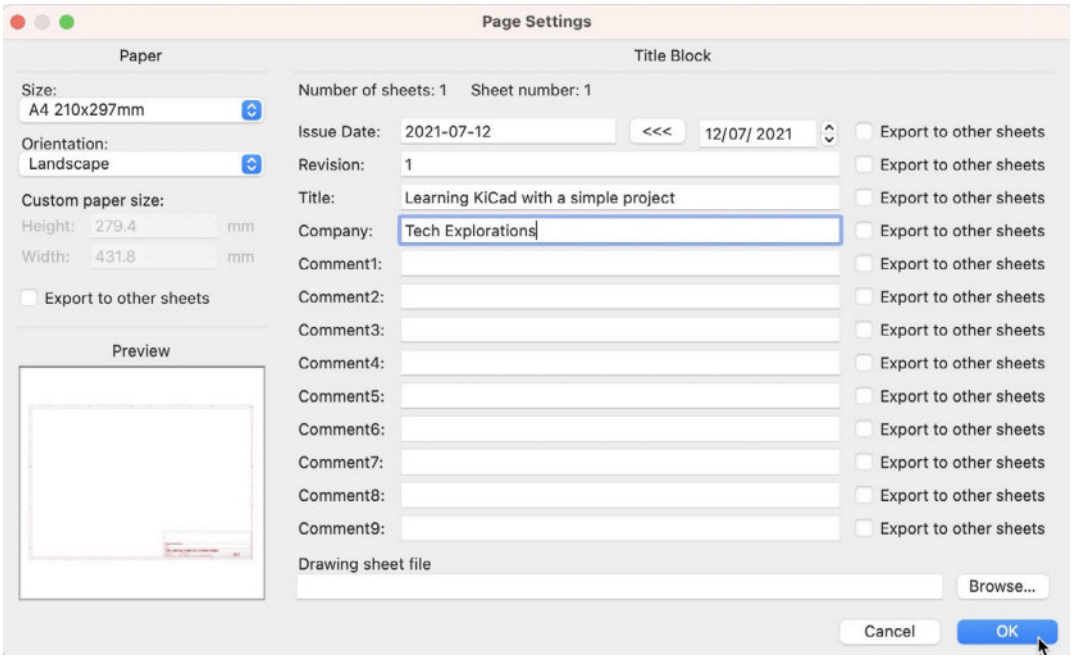


Figure 9.2.1.3: The schematic editor page settings.

This information will appear in the bottom-right corner of the schematic sheet.

Setup is complete; let's continue with step two of the process, adding the component symbols on the editor sheet.

2.2. 2 - Symbols

In this segment, you will find the component symbols and add them to the sheet. Most symbols are available in the KiCad libraries, but one is in Digikey's library. If you have done so yet, you can learn how to find and install a third-party symbol library in an earlier chapter in this book.

You can see a list of the components you will need to add in the project introduction chapter. You learned how to add a symbol to the sheet in an earlier chapter. Repeat the process for each component in the bill of materials.

Let's do the first one together.

With Eeschema opened, type "A" to open the symbol library chooser (or click the "Add a Symbol" button from the right toolbar). Allow a few seconds for the libraries to load into the cache. Once the cache is created, the symbol chooser window will open much faster.

As you already know the names of the symbols you want to use (i.e., they are in the first column of the bill of materials), use the Filter text box to find a symbol quickly. This project contains three capacitors. Enter "C" in the filter, and you will see the required symbol appear under "Device." The

symbol preview will appear on the right side of the window. You can see my symbol chooser window displaying the capacitor symbol below:

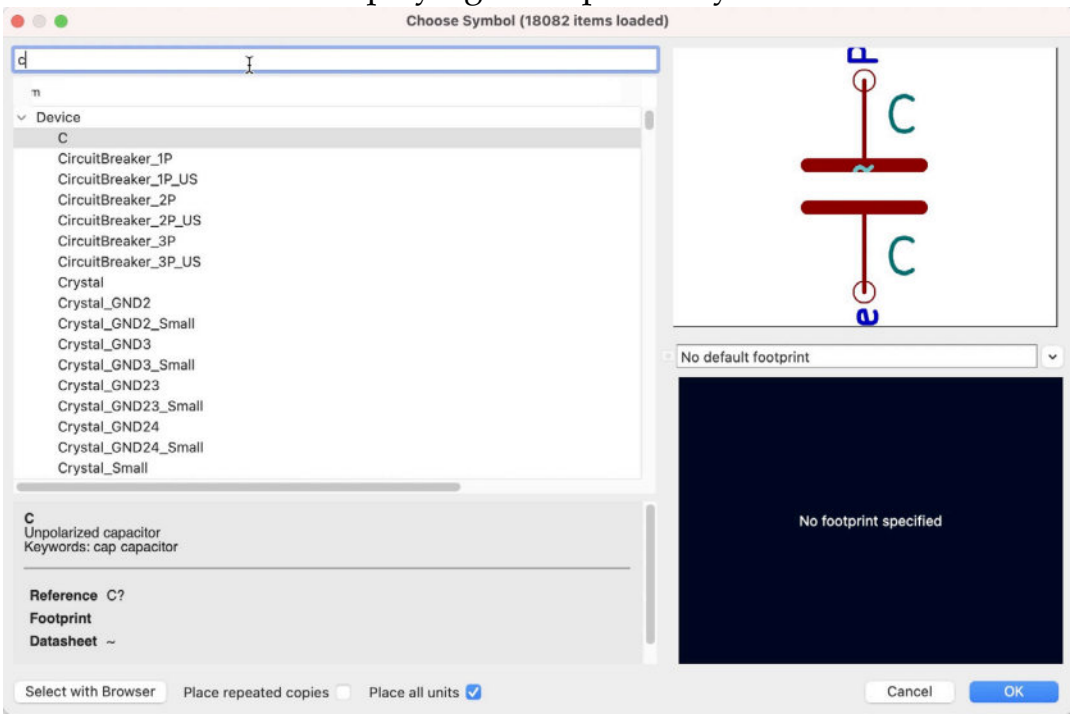


Figure 9.2.2.4: The symbol chooser with the capacitor symbol selected.

Double-click on the "C" row to insert this symbol into the sheet. The Chooser window will disappear. Click anywhere in the sheet to add the new symbol. I have placed mine close to the center of the sheet.

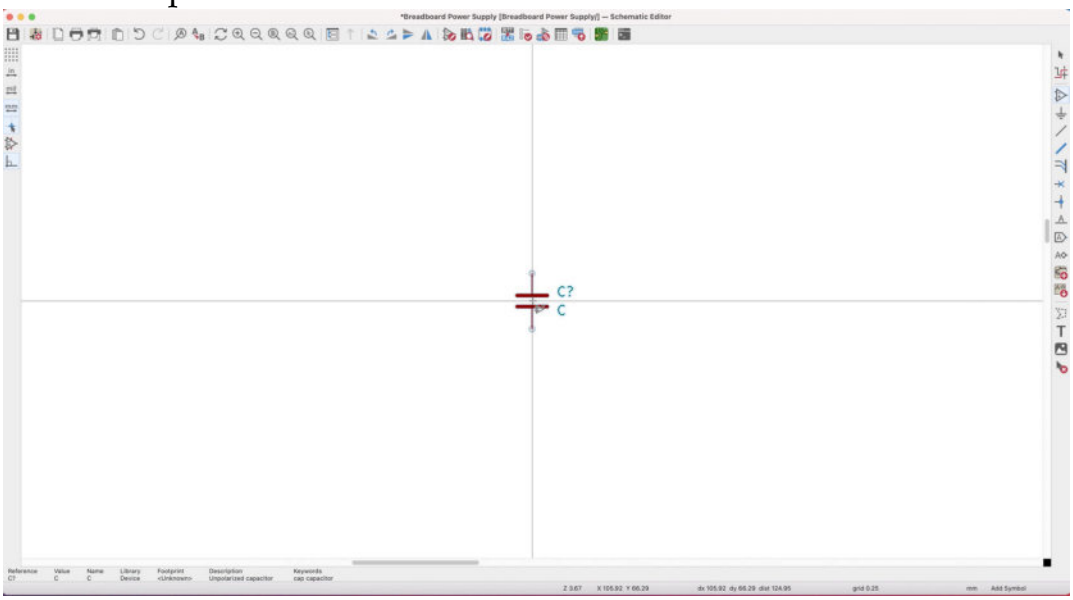


Figure 9.2.2.5: A new capacitor symbol in the sheet.

The power supply requires three capacitors. Instead of going back to the Symbol Chooser for the other two, you can create two more copies of the first one. Use the Ctr-D/Cmd-D shortcut to duplicate a selected item and create the two additional copies. You should now have three capacitors:

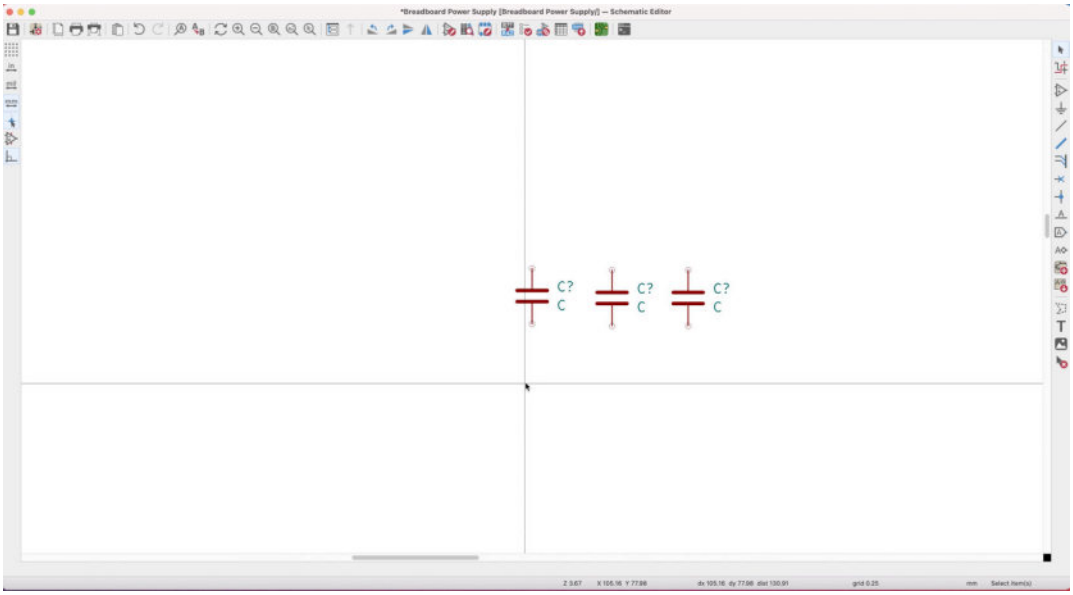


Figure 9.2.2.6: Three capacitor symbols in the sheet.

Continue with the rest of the symbols as you see them in the list in the introduction chapter. For the barrel connector, I have opted for a symbol from the Digikey library (see "S1" in Table 9.2.1). If you can't find this symbol in the Symbol Chooser, ensure that you have installed the Digikey symbol library to your KiCad instance.

Once you have added all symbols in the editor, your schematic sheet should look like this:

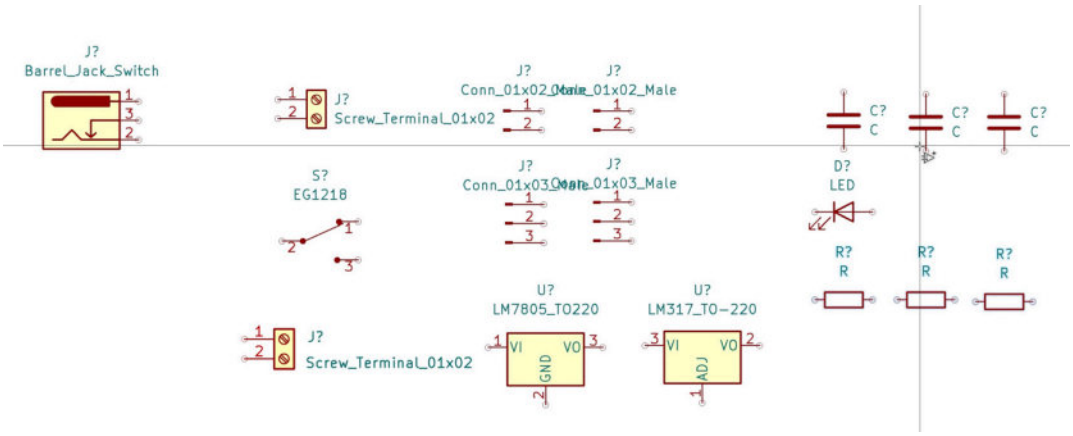


Figure 9.2.2.7: All project symbols placed in the sheet.

Before you continue with step three of the workflow, where you will arrange, annotate and associate the symbols, you will add component values to the capacitors and resistors.

2.3. 2 - Edit Component values

You are still working on step two of the workflow. The task now is to edit the Value fields for the capacitor and resistor symbols.

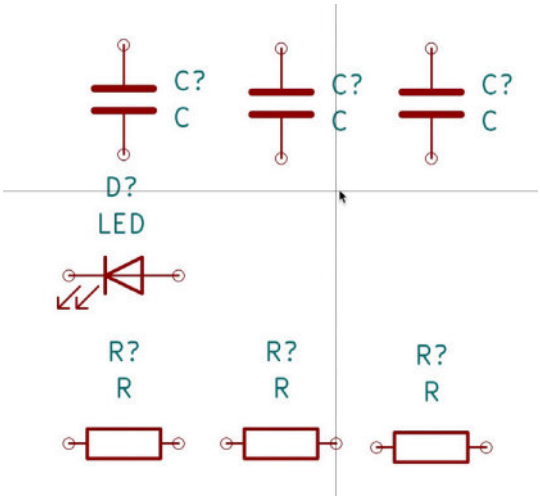


Figure 9.2.3.8: Add the resistor and capacitor values.

If you have symbols that share the same value, you can speed up this step by editing the Value field in the first symbol and duplicating it. The duplicated symbols will have the Value of the original.

You can see the Values for each symbol in the second column of Table 9.2.1.

To edit the Value field, double click on a symbol to bring up its Properties window. Click in the Value field and type in the respective value. Below you can see the Value field for the first capacitor, C2 (at the moment, its reference is “C?” because you have not done the annotation yet):

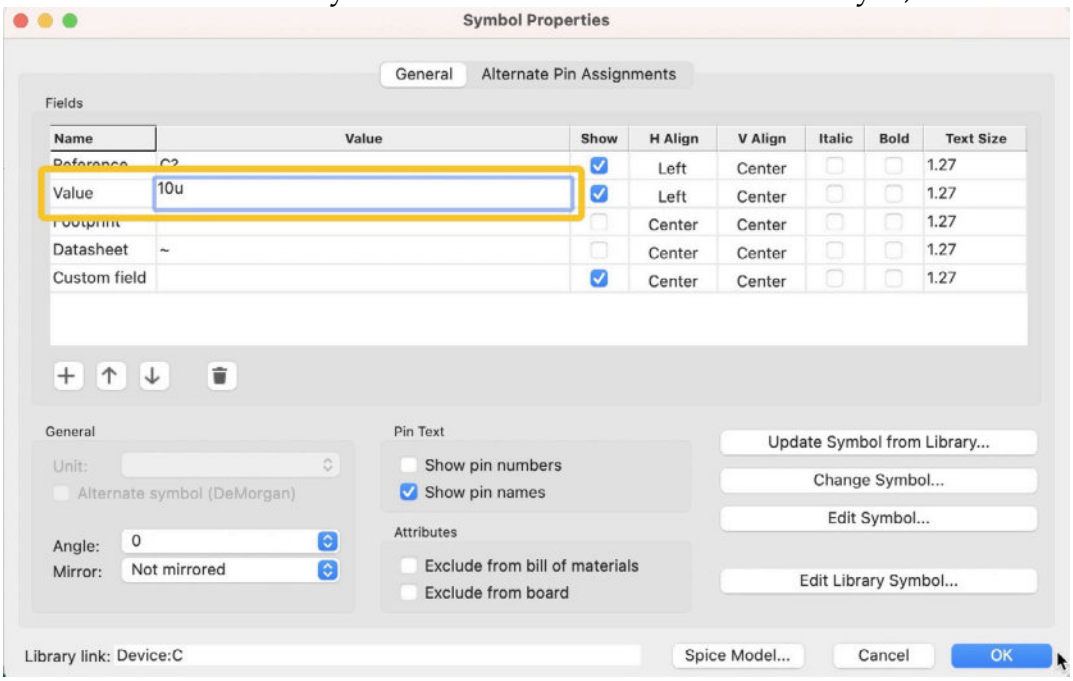


Figure 9.2.3.9: Edit the Value field.

Click OK to close the Properties window. The value will appear next to the capacitor:

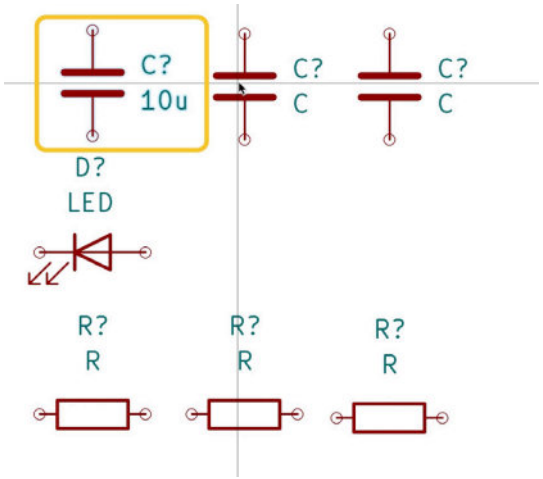


Figure 9.2.3.10: The value of the first capacitor is 10u.

Repeat the process for each capacitor and resistor, using the values from Table 9.2.1. Once complete, the schematic will look like this:

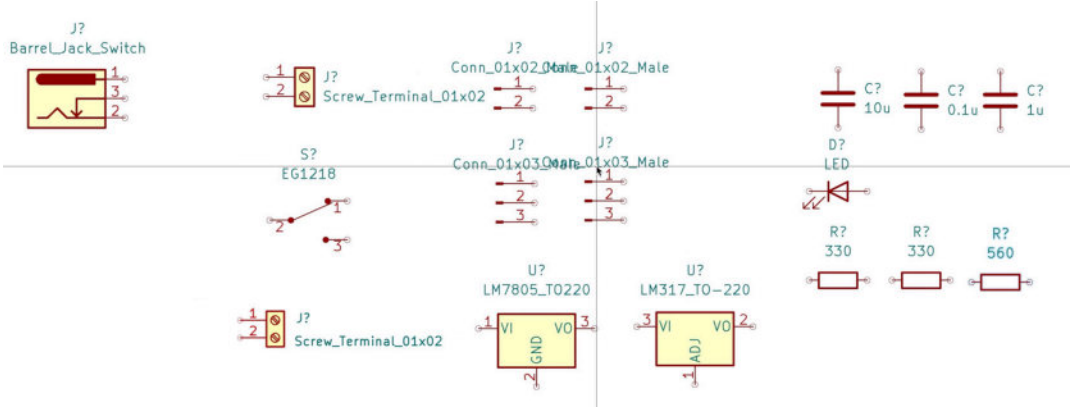


Figure 9.2.3.11: Project symbols with values added.

Step two is complete. Let's continue with Step three, where you will arrange the symbols according to which functional group they belong to, annotate them with unique identifiers, and associate them with footprints.

2.4. 3 - Arrange, Annotate

Step three of the schematic design workflow is where we:

1. Rearrange the symbols to the (usually) final locations in the sheet.
2. Annotate the symbols with their unique identifiers.
3. Associate the symbols with the footprints we'd like to use in the layout.

I have broken down step three into two parts to make the discussion that follows more manageable.

In this segment, you will do the arrangement and annotation of the symbols. In the following segments, you will finish step three with the association.

Arrange

In Figure 9.2.4.11 in the previous segment of this chapter, you can see the current arrangement of the symbols. This arrangement is not random. I typically place together symbols of the same kind. For example, I group capacitors with other capacitors and pin headers with other pin headers.

In step three of the schematic design workflow, you will change how the symbols are arranged according to the functional group to which they belong. For example, the 3.3V voltage regulator, with three resistors, a capacitor, and an LED, is the 3.3V functional group.

There is not a single correct way to group symbols in functional groups. A group may have more than one function, or a single symbol may be providing a direct service that is useful to multiple groups. The objective here is to look at the connection between the symbols and place them to make the schematic readable and electrically correct.

I typically start the process of arranging the symbols from the inputs. In this project, I will place the input (the power barrel jack) at the top left of the sheet. The next logical symbol is the switch that I have placed to the immediate right of the barrel jack. Below you can see the state of my schematic at this time:

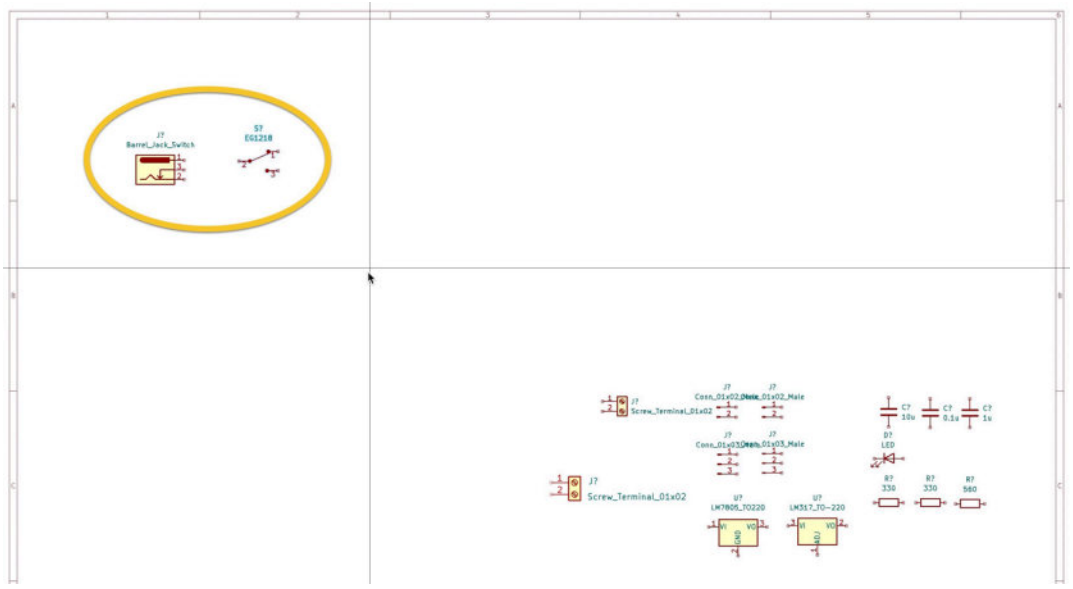


Figure 9.2.4.12: Arranged barrel jack and switch (input).

I continued with other symbols that belong to the power input group, like the LM7805 regulator, the regulator's capacitor network, etc.

Below you can see the final arrangement of the symbols:

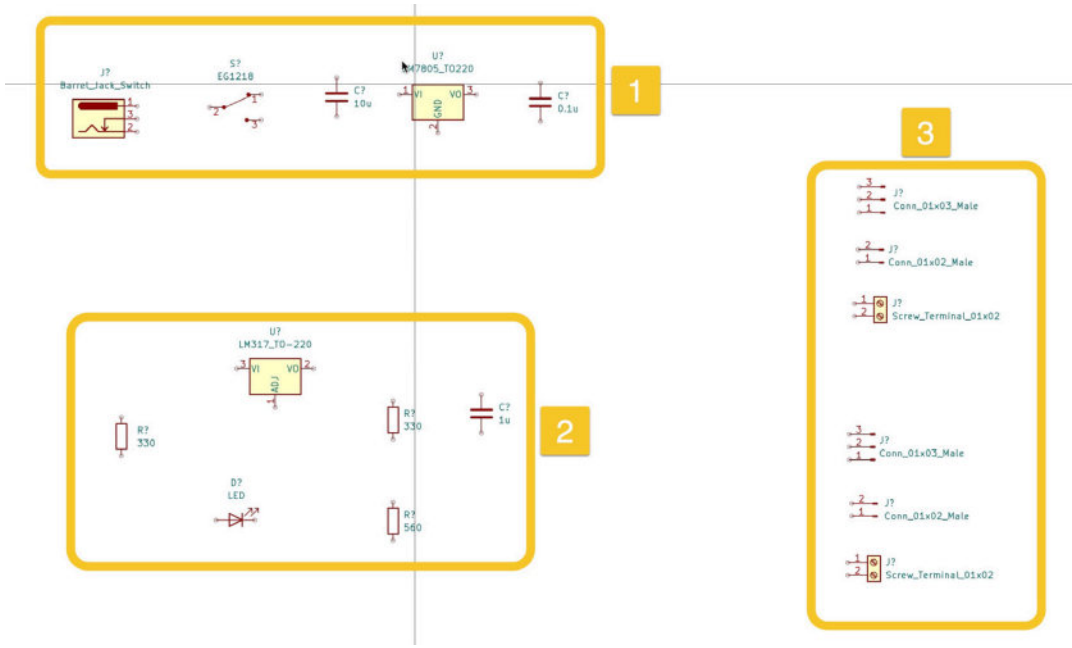


Figure 9.2.4.13: The final arrangement of the symbols.

I have marked the three functional groups:

1. Power input and 5V supply.
2. 3.3V supply.
3. Power output.

There are other ways to arrange the same simple circuit. For example, you could place the 7805 regulator and its capacitors in a separate 5V supply group and move the LED and its resistor to the Power input group. For this KiCad project, it does not matter exactly how you make the arrangement. You should place the symbols in a way that makes it easy to wire them in step four of the workflow.

With the arrangement complete, continue with the annotation.

Annotation

In most cases, it is best to use the automatic annotator tool. Click on the Annotator button in the top toolbar to bring up the Annotate Schematic window. It looks like this:

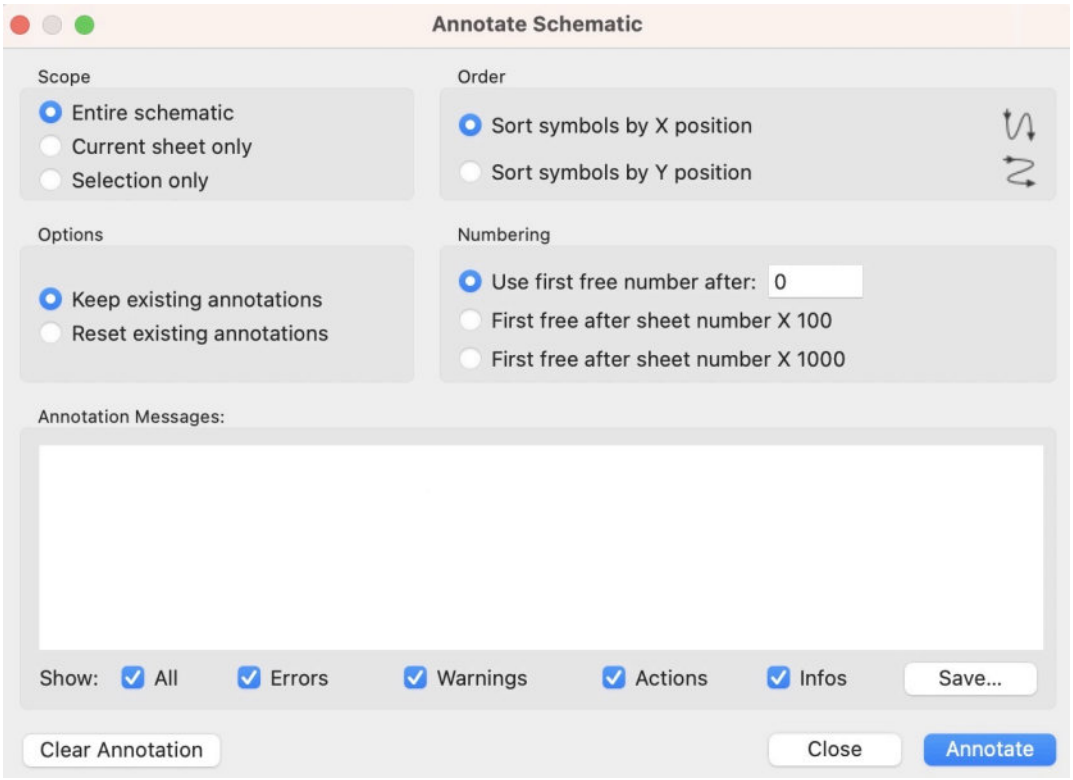


Figure 9.2.4.14: The Annotate Schematic window.

This is a new schematic, so there is no need to change the annotator settings. Click Annotate to complete the annotation and then Close.

The schematic sheet now looks like this:

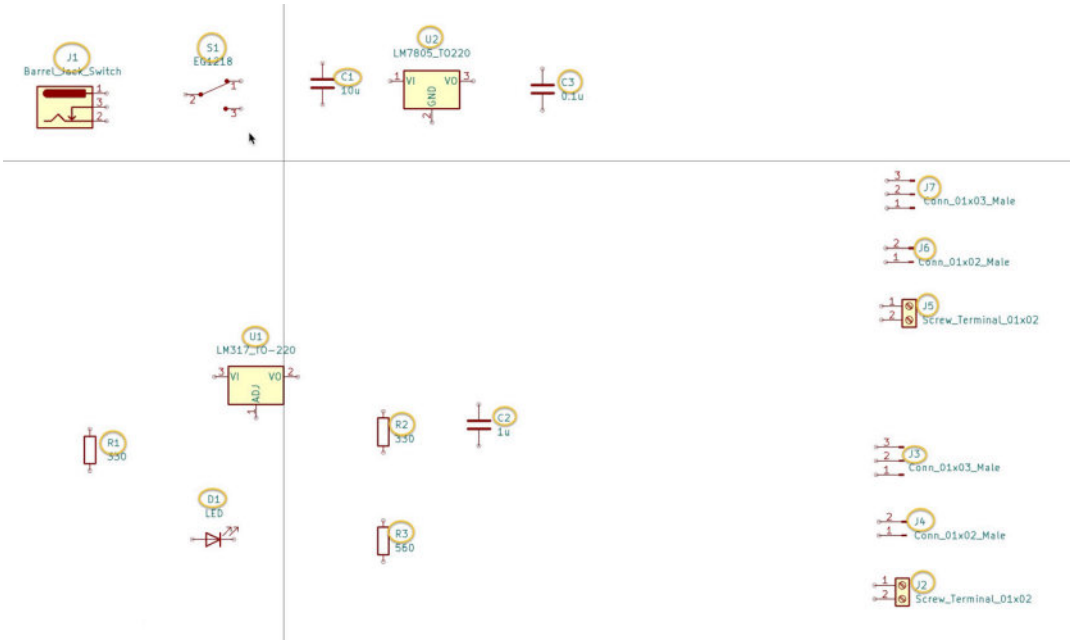


Figure 9.2.4.15: All symbols have unique identifiers.

All symbols in the schematic are now annotated with unique identifiers. Because you used the automated annotator tool, cross-check the identifiers in your schematics against those in the BOM of Table 9.2.1. If there are any

discrepancies, you can correct them manually via the symbol Properties window or keep them in mind for the remainder of the project.

Let's continue the last part of Step three, where you will do the symbol-footprint associations.

2.5. 3 - Associate

You are still in step three of the schematic design workflow. In this segment, you will complete this step by associating symbols with their footprints. You learned how to do this in a dedicated chapter earlier in this book. To keep this segment concise, I will not repeat the method ("how"), but I will show you the result of the association.

In Table 9.1.1, you can see the footprint that you will shortly associate with each symbol in your project (third column).

Each footprint reference consists of the library and the symbol name, joined by a ":". For example, take this footprint reference:

```
Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
```

This reference is for footprint "PinHeader_1x02_P2.54mm_Vertical" which you can find in library "Connector_PinHeader_2.54mm".

I will use the associations tool to do all associations in bulk instead of the slower method that involves going into each symbol's properties window and assigning a footprint in the Footprint field. If you need a refresher on how to use the associations tool, read the relevant chapter.

Go ahead and do the associations. By the end of the process, your associations table will look like this:

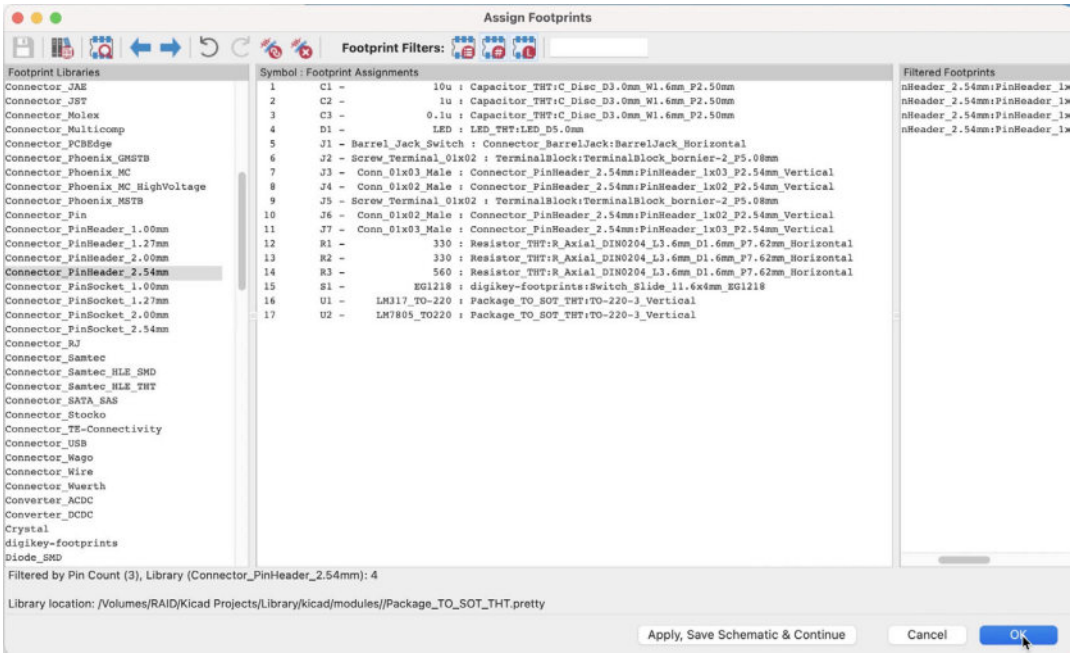


Figure 9.2.5.16: The final associations.

This completes step three of the workflow. Let's continue with the wiring in the next segment.

2.6. 4 - Wiring

In this segment, you will complete step four of the schematic workflow, adding wires to the schematic.

There are two ways to connect symbol pins:

1. Use line wires.
2. Use labels.

Since this is your first non-trivial project, you will primarily use line wires. To connect pins in symbols that belong to different functional groups, you will use net labels.

Because you arranged the symbols according to function in step three, the wiring will be easy. You will make most connections between pins that are close to each other.

When you draw the wires, remember to:

- Keep wire length to a minimum.
- Avoid drawing a wire over another wire.
- Use 90-degree angles.
- Use line wires to connect pins within the same functional group.
- Use labels to connect pins from different functional groups.
- Completely wire a group before moving to another group.
- Don't forget to create a net label for cross-group connections.

◦ Don’t forget to add power flags to the ground and other power nets.

These guidelines generally help create clean, readable schematics.

I started wiring my schematic from the power input group and continued towards the output.

Below you can see the completed wiring of the power input group.

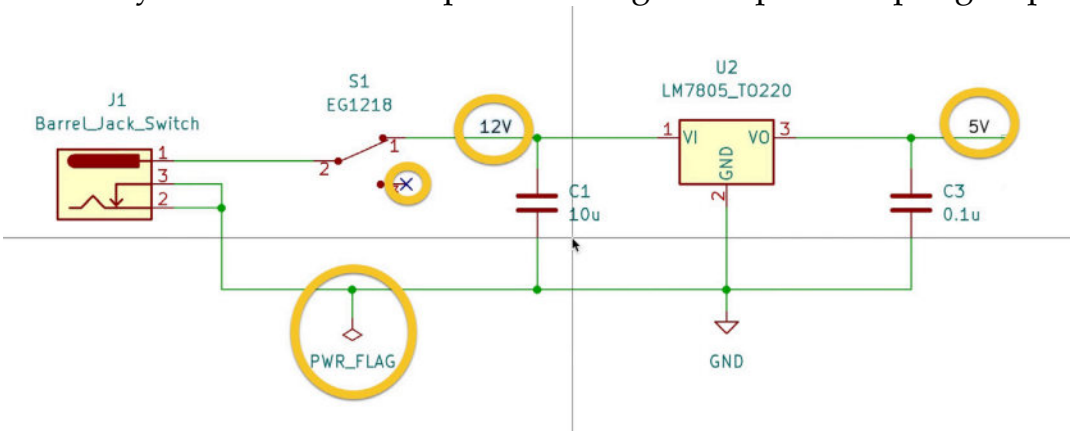


Figure 9.2.6.17: Completed wiring for the power input group.

I have marked in yellow circles the two net labels (“12V” and “5V”) and the power flag symbol (“PWR_FLAG”) attached to the GND network. I have also used a GND symbol connected to the U2 GND pin. The GND symbol automatically attaches the “GND” net label to the connected wires. This means that you don’t need to create and attach a “GND” net label manually.

You can find the PWR_FLAG symbol in the symbol chooser or in the specialized Power Symbol chooser (you will find the button for this window below the symbol chooser button in the right toolbar).

Also, notice that I have attached an “unconnected pin” symbol to pin 3 of S1. If you don’t do this, the ERC will bring up an unconnected pin violation.

Continue with the 3.3V group. Below is the result of this work:

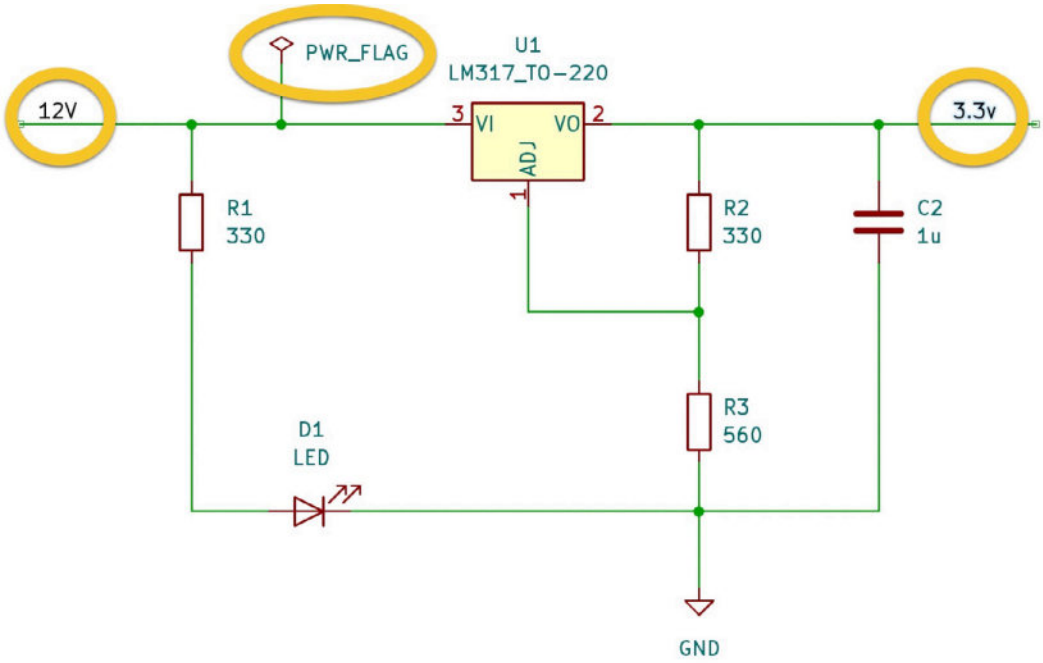


Figure 9.2.6.18: Completed wiring for the 3.3V group.

Finally, let's wire the power output group:

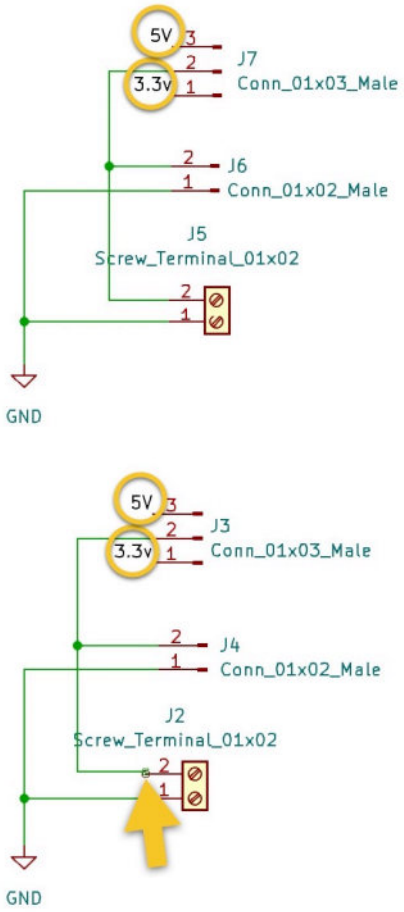


Figure 9.2.6.19: Completed wiring for the power output group.

In the figure above, I have marked a violation with an arrow. I did not correctly attach the wire to pin 2 of J2. The ERC will pick this violation later

but at this point, I want to leave it in the schematic so that I can demonstrate this common mistake shortly.

The wiring is now complete. This is an excellent opportunity to run the Electrical Rules Checker to find any problems with the schematic. Bring up the ERC tool from the top toolbar. When the ERC window appears, it will indicate that the schematic is not fully annotated (see "1" below).

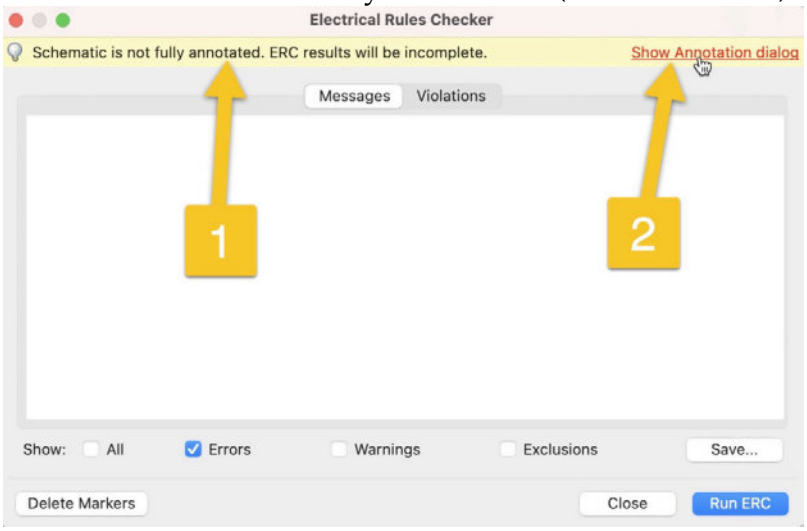


Figure 9.2.6.20: The ERC indicates the schematic is not fully annotated.

How can that be? You did the annotation in the previous step. Yes, but since then, you have added several new symbols: GND and POWER_FLAG. These are symbols that also must have a unique reference designator. Click on the "Show Annotation dialog" link ("2"), and run the annotator once again (see below).

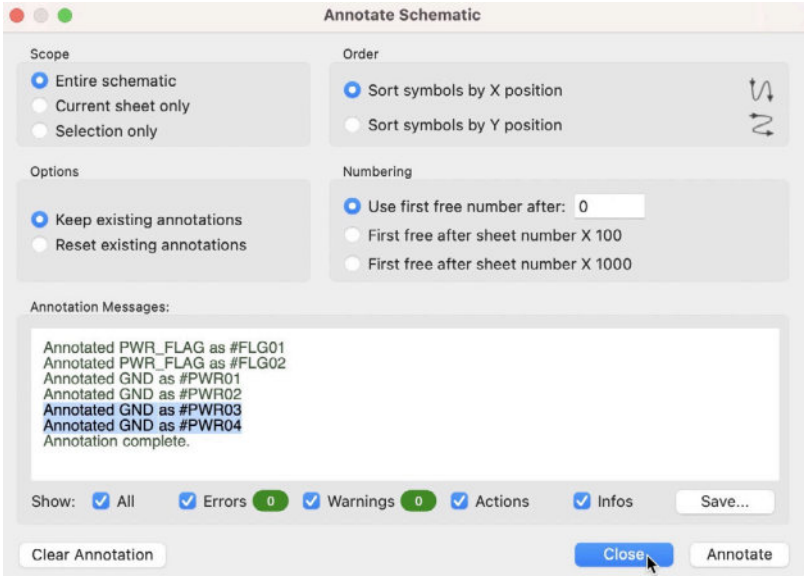


Figure 9.2.6.21: Annotation completed with no errors.

In the annotator window messages, you can see that six remaining symbols were annotated. Click Close and return to the ERC window. Click

“Run ERC”, and notice that one violation is reported: an unconnected pin in J2:

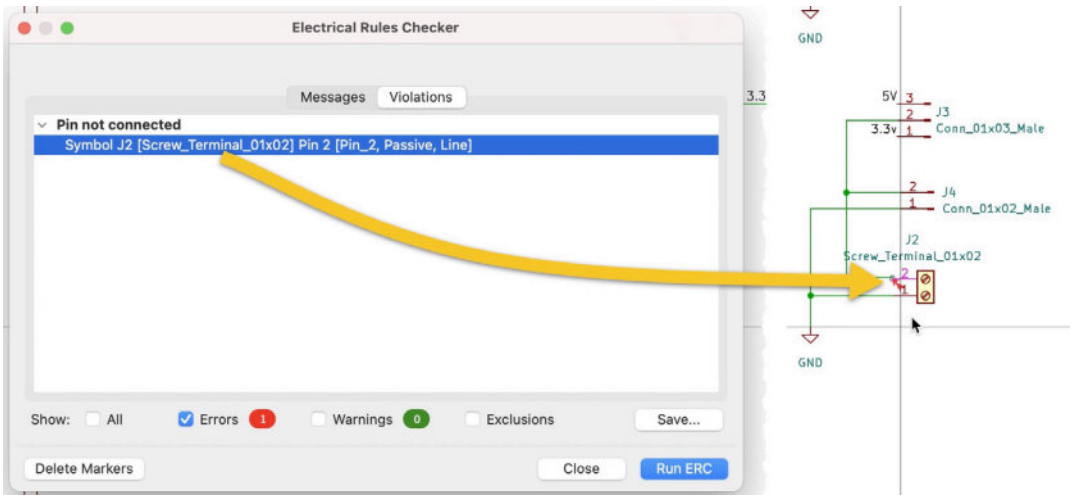


Figure 9.2.6.22: A common error: “pin not connected”.

This is the error I made earlier. The wire was not appropriately connected to pin 2 of J2, but I did not notice it until the ERC brought it up. Go ahead and fix the error. Rerun the ERC, and ensure there’re no more electrical errors.

This is the schematic at this point in the process, with the ERC showing no electrical violations:

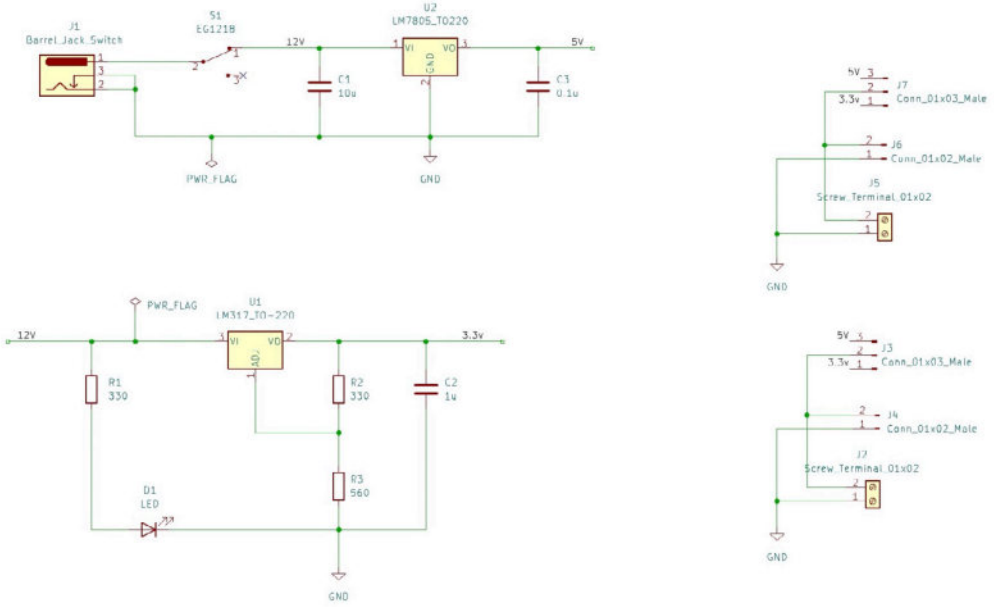


Figure 9.2.6.23: The schematic, fully wired.

This completes step three of the schematic workflow. In the next segment, I have combined steps five (nets) and six (ERC). This is because I have already done the bulk of the work relating to setting up nets, and the

ERC for this simple circuit is clear of violations. Nevertheless, I will take the opportunity of the next step to double-check my work.

2.7. 5 & 6 - Nets and Electrical Rules Check

In this segment, I will combine workflow steps five and six. This is because, in the wiring step, I created several nets and ran an ERC. As a result, the work that typically takes place in steps five and six is practically completed.

But, this is an excellent opportunity to review.

Nets

Open the Schematic Setup window, and click on Net Classes (under Project). As you can see in the Nets table, there are four named nets ("12V", "3.3v", "5V", "GND"), and several nets named automatically by the editor.

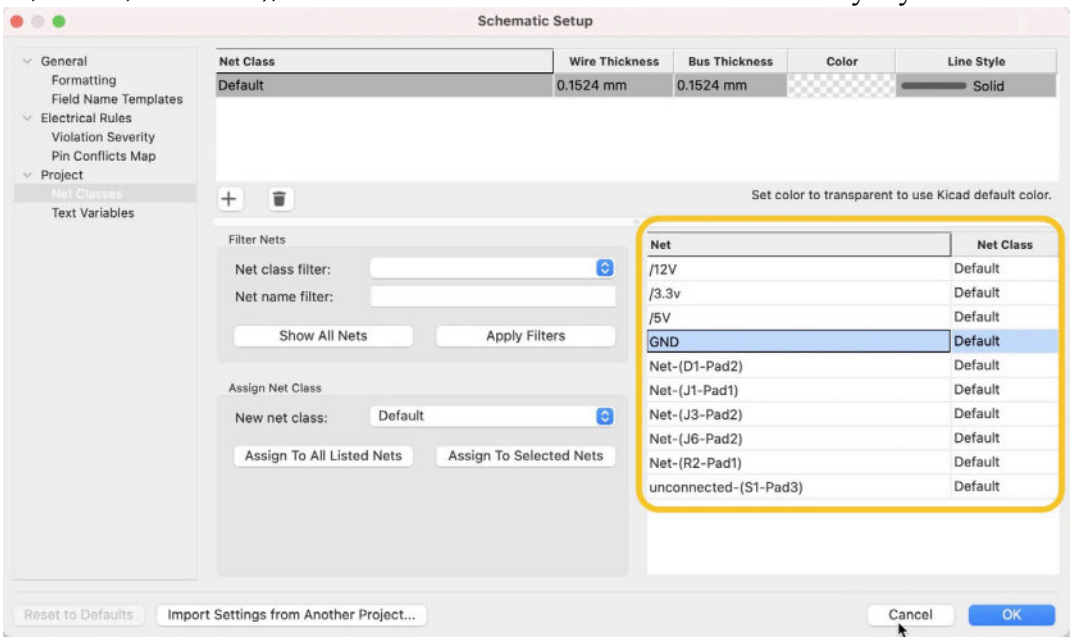


Figure 9.2.7.24: The schematic nets and net classes.

Apart from the existing nets, I'd like to add a couple more. I list them below:

1. A power input net, "PWR_input," for the wires and pins that connect to pin 1 of the barrel connector.
2. A power output net, "PWR_output," for the wires and pins that connect to the positive voltage of the screw terminal and pin connectors in the power output group.

You can see the new nets below (marked with a yellow oval):

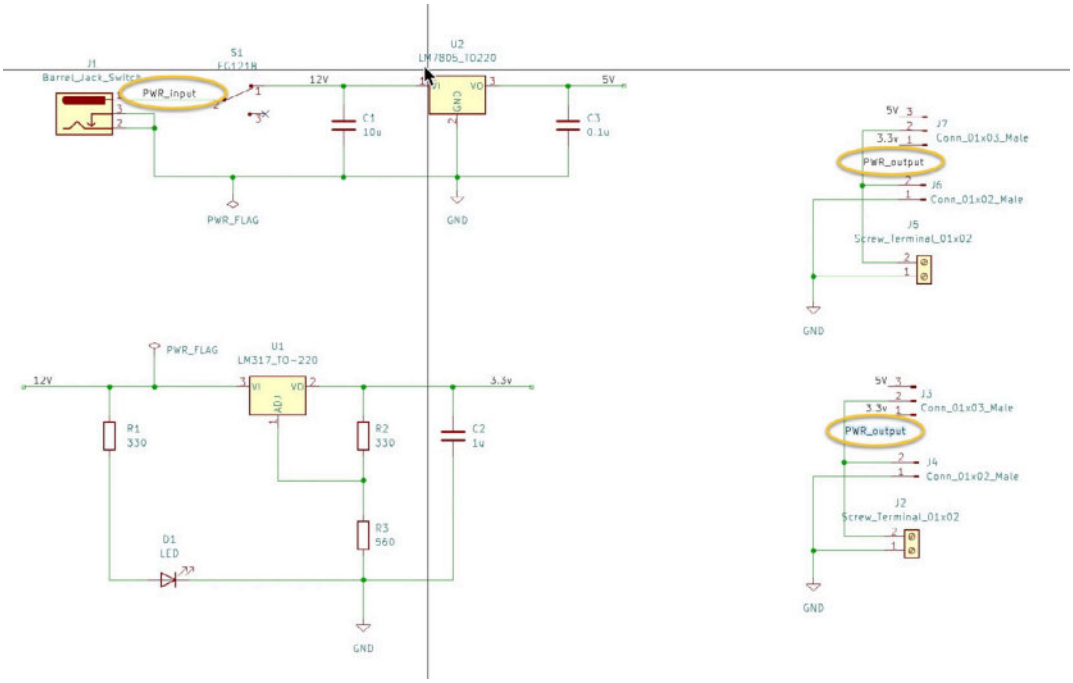


Figure 9.2.7.25: Two additional nets.

Go back to the Schematic Setup window, click on Net Classes, and confirm that the new nets are listed in the Nets table. In addition to the nets, let's create two new Net Classes: "power_input" and "power_output." Assign nets "12V" and "PWR_input" to the "power_input" class, and "3.3V", "5V" and "PWR_output" to the "power_output" class. The remaining nets can stay in the Default net class. The result is below:

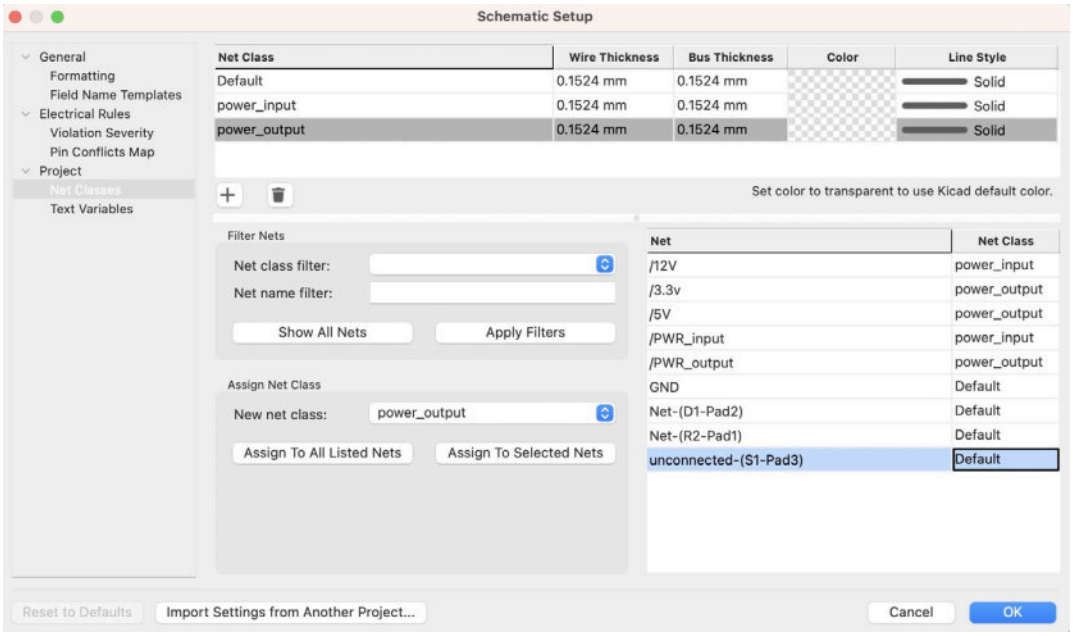


Figure 9.2.7.26: Final net and net class setup.

With Nets and Net Classes completed, we can do a final ERC.

Electrical Rules Check

I don't expect to see any violations, but I will run another ERC nevertheless. Here is the result:

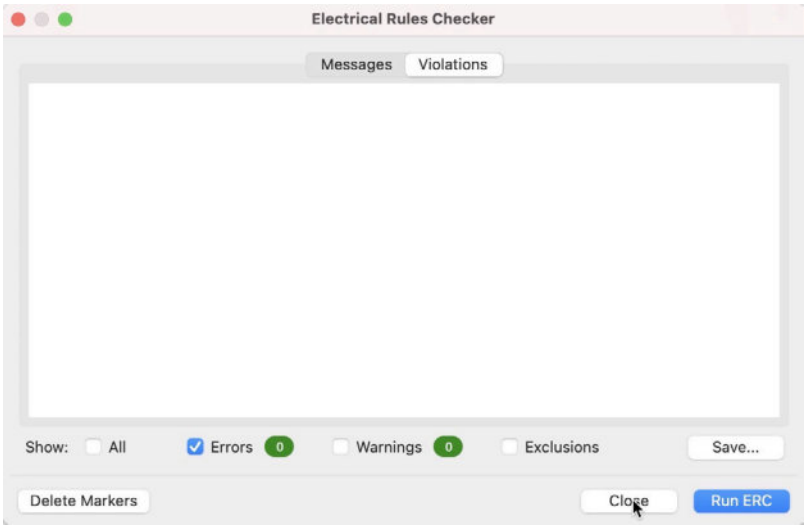


Figure 9.2.7.27: No violations.

All clear. Let's continue with step seven of the process, where you will add text and graphic comments.

2.8. 7 - Comments

In this segment, you will finish work in the schematic editor by adding text and graphic information to the sheet. This is similar to adding comments to software code. You will thank yourself later for taking the time to do this (and so will other people with whom you share this project).

Currently, the schematic looks like this:

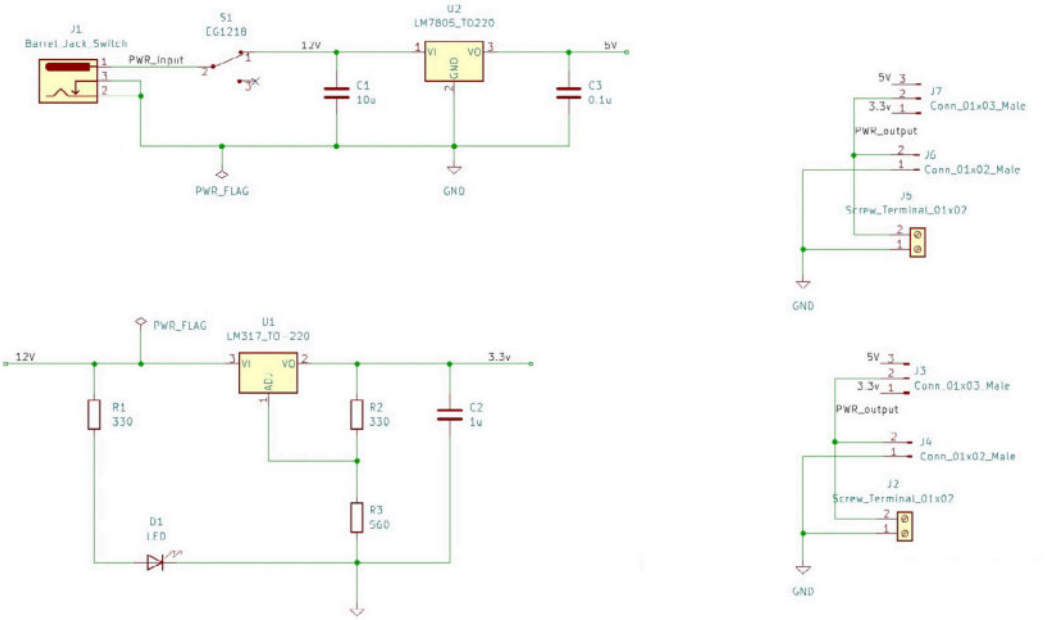


Figure 9.2.8.28: The schematic prior to commenting.

Use the graphics tools from the right toolbar to create three boxes around the three functional groups, and name them.

In addition to the regular text and graphic comments, is to use a custom field name where you can enter a short sentence that describes the purpose or function of a symbol. To add a custom field name:

1. Open the Preferences window and click on “Field Name Templates” under “Schematic Editor” (learn more about this).
2. Add a new row, and type “Purpose” in the field name.
3. Check the “Visible” box so that the contents of this custom field appear in the schematic editor.

   Here mine below:

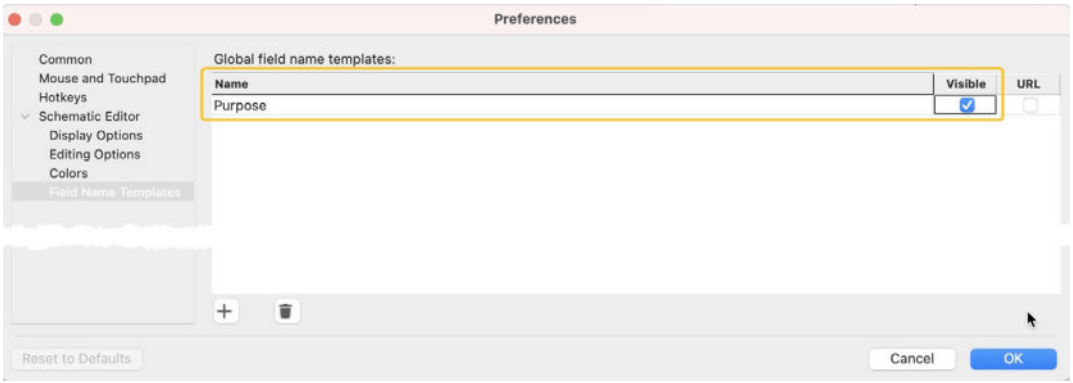


Figure 9.2.8.29: A custom field.

Let’s add some text to this custom field for some of the symbols. Double-click on the LED symbol, and add this text in the Purpose field: “Power input indicator.” See the LED symbol properties window below:

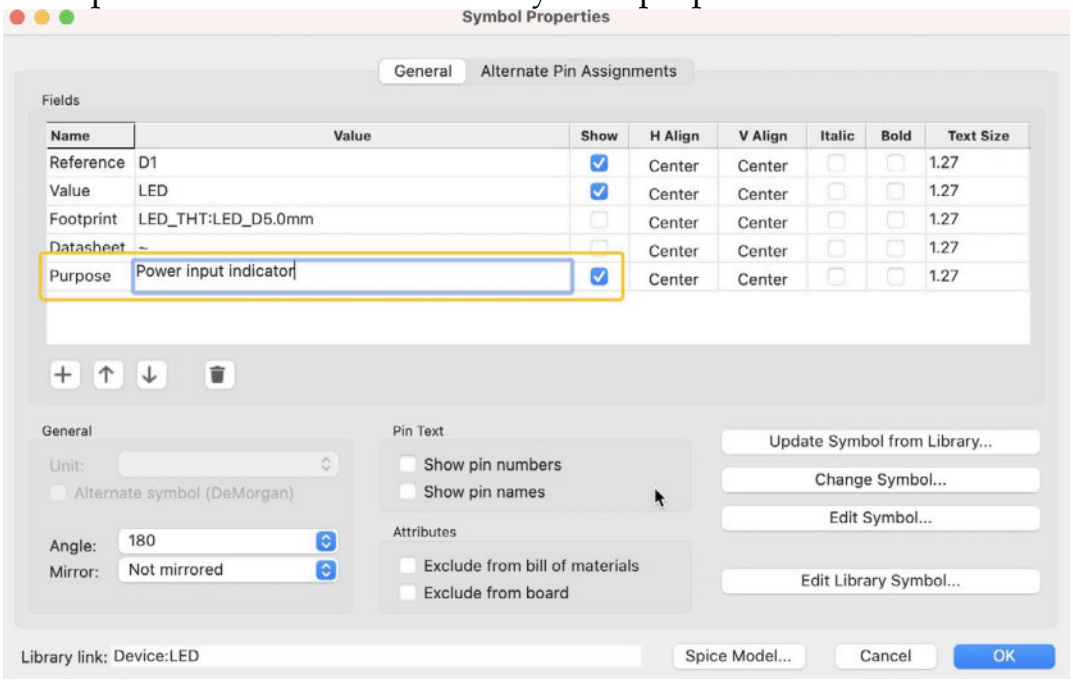


Figure 9.2.8.30: This LED has a purpose.

Click OK and notice that the text for the new field appears in the schematic editor:

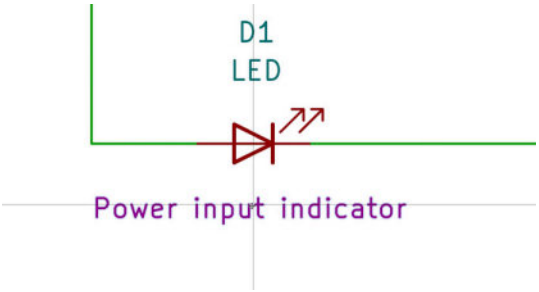


Figure 9.2.8.31: Showing the content of the Purpose field.

You can see my commented schematic, with Purpose field text for some of the symbols below:

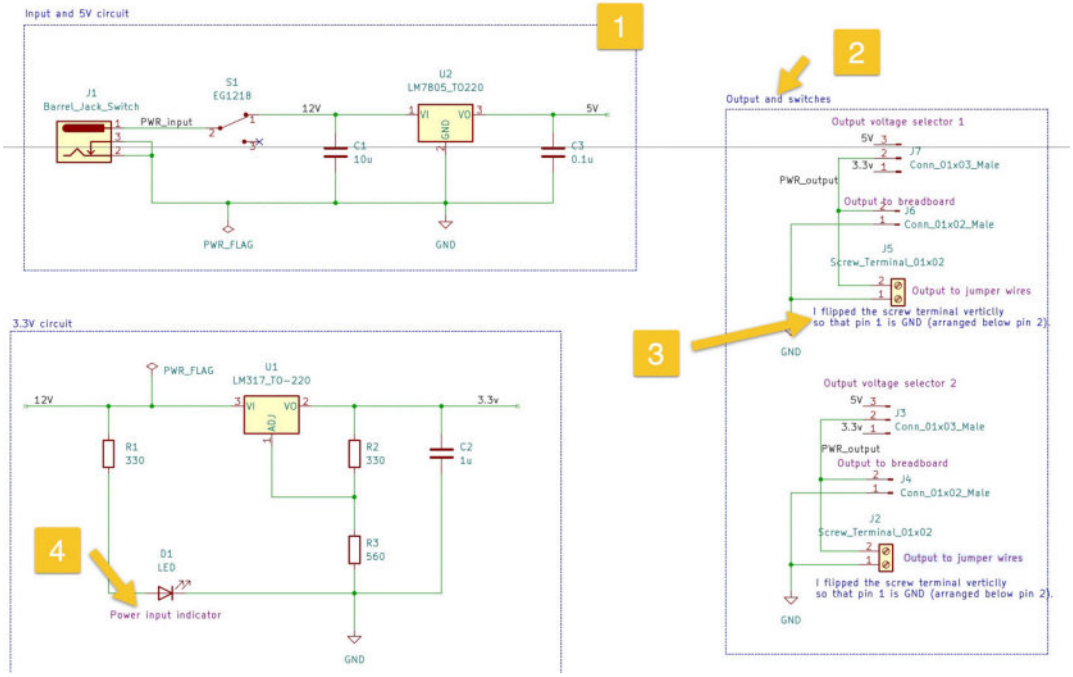


Figure 9.2.8.32: Final schematic.

I have used numbers and arrows to mark some of the graphics, text, and field values I have used:

1. Using graphic lines to create a border around a functional group.
2. Using a text label to give a name to the functional group.
3. Using a text label to provide information about something non-obvious (in this case, I used the flip function to re-orient a symbol).
4. An example of text in the Purpose custom field.

Go ahead and complete this step. This completes the schematic design workflow. Ready to proceed with the layout? Follow along in the next chapter.

3. Layout design editing

In the previous chapter, you completed the schematic design of the breadboard power supply PCB. In this chapter, you will work on the layout design following the layout design workflow from Part 6 of this book. You can see this workflow below:

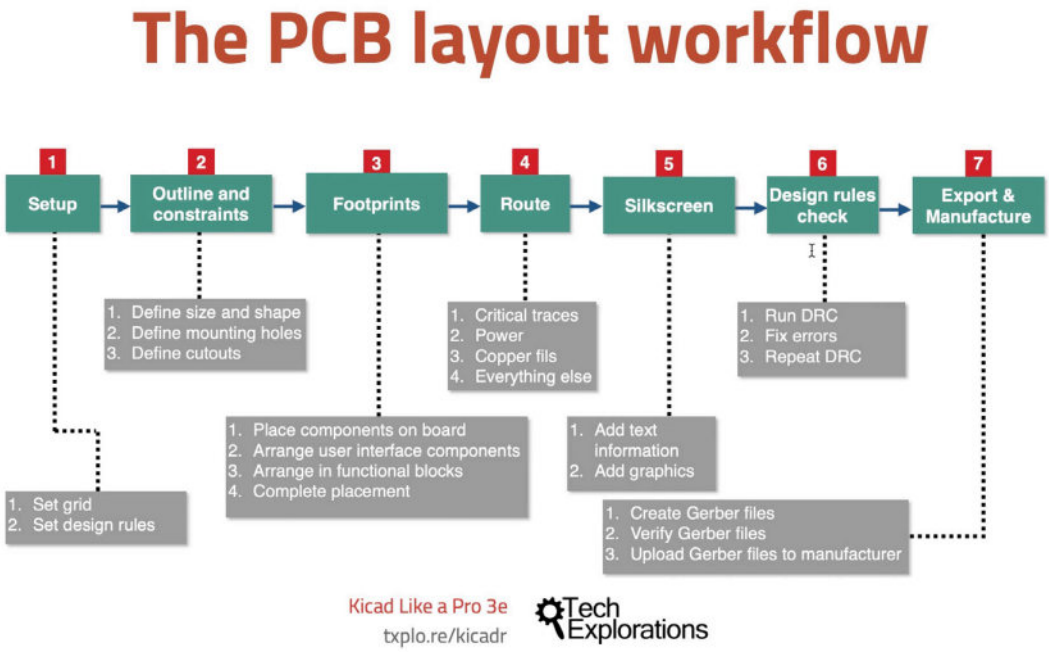


Figure 9.3.1: The layout design workflow.

The layout design work is constrained by the physical attributes of the mini breadboard on which the power supply will attach. In the photo below, I have placed an earlier version of the power supply PCB over a mini breadboard (the shape remains unchanged).

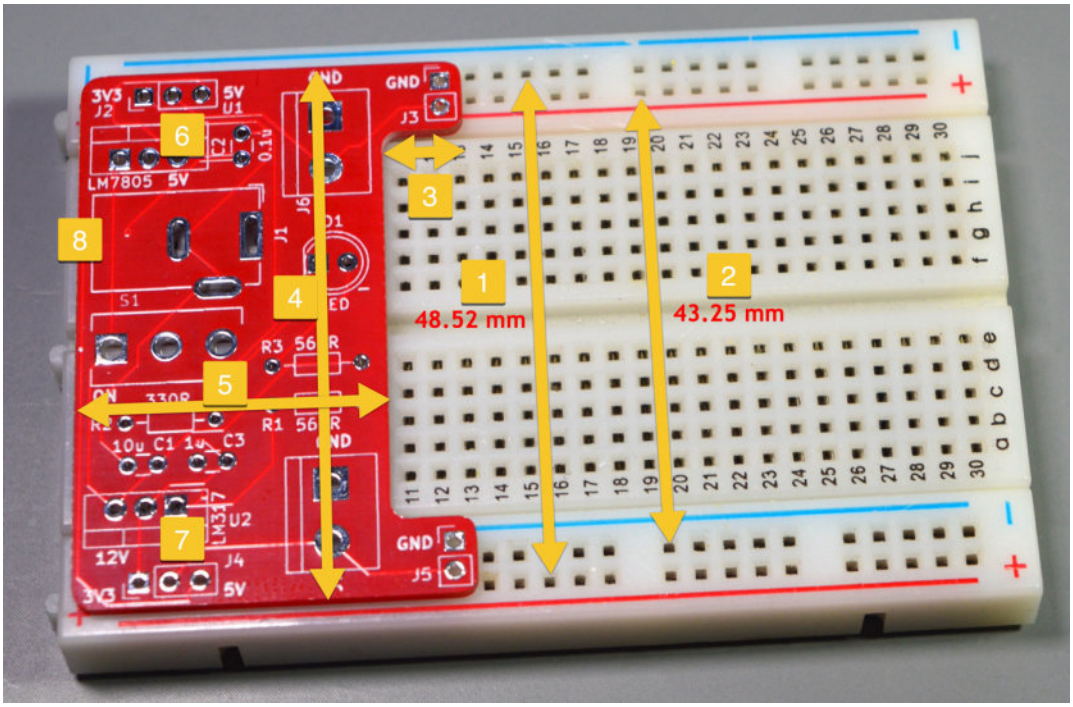


Figure 9.3.2: The distances between the power rows dictate the dimensions of the PCB.

The height of the PCB, “4” (i.e., the distance between its top and bottom edges as you are looking at it in the figure above), is dictated by the way it will attach to the mini breadboard. The primary constraints are the distances between the two blue and red power rails (“1” and “2” respectively in the figure above). The larger number (“1”) dictates the height of the PCB (“4”). For the breadboard power supply to work, you must ensure that the two double pin connectors (J3 and J5) are placed within a very small tolerance so that their outer pins are 48.52 mm apart.

Also, in the figure above, you can see that the two double pin header holes are placed within two notches on the right side of the board. The width of those notches is approximately equal to the distance between two columns in the breadboard. This ensures that when we attach the power supply to the breadboard, we will not obstruct any breadboard holes.

The width of the PCB, “5”, should be sufficient to contain the components comfortably; however, there is no rigid constraint. The width is constrained primarily by the large components, especially the barrel connector.

The total height of the PCB (“4”) should be equal to or slightly less than the height of the breadboard to ensure that there is no overhanging.

I have determined all distances on the mini-breadboard by using my calliper to make multiple manual measurements. The small errors in my measurements are within the tolerances of the typical mini-breadboard, so I

am not concerned by these errors. However, an alternative way to determine the dimensions of this PCB is to rely on the breadboard specifications instead of manual measurements. A typical breadboard has pins with distances between them that are multiples of 2.54mm.

For the mini-breadboard that I use in this project, the distance between the outer rows (marked "1" in the image above) should be 19 x 2.54 mm = 48.26 mm (I measured 48.52 mm). Similarly, the distance between the two inner-outer rails ("2") should be 17 x 2.54 mm = 43.18 mm (I measured 43.25 mm). I decided to rely on my measurements because I was unable to find definitive specifications, and the gap between the outer power rails and the rest of the breadboard prompted me to not trust my calculations. After several prototyping iterations of the PCB using my manual measurements, I have concluded that the manual measurements are correct.

Other considerations that influence the layout of components rather than the shape of the PCB are:

- The two voltage selector switch must be accessible from the sides of the board. Other components should not obstruct them.
- The power supply barrel connector should be accessible from the side that is opposite to the breadboard.
- The two screw terminals provide an alternate way to provide power to a circuit. They should be placed so that wires can be attached to the terminals from the side of the breadboard.
- The on/off switch is large enough to be accessible from the top of the PCB and should be placed next to the barrel connector ("5") as it is part of the power input group of components.
- The LED indicator is also part of the PCB user interface. It should be placed on the side of the breadboard to make it easy to determine the operation status of the power supply.

The only rigid dimensions that you will need to work with are those between the blue and red power rails ("1" and "2" in the figure above). To obtain these measurements, you can use a ruler. Better than a ruler is a caliper tool, like this:

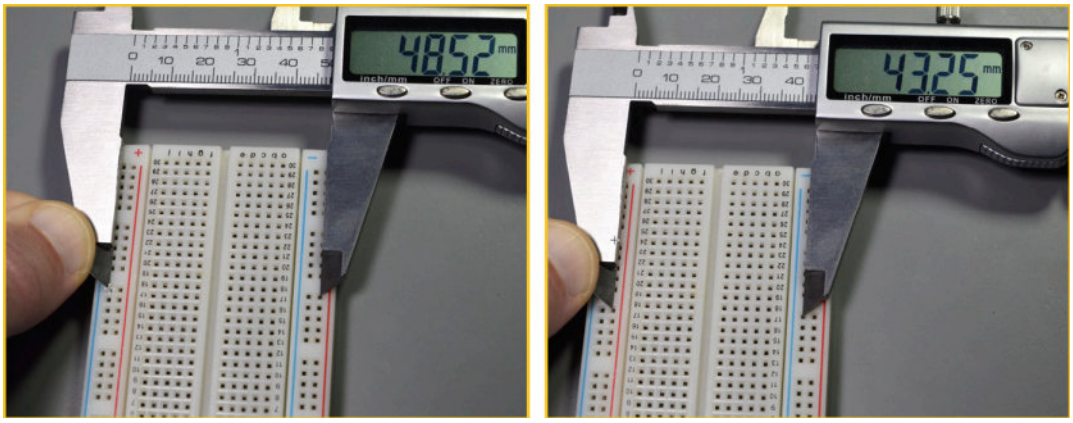


Figure 9.3.3: Use a caliber to make accurate measurements.

The objective of the layout workflow is to deliver a PCB that looks like this:

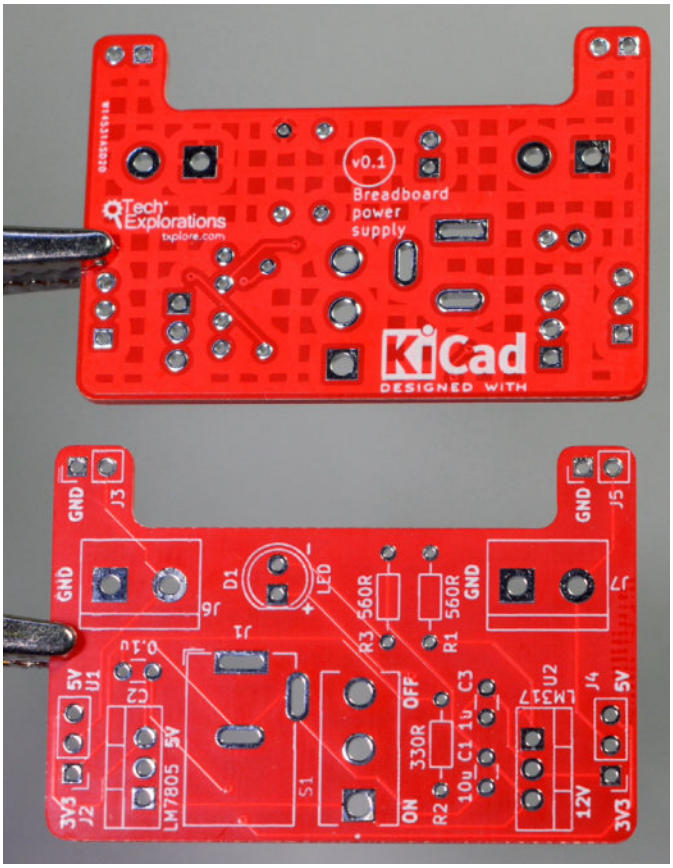


Figure 9.3.4: The layout design end product.

The board will have the dimension constraints and shape that I outlined above. It will contained the TH footprints that you already associated in the previous chapter. It will contain silkscreen and graphics on both sides. Finally, it will have a copper fill connected to the GND net in the bottom copper layer.

Now that you have set the requirement of the layout design, it is time to start work.

3.1. 1 - Setup

In this segment, you will set up the layout editor and import the PCB data from the schematic editor.

Start Pcbnew. Open the Board Setup window and review the most important settings.

- Under Board Stackup:
  - Physical Stackup, confirm that two copper layers are selected.
  - Other settings under Board Stackup are OK in their defaults.
- Text & Graphics:
  - No changes needed.
- Design Rules:
  - Constraints are OK as they are. I have used these constraints with boards manufactured with Pcbway.com, oshpark.com and nextpcb.com and had no issues.
  - Pre-defined sizes: ok to leave those blank for now.
  - Net Classes: The net class table will show the net classes you created in Eeschema. I have changed the track width of the power_input and power_output classes (from 0.25 mm to 0.35 mm). I also increased the via size to 0.9 mm and via hole to 0.5 mm. This will allow for more current to flow through the member tracks.
  - Custom Rules: no change to the defaults.
  - Violation Severity: no change to the defaults.

Below you can see the settings in the Net Class table.

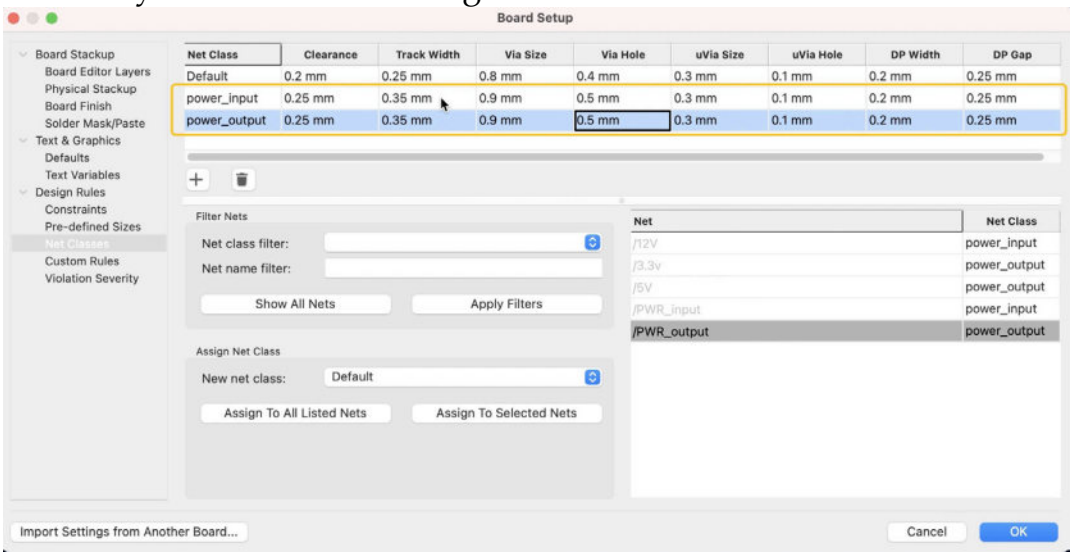


Figure 9.3.1.5: The Net Classes settings.

Go ahead and import the PCB data from the schematic editor. Click the PCB import button from the top toolbar to bring up the importer window:

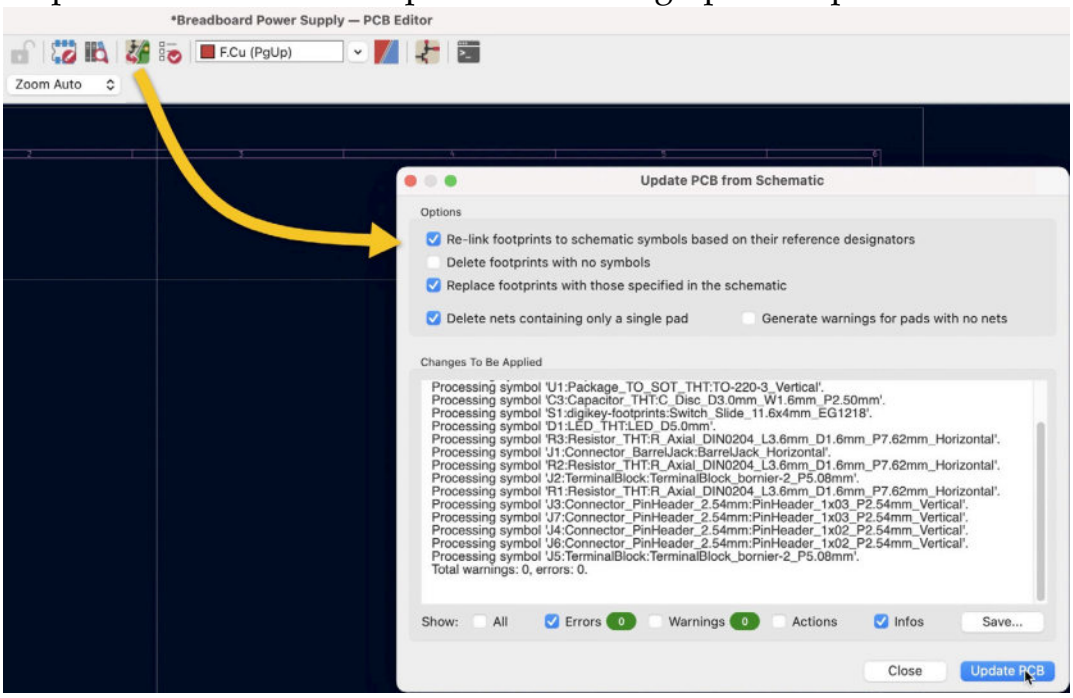


Figure 9.3.1.6: The PCB import window.

The import options are correct. Click Update PCB, and then click Close. The footprints are now clamped together in the layout editor:

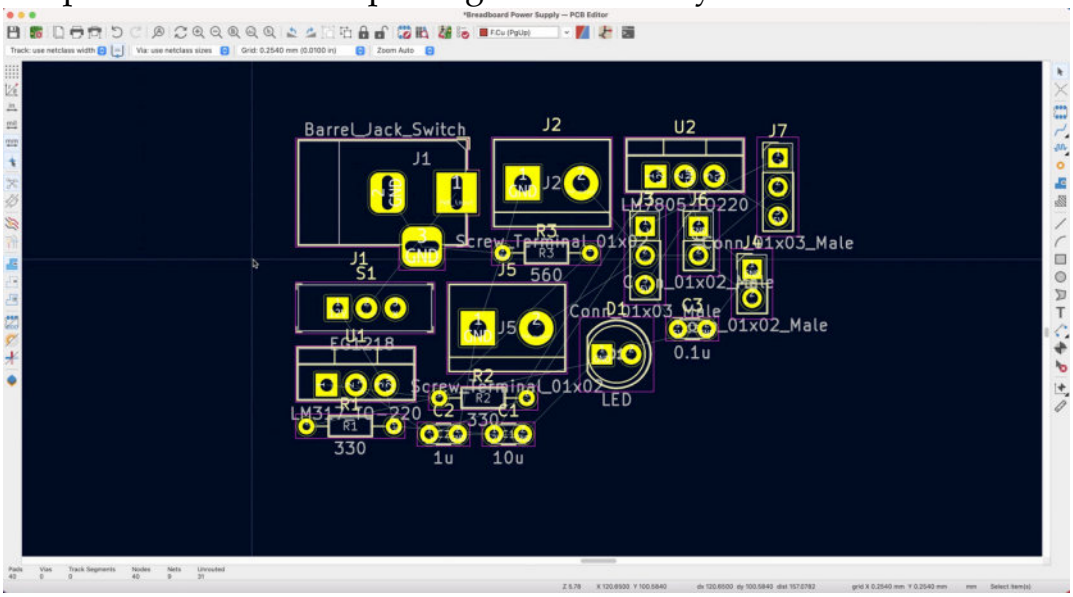


Figure 9.3.1.7: The imported footprints in the layout editor.

The first step of the workflow is complete. Let's continue with step two, where you will create a first rough outline of the board. You will refine this outline once the footprints are placed within the rough outline.

3.2. 2 - Outline and constraints

In this segment, you will use the measurements that dictate the rigid mechanical constraints for the PCB to draw a rough outline in the Edge.Cuts layer. In a standard breadboard, the distance between each set of holes is 2.54 mm. Therefore, concerning the caliper measurements I show in Figure 9.3.3, we can use either the outer or inner distance between the breadboard power rails to place the two 2-pad headers.

In my example below, I will go with the inner measurement of 43.25 mm.

Start by moving the two footprints for the 2-pad headers away from the rest of the footprints in a free region of the layout editor. Below, I have moved J6 and J4 towards the top-left side of the editor and the rest of the footprints at the bottom right corner.

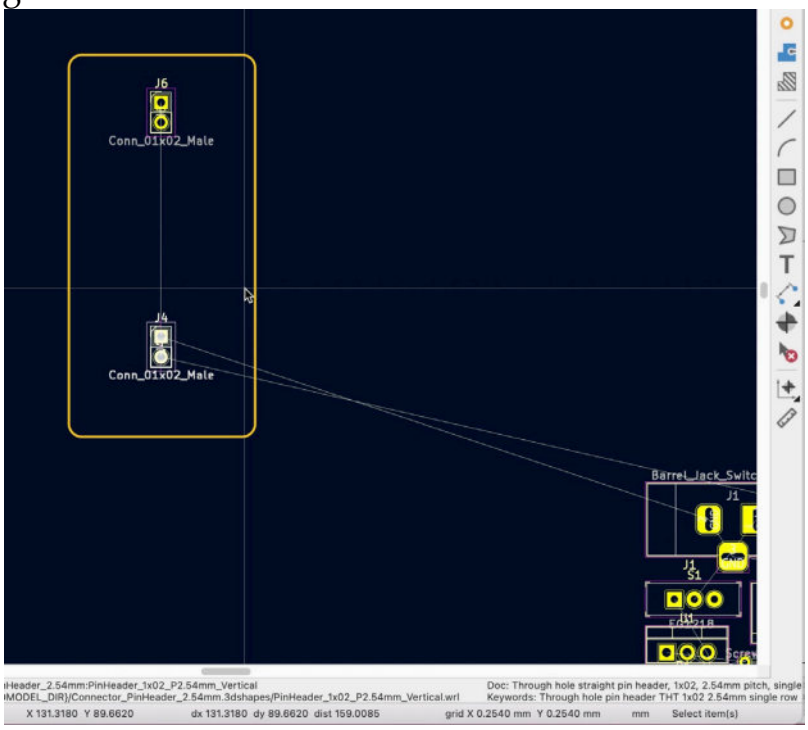


Figure 9.3.2.8: Working on J6 and J4.

Before arranging the two footprints in the correct distance against each other, you must consider their orientation. In the breadboard, the outer and inner power rails are arranged in pairs. The two outer power rails are blue and red, and the two inner power rails are red and blue. The pad headers must match the same arrangement. Zoom in J6 (which I have placed on the top side above) so you can see the pad net names. Orient J6 so that GND is placed up to match with the position of the blue power rail.

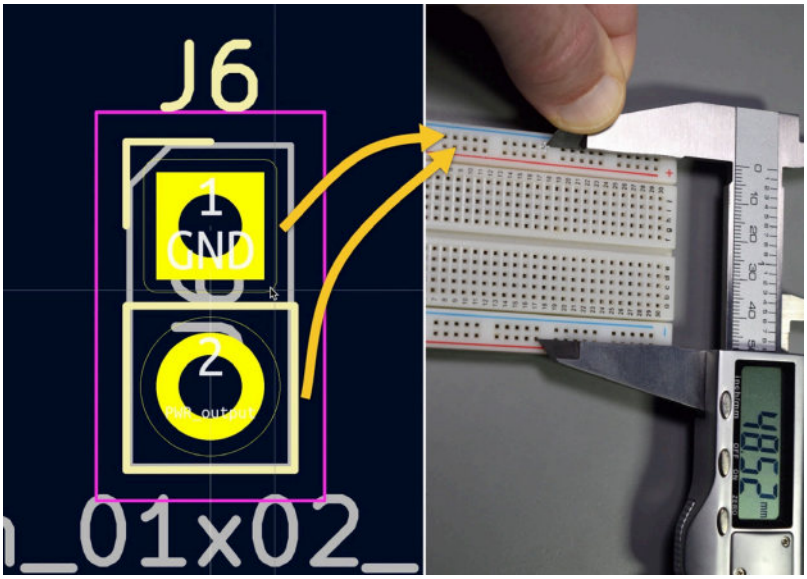


Figure 9.3.2.9: Set the orientation for the top pad header.

Repeat the process for J4 (the bottom pad header). For J4, pad 1 (GND) is also placed up to match with the blue power rail:

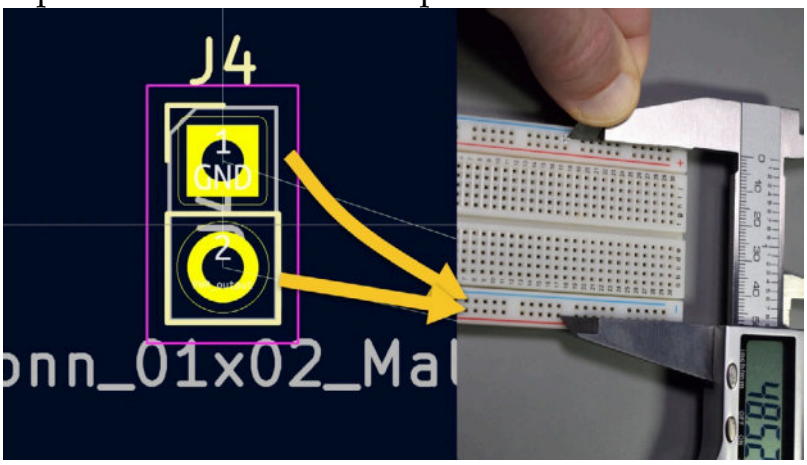


Figure 9.3.2.10: Set the orientation for the bottom pad header.

The two pad headers are now in the correct orientation. Proceed to set them at the correct distance (43.25 mm) using their inner pads (pad 2 of J6 against pad 1 of J4) as a reference.

Here is the technique that I use for tasks like this:

1. Chose one of the footprints to remain in position and the other that will move freely. I will keep J6 in place and move J4.
2. Because you want to make precise measurements and placement, select a small grid. For this example, 0.0508 mm works well.
3. Click on J4 to select it.
4. Move the mouse pointer to the middle of pad 1. You will measure the distance between pad 1 of J4 (selected) and pad 2 of J6 (static).
5. Zoom out as needed, and move the selected J4 so that J4 pad 1 is exactly over J6 pad 2. See the image below for a visual:

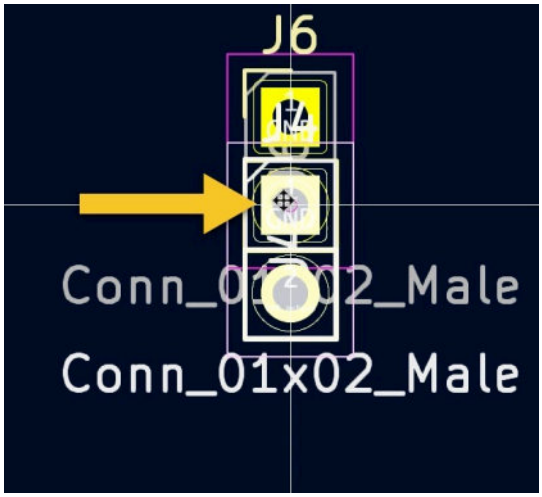


Figure 9.3.2.11: J4 pad one and J6 pad two overlap. This is where to start the distance measurement.

6. Press the space bar to reset the dx and dy values in the status bar.
7. Zoom out.
8. Select J4 if not already, and start moving it downwards. Keep an eye on the dy value, and ensure that dx remains 0 mm (to ensure that the is no horizontal movement of J4).
9. Continue moving J4 until dy becomes 43.25 mm or very close to it. With the grid size of 0.0508 mm, I was able to get dy to 43.2816 mm. This is within 1% of my caliper measurement. Considering that my measurement also contains a margin of error, I will accept a dy of 43.2816 mm.

You can see the final position of J4 below:

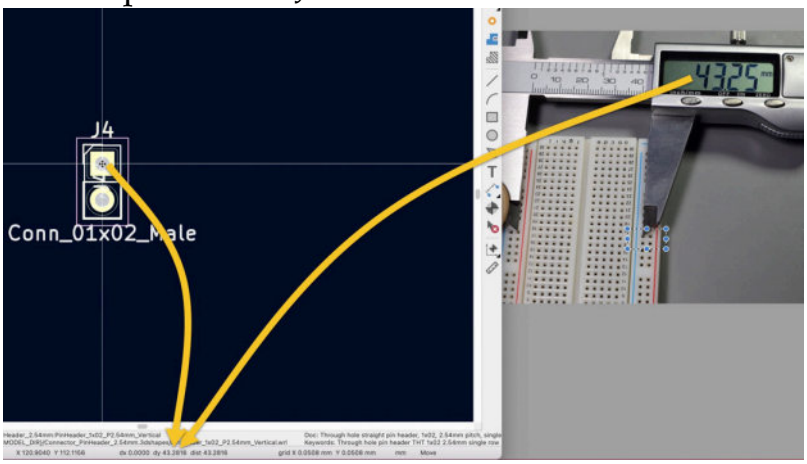


Figure 9.3.2.12: J4 in its final position.

Before doing any more work, lock the two footprints in position. Select both using click and drag to draw a rectangle around them, then right-click for the context menu and select Lock from the Locking submenu:

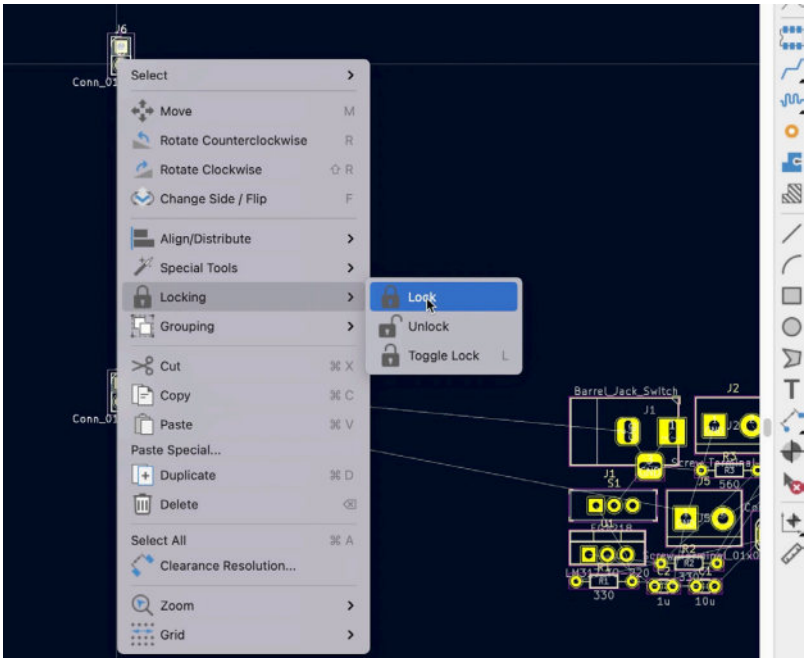


Figure 9.3.2.13: Locking J4 and J6.

Locking J4 and J6 will prevent accidentally moving these footprints in the future.

Go ahead to draw a rough outline for the board. This will give you an area to work within the next step of the workflow, where you will place the rest of the footprints within the outline of the PCB.

Select the Edge.Cuts layer from the Layers tab in the right toolbar, and then the rectangle drawing tool. Increase the grid size to 0.254 mm. Draw a rectangle that contains J4 and J6, with a generous amount of space on the left side of the connectors to hold the rest of the footprints. In my example below, I have drawn a rectangle with a height of 52.578 mm and a width of 35.560 mm.

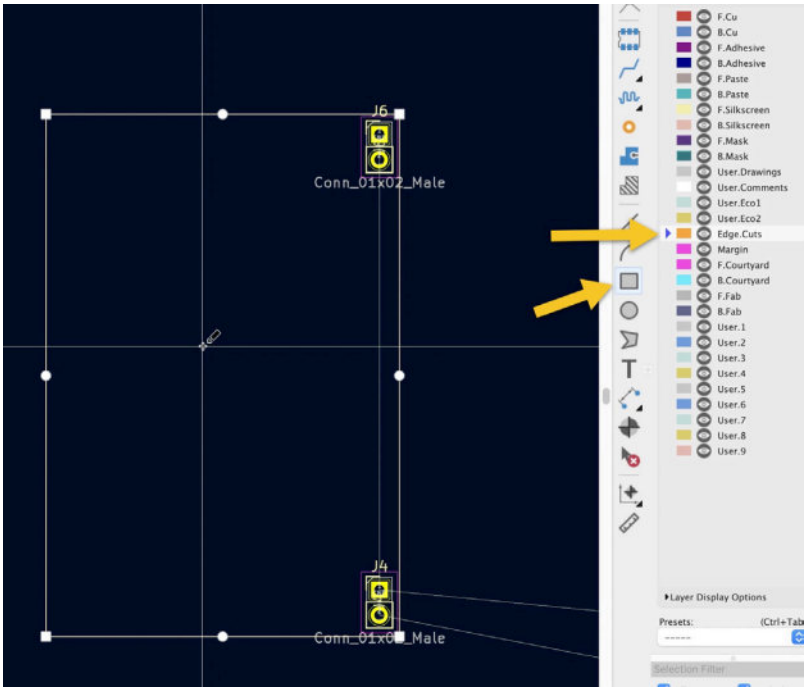


Figure 9.3.2.14: A rough outline for the PCB.

You have now satisfied the rigid mechanical constraint of this PCB and have created a rough outline that will help position the remaining footprints within the PCB in the next step. You can go ahead and work on the footprint placement.

3.3. 3 - Place footprints

In terms of the effect they have on the geometry of the layout, the two most important footprints in this PCB are J6 and J4. You already placed those footprints in a locked position and used them to draw the rough PCB outline. In this segment, you will place the rest of the components within this rough outline.

I will start with the two screw terminals (J2 and J5) because they are electrically connected directly to the two already placed pin headers. Place J2 next to J6 and J5 next to J4. Be careful to orient the footprints so that the screw terminal openings face outwards. Below you can see the correct placement of J2 next to J6:

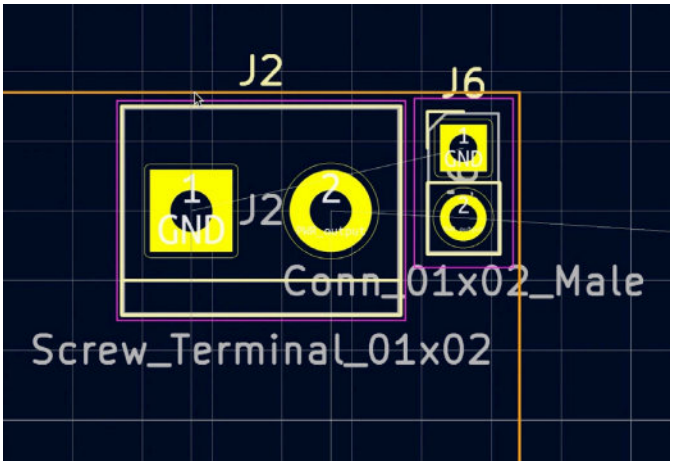


Figure 9.3.3.15: Placement of J2.

If the footprint reference designators in your instance of the project are different from what you see in these figures, use the ratsnest lines to determine the groupings of the footprints. For example, match the screw terminal footprint with the pin header that has the shortest ratsnest lines.

Place J5 next to J4, close to the bottom edge of the outline. To ensure that the two footprints are properly aligned, select them both (hold down the Shift key and click on each footprint to select it), then right-click for the context menu, and select "Align to Right" from the "Align/Distribute" submenu:

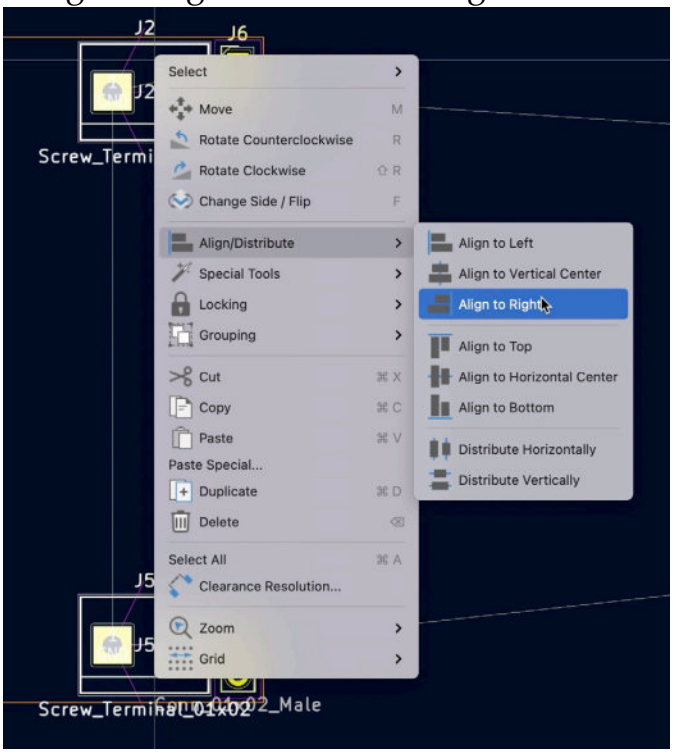


Figure 9.3.3.16: Align to right.

You can do some more alignment work between the J2-J6 and J5-J4 pairs. For example, align J2-J6 to top, like this:

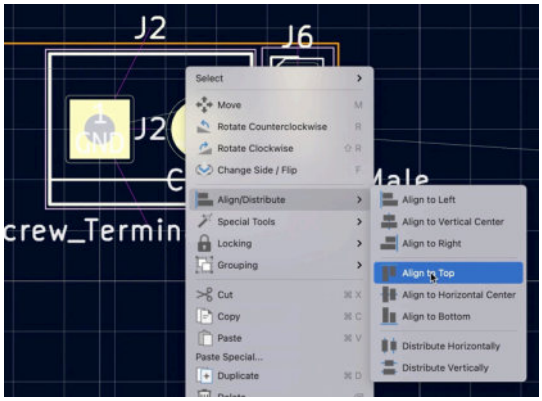


Figure 9.3.3.17: Align to top.

Use the same process to align J5-J4 to the bottom. Use the alignment tools to ensure that footprints are aligned against their neighbours and that no footprint is out of place.

You spent a lot of time positioning the screw terminal footprints. To prevent from accidentally moving them out of place, lock them before you continue. Below is a view of the PCB at this point:

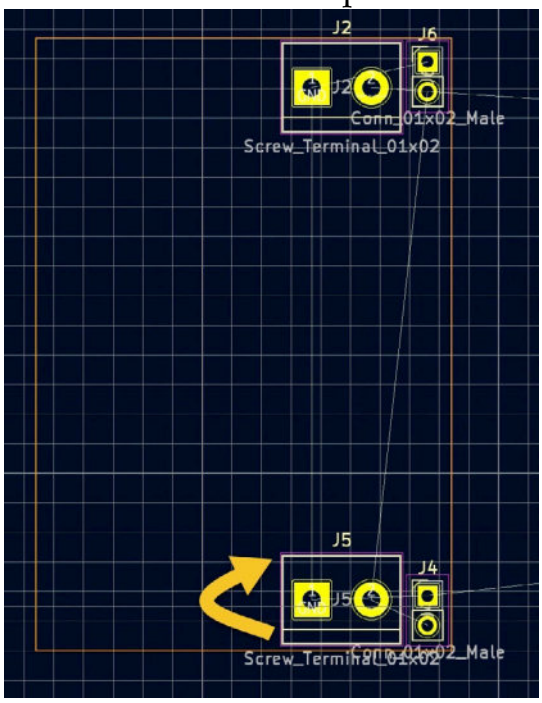


Figure 9.3.3.18: J5 should be rotated by 180 degrees.

Unfortunately, when I was working on this project, at this time, I did not notice that J5 was oriented incorrectly. The screw terminal openings must point towards the outside of the board; however, in the figure above, they point inwards. Keep this in mind for now, and I will fix it later in the workflow. Beware that screenshots later in this chapter may still show J5 with an incorrect orientation.

Let's continue with the placement of the other footprints. Don't lock anything in place. After the first placement, refine until you are satisfied and then lock the footprints in place.

In my footprint placement process, I continued with this sequence:

1. Footprints along the perimeter:
   - Switches J3 and J7.
   - Barrel connector J1.
   - LED D1.
2. Large internal components:
   - Voltage regulators U1 and U2.
3. Small internal components:
   - Resistors.
   - Capacitor.

Below you can see my first iteration of the footprints placement:

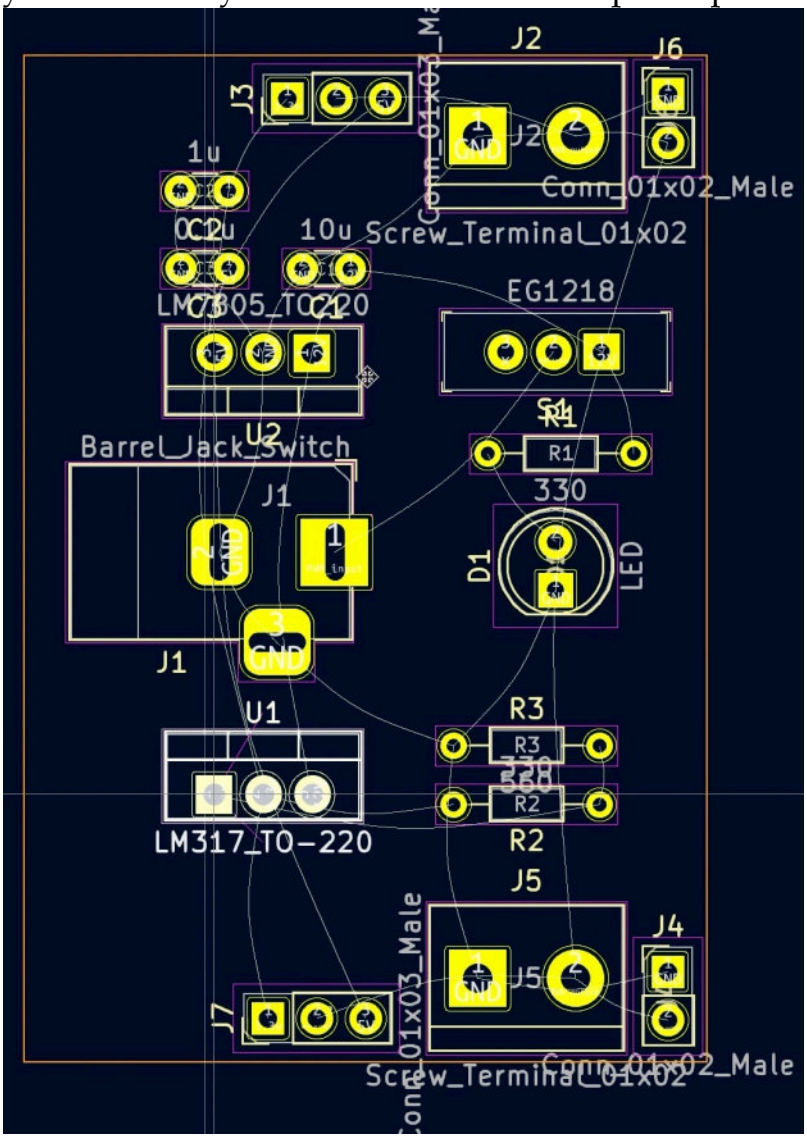


Figure 9.3.3.19: First iteration of footprints placement.

This rough placement is subject to significant improvements. Below I list some thoughts I had as I was looking at the layout in the figure above:

1. There is a lot of wasted space on the left side. I can reduce the cost of manufacturing by reducing wasted space.
2. I have placed the on/off switch (S1, EG1218) too far from the barrel connector and at a location that is not convenient to access.
3. The footprints on the right side (except for J2, J6, J4, and J4) are too close to the right edge. I want to move that edge towards the left to avoid blocking access to the two breadboard columns underneath it.
4. The small components (capacitors and resistors) can be moved to reduce the length of copper traces that connect them to the circuit.

With all this in mind, I made a second iteration of the footprints placement. The result is below:

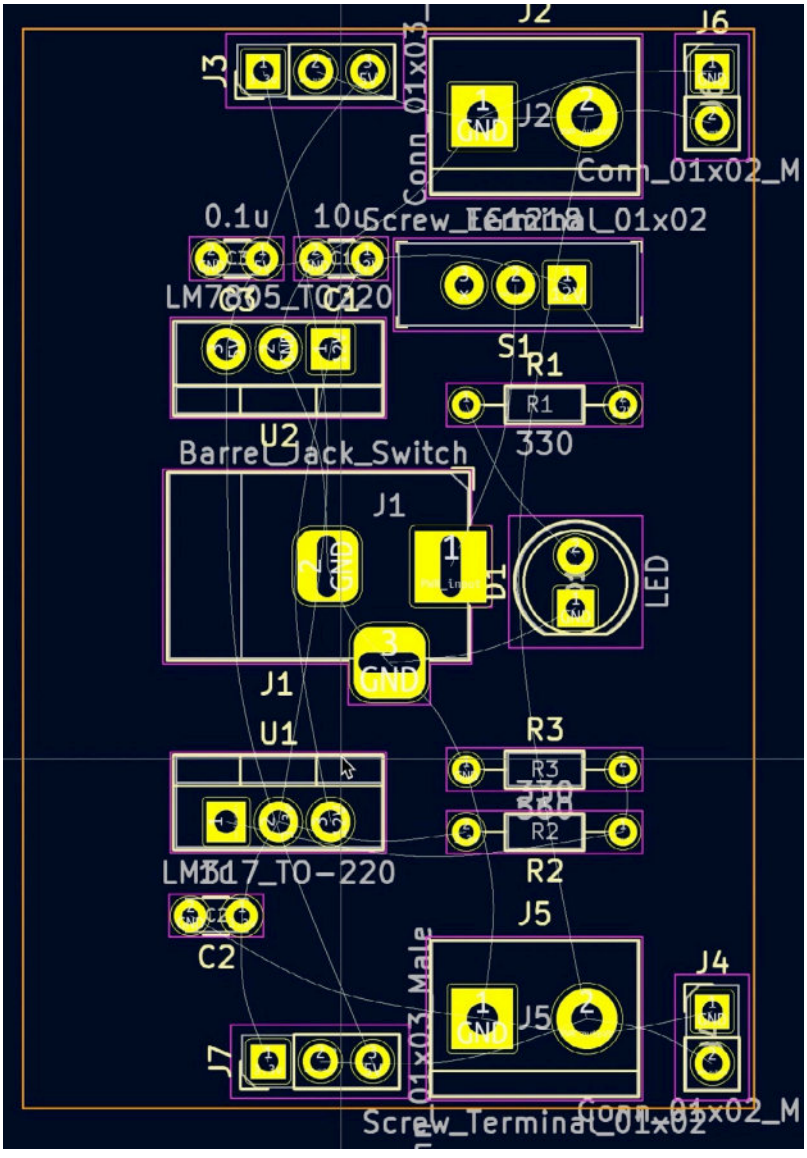


Figure 9.3.3.20: Second iteration of footprints placement.

With this iteration, I allowed enough room on the right side of the board for the edge to move left-wards and expose the breadboard columns underneath it. I also freed up space on the left side, allowing me to reduce the PCB's width, resulting in a lower manufacturing cost.

I am satisfied with the placement of the footprints, as you can see in Figure 9.3.3.20. Before continuing with step four of the workflow, I will go back to step two and refine the Edge outline.Cuts layer.

3.4. 2 - Refine the outline

At this point, the PCB outline in the Edge.Cuts layer has fulfilled its purpose: to help us position the two pin header footprints in their required positions and the rest in their final positions as per the requirements I listed in the introduction and the previous chapter.

In this segment, you will finish work in step two of the workflow and refine this outline. Here is what your work entails:

1. Move the left and right edges of the box inwards. This will reduce the total amount of space of the PCB (and reduce its cost) and expose the two first columns of the breadboard.
2. Round all corners for a modern look.

To achieve these changes, I will use the line and arc tools from the right toolbar. To draw the rounded corners, I will use the arc tool and the technique I demonstrated in the first project in this book (see here for details). To draw the straight segments of the outline, use the line tool.

Because I drew the rough outline using the rectangle tool, the only way to do what I described above is to delete the rectangle, and use the arc and line tools to draw a new outline. With the footprints in their final and fixed positions, this is relatively easy. This is how I planned to achieve this:

1. Switch to the Edge.Cuts layer.
2. Change the grid size to 0.254 mm.
3. Turn on the grid to help you position the mouse and close the joints between lines and arcs.
4. Delete the original rectangle.
5. Select the line tool.
6. Draw a closed polygon that contains all footprints with minimal space in the perimeter. Start drawing from the top right corner, and continue leftwards.
7. Use the arc tool to replace the 90-degree corners with rounded corners.

Below, I have documented the drawing of the polygon with a series of figures. I use numbers to annotate the corners and arrows to mark the cursor position where needed:

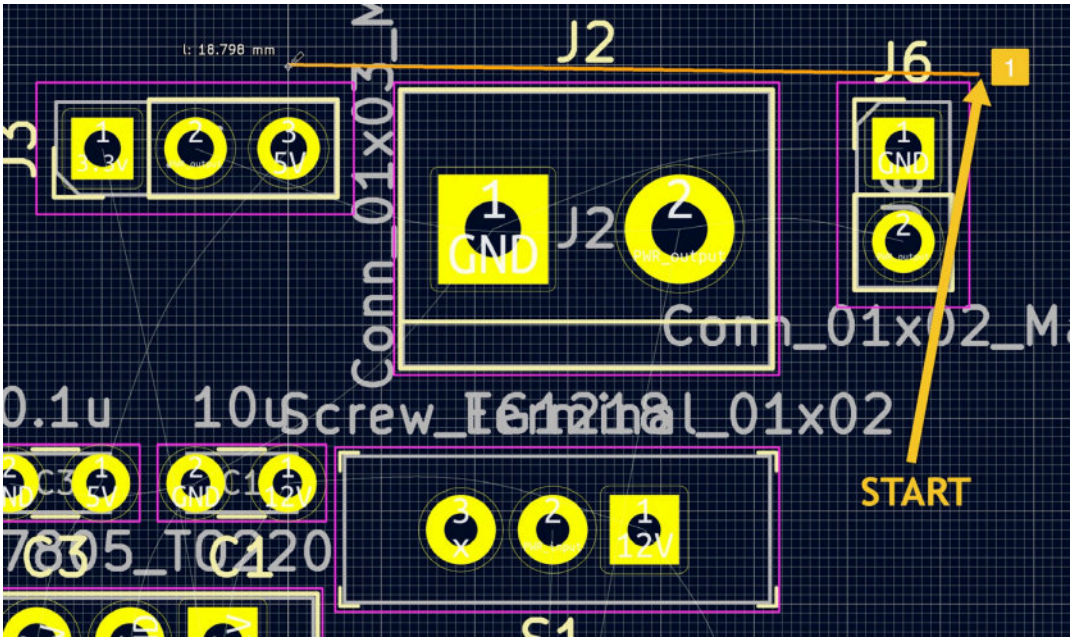


Figure 9.3.4.21: Polygon start corners “1”.

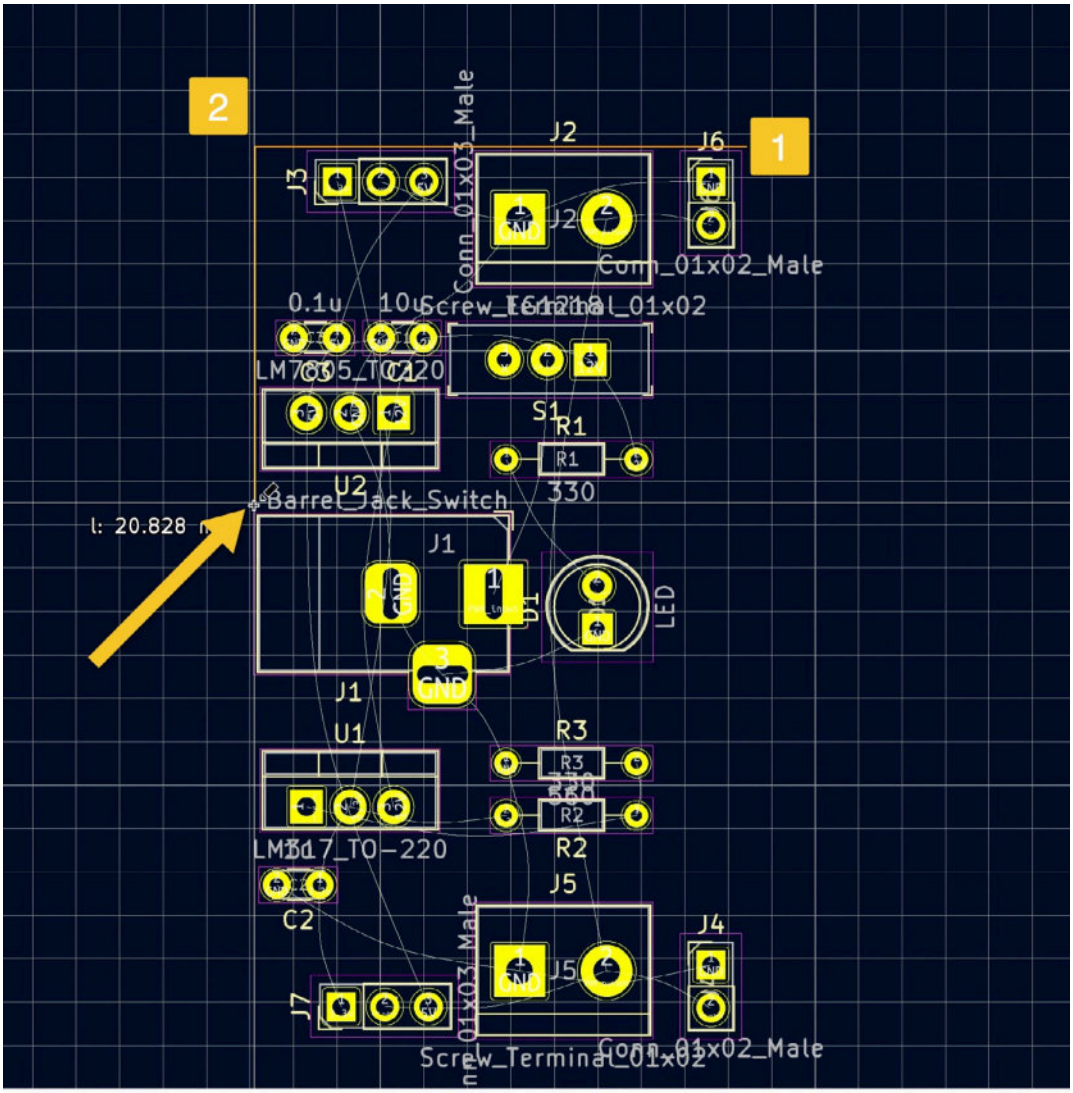


Figure 9.3.4.22: Polygon corners “1” and “2”.

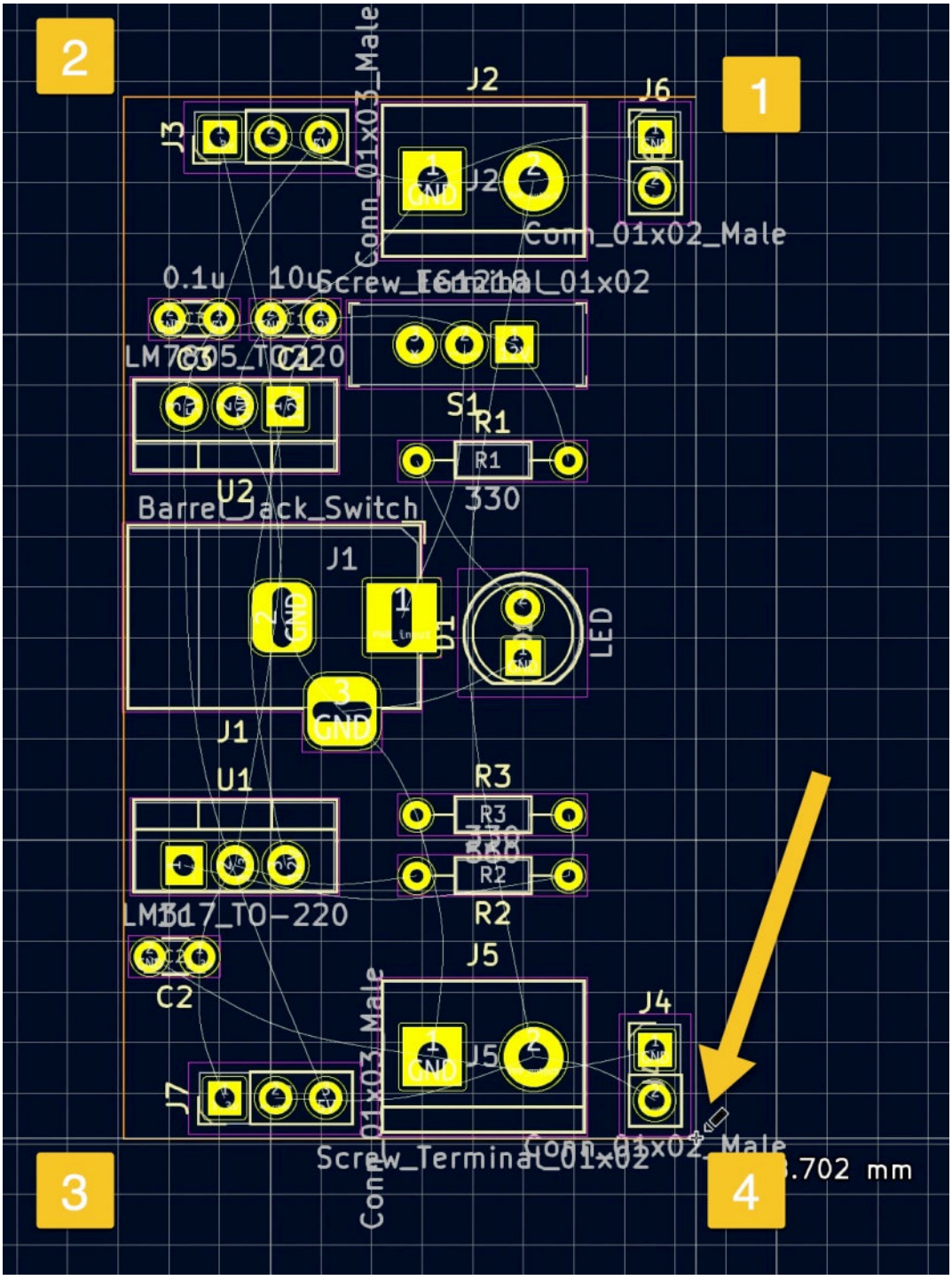


Figure 9.3.4.23: Polygon corners “1”, “2”, “3”, and “4”.

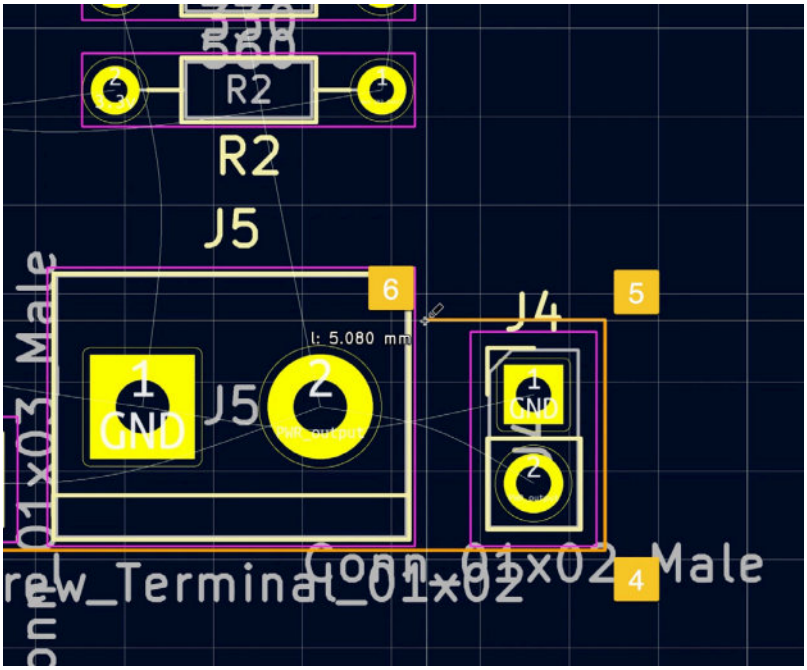


Figure 9.3.4.24: Polygon corners “4”, “5”, and “6”.

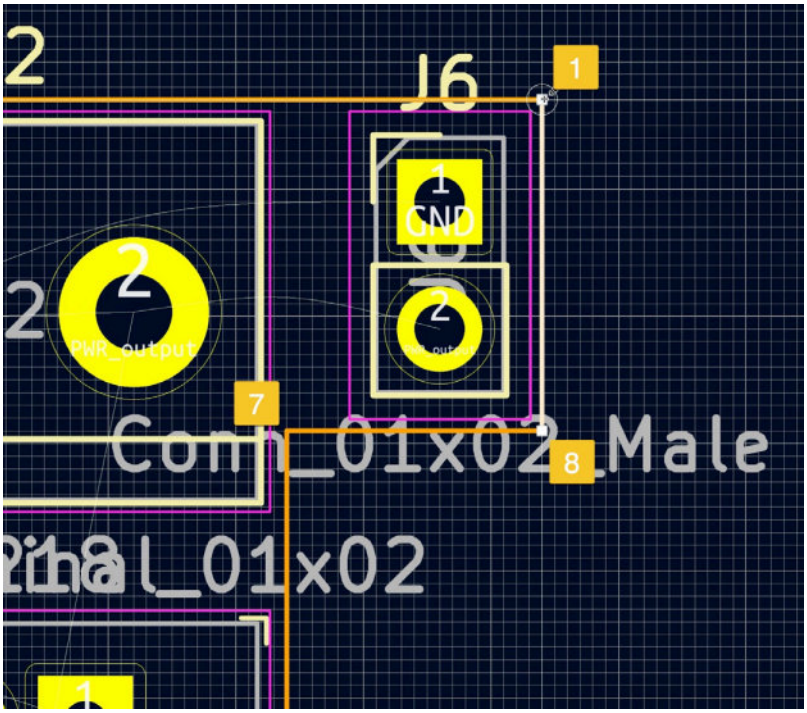


Figure 9.3.4.25: Polygon corners “7”, “8”, and closing back at “1”.

The completed polygon looks like this:

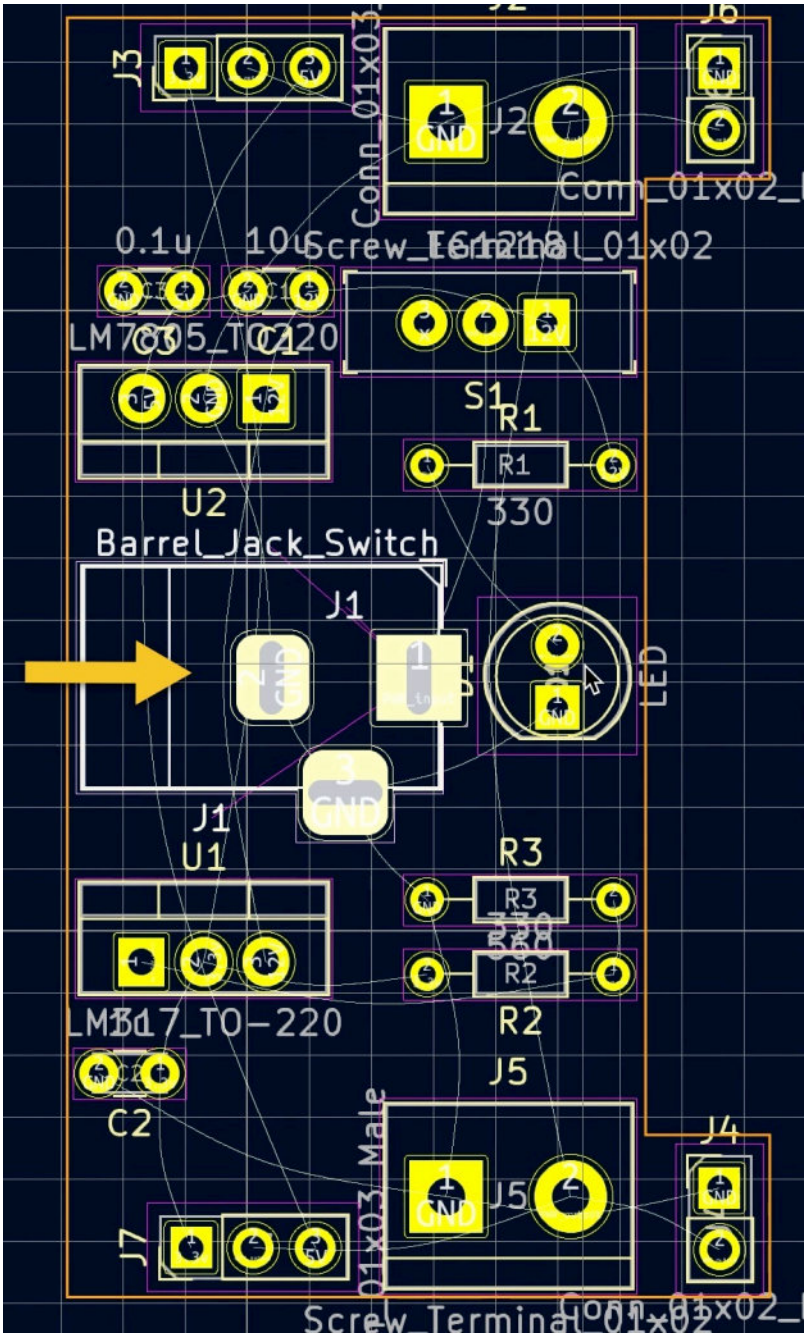


Figure 9.3.4.26: The completed polygon.

After completing the polygon, I also moved the barrel jack connector J1 footprint slightly to the right because the edge cuts line touched the footprint's courtyard. With this complete, the next task is to replace the 90-degree corners with rounded corners. You did this in the first project in this book, so please refer to that if you need a refresher.

Below I have documented the process. For the arc diameter, I have used a radius of 1.016 mm. Here is the first arc:

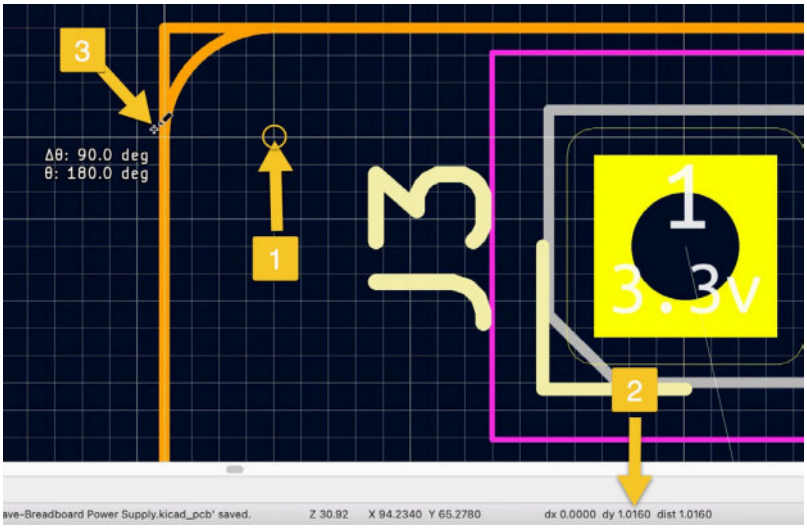


Figure 9.3.4.27: The first arc.

In the figure above, I am drawing the arc with a dx 1.016 mm from the corner. Arrow “1” points to where I click to start drawing and press the space bar to reset the counters. Keep an eye on the dy and dy value “2”. They show the distance between the reset position and the cursor. When my cursor reached the vertical line at “3”, the dy value was 1.016 mm. Click again to close the arc.

To finish work in the corner, resize the horizontal and vertical lines and connect them to the ends of the 90-degree arc:

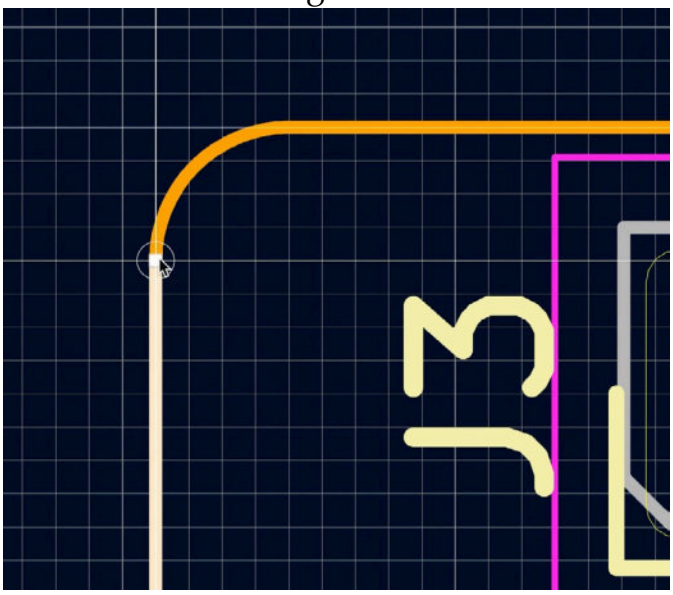


Figure 9.3.4.28: The firs arc is complete.

Continue to replace all corners. This is the final result:

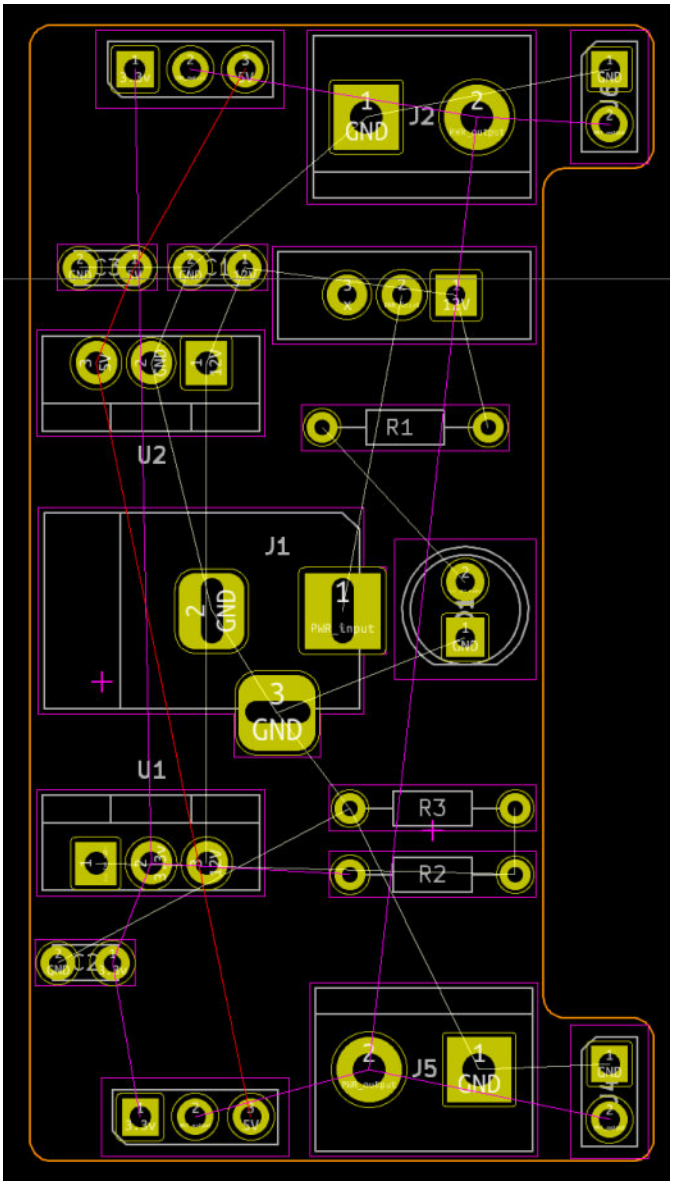


Figure 9.3.4.29: The final PCB outline.

Step two of the process is now complete. Let's continue with the routing in step four.

3.5. 4 - Route

In this segment, you will complete step four of the layout workflow and route the board. Here is the plan:

1. I will use the top and bottom copper layers.
2. I will use the bottom copper layer for GND routes and the top for all other routes.
3. I will use a copper fill in the bottom layer and connect the fill to the GND net. For this reason, I will not do any manual routing between

pads that belong to the GND net. I will create the copper fill in the next segment in this chapter (step four of the workflow).

4. I will allow the layout editor to automatically set the width of the track based on the net class settings I set in the setup step.
5. Where necessary, I will use vias to switch a copper route between the top and bottom layers.

I will demonstrate how I draw a couple of routes and leave you to do the rest. From the right toolbar, select the single route tool. Click on pad 3 of J3 and start drawing a route towards pad 1 of C3. Use the ratsnest lines to help your navigation. The interactive router will highlight your targets. I have set my interactive router to use the "Walk around" mode not to make changes to existing routes or un-locked footprints and vias. The figure below shows the start of the route drawing:

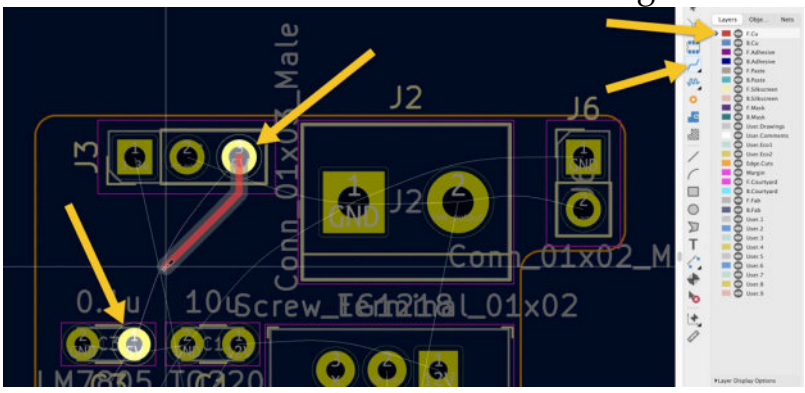


Figure 9.3.5.30: Drawing the first route.

This track belongs to the "5V" net and has inherited the track width from the power output net class (0.35 mm). Here is the completed route:

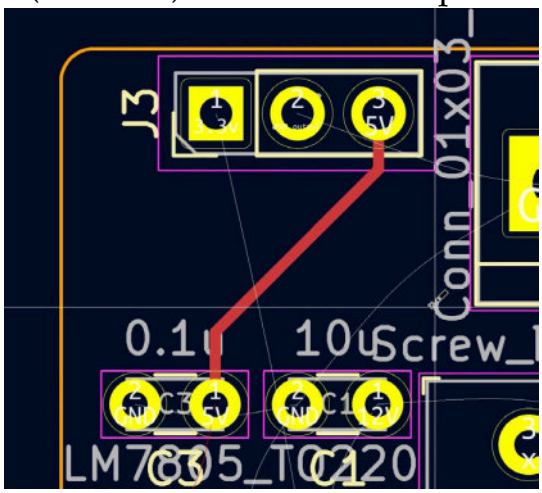


Figure 9.3.5.31: Completed the first route.

Continue on your own and draw the remaining copper tracks. As you can see below, I had to use vias to switch routes to the bottom copper layer.

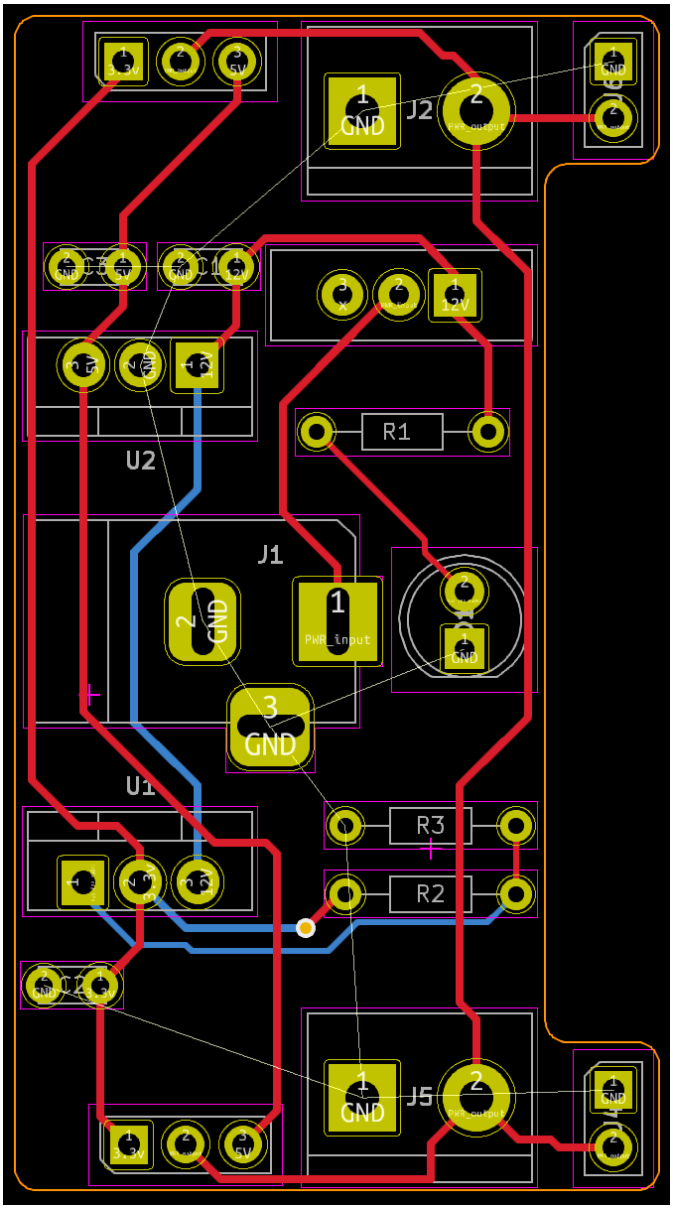


Figure 9.3.5.32: Completed routing except for the GND pads.

You can use a via to switch the drawing of a copper tracks between layers. KiCad has the “V” hotkey which allows you to insert a via during the drawing of a copper track and automatically switch between layers. In this project you are working on a two-layer PCB so each time you type “V” during the drawing of a track, drawing will switch to the alternate layer.

Let’s look at a quick example to practice using vias during the drawing of a track. In the figure below (left), I have selected the front copper layer (“1”), and started drawing a copper track (“2”).

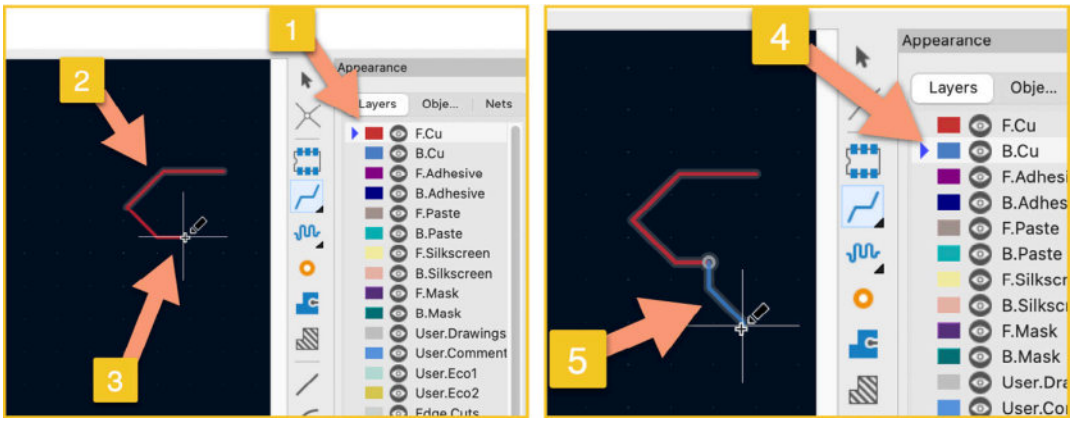


Figure 9.3.5.33: Demonstration of using a via to switch copper layers.

When I type "V" to insert a via ("3", left), Pcbnew automatically changes the active copper layer to B.Cu ("4", right), and then I can continue the drawing of the copper track in the bottom copper layer ("5"). Take a few minutes to practice this technique, and then apply it to the project layout.

In the figure above (Figure 9.3.5.32), I have routed all pads that don't belong to the GND net. I left those for the next step because I plan to use a copper fill connected to the GND net. This copper fill will complete all electrical connections between the GND pads without drawing copper tracks.

3.6. 5 - Copper fills

At this point, the PCB is routed except for the GND pads. In this segment, I will create a copper fill in the bottom copper layer and connect it to the GND net. This will result in a fully routed board. To learn how to create a copper fill, please refer to the relevant chapter in Part 8 of this book.

Select the B.Cu layer, GND net, and Hatch pattern for the fill type in the Copper Zone properties window. Select the B.Cu layer from the Layers tab, and click on the copper fills button from the right toolbar. Click at the top right corner of the PCB to start drawing the zone.

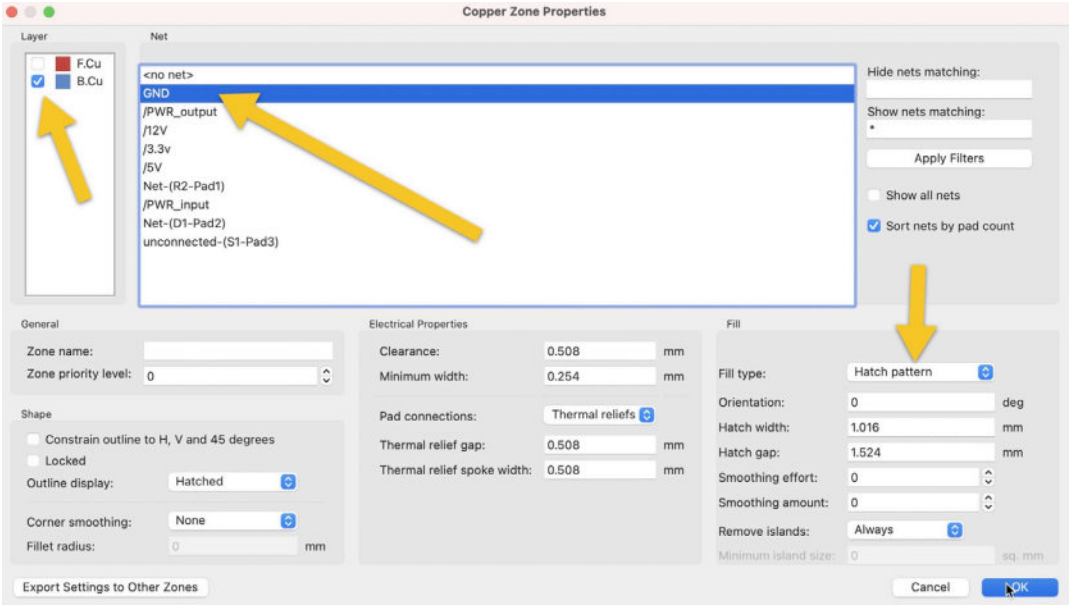


Figure 9.3.6.34: Copper Zone properties.

Start drawing the zone as close as you can along the perimeter of the PCB. When you finish drawing, right-click on the zone outline and click on Fill from the Zones submenu. The board should now look like this:

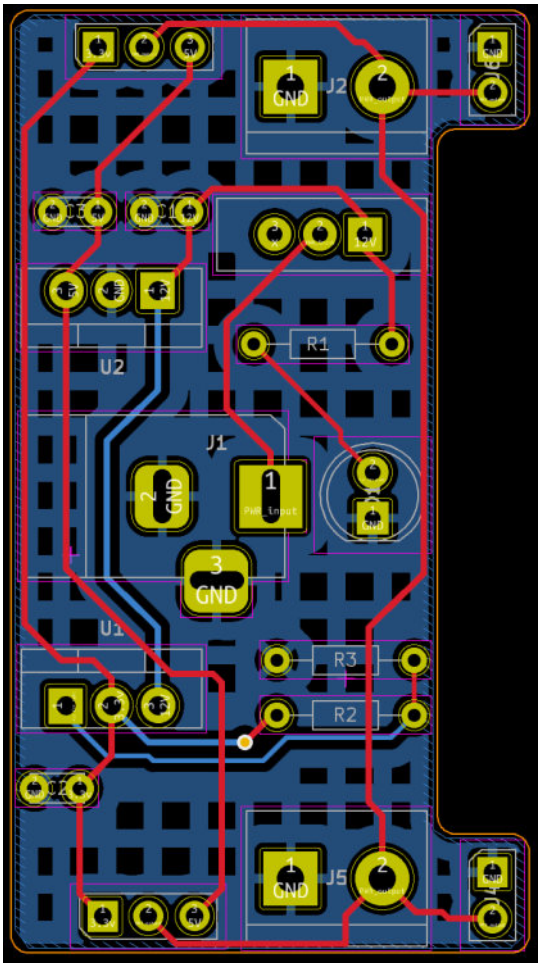


Figure 9.3.6.35: The PCB with the hatched patter copper fill in the back copper layer.

Before continuing to step six, run a DRC to ensure there are no unconnected pads. You can ignore other violations for the time being, such as issues with footprint silkscreen. The result of my DRC returned no unconnected items. Let's continue with step six of the workflow.

3.7. 6 - Silkscreen

In this segment, you will add text and graphics in the front and back silkscreen layers. Remember that your board already has content in the silkscreen as inherited by the footprints it contains. The additional silkscreen items I have added to my board are these:

- Front Silkscreen:
    - Text labels for the power on/off switch.
    - Text labels for the voltage selector switches.
    - Text labels to help me correctly assemble the board, such as "+ " and "-" for the LED and values for the resistors and capacitors.
- Back Silkscreen:
    - A version number for the board.
    - The name of the board.
    - A Tech Explorations logo.
    - A KiCad logo.

To reduce clutter in the editor, enable the outline mode for the copper fill (you will find the button in the left toolbar). Enable the F.Silkscreen layer. This is the starting point:

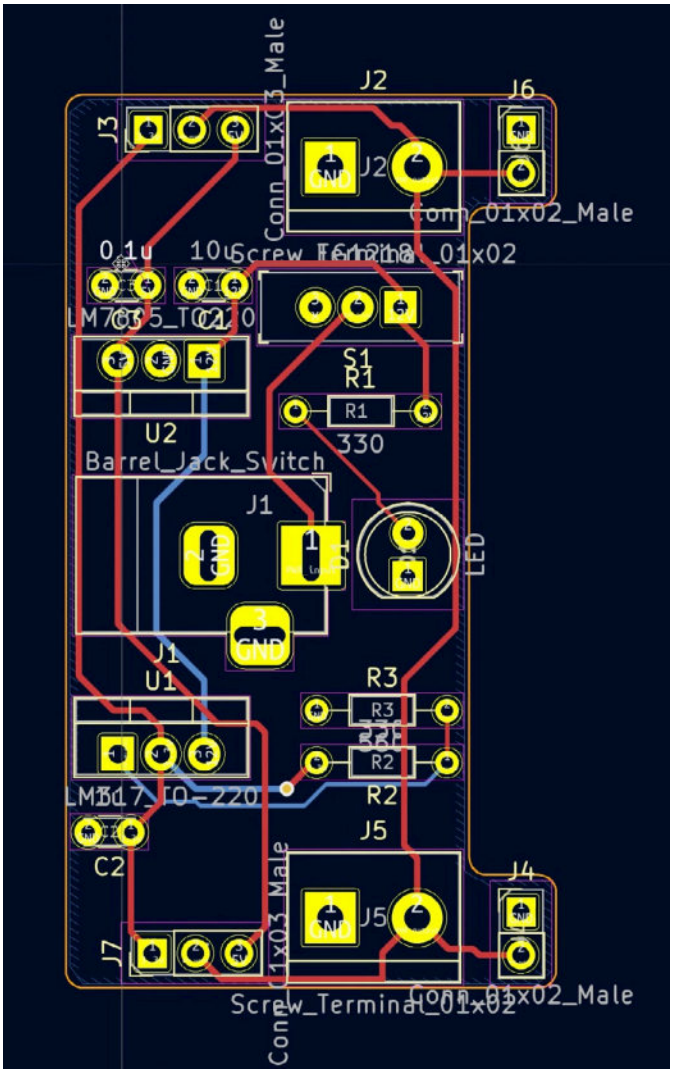


Figure 9.3.7.36: Starting point prior to adding Silkscreen text and graphics.

For the resistors and capacitors, remember that you added their values in the footprint properties in step two of the schematic design process. To show these values in the silkscreen, you will need to open the properties window for the value fields and change their layer to F.Silkscreen.

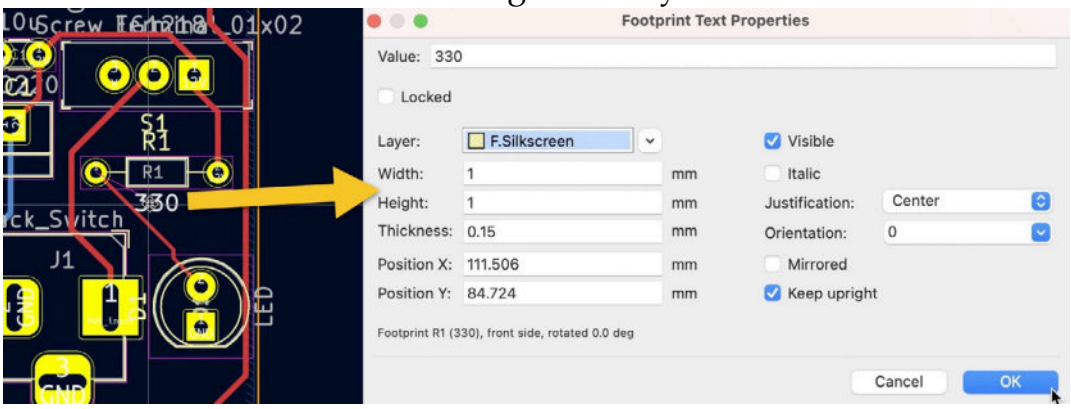


Figure 9.3.7.37: Change the layer of the value label to F.Silkscreen.

As you are changing the display layer for the capacitor and resistor values, take the opportunity to reposition them appropriately so that they don't overlap with other elements of the PCB.

Repeat the same process for any text label that you want to include in the front silkscreen.

To better use the available space, I have also reduced the size of the text labels in the silkscreen. You can do this in bulk, using the Edit Text and Graphics Properties window. You can learn more about this tool in a dedicated chapter.

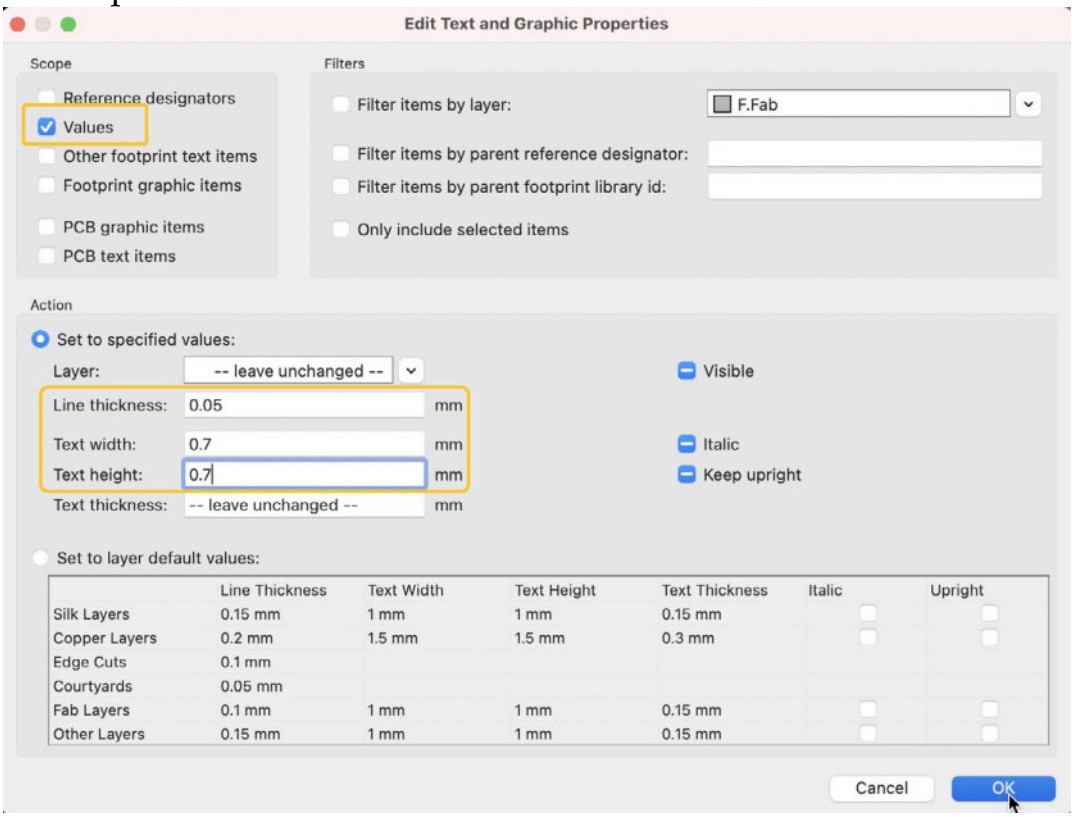


Figure 9.3.7.38: Changing text features in bulk.

Various text items are visible in the editor but are not necessary. To reduce clutter, I choose to make them invisible. For example, the name of the footprints for the voltage selector switches is rather long: "Conn_01x03_Male". You can turn off their visibility via their properties window:

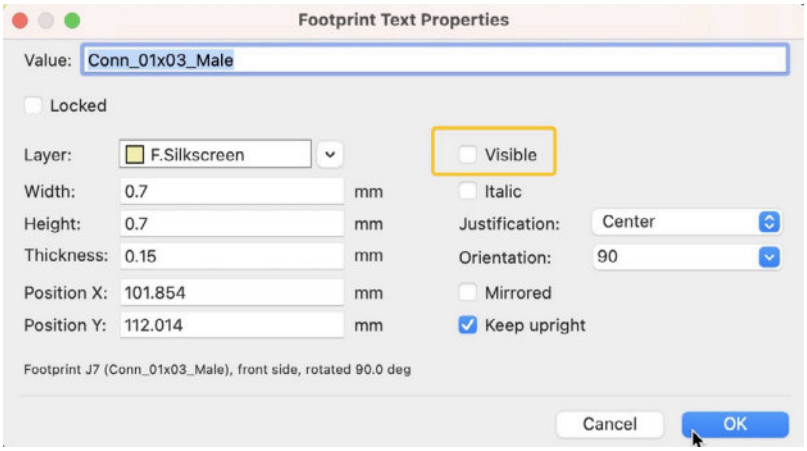


Figure 9.3.7.39: Make invisible.

Below you can see the completed work in the front silkscreen:

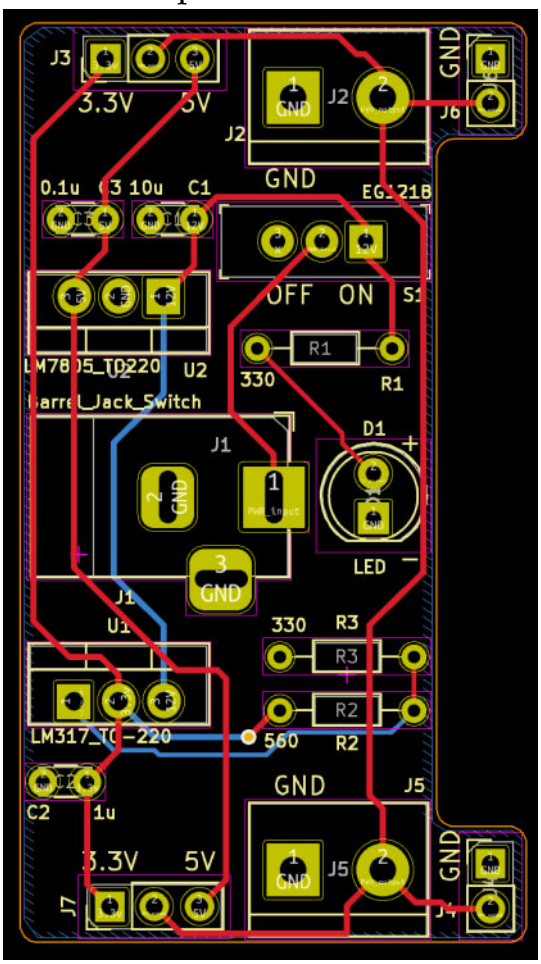


Figure 9.3.7.40: Completed front silkscreen.

The text in the front silkscreen uses bright yellow.

Continue with the back silkscreen layer next. The items to place include:

1. A version number with a circle around it.
2. The name of the board.
3. Two graphic logos.

4. You can find the KiCad logo in the KiCad footprint libraries. I have created the Tech Exploration logo as a silkscreen footprint using the process I describe in a dedicated chapter in the Recipes part of the book.

When you add text in the back silkscreen, remember to enable the "Mirrored" option in the Text properties window:

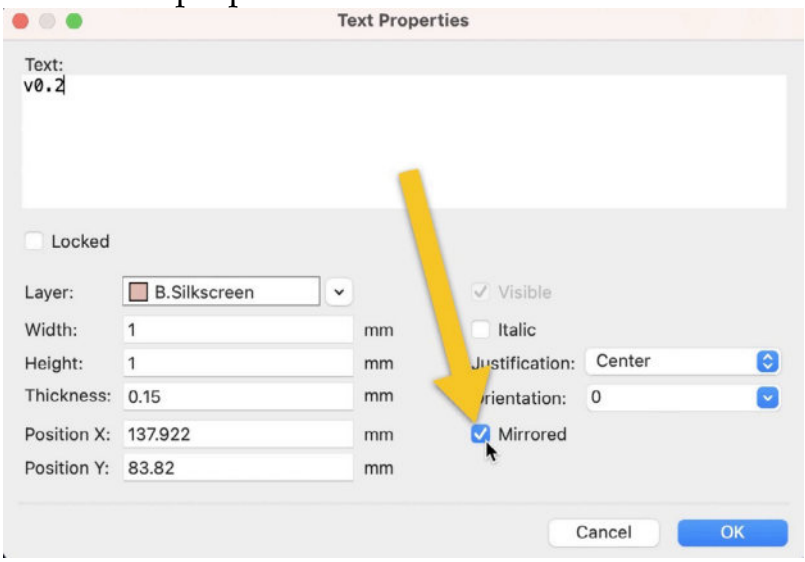


Figure 9.3.7.41: Text in the back silkscreen should be Mirrored.

Similarly, to add a footprint to the back layer (silkscreen or copper), change its position to "Back" in its properties window:

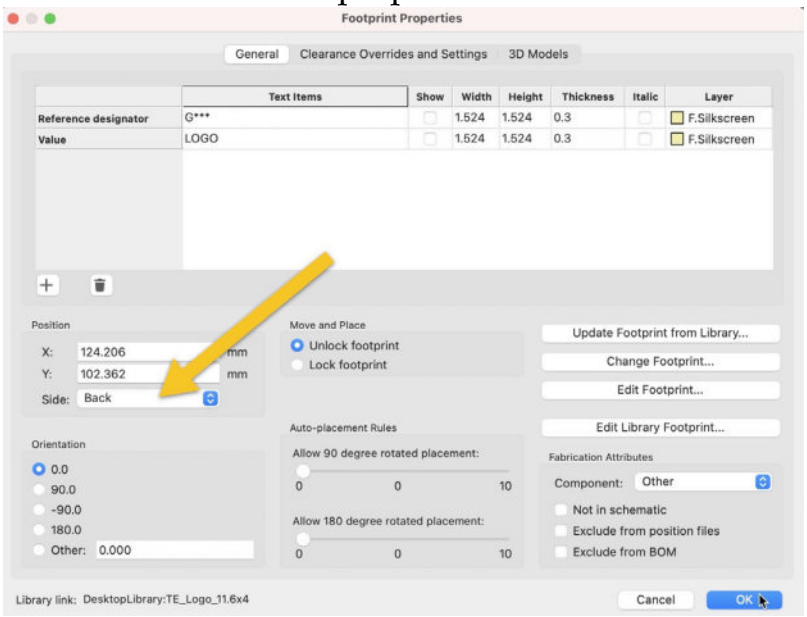


Figure 9.3.7.42: This footprint goes to the back layer.

Below you can see my version of the back silkscreen. I have disabled the front silkscreen so that there is no overlap:

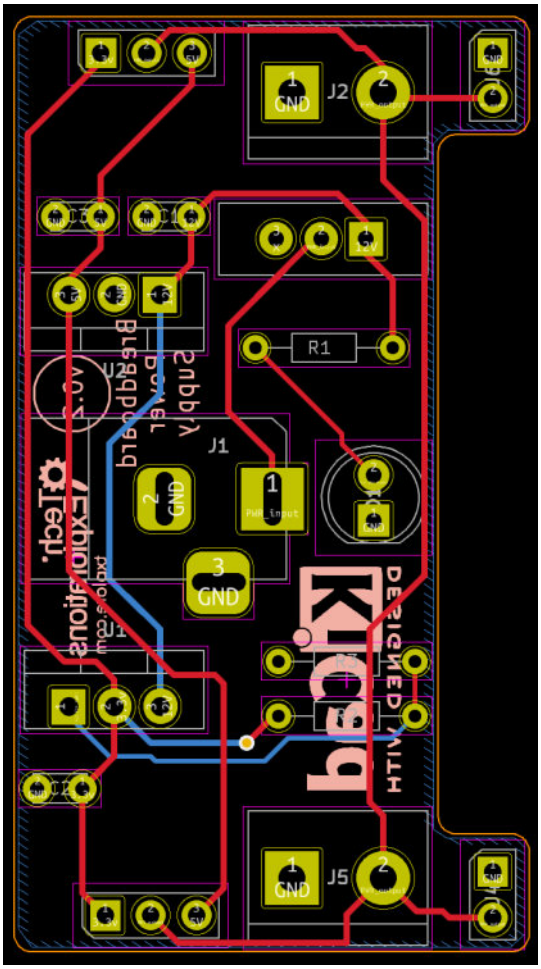


Figure 9.3.7.43: The back silkscreen.

You can also use the 3D viewer to see a rendering of the board with the silkscreen layers included.

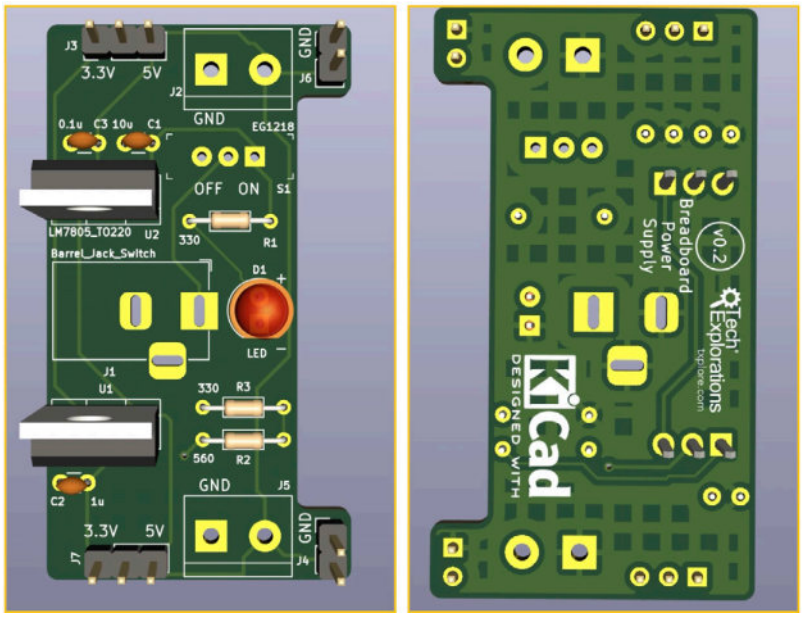


Figure 9.3.7.44: 3D rendering of the board.

This completes step six of the layout workflow. Let's do a quick DRC in the next step before we complete the project.

3.8. 7 - Design Rules Check

Before exporting the Gerber files in the next step, let's do a final DRC. I tried to be careful through the layout editing. I don't expect any unconnected pins, apart from the mistake with the orientation in J5, which I fixed.

My DRC results are below:

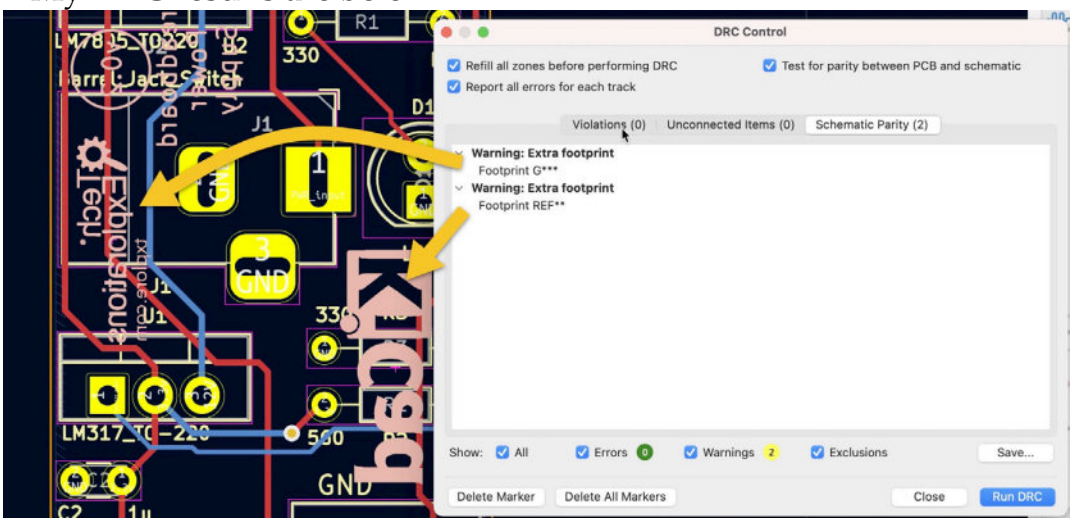


Figure 9.3.8.45: The are two violations to evaluate.

The two violations listed have to do with the two logo graphics on the back of the board. These logos are implemented as footprints. KiCad considers them equivalent to a resistor. The difference between a resistor footprint and the KiCad logo is that the KiCad logo only has a silkscreen. No pads, no courtyard, no edge cuts, etc.

The DRC expects that every footprint must be associated with a schematic symbol. If it finds a footprint without a symbol, it will give an "Extra footprint" warning. As long as you understand the origin of this message, you can choose to ignore it and continue.

If you want to complete the DRC with a perfect zero violation score, you can return to the schematic editor, add a simple symbol, such as a pin, and associate it with the logo graphic. KiCad will accept the association, and the DRC will be satisfied.

Having explained the origin of extra footprint warnings, I declare this board fit for manufacturing.

8 - Manufacturing postponed

Thanks to the

3.9. Export and Manufacture

If you plan to manufacture this project's PCB, you should first read the next chapter, titled "4. Finding and correcting a design defect". In that chapter, I show how to fix a bug that was reported by a reader of the beta version of this book. Once you fix this bug, you can follow the instructions in this chapter to learn how to order the physical PCB.

Let's manufacture this board. Start by exporting the Gerber files, including the drill files. Test your Gerbers using KiCad's Gerber Look app and at least one online Gerber viewer. You can follow the process I detail in the relevant chapter.

Below you can see the Gerber files for my instance of the project:

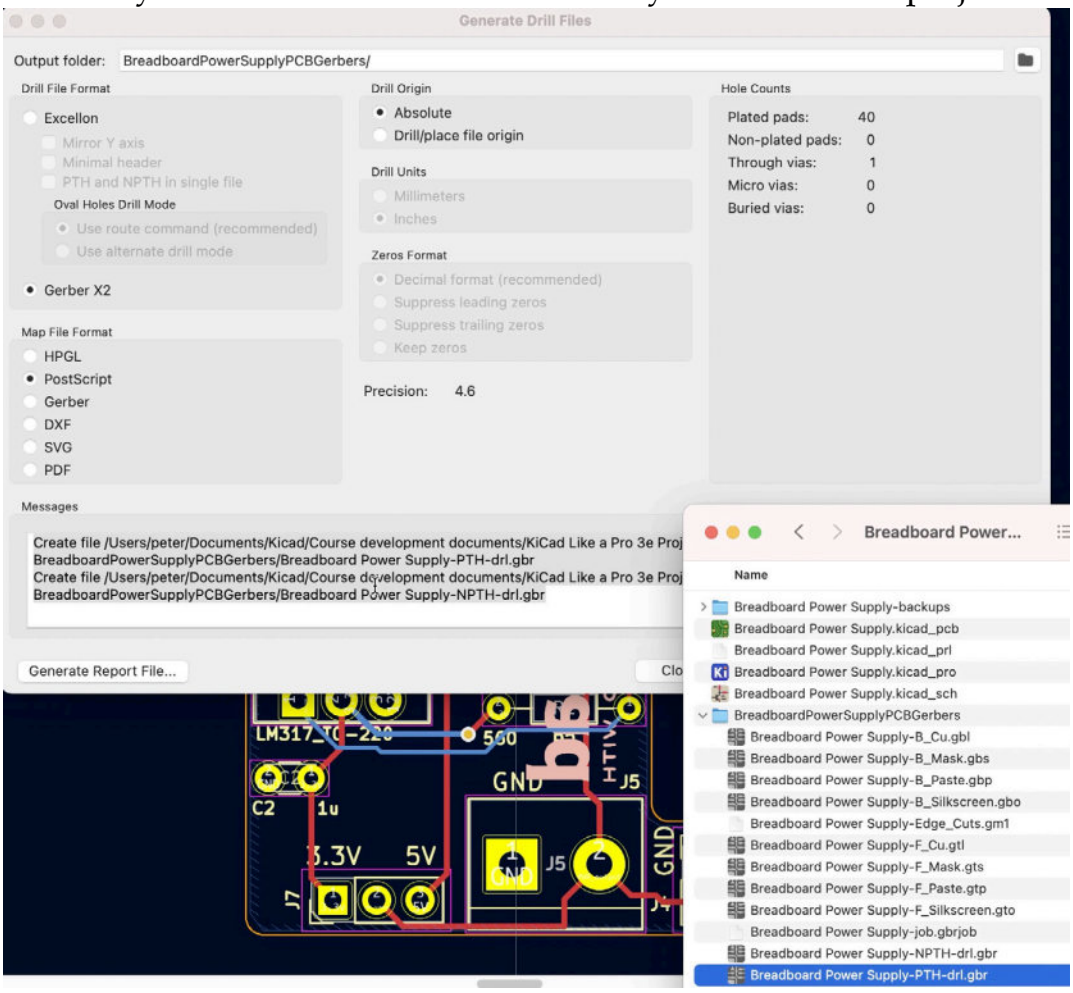


Figure 9.3.9.46: Project Gerber files.

Then, I did some thorough testing using KiCad's Gerber Viewer:

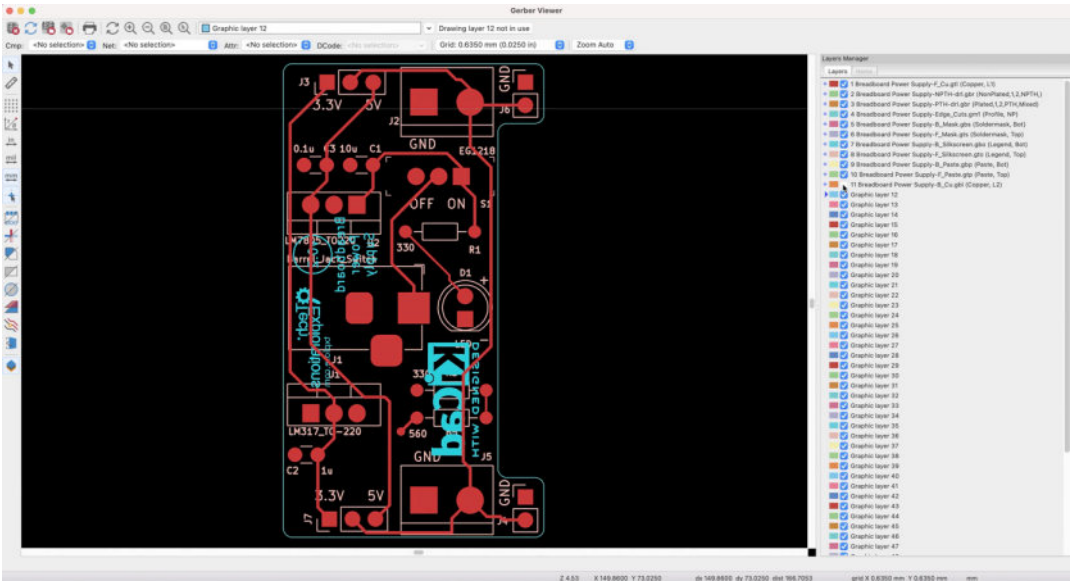


Figure 9.3.9.47: Testing in Gerber Viewer.

Because of the cost and time involved in having this board manufactured, doing more testing is justified. Below I am using www.gerber-viewer.com/Viewer for another opinion on the fitness of this board:

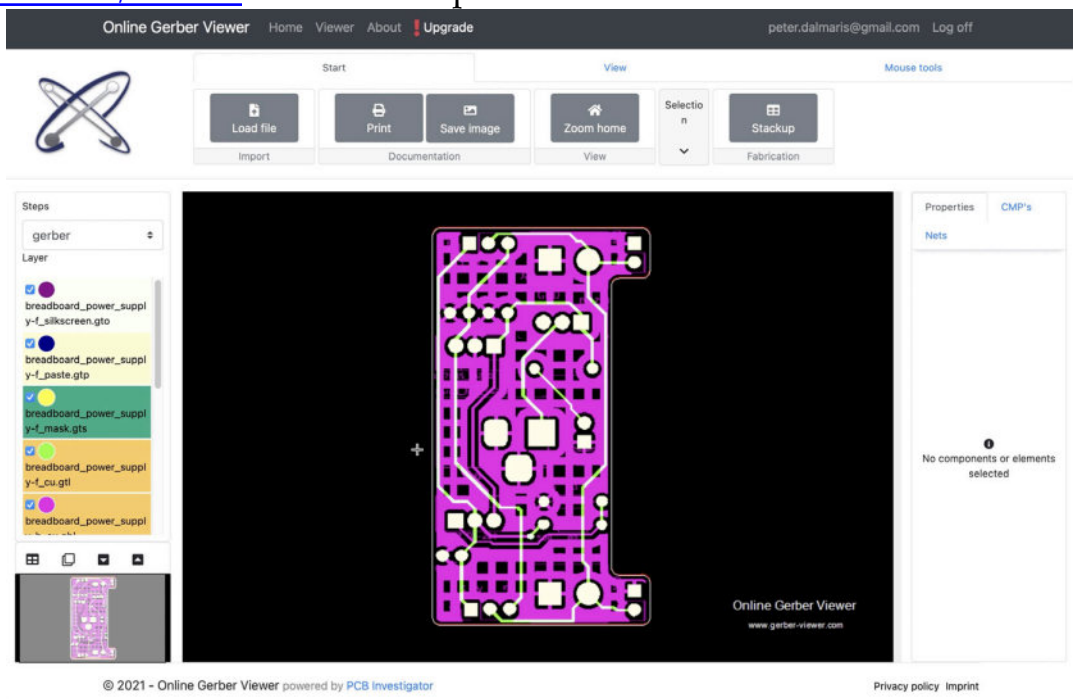


Figure 9.3.9.48: Testing in Online Gerber Viewer.

Most online manufacturers require the customer to enter the dimensions of the PCB. I'll use Pcbnew's measurement tools to do this. My board is 52.324 mm x 28.702 mm.

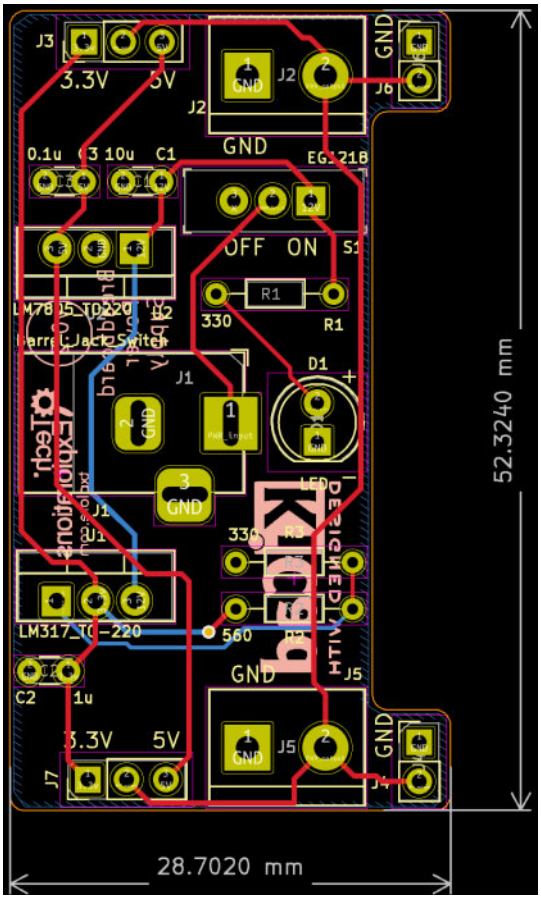


Figure 9.3.9.49: The dimensions of this PCB.

I will send the Gerber files to NextPCB for manufacturing. NextPCB can extract board dimensions from the Gerber files and pre-populate the relevant fields in the form. NextPCB also has a Gerber viewer. I use this tool to see what my board will look like from the manufacturer's perspective.

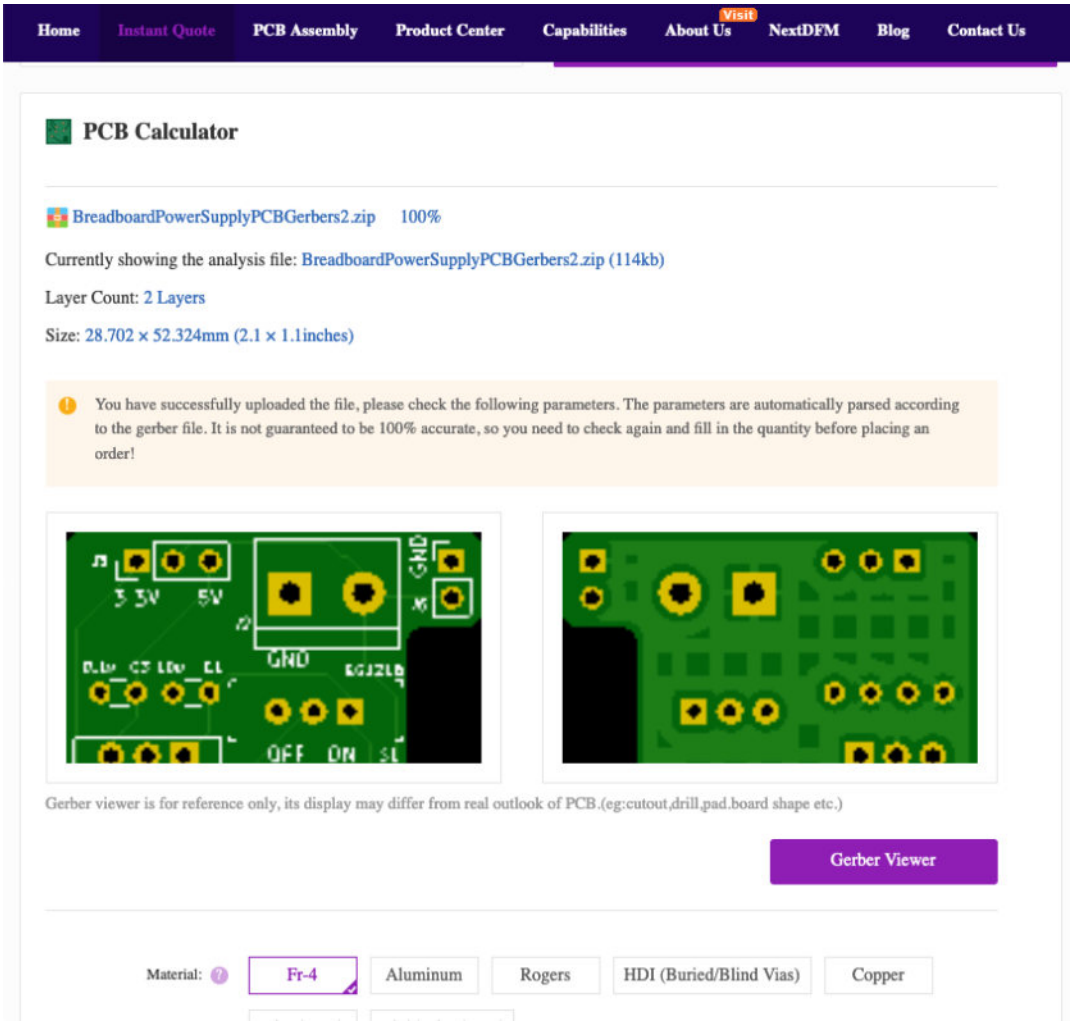


Figure 9.3.9.50: Project Gerber ZIP file uploaded to NextPCB.

Click on the purple Gerber Viewer button. This tool will render the information found in the Gerber files. You can enable/disable each layer and inspect. If everything looks good, continue with the order.

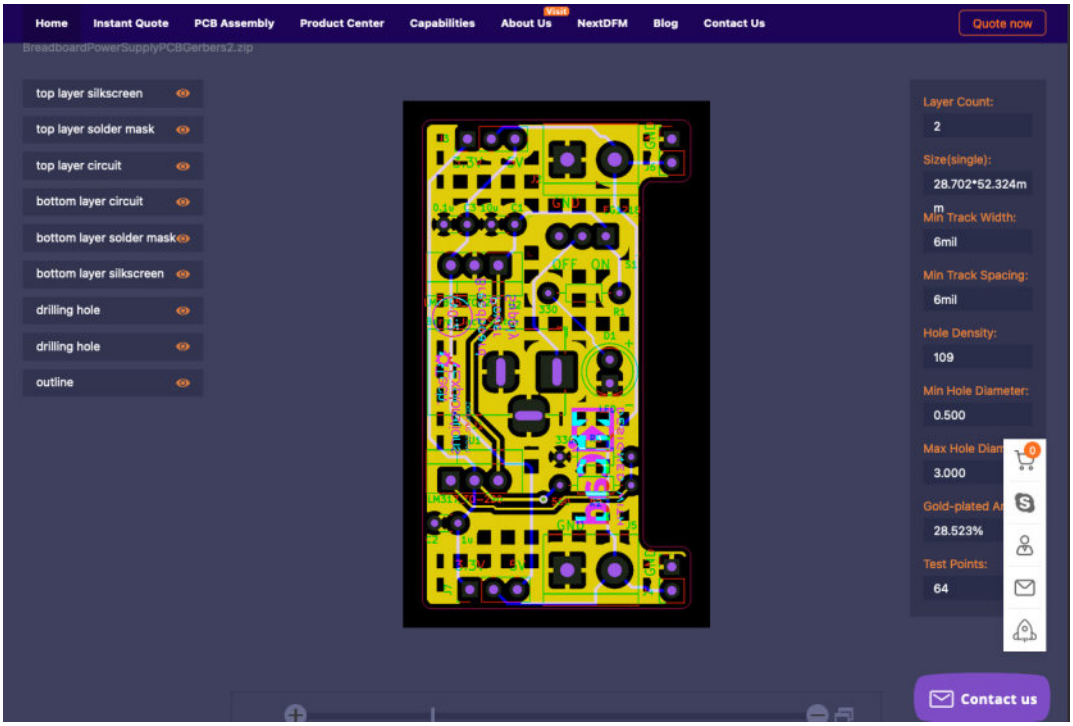


Figure 9.3.9.51: Nextpcb online Gerber viewer.

I'm confident my PCB is fit for production, so that I will continue with my order. Here are the order details with comments:

- Material: FR4.
- Layer count: 2 Layers.
- Board type: Single piece.
- Size: 28.7 x 52.3 mm (automatically entered).
- Break-away rail: N/A.
- Quantity: 5.
- PCB thickness: 1.6 mm (default).
- Solder Mask Color: Green (default - I usually select Red, but there is a big price spike, so I'll go for green).
- Silkscreen: White (default).
- Finished copper weight: 1oz (default).
- Min trace/space outer: 6/6mil (default).
- Min drilled hole: 0.3mm (default).
- Via process: Tenting vias (default).
- Surface Finish: HASL (default).
- Beveling of G/F: No (default).
- Electrical Test: Sample Test Free (default).
- Approve Working Gerber: Don't need (default).
- Plated Half-holes: No (default).
- Impedance: No (default).

- Microsection Alanysis Report: (default).

It is worth knowing that if you choose any non-standard option, the price of the board increases significantly. For example, the difference between the green solder mask and the red is around US$33. If you go for lead-free HASL, the difference is US$15.

Several weeks later, I received the manufactured PCB:

[TBA]

4. Finding and correcting a design defect

As part of the beta program of this book, reader Wayne found a bug in the schematic of this project. This bug required corrections in both the schematic and the layout. Instead of re-writing the chapters in this part of the book, I introduced this chapter. I preferred not to obscure the reality that errors are a natural part of the engineering that leads to fixes and improvements.

In this chapter, I will document the bug that Wayne found and how I iterated through the design of the project's PCB to fix it.

The primary defect

I introduced this defect in step five of the schematic workflow. In that step, I created the net label "PWR_output", and attached one to the net that connects pins 2 of J7, J6 and J5, and pins 2 of J3, J4, J2. If this power supply outputted a single voltage, this would have been not a problem. However, wanting to upgrade my breadboard power supply from the second edition of this book, I decided to use a second voltage regulator so that a single power supply could output 3.3V and 5V across its two output headers.

Below, I use arrows to highlight the location of the offending net labels.

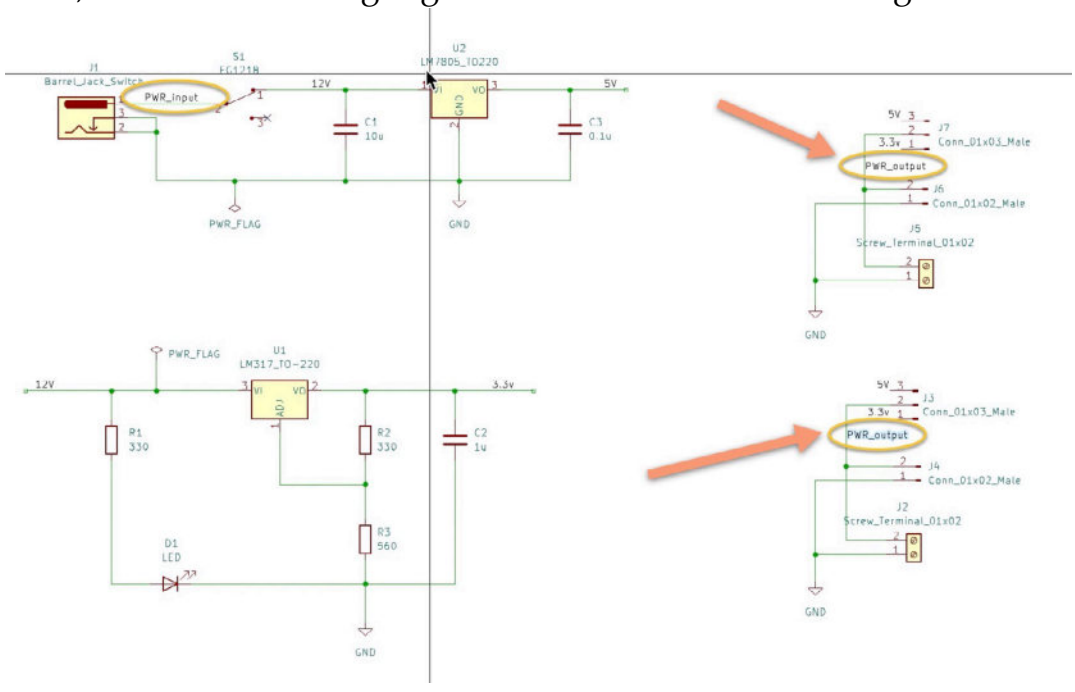


Figure 9.4.Subsection.1: The location of the primary defect.

If you set both voltage selector switches to the same voltage, there is no problem again. However, if you set one selector (say, J7) to 3.3V and the other (J3) to 5V, then a short circuit between the output pins of the voltage regulators will occur (pin 3 of U2 and pin 2 of U1). This short circuit will destroy the regulators and likely the components on the board. I have designed this power supply to draw power from a regular 12V wall power supply. Typically, these power supplies don't have a fuse or other protective circuitry.

Secondary defect

I took this opportunity to correct a second defect that Wayne also found. I introduced the defect in step three of the layout design workflow. There, I incorrectly placed J6 next to J2 (instead of J5) and J4 next to J5 (instead of J2).

I use the double-ended arrow below to show the offending footprints:

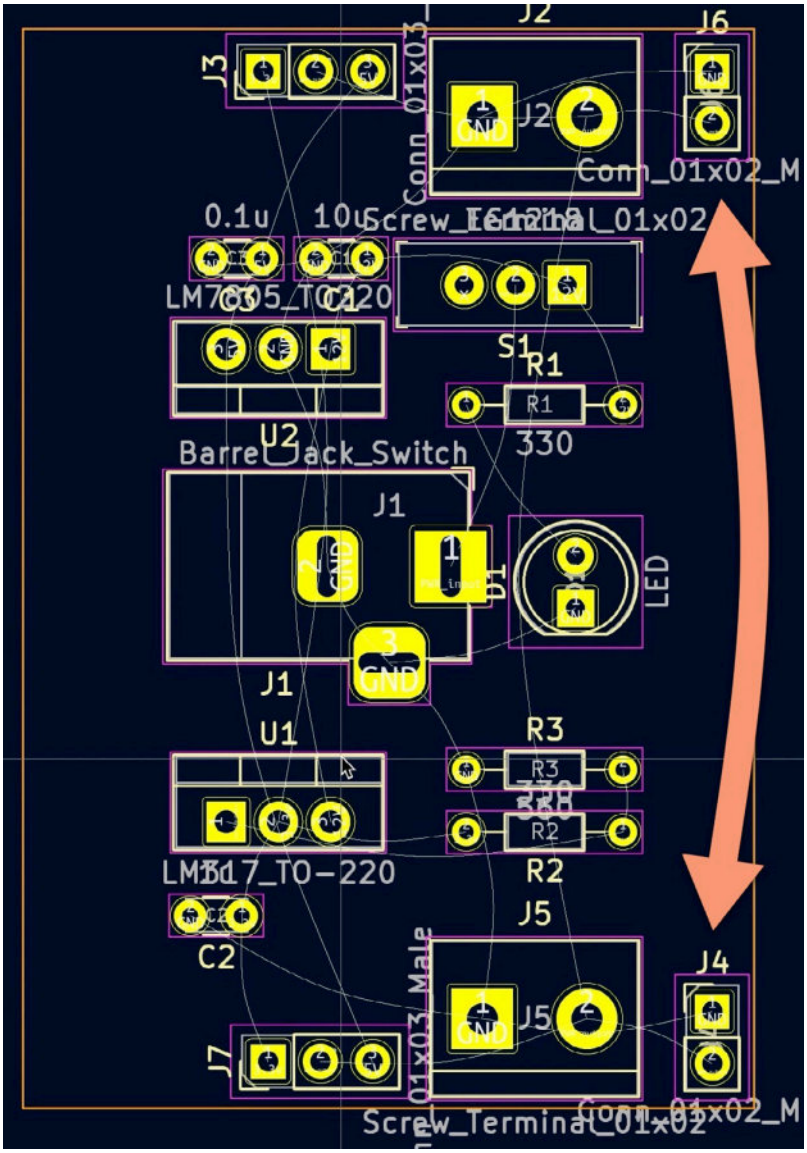


Figure 9.4.Subsection.2: Footprints J6 and J4 are incorrectly placed.

The fixes

I will delete the offending net label “PWR_output” and replace it with two new net labels to fix the primary defect. This will require re-drawing the relevant copper traces that connect pins 2 of J7, J6, J5, and pins 2 of J3, J4, J2.

To fix the secondary defect, I will interchange footprints J4 and J6. The main challenge in doing this is to ensure that the spacing between the outer pins of these footprints remains equal to the original. As you may remember, these footprints consist of the geometrical constraint for this PCB; the distance between the outer pins of J4 and J6 must be equal to the distance of the outer pins of the mini-breadboard.

4.1. Fix the schematic

I decided to make a copy of my project KiCad directory to preserve the original defects. You may choose to do the same so that you have a fall-back if your repair doesn't go well.

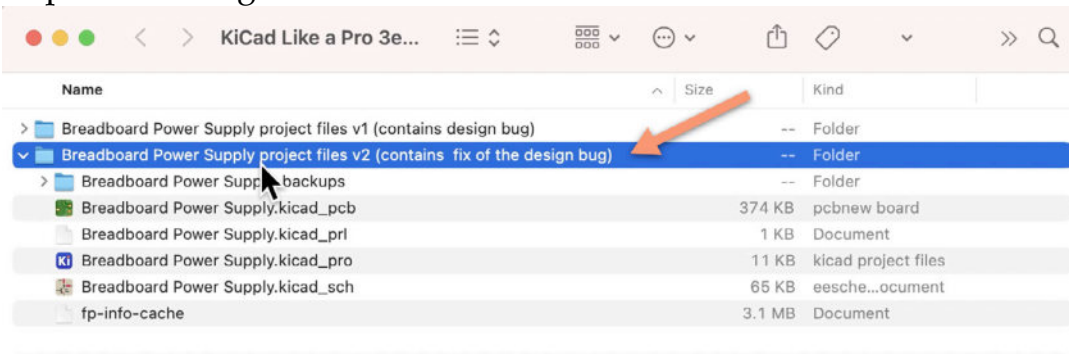


Figure 9.4.1.3: I have duplicated the original project directory.

Open the project in KiCad, and then open Eeschema. To fix the first design defect, delete the original "PWR_output" net labels, and replace them with two new labels: "PWR_OUT_TOP" and "PWR_OUT_BOT" (see below).

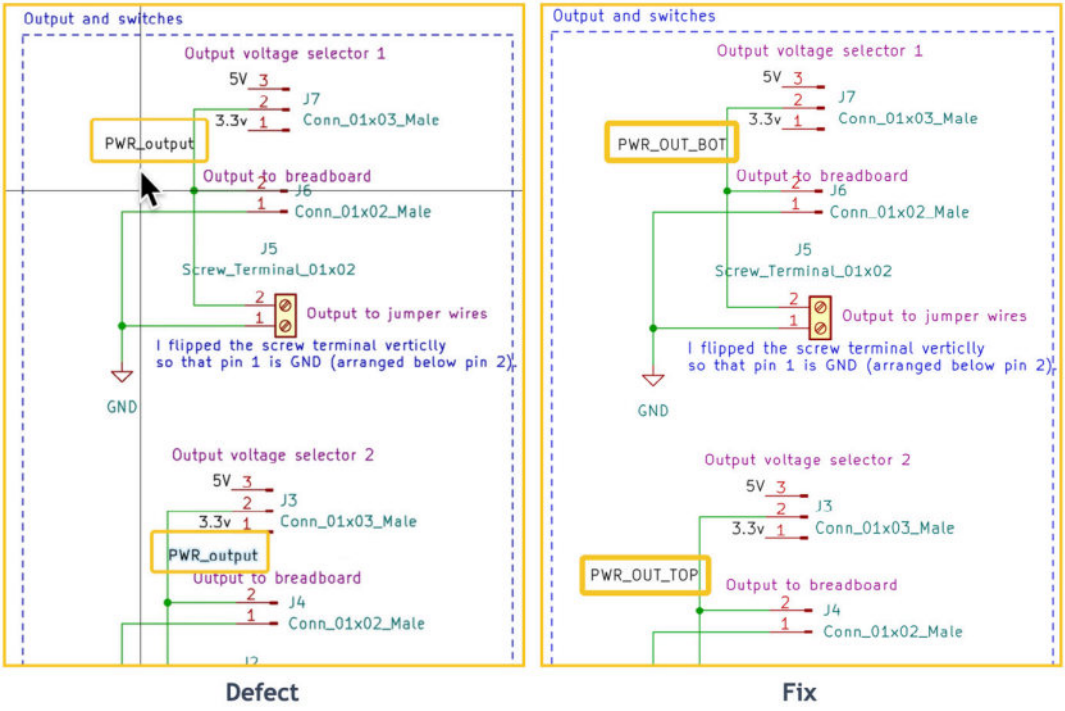


Figure 9.4.1.4: Fixing the net label defect.

I have now fixed the defect, but I want to improve the net classes setup. After introducing the two new net labels, KiCad automatically assigned them to the Default net class. I will re-assign these net labels to the "power_output" net class so that the member copper traces can inherit the "power_output" net class track geometry attributes (see below).

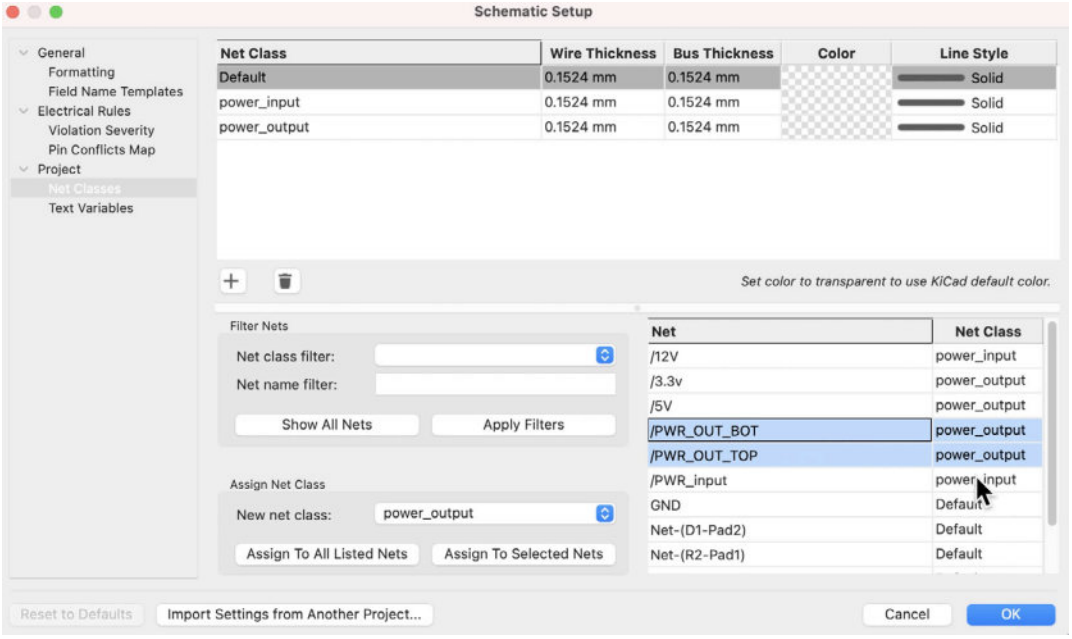


Figure 9.4.1.5: Added the two new net labels to the “power_output” net class.

This concludes the fix of the defect in the schematic diagram. Let’s continue with the layout.

4.2. Fix the layout

Open the layout editor, and import the changes from the schematic editor. Using white arrows in the figure below, I indicate the new ratsnest lines from this process.

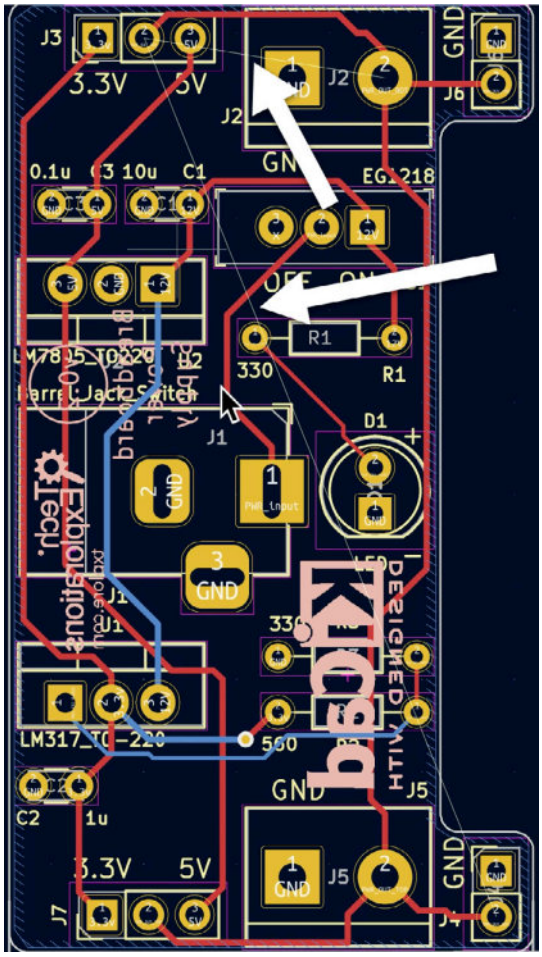


Figure 9.4.2.6: After importing the schematic changes, two new ratsnest lines appear.

My plan is this:

1. Reposition the J4 and J6 footprints and maintain the current distances between their pads.
2. Remove redundant tracks in the affected nets, and replace them with new tracks.

Repositioning of J4 and J6

Currently, J4 and J6 are locked. Go ahead and unlock them (click to select and type “L”), but take care not to move them yet. As per my measurements (see “3.2. 2 - Outline and constraints”), the distance between the two inside pads of J4 and J6 is 43.25 mm.

There’s a variety of techniques you can use to do repositioning. Here is what I did to minimize my workload.

Start by moving J4 (currently at the bottom of the PCB) and place it precisely over J6. This ensures that J4 is now placed at the exact location of J6.

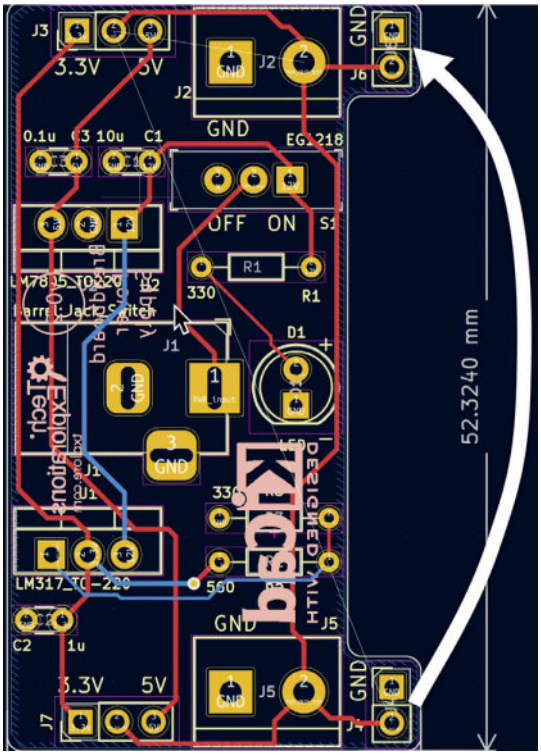


Figure 9.4.2.7: Move J4 over J6.

At this point, you have two identical footprints that overlap. Click on the two overlapping footprints to reveal the context menu, and select “Footprint J4”.

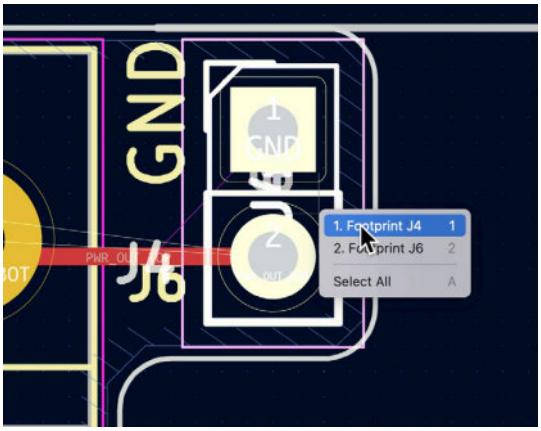


Figure 9.4.2.8: Select J4.

With J4 selected, type “L” to lock it in. Again, click on the overlapping footprints and this time select “Footprint J6”.

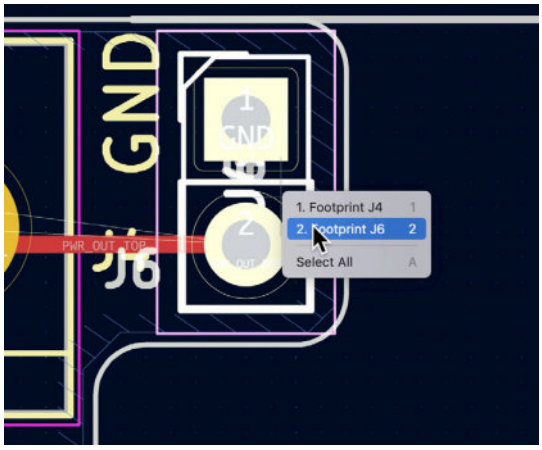


Figure 9.4.2.9: Select J6.

Place your cursor in the center of pad 2, and press the space bar to reset the status bar counters. This will help you ensure that the new placement of J6 will be on the same vertical coordinate as J4 (ensure the dx remains zero). Try to get the dx value to 45.77 mm (see below).

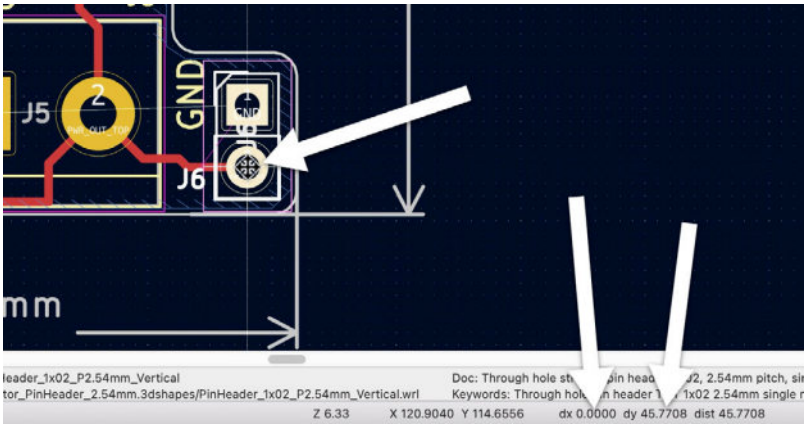


Figure 9.4.2.10: J6 repositioned to the bottom of the board.

With J4 and J6 repositioned, continue to measure the distance between the centers of their inner pads (pad 1 of J6 and pad 2 of J4). This distance should be 43.25 mm. Use the measure tool to confirm this distance (see below).

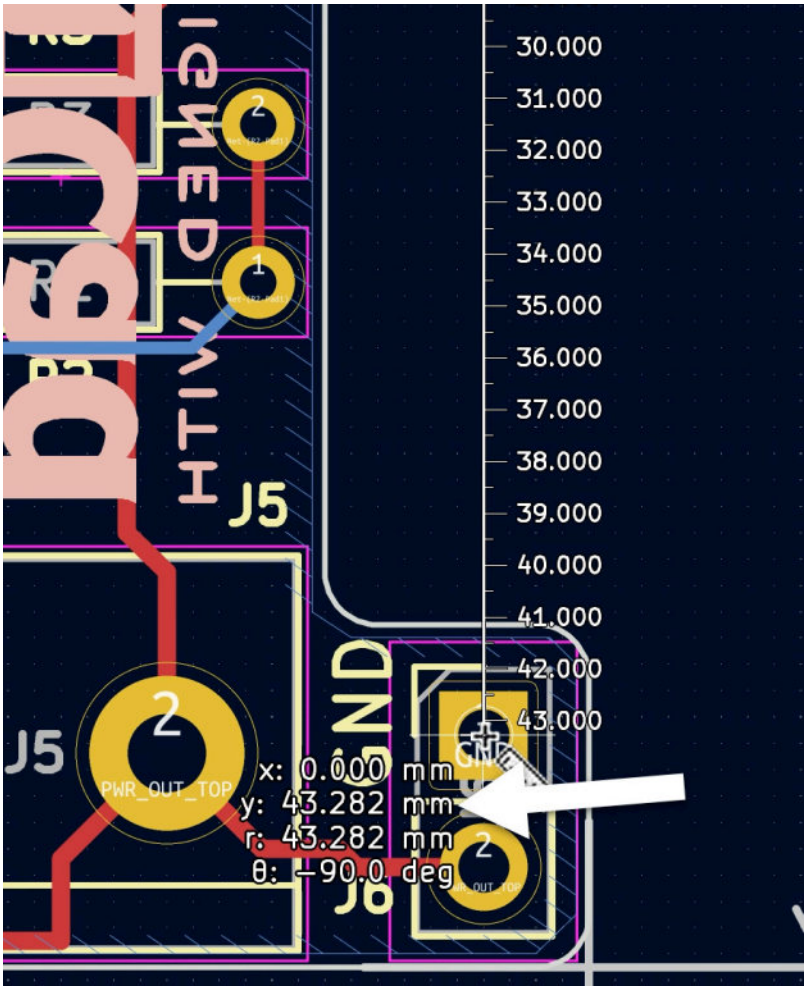


Figure 9.4.2.11: The distance between the centers of the inner pads is 43.282 mm.

The distance between the repositioned pads is slightly larger than the original, but I am comfortable with the difference because the tolerance of the mini-breadboard is larger.

Lock J6 in place and continue with the drawing of the new copper tracks.

Drawing of new copper tracks

Several copper tracks have been affected because of the net label changes in the schematic. The layout editor maintains the tracks you drew before the change in the schematic. Therefore, the first task to complete here is to delete incorrect or redundant tracks. Use the interactive delete tool for this purpose.

In the two figures below, I show details from the top and bottom parts of the board. I have deleted several of the original tracks, and the board is ready for re-wiring.

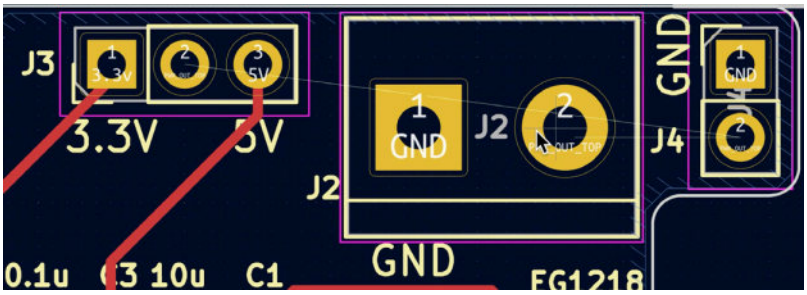


Figure 9.4.2.12: Detail, the top part of the PCB with deleted tracks.

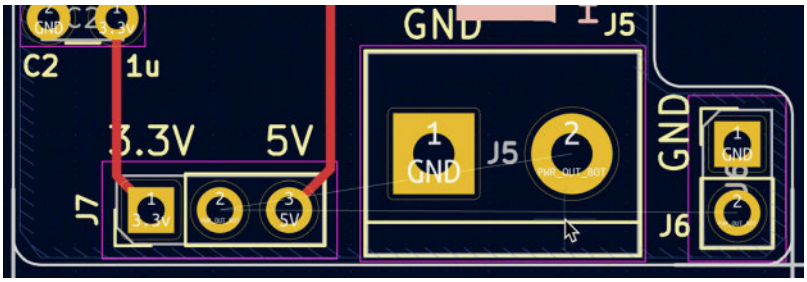


Figure 9.4.2.13: Detail, the bottom part of the PCB with deleted tracks.

Continue to draw the new tracks (except for those that connect to the GND pads) on the top copper layer. You can see the result below.

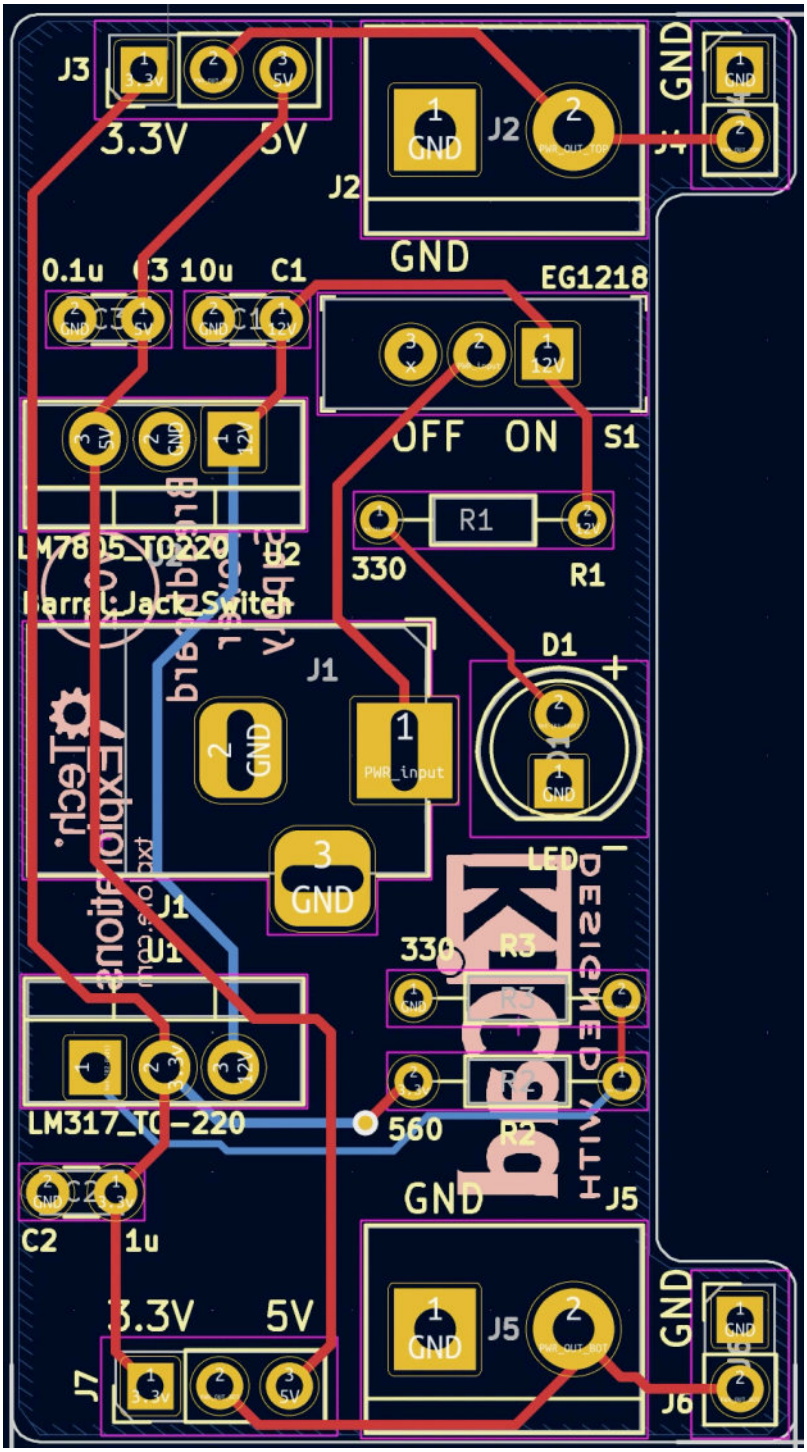


Figure 9.4.2.14: PCB with the new copper tracks.

Run a DRC to ensure nothing is broken. My DRC came out clean (no errors, no warnings).

Both defects are now fixed.

Part 10: Project - A 4 x 8 x 8 LED matrix array

1. Introduction

Welcome to Part Ten! In the chapters that follow, you will design a PCB that can hold four 8x8 LED matrix displays controlled by an Arduino Pro mini. The board also includes two push buttons to which you can assign arbitrary functions. I plan to use mine as a Pomodoro timer. I will use one button to select the lap duration (say, 15, 20, or 25 minutes) and the other button to reset the timer. When the duration I have set expires, the display will blink to let me know.

You can see the two sides of the populated PCB in the photograph below:

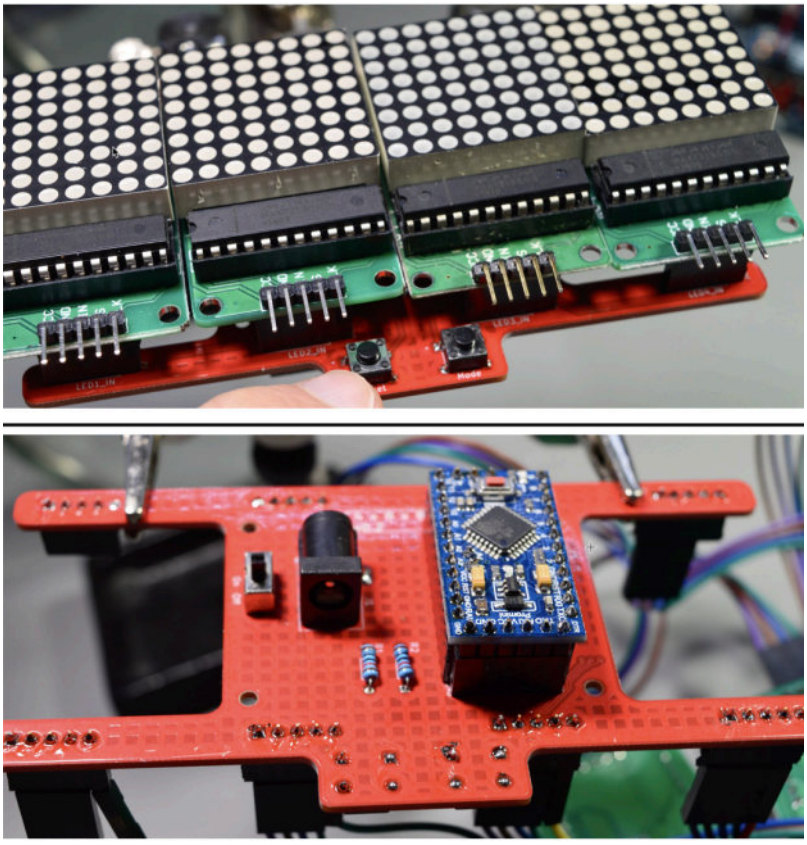


Figure 10.1.1: The PCB in this project, populated.

The inspiration for this project is that I forget to get up from my desk at regular intervals. I could use a desktop or phone Pomodoro app or even a classic mechanical Pomodoro timer, but why buy one when I can make one?

I decided to use an Arduino Pro Mini with the Atmega328P MCU and an on-board 5V voltage regulator, because:

- I have a lot of them.

- They are easy to find in the market
- They are very cheap.
- They are small.
- They have an onboard regulator.
- They are accurate enough to count short periods.
- I don't need a clock function, so the absence of a real-time clock is not an issue.

This board does not contain a UART to USB interface, so you must provide an external bridging device. This device is required for programming the microcontroller. I use a USD to serial adaptor from Freetronics, but there are many other options in the market.

I decided to use the 8x8 LED matrix display module with the MAX7819 controller chip because I like its versatility. An 8x8 LED matrix display can show numbers, text, and simple graphics. I also like the way it looks from a distance and the ability to create a simple animation. All this gives me scope to add features to my Pomodoro project in the future.

Below you can see the schematic for this PCB. This is the schematic you will create by the end of Chapter 10.2.

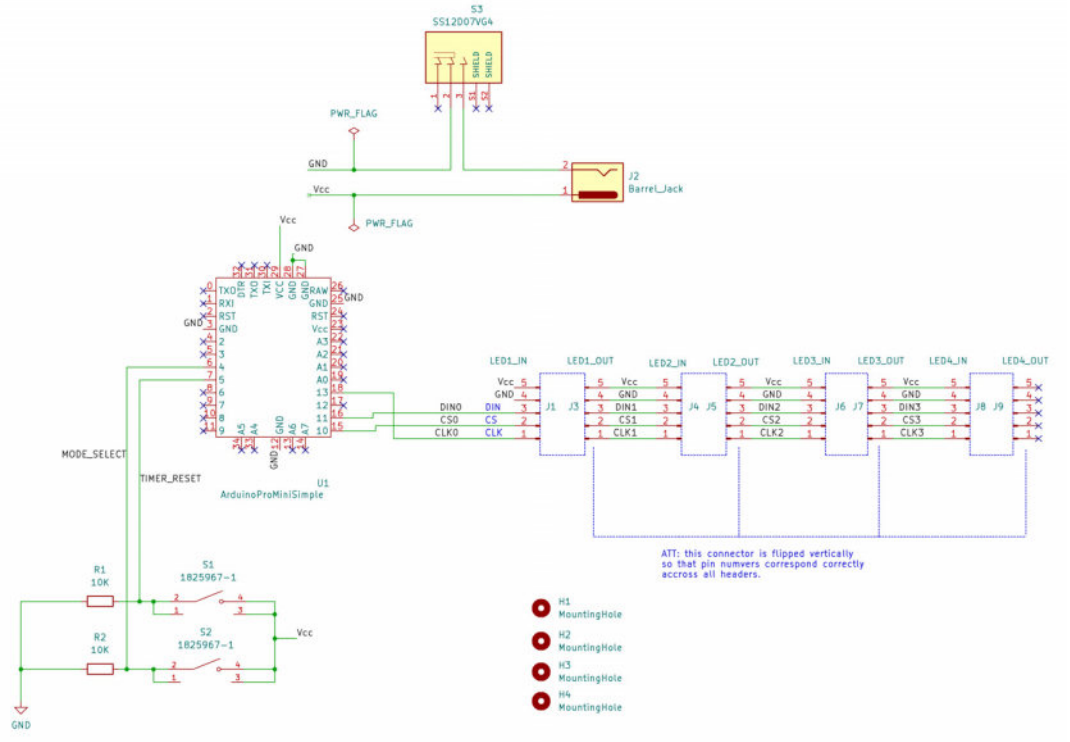


Figure 10.1.2: The project schematic.

To draw this schematic, I have opted to use line wires for all pins that are nearby. I have added net labels to all power, data, clock nets, and button signal nets. I was unable to find a schematic symbol for the Arduino Pro Mini

board that I planned to use, so I created one, along with its matching footprint. In the layout, I have included four mounting holes. To avoid getting error messages from the DRC, I have associated the mounting hole footprints with mounting hole symbols in the schematic.

Another consideration was how to deal with the LED matrix modules. I was not able to find a symbol and footprint for this device, so I had two options:

1. Create a custom symbol and footprint, as I did with the Arduino Pro Mini module.

2. Ignore the module, and concentrate on the headers.

Since I went with option 1 for the Arduino module, I opted for option 2 for the LED modules. Instead of treating each LED module as a single device, I treated it as a set of two-pin headers. My objective, then, was to wire the header symbols correctly and place their footprints precisely on the PCB (see the discussion on the PCB below).

Below is the layout of the PCB, as it will be at the end of Chapter 10.3:

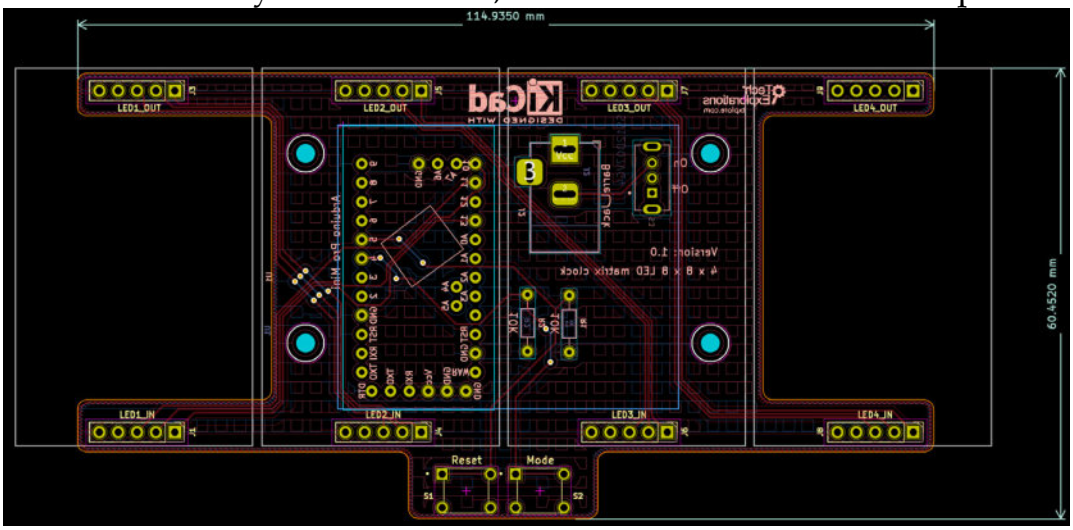


Figure 10.1.3: The project layout.

The dominating feature of this board is its shape. I wanted to experiment with a shape that uses “arms” that extend from its center to hold the LED modules rather than a conventional rectangular shape. I did not do this to reduce the manufacturing cost. Even though the shape you see above has two significant parts of the substrate material removed, the manufacturing cost relates to the all-inclusive height and width of the board (you can see those dimensions in the figure above). But I do think that the board with the four arms extending from the center looks great. Along with the rounded edges and the button notch at the bottom, I am satisfied with the physical design aspect of the board.

A significant challenge for the layout design of this board is the position of the pin headers for the LED modules. As I mentioned above, I decided to treat the LED modules as a set of two headers each (input and output). The positions of those headers must be very accurate. If the headers are too far from their neighbors, the four-module display will not look continuous but as four individual displays. If they are too close, the assembly will not be possible as there will not be enough space on the board to attach adjacent modules.

As a result, this project will give you the opportunity to use all of KiCad's measurement and alignment tools to make sure that the end product looks beautiful and works.

Below you can see the Bill of Materials for this project, as I have extracted it from the KiCad project (learn how later in this book):

Reference	Value	Footprint
H1-H4	MountingHole	MountingHole:MountingHole_2.5mm
J1	LED1_IN	Connector_PinHeader_2.54mm:PinHeader_1x05_P2.54mm_Vertical
J2	Barrel_Jack	Connector_BarrelJack:BarrelJack_Horizontal
J3	LED1_OUT	Connector_PinSocket_2.54mm:PinSocket_1x05_P2.54mm_Vertical
J4	LED2_IN	Connector_PinSocket_2.54mm:PinSocket_1x05_P2.54mm_Vertical
J5	LED2_OUT	Connector_PinSocket_2.54mm:PinSocket_1x05_P2.54mm_Vertical
J6	LED3_IN	Connector_PinSocket_2.54mm:PinSocket_1x05_P2.54mm_Vertical
J7	LED3_OUT	Connector_PinSocket_2.54mm:PinSocket_1x05_P2.54mm_Vertical
J8	LED4_IN	Connector_PinSocket_2.54mm:PinSocket_1x05_P2.54mm_Vertical
J9	LED4_OUT	Connector_PinSocket_2.54mm:PinSocket_1x05_P2.54mm_Vertical
R1, R2	10K	Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P7.62mm_Horizontal

S1, S2	1825967-1	1825967-1:SW_1825967-1
S3	SS12D07VG4	SS12D07VG4:SW_SS12D07VG4
U1	ArduinoProMiniSimple	DesktopLibrary:ArduinoProMiniCustom

Table 10.1.1: The Bill of Materials for this project.

Apart from the custom symbol and footprint Arduino Pro Mini, I used Snapeda to find the symbol-footprint pairs for the two buttons (S1 and S2) and the power barrel connector (S3).

Please note that if you want to build this gadget, you will need to modify the headers of the LED modules. I have written a chapter where I describe the process of modifying these headers, as well as how to assemble the gadget. You can find this chapter at the end of this part of the book.

If you are planning to assemble this gadget, please read the following carefully. I have designed this PCB to route power from the barrel connector to the Arduino Pro Mini's Vcc pin. As per the specifications of this Arduino, the Vcc pin requires regulated 5V power (I use the 5V version; a 3.3V version is also available). The Vcc pin bypasses the onboard voltage regulator and routes the 5V input directly to the microcontroller. As a result, you must be careful to connect your gadget to a trusted and regulated 5V power supply. I use a mains power supply that provides stable 5V power, at 500mA (see photograph below).

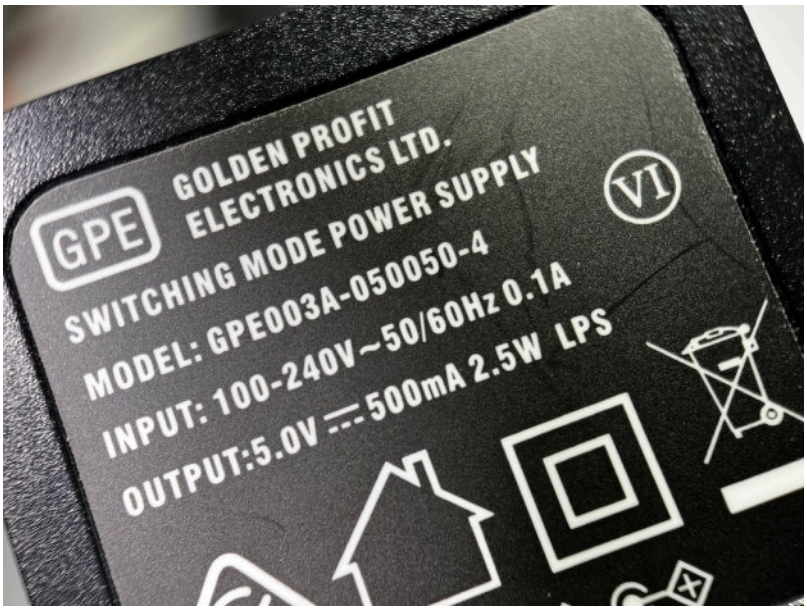


Figure 10.1.4: The power supply I use in this project.

This power supply provides sufficient current for the Arduino and the four LED displays.

If you don't have a mains power supply, you can use a USB-to-barrel connector cable, similar to the one in the photograph below.

Figure 10.1.5: A USB to barrel jack connector cable.

With the USB to barrel jack connector cable, you can power this gadget from your computer or a regular USB power supply. Either option (the regulated 5V mains power supply or the USB to barrel jack connector cable) allow you to use this gadget without any modifications.

An alternative design choice is to re-wire the positive pin of the (pin 1 of J2 in Figure 10.1.2) to pin 26 (RAW) of U1. The RAW pin, according to the specifications, can receive unregulated power between 5V and 12V for the 5V model or 3.35V and 12V (for the 3.3V model). This wiring option makes use of the onboard voltage regulator and gives you a wider range of safe-to-use power supplies. Consider which option you want to adopt, and make the necessary modifications to the circuit in the wiring step of the schematic workflow.

Let's begin with the schematic design in the next chapter.

2. Schematic design

In this chapter, you will complete the schematic design of this PCB by following the schematic design workflow. You learned about this workflow in Part 6 of the book.

2.1. 1 - Setup
Schematic design
Schema 1 - Setup
Schema 2 - Symbols
Schema 3 - Arrange, Annotate
Schema 3 - Associate
Schema 4 - Wiring
Schema 5 - Nets
Schema 6 - Electrical Rules Check
Schema 7 - Comments
Schema - Last-minute edits
Layout design
Layout 1 - Setup
Layout 2 - Outline and constraints
Layout 3 - Place components
Layout 2 supplemental - Refine outline
Layout 3 supplemental - Move footprints to back layer
Layout 4 - Route
Layout 4 - Copper fills
Layout 5 - Silkscreen
Layout 6 - Design Rules Check
Layout 7 - Manufacture
Bonus - 3D shapes
Bonus - Found a bug in the schematic! (and fix)
The assembled and working PCB

Let's begin with the schematic design and set up the project. Start KiCad, and from the main project window, click File —> New Project. Select a

location for the new project and give it a name. I called mine “4x8x8 LED Matrix Clock”.

KiCad’s main project window and the project directory look like this:

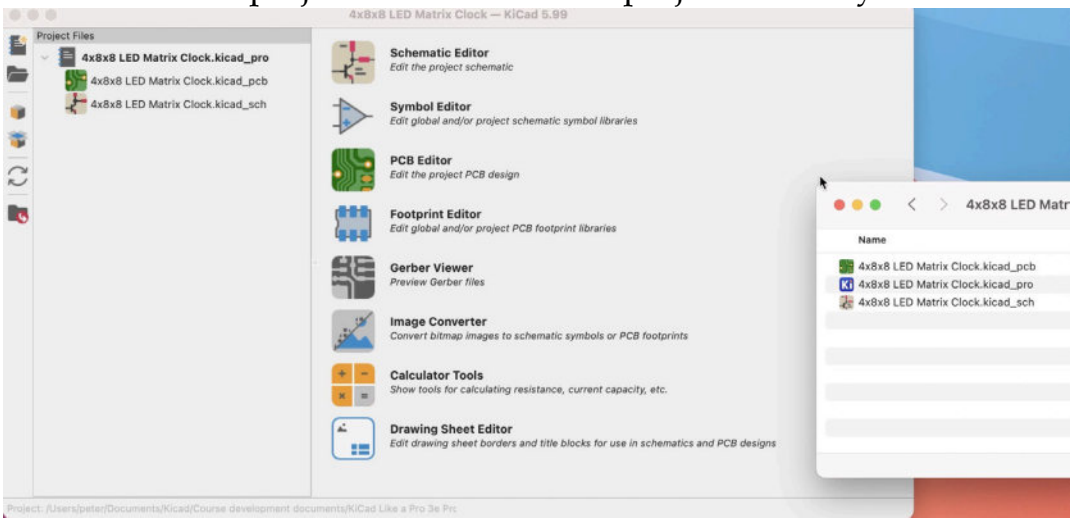


Figure 10.2.1.1: The new project files.

Let’s continue with the setup of the schematic editor. Start Eeschema, and inspect your default settings in the Schematic Setup window. I have only made two changes:

1. In Net Classes, I have added the Power net class. I will assign nets to this class step five of the workflow.
2. In Text Variables, I have added a variable with the name “design_version.” For the value, I have set “1.0”. I will update this variable when I update the project and export it in the information box of the schematic and layout editors.

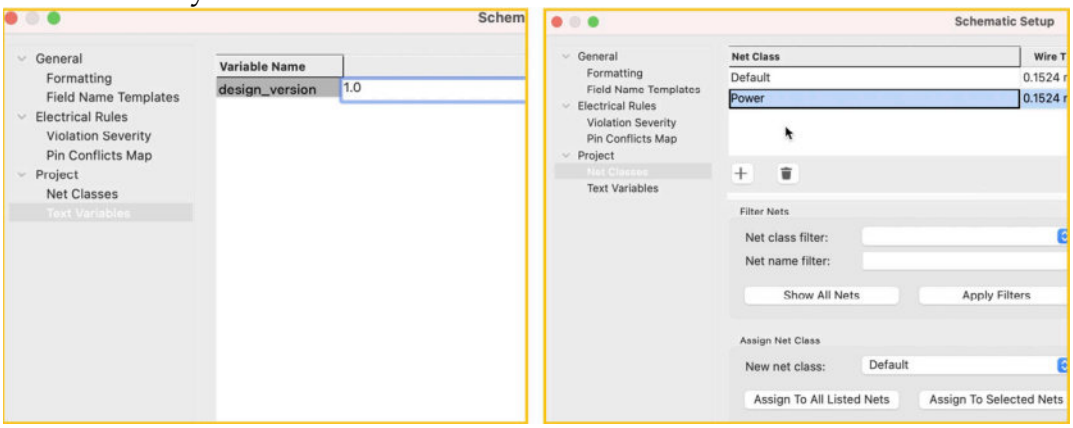


Figure 10.2.1.2: Changes to the Schematic Setup.

Next, inspect the settings in the Preferences window. I have made two changes to the default settings here:

1. Under Colors, I have changed the background to white.
2. Under Field Name Templates, I have added the field name “Purpose.”

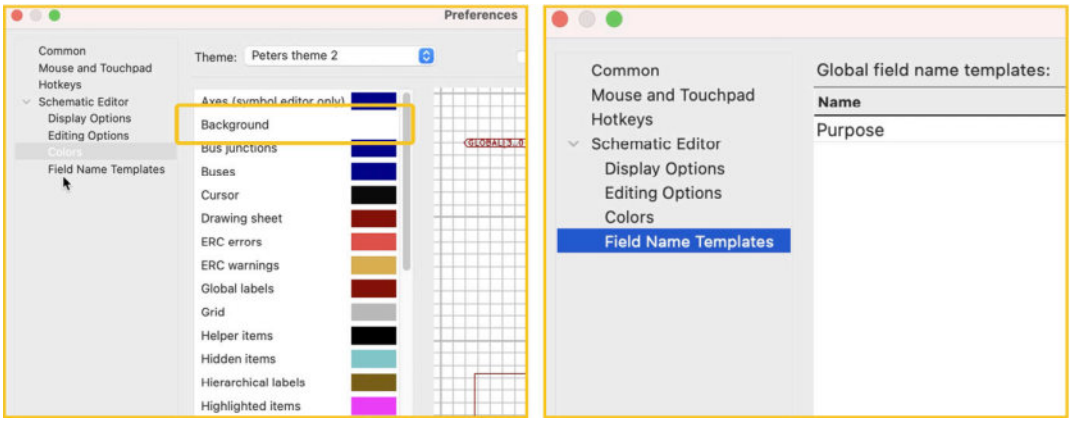


Figure 10.2.1.3: Changes to the Preferences.

Finally, open the Page Settings window and fill in the project information. Notice that in the Revision field, I am using the text variable notation "${text_variable}" to import the text variable with the name "design_version." You can use text variables anywhere there is text.

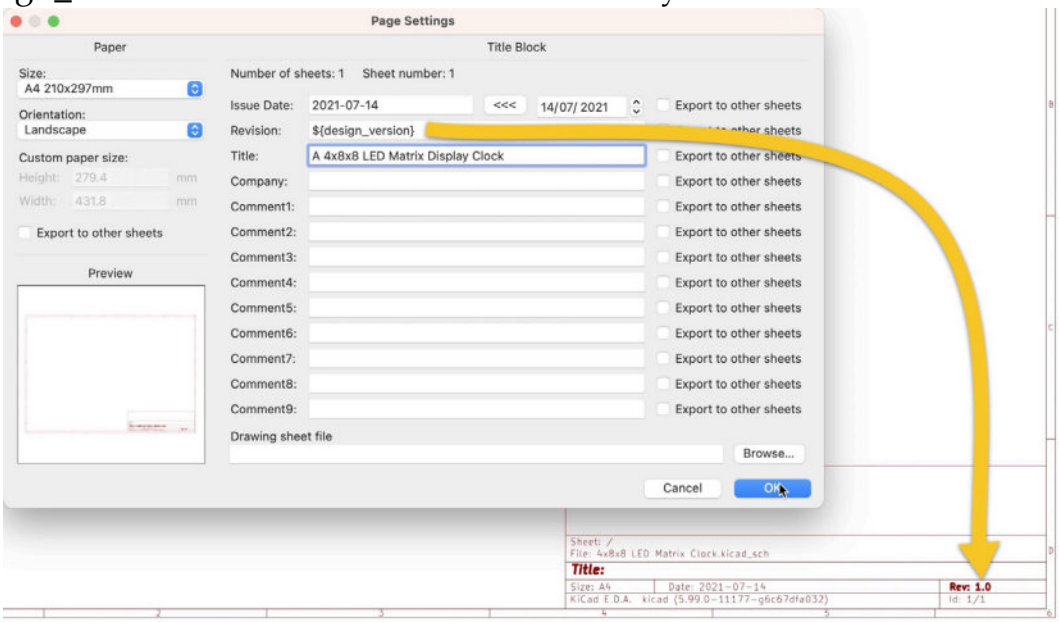


Figure 10.2.1.4: The Page Settings window, using a text variable.

The project setup is complete. Let's continue with the symbols in the next step of the workflow.

2.2. 2 - Symbols

In this segment, you will complete step two of the schematic design workflow. In other words, you will add the symbols required for your schematic to the schematic editor.

Most of the symbols needed for this project are available in KiCad's symbol libraries. For this project, I will also be using symbols from the Digikey library and a couple that I have downloaded from Snapeda. There is

also one that I have created. You can see the relevant part of my Symbol Libraries window below:

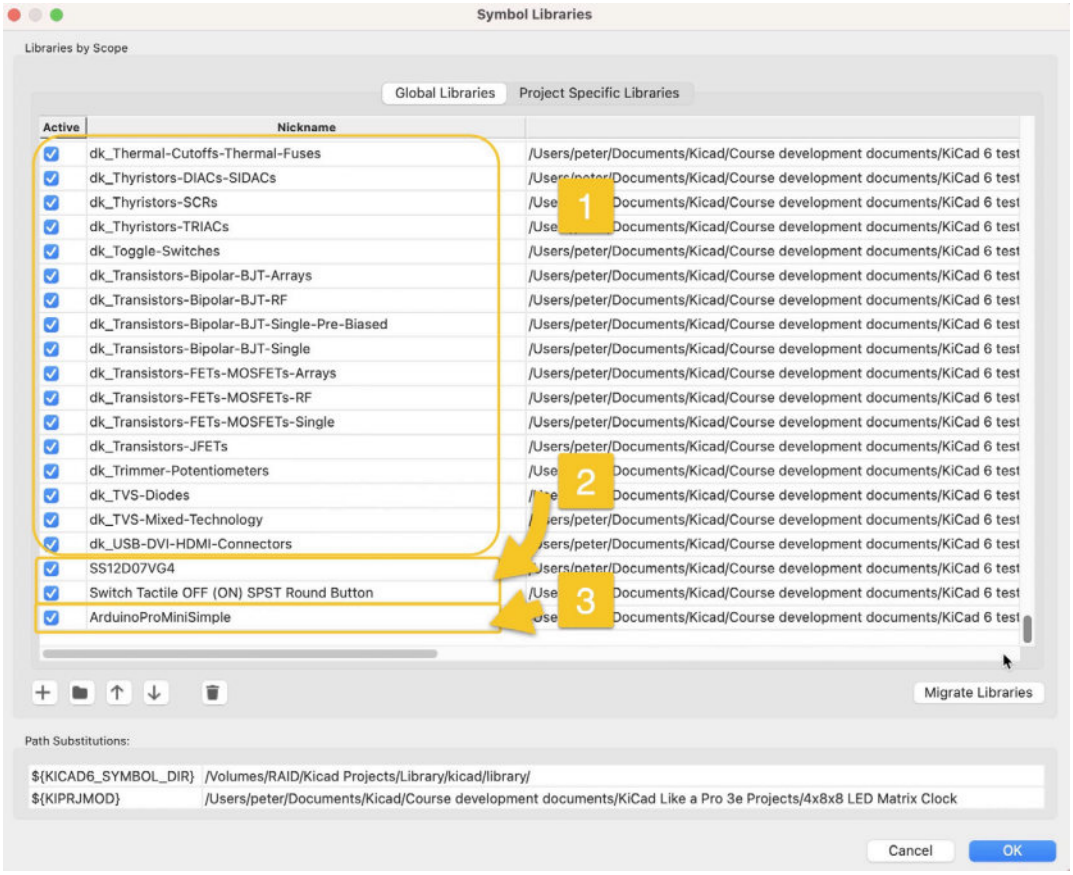


Figure 10.2.2.5: My project symbol libraries.

Here are more details about the symbols I am using in this project:

1. Slide switch, symbol "SS12D07VG4" from Snapeda, "2" in the figure above. (download here).
2. Tactile button "Switch Tactile OFF (ON) SPST Round Button" from Snapeda, "2" in the figure above (download here). If you prefer, you can also use a tactile switch from the Digikey ("dk_") library, "1" in the figure above.
3. ArduinoProMiniSimple, custom made by Peter, "3" in the figure above.
4. Everything else is available in KiCad's libraries

You can find all symbols and footprints I'm using in this project in the project's Github repository (see under "Libraries." In the same place, you will also find the complete KiCad project files.

For a refresher on using Snapeda, please read chapter "7. How to find schematic symbols on the Internet". For a refresher on installing symbols, read chapter "8. How to install symbol libraries in bulk". And if you want to learn how to create the Arduino Pro Mini custom symbol

"9. How to create a custom symbol". All these chapters are in Part 7 of this book.

Once you have installed the required symbols in the KiCad symbol library, add them to the sheet. Below, I am using the Symbol Chooser to add the symbol for the Arduino Pro Mini:

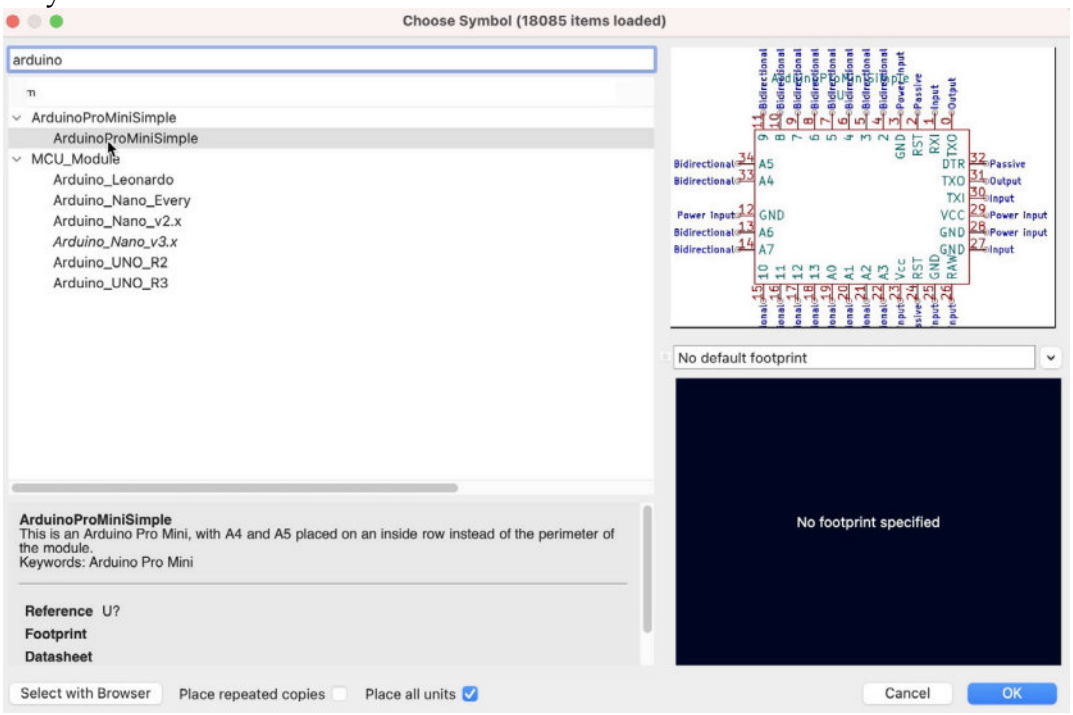


Figure 10.2.2.6: Adding the symbol for the Arduino Pro Mini.

Continue with the rest of the symbols until all symbols I have listed in Table 10.2.1 are in the schematic sheet. By the end of this process, your schematic sheet should look like this:

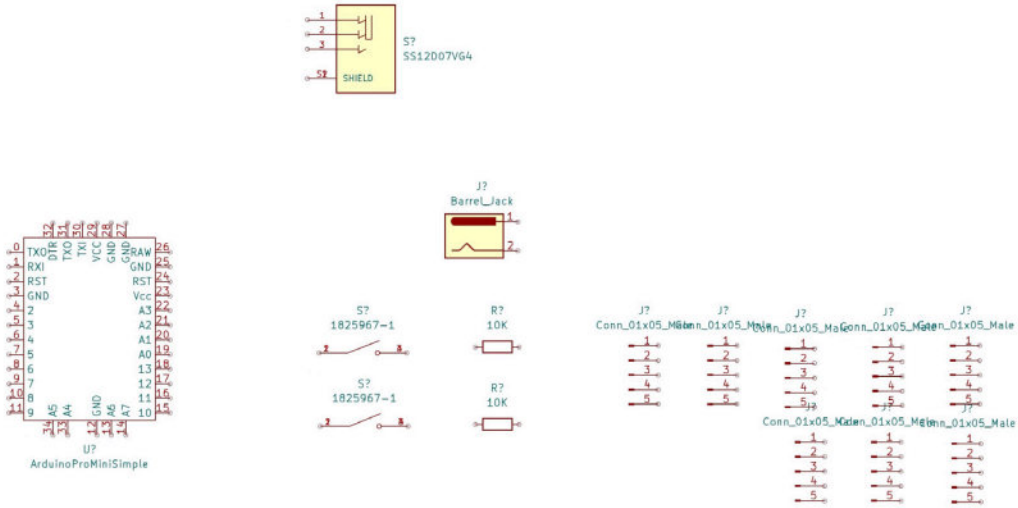


Figure 10.2.2.7: All symbols added to the schematic sheet.

In the figure above, notice that I have added the values for the two resistors, “10K” for each.

With the schematic symbols added to the sheet, you can now continue with step two of the workflow: arrange and annotate the symbols.

2.3. 3 - Arrange, Annotate

In this segment, you will arrange the symbols on the sheet to prepare them for wiring in step four. After that, you'll use the automated annotator to set identifiers for each symbol.

The circuit is simple, so the arrangement of the symbols on the sheet is straightforward.

I'll place the two symbols that make up the power input group together at the top part of the schematic. I'll place the Arduino module towards the left of the schematic to allow enough space for the connectors on the right side. Finally, I'll place the two switches and their resistors in a group below the Arduino symbol.

You can see the final placement of the symbols below:

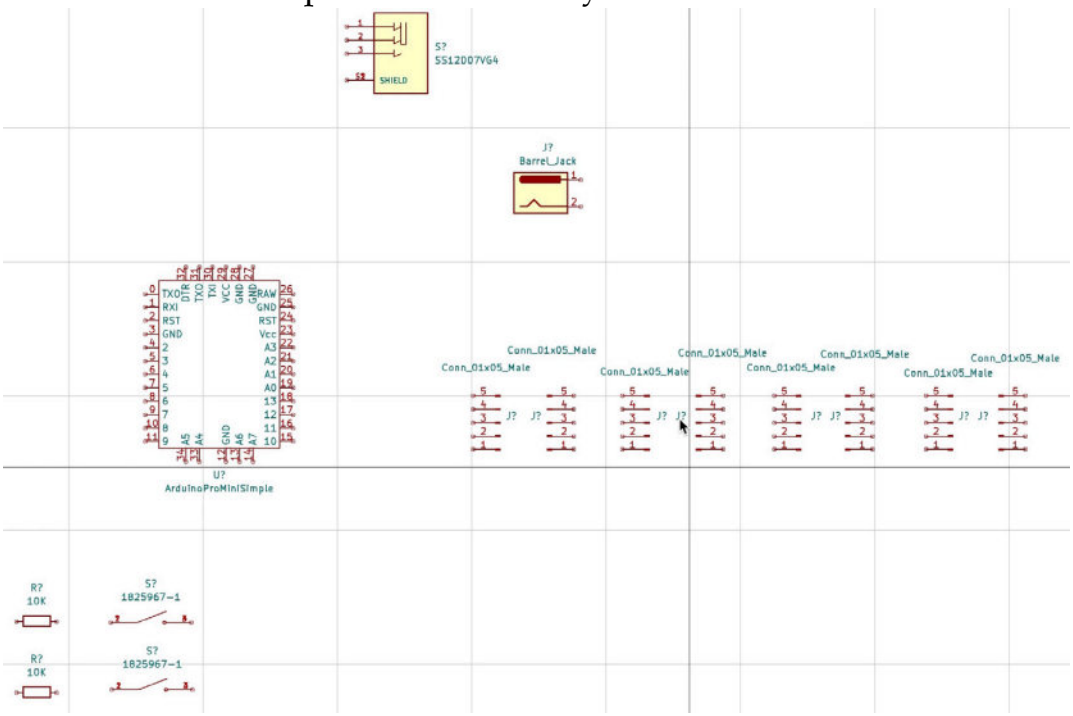


Figure 10.2.3.8: Symbols placed in the schematic sheet.

I spent most of my arrangement time with the connectors. The 8x8 LED display modules use the SPI interface to communicate with the Arduino and other modules in series. They have one five-pin connector to receive data from

the Arduino or another module in the series and a second five-pin connector to provide data, clock, and power to the next module.

To represent these connectors, I use the "Conn_01x05_Male" symbol. Remember that wires connect to the end of the pin with the small circle. This is important when it comes to orienting the connectors appropriately. For example, the connector that represents the input header of the first display module must be oriented so that the pin circles point towards the Arduino symbol. On the other side of the first display module is the second five-pin connector. This connector's pins must have their circular ends pointing towards the second display module.

Because of these orientations, the pin numbers will not match. Instead, you will see that pin 5 of header 1 is opposite pin 1 of header two, as you can see below:

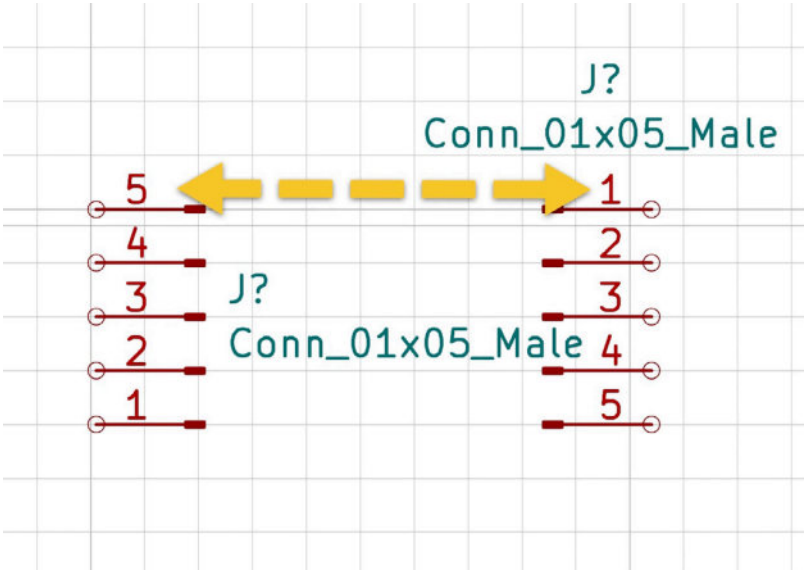


Figure 10.2.3.9: Pins are mismatched for neighboring header symbols.

To make the PIN numbers match and properly represent how the pins are arranged in the LED display module, you must flip the data output header vertically. This will maintain the horizontal orientation of the module and only change the pin numbers. To flip a symbol vertically, right-click on the symbol to reveal its context menu, and select "Mirror Vertically":

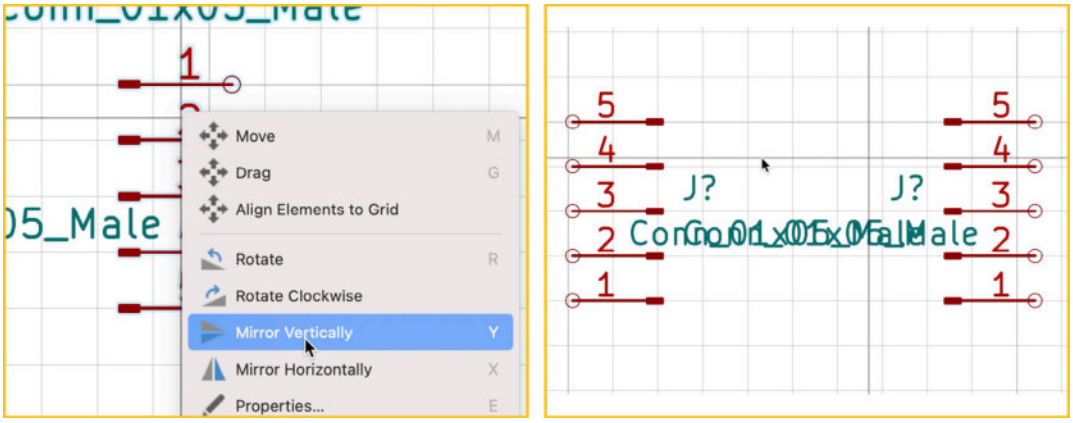


Figure 10.2.3.10: Mirror symbol vertically.

Arrange the connectors and place them appropriately so that the schematic looks like the one in figure 10.2.3.8.

Finally, run the automated annotation tool. In the figure below, you can see the result in my instance of the project.

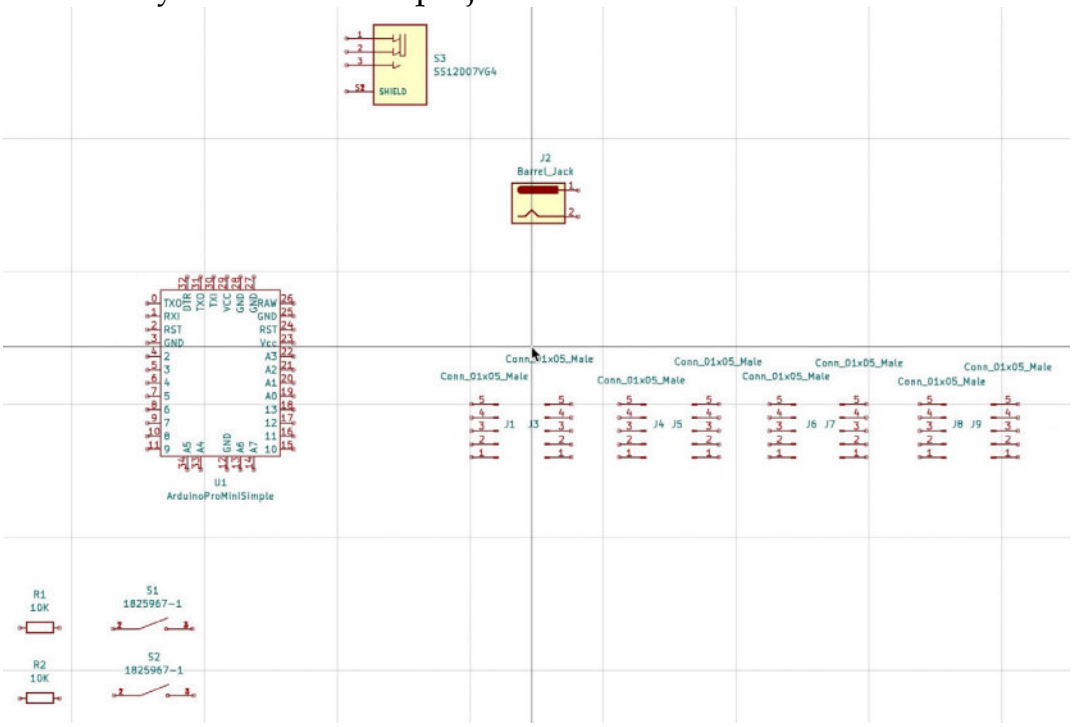


Figure 10.2.3.11: The annotated schematic symbols.

The designators in your instance should match those in the figure above. If there are any discrepancies, you can either leave them as they are and remember them in the remainder of this project or manually change the designators to match what you see above.

When you are ready, continue with the symbol-footprint associations in the next segment.

2.4. 3 - Associate

In this segment, you will complete step three of the schematic workflow, where you will set the associations between schematic symbols and their footprint counterparts. In Table 10.2.1 you can see the project symbols in column two and their associated footprints in column three.

The figure below shows the footprint libraries I set up in my KiCad's Footprint Libraries table. I have marked the specific libraries I will be using in this project in the yellow box.

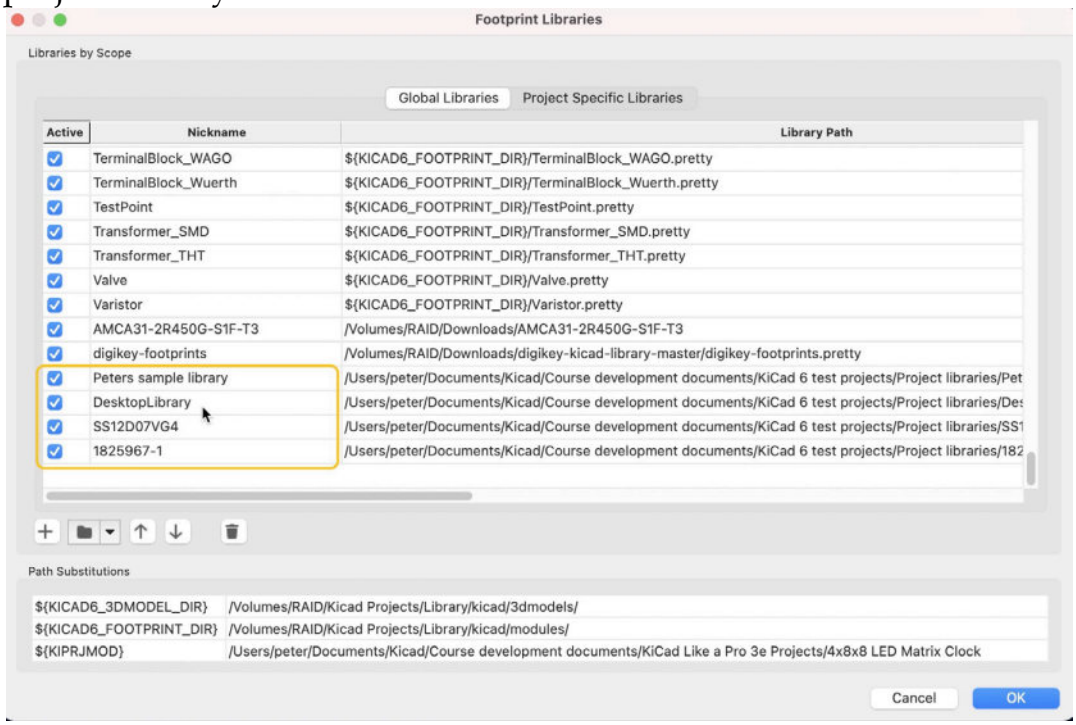


Figure 10.2.4.12: The footprint libraries I will use in this project.

You can find all symbols and footprints I'm using in this project in the project's Github repository (see under "Libraries." In the same place, you will also find the complete KiCad project files.

Once you install the necessary footprints, return to Eeschema and open the Associations window. Use the information in Table 10.2.1 to help you find the libraries and footprints in the left and right panes of the window. If KiCad asks you if you want to convert legacy footprints to the new format, select “yes”.

By the end of the process, your association's table will look like this:

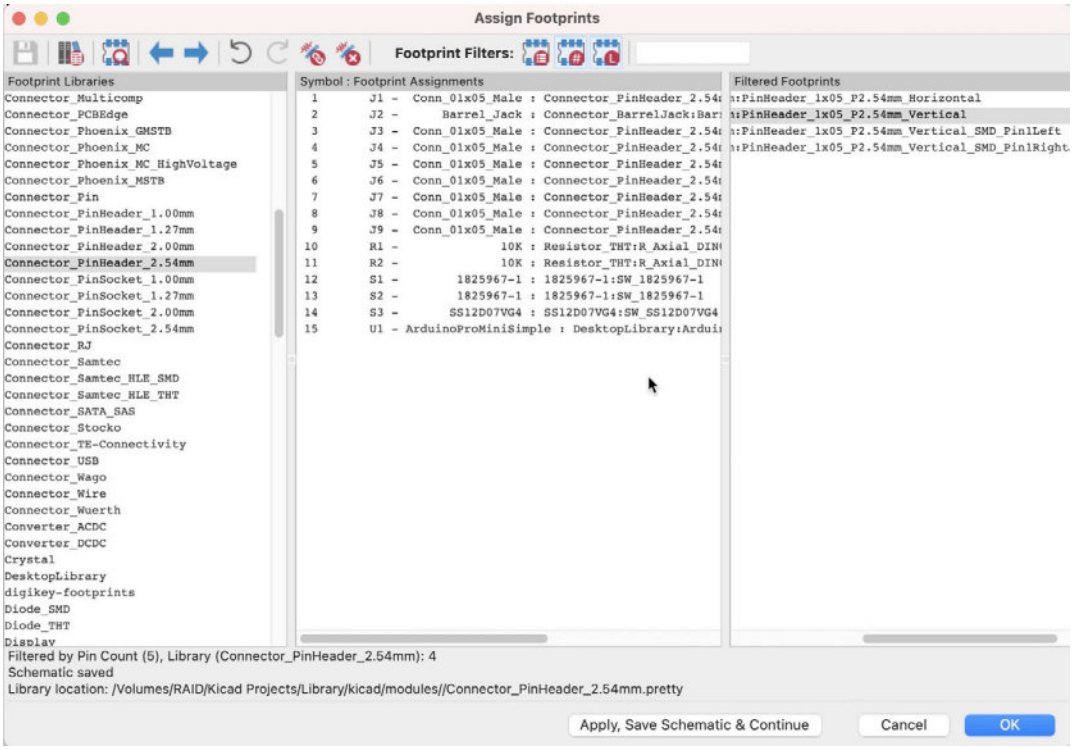


Figure 10.2.4.13: The symbol-footprint association's table.

This completes step three of the schematic design workflow. Let's continue with step four, wiring.

2.5. 4 - Wiring

In this segment of the chapter, I will work on step four of the schematic design workflow. I will connect pins by:

1. Drawing wires using the Wire tool.
2. Partially complete step five of the workflow by creating some of the named nets. You will create the remaining named nets in step five of the workflow.

At this point, the schematic looks as I left it at the end of the last segment of this chapter (see Figure 10.2.5.11). I will begin the wiring process using the Wire tool from the right toolbar to draw lines connecting nearby pins. I will begin from the buttons in the bottom left corner of the schematic:

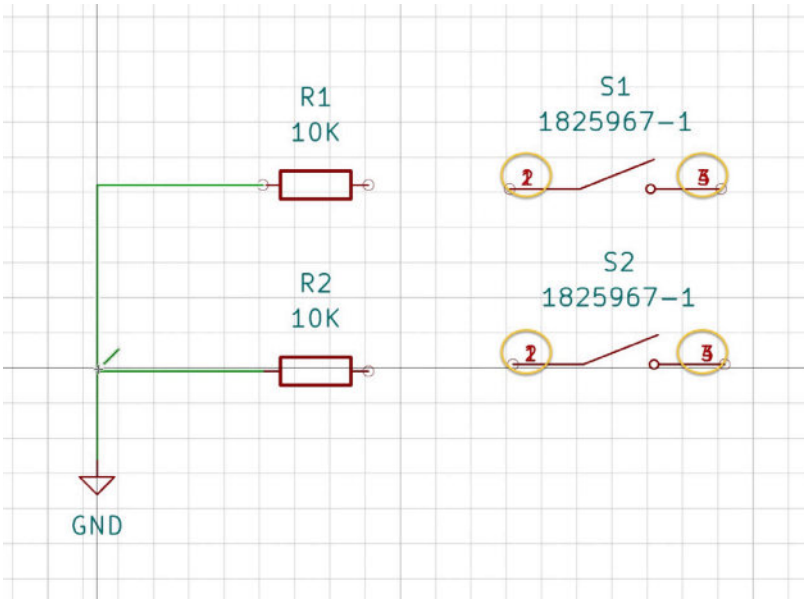


Figure 10.2.5.14: Using the wire tool to connect pins.

As in the figure above, connect the left pin of the two resistors to a GND symbol. Notice that the S1 and S2 symbols have overlapping pin numbers. I have marked those with a yellow circle. I have noticed this with other symbols too, so this is an opportunity to understand what is happening here and how to solve this problem.

Let's take a closer look at S1. Double-click on it to bring up its Properties window. Then, click on Edit Symbol.

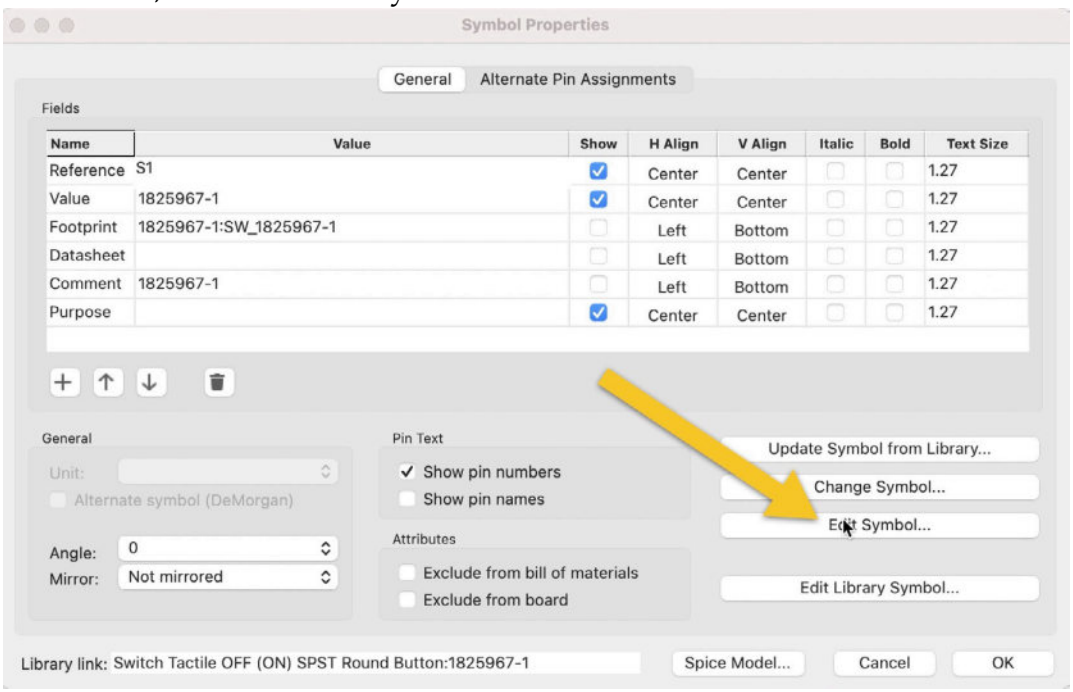


Figure 10.2.5.15: The properties window for S1.

The symbol editor will come up, displaying the S1 symbol. Notice that on both sides of the symbol are overlapping pins. In the figure below, on the left side is the original state of the symbol with the overlapping pins. On the right side, I have separated the pins by clicking and dragging them downwards.

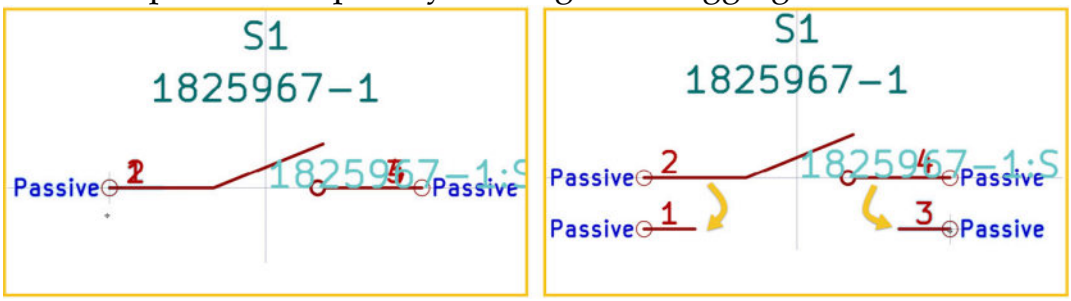


Figure 10.2.5.16: Separate the overlapping pins by dragging down.

After you have separated the pins, save the symbol and close the symbol editor. You will see that S1 is updated in the schematic sheet, and all four of its pins are visible. Repeat the same process for S2, and complete the wiring so that the resistor and button network group look like this:

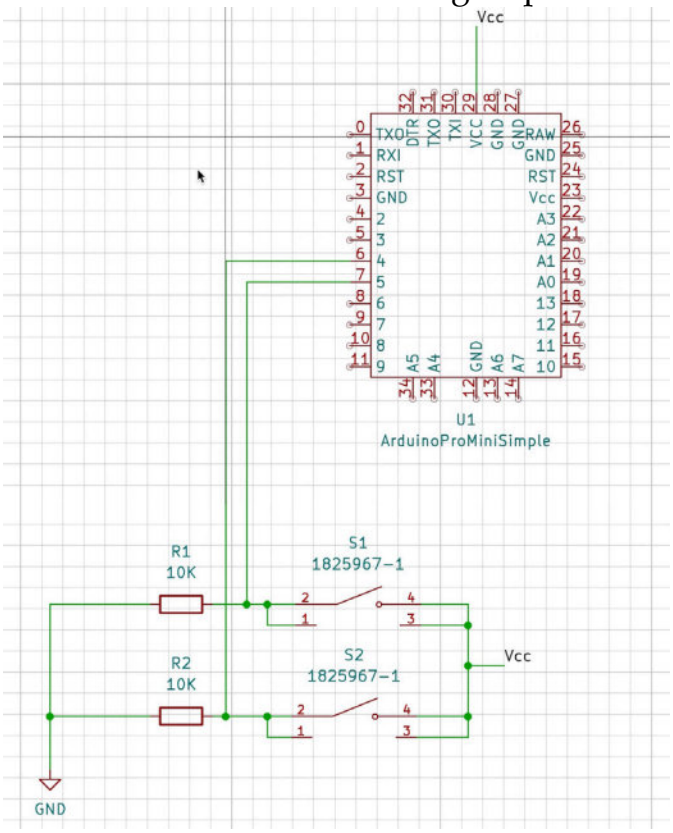


Figure 10.2.5.17: Resistors and buttons are wired.

In the figure above, I have used a net label to connect pins 4 and 3 of S1 and S2 with pin 29 of U1. The net label is "Vcc." I will create more such labels as needed.

Continue with the power input group:

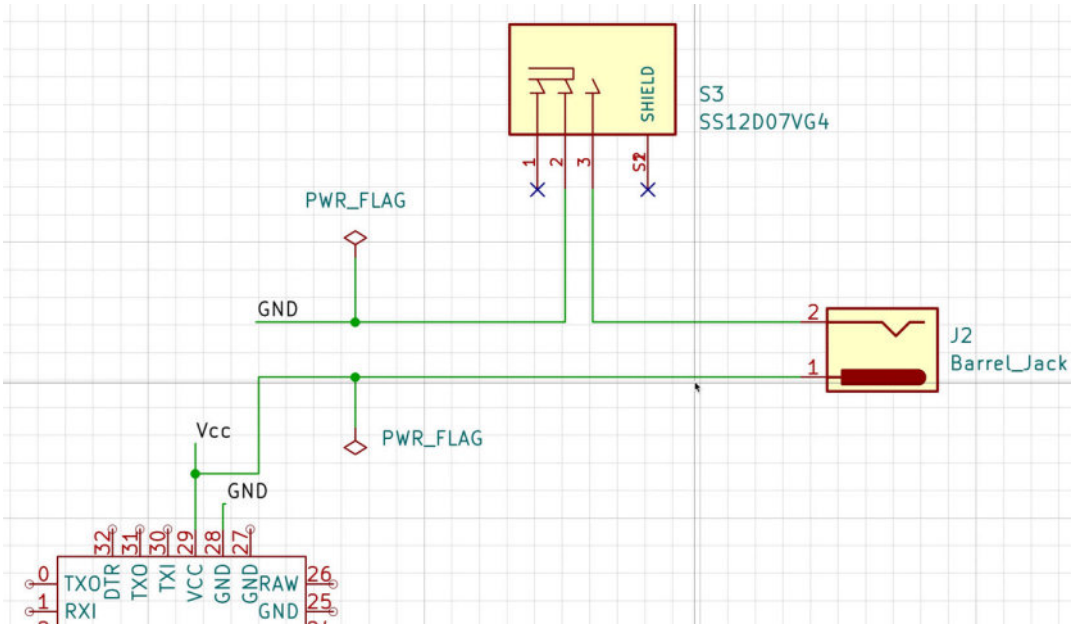


Figure 10.2.5.18: Power input group wired.

In the figure above, notice:

1. In S3, two shield pins overlap. Since those pins are unconnected, I choose to leave those pins as they are instead of separating the pins as I did with S1 and S2. As you will find out later when I do the ERC, this decision turned incorrect. I will need to separate the two pins to pass the ERC. More about this in the segment on step six of the workflow.
2. S3 pin 1 is unconnected, as are the two SHIELD pins.
3. Use the PWR_FLAG symbol to mark the GND and Vcc nets as power nets.

Continue with the connectors. Below is the final wiring for this group of symbols (not final, more work needs to be done there):

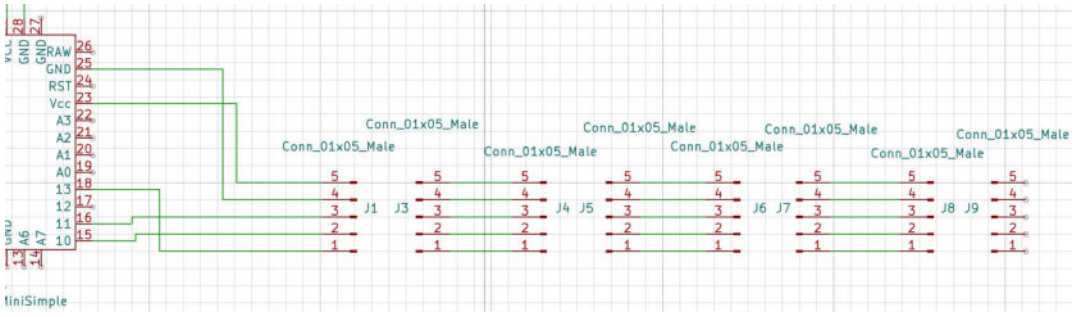


Figure 10.2.5.19: LED modules wiring (in progress).

The wiring is almost complete. Two tasks remain:

1. Mark any unconnected pins.
2. Add remaining net labels. You will work on this in the next segment (step five of the workflow).

Below is the fully wired schematic before finishing the remaining named nets:

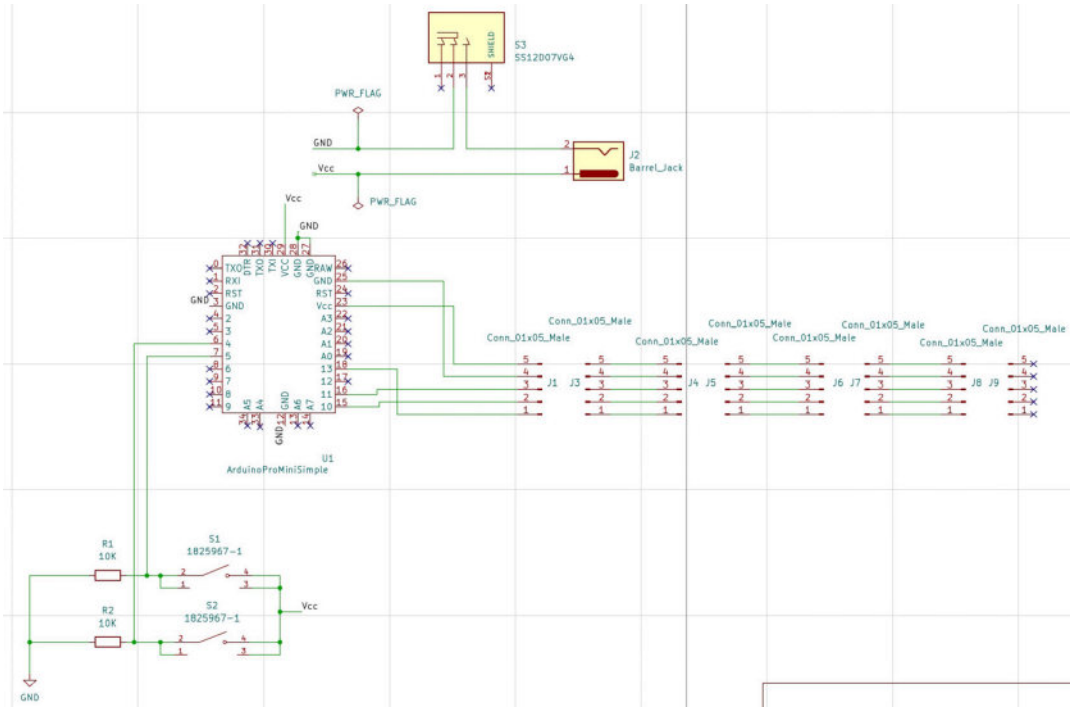


Figure 10.2.5.20: Fully wired schematic.

Let's continue to step five of the workflow, where you will add the remaining named nets.

2.6. 5 - Nets

In this segment of the chapter, you will add the remaining named nets. Most of them will go in the LED display modules, and two of them belong to the button signal nets.

Start with the button signal nets:

- MODE_SELECT
- TIMER_RESET

Here is the updated schematic for the group:

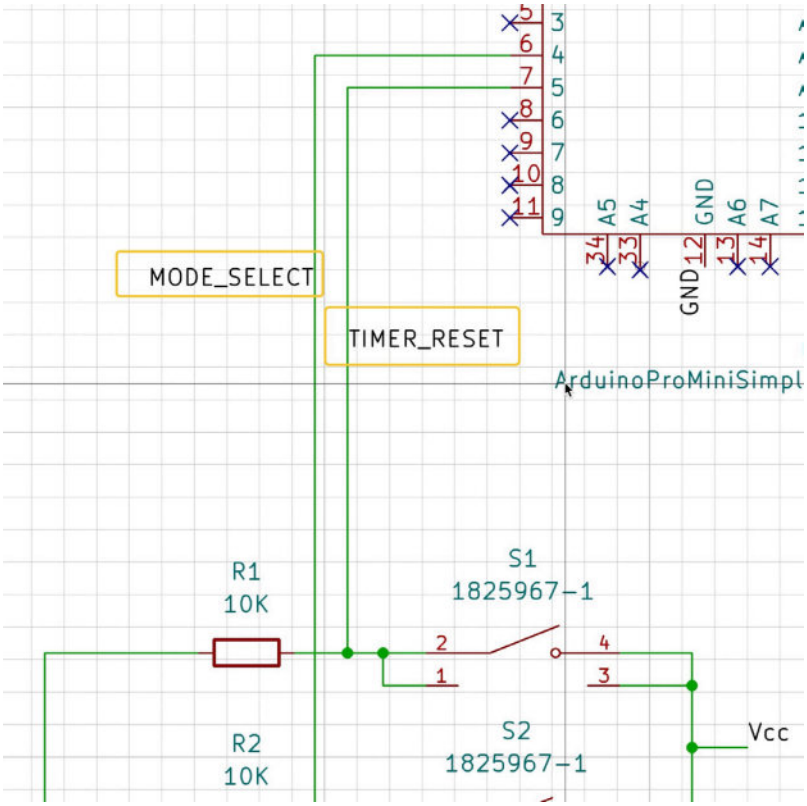


Figure 10.2.6.21: Named nets the button signal nets.

Then, continue with the LED displays group:

- Vcc
- GND
- CLKx
- DINx
- CSs

In the list above, "x" denotes a number from zero to three for the four modules. Here is the updated schematic for this group:

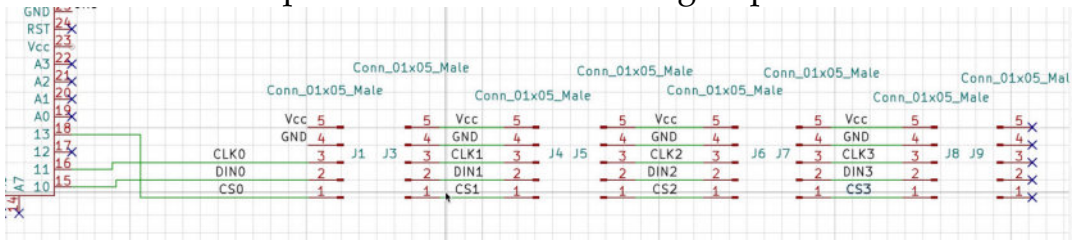


Figure 10.2.6.22: Named nets the LED display nets.

In the figure above, notice that I have removed the original wire lines that connected pins five and four of J1 to pins 29 and 28 of U1 to de-clutter the schematic. I have marked the pins of J9 with the unconnected symbol.

This completes step five of the schematic design workflow. The schematic is fully wired, and the important nets are named. Below is the final version of the schematic:

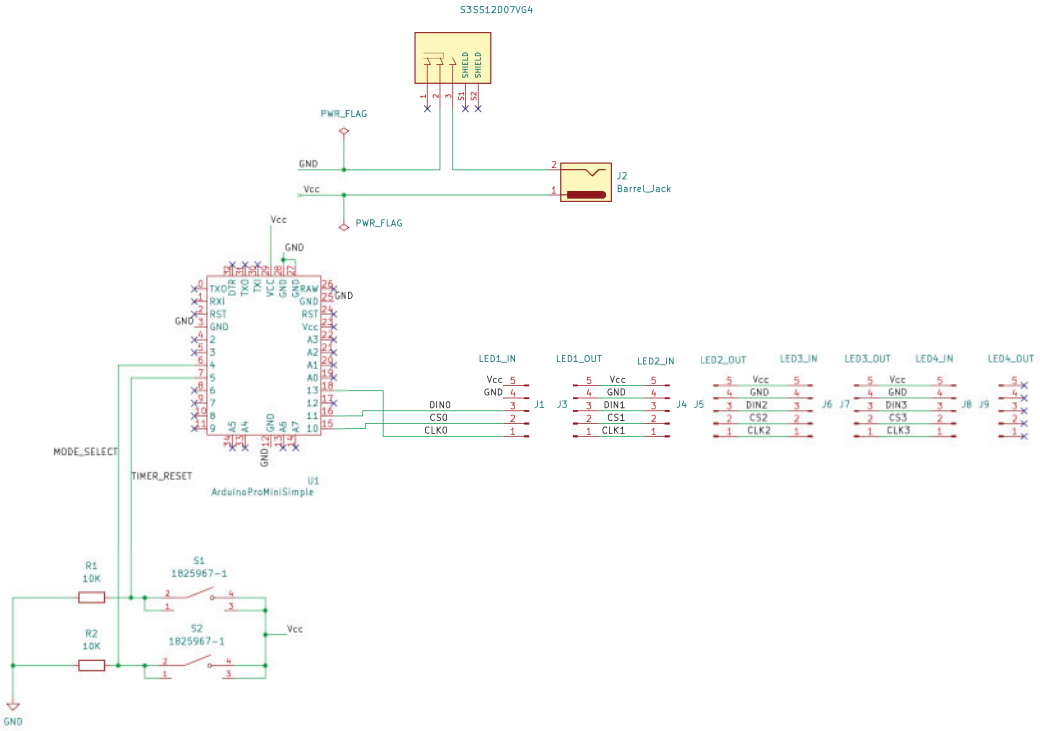


Figure 10.2.6.23: Wiring is complete.

Let's continue with an ERC.

2.7. 6 - Electrical Rules Check

With the schematic wiring complete, it is time to do a final Electrical Rules Check. Bring up the ERC window and click "Run ERC." The ERC will show a message indicating that the schematic is not fully annotated. The symbols that are not annotated in my instance of the schematic are the two PWR_FLAG symbols. Click on the "Show Annotation dialog" link to open the auto-annotations window and annotate the last two symbols. Go back to the ERC window and click "Run ERC' again.

Here is the result:

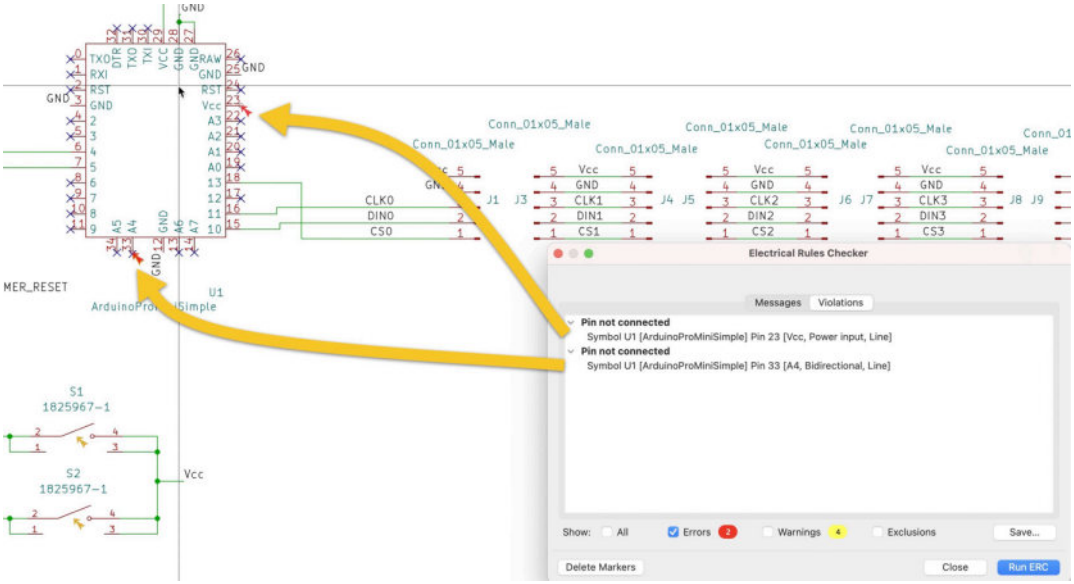


Figure 10.2.7.24: Two violation that I must fix.

I was wrong! There are two "pin not connected" violations that I have to fix before continuing. The first one is pin 23; I forgot to place a "not connected" symbol on the pin.

The second one is about pin 33. I did have a Not Connected symbol there, but it wasn't correctly attached to the pin. This is what it looks like when zoomed-in:

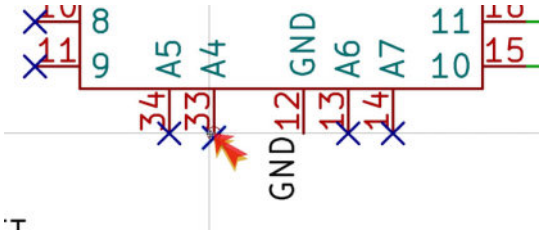


Figure 10.2.7.25: “x” not on the spot.

I have fixed the first violation by placing a "not connected" symbol on the pin and the second violation by correcting the symbol's position.

I repeat the ERC, and here is the result:

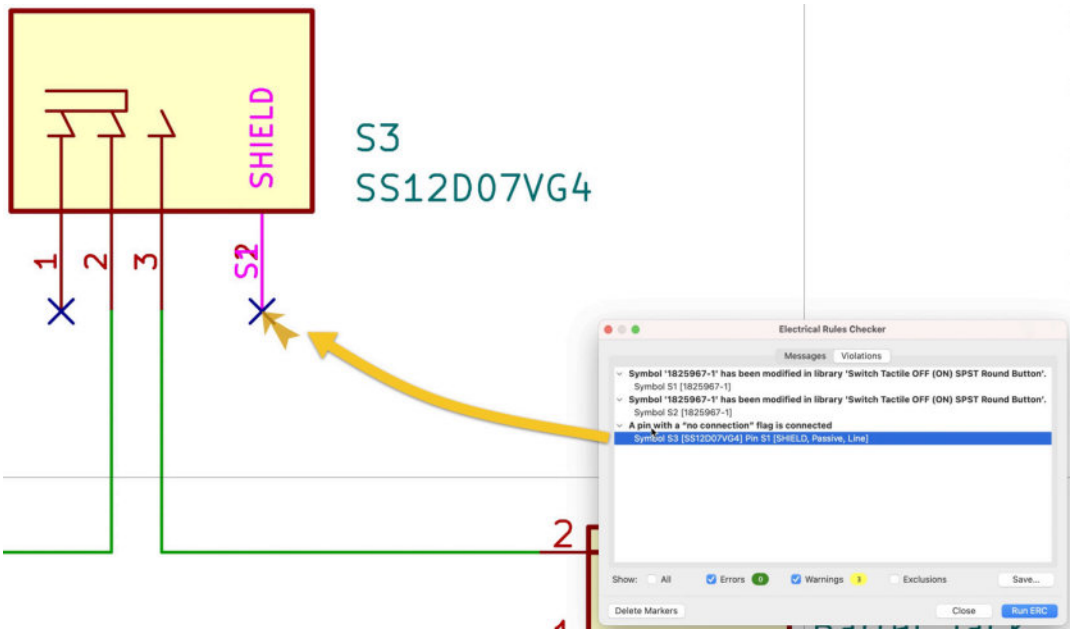


Figure 10.2.7.26: Three violations. I have to fix one of them.

Unfortunately, the ERC returned three violations this time. I can ignore the first two as they related to the two button symbols that I modified and separated their overlapping pins.

The third one relates to a similar issue. In symbol S3, there are two overlapping SHIELD pins. Earlier, I had chosen not to separate those pins. I did attach the unconnected symbol to one of them, but the second overlapping pin remains unconnected. To solve the last "no connect" violation, I have to separate the two overlapping pins in the same way, I did for the two buttons.

In the symbol editor, drag the overlapping pin-up and save the symbol:

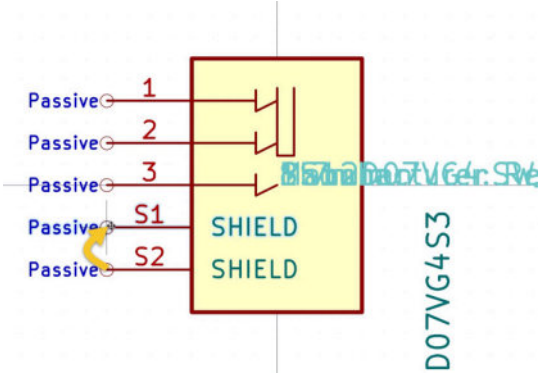


Figure 10.2.7.27: Separated pins S1 and S2.

Save the symbol and return it to the schematic editor. Notice that symbol S3 shows the two (previously overlapping) pins S1 and S2 (below, left). One has the "X" ("not-connected") symbol, while the other does not. Add the unconnected pin symbol to the second pin (below, right):

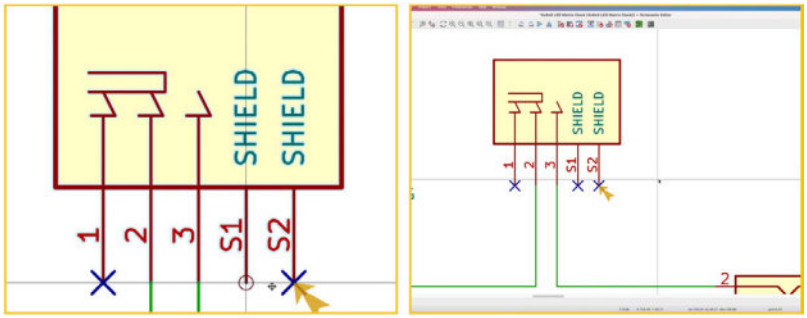


Figure 10.2.7.28: Add an unconnected pin symbol to S1.

Let's try the ERC one more time. Here is the result:

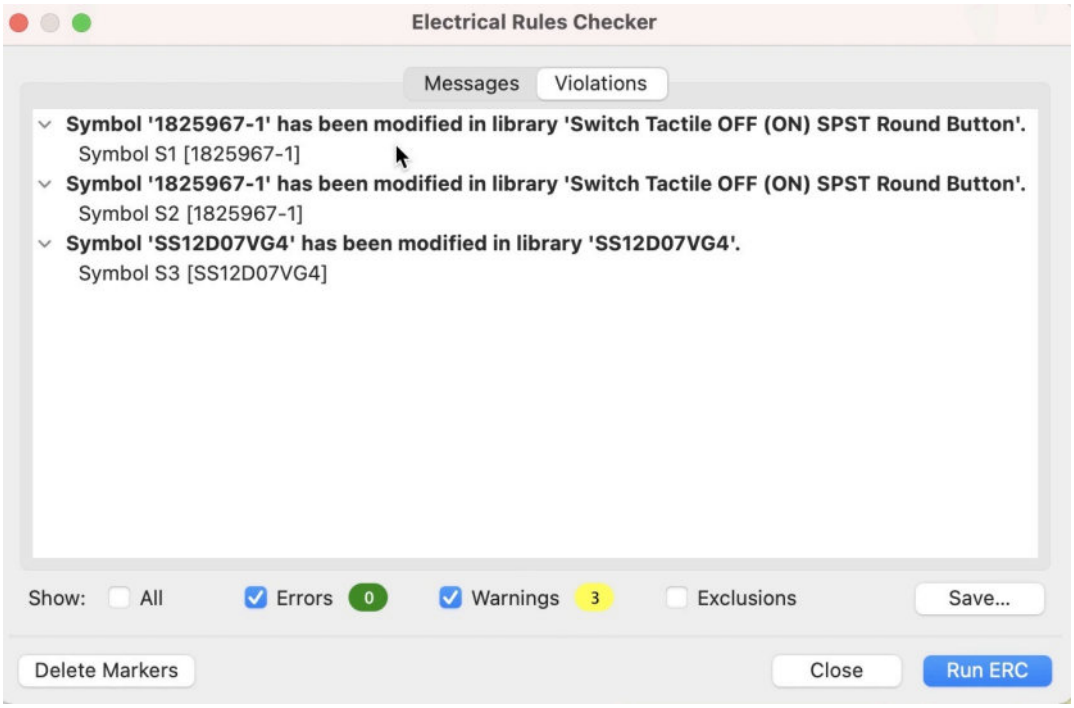


Figure 10.2.7.29: Three violations that I can ignore.

You can review each violation by clicking on it. The editor will pan the schematic and place the crosshair on the location of the violation. This way you can quickly review violation in the schematic, in addition to the ERC listing. You can also mute a violation by right-clicking on it and select “Exclude this violation”.

There are three violations that I can ignore. All three warn me that three symbols have been modified in the library.

After the fixes brought forward by issues revealed by the ERC, the schematic looks like this:

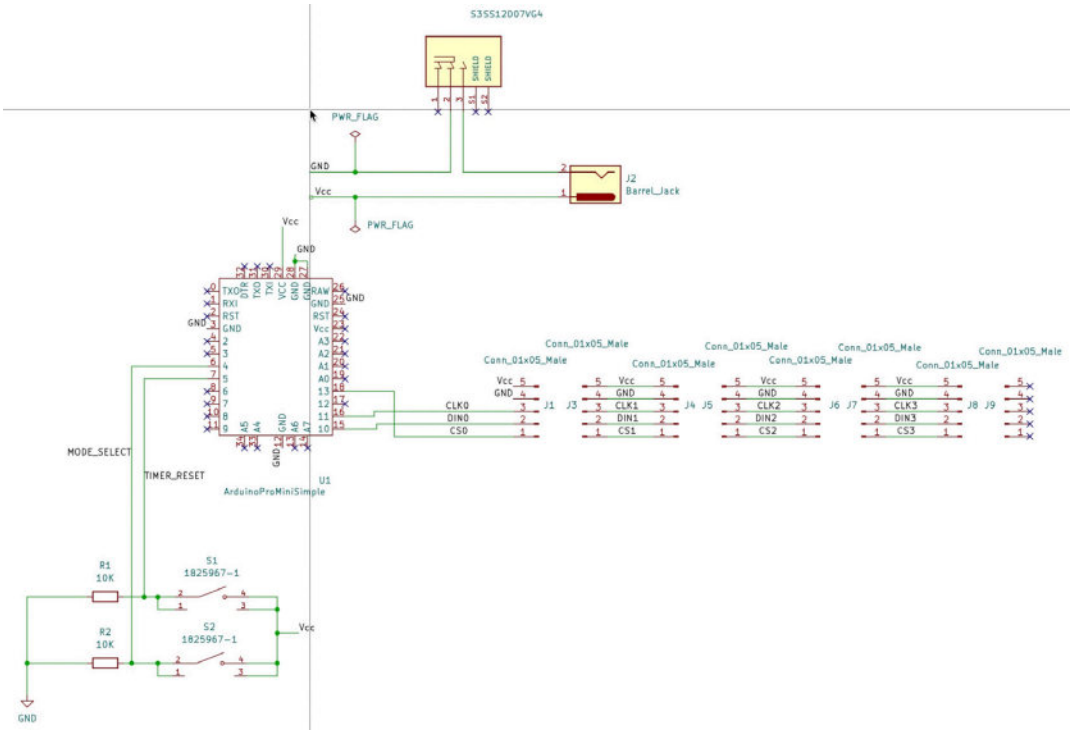


Figure 10.2.7.30: The project schematic, final.

Let's proceed with step seven of the workflow, and add a few explanatory comments to the sheet.

2.8. 7 - Comments

In this step of the workflow, you will add graphics and text comments to improve the readability of the schematic.

Here are the comments I suggest you add:

1. Simple line boxes that join the LED display connectors resemble a single module each. They can look like this:

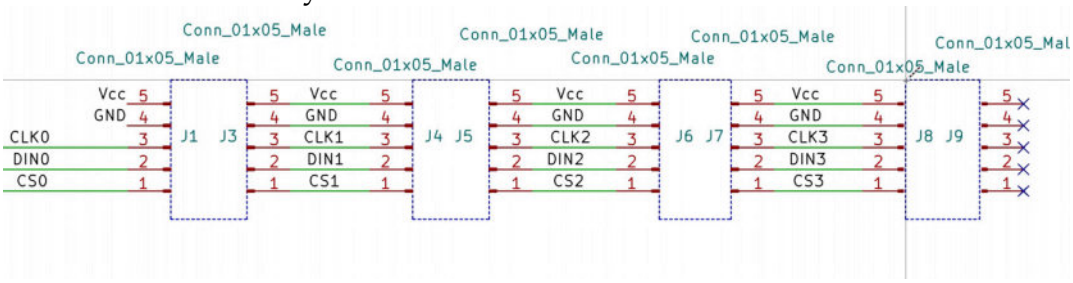


Figure 10.2.8.31: Boxes represent the LED display modules.

2. Replace the original pin header names with more descriptive ones. For example, ”Conn_01x05_Male” becomes “LED1_IN”. Here is the display block after editing the names:

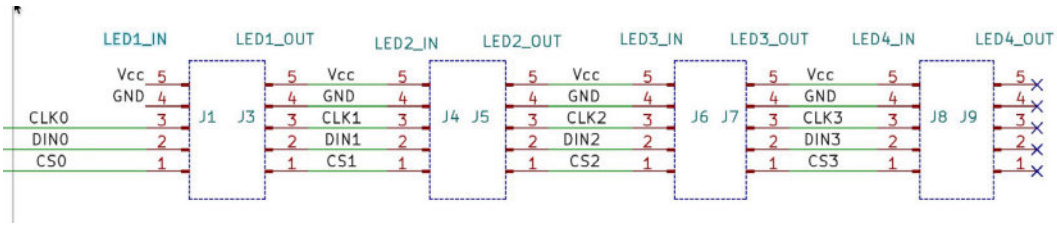


Figure 10.2.8.32: Pin header symbol names edited.

3. A text box with a reminder that specific headers have been flipped.

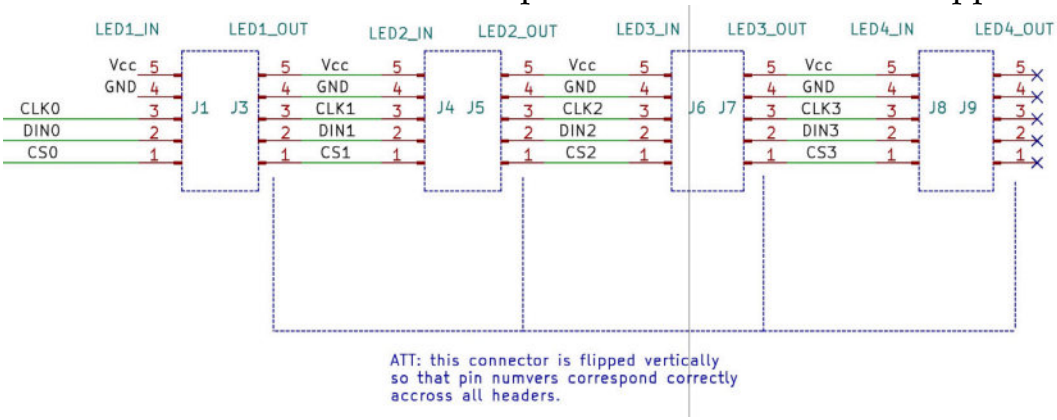


Figure 10.2.8.33: A reminder to self.

With the comments added, the sheet looks like this:

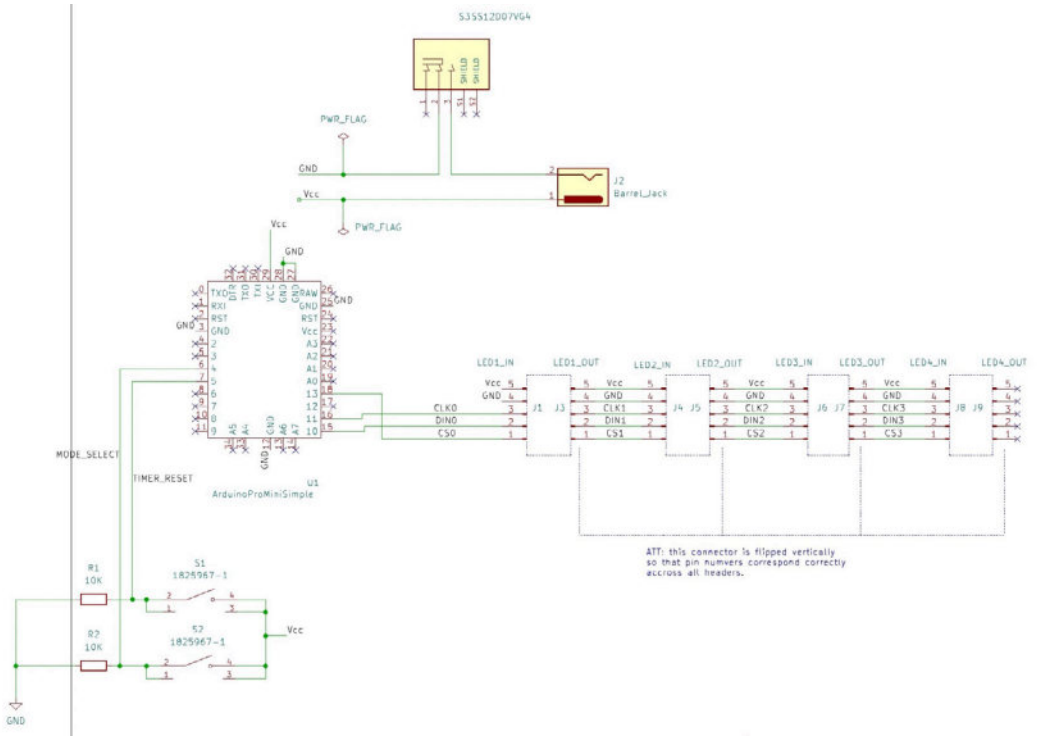


Figure 10.2.8.34: The schematic diagram with comments.

My original plan was to complete the schematic design workflow at this point and continue with the layout. However, I realized that there were a couple of last-minute edits I wanted to do. Let's look at those right away.

2.9. Last-minute edits

There's a couple of last-minute edits I'd like to make to the schematic before continuing with the layout. These are:

1. Add additional net labels that will be useful in the layout design.
2. Add symbols for the mounting holes and associate them with suitable footprints.

Begin with adding four mounting holes to the sheet. From the Symbol Choosers, look for the "MountingHole" symbol in the Mechanical library:

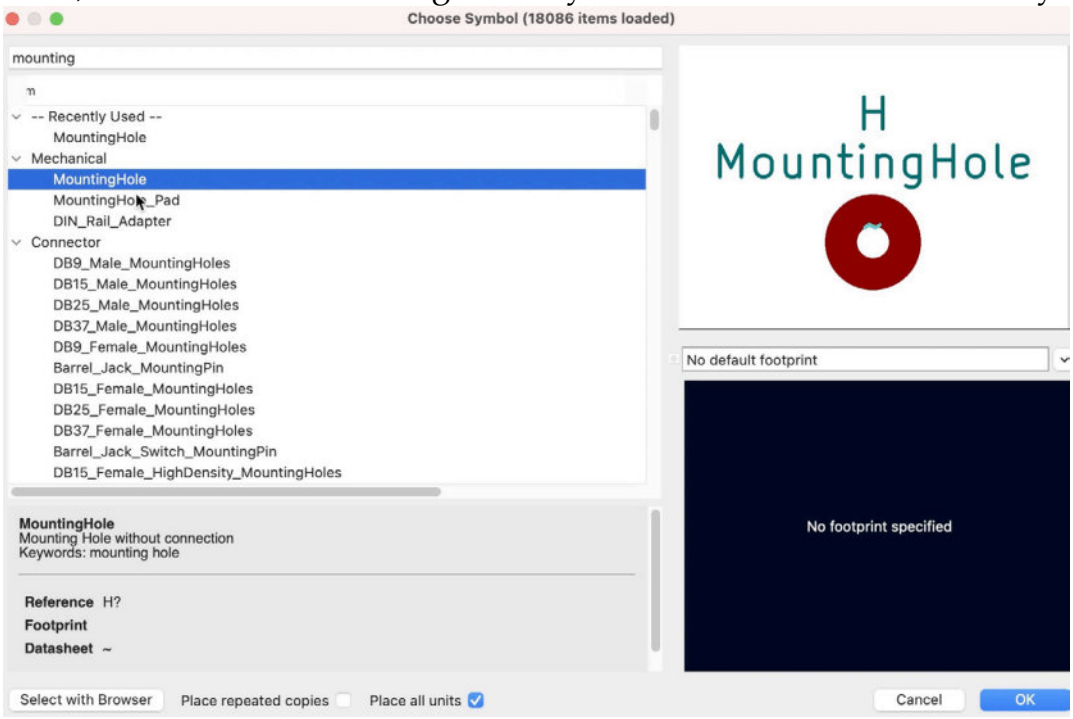


Figure 10.2.9.35: The MountingHole symbol.

Add the first mounting hole symbol to the sheet. Before duplicating it, double-click it to open its properties window and set a footprint. For the footprint, set it for the MountingHole_2.5mm option under the MountingHole library:

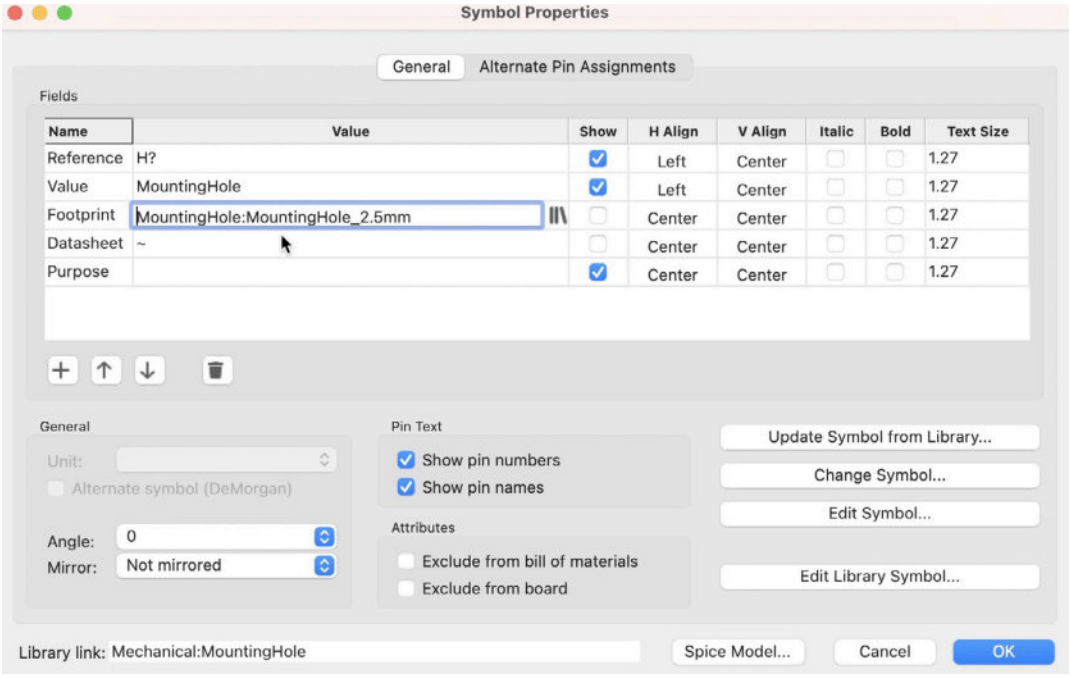


Figure 10.2.9.36: The mounting hole footprint.

Now, go ahead and create three duplicates of this symbol (they will all inherit the same footprint). Use the annotator to assign unique identifiers to the new symbols. Here is the result:

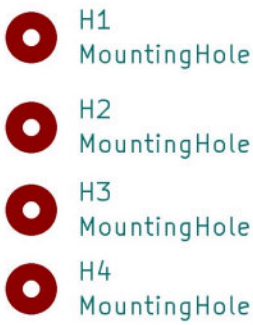


Figure 10.2.9.37: Four mounting hole footprints.

Next, work on editing and creating new net classes, and then assign net to the net classes:

- Power (set earlier), change its name to “GND.”
- Vcc
- Signal

Set the net class memberships:

- Vcc net belongs to the Vcc net class.
- GND net belongs to the GND net class.
- All other nets (like CLK0, CS1, etc.) below to the Signal net class.

See the net class table in the Schematic Setup window below:

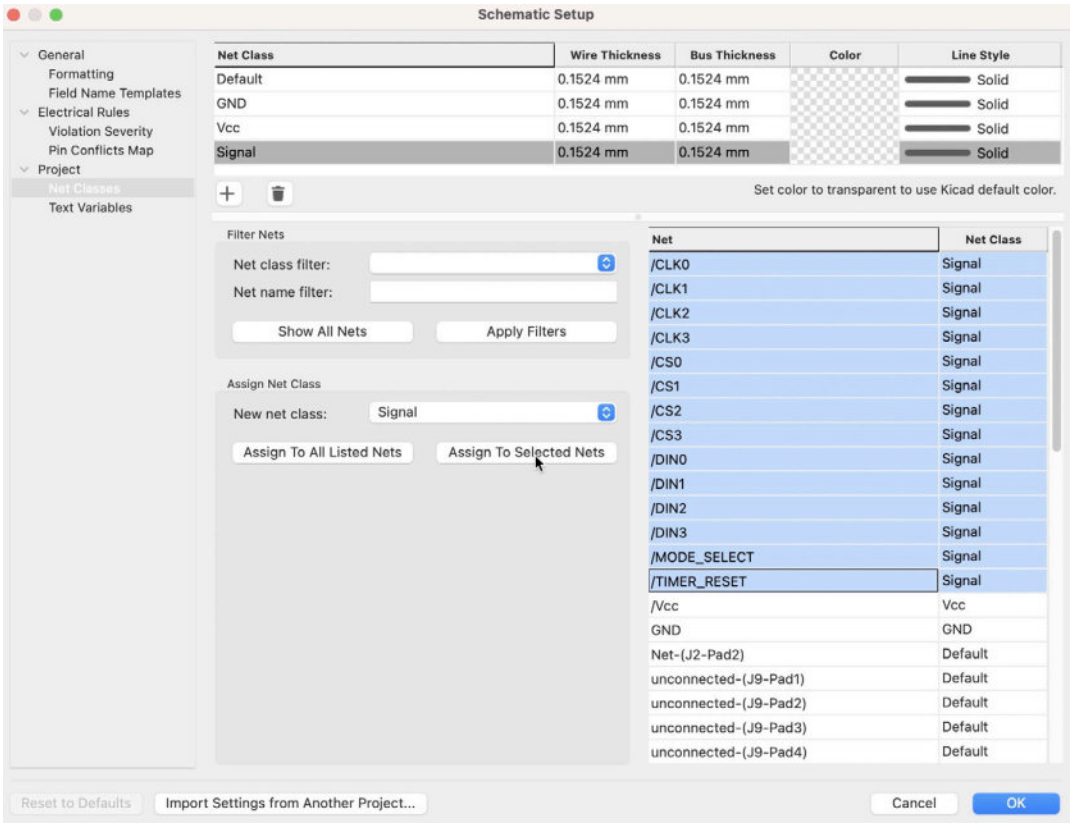


Figure 10.2.9.38: Nets and Net Classes.

This completes work in the schematic design workflow. Let's continue with the layout process in the next chapter.

3. Layout design editing

In the previous chapter, you completed the schematic design of the breadboard power supply PCB. In this chapter, you will work on the layout design following the layout design workflow from Part 6 of this book.

The main challenge in the layout design is the geometrical constraints set by the LED matrix display.

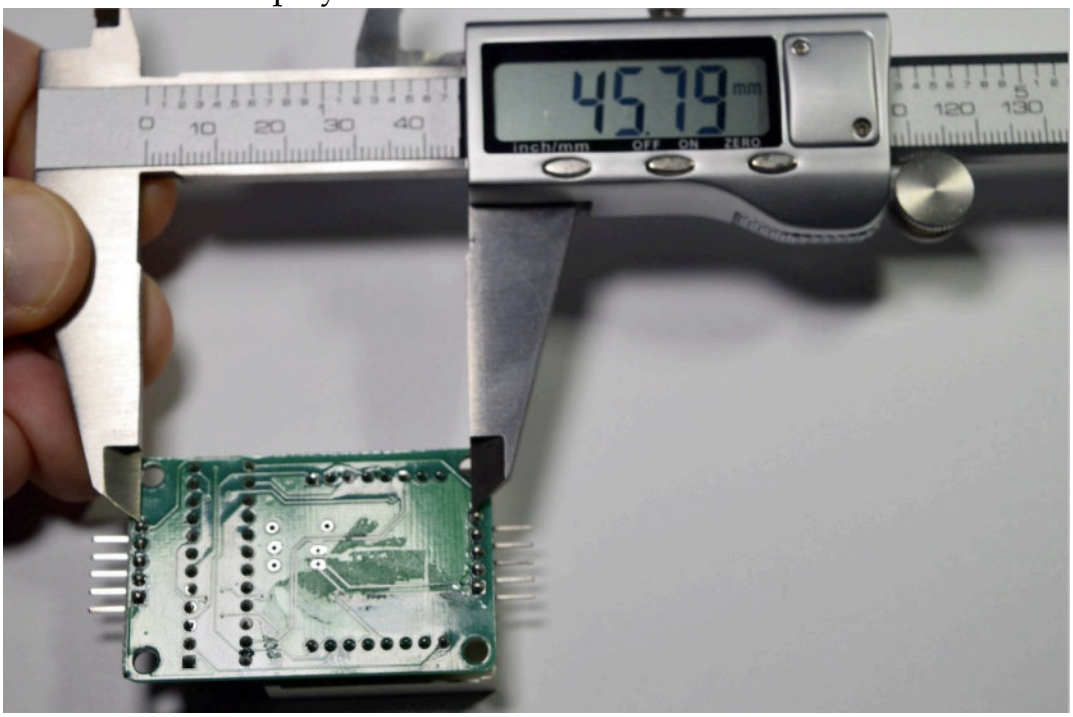


Figure 10.3.1: Measuring the LED matrix display module.

If measurements are not precise enough, it will not be possible to assemble the PCB. I will be taking multiple measurements to help me precisely arrange the LED matrix display pin headers on the PCB.

Let's begin with the setup step.

3.1. 1 - Setup

The schematic for this PCB is complete. In this segment, I will continue work in Pcbnew. In this segment of the chapter, I will set up the layout editor.

Start Pcbnew, and open the Board Setup window. Below I list the values for the most important settings. Assume that all other values are the defaults.

- Board Stackup.
  - Physical Stackup.

        - Copper layers: 2 (this is a simple board, so two layers suffice).
    - Board Editor Layers.
        - F.Cu: mixed (I will route signal and power tracks in this layer).
        - B.Cu: mixed (I will route signal and power tracks in this layer).
- Text & Graphics.
    - Text Variables: you will see the variable you set in the schematic design editor, named “design_version.”
- Design Rules.
    - Net Classes: you will see the net classes and memberships you set in the schematic editor. Set the following values in the net classes table:
        - Vcc track width: 0.3 mm.
        - Vcc Via size: 0.85 mm.
        - Vcc Via Hole size: 0.45 mm.
        - GND track width: 0.3 mm.
        - GND Via size: 0.85 mm.
        - GND Via Hole size: 0.45 mm.

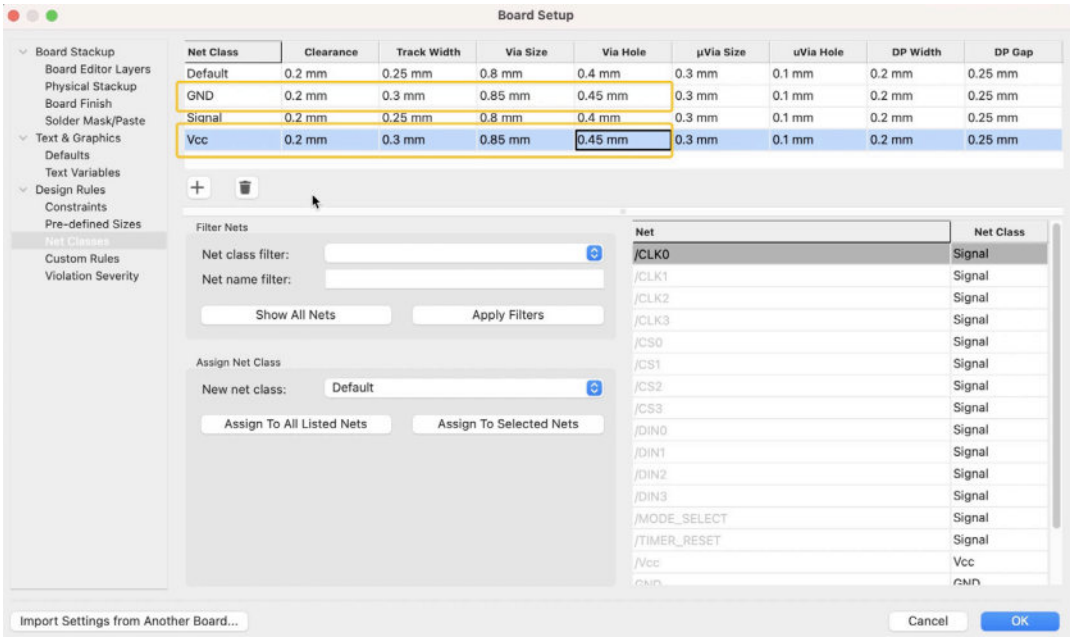
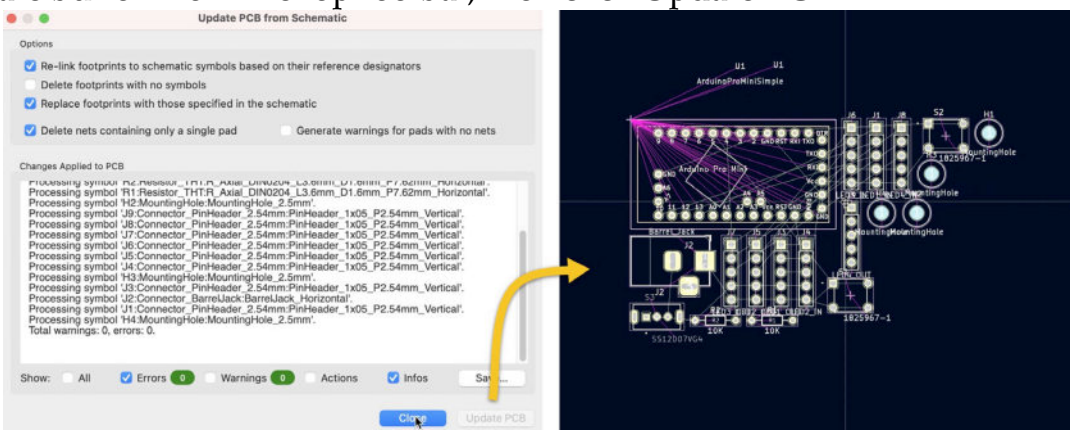


Figure 10.3.1.2: The changes in the Net Classes table.

Finish the setup by updating the PCB from the Schematic. Click on the update button from the top toolbar, then click Update PCB.



10.3.1.3: Updating the PCB from the schematic data.

With the layout editor updated with the latest schematic data, you can continue with step two of the layout process, where you will draw the rough outline of the PCB based on the board geometrical and user interface constraints.

3.2. 2 - Outline and constraints

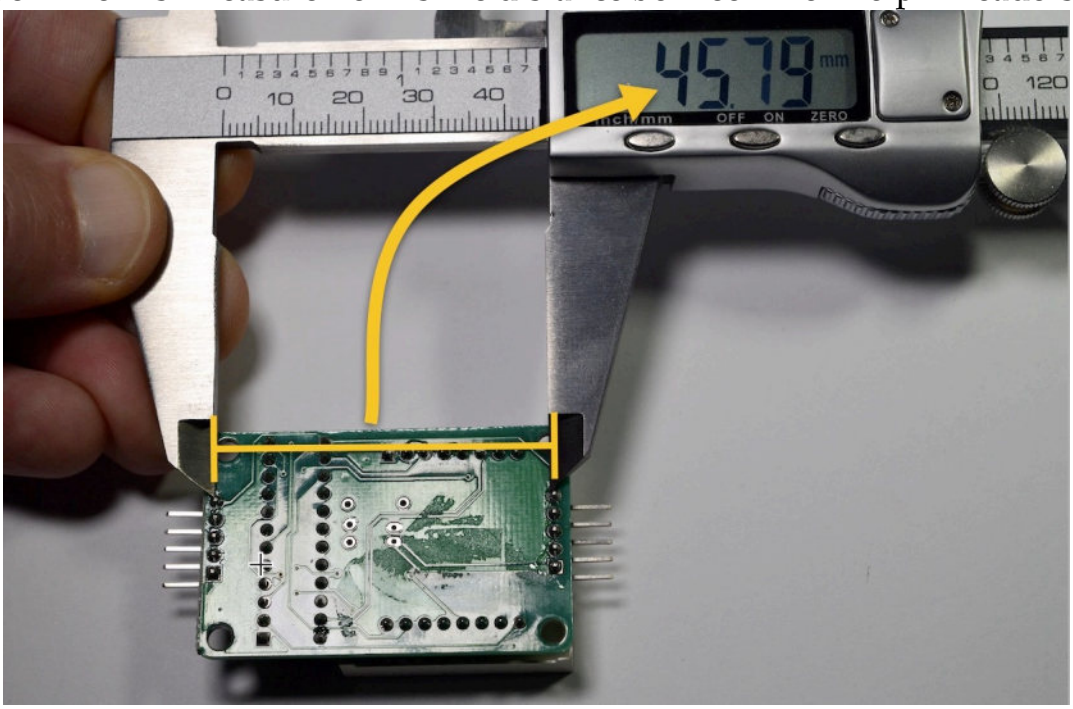
In this segment of the chapter, you will draw a rough outline for the PCB. To draw this outline, you will use measurements primarily from the LED matrix display module. You will also take into account the dimensions of the other footprints and the user interface requirements.

The user interface requirements relate to the position of the buttons that the end-user will be able to press to control the functions of the device and the barrel connector and power switch.

With the help of the rough outline of the PCB, you will be able to place the footprints inside the PCB in the next segment of this chapter (step three of the process). Following the placement, you will be able to refine the outline to its final shape.

Let's begin with the measurements. The final PCB will need to accommodate the four LED matrix displays in a tight arrangement so that the four combined displays look like a single unit. If the distance between the display pin headers is too small, even by a millimeter, the assembly will not be possible. If it is too large (even by a millimeter), the gaps between the individual displays will spoil the illusion of a single large display.

To take precise measurements of the LED matrix display module, use a caliper. The first measurement is the distance between the two pin headers:



10.3.2.4: The distance between the pin headers.

The caliper shows 45.79 mm for this distance.

You will also need the width and height of the LED matrix module:

10.3.2.5: The width (left) and height (right) of the LED matrix module.

The width of the LED module is 31.88 mm, and its height is 50.59 mm.

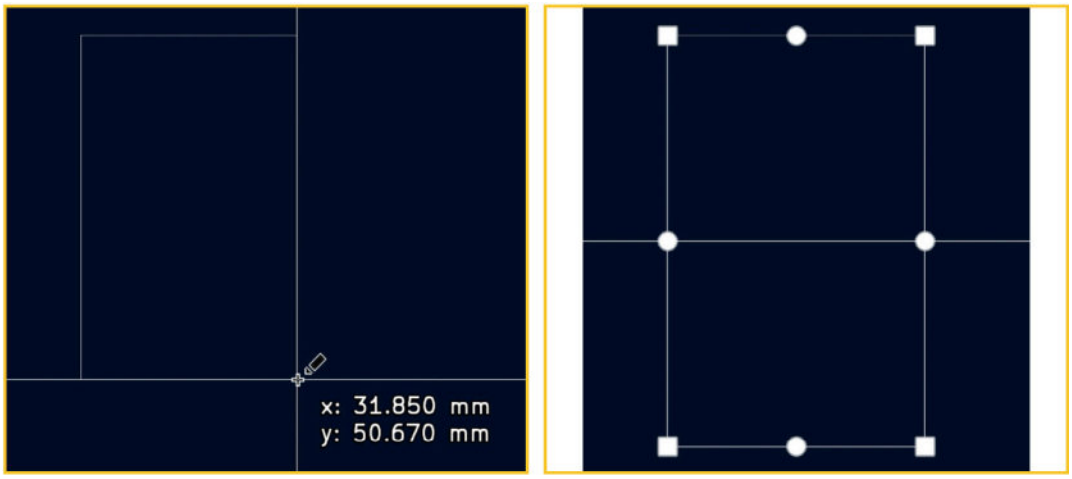
The width and height measurements are sufficient to help you draw the rough outline of the board. You will need further measurements to help with the placement of the footprints within this outline, but you will do them in step three of the workflow (in the next segment of this chapter).

Return to the layout editor, and switch the active layer to User.1. You can use this layer to draw a rectangular graphic element that represents the LED display module. Because you will be using User.1 layer, this graphic element

will not be included in the Gerber files. It will simply assist you in creating the rough outline (and then the final refined version of the outline) in the Edge.Cuts layer.

With User.1 layer active, use the rectangle graphics tool from the right toolbar to draw a rectangle as close as possible to 31.88 mm X 50.59 mm. Use a small grid size to achieve better accuracy. I used 0.01 mm for the grid size.

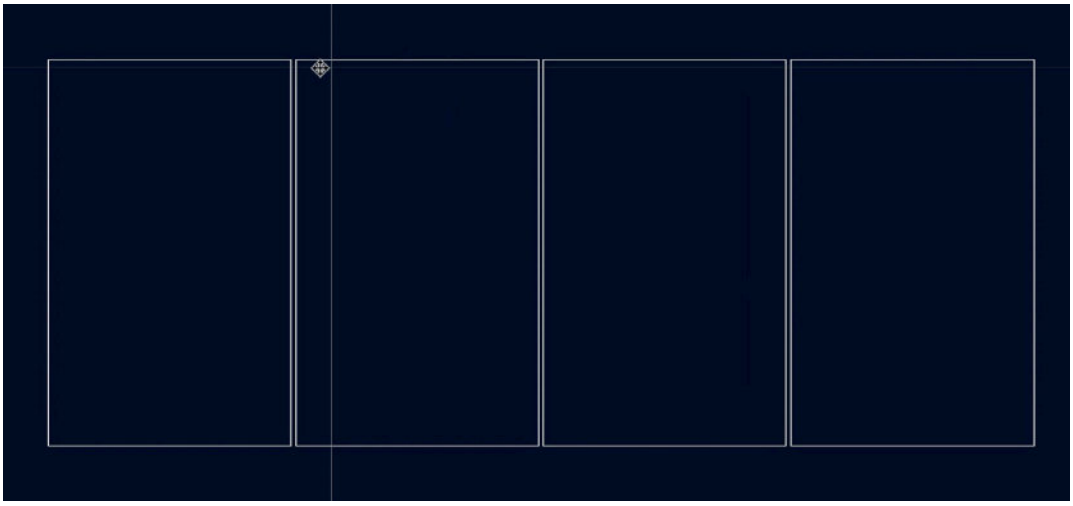
Start drawing the rectangle, and get the horizontal dimension as close as possible to 31.88 mm. In my example below, I started the drawing at point "1" and dragged the mouse pointer to the right and down. The objective is to get the "x" value as close as possible to 31.88 mm, and the "y" close to 50.59 mm. My "x" is at 31.850 mm, and my "y" is 50.670 mm, which is acceptable. I am confident that these sizes are correct because when I measured the width of the module with the caliper, I allowed for a small but real gap between the caliper arms and the module board.



10.3.2.6: Drawing a rectangle representing an LED array module (left) and completed (right).

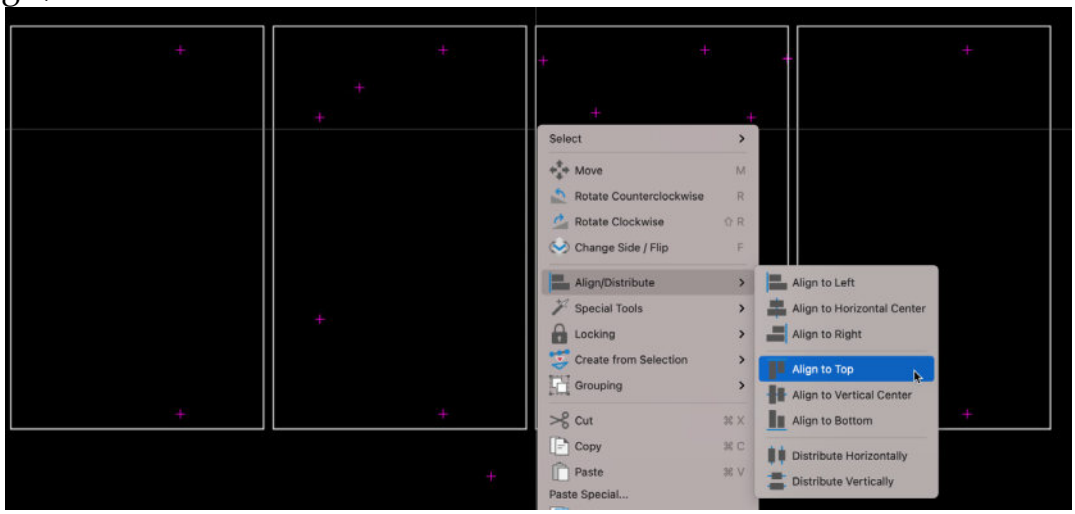
Now, you have a rectangle that represents the outline of a single LED matrix display module. The PCB will contain four of those modules. Create three copies of the rectangle, and place them next to each other. To duplicate the rectangle, select it, and use the Ctr-D/Cmd-D shortcut.

The result is below:



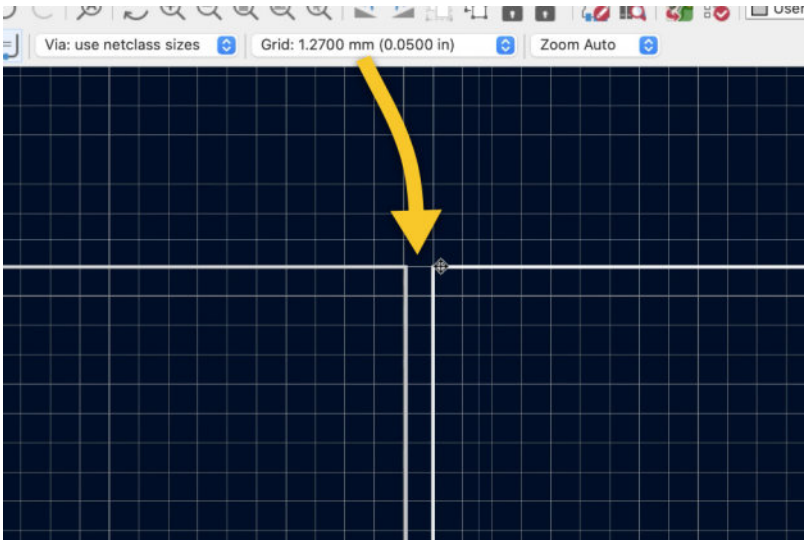
10.3.2.7: The four LED matrix displays.

Ensure that the four rectangles are equally spaced and justified. I have found that in the version of KiCad I used during the writing of this book, it is not possible to multiple-select graphical elements. This capability was added later. You should be able to select all rectangles using mouse click and drag, or hold down the shift key and click on each rectangle. Once all rectangles are selected, use the "align" and "distribute" functions under the context menu's "Align/Distribute" sub-menu.



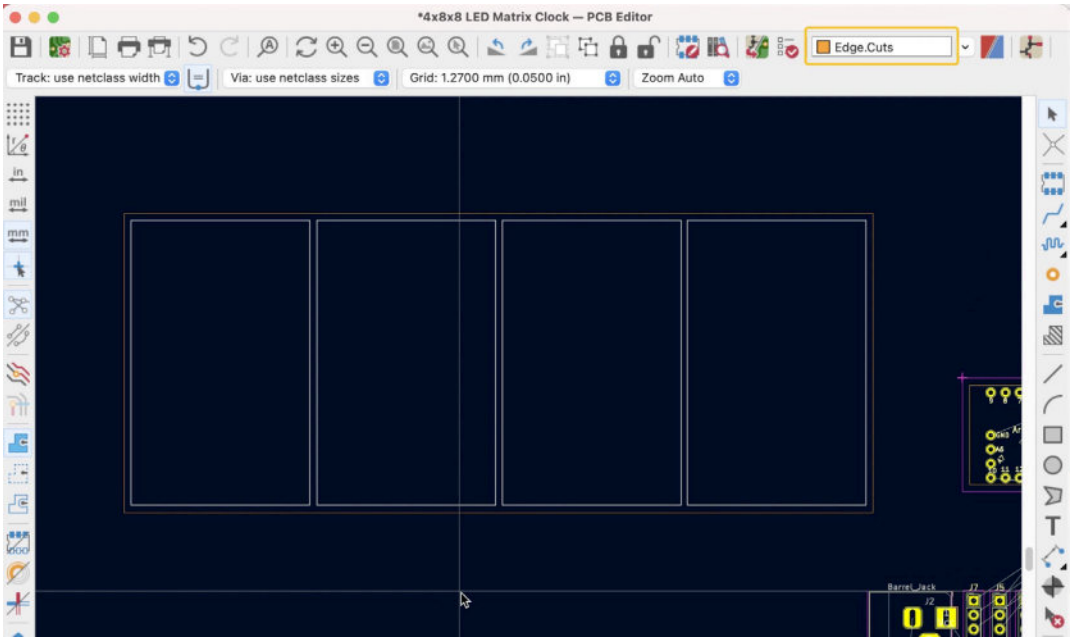
10.3.2.8: Align to top, and distribute evenly horizontally.

Another useful placement technique is to use the grid. Set the grid to a value that you wish to use for spacing the rectangles, and then zoom in so you can see the grid lines. Move the rectangles so that they are precisely one grid block apart, like this:



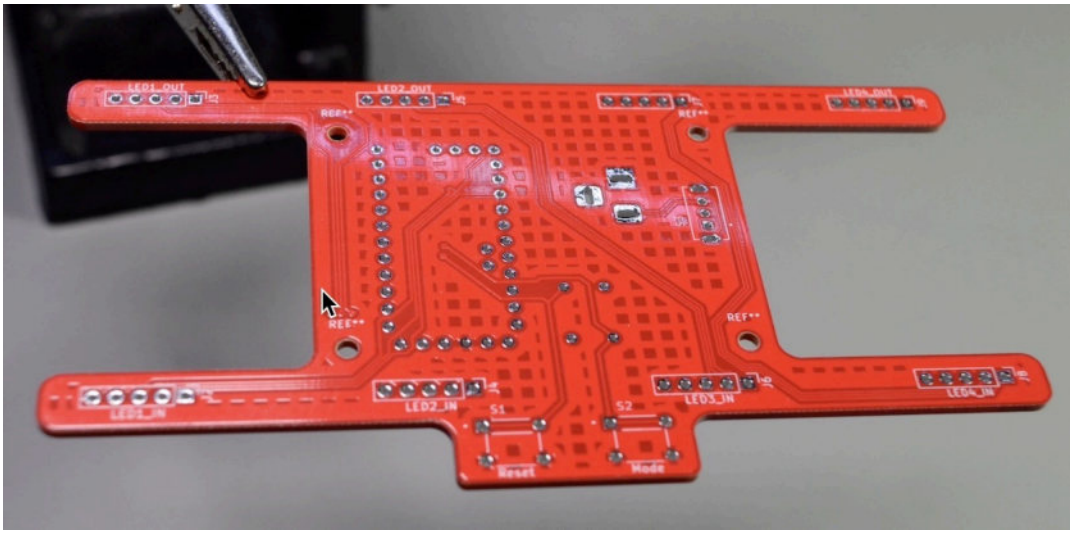
10.3.2.9: Using the grid to place the LED module rectangles.

With the LED matrix display module rectangles complete, you can draw the rough outline for the PCB. Switch the active layer to Edge.Cuts. Use the rectangle tool to draw a new graphic that encloses the four smaller rectangles. The result looks like this:



10.3.2.10: Drawing the rough outline in Edge.Cuts.

I will place the component footprints within this rough outline in step three of the workflow; however, I need more guidance. I remind you that my objective for the PCB shape is to have a center portion that contains the components and four arms that hold the first and fourth LED matrix module. I want to place all footprints in the center of the board to make it possible to cut out the substrate of the PCB on the left and right sides. See the final PCB below:

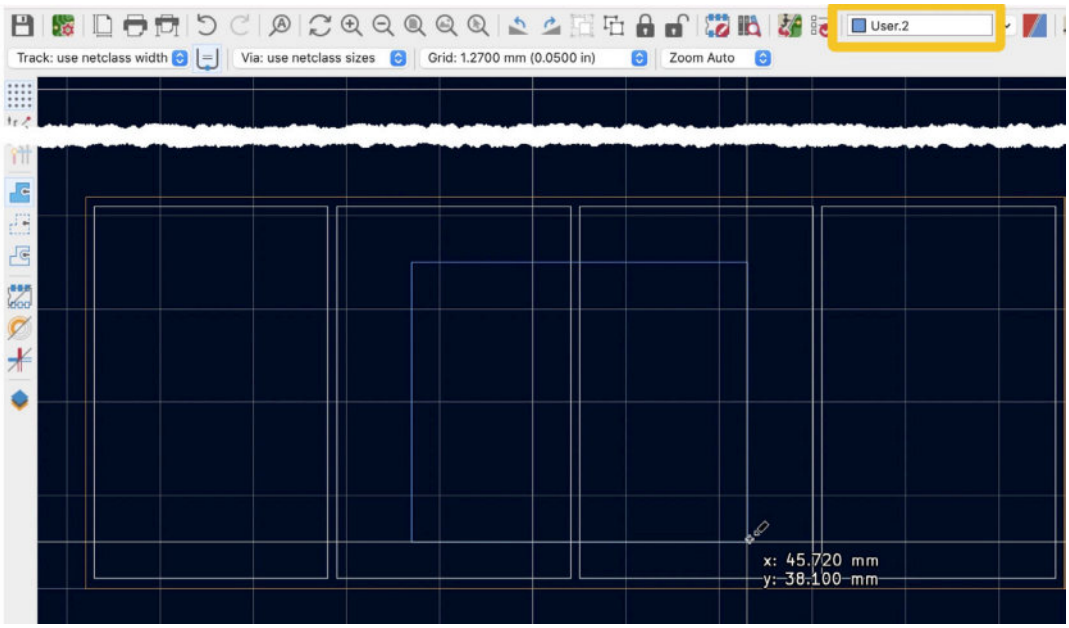


10.3.2.11: A reminder of what the final PCB will look like.

To help me with the footprint placement in the next step:

1. Switch to User.2 layer, and use the rectangle tool to draw a new rectangle in the middle of the rough outline.
2. Size this rectangle to be large enough to contain the footprints for the Arduino Pro Mini, the power barrel jack, the on-off switch, and four mounting holes.
3. Use the grid (I set mine to 1.27 mm) and the dx, dy values to help you draw.

Here is the result:



10.3.2.12: This rectangle will assist with the footprint placement.

The new rectangle will be helpful in the next step of the workflow. Even if it is too small or too large, it will still guide you towards placing the footprints

in the appropriate positions. Because it is in User.2 layer, you will be able to switch it off when you no longer need it.

I have not made a provision of the buttons; however, this is not a problem. I will place the buttons using the rough outline as a reference and then enclose them in the perimeter of the PCB when I do the refinement.

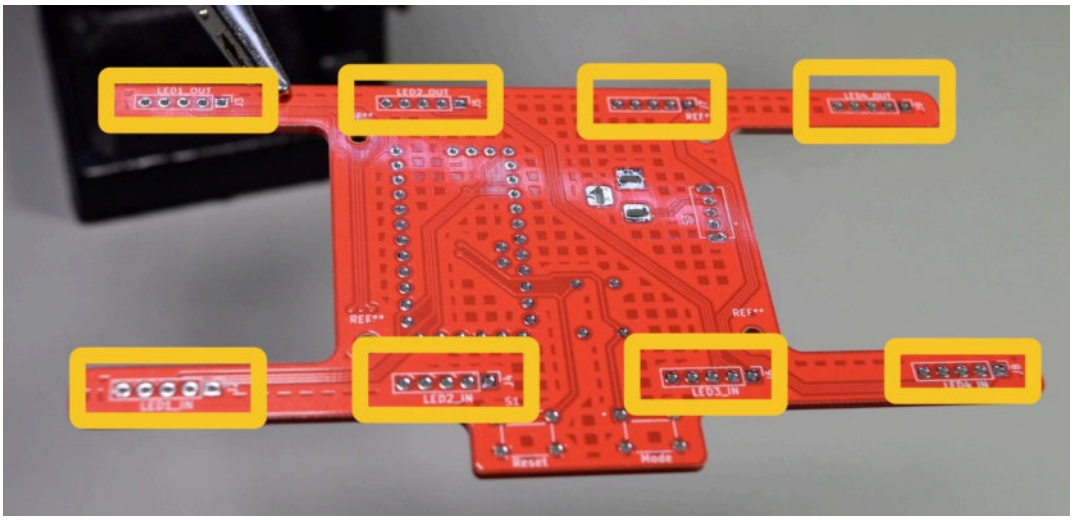
Let's continue with the placement of the component footprints in the next segment of this chapter.

3.3. 3 - Place components

In the previous segment of this chapter, you prepared the layout editor for the placement of the component footprints. You created a rough outline for the PCB and marked the positions of the LED matrix display modules.

In this segment, you will complete step three of the layout workflow and place the component footprints within the rough outline of the PCB.

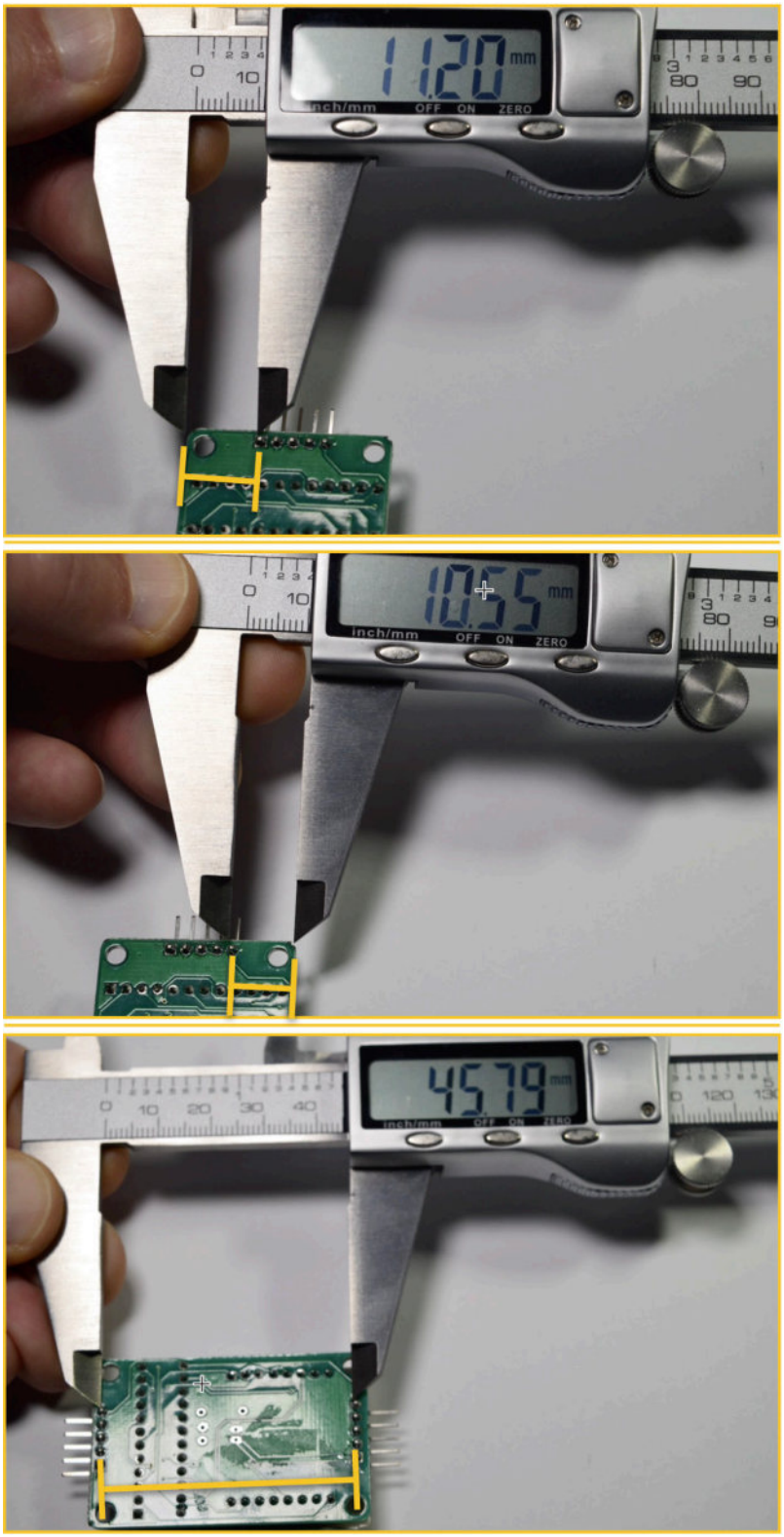
You will start the footprint placement with the footprints that define the most constrain-important elements of the PCB: the LED matrix display pin headers.



10.3.3.13: Starting with the LED matrix display pin headers.

To help accurately position these headers within the display rectangles in User.1 layer, you will need additional measurements using your caliper. Unfortunately, the pin headers are not placed symmetrically in the LED matrix display modules that I am using. The distances between the headers and the sides of the modules are different. This means that you will need to measure the distances between the header and the two sides of the PCB and between the two headers.

See my measurements below:



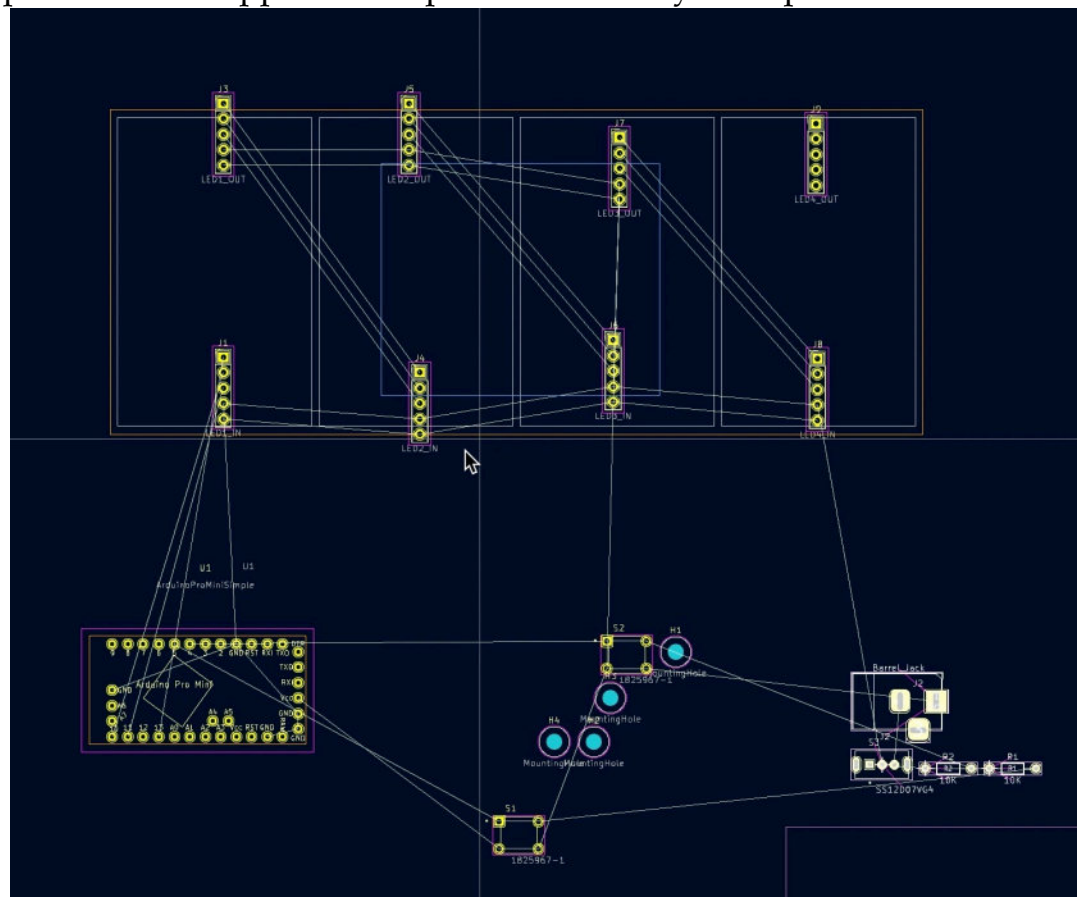
10.3.3.14: Pin header measurements.

The important pin header distances are:

- Left board edge to left-most pin: 11.20 mm.
- Right board edge to right-most pin: 10.55 mm.
- Distance between pin headers: 45.79 mm.

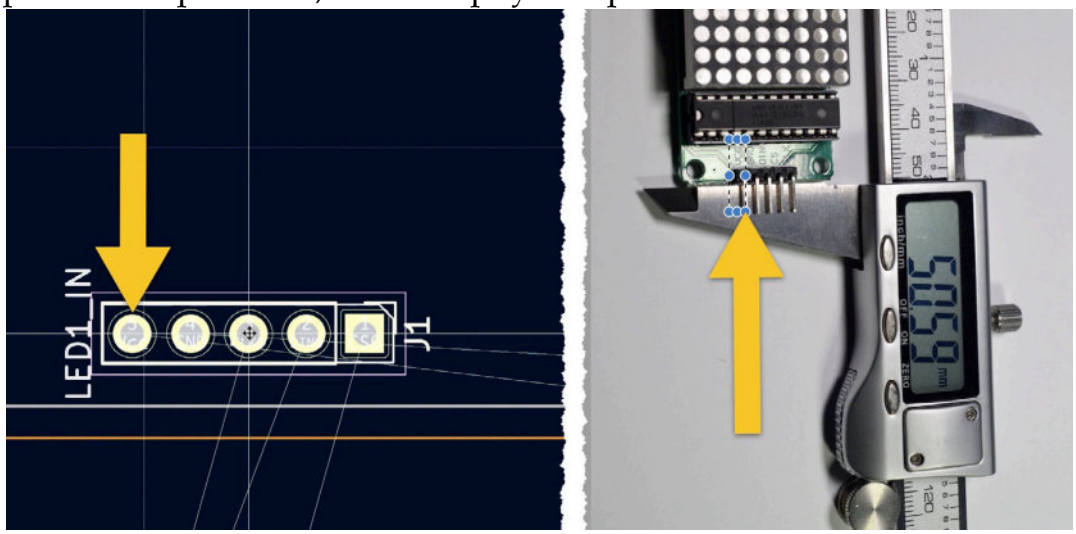
My working grid size is 0.254 mm.

Start by sorting out the various footprints. Move the LED display footprints to their approximate positions. See my example below:



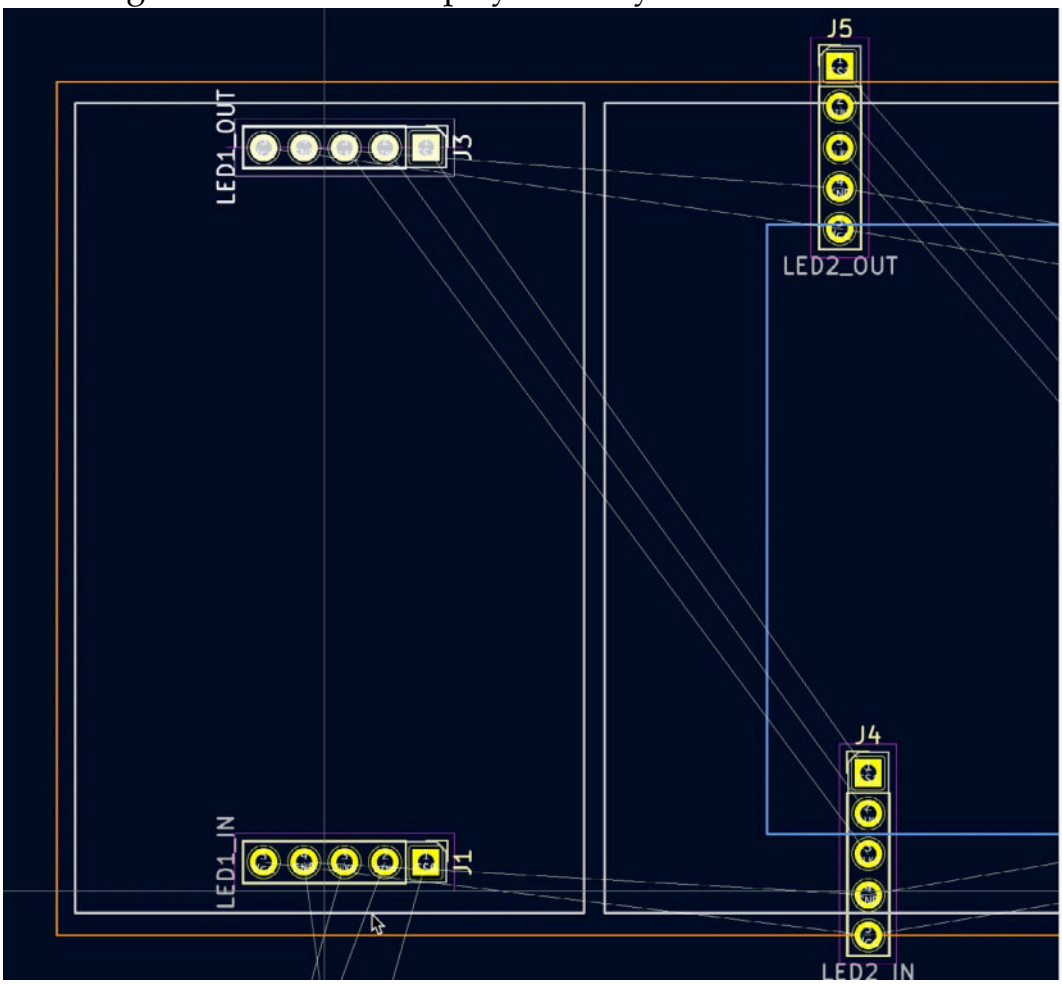
10.3.3.15: Unbundled footprints.

Orient the pin header footprints. Use the original part to find out the correct orientation. For example, for J1 (input header for the first display), the Vcc pin should point left, as in the physical part:



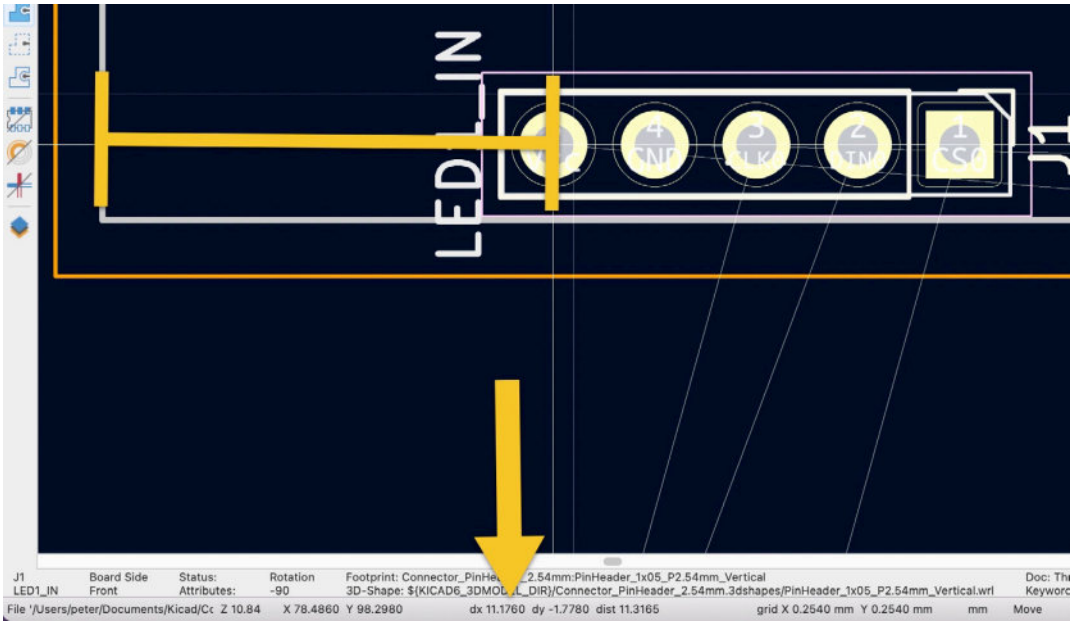
10.3.3.16: Pin header orientation.

The orientation of J3 (the output header for the same display) should have the same orientation, with the Vcc pin towards the left. Below are J1 and J3 that belong to the first LED display correctly oriented:



10.3.3.17: J1 and J3 oriented.

With J1 and J2 correctly oriented, the next task is to set them in the correct position using the measurements at the segment's start. Use the dx and dy values to measure the center of pins five and one of J1 against the bottom, right, and left edges of the left-most display module rectangle. Below, you can see the positioning of J1 against the bottom and left edge of the rectangle:



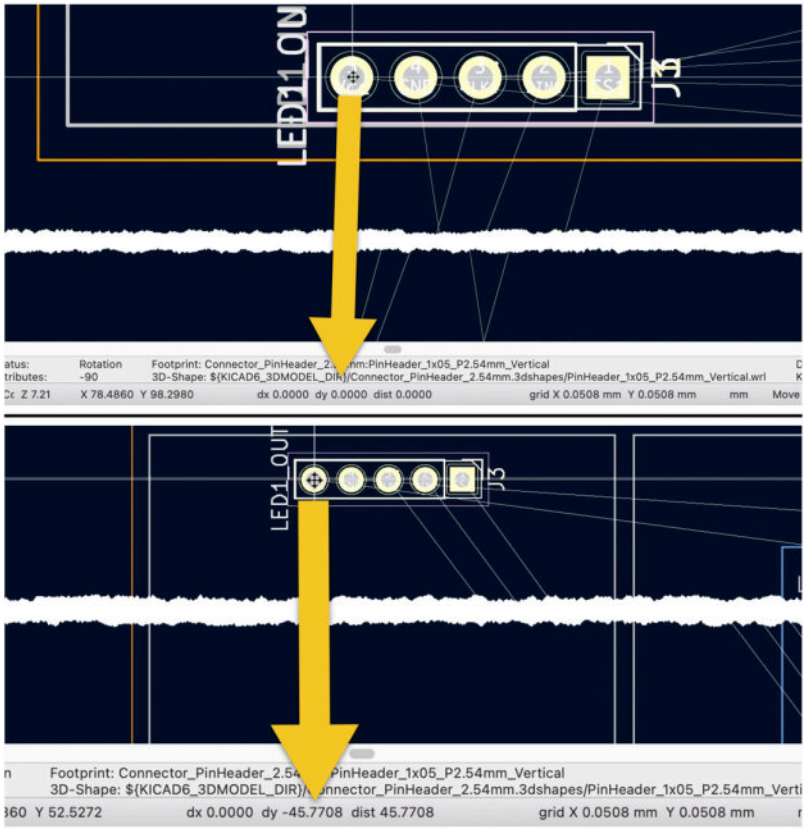
10.3.3.18: J1 pin five is 11.1760 mm from the left edge.

For J1, pin five is 11.1760 mm from the left edge. This is close enough compared to the measurement of 11.20 mm. The distance from J1 pin 5 to the bottom edge is not critical. What is critical is to ensure the correct distance between J1 and J3. In my instance, I placed J1's footprint bottom borderline (courtyard) just above the white module outline in User.2 layer that I drew in the previous step. You can choose to use a different method. For example you can opt to center headers J1 and J3 inside the module outline. To do this, subtract the distance between J1 and J3 from the module outline, and then half it (50.59-45.79=4.8 / 2 = 2.4mm). You can then distance J1 around 2mm to 3mm from the bottom edge.

Continue with J3, which is the output header for the same display module at the top of the rectangle. Two considerations dictate the placement of J3:

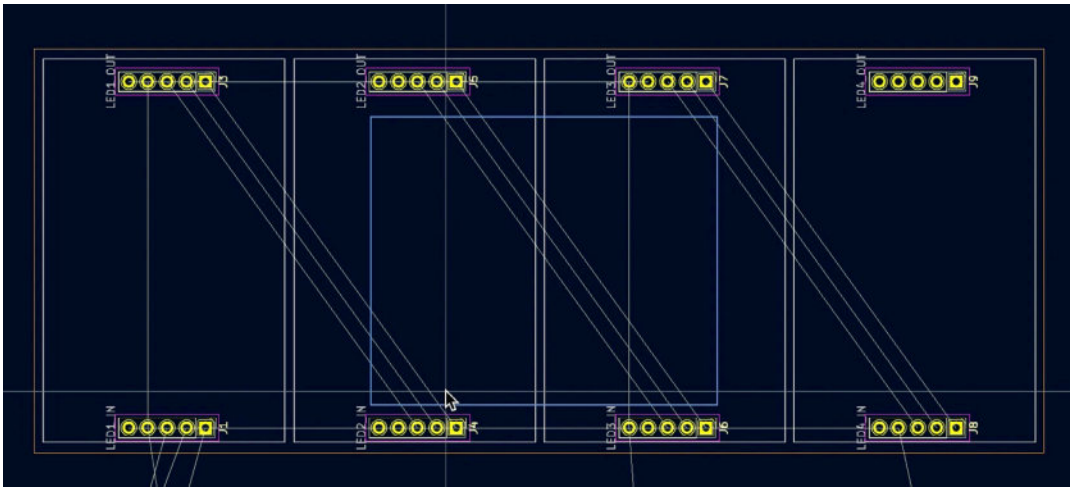
1. The distance between J3 pin five and the left edge of the rectangle. This distance must match that same distance for J1.
2. The distance between J1 and J3 much be equal to or close to 45.79 mm.

Start with the second consideration. Select J3 and move it on top of J1. Press the space bar to reset the dx and dy counters, and then move J3 upwards. You want to maintain dx to zero and set dy close to 45.79 mm. I am using a grid size of 0.0508 mm. With dx equalling zero, you know that J3 is the same distance from the left edge as J1.



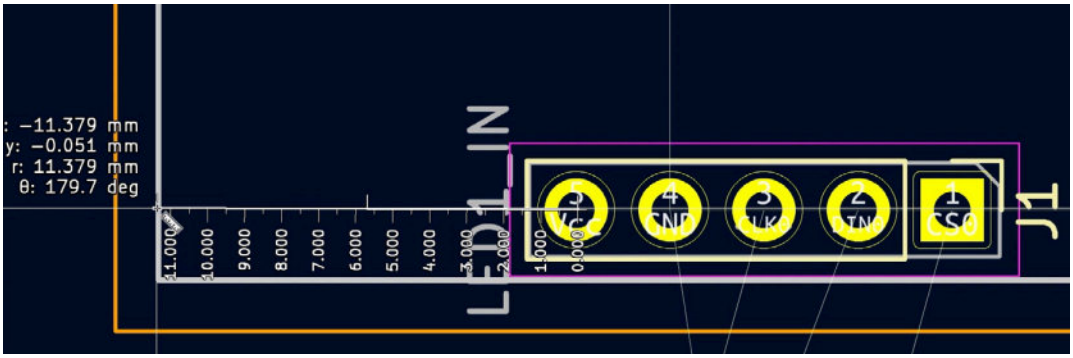
10.3.3.19: Placing J3 in relation to J1.

Now that you have positioned J1 and J3 lock them in place to avoid accidental movement. Follow the same process to place the remaining display pin headers. The results are below:



10.3.3.20: Display pin headers are now in place.

Before continuing, I will do one last set of measurements to ensure I have not made any errors with the placement. For this round of measurements, I will use the interactive ruler from the right tool bar:



10.3.3.21: Sanity distance measurements using the interactive ruler.

With these measurements, I confirm that the placements are correct to continue with the placement of the remaining footprints. Try to make everything fit within the blue rectangle in the User.2 layer. Below is the order by which I continued, with comments where necessary:

1. The Arduino Pro Mini as it is the largest component and will dictate the positions of other components around it.
2. The barrel jack. I placed oriented so that the power cable can plug in upwards. I have placed it with plenty of clearance from below to accommodate for rigid power cables.
3. The slide switch (on/off) next to the barrel connector.
4. The resistors.
5. The buttons, in between pin headers J4 and J6 and below the bottom edge of the board outline. You will enclose them within the board outline when you refine the outline in the next segment.
6. The mounting holes.

Use the grid and the object alignment tools to justify the buttons horizontally and ensure that they are symmetrically around the center axis of the board.

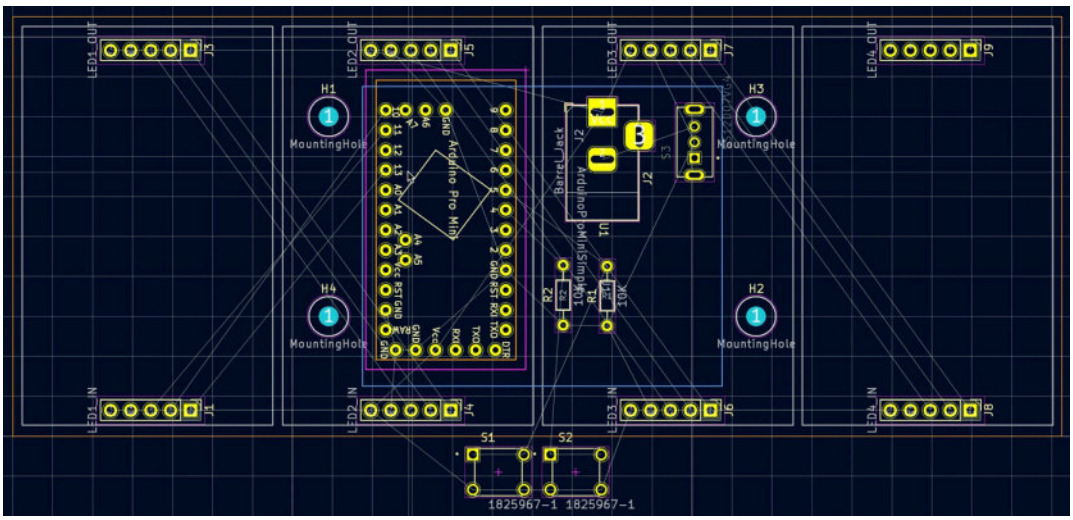
Do the same for the mounting holes. You want the holes to be symmetrical against the center axis of the board and justified horizontally and vertically to design an enclosure potentially. This will be easier if the hole positions are regular rather than irregular.

Another consideration for the buttons is that given their size, their position will affect the routing of the copper tracks. Don't place them too close to the Arduino Pro Mini footprint because it will make it more difficult to route tracks to and from the Arduino.

For the buttons, justification is essential to ensure that your board does not end up with miss-aligned buttons.

When you have finished with the placement, lock all footprints in place.

Below you can see how I placed the footprints in my instance of the board:

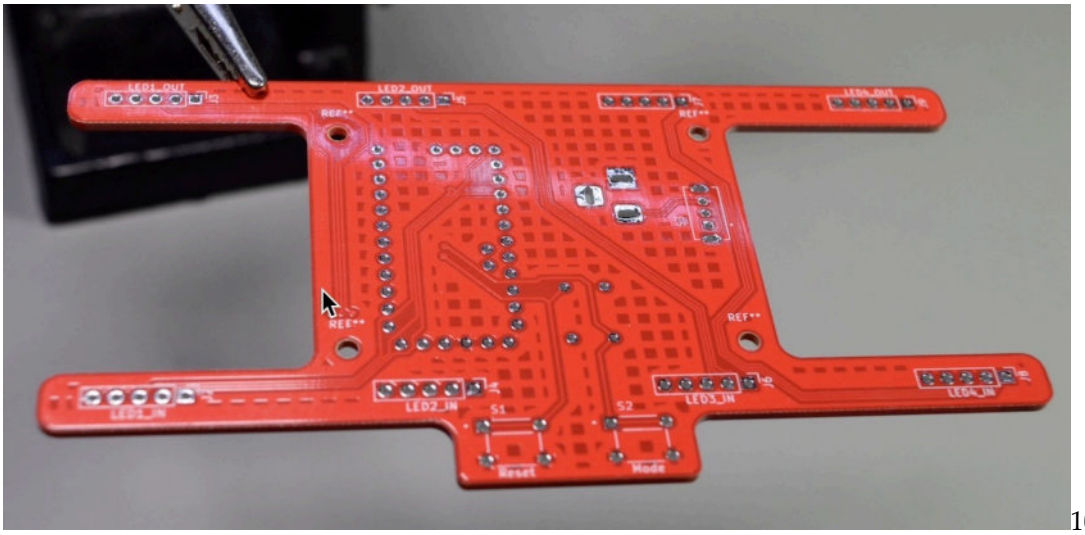


10.3.3.22: Completed positioning of footprints.

With the positioning of the footprints complete, let's refine the outline of the PCB.

3.4. 2 - Refine outline

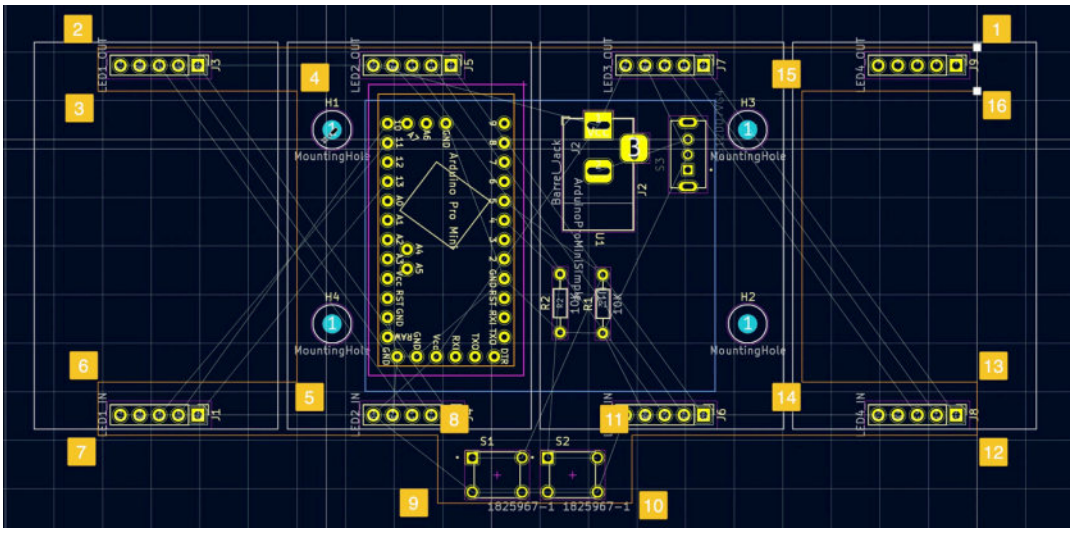
Let’s complete the outline of the board. Your objective is to design an outline in the Edge.Cuts layer so that the outcome produces the result you can see below:



10.3.4.23: Draw this shape in the Edge.Cuts layer.

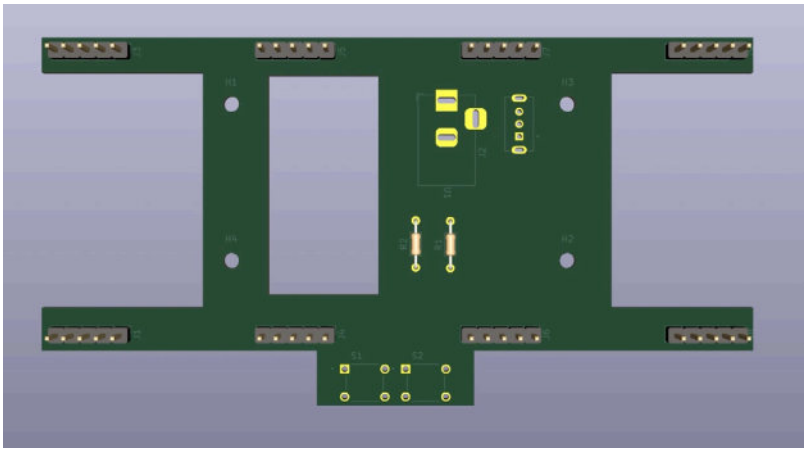
You no longer need the rectangle you draw in User.1 layer. You can either delete it or make it invisible. Either way, the graphics in the User.2 layer and the footprints provide enough guidance for drawing the refined outlines in Edge.Cuts.

Activate the Edge.Cuts layer, select the line tool from the right toolbar and start drawing from the top right corner of the board. My grid size is 0.6450 mm. Work your way around the display module pin headers and the pushbuttons. Use the full-screen crosshair cursor and the dy/dy values to help you draw with symmetry. Below you can see the result of this work. The numbers mark the positions and sequence of the clicks during my drawing session.



10.3.4.24: Outline with 90-degree corners.

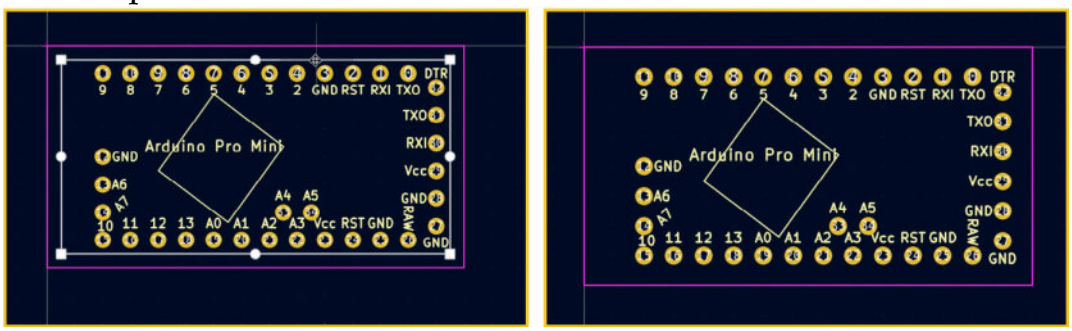
At this point, you have a board outline in the Edge.Cuts layer. It contains all footprints and has 90-degree angles. You can use the 3D viewer to see it in 3D.



10.3.4.25: Board in 3D. There is a hole where the Arduino footprint should be.

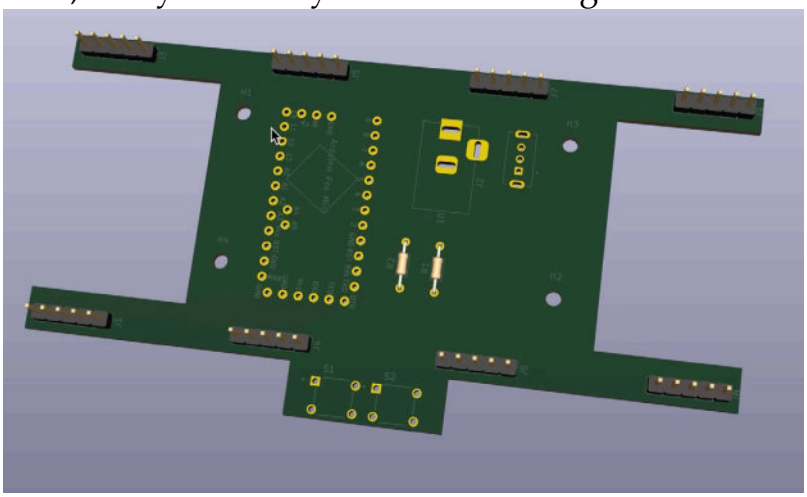
The 3D viewer shows a problem with the Arduino footprint. Instead of the Arduino Pro Mini headers, the 3D viewer shows a rectangle opening. This is because the Arduino Pro Mini footprint I designed contains a rectangle in the Edge.Cuts layer. In orange, you can see this rectangle enclosing the Arduino module's pin headers in 10.3.4.24. I will need to fix this before I continue.

Right-click on the Arduino Pro Mini footprint and select “Open in Footprint Editor.” Select the rectangle in the Edge.Cuts layer, delete it and save the footprint.



10.3.4.26: Arduino Pro Mini footprint with Edge.Cuts outline (left), and without (right).

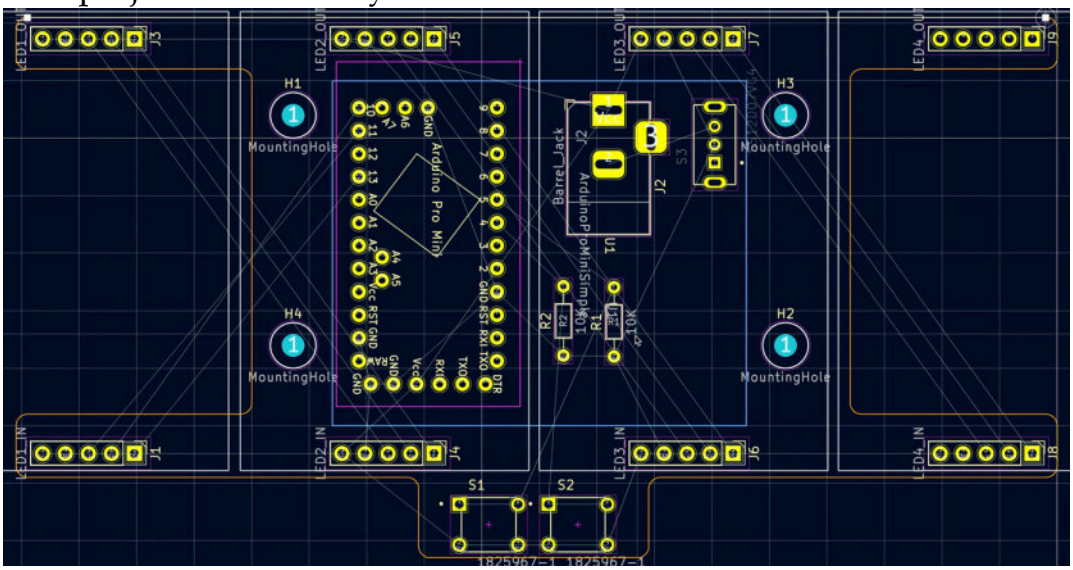
Close the footprint editor and return to the layout editor. The PCB layout is updated, and you can try the 3D viewer again.



10.3.4.27: The Arduino Pro Mini footprint correctly rendered in 3D.

The Arduino Pro Mini footprint is now rendered correctly in the 3D viewer.

Continue to round the corners of the layout as you have done in the previous project. Below is my final refined PCB outline:



10.3.4.28: The final PCB outline with rounded corners.

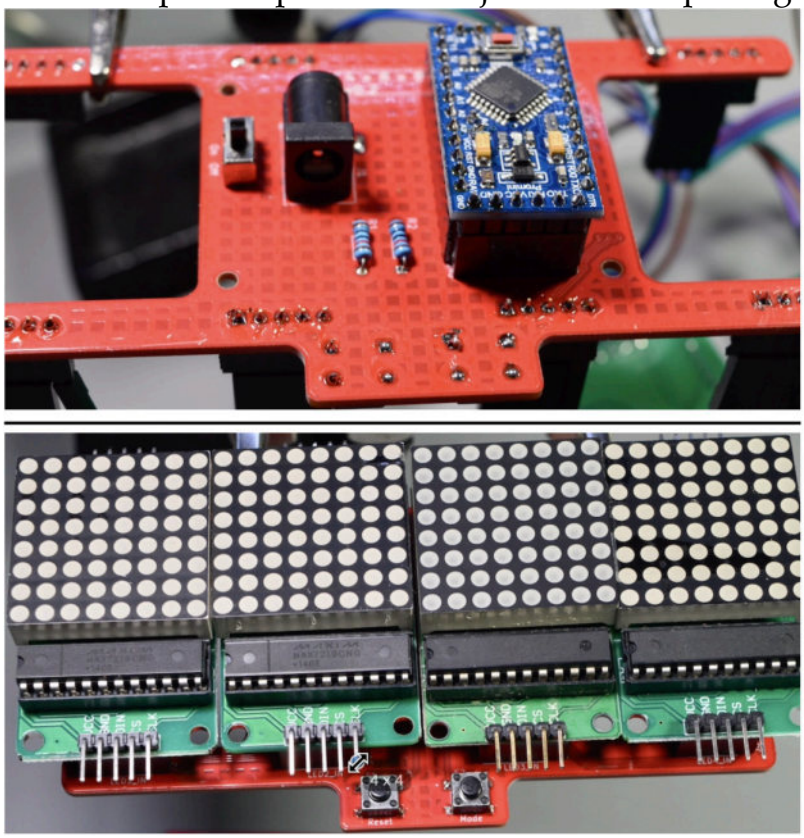
Now that you have refined and completed the board outline, you are almost ready to work on the board copper routes. However, there are first a few changes that I would like to make relating to the footprints. At the moment, all footprints are placed on the top copper layer, as this is the default. I want to change the position of the Arduino Pro Mini, barrel connector, switch, and resistors to the back copper layer. This will leave the top copper layer for the displays and buttons.

Let's refine the footprint positions next.

3.5. 3 - Move footprints

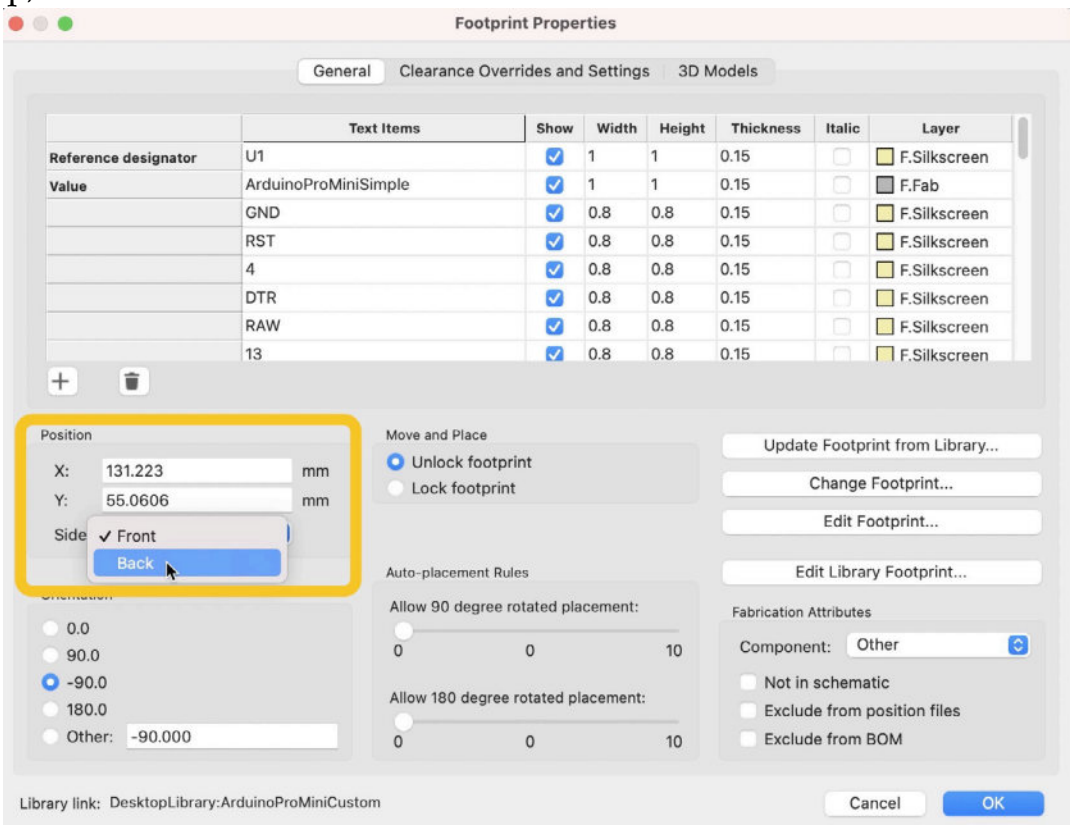
For this PCB, I'd like to place all footprints other than the LED displays and the buttons to the back copper layer. This will allow me to assemble these components out of view of the user and retain the front layer for the user interface components.

You can see the component placement objective in the photographs below:



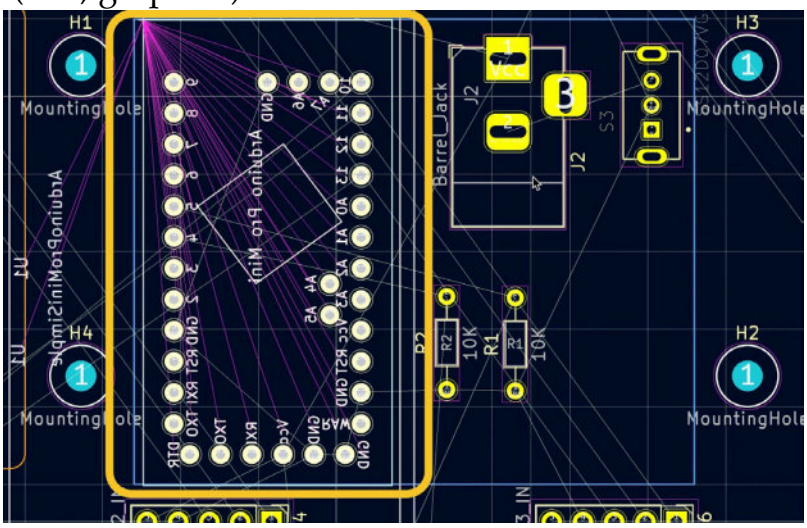
10.3.5.29: The placement of components in the back (top) and front (bottom) copper layers.

To select a placement layer for a footprint, bring up the footprint's properties window (double-click on the footprint), and in the "Position" group, set Side to "Back."



10.3.5.30: Setting the position of a footprint to “Back.”

Click OK. The editor will flip the footprint over to the back copper layer and may also move. Move the footprint back to its correct position. In the example below, I have moved the Arduino Pro Mini footprint to the back copper layer and corrected its position. Notice that the elements that make up this footprint (text, graphics) now look mirrored.

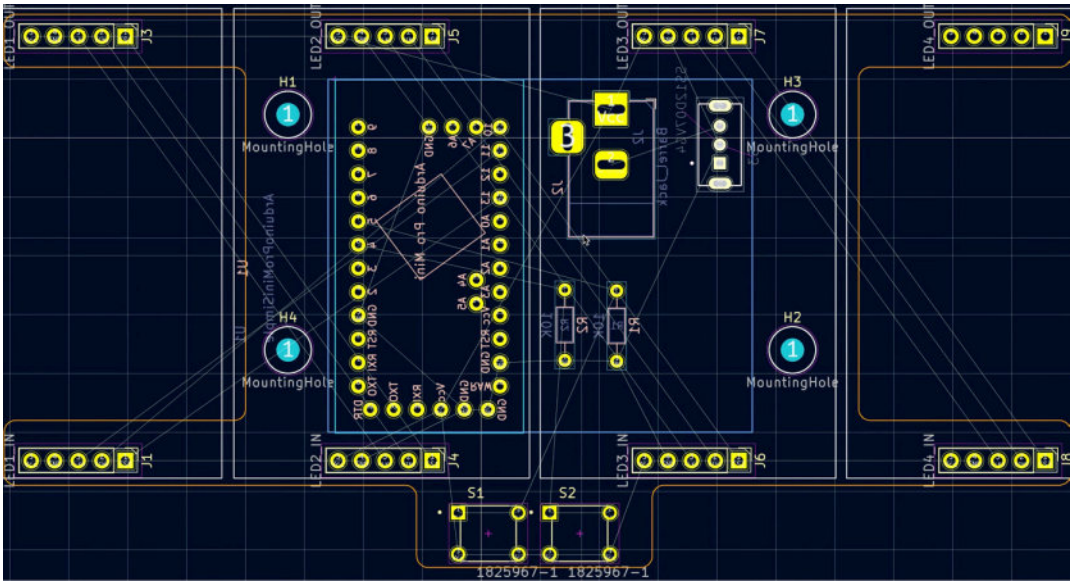


10.3.5.31: The Arduino Pro Mini footprint is now in the back copper layer.

Continue to set the position of the following footprints to "Back":

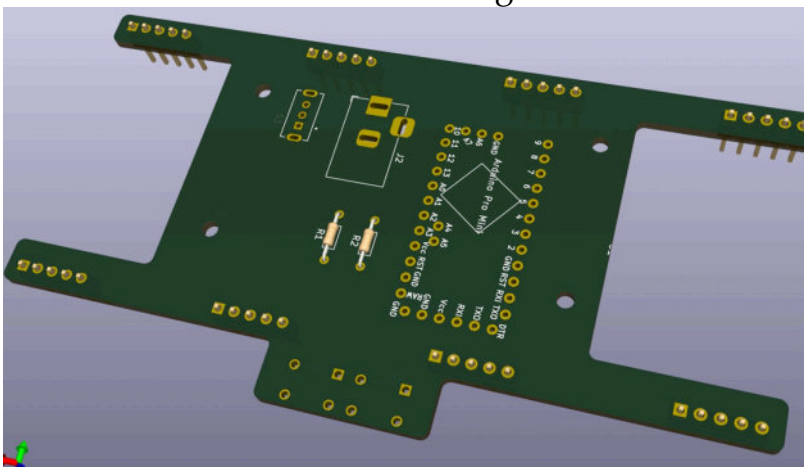
1. Arduino Pro Mini, U1.
2. Barrel connector, J2.
3. On/Off switch S3,
4. Resistors R1, E2

After this work, the board will look like this:



10.3.5.32: All listed footprints are now in the back copper layer.

Use the 3D viewer to see a 3D rendering of the back of the board:



10.3.5.33: 3D rendering of the back of the board.

The 3D viewer shows the footprints of the listed components in the back of the board.

With the work on the board outline and the footprint placement complete, you can now continue with the routing.

3.6. 4 - Route

With the board outline and footprint placement complete, the next step in the layout workflow is the routing of the copper tracks. You are working on a two-layer board.

The back copper layer is where you will create the majority of the connections between the GND pads. Instead of drawing copper tracks, you will draw a single copper fill connected to the GND net.

Use the front copper layer for all other connections. If it is not possible to complete a copper route in the front copper layer, you can use vias to switch to the bottom to complete the routing.

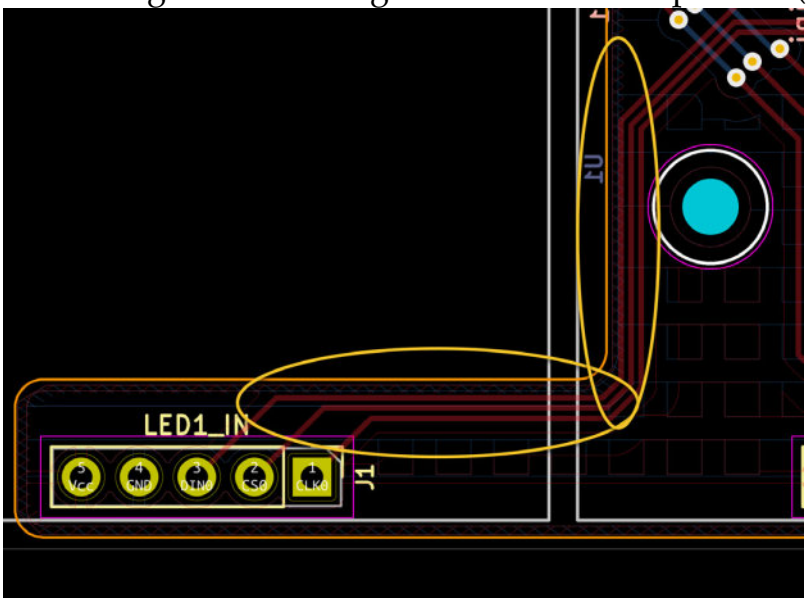
In the front copper layer, you will use a copper fill connected to the Vcc net to connect Vcc pads.

Switch to the front copper layer, and begin the routing process.

I started my routing session from the J1 and J3 pin headers and continued with the rest of the pin headers, followed by the components in the back copper layer. I set the interactive router to the "walk-around" mode.

Here are some guidelines to help you with the routing process:

1. Don't route the GND pads. You will route GND pads using the copper fill method also used in the previous project.
2. Don't route the Vcc pads. You will route Vcc pads using a copper fill in the front copper layer, connected to the Vcc net.
3. For the routes that originate in J1, J3, and J8, be careful to draw them close to the edge of their respective PCB arm and the routing hole. This leaves ample space for routing the remaining tracks later in the process (see below).



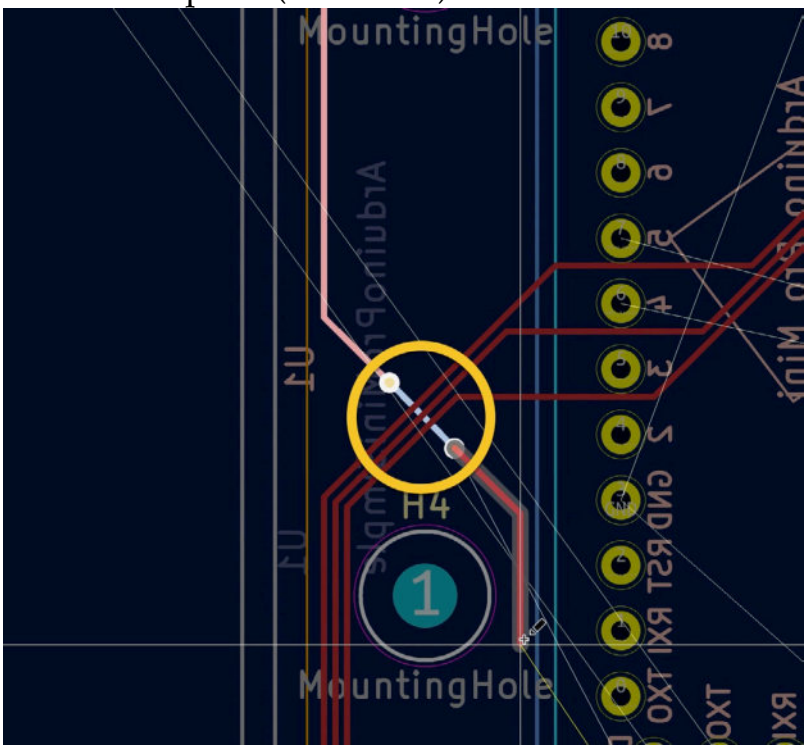
10.3.6.34: Drawing the J1 routes close to the edges of the board.

4. As much as possible, don't leave spaces between tracks. Use the Move command (click track to select and type "D" to move) to move tracks

and reduce gaps to the minimum track spacing allowed by the DRC. Large gaps between tracks waste space and make it harder to draw new routes.

5. Opt for shorter tracks whenever possible.

6. If you have to choose between a long or shorter track that requires vias, opt for the second option (see below).

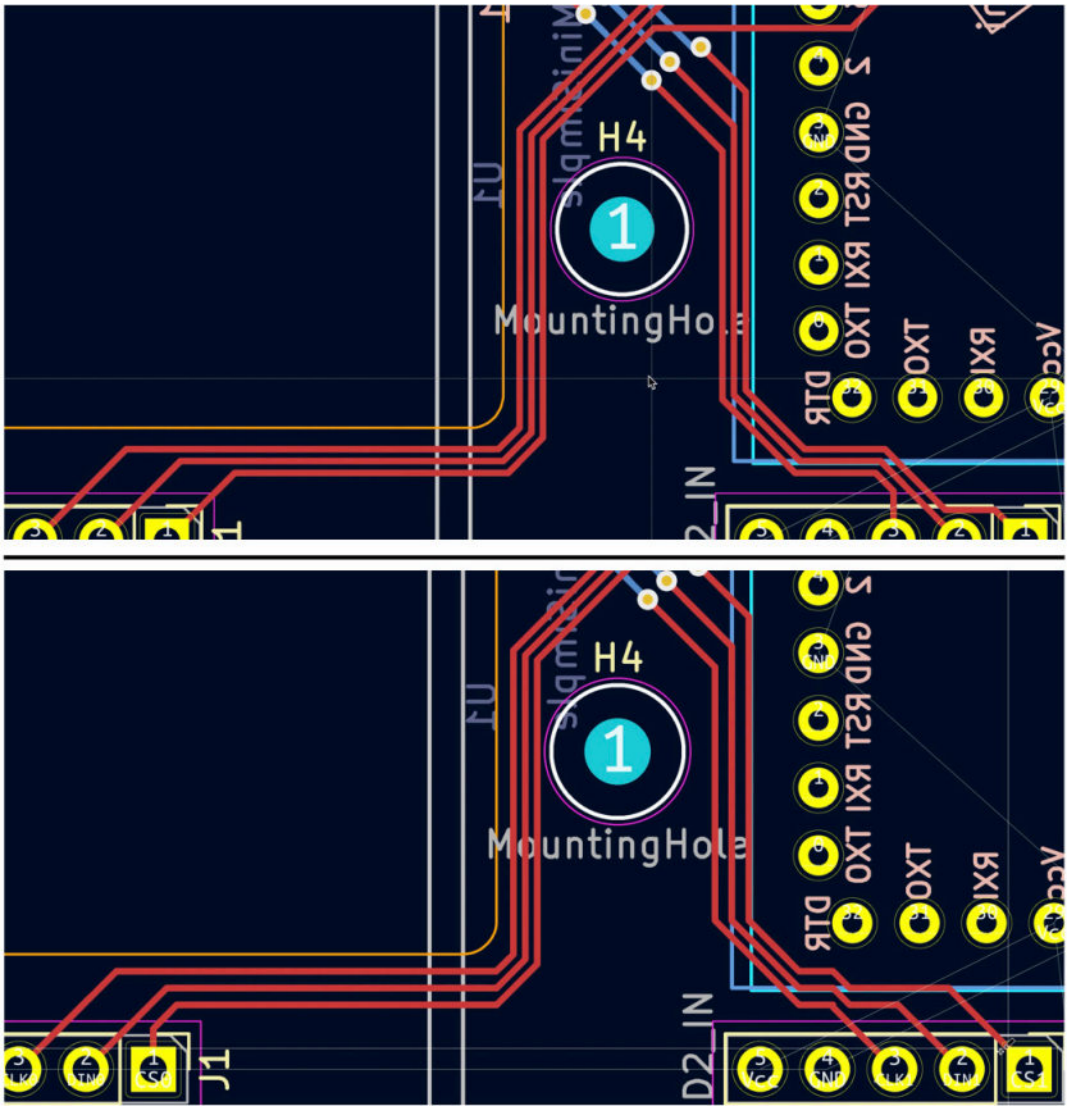


10.3.6.35: Using vias to shorten the total length of a track.

7. When you must switch a signal track to the bottom layer via a track, minimize the length of the track in the bottom copper layer. See the example above.

8. Often, it is possible to complete routing by moving existing routes and releasing space on the board instead of using vias. Explore this option first instead of creating a new via.

9. Optimise use of space by moving existing copper routes when you finish routing a group of pins. For example, in the screenshots below, you can see a completed but not optimized set of routes (bottom) and the same optimized set (top).

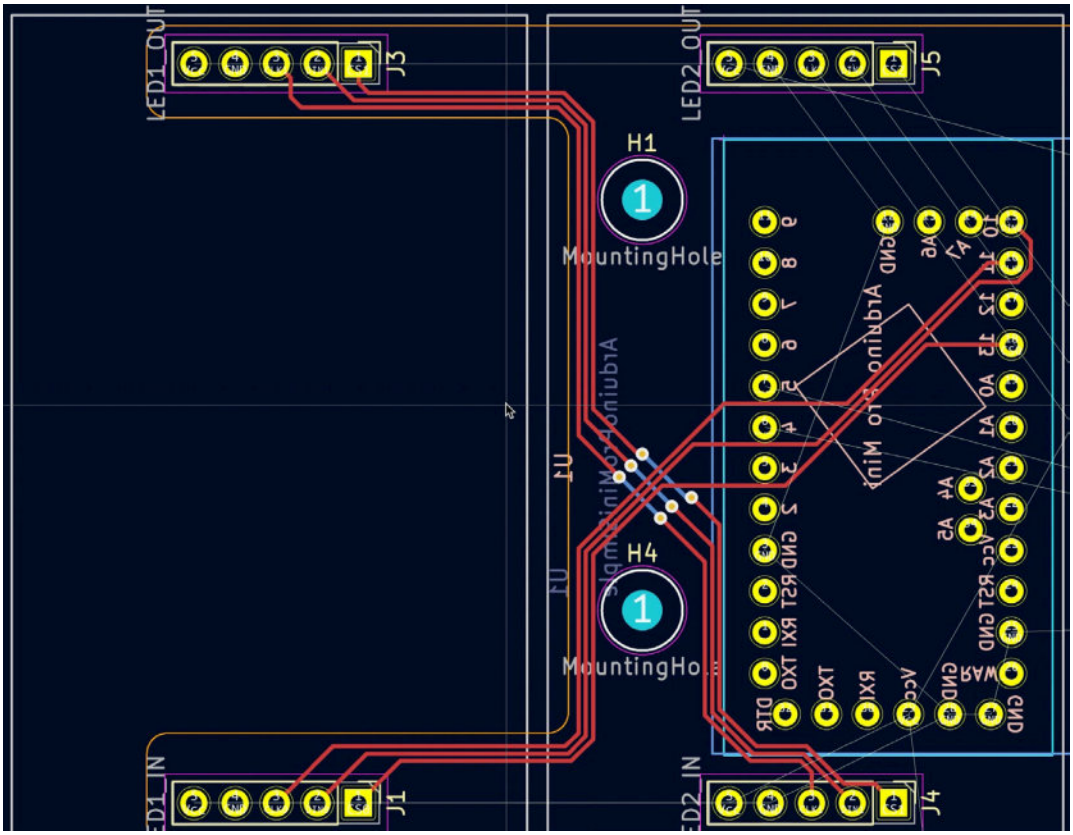


10.3.6.36: Copper track shapes original (bottom) and optimized (top).

10. If you are satisfied with a route(s), you should lock it in place (select the route and type “L”).

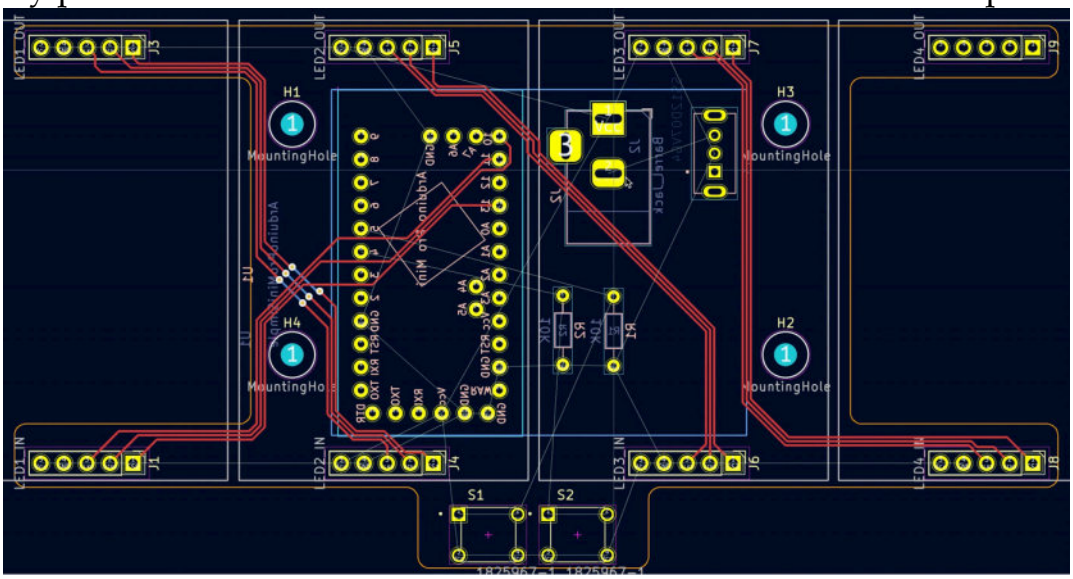
11. Use common sense. There is no single correct way of routing a PCB.

Below you can see the routing completed for the first three-pin headers:



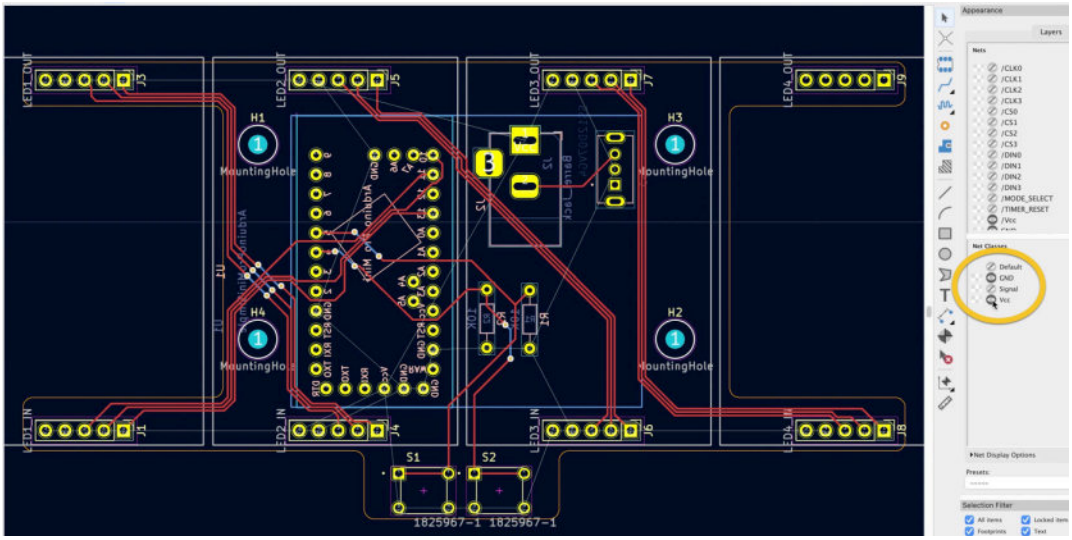
10.3.6.37: Completed routing for J1, J3, J4.

Below you can see the state of the layout after I have routed all LED display pin headers. Notice that I have not routed the GND and Vcc pads:



10.3.6.38: Routing for LED display pin headers is complete.

Continue with the buttons, the resistors, and, finally, the barrel connector and on/off switch. Below you can see the result of this process in my instance of the project:



10.3.6.39: Routing is complete, except for the GND and Vcc nets.

In the figure above, I have completed routing for all nets except for GND and Vcc. In the Appearance pane, you can toggle the GND and Vcc net class visibility to find any non-GND and non-Vcc nets yet to be routed. Turn visibility off for the two nets, and look for unconnected nets that white ratsnest lines would mark. In my example above, the only ratsnest lines that remain belong to the GND and Vcc nets.

To finish the routing, you will create two copper fills. One will connect to the GND net, and the other to the Vcc net.

3.7. 4 - Copper fills

Let's finish the routing by creating two copper fill zones.

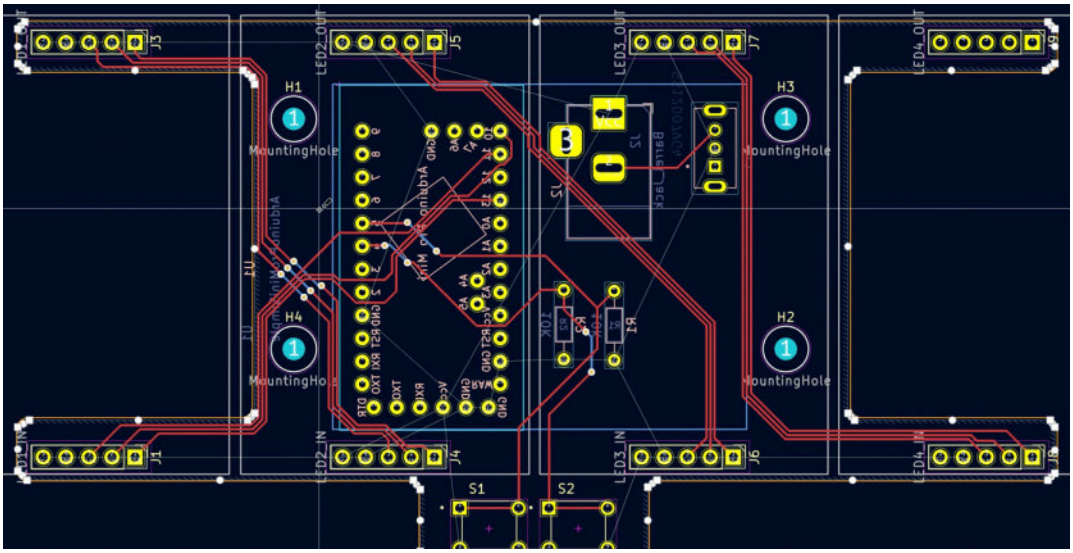
Create the first copper fill zone in the bottom copper layer, and connect it to the GND net. This copper fill one will also connect all GND pads.

Create the second copper fill zone in the top copper layer, and connect it to the Vcc net. This copper fill one will also connect all Vcc pads.

Activate the bottom copper layer from the layer chooser, and click on the copper fills button from the right toolbar. Set the grid to 0.2540 mm, and click in the top right corner of the PCB outline. In the window that appears, made these selections:

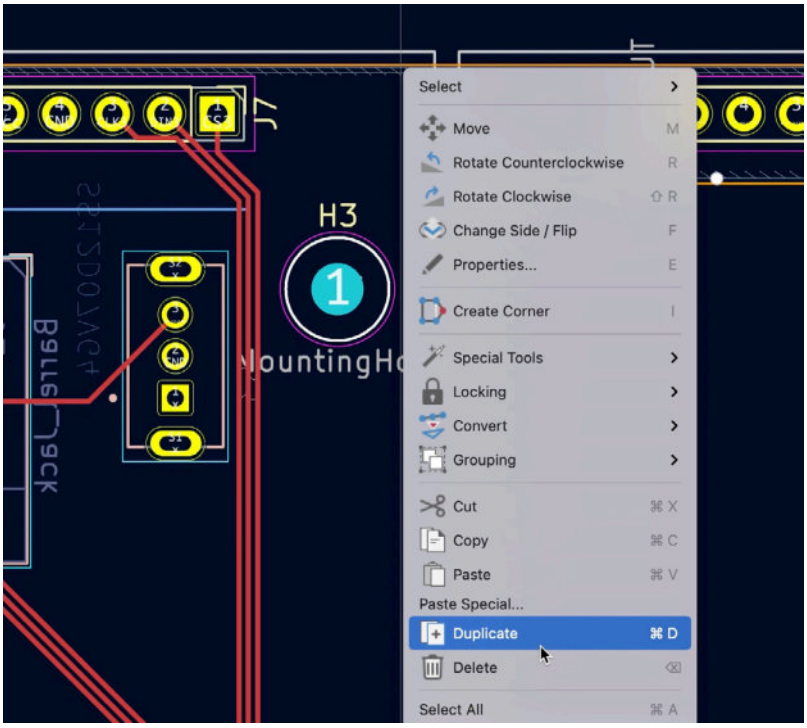
- Layer: "B.Cu."
- Net: GND.
- Fill, Fill type: Hatch pattern.

Click OK and start drawing the fill zone as close to the outline as you can. See the result below:



10.3.7.40: Bottom copper fill connected to GND.

You can repeat the process I outline above for the top copper layer, or you can duplicate the existing copper fill and change the layer to “F.Cu” and the net to Vcc for the new zone. Duplication will save you the time of drawing the multi-point polygon. To duplicate a zone, right-click on it and click “Duplicate” from the context menu.

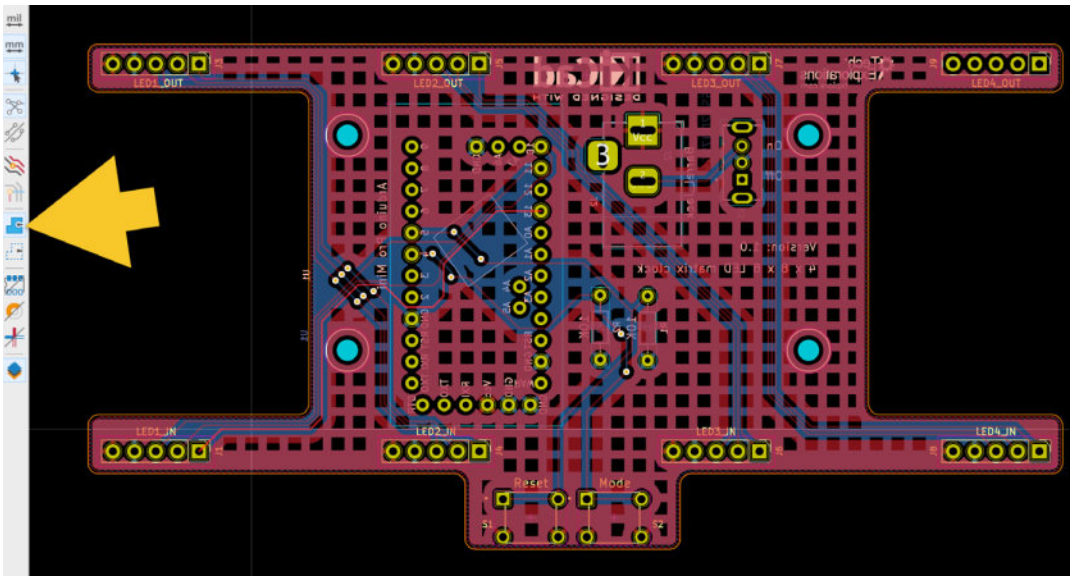


10.3.7.41: Duplicating a filled zone.

When the new zone is created, it will have the same shape as the original zone. If you place it exactly over the original zone, it may not be easy to select, so give it an offset by a few pixels. Double-click the new zone to bring up its properties window, and change the Layer to “F.Cu” and the net to

Vcc. Move the new copper fill zone over the original so that they overlap. Finally, right-click on the edge of the zones to bring up its context menu, and click “Fill All” from the Zones submenu.

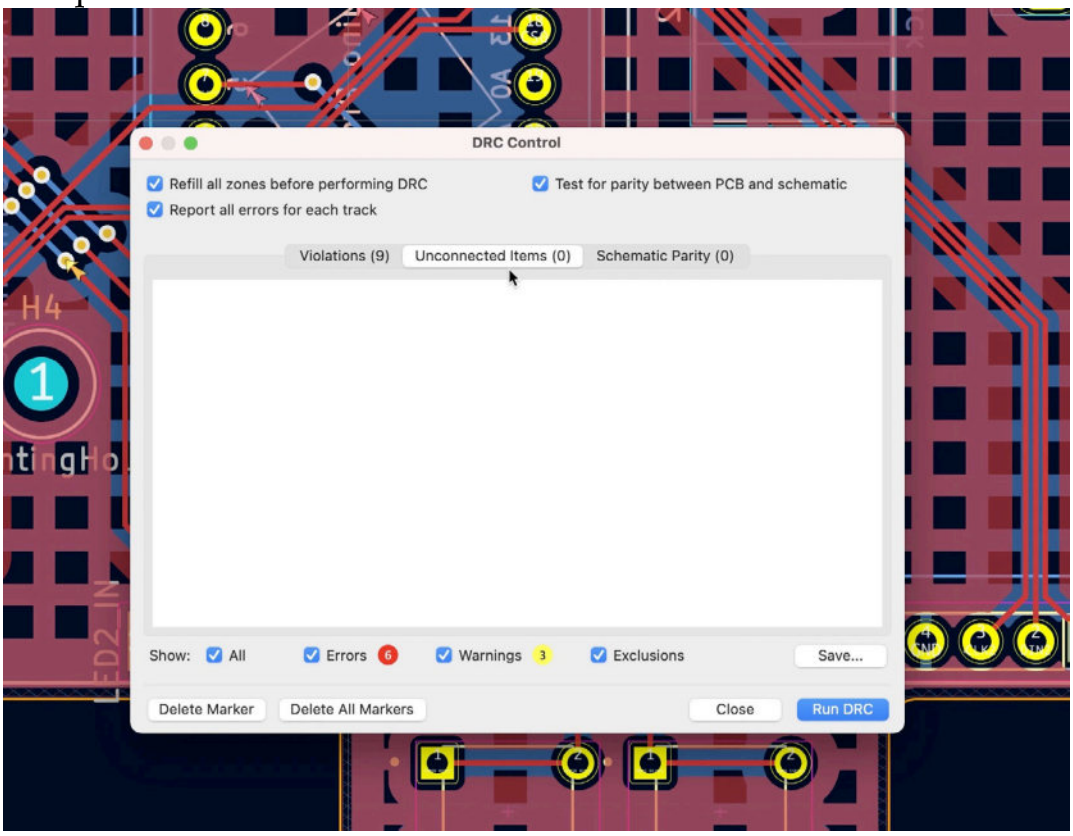
Here is the result:



10.3.7.42: Copper-filled zones are complete.

In the example above, I have clicked on the filled zones button mode in the left toolbar to depict the zones filled with their respective layer color.

A quick DRC shows no unconnected items.



10.3.7.43: No unconnected items.

This means that the copper fills completed the connections between the GND and Vcc pads. You can now continue with step five of the workflow, the silkscreen.

3.8. 5 - Silkscreen

In this chapter segment, you will add silkscreen graphics and text in the top and bottom silkscreen. With this work, you will complete step five of the layout design workflow.

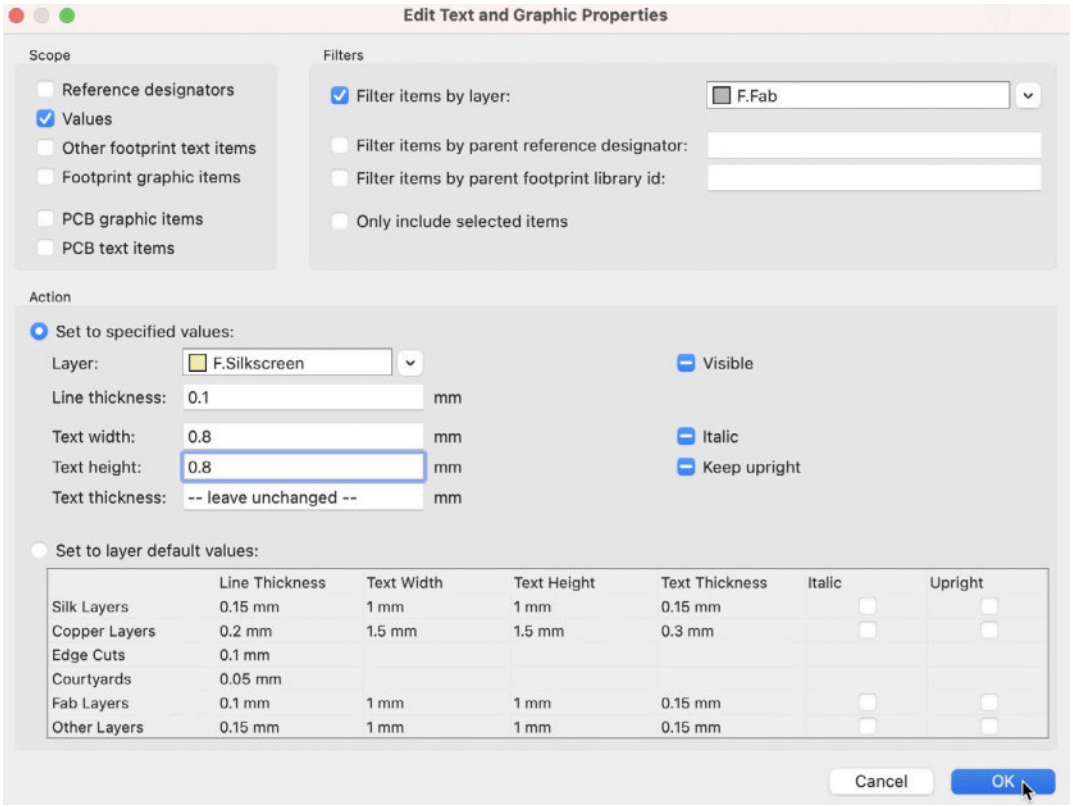
Start with adding or editing the labels for the LED matrix display pin headers. These footprints already have appropriate labels; however, they exist in the F.Fab layer. Change their layer to F.Silkscreen via their properties window, preferably in bulk using the Edit Text and Graphics Properties from the Edit menu.

Below you can see the text label attached to J3, which by default exists in F.Fab:

10.3.8.44: The text label for J3 exists in F.Fab.

Bring up the Edit Text and Graphs Properties, and set these options (if I don't mention a value below, leave it at the default):

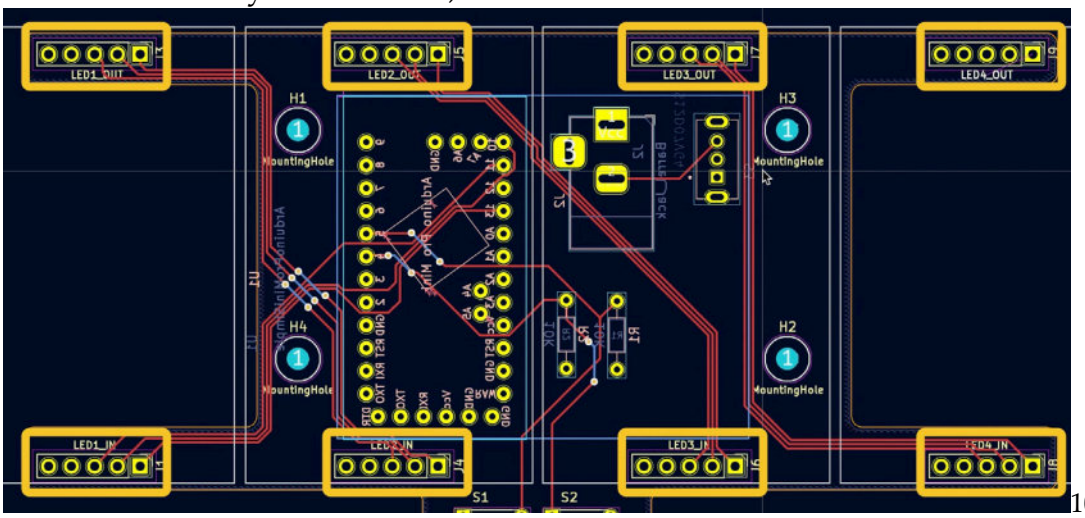
- Scope: Values.
- Filter items by layer: F.Fab.
- Set to specified values:
    - Layer: F.Silkscreen.
    - Line thickness: 0.1 mm
    - Text height: 0.8 mm



10.3.8.45: Moving text labels to F.Silkscreen and changing text attributes.

Click OK to close the properties window.

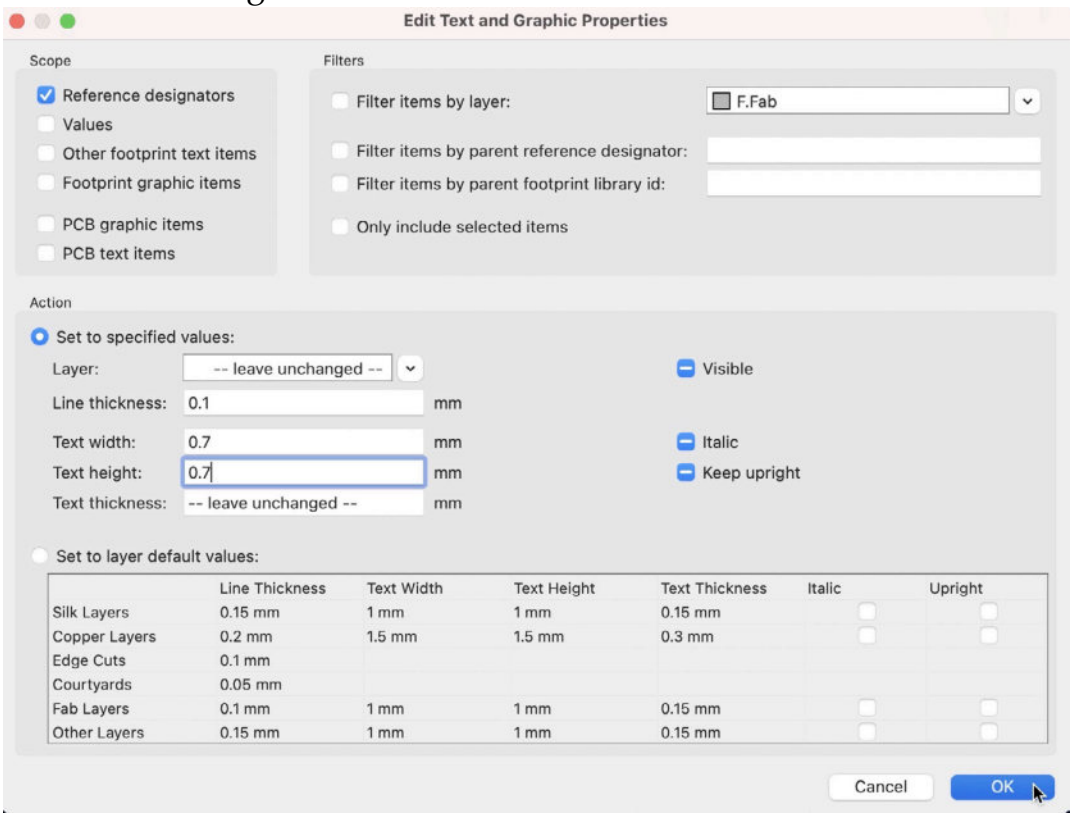
The text labels for the pin headers are now in the F.Silkscreen layer. Move them into position above or below the pin header footprints (see below, text labels in the yellow boxes):



10.3.8.46: Text label for the pin headers in F.Silkscreen.

Continue with the reference designators for all footprints. These items are already in the silkscreen (front or back, depending on the location of the footprint), but I'd like to change their size (text width and height). Again, use the Edit Text and Graphics Properties window to change those attributes for all reference designator text labels quickly. Bring up the Edit Text and Graphics Properties and use these settings:

- Scope: Reference designators.
- Action:
    - Line thickness: 0.1 mm
    - Text width: 0.7 mm
    - Text height: 0.7 mm



10.3.8.47: Changing text attributes for reference designator text labels.

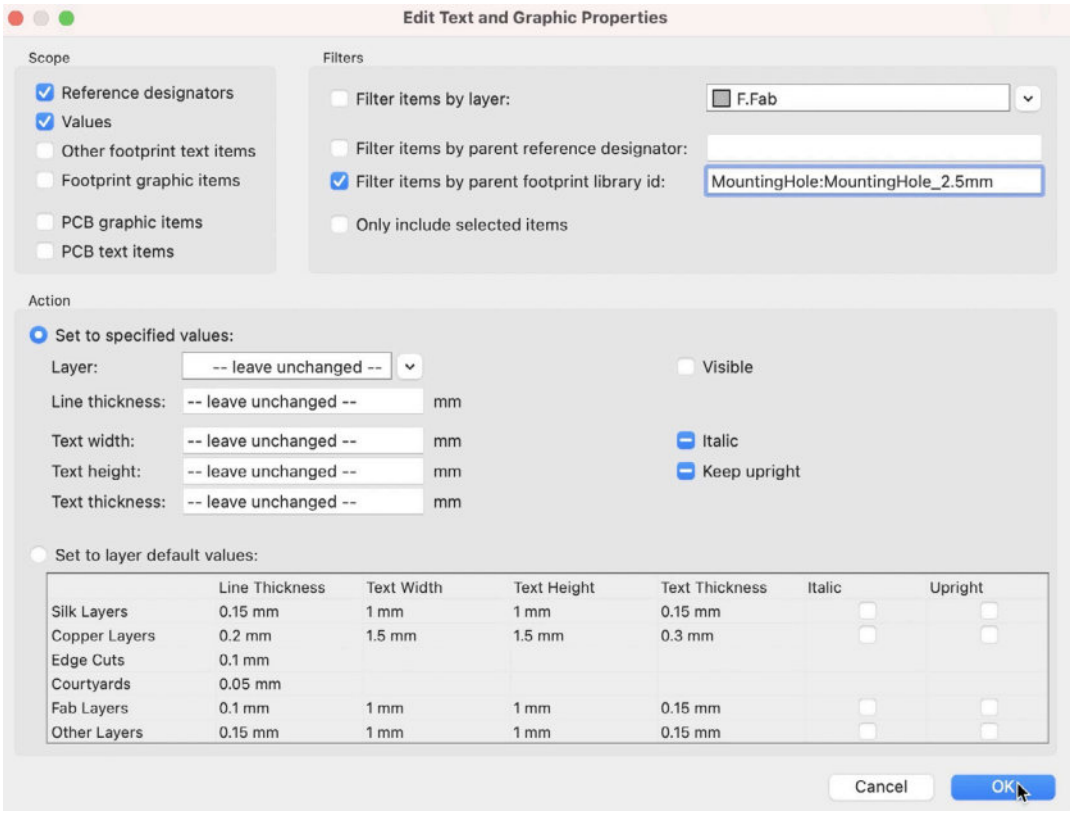
Click OK. Review the reference designator text labels and move them to appropriate positions if needed. In my case, most of these labels were in good positions, though I had to move some of them because they were placed outside the board outline.

Several text labels already exist in the F.Silkscreen layer that I prefer to make invisible because they are not helpful. For example, four such labels are the ones with the text "Mounting Hole." The mounting hole footprints have reference designators H1, H2, H3, and H4, which I also want to make invisible. Those footprints share the same library link, "MountingHole:MountingHole_2.5mm". You can use this library link to make these labels not visible using the Edit Text and Graphic Properties window.

Open the Edit Text and Graphic Properties window and set these settings:

- Scope:
    - Reference designators.
    - Values.

- Filters:
    - Filter items by parent footprint library id: MountingHole:MountingHole_2.5mm
- Action:
- Visible: uncheck.



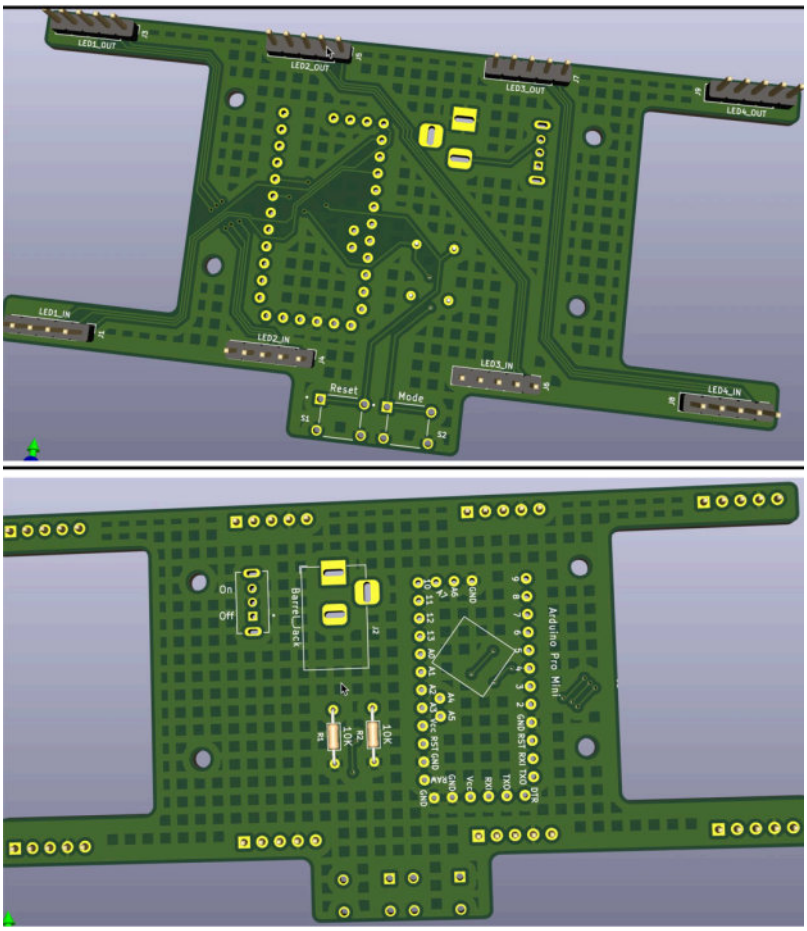
10.3.8.48: Making all mounting hole values and reference designators invisible.

Here is a list of other silkscreen work that I suggest you do at this time:

1. Move reference designators to positions that do not overlap with other silkscreen elements outside the footprint's outer boundary. This is so that the designators are visible even when the board is assembled.
2. For footprints S1 and S2, make the value text labels invisible.
3. Add two text labels next to S1 and S2: Reset (over S1) and Mode (over S2).
4. Add two text labels for the on/off switch: "On" (close to pin 3), "Off" (close to pin 1). Notice that the on/off switch footprint exists in the bottom copper layer. Therefore, the new text labels should be in the B.Silkscreen layer, and the Mirrored checkbox selected.

5. For the resistors, change the layer for the values to B.Silkscreen.
6. For the Arduino footprint, move the footprint text property ("Arduino Pro Mini") outside the footprint's perimeter.
7. For the Arduino footprint, make the value name ("ArduinoProMiniSimple ") invisible.
8. For the Barrel Jack, J2, change the layer of the value attribute ("Barrel_Jack") to B.Silkscreen.

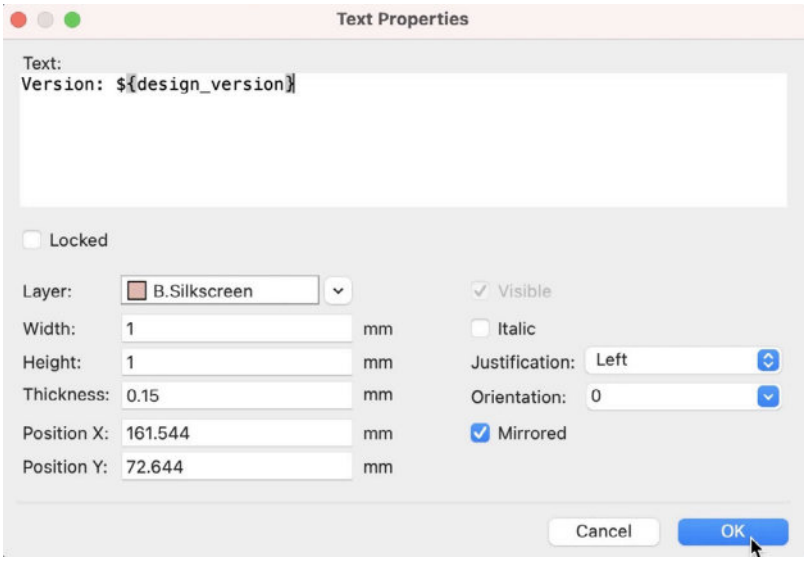
At this point, the back and front of the board look like this:



10.3.8.49: Inspecting the front and back silkscreen in 3D.

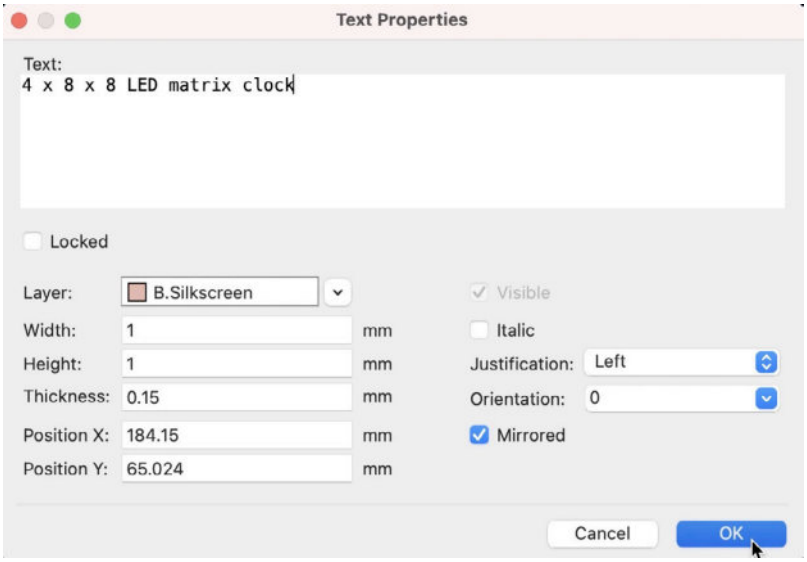
The front and back silkscreen look good. I will complete this work by adding the final silkscreen elements in the back silkscreen:

1. Power information (input voltage and connector polarity.
2. The Kicad logo. Use the footprint from the Symbol library with the name "KiCad-Logo2_5mm_Silkscreen". In the footprint properties, select "Back" from the Side dropdown.
3. The Tech Explorations logo.
4. A text label with information about the project. I will use the project version from the text variable with the name "design_version" that I created earlier in the project (see below).



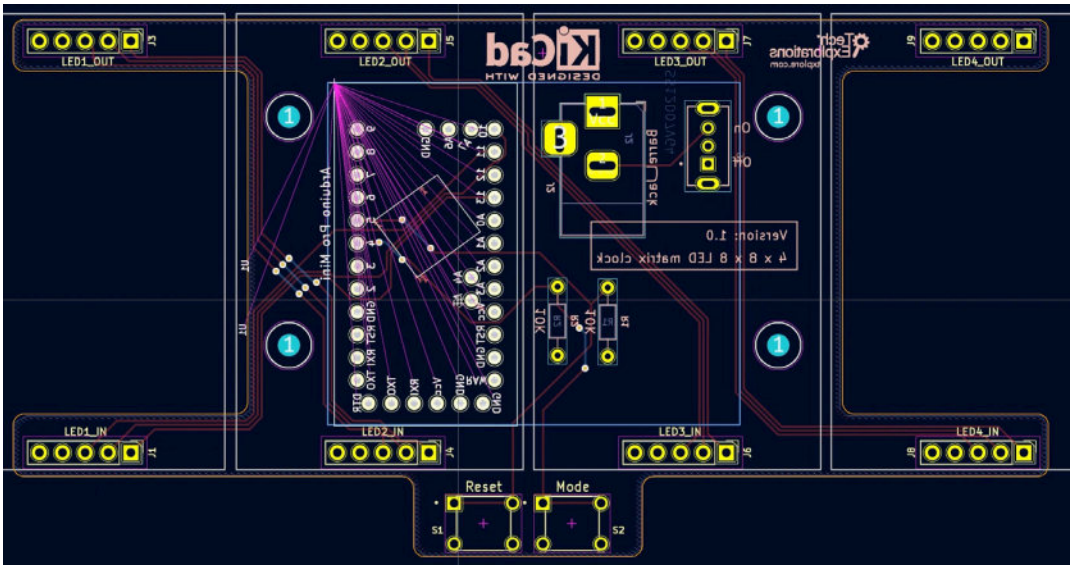
10.3.8.50: Using a text variable in a text label.

4. A text label that contains the name of the project and board, with a box around it:



10.3.8.51: A text label that contains the name of the board and project.

Here is the current state of the silkscreen:



10.3.8.52: Silkscreen work is complete.

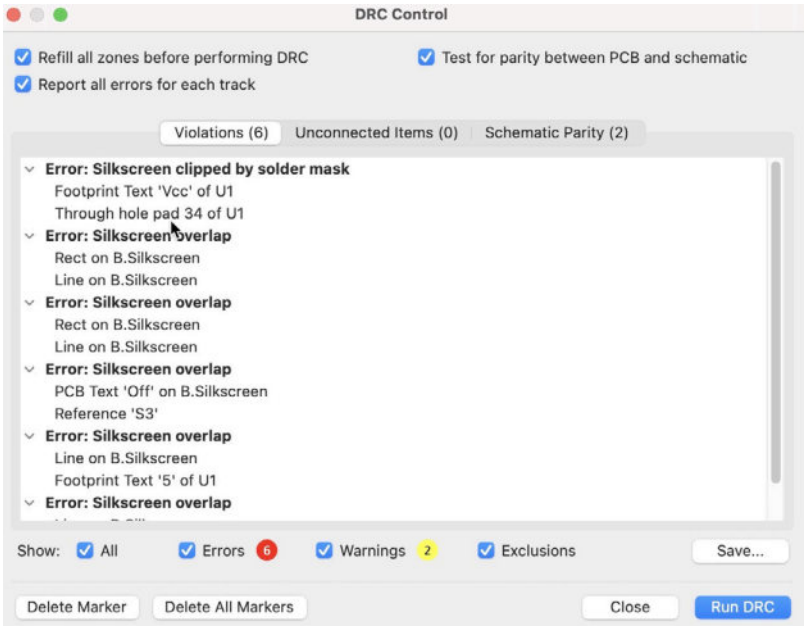
In the current state of the silkscreen, there may be a few issues that will need to be corrected. For example, you can see that the silkscreen lines that mark the perimeter of the barrel jack footprint overlap with the box around the name and version of the project. The DRC will list this overlap as a rules violation. The power input information is also missing, and will add it in the next lecture.

Instead of doing these corrections now, I will proceed to the workflow's next step, where I will run the DRC and correct each violation listed by the checker.

3.9. 6 - Design Rules Check

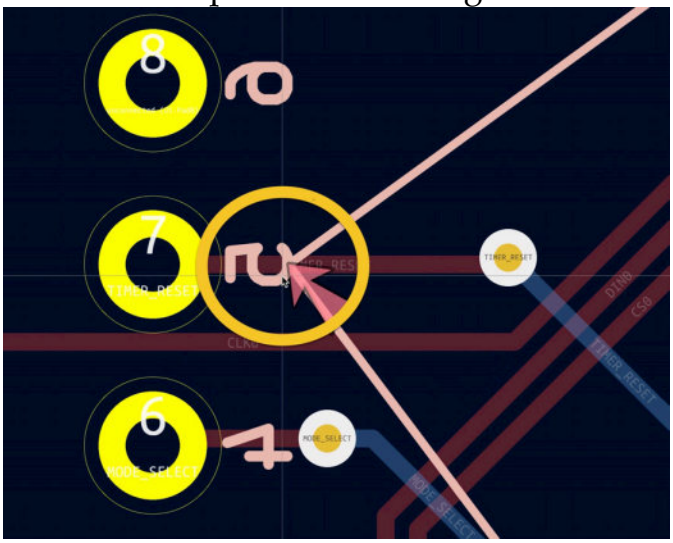
You've done a lot of work on this project, and you are close to completion. Are there any defects? Let's run a final DRC to find out.

Bring up the DRC tool and click on Run DRC. The results are below:



10.3.9.53: Several DRC errors relate to problems with silkscreen items.

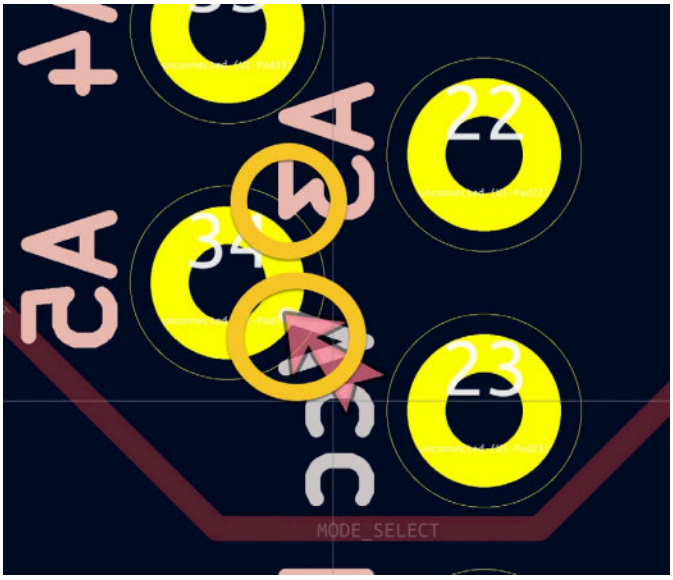
Several DRC errors relate to problems with silkscreen items. I have captured a common silkscreen problem in the figure below:



10.3.9.54: A common silkscreen error: two silkscreen items overlapping.

In this example, the problem is with a text label ("5") overlapping two graphic lines. These silkscreen items belong to the Arduino Pro Mini footprint. It is an example of an error that I made when I designed this footprint but did not discover until I ran the DRC as part of this project.

Below is another common problem:

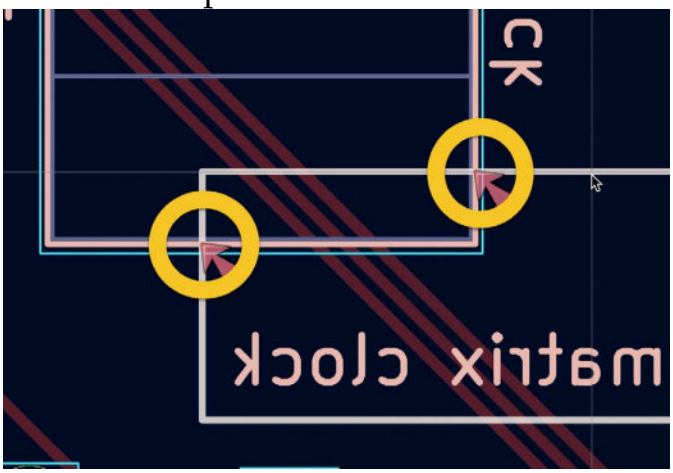


10.3.9.55: A common silkscreen error: silkscreen clipped by solder mask.

In this case, two text items in the silkscreen are inside the solder mask boundary of pad 34.

If these issues appear in the actual layout, you can move the silkscreen items to correct positions so they don't creep in a pad's solder mask or overlap with another silkscreen item. However, because these errors occur within my custom Arduino Pro Mini footprint, you will need to use the footprint editor to correct the errors in the footprint.

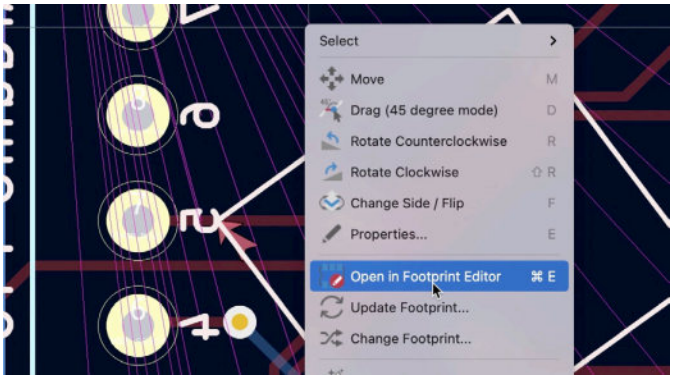
Here's another common error. This one exists in the actual board layout, instead of inside a custom footprint:



10.3.9.56: Silkscreen lines overlapping.

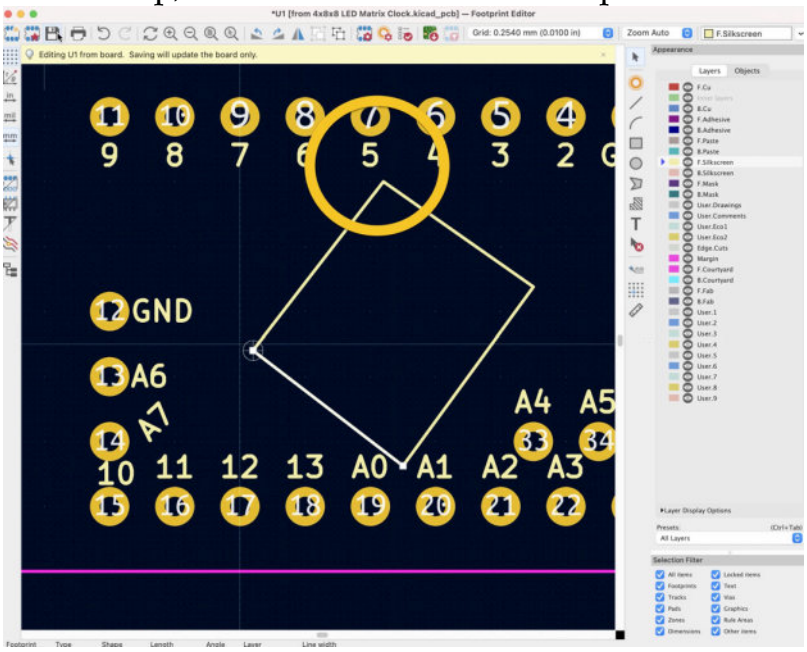
Let's fix those errors.

For the error inside the Arduino Pro Mini footprint, right-click on the footprint and select "Open in Footprint Editor" from the context menu.



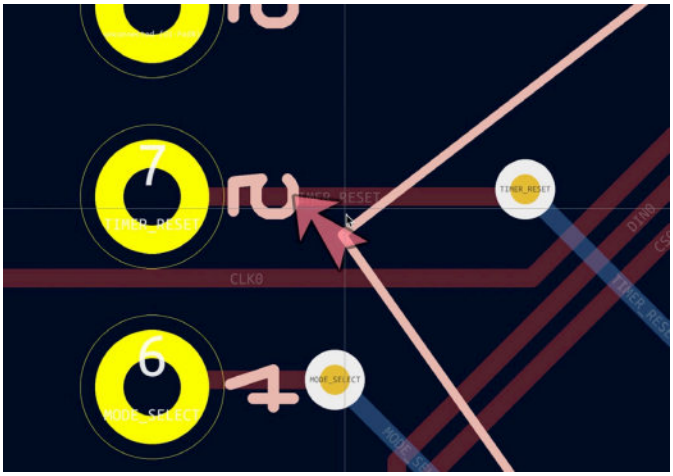
10.3.9.57: Open in Footprint Editor.

In the footprint editor, move the two lines that join over text label “5” so there is no more overlap, as I have done in the example below:



10.3.9.58: Fixing the silkscreen overlapping error.

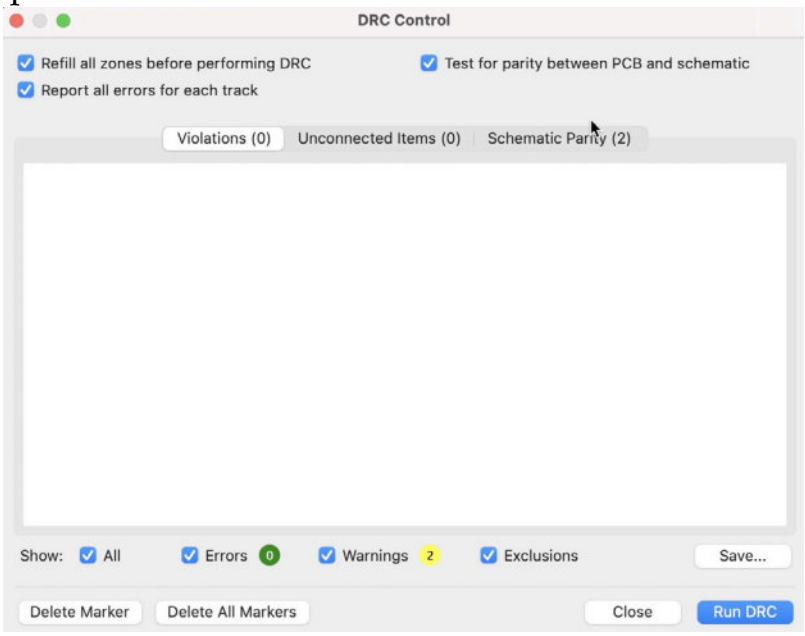
Save the updated footprint, close the footprint editor, and return to Pcbnew. Confirm that there is no more overlap between text label “5” and the two graphic lines:



10.3.9.59: Problem fixed.

For the problem with the overlapping lines that belong to the barrel connector footprint outline and the board version and name, I have deleted those lines.

Let's repeat the DRC:



10.3.9.60: Zero errors.

The DRC shows no more errors. There are a couple of schematic parity warnings that you can safely ignore. These are caused by the KiCad and Tech Explorations logos which exist as footprints in the layout but have no symbol counterpart in Eeschema.

Step six of the workflow is complete. Let's continue with the last step, where I will export the Gerber files and upload them for manufacturing.

3.10. 7 - Manufacture

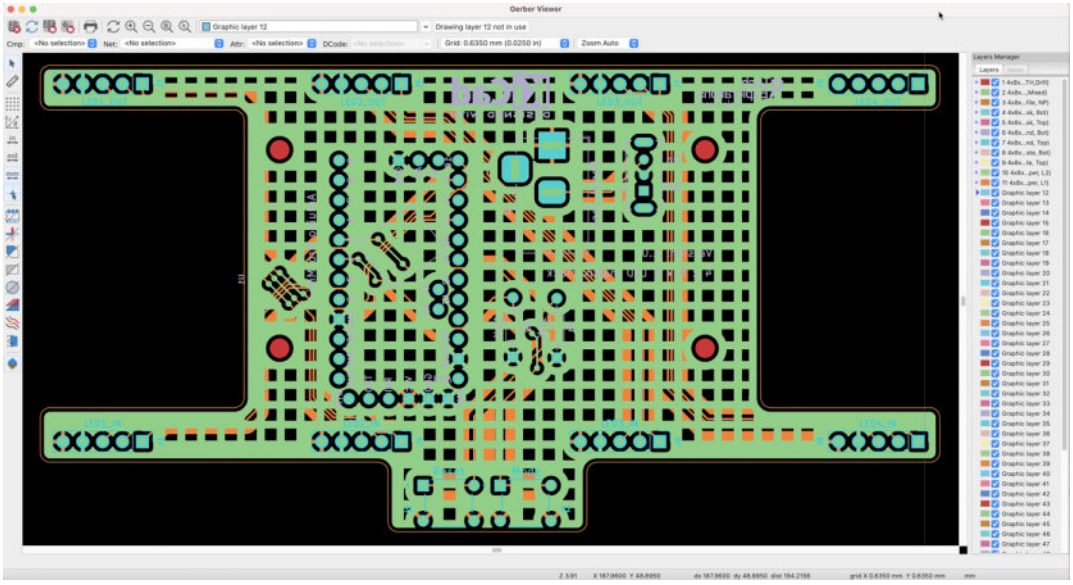
Let's complete the project by exporting the Gerber files, using the Gerber Viewer app to inspect them, and uploading the files to the manufacturer's website.

ATTENTION: Please do not manufacture this version of the PCB yet! There is a correction that I have documented in a "bonus" chapter later in this part of the book. I recommend that you go ahead with manufacturing once you have read the chapter that contains the correction.

In Pcbnew, click on the plotter button (top toolbar) to bring up the Plot window. Set the output directory, check the layers to include in Gerber file export, and generate the Gerber layer and drill files. Please refer to chapter

“17. How to export and test Gerber files” if you don't remember how to do this.

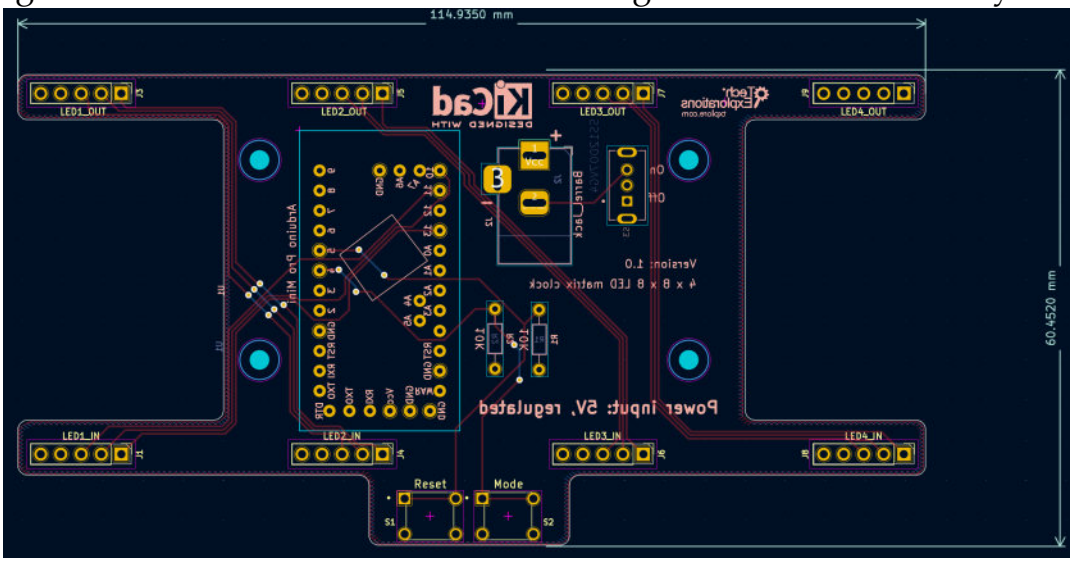
Once you generate the Gerber files, use the Gerber viewer to evaluate the fitness of these files for manufacturing. My Gerber viewer instance looks like this:



10.3.10.61: Examining the exported Gerber files in Gerber Viewer.

Patiently inspect each layer separately to ensure that they are correct. Be particularly careful with text items in the silkscreen and look for misspellings and typos.

When you finish reviewing the board in Gerber Viewer, return to Pcbnew to measure the board's dimensions. Some manufacturers require you to type those values in the order form. I have used the measurement tool from the right tool mark to mark the width and height in one of the user layers.



10.3.10.62: The final PCB with measurements.

Proceed to create a ZIP archive with the Gerber files and upload this file to your preferred manufacturer's website.

The project is now complete. However, there are three more chapters in this part of the book that are important to read. In the next chapter, you will learn to add 3D shapes for the switch and barrel connector footprints. Then, a chapter shows how to fix a design bug that I found in the PCB. This bug is something that the DRC cannot detect, and I only discovered it after testing the manufacture prototype board. The last chapter contains photographs of the manufactured and assembled PCB.

4. Bonus - 3D shapes

Bonus - Found a bug in the schematic! (and fix)

Bonus - 3D shapes

Bonus - Found a bug in the schematic! (and fix)

The assembled and working PCBThe assembled and working PCB

The 3D viewer renders a 3D representation of your PCB. When footprints on the PCB have associated 3D shapes, the 3D viewer will also renter those shapes and give you a PCB visualization with the components to approximate a "real world" representation of the PCB.

Without any footprint and 3D shape associations, the 3D representation of the board of this project looks like this:

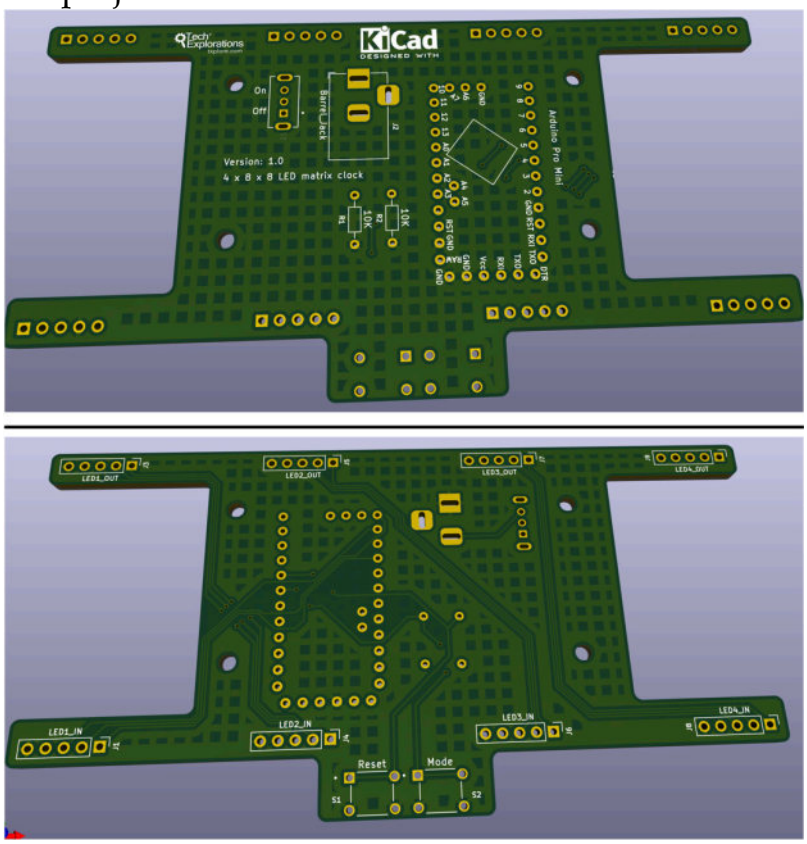


Figure 10.4.1: The 3D viewer showing only the core PCB elements.

Once you associate footprints with appropriate 3D shapes, the 3D-rendered board will look like this:

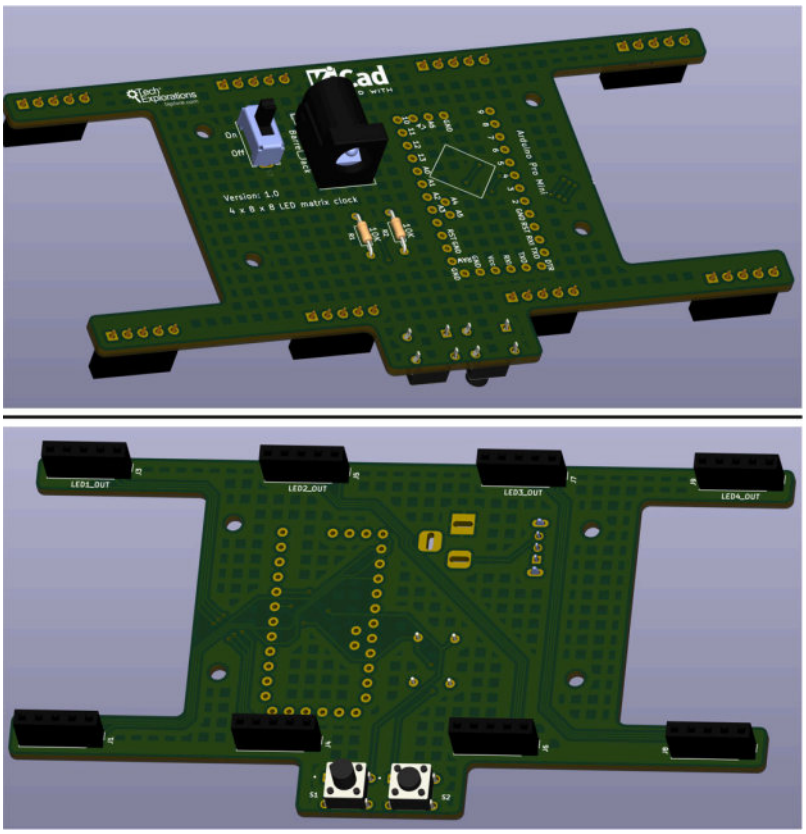


Figure 10.4.2: The 3D viewer showing the PCB with several 3D shapes.

In this chapter, you will learn how to find a 3D shape for the slide switch and associate it with the relevant footprint so that the 3D viewer includes it in the rendering of the PCB.

Please also refer to a chapter dedicated to the 3D viewer in Part 8 of this book.

The 3D shape I am looking for is for the slide switch, with reference designator S3. Open this footprint's properties window, and click on the 3D Models tab.

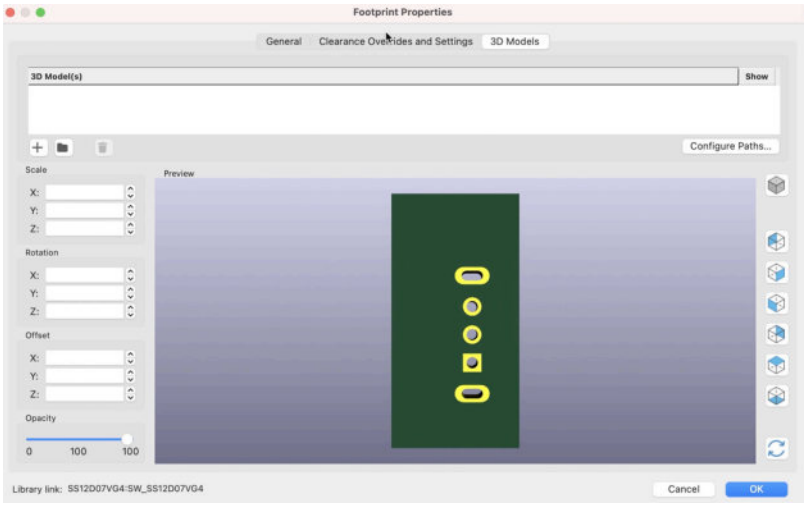


Figure 10.4.3: This footprint has no 3D shape associated.

The 3D Models table is empty. This means that there is no 3D shape associated with this footprint. Let's find one and add it to this table. The model of this slide switch is "SS12D07VG4". You can find this in the Value property of the General tab. I found this footprint in Snapeda, so I will use Snaped again to look for a matching 3D shape. A matching 3D shape does not necessarily need to have the same model code. The same shape can match more than one footprint. So, instead of searching only for "SS12D07VG4", you can search for a more general term like "slider switch." With a narrow search, you will find a matching shape quickly - if there is a match.

Below, I have done a comprehensive search and found a matching 3D shape. You can access this shape [using this URL](https://www.snapeda.com/parts/OS102011MS2QN1/C&K/view-part/3915).

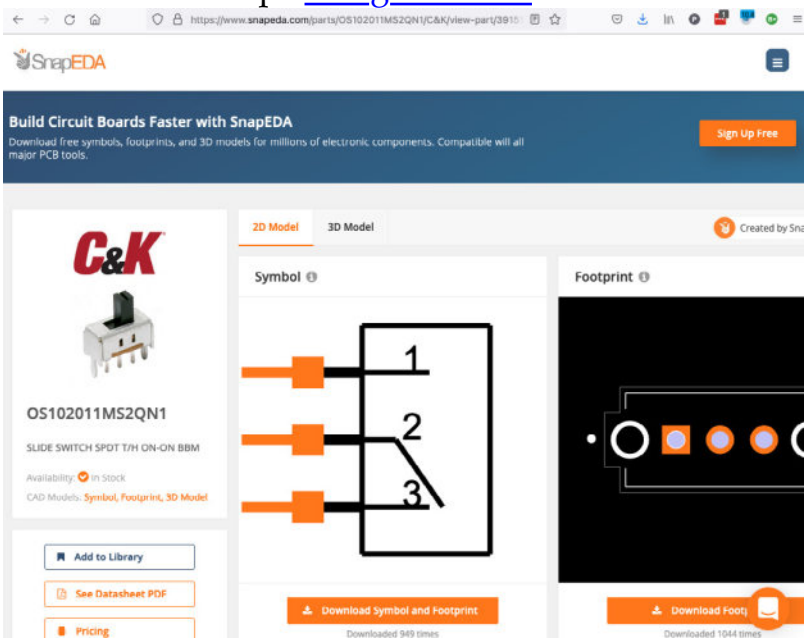


Figure 10.4.4: Found a matching 3D shape.

Download the 3D shape as I explain in the chapter “16. Finding and using a 3D shape for a footprint”. Expand the ZIP file, and note the location of the file with the ".STEP" extension:

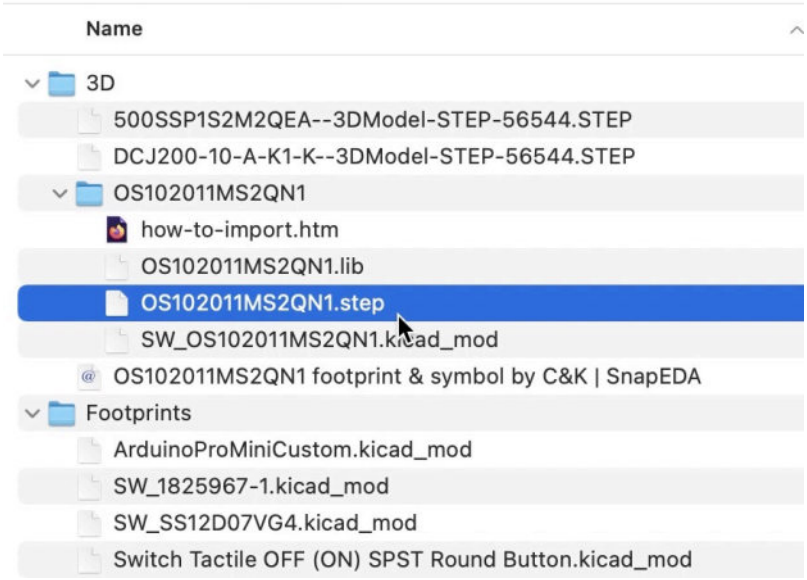


Figure 10.4.5: The 3D shape file ready to use.

Return to Pcbnew and open the slide switch footprint's properties window. Click the 3D Models tab. Under the empty 3D shapes table pane, click on the "+" button to add a new row. Then click on the folder button on the right side of the row to open the file browser. Navigate to the ".STEP" file's location and double-click on it to add it to the table. The "3D models" tab looks like this:

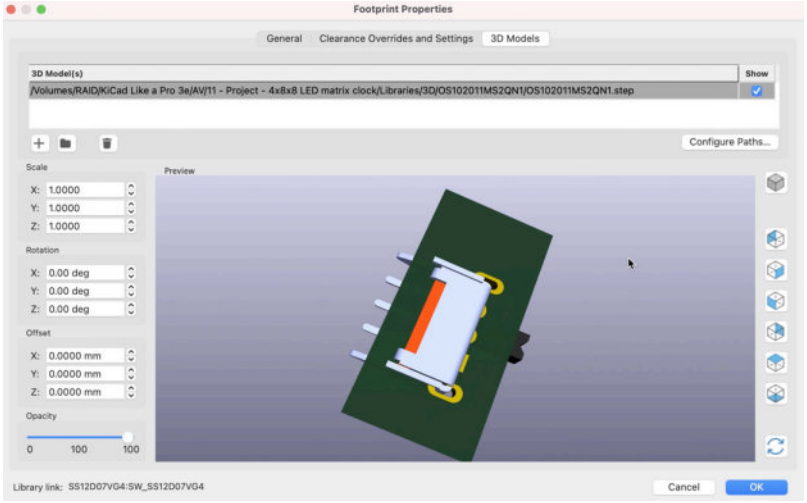


Figure 10.4.6: The 3D shape is associated with the footprint but is misaligned.

The 3D shape is now associated with the footprint but is misaligned. Use the widgets in the Scale, Rotation, and Offset groups to align the 3D shape and fit with the footprint. Here's mine, after alignment:

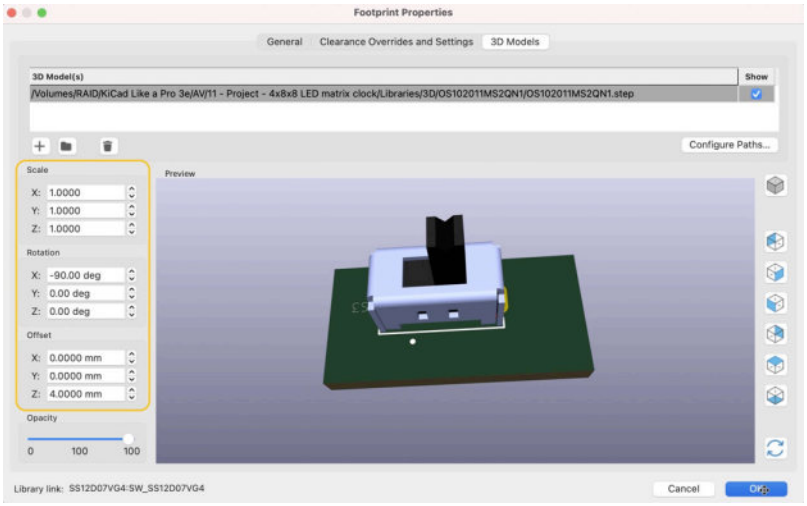


Figure 10.4.7: The 3D shape is associated with the footprint aligned.

Click OK to close the properties window and open the 3D viewer to check the result of this work.

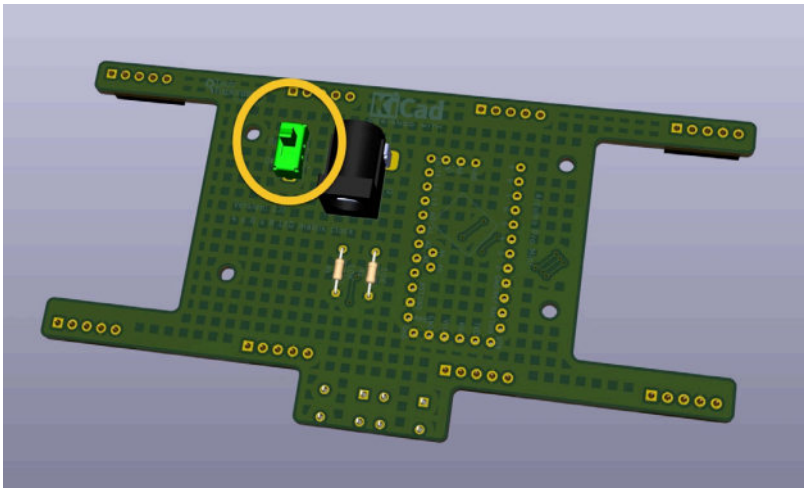


Figure 10.4.8: The new 3D shape is included in the board rendering.

The new 3D shape is now included in the board's rendering in the 3D viewer. Below you can see the slide switch in the assembled board:

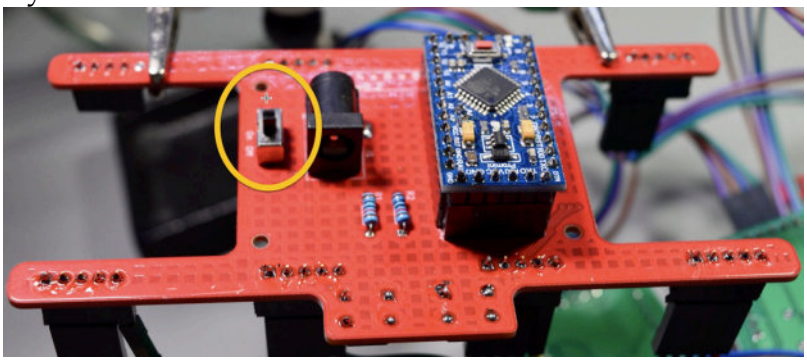


Figure 10.4.9: The slide switch in the assembled PCB.

Below you can see the final 3D rendering of this PCB, after including the power input information and replacing the LED display module connectors from the original male headers to female connectors.

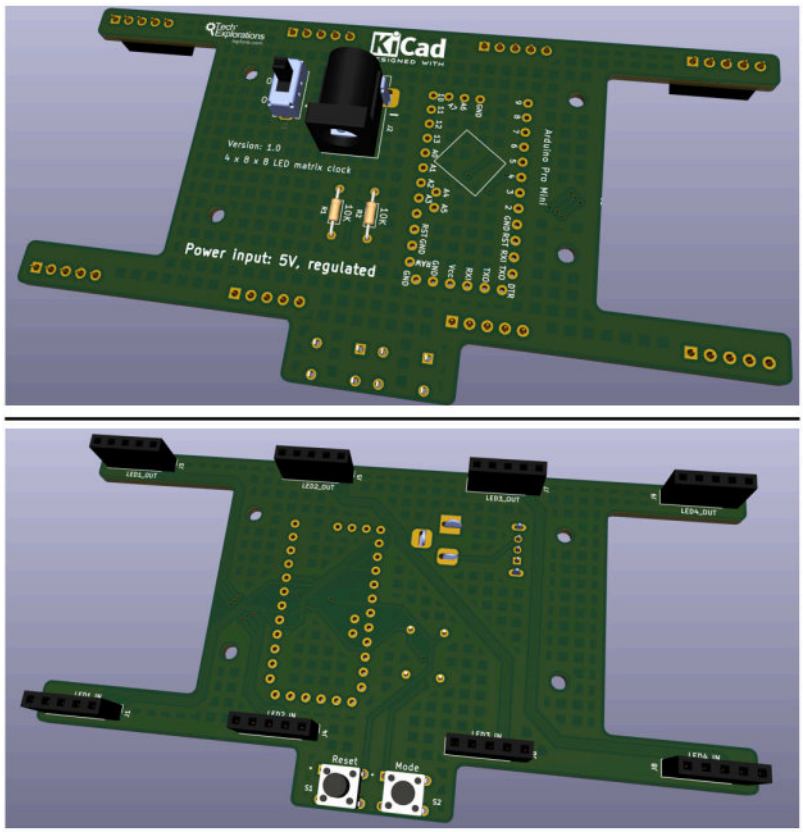


Figure 10.4.10: Final 3D rendering showing all silkscreen.

Finding and adding 3D shapes to your project helps to create a more realistic depiction of your board. However, to increase realism, you will need to expend an additional amount of effort. Is it worth it? In some cases, it does; in others, it does not. In general, when I share a project with other people, it is worth increasing the realist of the 3D rendering. I rarely add 3D shapes for projects that I don't share with others.

5. Bonus - Found a bug in the schematic! (and fix)

As I was wrapping up this project, I discovered a minor bug in the schematic. Specifically, I found that I had made a mistake in the net labels attached to some of the wires in the LED matrix display group of pin headers.

Luckily, the error was only in the net labels. The wiring was correct. In this chapter, I will show you how to correct this error in Eeschema and update Pcbnew.

First, let's look at the error. See the figure below.

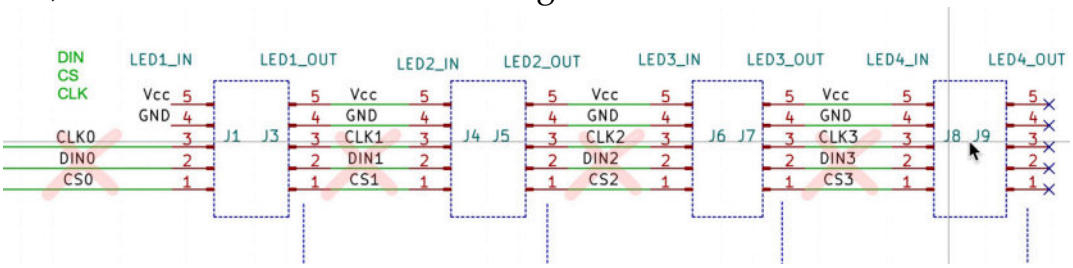


Figure 10.5.1: I have marked the errors with the red "X."

The net labels for pins 1, 2, and 3 are incorrect. Those are marked with the red "X." The correct labels are "DINx", "CSx" and "CLKx", where "x" is 0, 1, 2, 3.

To fix this error, delete all incorrect labels and replace them with the correct ones. The corrected LED matrix display group is below:

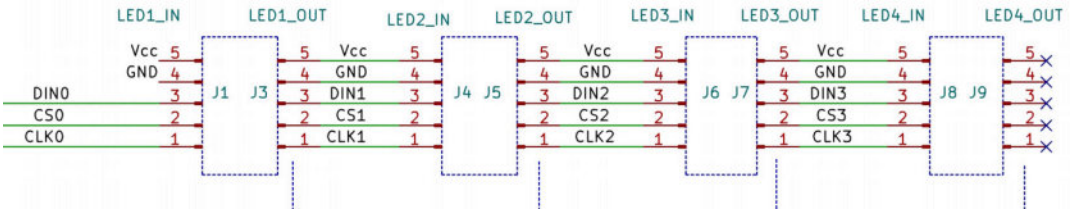


Figure 10.5.2: Net labels corrected.

The schematic is corrected. You now need to update the layout editor with the changes from the schematic. Click on the Update button from the top toolbar and click Update PCB.

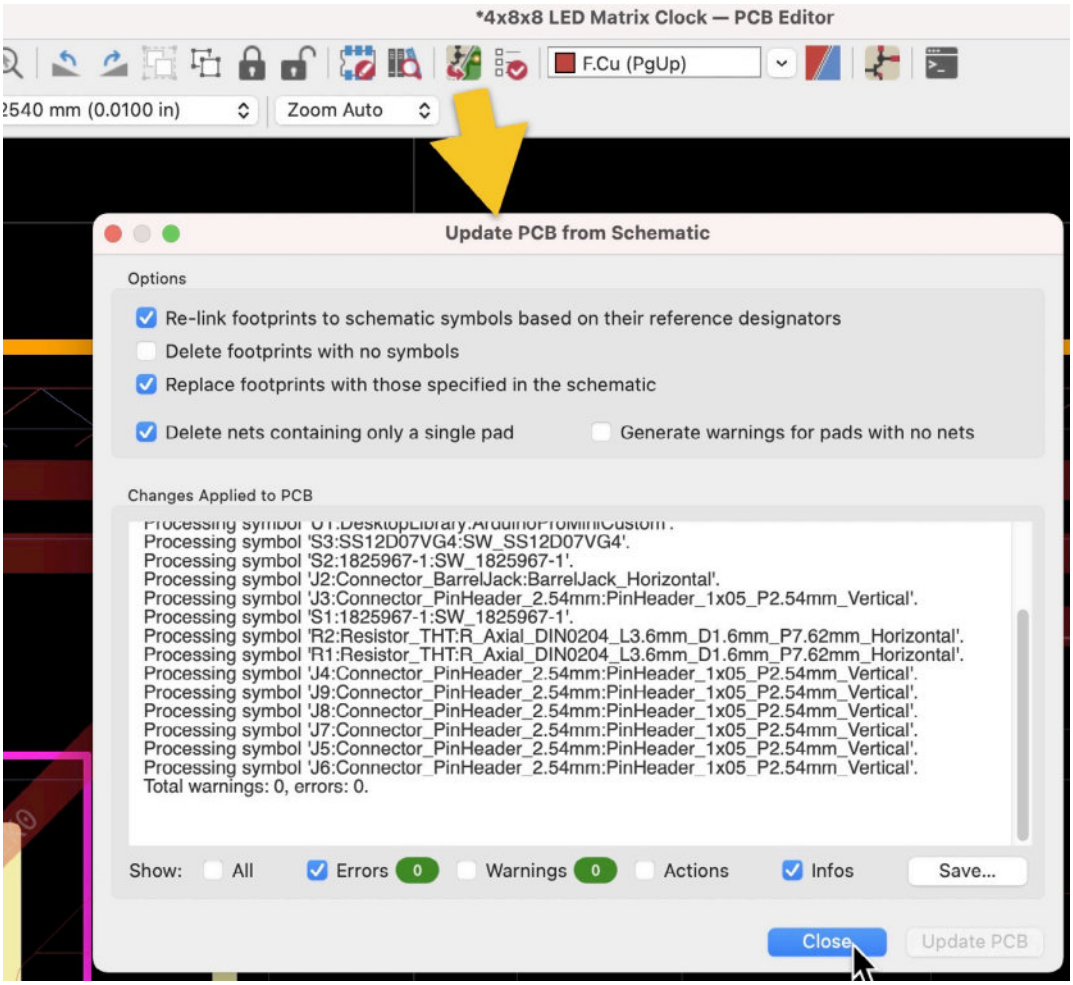


Figure 10.5.3: Updated PCB from schematic.

Click Close and return to the layout editor.

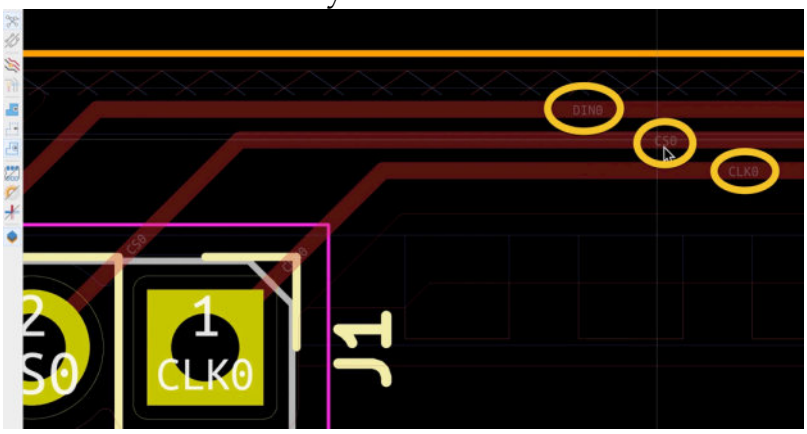


Figure 10.5.4: Confirming the changes in the PCB.

To confirm that the changes in the schematic update the layout, look at the tracks that come out of J1. For example, J1 pin 1 is connected to the CLK0 net. The track that is connected to this pin also belongs to CLK0. Similarly, in the schematic in Figure 10.5.2, pin 1 of J1 is connected to net CLK0.

Double-check with a few other pins and nets to confirm that the schematic and layout agree.

Because there is no change in the tracks, you don't need to re-export the Gerbers, and if you have already ordered a manufactured PCB, it will still work. This was an example of how to quickly fix an error that originates in the schematic but has no effect on the electrical characteristics of the PCB.

6. The assembled and working PCBSchematic design

Schema 1 - Setup

Schema 2 - Symbols

Schema 3 - Arrange, Annotate

Schema 3 - Associate

Schema 4 - Wiring

Schema 5 - Nets

Schema 6 - Electrical Rules Check

Schema 7 - Comments

Schema - Last-minute edits

Layout design

Layout 1 - Setup

Layout 2 - Outline and constraints

Layout 3 - Place components

Layout 2 supplemental - Refine outline

Layout 3 supplemental - Move footprints to back layer

Layout 4 - Route

Layout 4 - Copper fills

Layout 5 - Silkscreen

Layout 6 - Design Rules Check

Layout 7 - Manufacture

Bonus - 3D shapes

Bonus - Found a bug in the schematic! (and fix)

The assembled and working PCB

A couple of weeks after ordering the PCB for this project, it arrived in the mail. I did the assembly and testing, and I am glad to say that it worked. Below is a snapshot of the assembled PCB displaying the rolling word "Arduino":

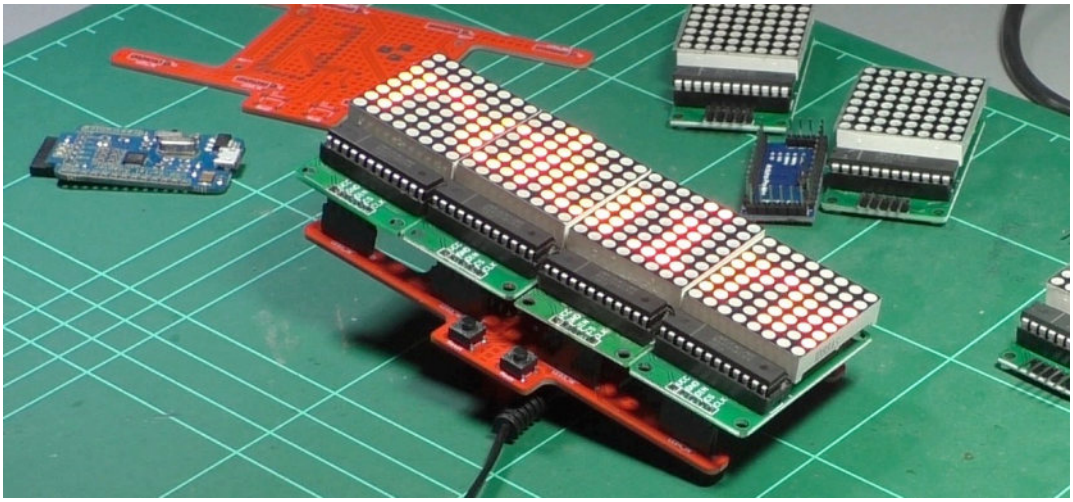


Figure 10.6.1: The assembled and working PCB.

The most challenging aspect of the assembly is related to the LED matrix displays. These displays contain the MAX7819 controller chip and come with 90-degree headers. Here is an example from Amazon. I had to desolder the original headers and replace them with straight headers to plug into the female headers on the PCB.

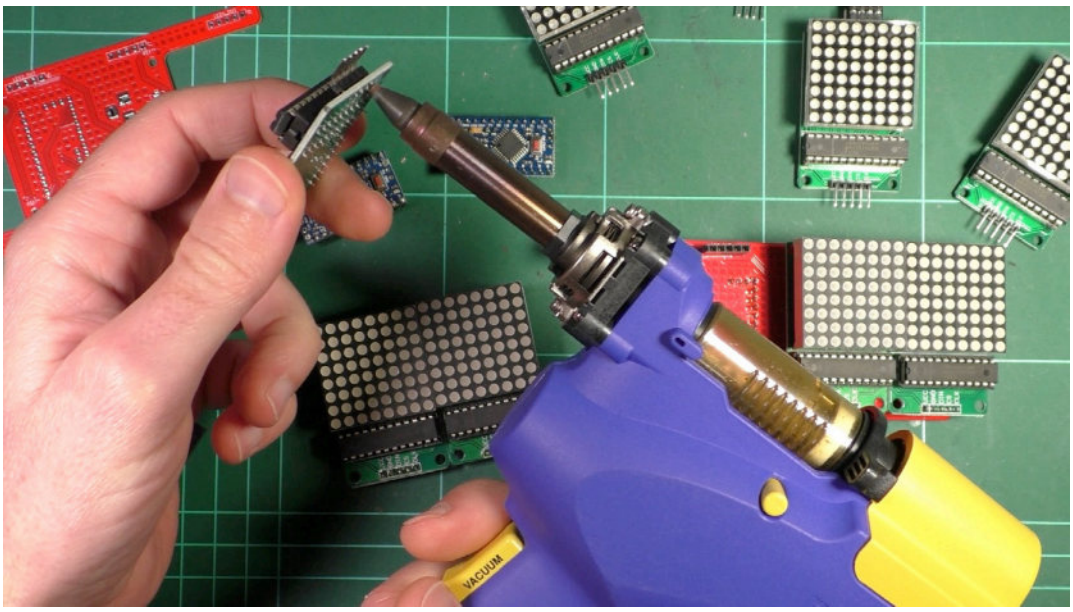


Figure 10.6.2: Desoldering the original headers from an LED matrix display.

Then, attach the displays to the board via the female headers:

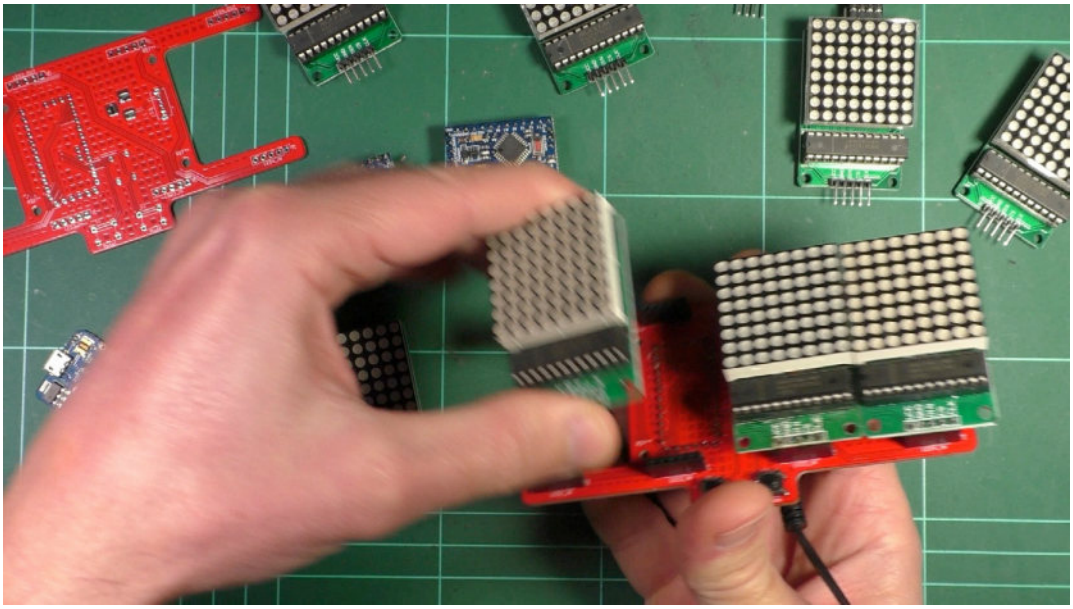


Figure 10.6.3: Attaching an LED matrix display to the board.

The back of the PCB contains the Arduino Pro Mini, barrel connector, slide switch, and two resistors.

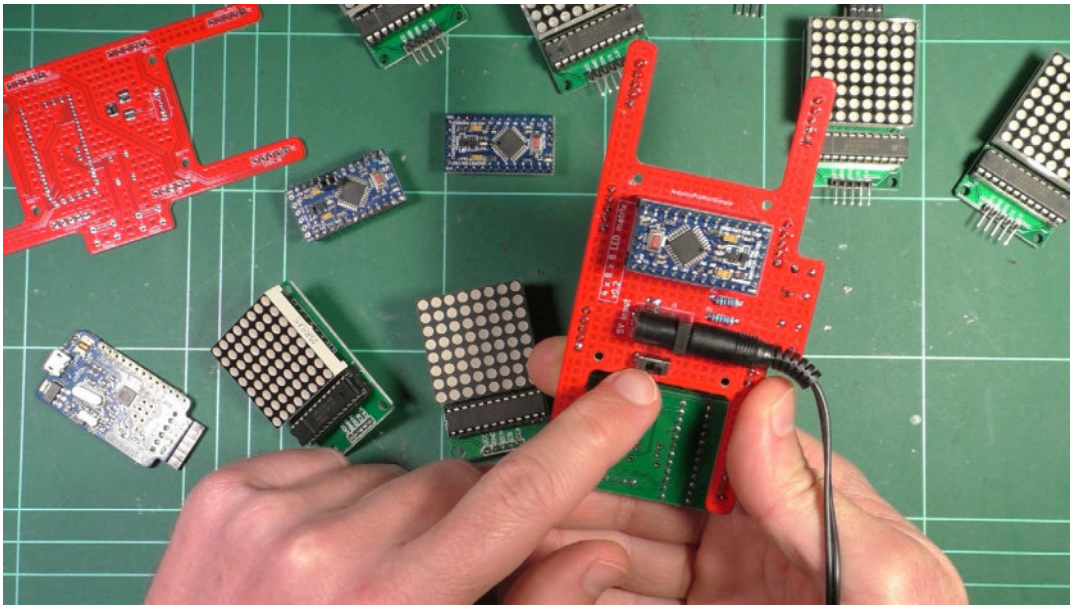


Figure 10.6.4: The back of the PCB.

For the Arduino Pro Mini, I used female headers to detach the Arduino for programming. The Arduino Pro Mini does not contain a programming USB interface, so I used an external USB to UART module for programming.

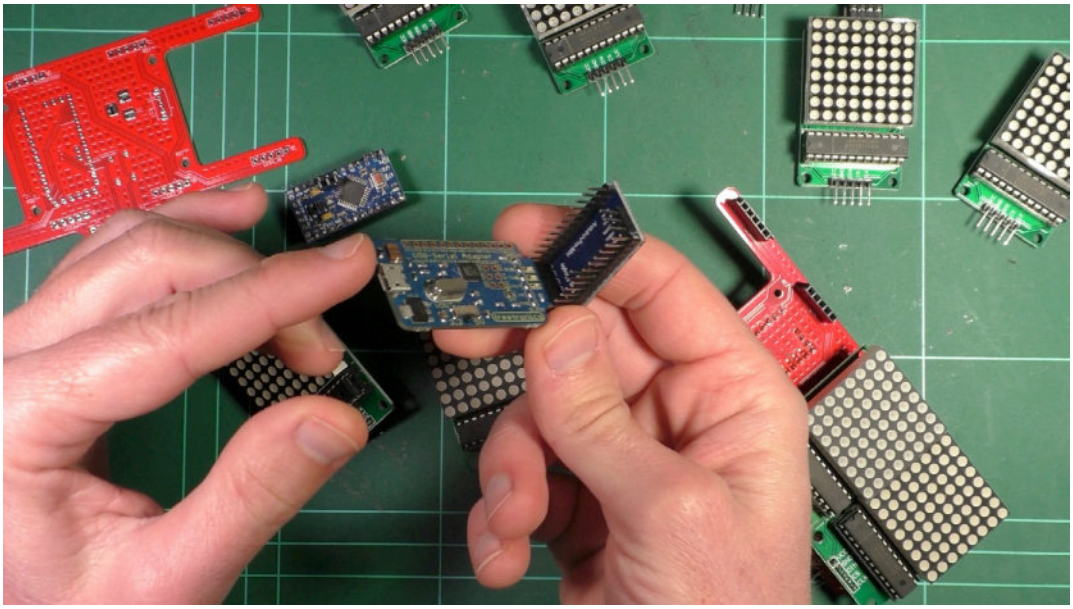


Figure 10.6.5: Programming the Arduino Pro Mini via a USB-UART adaptor.

Here's a view of the final working PCB running this sketch (find it on Github):

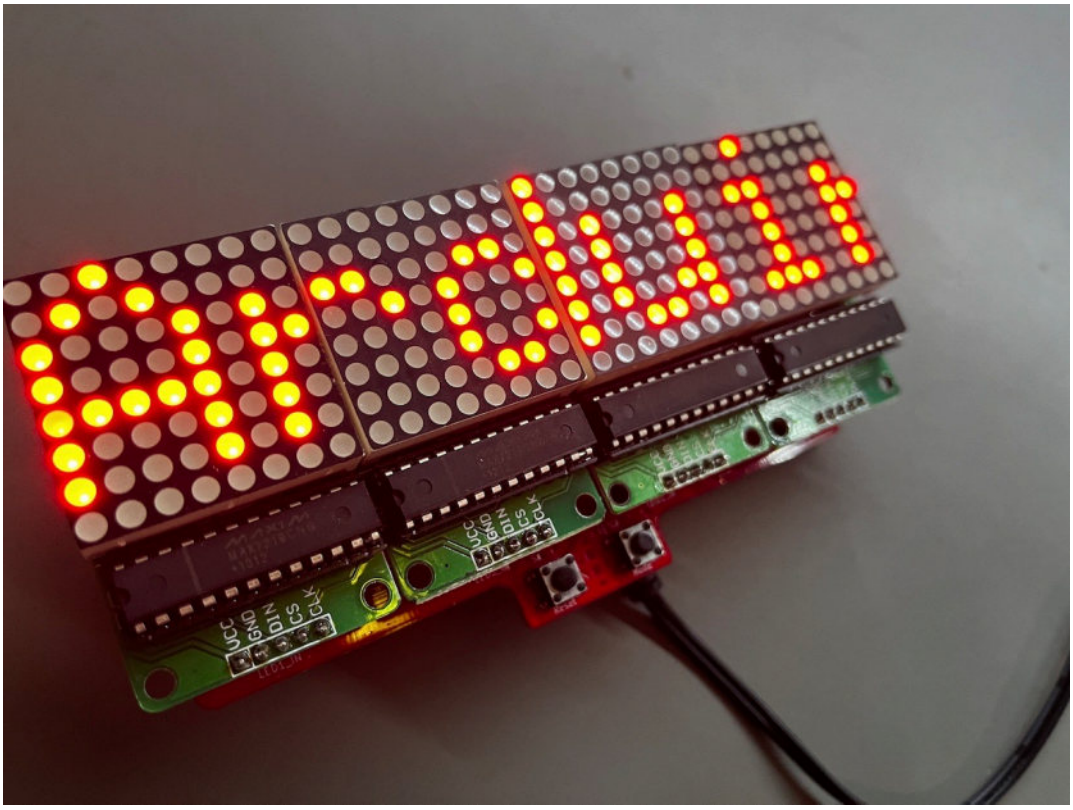


Figure 10.6.6: The final working PCB.

Part 11 : Project - MCU datalogger

1. Project - Introduction

Welcome to Part 11. In this Part, you will design a printed circuit board for a microcontroller data logger. The data logger is based on an Atmega 328P-AU microcontroller and is supported by two EEPROMs and a real-time clock. Additional components on the board, such as status LEDs with their supporting resistors, two crystal oscillators, connectors, and capacitors.

You will use SMD packages for most components on a rectangular two-layer board with mounting holes on the four corners.

The project highlights are:

1. You will use Git to capture the history of the project's development.
2. You will design two versions of the PCB: one with two layers and one with four layers. Both will use data from the same schematic design.

KiCad, on its own, does not allow the creation of more than one layout for a schematic. Git makes this possible with the use of branches. This project will allow you to practice this aspect of Git-powered PCB design with KiCad.

The schematic design contains components distributed across two sheets. You can see the final schematic below (sheet 1):

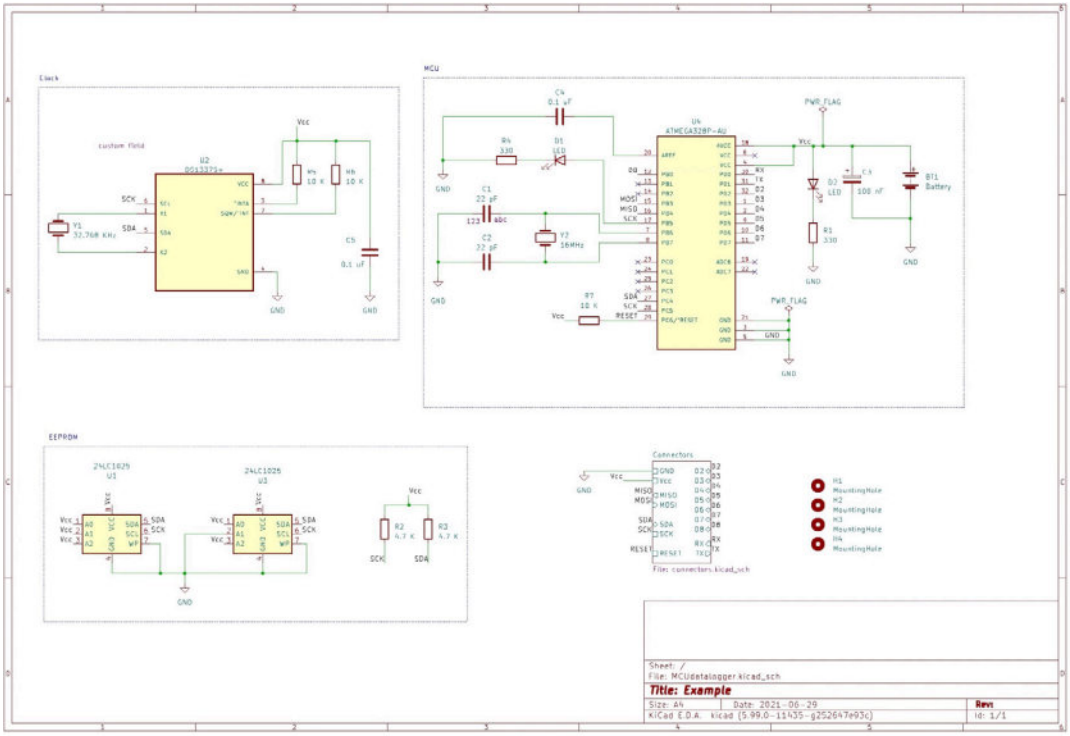


Figure 11.1.1: Sheet 1 of the project's final schematic design.

In Sheet 1, I have placed the main components of the board. Sheet 2 contains the connectors:

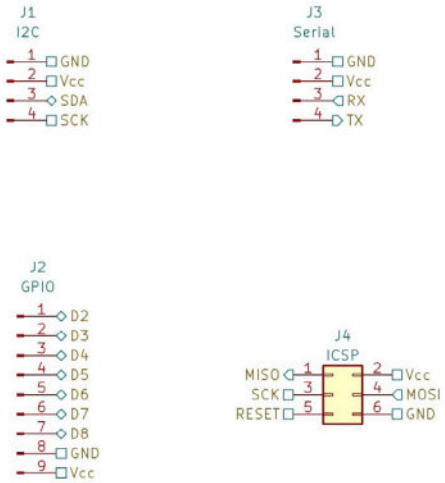


Figure 11.1.2: Sheet 2 of the project's final schematic design.

In the schematic design, I have used a combination of line wires and net labels. Other than the distribution of the components across the two sheets, the techniques I have used to draw the schematic should be familiar to you from previous projects.

The most exciting aspect of this project is the layout design: you will design two versions of the PCB. A two-layer and a four-layer version. You can see the final version of the two-layer PCB layout below:

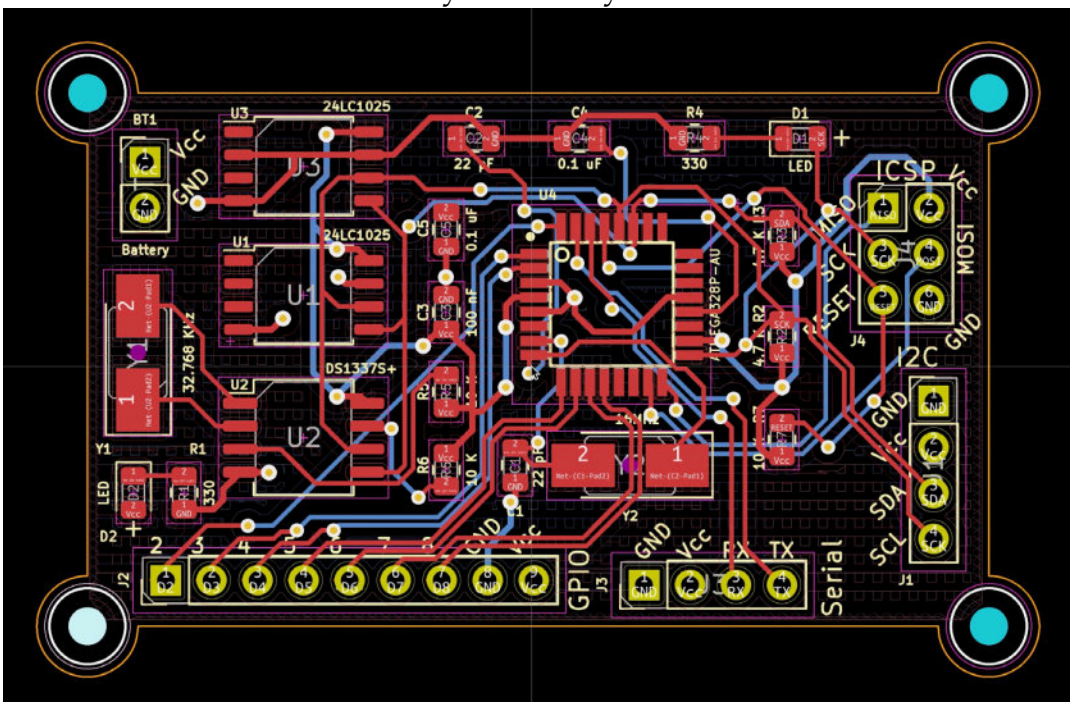


Figure 11.1.3: The project's final layout design (two layers).

You can see the final four-layer PCB below:

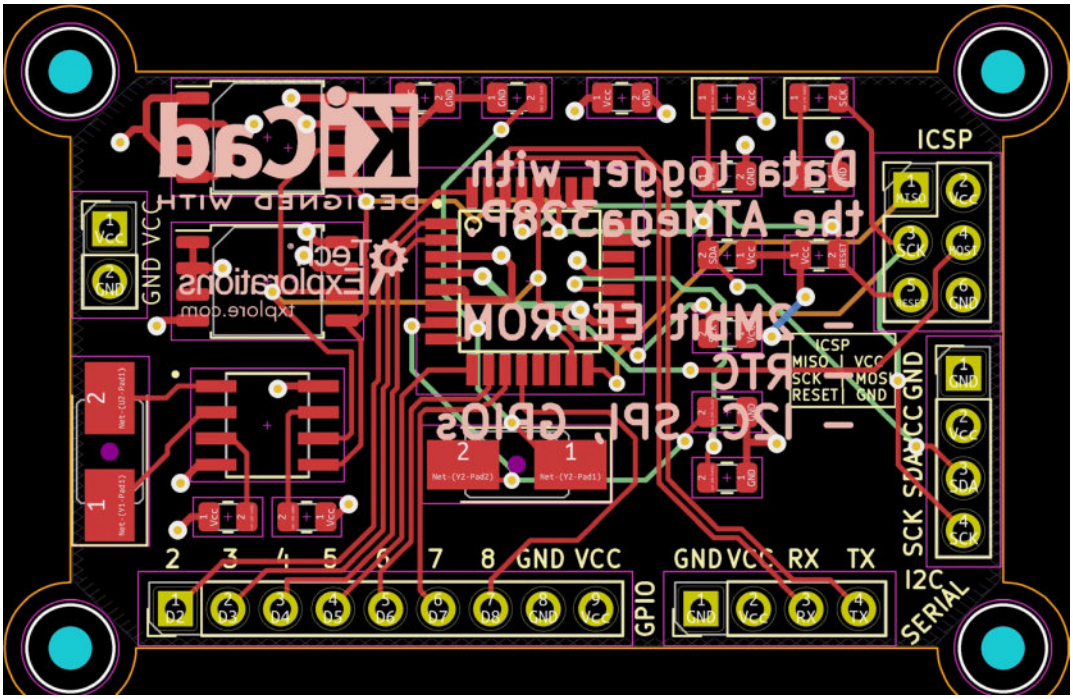


Figure 11.1.4: The project's final layout design (four layers).

The main objectives of this project are:

1. To help you practice skills you acquired in previous projects.
2. To use Git in a non-trivial project to extend KiCad's use cases in a single-schematic and multi-layout project.
3. To gain experience in creating multi-layer PCBs.

Below you can see the Bill of Materials for this project, as I have extracted it from the KiCad project (learn how later in this book):

Reference	Value	Footprint
BT1	Battery	Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
C1, C4	0.1uF	Capacitor_SMD:C_0805_2012Metric
C2, C3	22pF	Capacitor_SMD:C_0805_2012Metric
C5	100nF	Capacitor_SMD:C_0805_2012Metric
D1, D2	LED	LED_SMD:LED_0805_2012Metric
H1-H4	MountingHole	MountingHole:MountingHole_2.1mm
J2	Conn_01x09_Male	Connector_PinHeader_2.54mm:PinHeader_1x09_P2.54mm_Vertical

J1, J3	Conn_01x04_Male	Connector_PinHeader_2.54mm:PinHeader_1x04_P2.54mm_Vertical
J4	Conn_02x03_Odd_Even	Connector_PinHeader_2.54mm:PinHeader_2x03_P2.54mm_Vertical
R1, R2, R6	10K	Resistor_SMD:R_0805_2012Metric
R3, R4	4.7K	Resistor_SMD:R_0805_2012Metric
R5, R7	330	Resistor_SMD:R_0805_2012Metric
U2	DS1337S+	Footprints:SOIC127P600X175-8N
U1, U3	24LC1025	Package_SO:SOIC-8_5.23x5.23mm_P1.27mm
U4	ATMEGA328P-AU	Footprints:QFP80P900X900X120-32N
Y1	32.768 KHz	Crystal:Crystal_SMD_5032-2Pin_5.0x3.2mm_HandSoldering
Y2	16 MHz	Crystal:Crystal_SMD_5032-2Pin_5.0x3.2mm_HandSoldering

Table 11.1.1: The Bill of Materials for this project.

I used Snapeda to find the symbol-footprint pairs for U4. You should be able to find all other symbols and footprints in KiCad's libraries.

In the next chapter, you will begin work on this project by creating a new KiCad project and Git repository.

2. Create the new project and Git repository

A primary objective of this project is to help you learn how to use Git within a KiCad project. If you are not familiar with Git, please consider reading the chapter “25. KiCad project management with Git” in the Recipes part of this book before continuing with the project. To keep the project flowing, I will not explain the meaning of each Git command that I use. Instead, I assume that you already know the basics of Git, and in the chapters that follow, I will focus on showing you how to use this knowledge in the context of a complete KiCad project.

Start KiCad, and create a new KiCad project. I have named my instance of the project "MCU Datalogger."

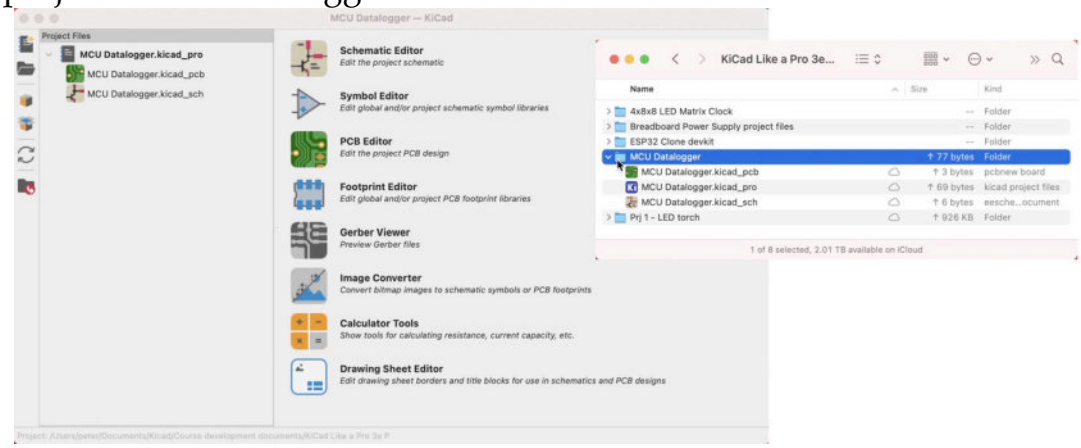


Figure 11.2.1: Starting a new project.

Now that you have a project directory for the project files create a new Git repository. Open a terminal window and navigate to the project directory. Issue the "init" command to initialize a new repository:

```
% git init
```

Below is my command line session. The second rectangle points to the new ".git" hidden directory, which will contain the project's history.

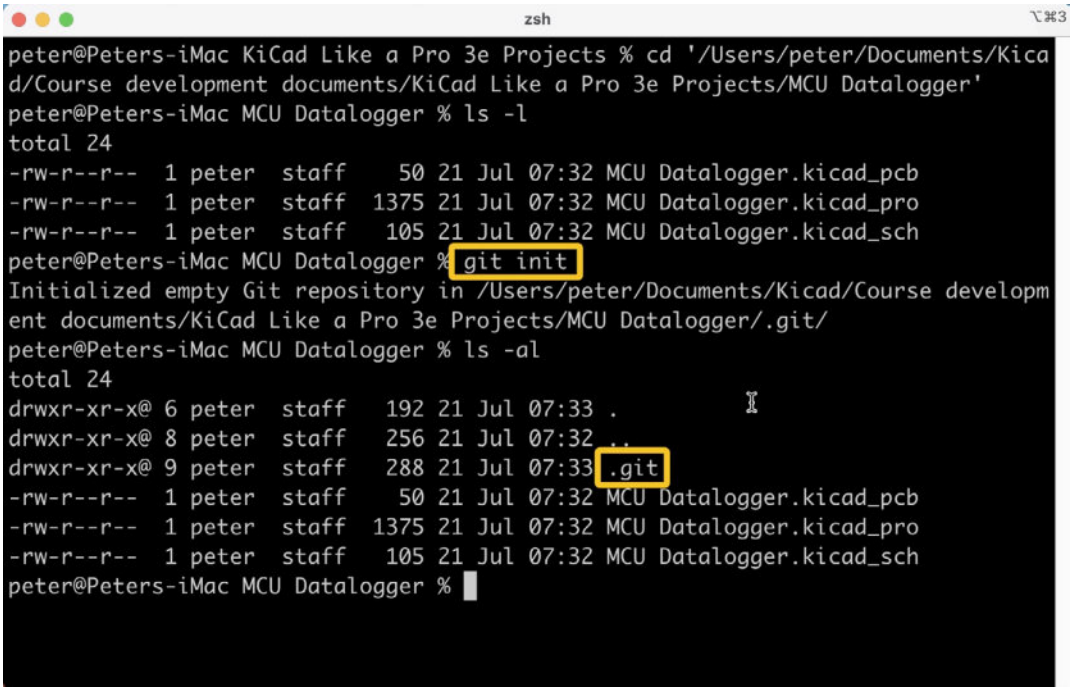


Figure 11.2.2: Creating a new Git repository.

The new Git repository is ready. Let's start tracking the project files. Get the status of the repository to see what is not being tracked:

```
% git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    MCU Datalogger.kicad_pcb
    MCU Datalogger.kicad_pro
    MCU Datalogger.kicad_sch

nothing added to commit but untracked files present (use "git
add" to track)
%
```

The repository is not tracking any files yet, and there are no commits. The working branch is "master." Three files are not being tracked. Let's track them. Use the "add" command:

```
% git add .
```

You can use "git status" again to confirm that the three files are in the staging area but not yet committed. Go ahead and commit them using the "commit" command:

```
% git commit -am "First commit of new project."

[master (root-commit) c268ef2] First commit of new project.
3 Files changed, 68 insertions(+) E
create mode 100644 MCU Datalogger.kicad_pcb
create mode 100644 MCU Datalogger.kicad_pro
create mode 100644 MCU Datalogger.kicad_sch
%
```

Use the "status" command to confirm that the working tree is clean:

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

KiCad automatically creates backup and cache files that don't have to be captured in the repository. I also want to exclude the contents of the Gerbers directory since I can always generate them at will. To exclude those files, use your text editor to edit the" .gitignore" file. Below you can see the contents of ".gitignore":

```
fp-inFo-cache
MCU Datalogger-backups/*
MCUDataloggerGerber/*
```

You can see the contents of my ".gitignore" file below:

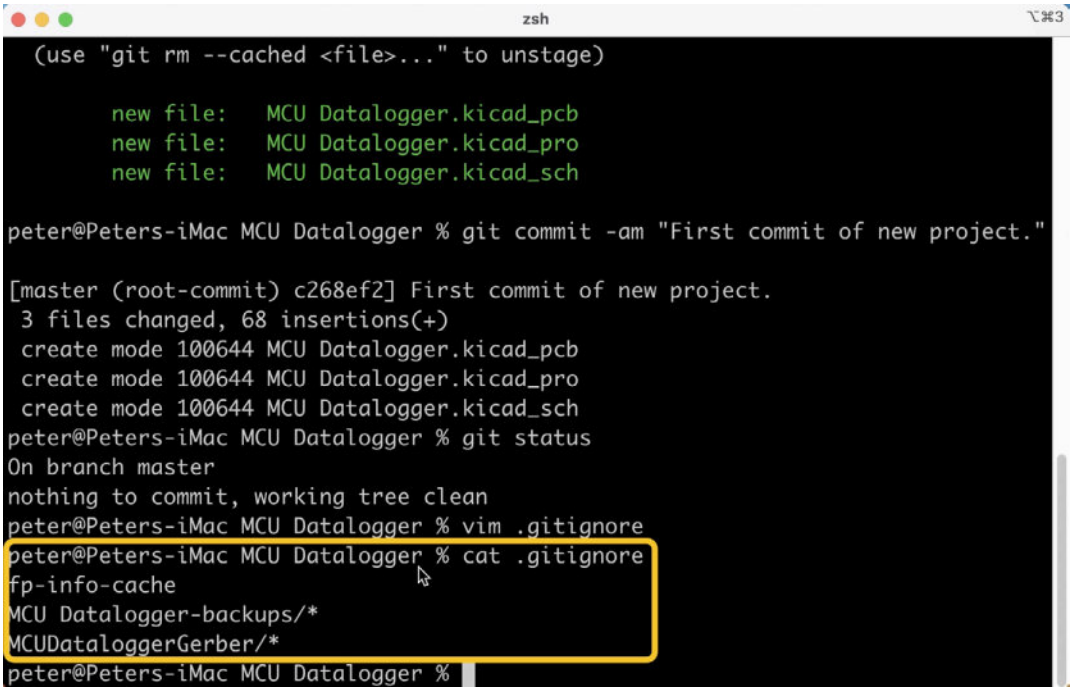


Figure 11.2.3: The contents of the ".gitignore" file.

At this point, you have a new KiCad project, and you are tracking it using Git. Let's continue with the schematic design workflow.

3. Schematic design

In this chapter, you will complete the schematic design of this PCB by following the schematic design workflow. You learned about this workflow in Part 6 of the book.

3.1. Schema 1 - Setup

In this chapter, you will set up your schematic editor. Begin by opening the Preference window (KiCad —> Preferences). I have kept the defaults, except for the following settings:

- Schematic Editor
  - Display Options
    - Grid thickness: 1 px
    - Min Grid spacing: 15 px
  - Colors: Using a custom theme with white background.
  - Field Name Templates: A new field named "Purpose"; visible.
- Schematic Setup
  - Project
    - Net Classes: create a new net class with the name "Power."
    - Text Variables: created a new variable:
      - Variable Name: project_name
      - Text Substitution: MCU Datalogger with memory and clock
- Page Settings:
  - Issue Date: copied today's date from the date field.
  - Revision: 1
  - Title: ${project_name}

The setup is complete. Don't forget to save the project.
Let's continue with updating the Git repository. Use the "status" command to get an update:

```
% 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: MCU Datalogger.kicad_pro
     modified: MCU Datalogger.kicad_sch

Untracked files:
  (use "git add <File>..." to include in what will be committed)

     .DS_Store
     .gitignore
     MC Datalogger.kicad_prl

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

Git reports that there are changes in the project and schematic files and three new untracked files. I do not want to track the Mac OS system file ".DS_Store" so I will add it to the ".gitignore" file. The other two files should be tracked. Here is my updated ".gitignore" file:

```
.DS_Store
fp-inFo-cache
MCU Datalogger-backups/*
MCUDataloggerGerber/*
```

Add the new files to Git (using the file name instead of the wildcard "." character), and commit them:

```
% git add .gitignore
% git add “MCU Datalogger.kicad_prl”
% git commit -am "Setup of EEschema for new project."
[master ddb72ca] Setup of EEschema for new project.
4 files changed, 391 insertions(+), 9 deletions(-)
create mode 100644 .gitignore
create mode 100644 MCU Datalogger.kicad_prl
%
```

Use the “log” command to see Git’s recent history:

```
% git log
commit ddb72ca7e5e8778Fcb7c0b1e8bf480f262635a6a (HEAD -> master)
Author: Peter Dalmaris <peter@txplore.com>
Date: Wed Jul 21 07:51:26 2021 +1000
```

```
    Setup of EEschema for new project.

commit c268ef258069718c44511d36a8ed03d4038570c9
Author: Peter Dalmaris <peter@txplore.com>
Date: Wed Jul 21 07:35:12 2021 +1000

    First commit of new project.
%
```

The log shows the two commits done so far, along with their commit IDs and other information.

In the previous chapter, where I created the new Git repository, I forgot to change the name of the "master" branch into "main." The naming convention for the primary branch of Git repositories is now "main." Online repositories, like Github, also follow this convention. If you plan to share your KiCad project with other people using Github, you should consider changing the primary branch of your project repository to "main" to avoid compatibility issues. I will make the name change now, so I don't forget again:

```
% git branch
* master
% git branch -m master main
% git branch
* main
%
```

In the session above, I used the "branch" command to check the name of the current working branch. The response was "master." In the third line, I used "branch -m" to rename the "master" branch into "main." In the fourth line, I confirmed that the renaming worked.

If (like me) you tend to forget to remake your Git repository's default branch, you can set it in the Git configuration like this:

```
% git config --global init.defaultBranch main
```

In the next segment of this chapter, you will add the schematic symbols into the editor.

3.2. Schema 2 - Symbols

Let's add the schematic symbols to the editor sheet. Use the BoM table from the introduction chapter to help you find the required symbols. Add all

symbols except for the headers. I plan to add the symbols in a second sheet in the next segment of this chapter.

Download the files from Snapeda (all three types: symbols, footprints, and 3D models if available), and store them in your project folder. Install them as I describe in the relevant chapter from Part 7. You should be able to find all symbols in KiCad's symbol libraries, except for those for the microcontroller ("U4") and the real-time clock ("U2"). I installed these libraries in my Project Specific Libraries tab:

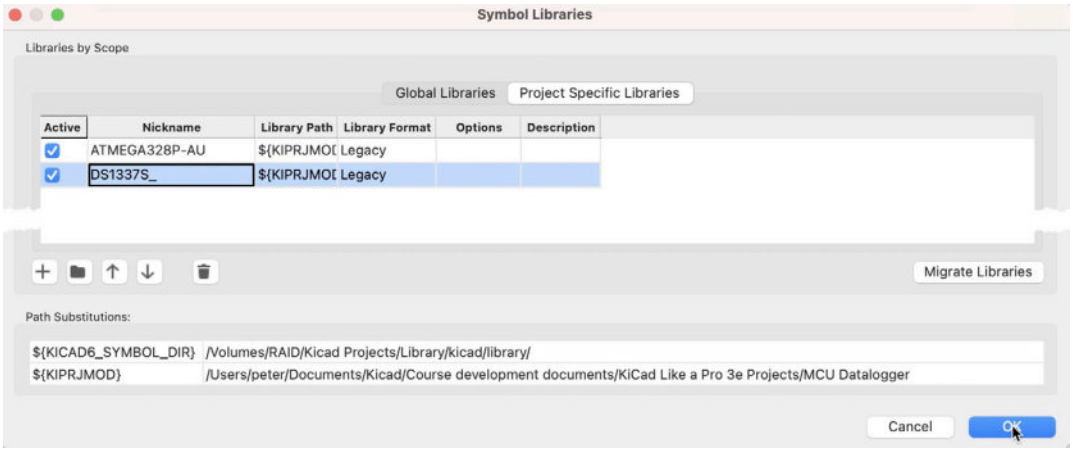


Figure 11.3.2.1: Changes to the Installed external symbol libraries.

After adding all symbols (except for the headers), your schematic editor sheet will look like this:

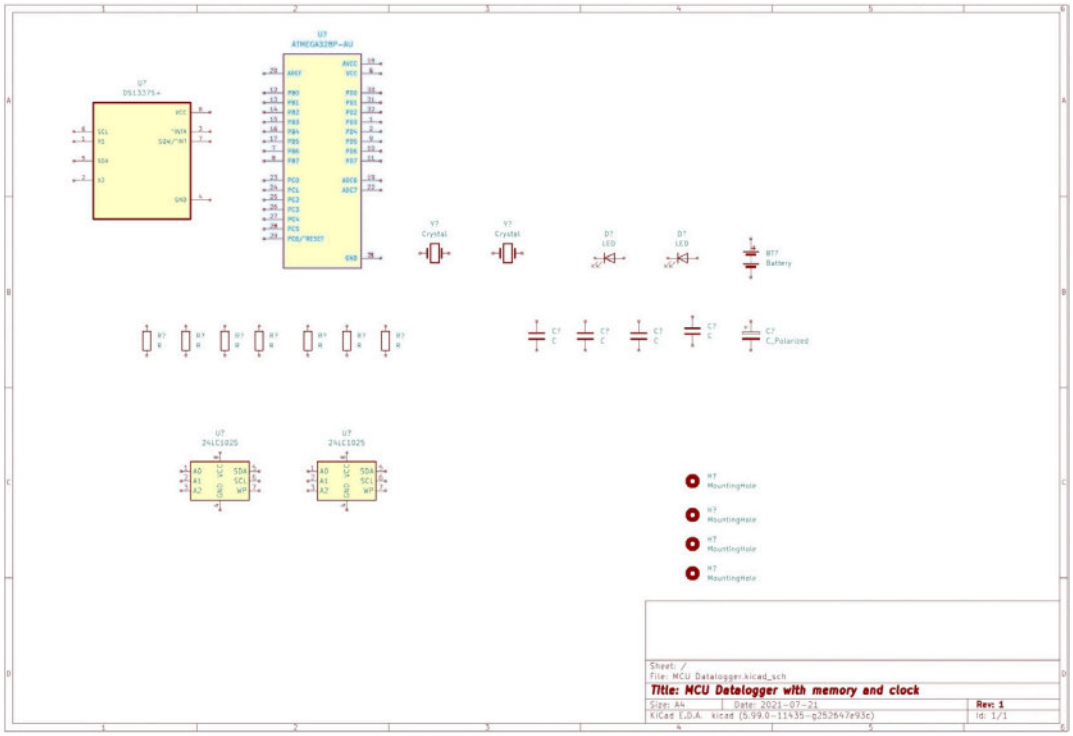


Figure 11.3.2.2: The first sheet of the schematic design.

For the remaining symbols, the headers, you will create a second sheet. You will do this in the next segment of this chapter.

3.3. Schema 2 - Sheet two

To create a new sheet to contain the connector symbols, create a hierarchical sheet. Select the hierarchical sheet tool from the right toolbar, and type in an appropriate name for the sheet and its file. I called mine "Connectors" and "connectors.kicad_sch," respectively.

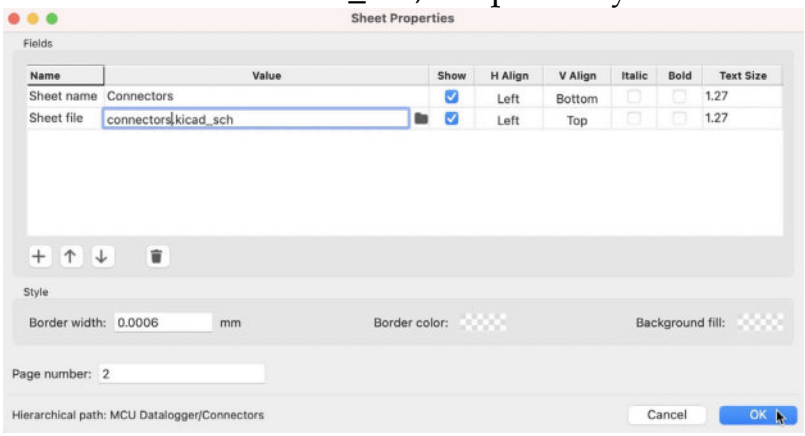


Figure 11.3.3.3: Creating a hierarchical sheet.

Place the hierarchical sheet symbol on the lower-right side of the schematic:

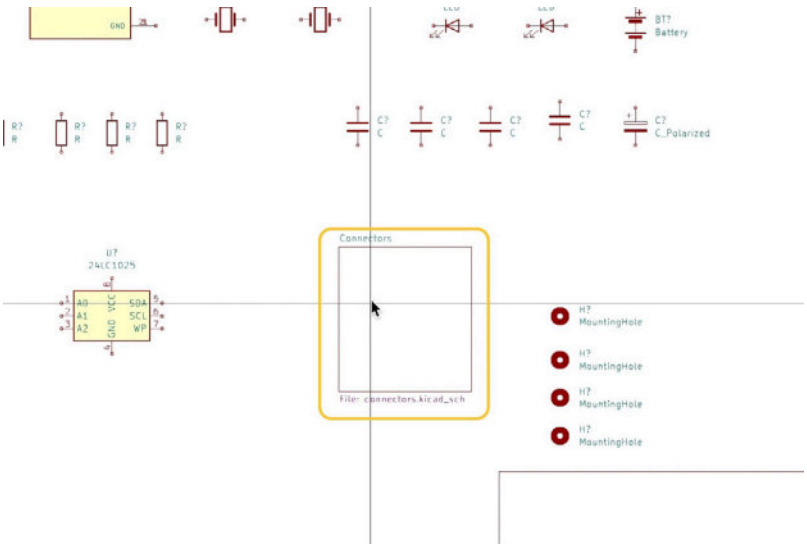


Figure 11.3.3.4: The new hierarchical sheet symbol.

Double click on the hierarchical sheet button to enter the new sheet in the editor. Add the final four symbols for the connectors. The connectors sheet will look like this:

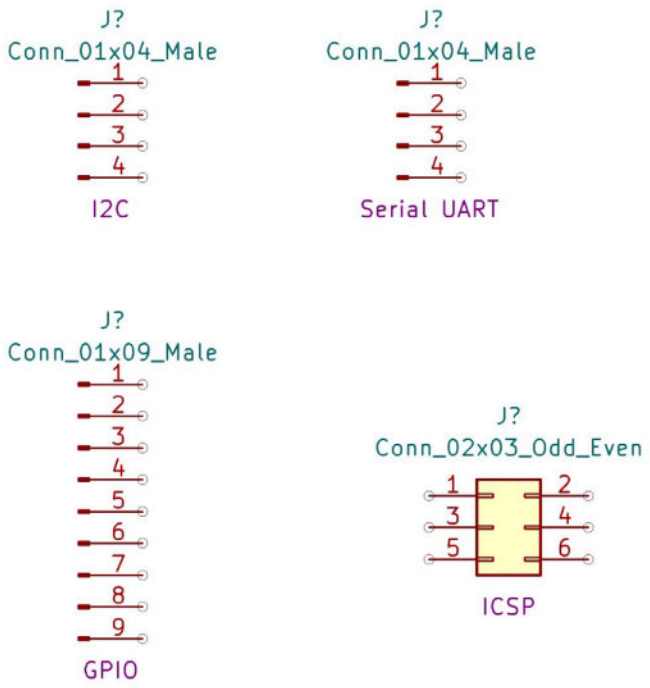


Figure 11.3.3.5: The contents of the “Connectors” sheet.

At this point, all the symbols are in the schematic editor. This is an excellent opportunity to capture the progress in the Git repository. Save your work with KiCad, and check the current status of the repository:

```
% git status
On branch main
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: MCU Datalogger.kicad_pro
      modified: MCU Datalogger.kicad_sch

Untracked files:
   (use ”git add <file>..." to include in what will be committed)

      Libraries/
      connectors.kicad_sch
      sym-lib-table

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

Git reports that two tracked files have changed and a new directory ("Libraries"), and three other files are not being tracked. Start tracking for the three new files, and commit the changes:

```
% git add .
% git commit –am "Have added required symbols to the schematic
sheets."
[main f488540] Have added required symbols to the schematic sheets.
 15 Files changed, 49834 insertions(+)
 create mode 100644 Libraries/3D/D81337S.STEP
 create mode 100644 Libraries/3D/atmega328p–au.stp
 create mode 100644 Libraries/ATMEGA3Z8P–AU/ATMEGA3Z8P–AU.lib
 create mode 100644 Libraries/ATMEGA328P–AU/ATMEGA3Z8P–AU.step
 create mode 100644 Libraries/ATMEGA328P–AU/
QFP80P900X900X1Z0–3ZN.kicad_mod
 create mode 100644 Libraries/ATMEGA328P–AU/how–to–import.htm
 create mode 100644 Libraries/ATMega328P–edited.kicad_sym
 create mode 100644 Libraries/DSl337S_/D51337S_.lib
 create mode 100644 Libraries/DSl337S_/D51337S_.step
 create mode 100644 Libraries/DSl337S_/SOIC127P600X175–8N.kicad_mod
 create mode 100644 Libraries/DSl337S_/how–to–import.htm
 create mode 100644 connectors.kicad_sch
 create mode 100644 sym–lib–table
%
```

Use "git status" to confirm that there are no pending commits.

3.4. Schema 3 - Arrange, Annotate

In this segment, you will arrange the symbols on the sheet to prepare them for wiring in step four. After that, you'll use the automated annotator to set identifiers for each symbol.

I will arrange the symbols according to functional groups:

1. The microcontroller and power group.
2. The real-time clock group.
3. The EEPROM group.
4. The mounting holes group.
5. The connectors.

Below is the final arrangement of the symbols in the root sheet before the automatic annotation:

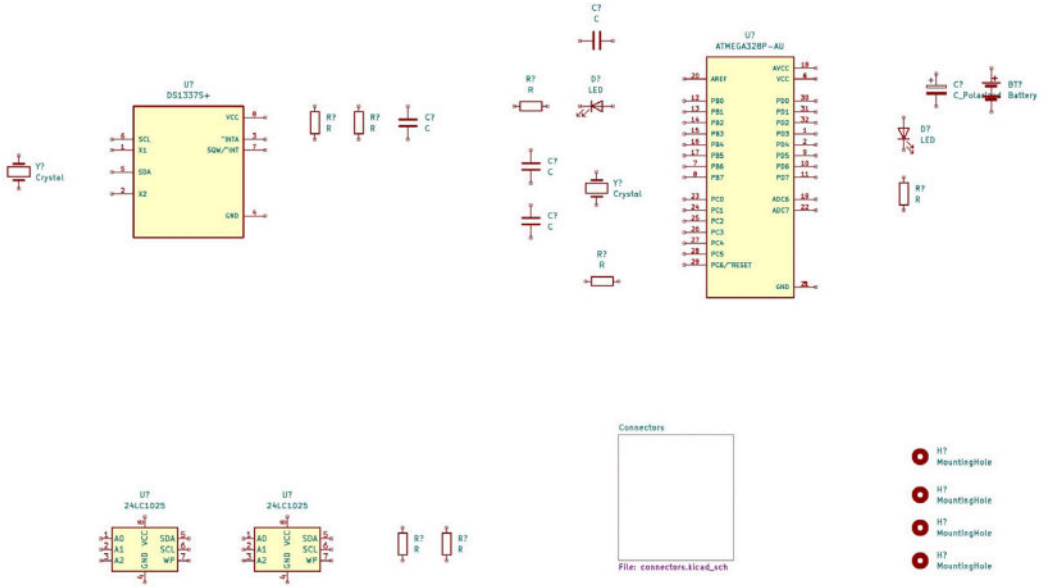


Figure 11.3.4.6: Final arrangement of symbols in the root sheet.

Bring up the annotator tool, and using the default settings, annotate the schematic. The result is below:

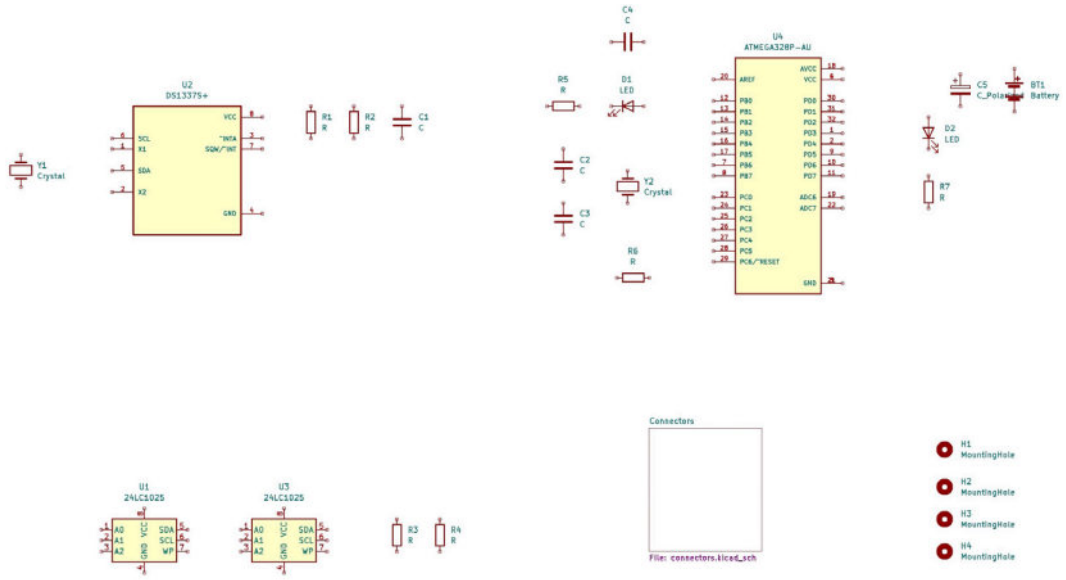


Figure 11.3.4.7: Annotated symbols in the root sheet.

This is an opportunity to commit the change to the Git repository. Here is my session on the command line:

```
% git status
On branch main
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: MCU Datalogger.kicad_sch
        modified: connectors.kicad_sch

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Changed locations of symbols (arrange)."
[main 41550e3] Changed locations of symbols (arrange).
  2 files changed, 215 insertions(+), 215 deletions(-)
%
```

With the change committed, you can continue with the next step of the project, adding the component values.

3.5. Edit component values

In this chapter segment, you will add the values for components such as the capacitors and the resistors. Use the information in the BOM table in the introduction of this project as a source. You can use one of these methods to edit the component values:

1. Open each symbol's Properties window (double-click on a symbol), and type the value in the Value field, as in the example below:

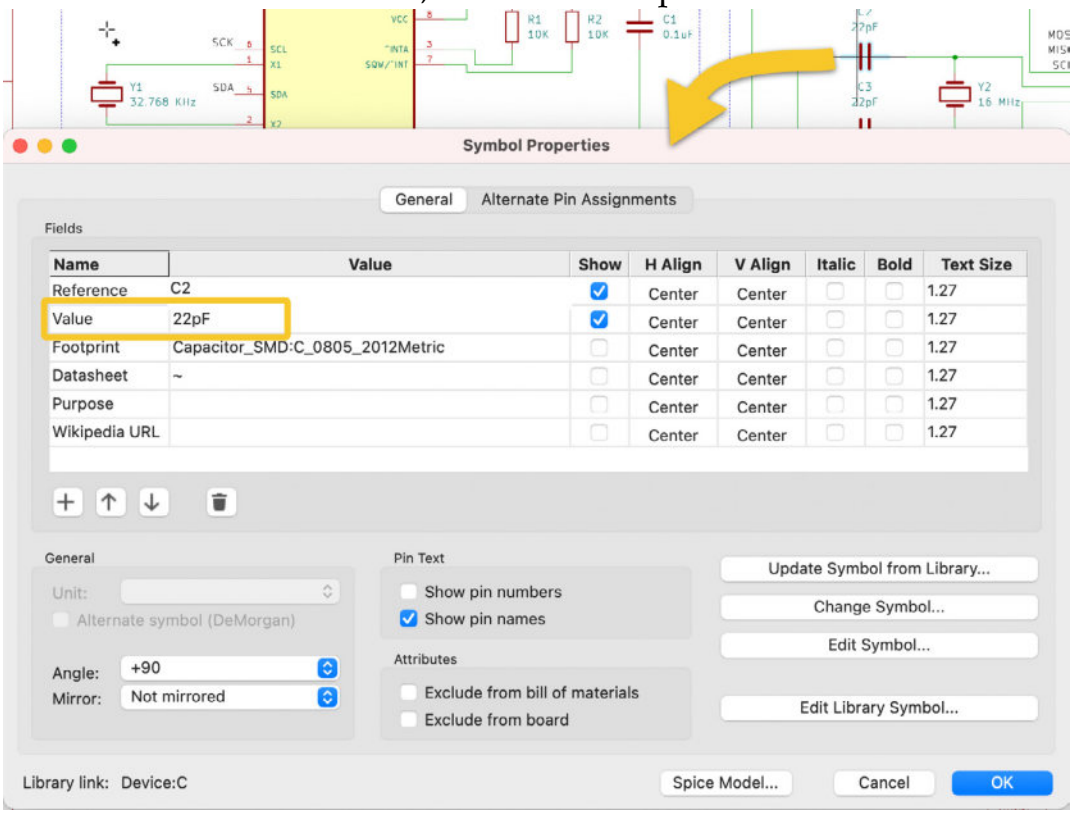


Figure 11.3.5.8: Editing the value field of a symbol.

2. Edit the symbol value fields in bulk using the Symbol Fields Table. Click on the Fields Table button from the top toolbar to open the table window, as in the example below:

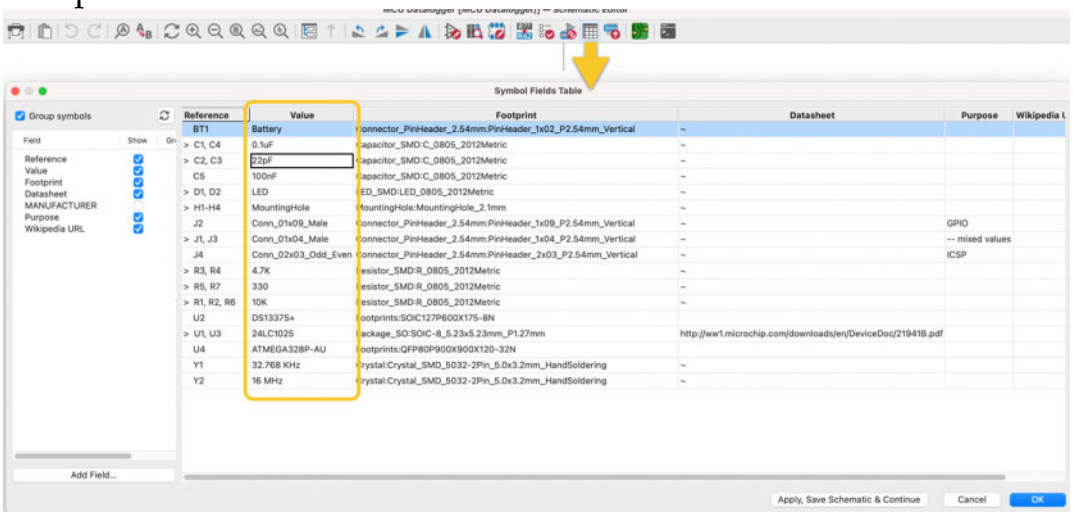


Figure 11.3.5.9: Editing the value field of all symbols using the Symbol Fields Table.

By the end of this process, the values for each of the symbols in the schematic should look like those in Figure 11.3.5.9.

Let's update the Git repository:

```
% git status
On branch main
Changes not staged For commit:
  (use "git add <Fil e>..." to update what will be committed)
  (use "git checkout —— <file>..." to discard changes in working
directory)

      modified: MCU Datalogger.kicad_sch

no changes added to commit (use "git add" and/or "git commit —a")
% git commit —am "Added component values."
[main Z90fba6] Added component values.
  1 file changed, 28 insertions(+), 28 deletions(—)
%
```

Let's continue with the "Associate "part of Step 3 of the schematic workflow.

3.6. Schema 3 - Associate

In this step, you will associate the symbols with their footprint counterparts. As with the editing of the values that you finished in the previous segment, you can choose to associate symbols with footprints in two ways:

1. By editing the Footprint field in the symbol's Properties window.

2. By doing the associations in bulk, using the "associations" tool.

Since there are several symbols to work with, I suggest that you use the bulk option. Draw from the information present in the BOM table in the introductory chapter. The "Footprint" column contains the name of the footprint to look for in the footprints library.

Open the Assign Footprints window by clicking on the "associations" button in the top toolbar. The middle pane of the window contains the current associations and will be mostly blank (see below).

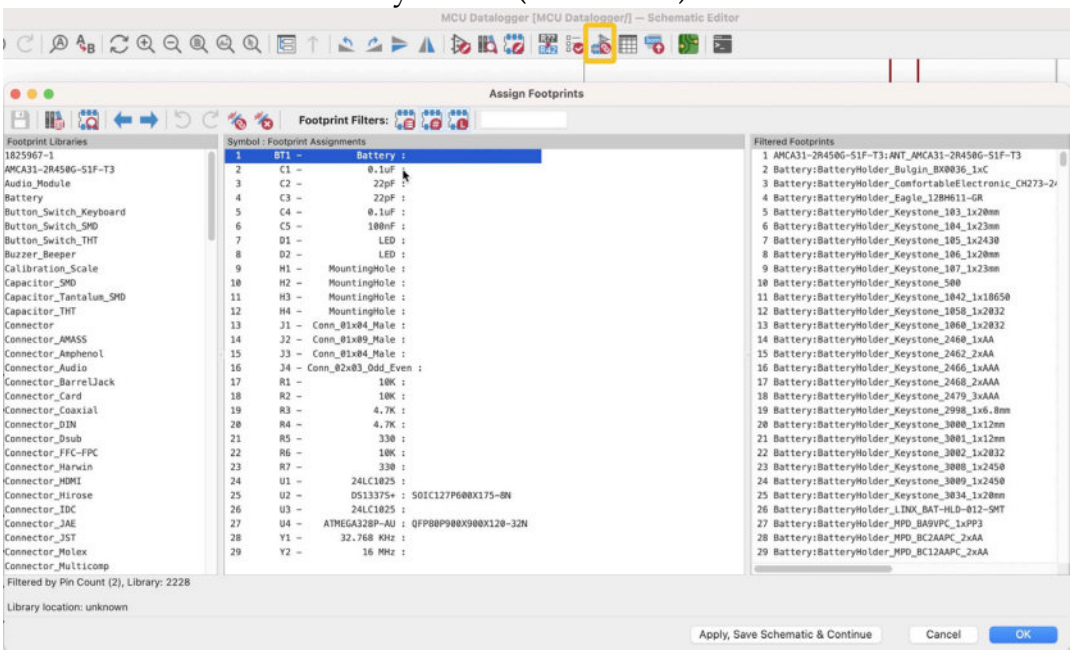


Figure 11.3.6.10: Staring to set the symbol-footprint associations.

Follow the process you learned earlier in this book to associate each symbol with a footprint. You can find the footprint names in the BoM table in the introduction. The final Assign Footprints window should look like this:

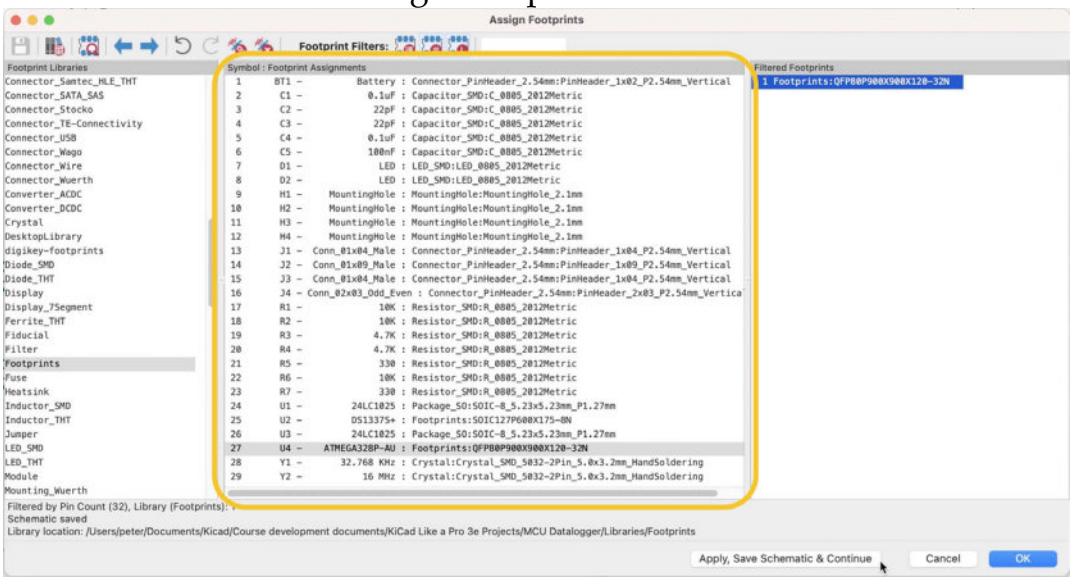


Figure 11.3.6.11: The final symbol-footprint associations.

Let's complete this step by committing the changes to the Git repository:

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

         deleted: Libraries/ATMEGA3Z8P-AU/QFP80P900X900X120-
3ZN.kicad_mod
         deleted: Libraries/ATMega3Z8P-edited.kicad_sym
         deleted: Libraries/D51337S_/SOIC127P600X175-8N.kicad_mod
         modified: MCU Datalogger.kicad_prl
         modified: MCU Datalogger.kicad_pro
         modified: MCU Datalogger kicad_sch
         modified: connectors.kicad_sch

   Untracked files:
     (use "git add <file>..." to include in what will be committed)

         Libraries/Footprints/
         fp-lib-table
   no changes added to commit (use "git add" and/or "git commit -a")
   % git add .
   % git commit -am "Completed symbol-footprint associations."
   [main d52702e] Completed symbol-Footprint associations.
   8 files changed, 158 insertions(+), 64 deletions(-)
   rename Libraries/{ => Footprints}/ATMega328P-edited.kicad_sym
(100%)
   rename Libraries/{ATMEGA328P-AU => Footprints}/QFP80P900X900X120-
32N.kicad_mod
   (100%)
   rename Libraries/{DSl337S_ => Footprints}/
SOIC127P600X17S-8N.kicad_mod (100%)
   create mode 100644 Fp-lib-table
   %
```

Let’s continue with the wiring, which is Step 4 in the schematic design workflow.

3.7. Schema 4 - Wiring of sheet 1

Because the schematic is broken into two sheets, I will do the wiring in two parts. First, I will wire the root schematic using a combination of graphical line wires and net labels. Then, I will connect the symbols in the Connectors sheet with those in the root sheet using hierarchical labels.

To make the wiring process systematic, I will wire symbols that belong to the same functional group before continuing with other components. I will start with the real-time clock group. When the wiring is complete, this group looks like this:

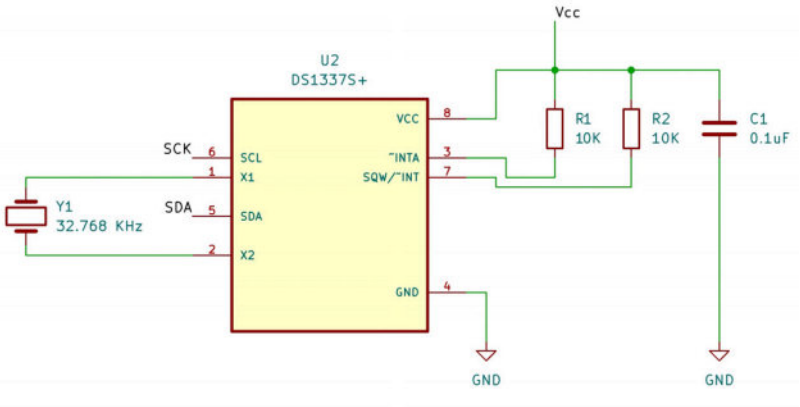


Figure 11.3.7.12: Real-time clock group wiring is complete.

Continue with the memory groups:

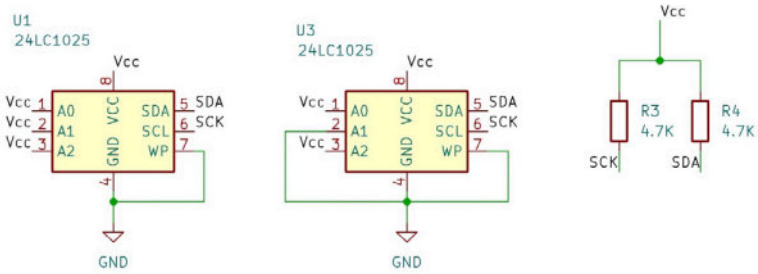


Figure 11.3.7.13: Memory group wiring is complete.

Next, the microcontroller and power group:

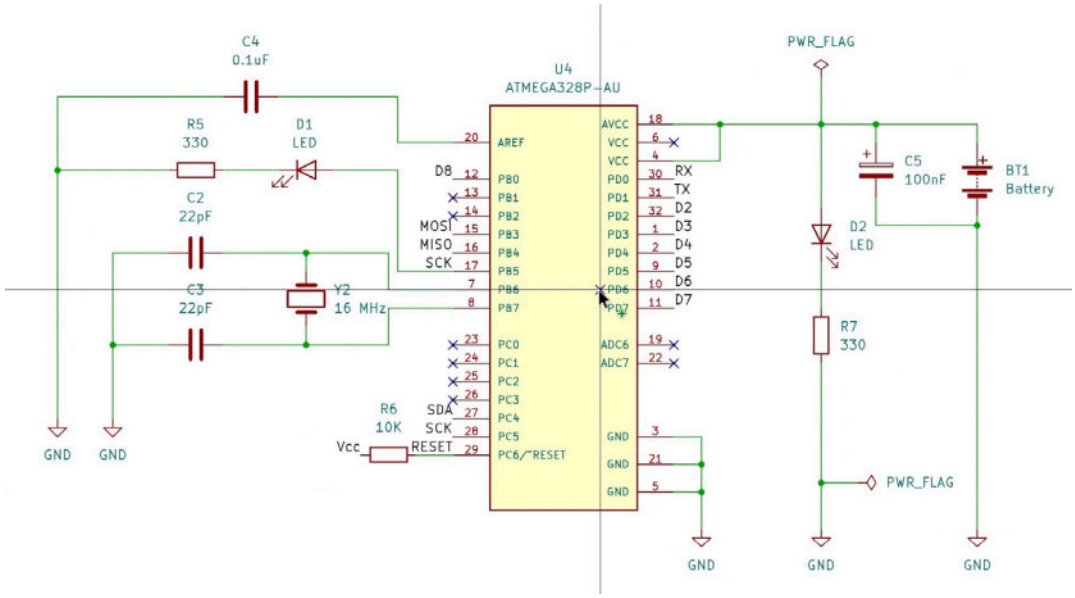


Figure 11.3.7.14: MCU and power group wiring are complete.

The wiring for the root sheet is complete. You can see it below:

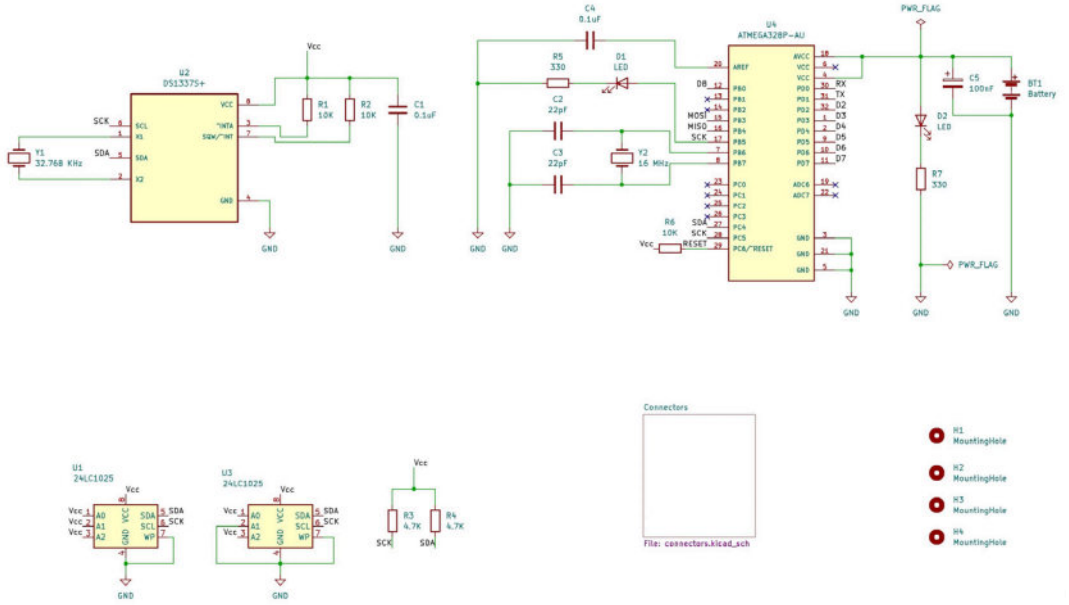


Figure 11.3.7.15: Root sheet wiring is complete.

The second sheet remains to be wired, but this is an excellent opportunity to commit the changes to the Git repository:

```
% git status
On branch main
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: MCU Dataiogger.kicad_sch

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Completed wiring of main sheet."
```

```
[main e0392b3] Completed wiring of main sheet.
  1 file changed, 887 insertions(+), 70 deletions(-)
%
```

Let's continue with the wiring of the Connectors sheet.

3.8. Schema 4 - Wiring of sheet 2

The wiring of the symbols in the Connectors sheet is pending. Let's complete it now.

The Connectors sheet contains four connector symbols. The pins on those symbols must be connected to pins on symbols in the root sheet. For this reason, I will not be using any graphical line wires. Instead, I'll be using a combination of net labels and hierarchical sheet labels.

Open the Connectors sheet by double-clicking on the hierarchical sheet symbol in the root sheet. To create a hierarchical label, click on the hierarchical label button in the right toolbar. In the Properties window that appears, give the label a name, and set its properties (see example below).

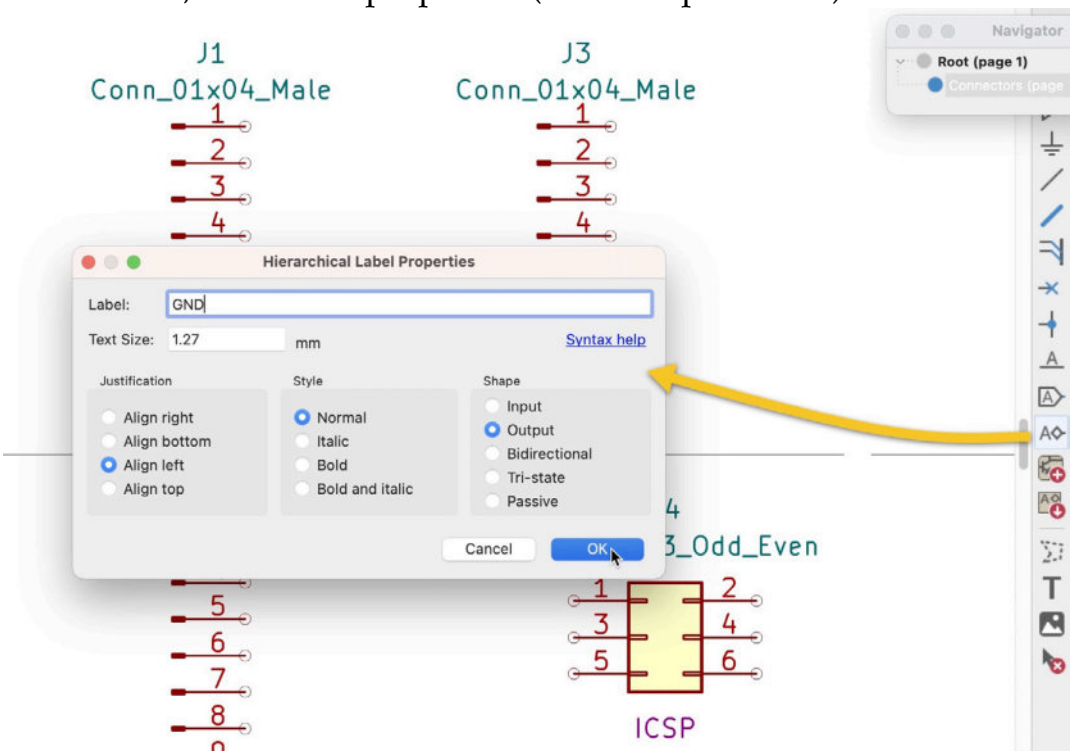


Figure 11.3.8.16: Creating a hierarchical label.

Attach this label to J1 pin 1, like this:

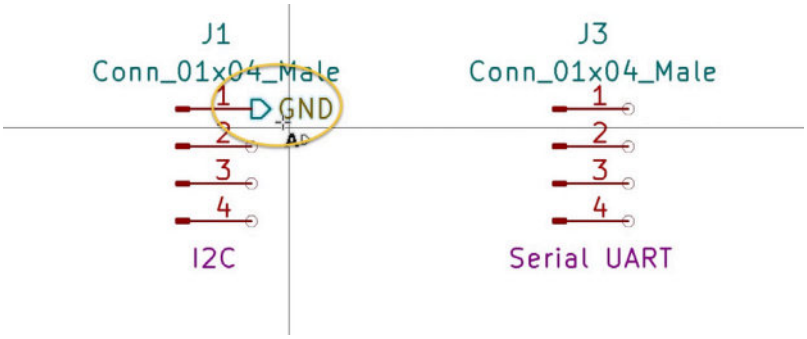


Figure 11.3.8.17: Attaching a hierarchical label to J1 pin 1.

Note that the symbol representing the hierarchical label depends on the Shape you chose in its Properties windows. The shape that you see in the example above represents an "output" label. As this pin is connected to the GND net, I will select the "passive" shape.

Continue in the same way until you have created hierarchical labels for all pins in this sheet. Once completed, the Connectors sheet will look like this:

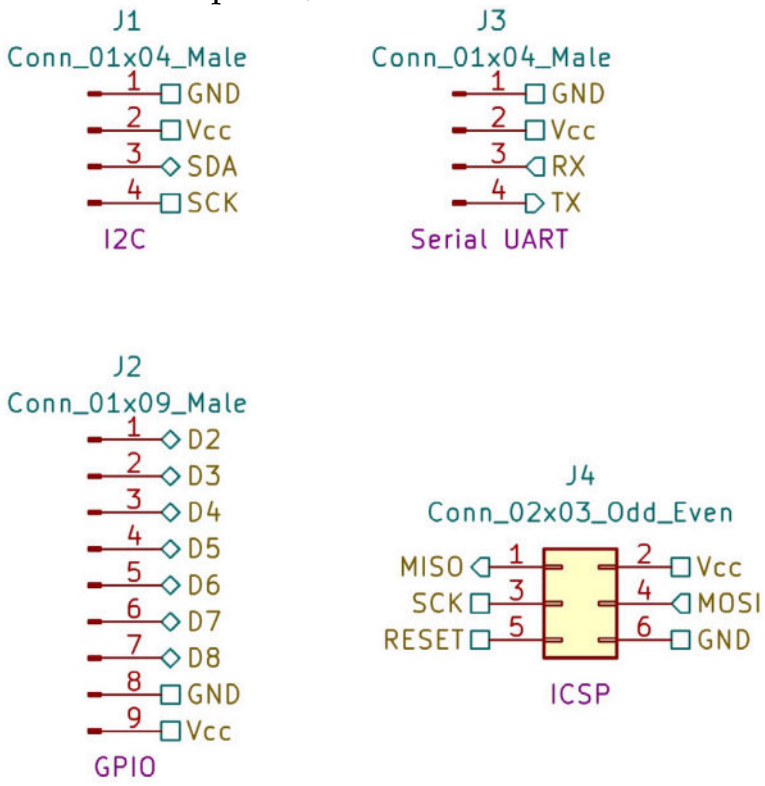


Figure 11.3.8.18: Hierarchical labels attached to all pins.

Save your work and open the root sheet. The hierarchical labels exist in the Connectors sheet. You will need to:

1. Expose those labels to the root sheet.

2. Connect the exposed hierarchical labels to other pins using either graphical line wires or net labels.

Use the “Add a hierarchical sheet (S)” tool from the right toolbar to complete step one. Click anywhere inside the hierarchical sheet box to drop one of the hierarchical labels and place it along the perimeter with the tool selected. You can see the first hierarchical label attached to the left of the box below:

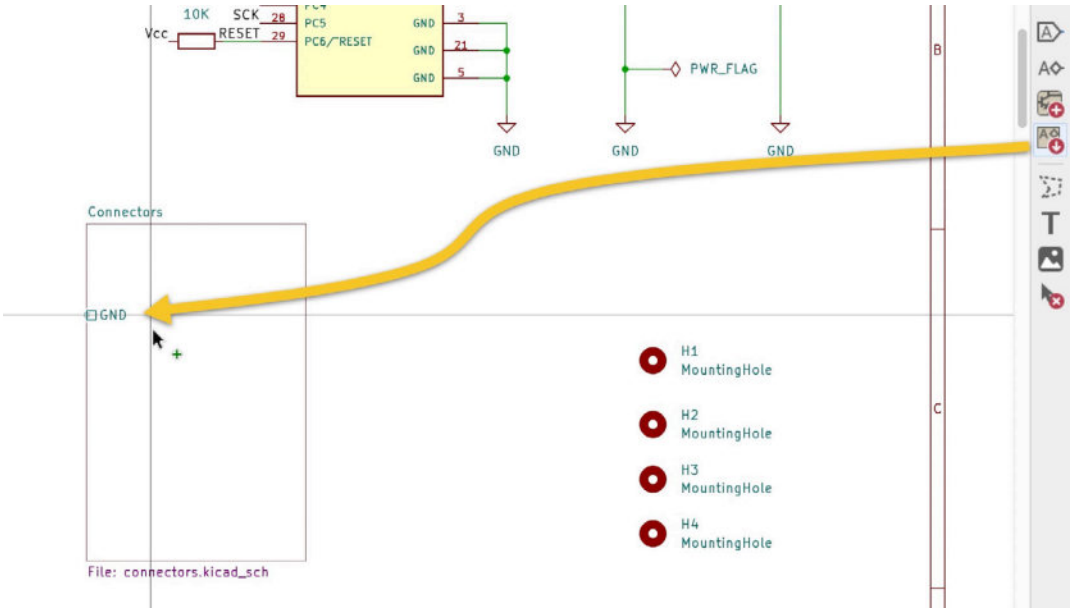


Figure 11.3.8.19: Exposing the hierarchical labels.

Continue in the same manner to expose and place the remaining hierarchical labels. Here is the completed Connectors sheet symbol, with all hierarchical labels in their final positions:

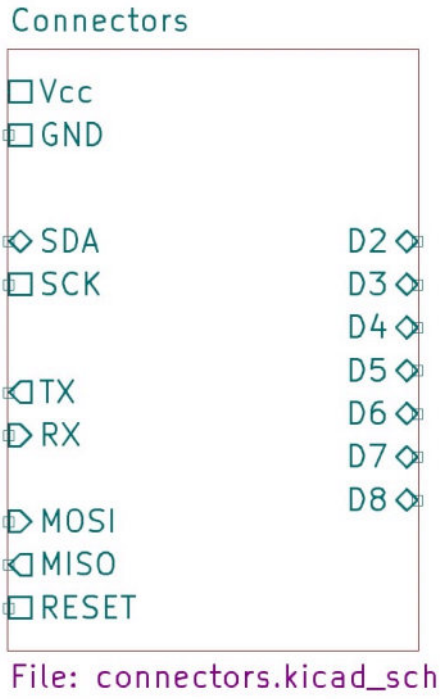


Figure 11.3.8.20: The Connectors hierarchical sheet symbol with exposed hierarchical labels.

I have placed all labels that connect to a digital pin on the right side and the rest along the left side.

The next step is to connect the hierarchical labels to their corresponding pins in the root sheet. In the example below, I have used a Vcc net label, a GND symbol, and graphical line wires to connect the Vcc and GND hierarchical labels to the local news:

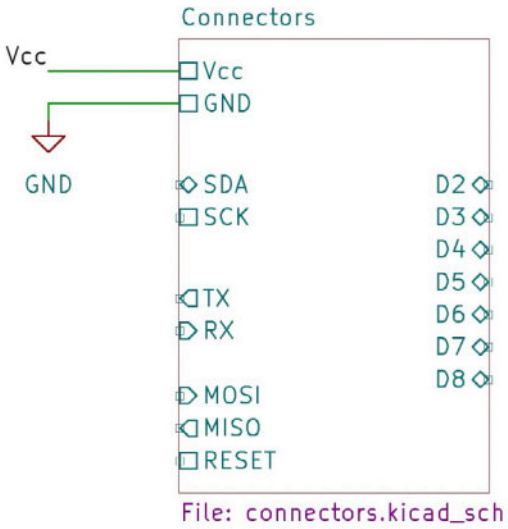


Figure 11.3.8.21: Connected the Vcc and GND hierarchical labels to local nets.

Continue to complete the rest of the connections using net labels. The result of this work is below:

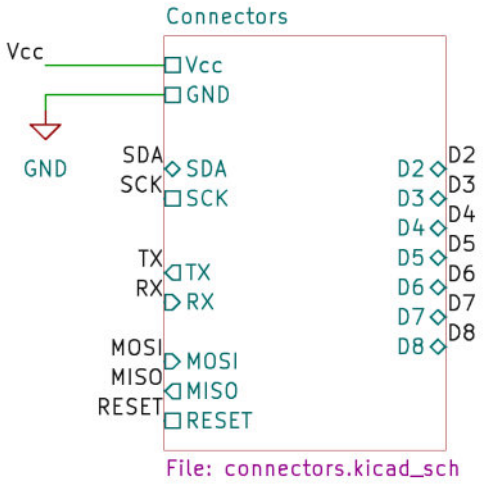


Figure 11.3.8.22: Completed connections of hierarchical labels to local nets.

At this point, the schematic is fully wired. In the next couple of segments, you will allocate nets to net classes, conduct an electrical rules check, and finish the schematic design workflow by adding comments to the schematic.

3.9. Schema 5 - Nets

You completed most of the work relating to named nets during the wiring step, where you used net labels and hierarchical labels to complete the wiring. In this step of the workflow, you will:

1. Cross-check all nets to ensure you haven't missed any.
2. Assign nets to net classes.

You can do the cross-checking visually and confirm that no nets remain unnamed.

For the net to net class assignments, open the Net Classes tab under "Project" in the Schematic Setup window:

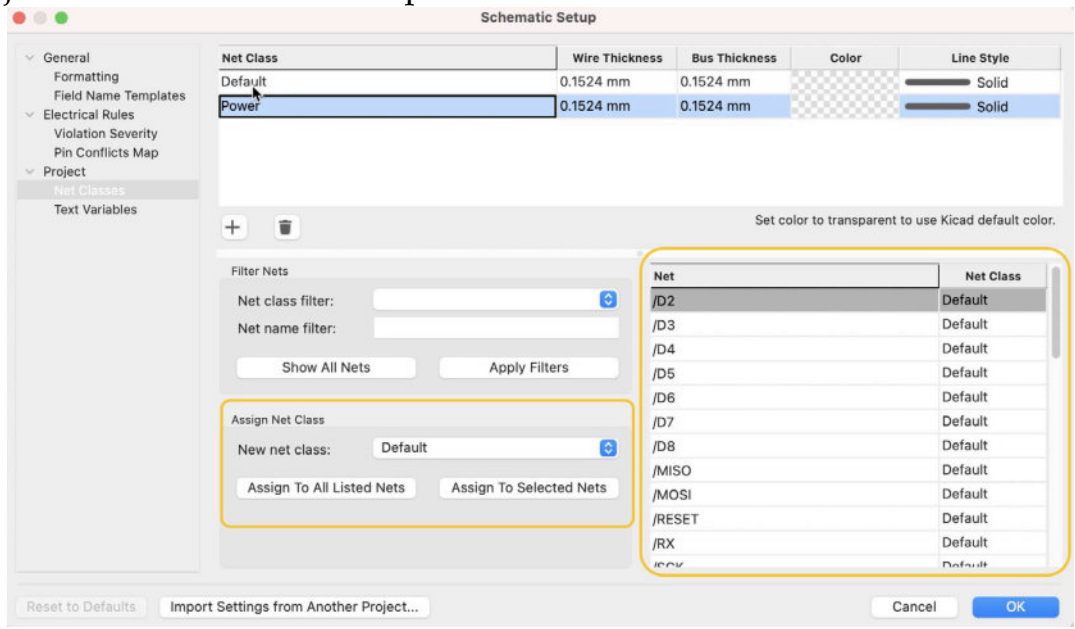


Figure 11.3.9.23: Allocating nets to net classes.

During the setup step of the workflow, you created the Power net class. Use the net table on the right side of the window (above) and the "Assign Net Class" widgets (middle-bottom of the window above) to allocate the Vcc and GND nets to the Power net class.

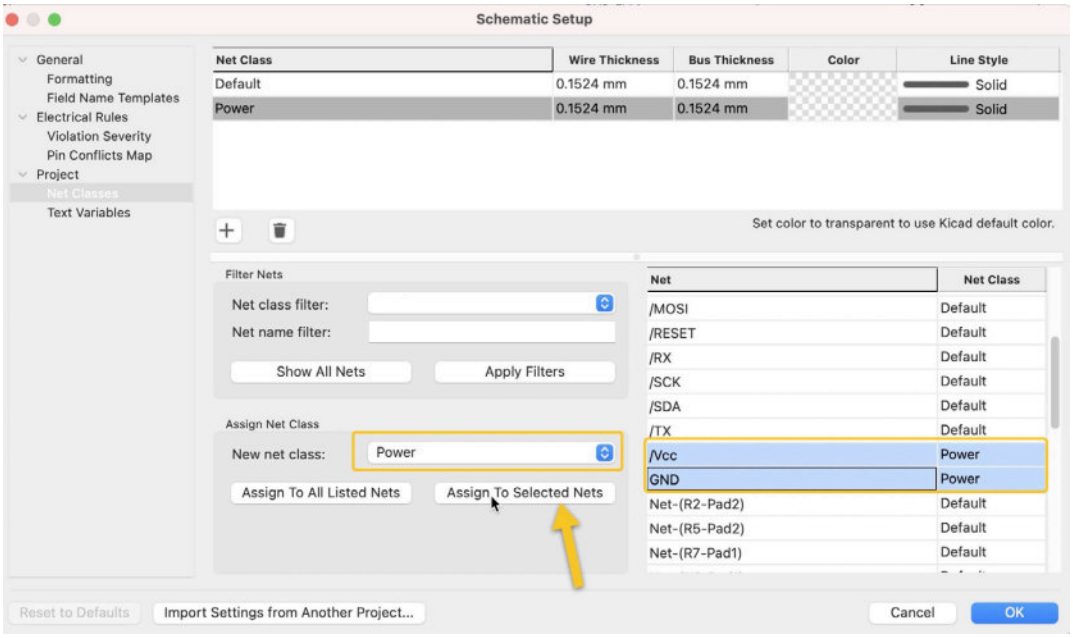


Figure 11.3.9.24: Allocated Vcc and GND nets to the Power net class.

In the figure above, I have selected the Vcc and GND nets in the nets table on the right side and assigned them to the Power net class using the widgets on the left side. All other nets can remain in the Default net class.

You can create additional net classes if you wish, such as one for the digital pin nets and another one for the communications nets. For this demonstration project, I will continue with the two net classes you can see in Figure 11.3.9.24.

Save your work.

The work in this step of the workflow is complete, so time for a new commit to the Git repository:

```
% git status
On branch main
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: MCU Datalogger.kicad_pro

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Allocated GND and Vcc nets to the Power net
class."
[main c23c884] Allocated GND and Vcc nets to the Power net class.
1 File changed, 4 insertions(+), 1 deletion(-)
%
```

Continue with a final ERC in the next segment of this chapter.

3.10. Schema 6 - Electrical Rules Check

This final ERC should not reveal any defects if you have been careful with your work during the schematic design workflow. Nevertheless, always do an ERC before starting work on the layout design.

Open the ERC window, and run the check. Mine revealed only one minor issue:

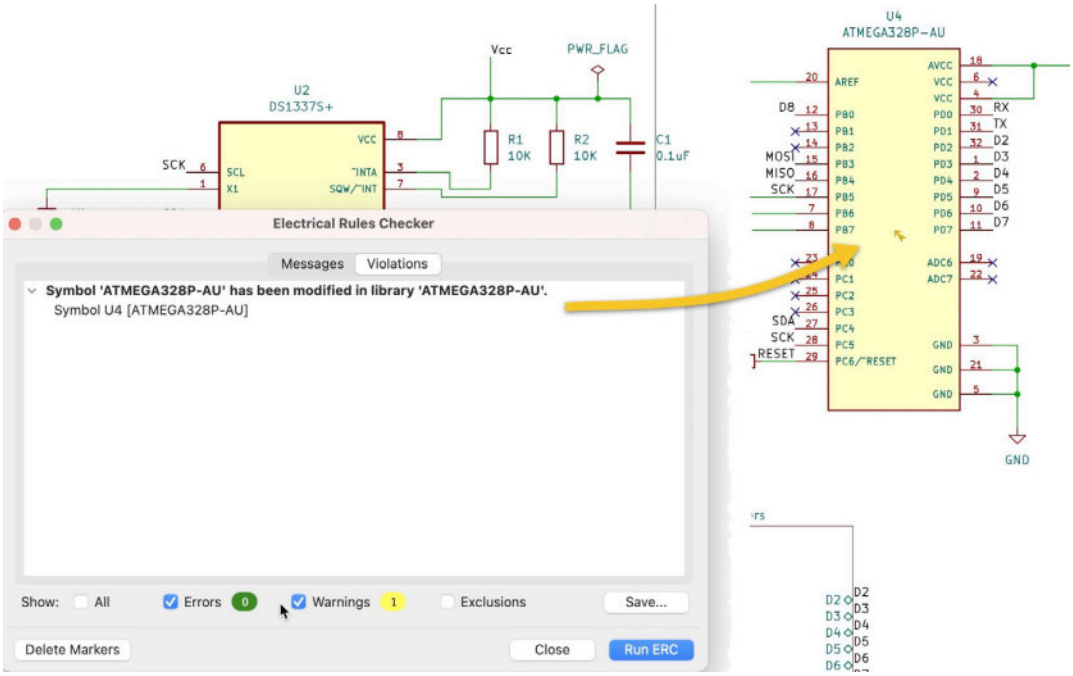


Figure 11.3.10.25: The ERC shows one warning.

The single ERC warning relates to a modification that I made to the Atmega328P-AU symbol. It is warning me that the symbol I use in the schematic is different from the one stored in the library. That is not something that concerns me for this project, so I will safely ignore this warning.

A Git status check shows no change in the project, so there is no need to do a new commit.

Let's complete the work in the schematic design workflow in the next segment by adding comments to the schematic.

3.11. Schema 7 - Comments

To complete work in the schematic design workflow, you will add simple graphics (boxes) and text. Here is a list of comment items I have added to my schematic:

- Use lines to box-in symbols that belong in the same functional group:

- Real-time clock.
- MCU.
- EEPROM.
- Mounting holes.
- Connectors.

- Use text labels to give a name to each functional group.

Below you can see the two sheets, with my comments:

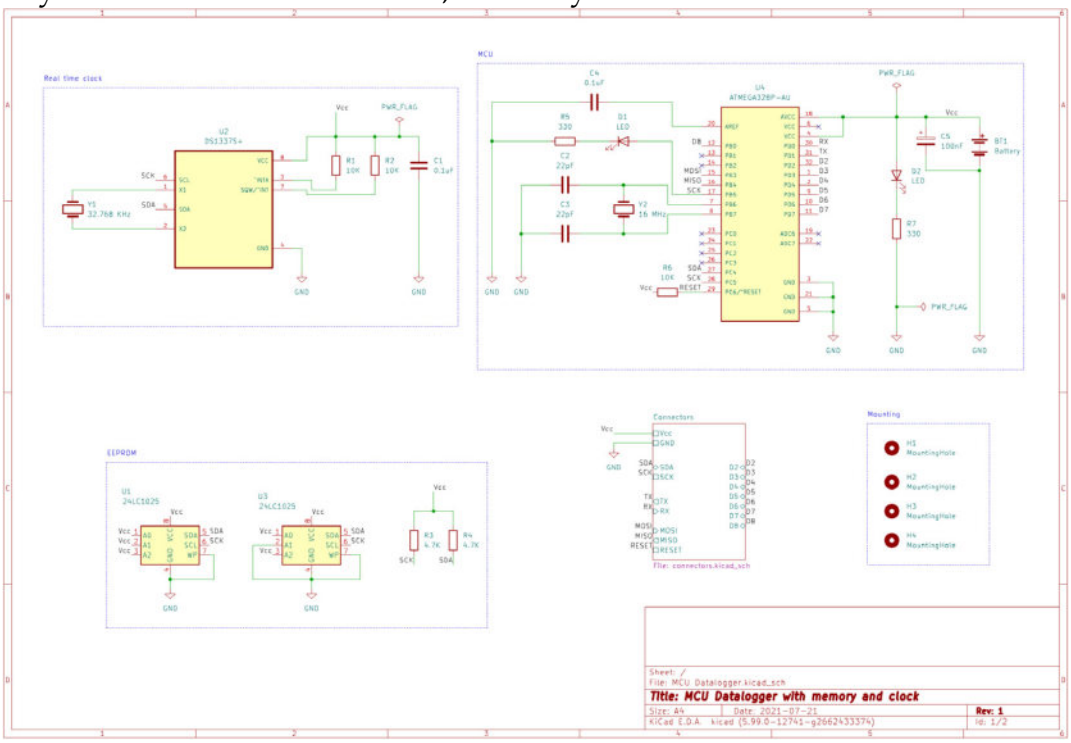


Figure 11.3.11.26: Root sheet with comments.

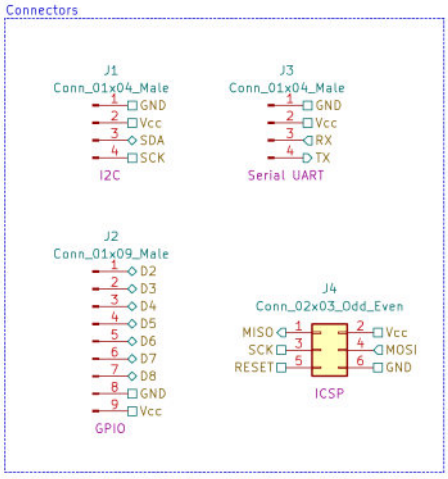


Figure 11.3.11.27: Connectors sheet with comments.

This completes all work in the schematic design workflow. Save the changes, and commit to Git:

```
% git status
```

```
On branch main
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: MCU Datalogger.kicad_sch

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Completed comments."
[main 8fd1ab7] Completed comments.
  1 file changed, 151 insertions(+), 70 deletions(-)
%
```

In the next chapter, you will begin work on a two-layer version of the layout design. In a later chapter in this project, you will also create a four-layer layout. To keep the two layouts separate, you will create dedicated Git branches.

So, before starting work on the two-layer layout, you will create the two-layer Git branch. Let's do that in the next chapter.

4. Create the 2-layer branch in Git

KiCad cannot support multiple layouts for a schematic. However, this is something that becomes possible with the use of Git. In this project, one of the objectives is to design two versions of the layout: a two-layer PCB and a four-layer PCB. Since KiCad can only "see" one layout file, you will create a dedicated Git branch for each layout version.

In this chapter, you will create a new branch for the two-layer PCB. Later in this project, you will create a separate branch for the four-layer PCB.

Ensure that you are working in the main branch (use "git branch" for this). Then, create a new branch named "2layer" and switch into it. You can do both in a single command, like this:

```
% git checkout -b 2layer
M   MCU Datalogger.kicad_pro
Switched to a new branch '2layer'
%
```

Check the status of the repository to confirm that its contents are clean, meaning that it matches with the contents of the main branch from where the new branch stems:

```
% git status
On branch 2layer
nothing to commit, working tree clean
%
```

The new branch is clean, and there are no changes to commit.

Finally, use the "branch" command to get a list of branches in the repository:

```
% git branch
* 2layer
  main
%
```

The repository now contains two branches. The active branch is "2layer". You will continue to commit changes to this layer for all work I describe in the next chapter.

An important question to ask here is how to deal with changes to the schematic? You can answer this question in several ways, but I will describe my tested and simplified process below.

If you are working on the layout of a PCB in a dedicated branch, and you need to make a change to the schematic, follow this process:

1. Commit any changes to the layout branch.
2. Switch to the main branch, which contains the authoritative version of the schematic.
3. Make the necessary changes to the schematic, and save the sheet.
4. Commit the changes to the main branch.
5. Switch to the layout branch.
6. Merge the main branch into the layout branch.
7. Continue work in KiCad and import schematic changes into the layout editor.

You can learn how to merge changes between branches in the Git chapter in the Recipes part of this book.

Now that you have created the two-layer Git branch, you can continue with the layout workflow in the next chapter.

5. Layout design

In this chapter you will work on the layout design of the two-layer version of the PCB. You will be working in the "2layer" branch of the project Git repository that you created in the previous chapter.

Let's begin by setting up the layout design editor.

5.1. Layout 1 - Setup

Open Pcbnew, and confirm the editor settings. In the bullet list below, I show any non-default configuration I have applied in my project instance.

- KiCad Preferences.
  - PCB Editor.
    - Display Options.
      - Min grid spacing: 15 px.
- Board Setup.
  - Board Stackup.
    - Physical Stackup.
      - Copper layers: 2
    - Board Editor Layers.
      - F.Cu: mixed.
      - B.Cu: mixed.
  - Text & Graphics.
    - Text Variables: "project_name" variable inherited from Eeschema.
  - Net Classes.
    - Power.
      - Clearance: 0.25 mm
      - Track Width: 0.35 mm
      - Via Size: 0.9 mm
- Page Settings.
  - Issue Date: copy today's date.
  - Revision: 1
  - Title: ${project_name}

- Comment1: 2-layer PCB version.

Save the setup changes, and then update the PCB from the schematic.

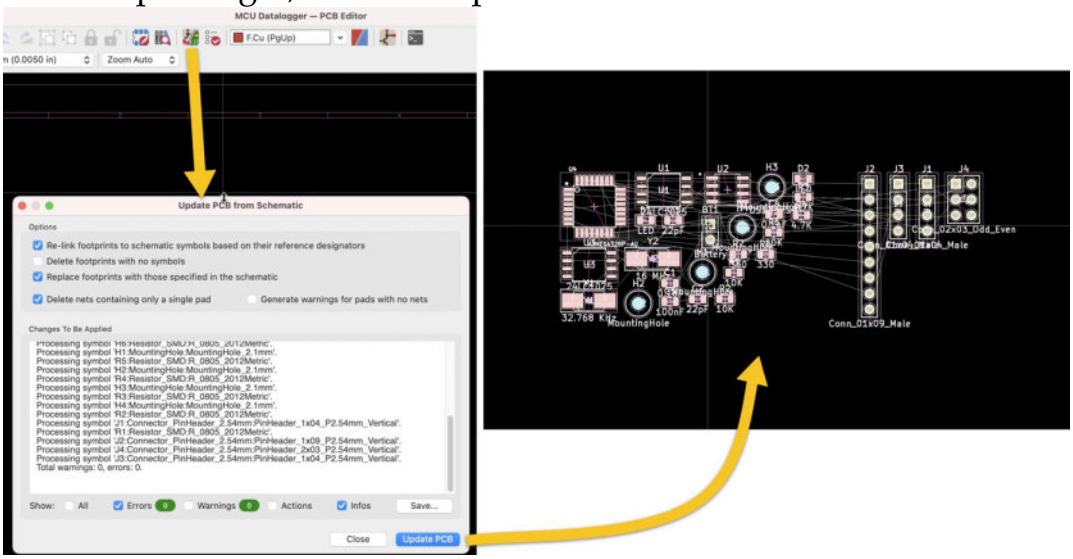


Figure 11.5.1.1: Updated PCB from the schematic.

Save the changes in the layout editor, and commit them to the Git repository. You can see my command line session below:

```
% git status
On branch 2layer
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: MCU Datalogger.kicad_pcb
        modified: MCU Datalogger.kicad_pro

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Setup Pcbnew."
[2layer 1163b1e] Setup Pcbnew.
2 Files changed, 1028 insertions(+), 1028 deletions(-)
%
```

Continue with step two of the layout design workflow, where you will draw the rough outline.

5.2. Layout 2 - Outline and constraints

I will follow the same process as in the previous projects, where I first draw a rough outline for the PCB. I will re-draw the final, refined outline once I have placed all footprints within the rough outline.

For the rough outline, I am about to draw, I have listed the following constraints:

1. The outline should be large enough to accommodate all footprints.
2. The outline should be as small as possible to minimize manufacturing costs.
3. The outline should be rectangular, with four mounting holes at its edges, to make it easy to fit inside a project box.
4. The headers should be along the edges of the PCB to make them easier to access.

I start by evaluating the total dimensions of the footprints, as they appear in the editor after importing them from Eeschema. You can see the footprints bundled together below:

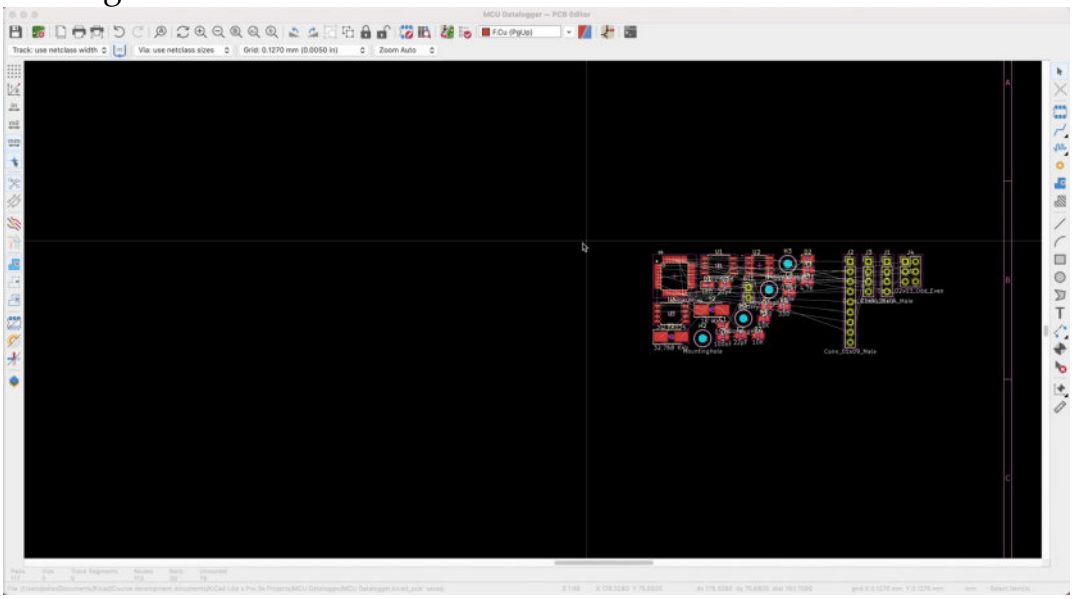


Figure 11.5.2.2: I will create a rough outline for the bundled footprints.

Enable the Edge.Cuts layer and switch the grid size to 1.27 mm. I used the measuring tool to measure the width and height of the footprint bundle. For the width, I measured approximately 50 mm, and for the height, around 30 mm. This gives me enough information to draw a rectangle that can accommodate the footprints.

Use the rectangle drawing tool from the right toolbar, and draw a rectangle. Based on my selected grid size, my rough outline measures 50.800 mm x 30.480 mm.

You can see the result below:

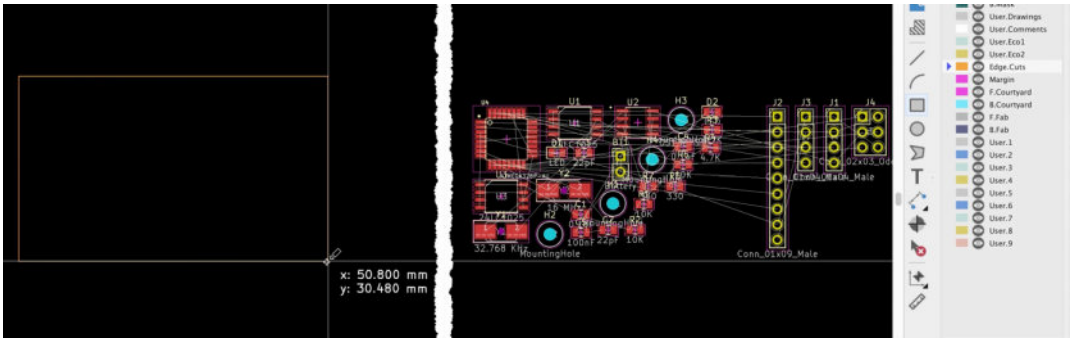


Figure 11.5.2.3: Drawn a rough outline of the board.

The footprints should fit comfortably in this outline. Let's continue in the next segment, where you will place the footprint within the board outline.

5.3. Layout 3 - Place components

The rough outline you created in the previous segment will make it easier to place the footprints within it. Compared to the LED matrix display PCB, the footprint placement step in this project is much easier. No external constraints dictate the exact position of footprints.

I have followed a simple process to help me with the placement, which I outline below:

1. Change the grid size to 0.635 mm. This will provide a fine grid to allow for precision spacemen with minimal wasted space.
2. Place the microcontroller footprint in the middle of the board. This will make it easier to connect it to the various peripherals and connectors around it (except for the power connector).
3. Place the mounting holes in the four edges of the outline.
4. Place the connectors along the bottom and right edges.
5. Place the EEPROM and real-time clock footprints on the left of the MCU.
6. Place the oscillator for the real-time clock on the left of the RTC footprint.
7. Place the oscillator for the MCU below the MCU footprint.
8. Place the battery connector on the left edge of the outline.
9. Place the capacitors in the remaining space; make an effort to minimize their distance to their main components using the ratsnest lines as a guide. For example, C1 should be next to U3 and C2 near Y2 to minimize their total copper track widths.

10. Place the resistors last, as close as possible to their main components. Again, try to minimize their copper track lengths using their ratsnest lines as a guide.
11. Use the editor alignment tools to ensure that all footprints are correctly aligned.

You can see my final placements below. Your's may differ, of course, but the figure below gives you an example.

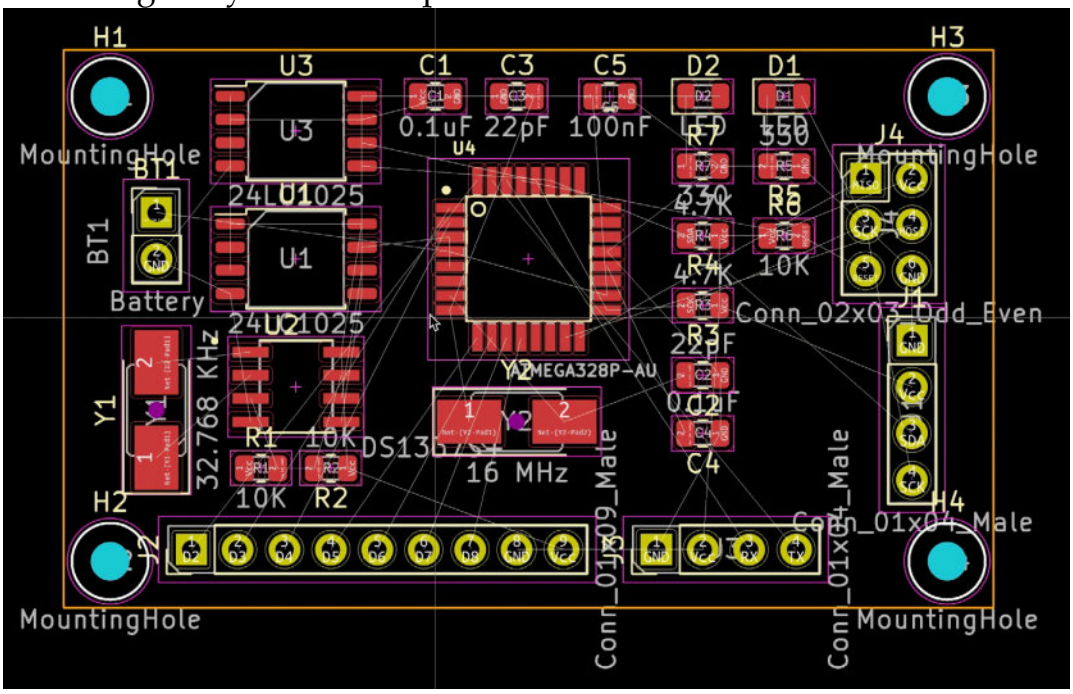


Figure 11.5.3.4: The final placement of the footprints in the rough board outline.

Save your work, and commit the changes to the Git repository:

```
% git commit -am "Completed placement of footprints."
[2layer 9fe697f] Completed placement of footprints.
  2 files changed, 76 insertions(+), 74 deletions(-)
%
```

With the footprint placement complete, you can continue and refine the board outline.

5.4. Layout 2 - Outline refinement

The rough outline of the board helped you to place the footprints within its borders. With the footprints now placed, you can refine this outline and complete the drawing in the Edge.Cuts layer.

Looking at the result of the last two steps of the process (below), you can see that it is possible to reduce the size of the board further by:

1. Reducing the gap between the outer footprints and the outline edges.
2. Placing the mounting holes in outdents in the four corners of the board.

In the figure below, I have super-imposed the rough outline of the board with the refined version so that you can see the differences:

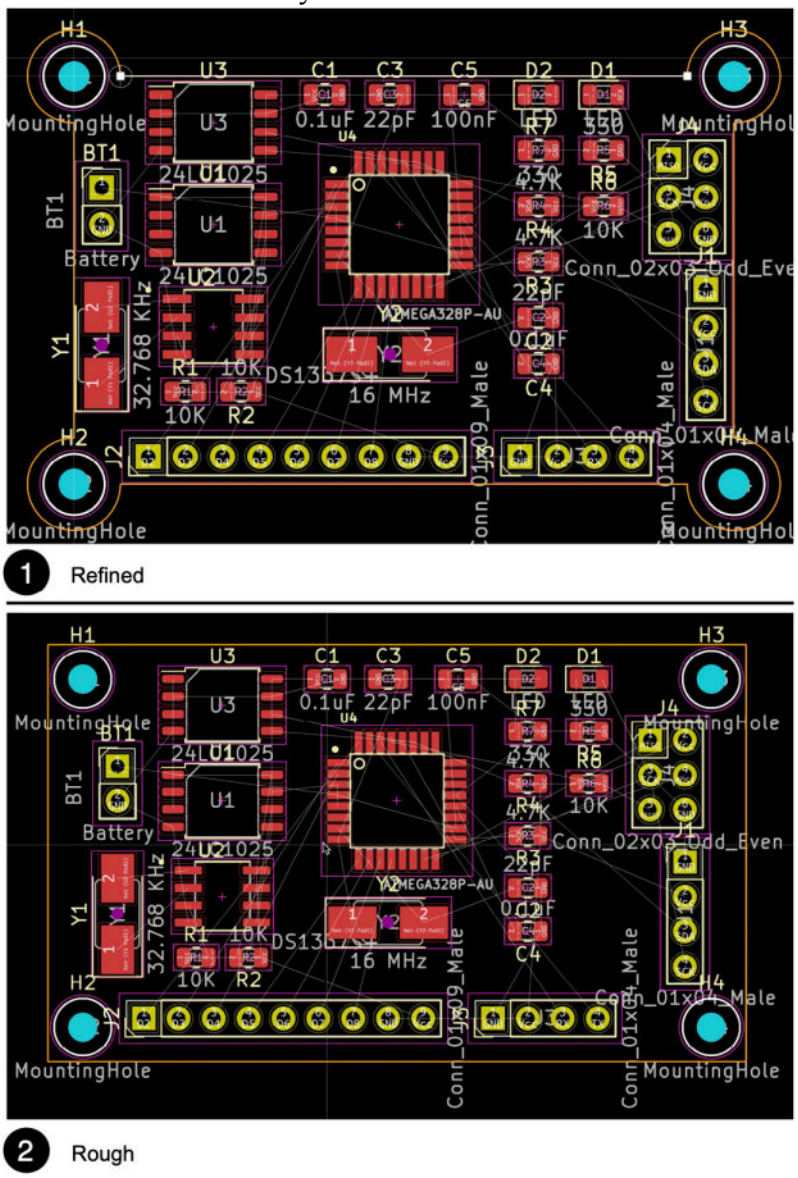


Figure 11.5.4.5: Comparing the rough board outline against the refined version.

You can create the circuital segments of the board using the arc tool. The placement of the mounting holes within the outdents makes it possible to (carefully) snap them off the board if you don't want to use them so that the board can fit in a smaller box.

Below you can see a larger version of the refined outline:

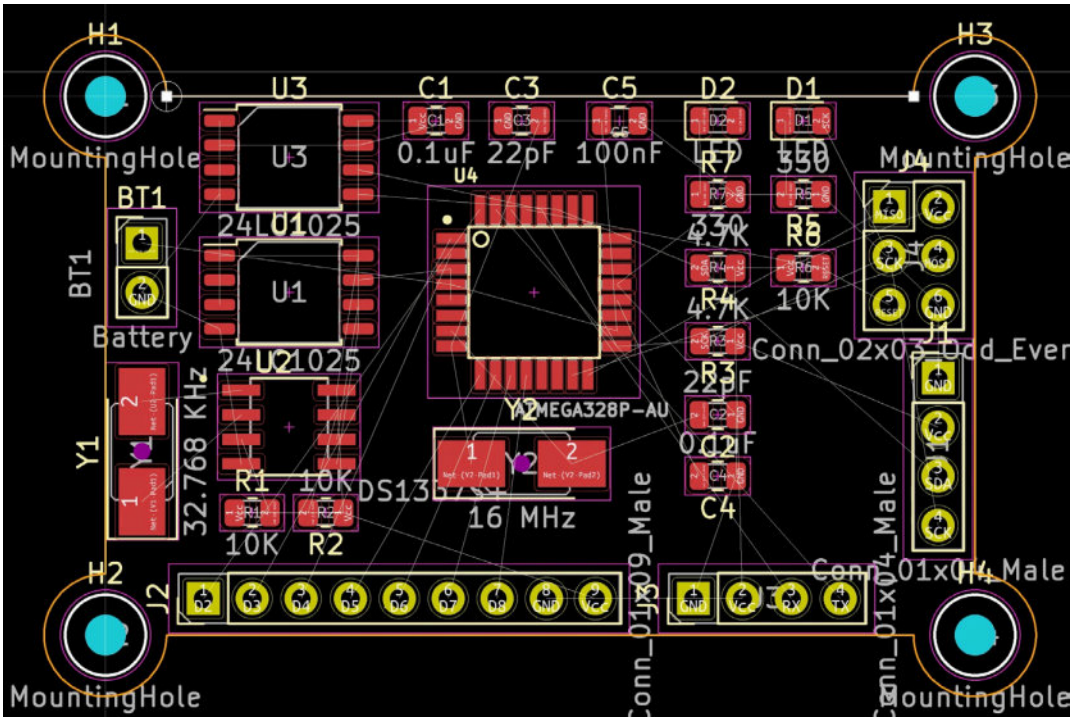


Figure 11.5.4.6: The final refined board outline.

Save the changes and commit them to the Git repository:

```
% git status
On branch 2layer
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: MCU Datalogger.kicad_pcb

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Refined and completed outline
[2layer 9d8b9cd] Refined and completed outline.
  1 File changed, 14 insertions(+), 7 deletions(-)
%
```

In the next step of the layout workflow, you will do the routing of the copper tracks on the top and bottom layers.

5.5. Layout 4 - Route

The manual routing of a board is usually the most time-consuming step of the layout workflow. It took me approximately one hour to route this board into two layers. This time includes several cases of undoing work that I had already completed to improve and optimize the placement of tracks.

For the entire routing process, I used the Interactive Router in "Walk around" mode.

Here is an outline of the strategy that I followed:

1. General rule #1: Draw signal and Vcc tracks in the top copper layer.
2. General rule #2: Use a copper fill connected to the GND net to connect all THT pads that belong to the GND net. Therefore, leave THT GND pads unconnected until the copper fills step (in the next segment in this chapter). You may follow the same rule for the Vcc pads and use a copper fill in the top layer connected to the Vcc net to route them.
3. General rule #3: Minimise the space between tracks to reduce unusable space. In the example below, I have placed the red tracks at minimum distances from each other ("1"), leaving enough space for at least one more track ("2").

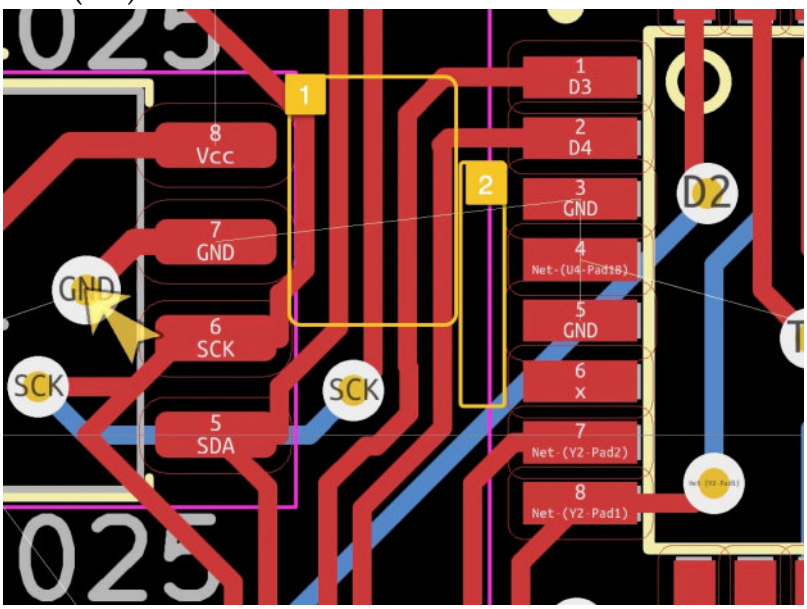


Figure 11.5.5.7: Don't waste space.

4. General rule #4: Exhaust your search for a path for a non-GND track in the top layer before using vias. In many cases, you will be able to complete these tracks in the top layer simply by repositioning existing tracks.

5. General rule #5: For SMD pads that belong to the GND net are not close to other GND pads, you can create a nearby via and connect it to the bottom layer copper fill. You can see an example of this below:

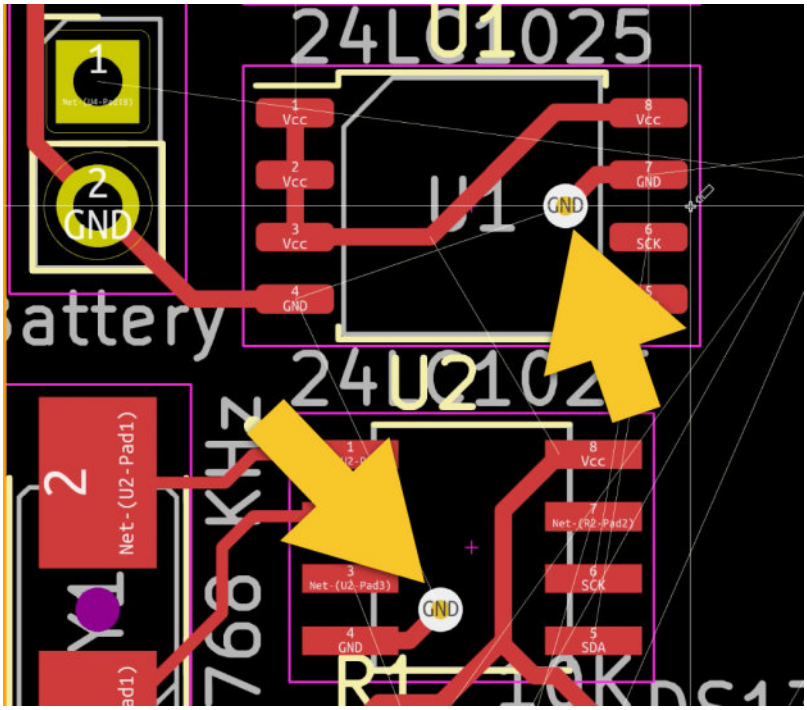


Figure 11.5.5.8: Using nearby vias to connect SMD GND pads to the GND copper fill in the bottom layer.

6. Start by drawing copper routes between adjoining pads. This includes Vcc and GND pads for the capacitors and resistors. Because most footprints in this PCB are SMD, I have drawn GND copper tracks in the top layer and used vias to connect them to the GND copper fill in the bottom layer. Here are a few example tracks that connect adjoining pads:

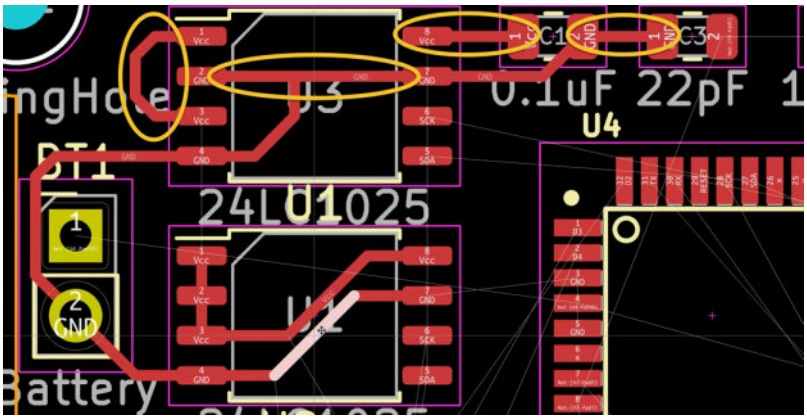


Figure 11.5.5.9: Tracks connecting adjoining pads.

7. Continue by drawing tracks that connect the peripherals to the central MCU footprint.

8. Draw the tracks to/from the connectors.

9. Create a copper fill in the bottom layer and connect it to the GND net. This should complete the routing for all THT pads and vias that belong to the GND net (I'll do this in the next segment of this chapter).

10. Create a copper fill in the top layer and connect it to the Vcc net (I'll do this in the next segment of this chapter).

After following this strategy, I produce this partially-routed layout:

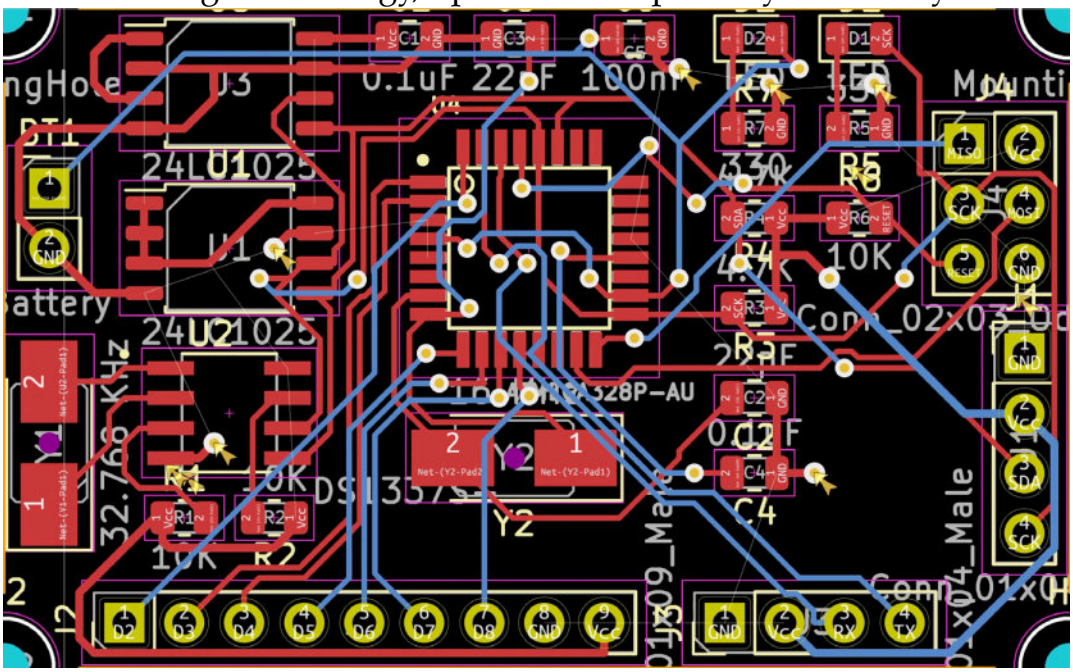


Figure 11.5.5.10: Partially routed PCB; GND pads not connected yet.

The missing tracks are those that complete the connections between the GND pads. My DRC check looks like this:

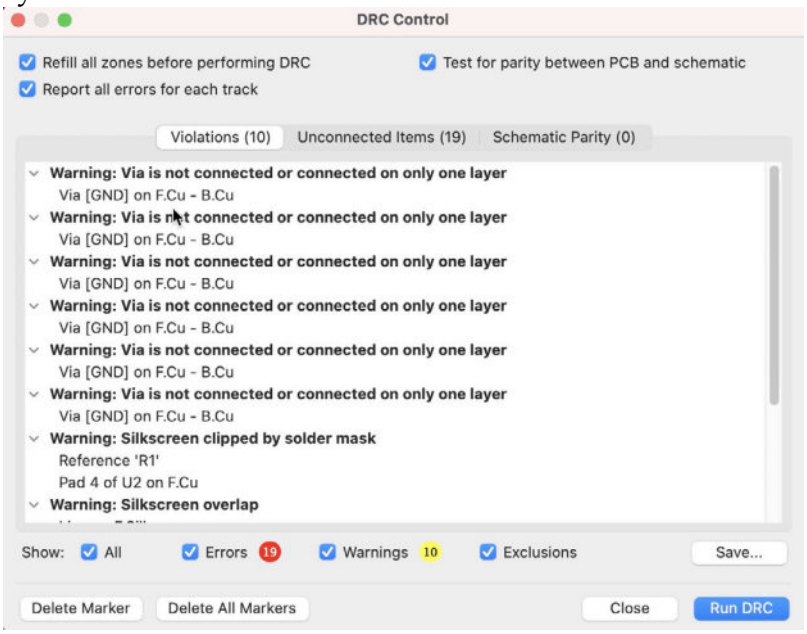


Figure 11.5.5.11: DRC shows unconnected GND pads and vias.

Indeed, the DRC shows that the only unconnected items are GND vias and pads. The "Unconnected Items" list also shows a couple of unconnected Vcc pads (not showing in the list of the figure above). I will take care of both issues in the next segment in this chapter.

There are also possible improvements in the shape of some of the tracks. In particular, I will be changing the shape of a few tracks that contain 90-degree corners and optimizing the use of space. I will make those improvements after creating the copper fills.

Save your work, and commit the changes to the Git repository:

```
% git status
On branch 2layer
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: MCU Datalogger.kicad_pcb
    modified: MCU Datalogger.kicad_prl

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Routing sans GND Vcc complete."
[2layer 56c8f14] Routing sans GND Vcc complete.
  2 files changed, 382 insertions(+), 13 deletions(-)
%
```

Let's continue with the copper fills.

5.6. Layout 4 - Copper fills

In this step of the layout design workflow, you will create two copper fills. You will place the first copper fill in the bottom copper layer and connect it to the GND net. Place the second copper fill in the top copper layer and connect it to the Vcc net. After creating these copper fills, the remaining unconnected items violations that the DRC reported in the previous segment of this chapter should be resolved.

Below is the Properties window for the first copper fill (bottom copper layer, connected to GND):

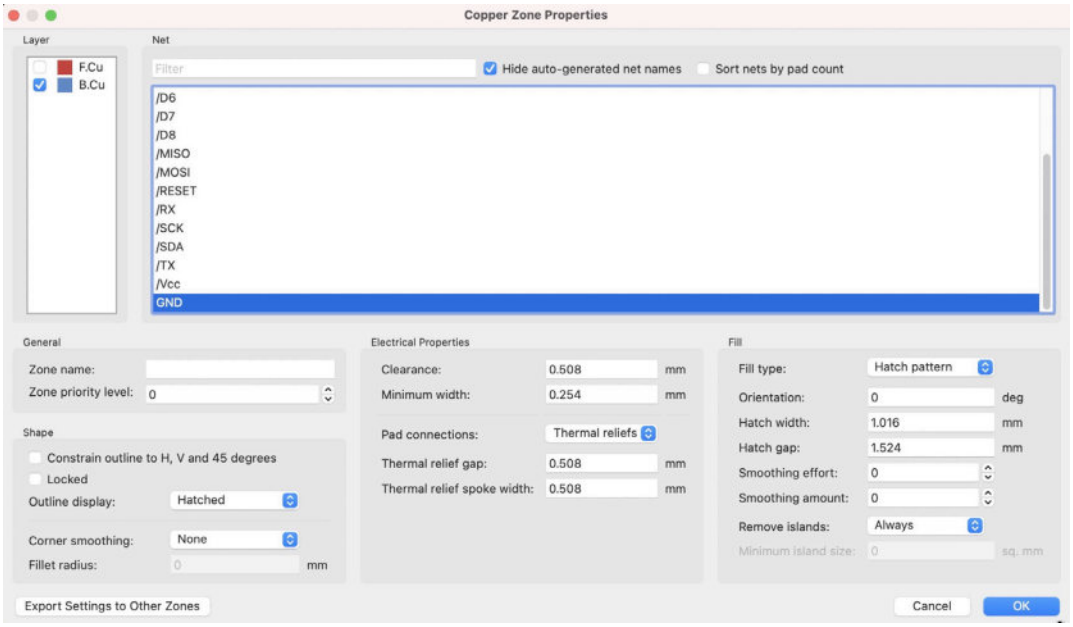


Figure 11.5.6.12: Copper fill properties for the bottom layer.

If you need a reminder on drawing a copper fill, please read the relevant chapter in Part 8 of this book.

Go ahead and draw the copper fill in the bottom layer.

For the copper fill in the top layer, I will be using a shortcut that can save a lot of time. Instead of drawing the new copper fill, I will duplicate it and connect it to the Vcc net. This way, the top, and bottom copper layers will have the same shape. I describe this technique below:

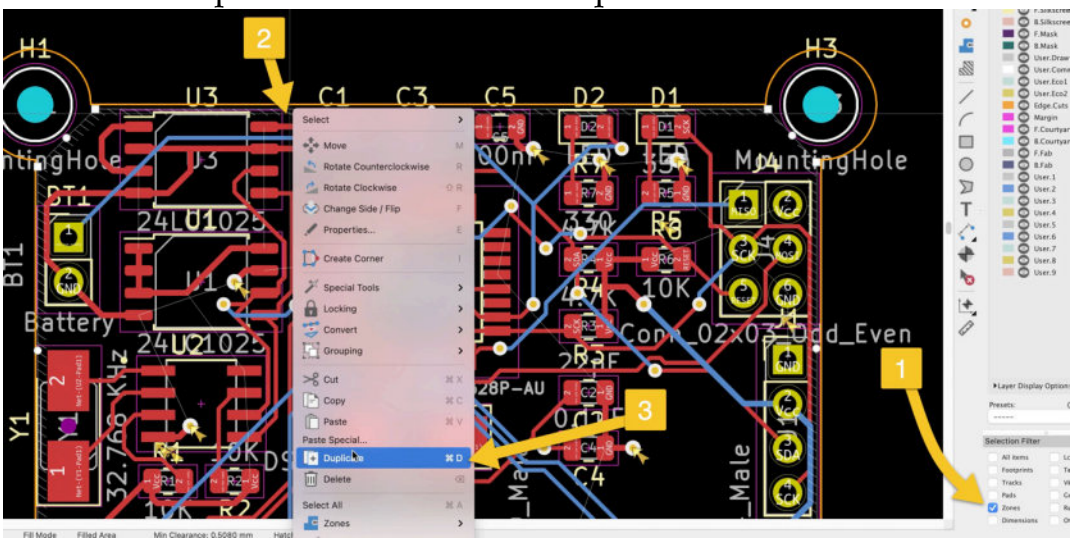


Figure 11.5.6.13: Duplicating a copper fill.

First, ensure that "Zones" is enabled in the Selection filter ("1", above).

Next, right-click on the border of the existing copper fill to bring up the context menu ("2") and click "Duplicate" ("3"). The new copper fill outline will appear attached to the cursor. Move the outline slightly outside the board so

that the two copper fills don't overlap, and double-click on it to open its properties window.

Change the Layer setting to "F.Cu" and the Net to "Vcc," as in the example below:

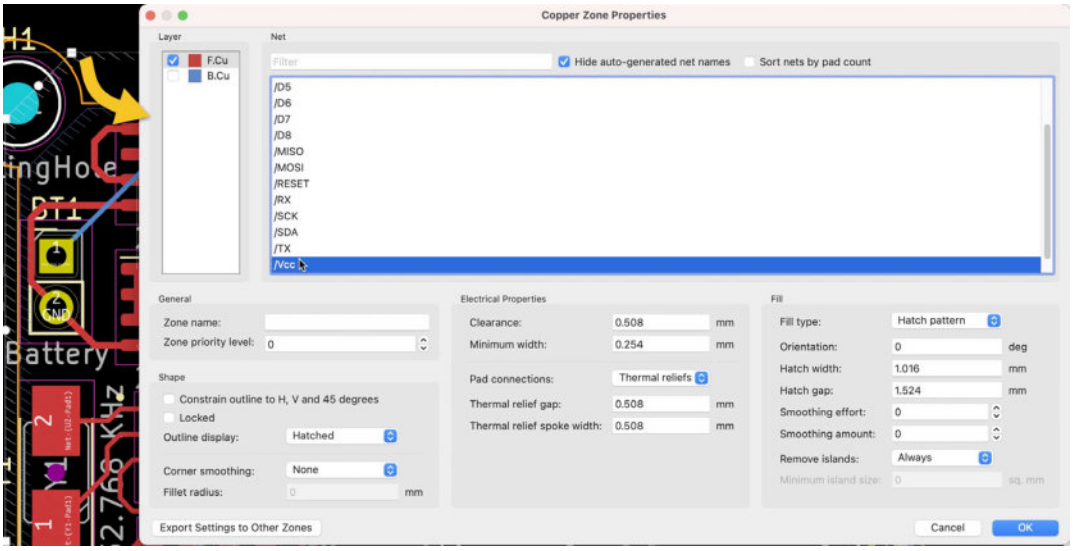


Figure 11.5.6.14: Copper fill properties for the top layer.

Click OK to close the Properties window and move the fill outline to overlap the bottom copper fill. See a detail of the two copper fills overlapping below:

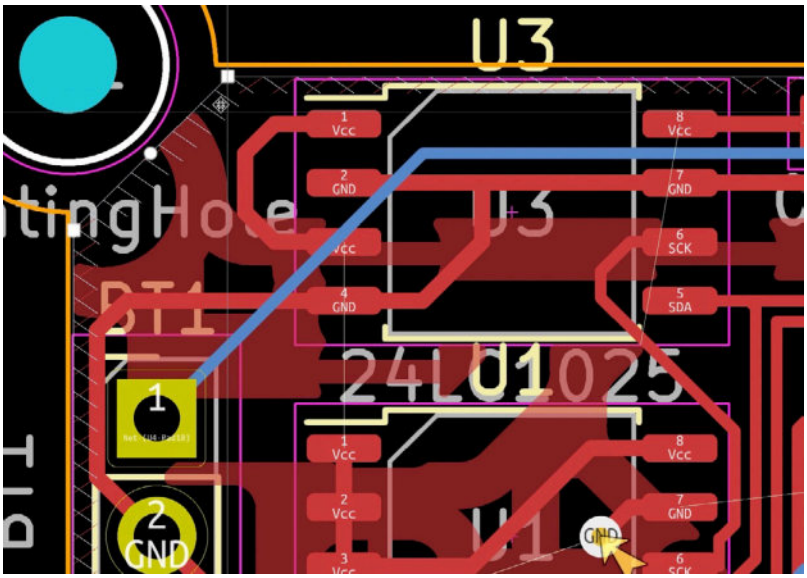


Figure 11.5.6.15: The two copper fills overlapping.

Fill both copper fills using the “Fill All Zones” context menu:

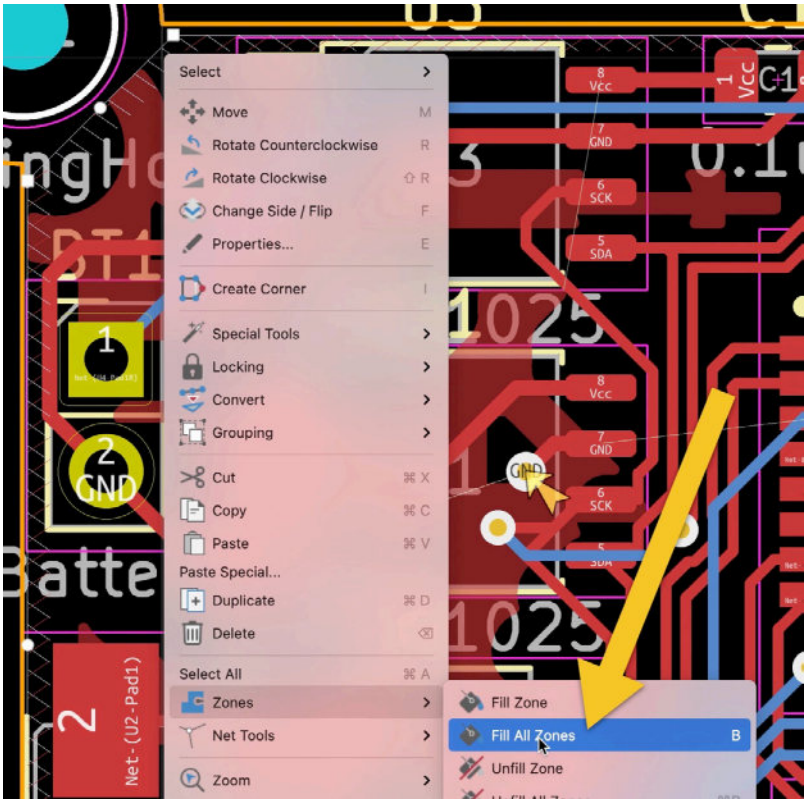


Figure 11.5.6.16: Fill all copper fills.

The result of this work is below:

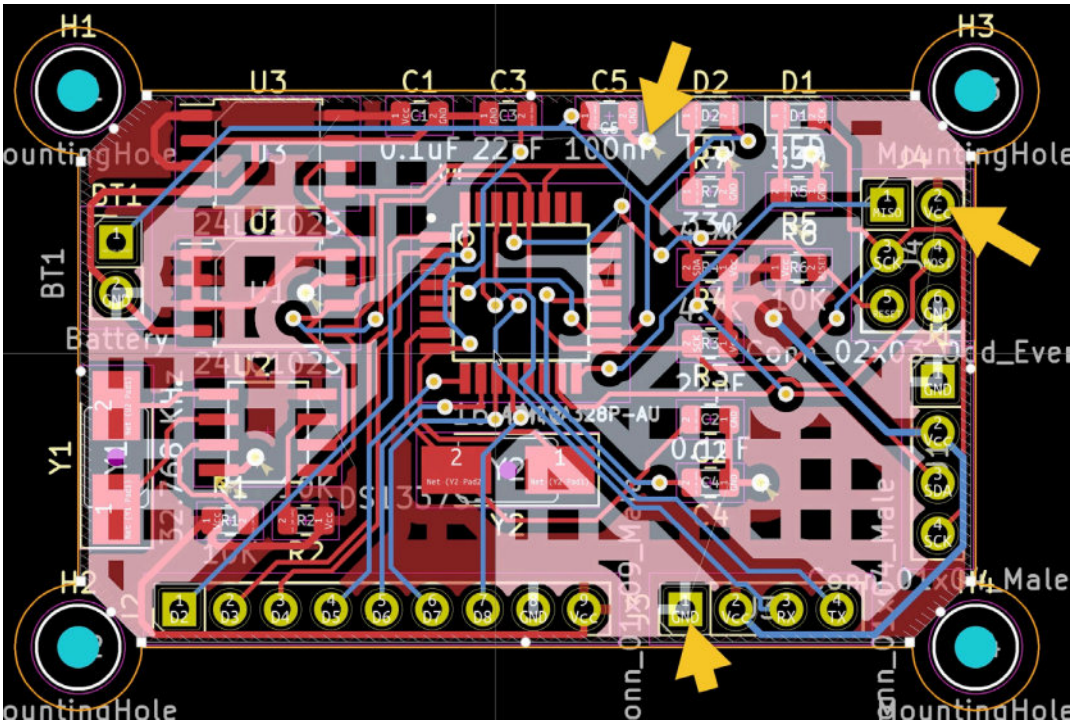


Figure 11.5.6.17: Top and bottom copper layers, filled.

After creating the copper fills, you may see that one or more Vcc or GND pads remain unconnected. This is usually due to segments on the top or

bottom layers being inaccessible by the copper fill; there is simply no gap that is large enough for the fill area to enter into those segments.

In the figure above, I have used an arrow to highlight two pads and one via that remain unconnected.

If this happens in your layout, you will need to change the position and shape of the existing tracks and vias to increase the available space through which the copper fills may be able to reach the unconnected items.

I had to spend an additional thirty minutes reworking the tracks to finally make the copper fills reach all Vcc and GND pads. The result is below:

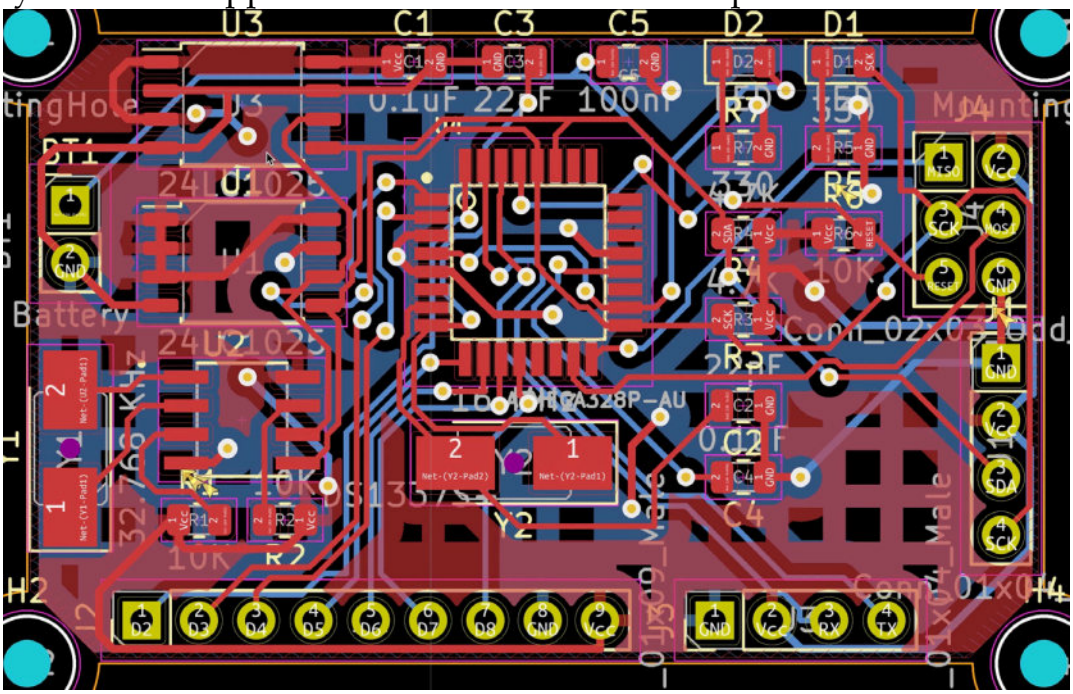


Figure 11.5.6.18: Top and bottom copper layers, filled, completed.

Save your work and commit changes to the Git repository:

```
% git status
On branch 2layer
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: MCU Datalogger.kicad_pcb
    modified: MCU Datalogger.kicad_prl
    modified: MCU Datalogger.kicad_pro

Untracked files:

  (use "git add <file>..." to include in what will be committed)

    _autosave-MCU Datalogger kicad_pcb
```

```
no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Completed routing."
[2layer 55d972c] Completed routing.
  3 files changed, 6765 insertions(+), 3 deletions(-)
%
```

There is one autosave untracked file, which you can ignore as autosave files do not need to be tracked. In the next segment, I will discuss some of the board's routing improvements that I made.

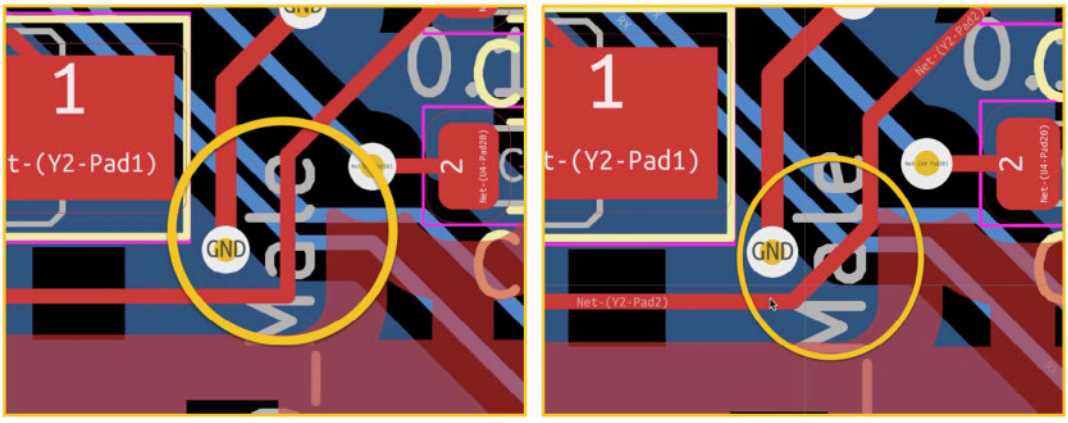
5.7. Layout 4 - Routing improvements

There is always scope for improvement. You can improve your board design in several ways. Below is my top-four list:

1. Optimize the positioning of the footprints.
2. Shorten the length of the copper tracks.
3. Improve the shape of the board.
4. Improve the shape of the tracks.

Here, still working in step four of the layout workflow, I took some time to improve the shape of some of the tracks. The improvements that I made are:

1. Replaced any 90-degree corners with 45-degree corners. See an example here:



11.5.7.19: Removing 90-degree corners.

2. Fixed stray track segments, like the one below:

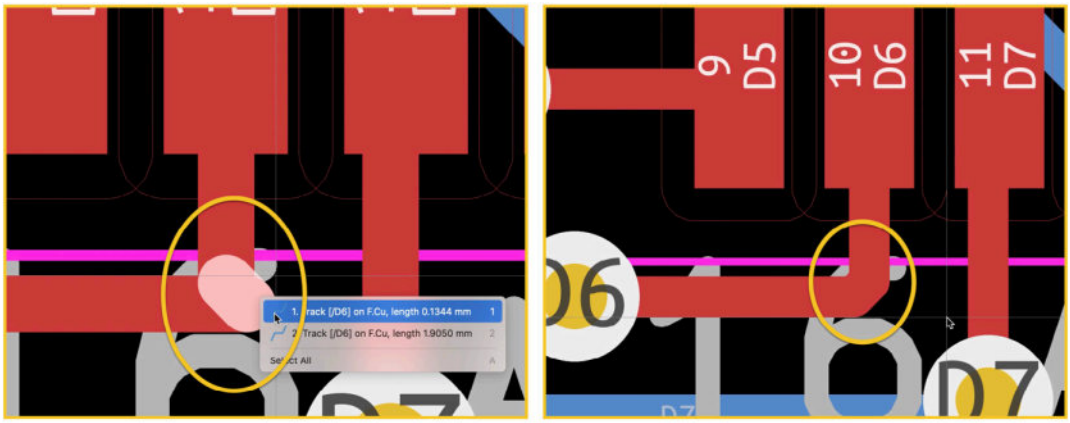
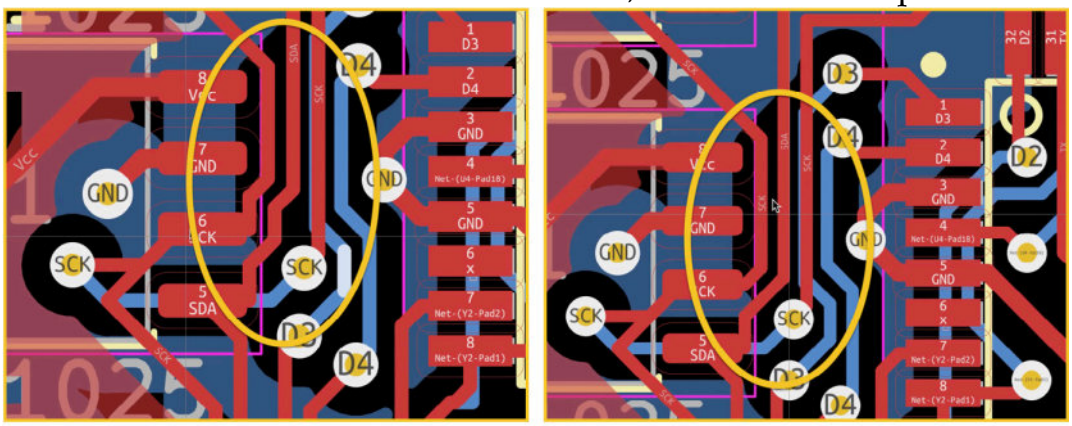


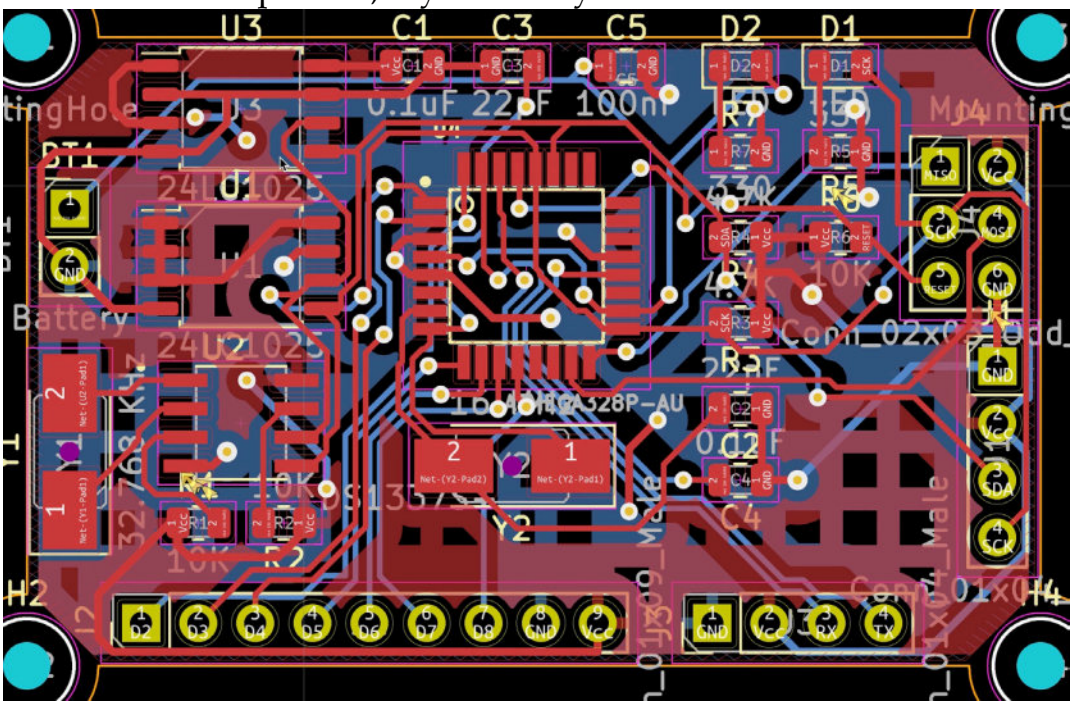
Figure 11.5.7.20: Removing a stray track segment.

3. Reduced the number of bends in a track, like in this example:



11.5.7.21: Reducing the twists and turns.

At the end of this process, my board layout looks like this:



11.5.7.22: Board layout with optimized routing.

You could certainly continue with the optimization process forever. However, this is not practical, so at some point, you should stop and continue to the next step of the workflow.

Save your work and commit the changes to Git:

```
% git commit -am "Refined routing complete."
[2layer 477bb71] Refined routing complete.
  1 file changed, 1710 insertions(+), 1035 deletions(-)
%
```

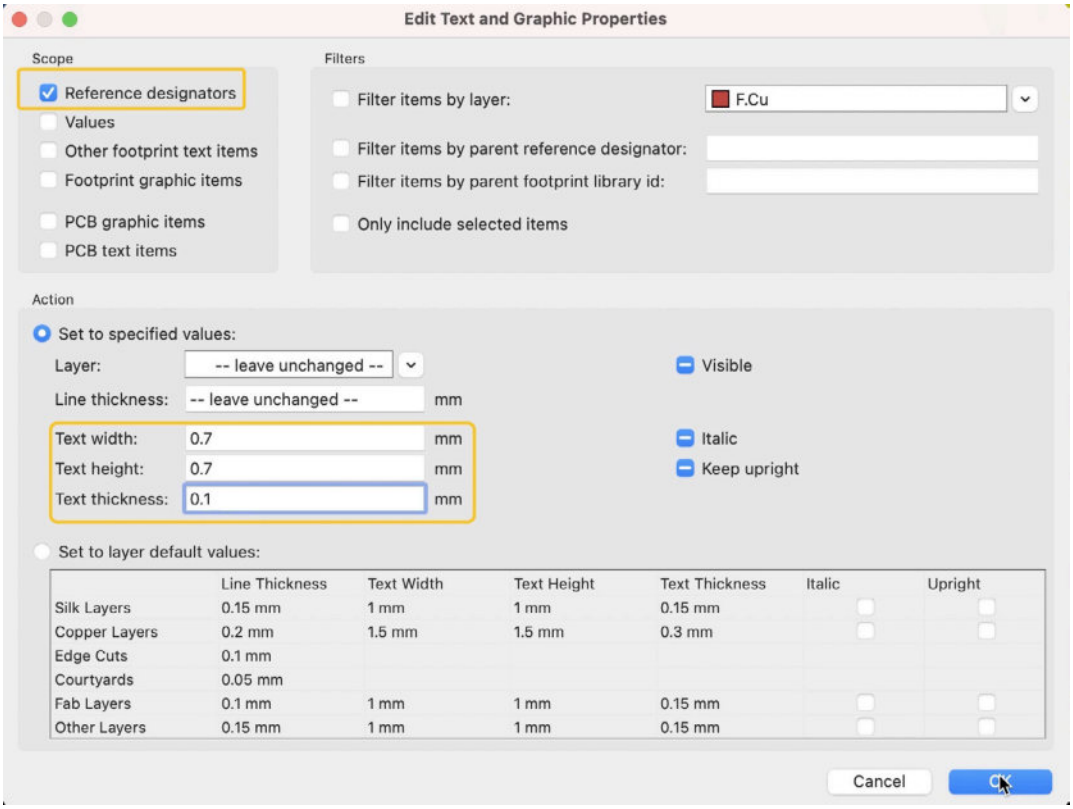
Let's continue with the silkscreen.

5.8. Layout 5 - Silkscreen

In step 5 of the layout workflow, you will add text and graphics in the top and bottom silkscreen layers. Because you will be working specifically in the top and bottom silkscreen layers, you can make invisible any items that exist in layers, such as the front and back fabrication layers. This will remove text clutter in the editor. For the same reason, change the mode of the filled zone dedication to "Show only zone boundaries" and reduce the intensity of the tracks. All this will help you focus on the silkscreen layers and content.

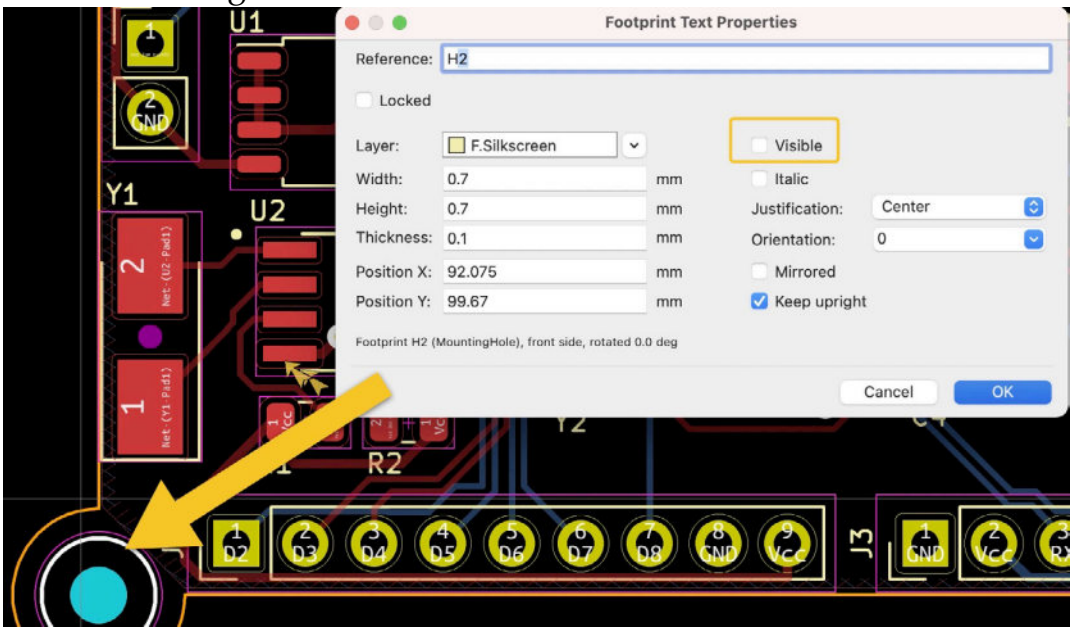
I organize this work in the following parts:

1. Place existing content (such as footprint values and references) in the correct layer and location. Use the bulk editing tools to standardize the size of this content. In the example below, I use the "Edit Text and Graphic Properties" window to reduce the size of all reference designators so they can fit in the available space:



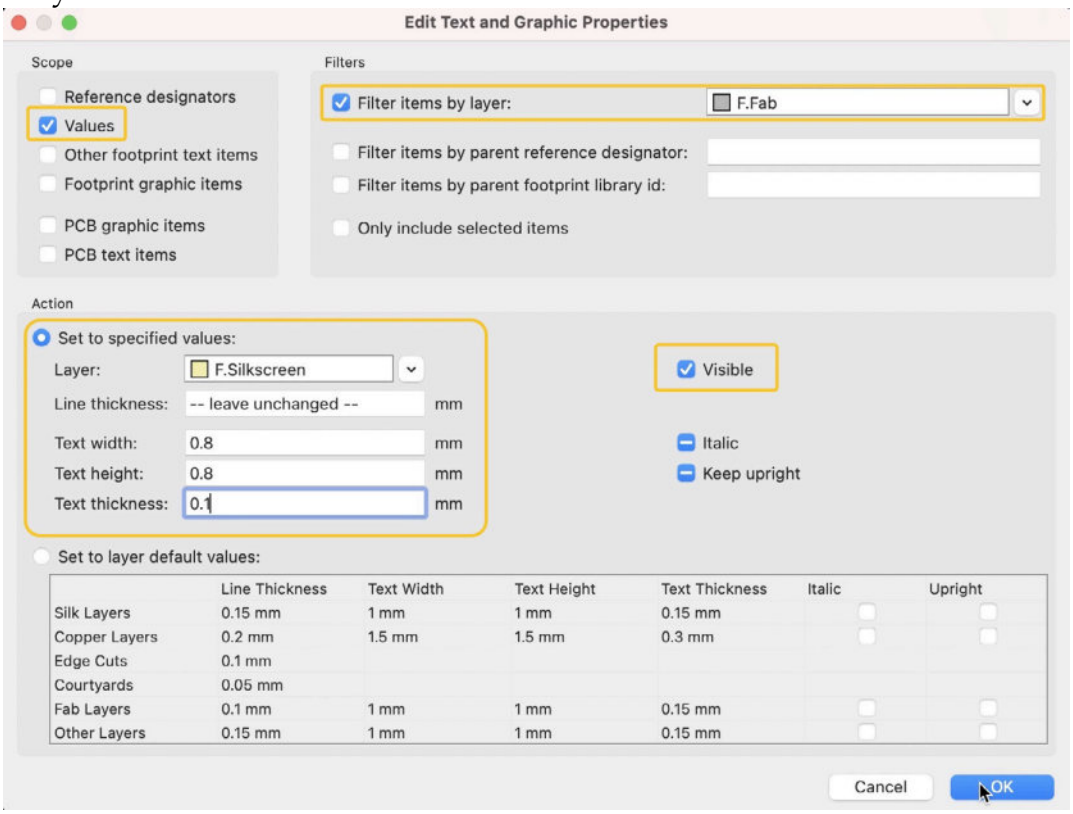
11.5.8.23: Reducing the size of the Reference Designators text.

2. Look for any visible content that should be made invisible, and make it so. For example, there is no need to show the values for the mounting hole footprints. Uncheck their “visible” property in their Properties window. In the example below, I have done this for reference designator H2, which belongs to one of the mounting holes:



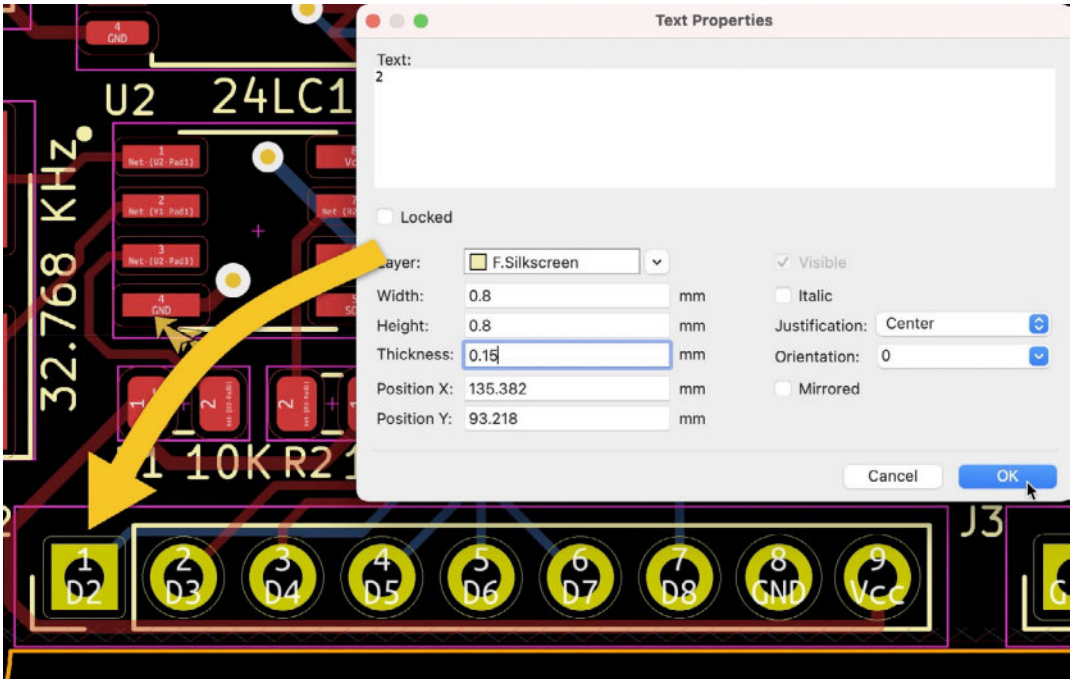
11.5.8.24: Made H2 invisible.

3. Look for content in a non-silkscreen layer containing essential information, and move it to a silkscreen layer. For example, the crystal oscillator footprints contain their operational frequency in one of the fabrication layers. I want to show this information on the front silkscreen. Below, I use the text bulk edit tool to do this for all values that exist in the F.Fab layer:



11.5.8.25: Bulk edit values in the F.Fab layer.

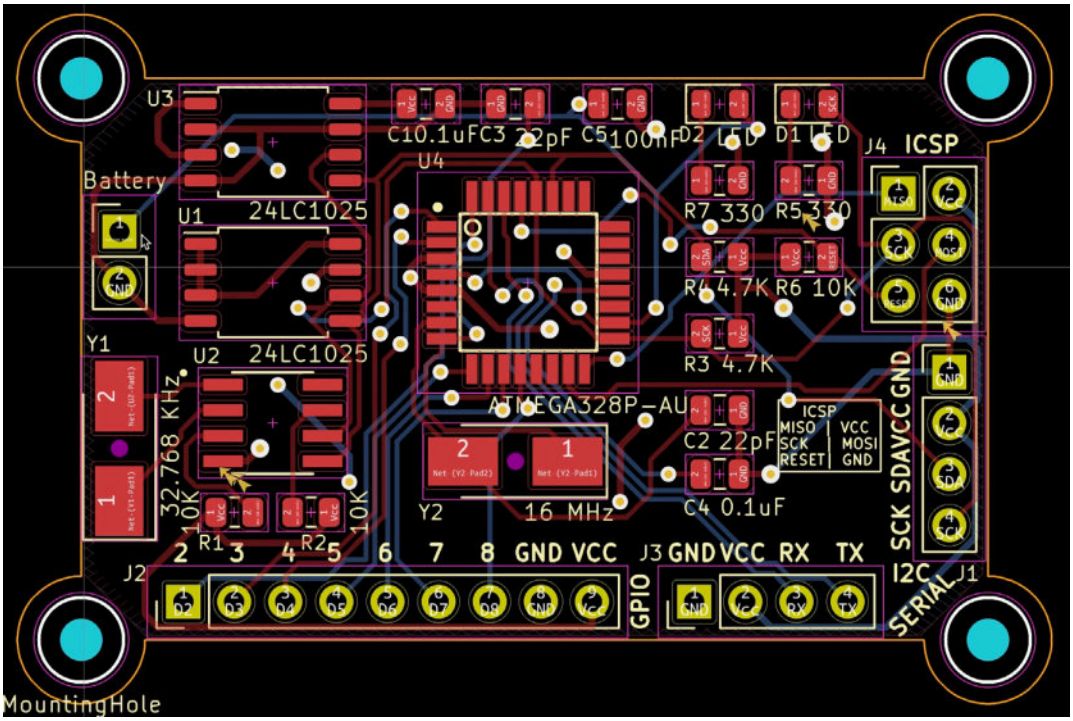
4. Add new text labels to convey pins roles, input voltage, etc. In the example below, I'm creating a text label with the content "2". I will place this label over pin D2 in header J2:



11.5.8.26: Creating a text label.

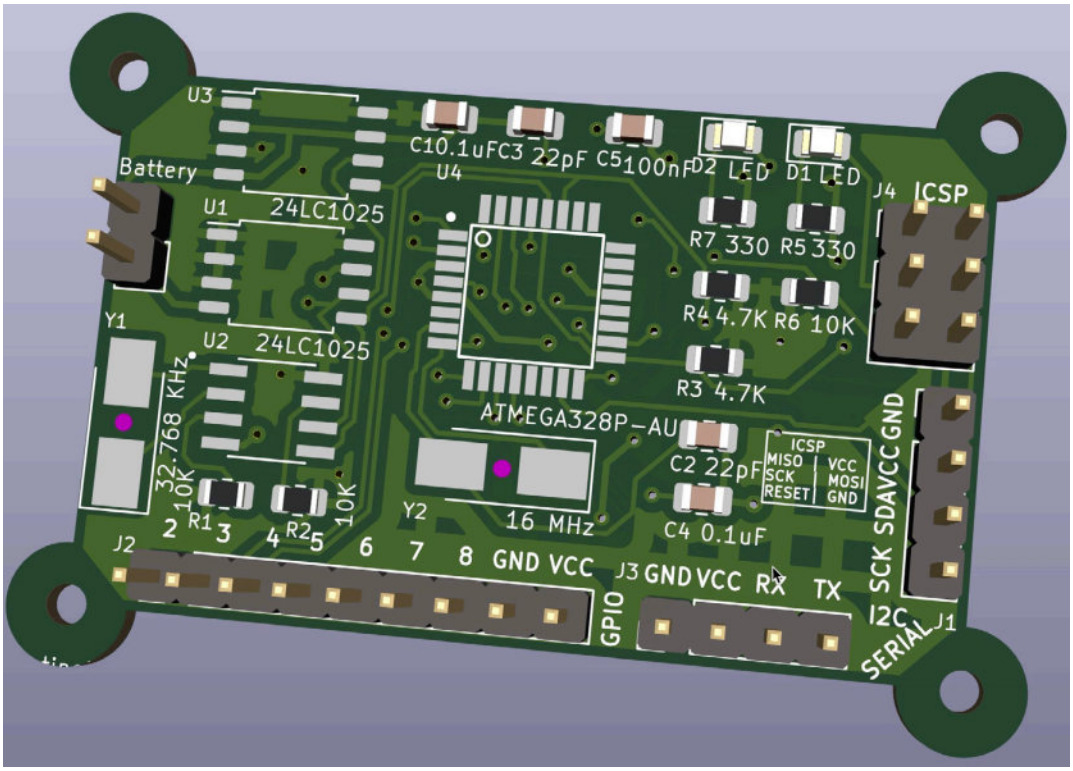
5. Add other elements, such as the board name and version number, logos, etc.

Below you can see the latest version of my board, with a few outstanding issues that I will address later:



11.5.8.27: Making progress with the silkscreen elements.

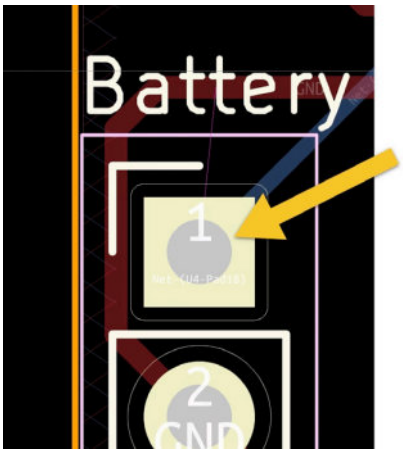
And here is a 3D rendering of the front side of the board:



11.5.8.28: A 3D render of the front of the board showing the silkscreen progress.

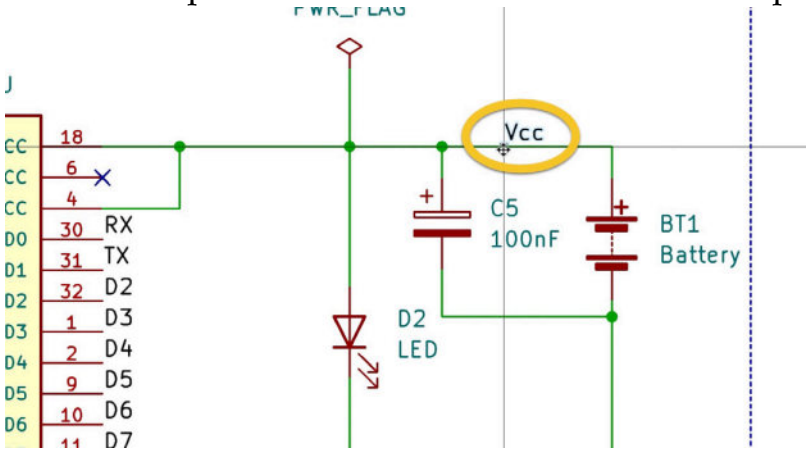
The back and front silkscreen are not yet complete. The DRC shows several violations involving silkscreen elements ("silkscreen overlap") and clearance violations. I have also noticed a couple of text labels that I must make invisible (such as the value for the lower-left mounting hole footprint, see Figure 11.5.8.26). I will fix these issues in the next segment of this chapter since they require special attention.

I have also noticed that pad one of the battery connector is not connected to the Vcc net. This reveals a bug in the schematic that the ERC was not able to pick. See below:



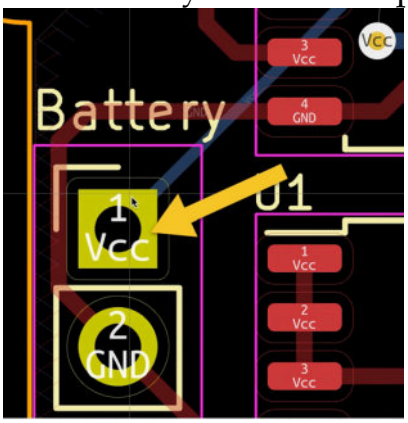
11.5.8.29: Pad one should be connected to Vcc.

To fix this problem, return to Eeschema, and add the Vcc net label to the wire that connects to the positive electrode of Bt1. See the example below:



11.5.8.30: Fixing a bug in the schematic.

Save the schematic and return it to Pcbnew. Update the PCB from the schematic, and confirm that the battery connector's pad one belongs to Vcc:



11.5.8.31: Pad one is now connected to Vcc.

This fix did not have any follow-on implications and did not introduce new problems. Remember that the schematic now contains changes that will be committed to the "2layer" branch. You will need to merge this change to the main branch.

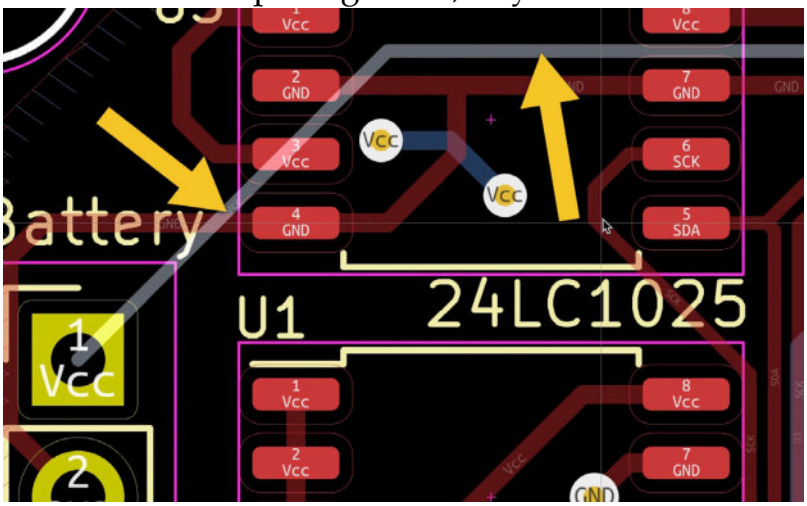
Let's continue to the next segment to fix the remaining DRC violations.

5.9. Layout 4 - Routing violations and complete silkscreen

The work so far has left several DRC violations that I will fix now. There is one design defect, however, that the DRC did not find. This defect relates to the battery connector's pad one fix I made at the end of the previous segment (where I attached it to the Vcc net).

I will work on the battery connector's pad one issue first and then clear the DRC violation.

When I attached pad one of the battery connector to the Vcc net, KiCad did not automatically apply the Power net class track width to the tracks connected to pad one. I will need to do this manually. The track that comes out of pad one contains multiple segments, as you can see below:

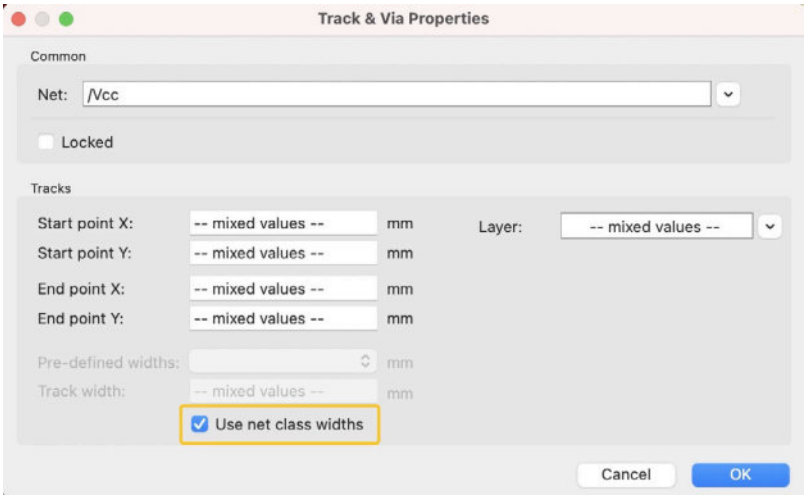


11.5.9.32: Vcc track from pad one of the battery connector.

The fastest way to apply the Power class's track width to the entire track is to use the "Select All Tracks in Net" option in the selected track context menu. Right-click anywhere on the track, and click on "Select All Tracks in Net" under "Select":

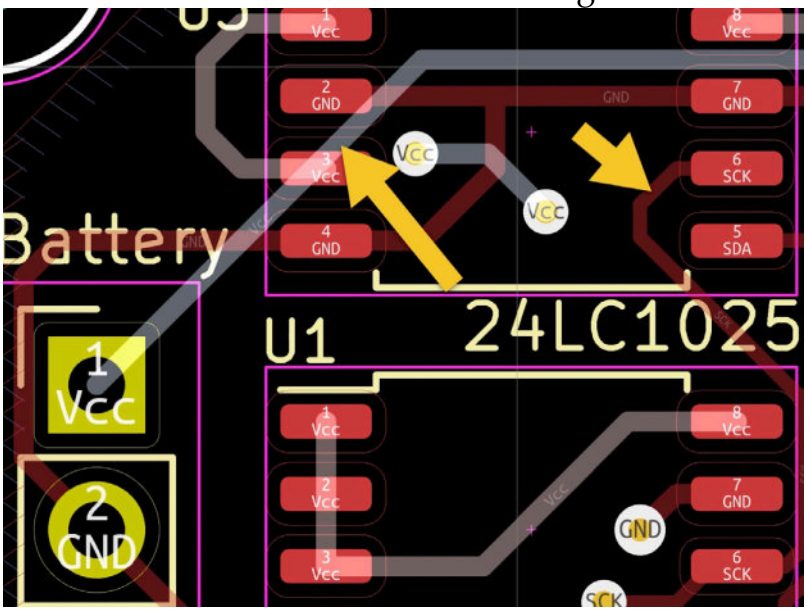
11.5.9.33: Select all tracks in net.

Then, right-click again on the track, and select Properties. In the Properties window, check the "Use net class widths" option, and click OK.



11.5.9.34: Using net class widths for the Vcc track.

The Vcc track from pad one of the battery connector should now appear slightly wider than the rest of the tracks. In the example below, the two arrows point to the edited Vcc track and a track that belongs to the Default class net.



11.5.9.35: Power nets are wider then default nets.

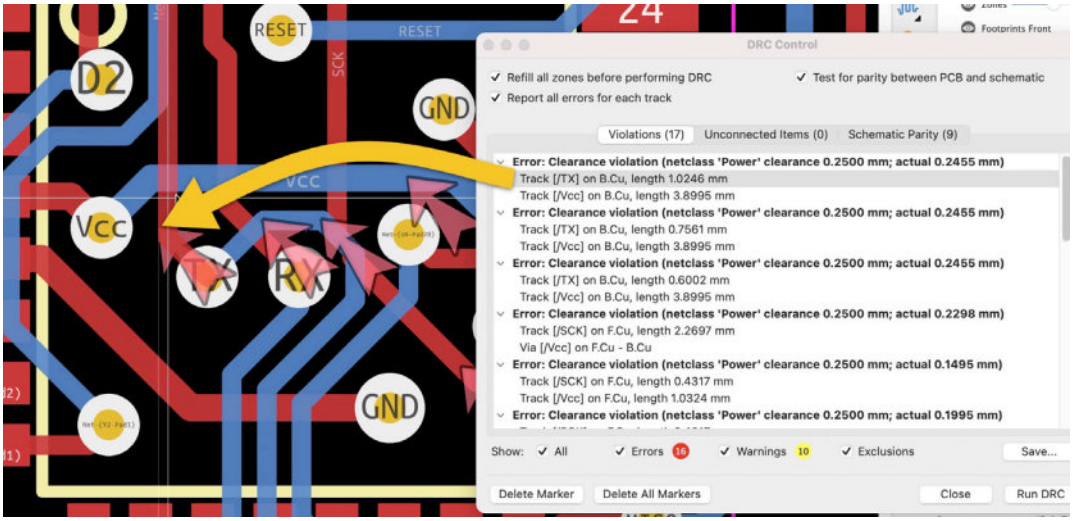
I will now continue with the DRC violations. There are two types of violations that are outstanding:

1. Clearance violations caused by Vcc net vias and tracks that are too close to other tracks. Remember that the Vcc track belongs to the Power net class and has increased clearance requirements.

2. Silkscreen overlaps. In some cases, two silkscreen items overlap. These are easy to fix by moving one or both items so that they don't overlap.

My process is one of step-by-step elimination. After running the DRC to get a fresh list of violations, I use the violations list as my to-do list. I work my

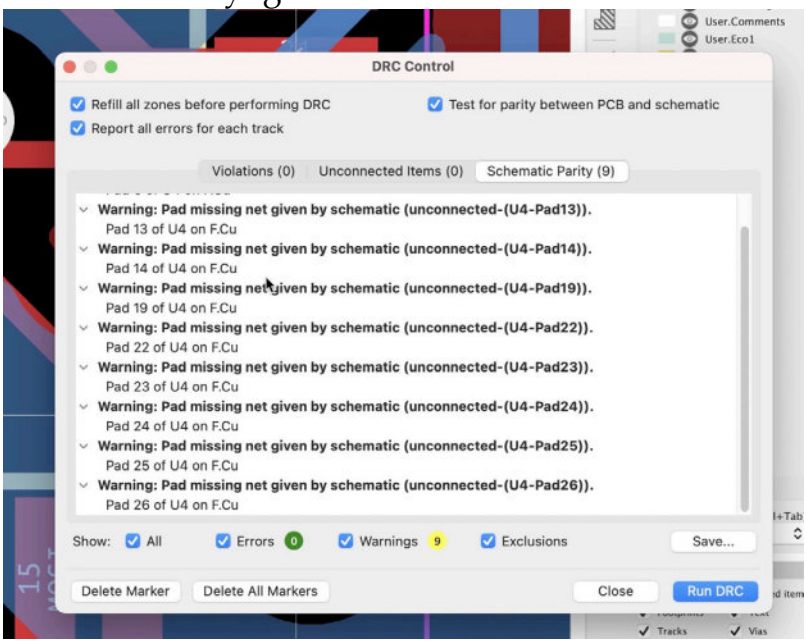
way from the top of the list until all items are cleared. In the example below, the first item in the list indicates that a track is too close to a Vcc via:



11.5.9.36: A typical clearance violation.

I will drag the red track towards the right, away from the Vcc via to clear this violation. When finished re-drawing, rerun the DRC to update the list. You should have one fewer violation.

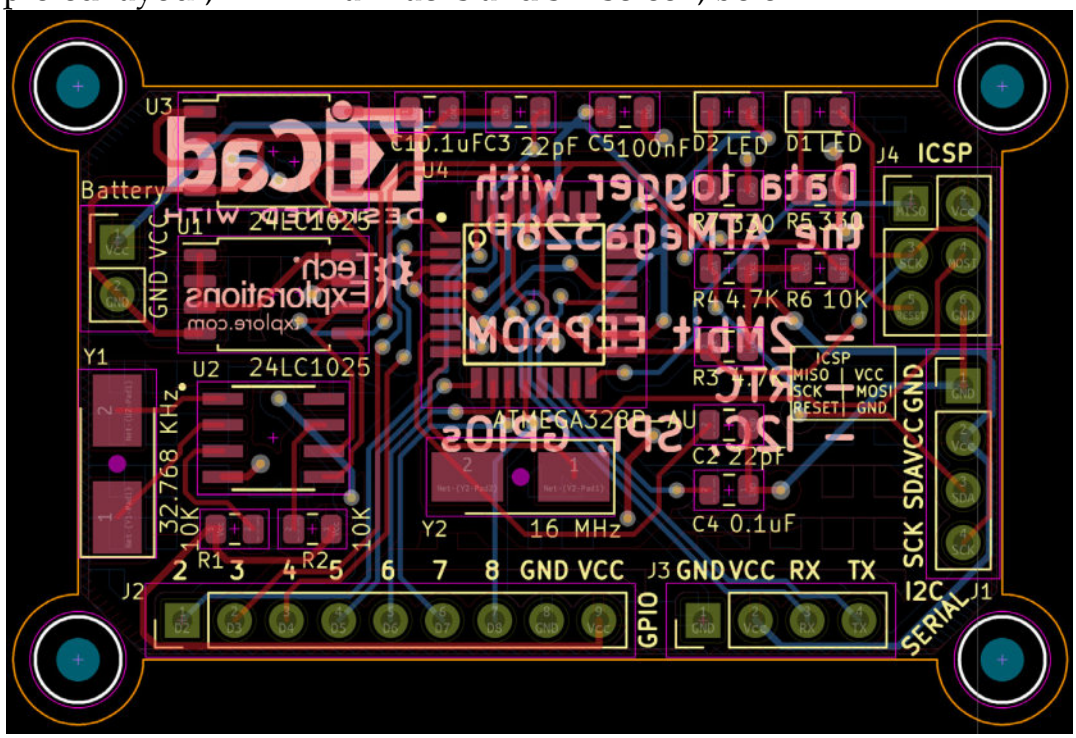
Continue in the same way until you get to zero unconnected items and violations. You can see my last DRC below, showing only nine schematic parity issues that I can safely ignore:



11.5.9.37: Zero violations and unconnected items.

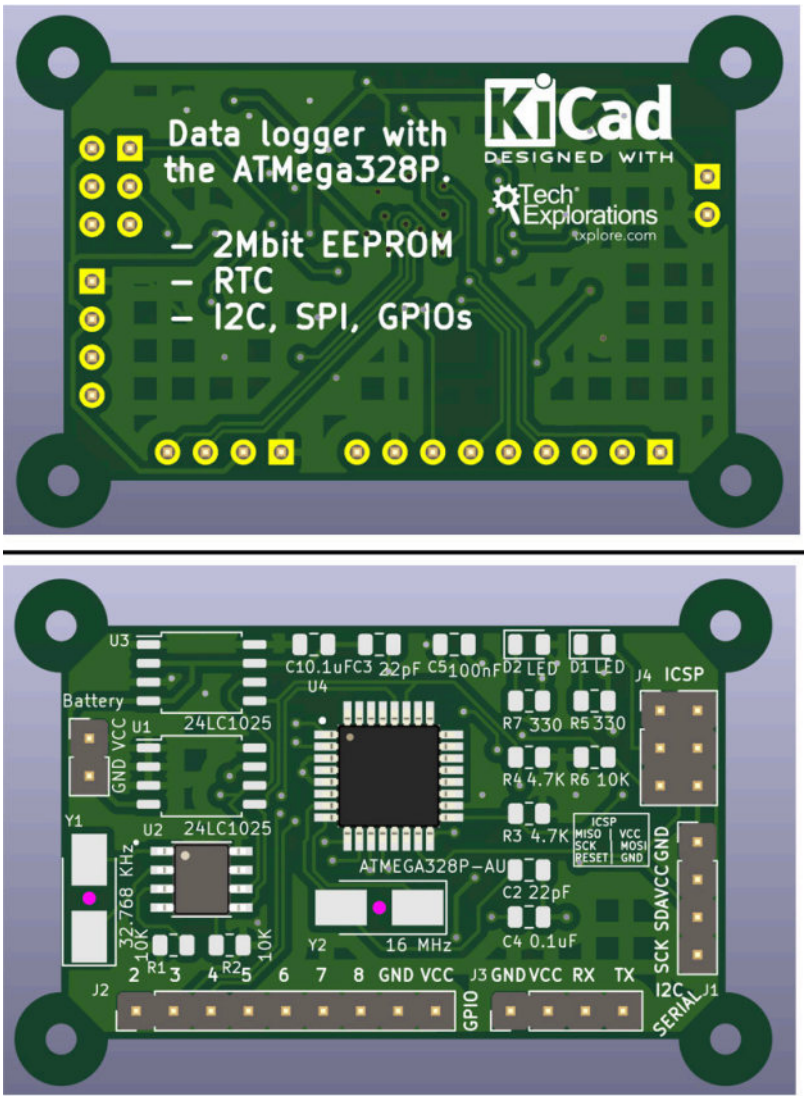
With all DRC issues fixed (and some ignored), I will finish this segment by completing the silkscreen work. I want to add a few text labels with board information and two logos in the back silkscreen layer. I will also add "Vcc"

and "GND "text labels in the battery connector terminals. You can see the completed layout, with final tracks and silkscreen, below:



11.5.9.38: Final version of the two-layer board.

In the figure above, I have reduced the opaqueness of all elements except for the silkscreen. This way, it is easier to see the silkscreen text and graphics. The 3D viewer produces a better view:



11.5.9.39: A 3D rendering of the board.

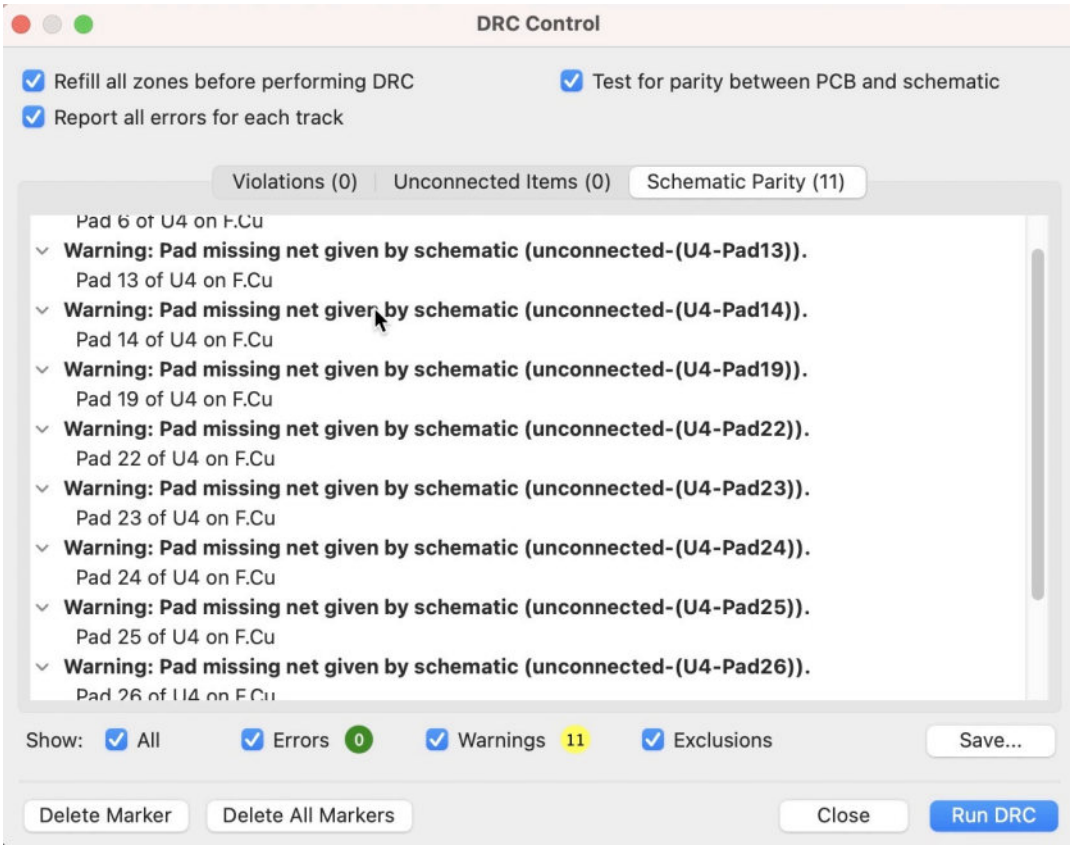
Let's save the changes and commit them to the Git repository:

```
% git status
On branch 2layer
Changes not staged for commit:
  (use "git add <fi1e>...” to update what will be committed)
  (use "git checkout —— <file>..." to discard changes in working
directory)

    modified: MCU Datalogger.kicad_pcb
    modified: MCU Datalogger.kicad_prl
    modified: MCU Datalogger.kicad_sch

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Fixed outstanding issues."
[2layer f773817] Fixed outstanding issues.
  3 files changed, 7230 insertions(+), 2214 deletions(-)
%
```

You are close to completing this project. We'll do a final DRC in the next segment and then export the Gerber files and upload them to the manufacturer's website.

5.10. Layout 6 - Design Rules Check

You have done a lot of work on this PCB, and you are very close to passing the finish line. In the previous segment of this chapter, I spent additional time fixing violations that I introduced when I changed the width of the Vcc track that came out of pad one of the battery connector. It is time for one last DRC before exporting the Gerber files to ensure that I (or you) haven't missed any violations.



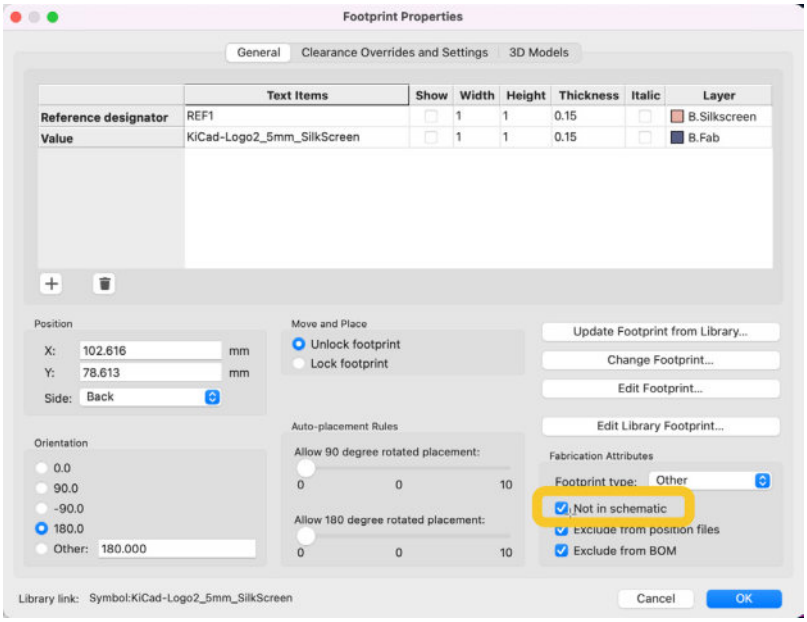
11.5.10.40: The last DRC reveals several warnings.

The last DRC reveals several schematic parity warnings. Here's a list of the two warning types and what they mean:

- "Warning: Extra footprint. " This warning relates to a footprint that exists in the layout editor that has no symbol counterpart in the schematic. I had two such warnings, one of each logo in the back layer (the Tech Explorations and KiCad logos). Such warnings are safe to ignore. In KiCad 6.0 or newer, you can mark a footprint as "Not in

schematic" to prevent similar warnings in the future. You will find the "Not in schematic" checkbox in the footprint's properties window (see screenshot below).

- "Warning: Pad missing net given by schematic…". This warning relates to pads that I have marked as "unconnected" in the schematic. Also safe to ignore.



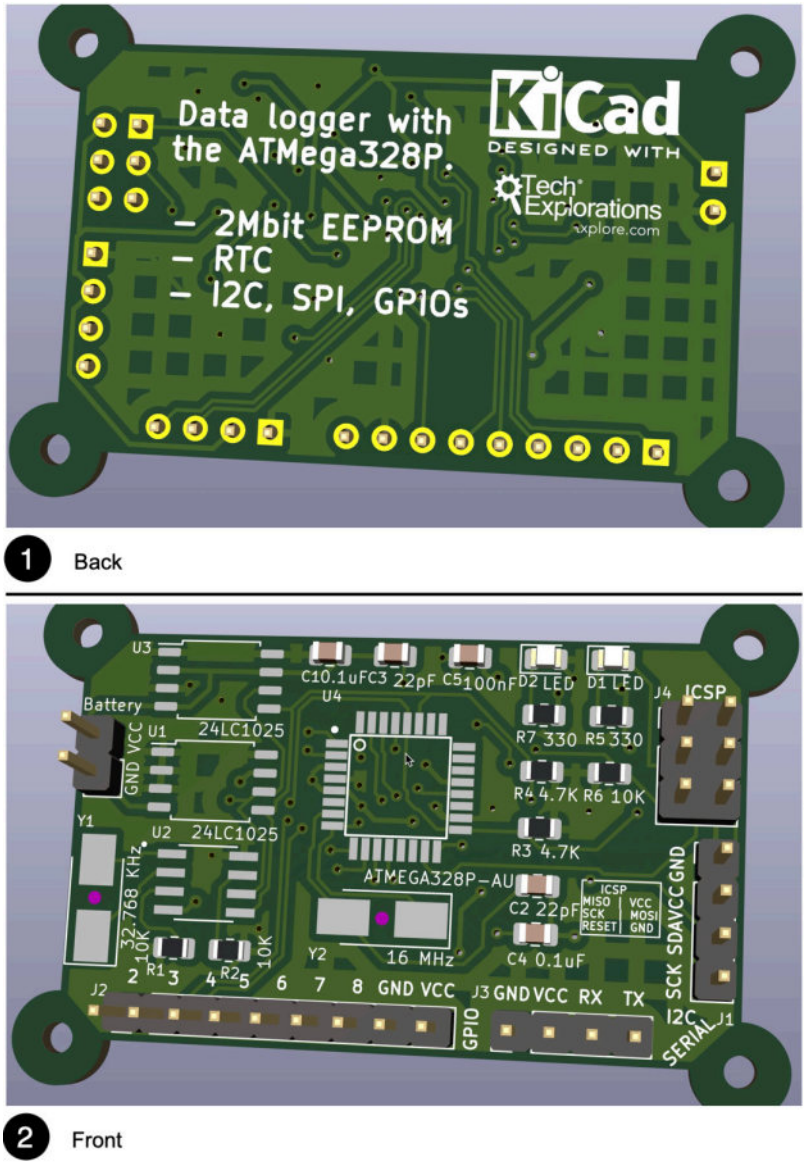
11.5.10.41: If a footprint exists only in the layout file, you can mark it as "Not in schematic" in the footprint's Properties window.

All of the items in the DRC belong to the two types of warnings that I can safely ignore. I can now declare that this board is ready to export and manufacture.

5.11. Layout 7 - Manufacture

Let's finish the layout design workflow. In this segment, you will export the Gerber files for your new board and upload them to your preferred manufacturer's website.

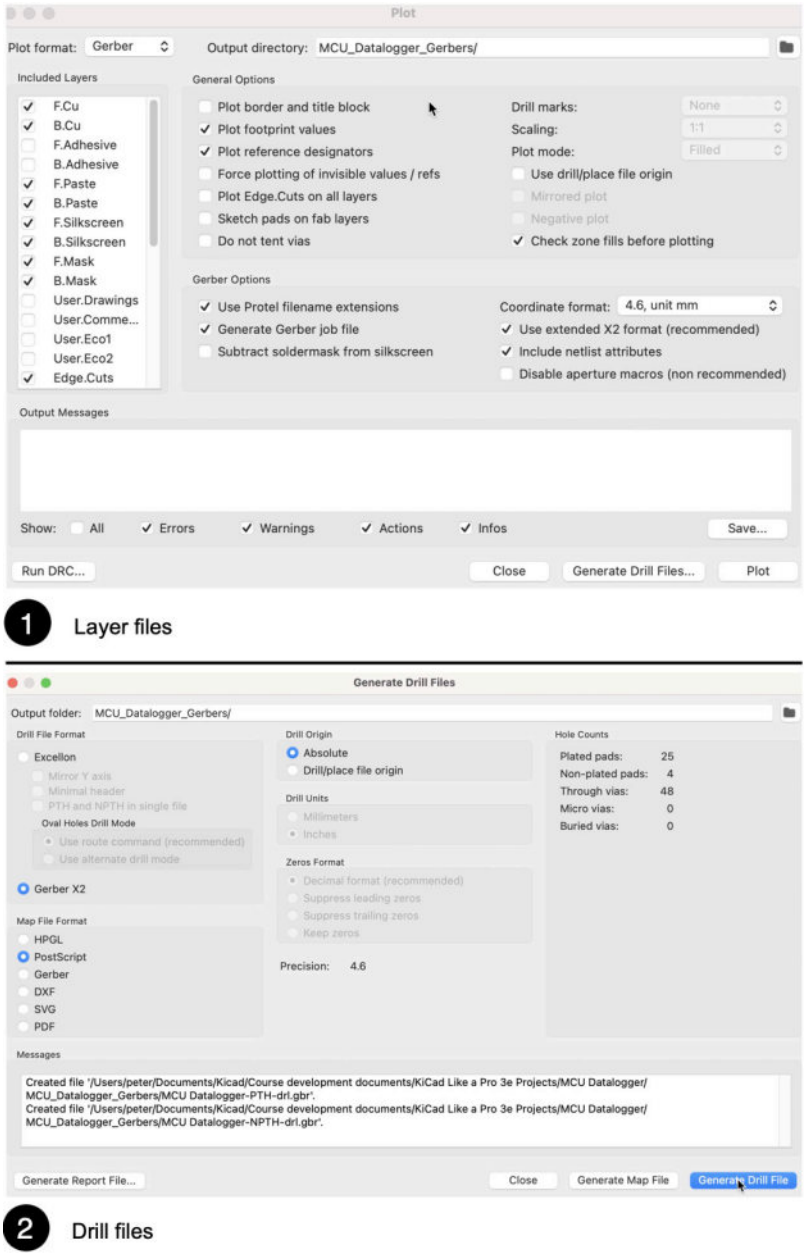
Start by doing a visual inspection of the front and back layers using the 3D viewer. Check for issues in the silk-screen, pads, copper fills, and missing or mal-positioned components.



11.5.11.42: Inspecting the board in the 3D viewer.

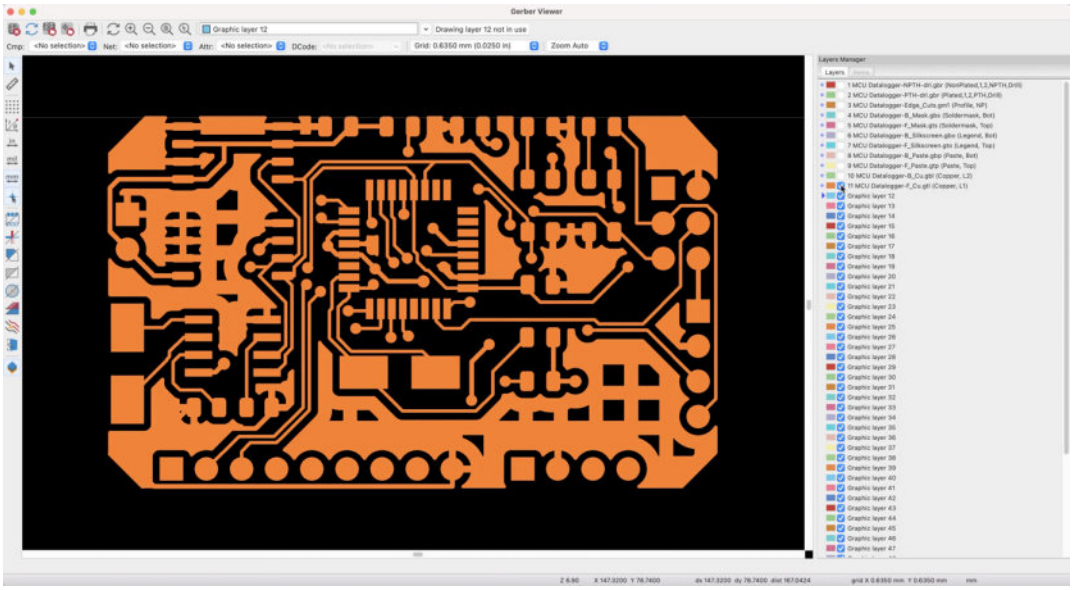
The only element that I changed after looking at the 3D rendering of the board was to move the ICSP pinout table upwards so that it is closer to the ICSP header.

Save the design, and open the plotter window (see chapter "17. How to export and test Gerber files" if you need a refresher on how to export the Gerber files). You can see my settings for the Gerber and drill files below:



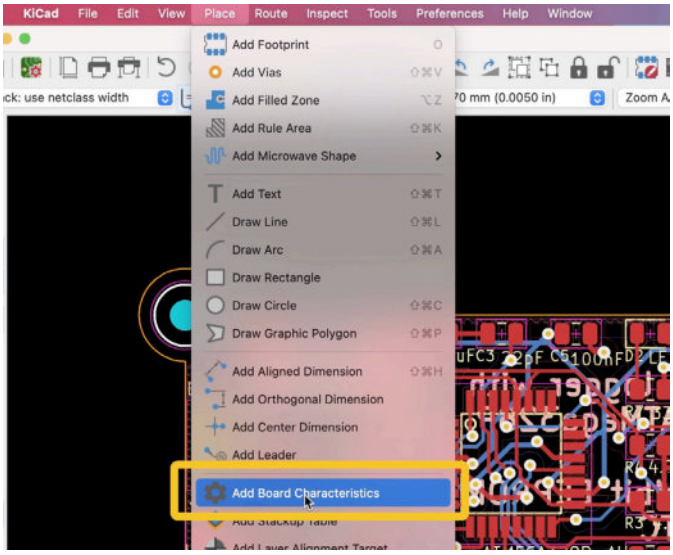
11.5.11.43: Generating the Gerber files.

Open the KiCad Gerber viewer to inspect the new Gerber files. Enable and disable each layer and assess their validity. Is anything missing? Is something not looking correct? Now is the time to address any issue that may affect the manufactured PCB.



11.5.11.44: The Gerber viewer displaying the front copper layer.

I did not see any problems in my instance of the Gerber files, so I continued with the upload to the manufacturer. To place the order on my manufacturer's website, I need the dimensions of the board. So, go back to Pcbnew and use the measurement tool to get the width and height of the board. An alternative method is to add the full board characteristics to one of the user layers. Use "Add Board Characteristics" under the Place menu item:

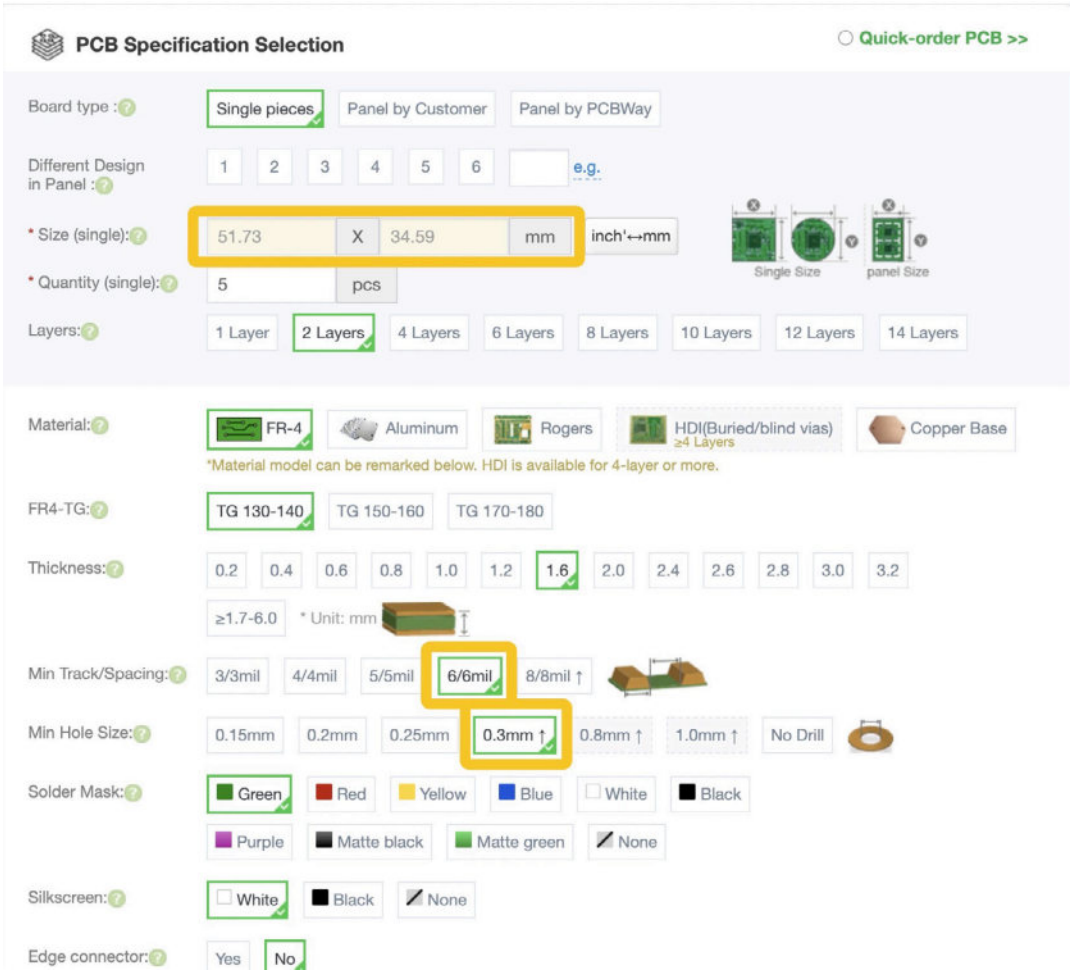


11.5.11.45: Adding the board characteristic to the design.

As you can see below, this board measures 51.73 mm X 34.59 mm:

11.5.11.46: The board dimensions, and other information.

For this board, I will use PCBway as the manufacturer. You can see the order form below after I have entered the board dimensions:



11.5.11.47: My PCB order form.

Most of the options in this order form as straightforward, but there are a couple I'd like to confirm: the minimum track spacing and minimum hole size. The values I have marked in the figure above will give you the lowest manufacturing price for the board. But are they correct?

My board characteristics information shows that minimum track spacing is 0.2 mm and minimum hole diameter is 0.3 mm. I have selected 6 mil for the spacing and 0.3 mm for the hole diameter in the order form. With the help of a converter, I learn that 0.2 mm is 7.87 mils. Therefore, the manufacturer's minimum track tolerance is lower than that of my board, and as a result, I can continue with the order without making any changes. If my board's minimum track spacing were lower than my selection in the order form, I would have had to select a smaller minimum track spacing, such as 4 or 5 mils.

The hole diameter at 0.3 mm for both my board and the order form are also in agreement. Therefore, I can continue with the order.

There is one last thing to do before you complete the two-layer PCB project: commit your latest changes to the Git repository. Here is my command line session:

```
% git status
On branch 2layer
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: MCU Datalogger.kicad_pcb
        modified: MCU Datalogger.kicad_prl

Untracked files:
  (use "git add <File>..." to include in what will be committed)

        MCU_Datalogger_Gerbers/

no changes added to commit (use "git add" and/or "git commit -a")
% git commit —am "Completed 2—layer PCB project."
[2layer bbeffea] Completed 2—layer PCB project.
  2 files changed, 236 insertions(+), 167 deletions(—)
%
```

In the Git command line session above, you can see that Git detected new untracked files in the "MCU_Datalogger_Gerbers" directory. I chose not to track those files because they are not needed. Gerber files can always be generated, and keeping them in the repository would only increase bloat.

At this point, the project is complete. However, there are two more things that I would like to show you.

First, how to add the missing 3D shapes so that the 3D render of the board looks as realistic as possible.

Second, how to use the same schematic to produce a four-layer version of the board. This will also allow you to practice your Git skills further.

6. 3D shapes

In this chapter, you will improve the realism of the 3D viewer. Currently, the 3D viewer renders the board where only the resistors, capacitors, and pin headers have 3D models. Let's address this by adding the 3D models for the remaining footprints.

In the figure below, you can see the current 3D viewer rendering (left) and its look at the end of this chapter (right).

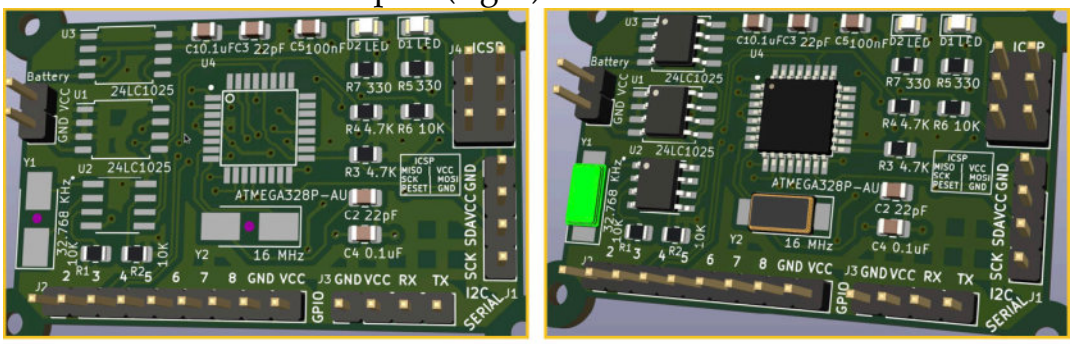


Figure 11.6.1: “Before” and “After” renders from the 3D viewer.

To add the missing 3D models, you will need to open the properties window for each related footprint. These footprints are missing a 3D model:

- U4 (the MCU).
- U3, U1, U2 (the two EEPROM chips and the real-time clock).
- Y1 and Y2 (the two crystal oscillators).

If you need a reminder about adding a 3D model to a footprint, please read the relevant chapter.

For the MCU footprint, I used the TQFP-32_7x7mm model. This model is available in the "Package_QFP" library that comes with KiCad. You can see my settings below:

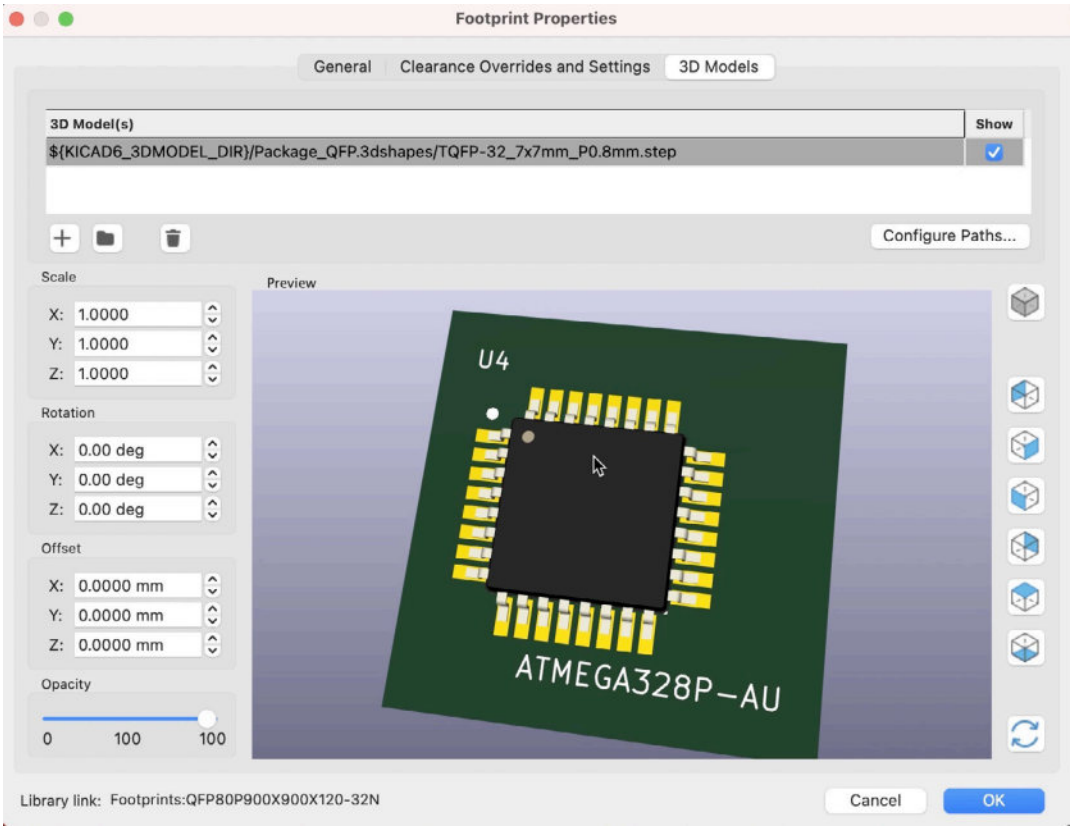


Figure 11.6.2: Settings for the MCU 3D model.

Continue with U1. The 3D model I used for this footprint is the "SOIC-8_3.9x4.9mm_P1.27mm.wrl" in the "Package_SO" library. See my configuration below:

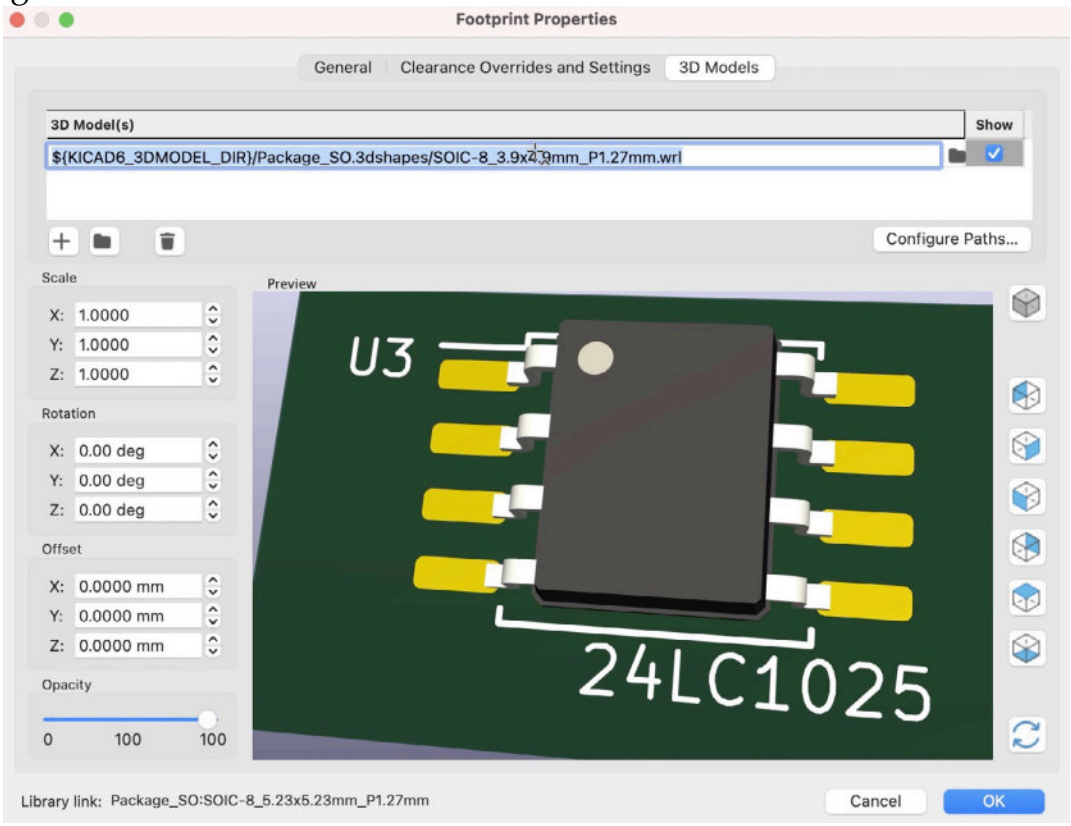


Figure 11.6.3: Settings for the EEPROM chip 3D model.

I used the same model for the second EEPROM chip and the real-time clock. Copy the 3D model path and file, and paste it in the 3D model's row for U2 and U3.

Two models remain for the crystal oscillators. For Y2, use model "Crystal_SMD_0603-2Pin_6.0x3.5mm" in the "Crystal" library:

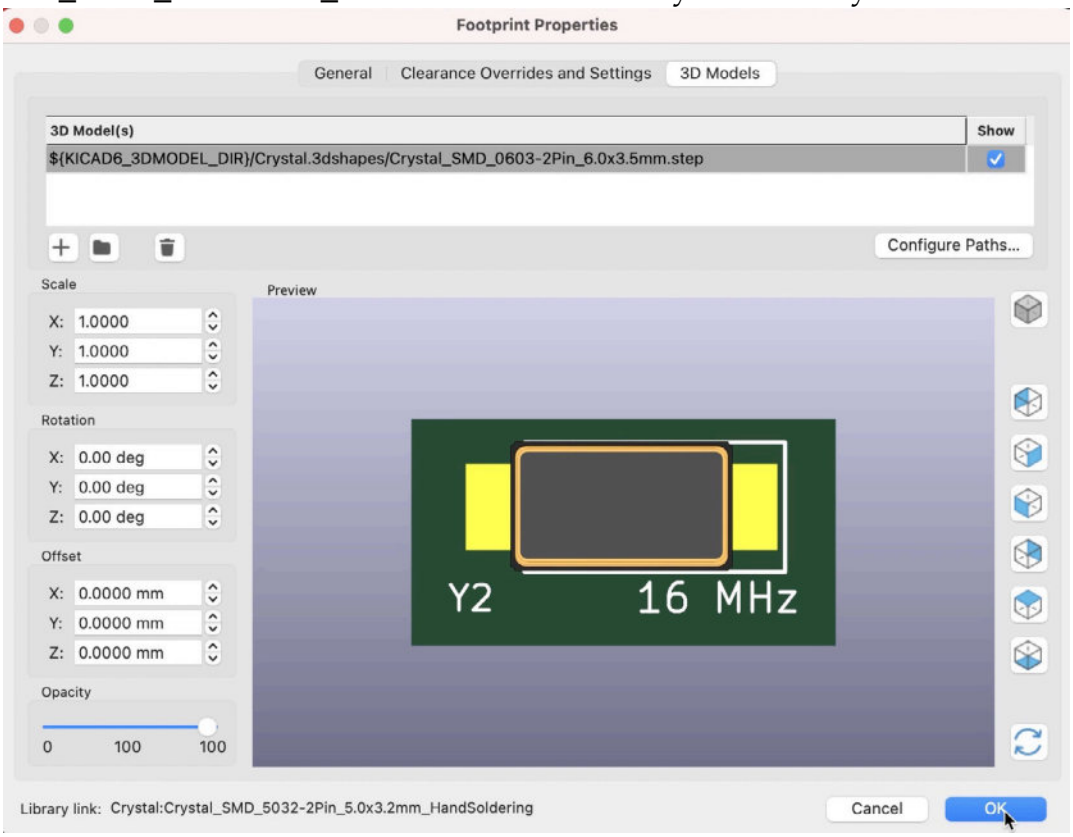


Figure 11.6.4: Settings for the Y2 crystal oscillator.

For Y1, use the same model as in Y2 to copy the model path across.

All footprints now have a 3D model. Here's the resulting 3D rendering:

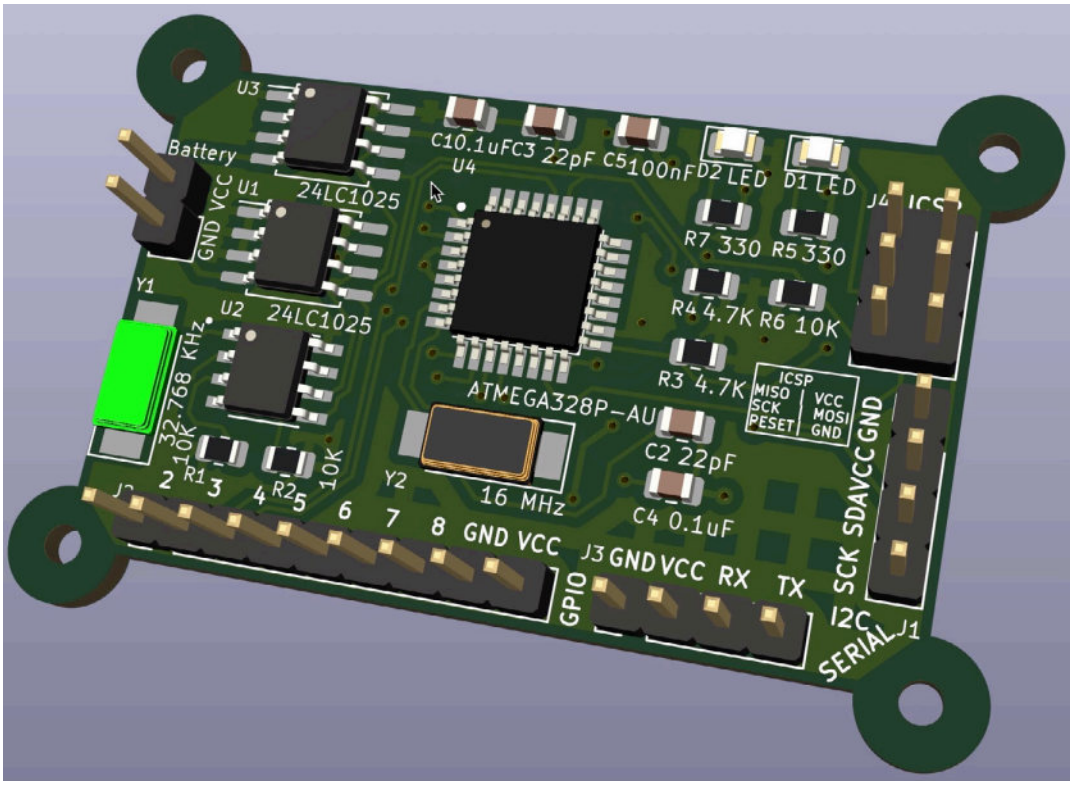


Figure 11.6.5: 3D render of the board with all 3D models assigned.

The 3D viewer can now produce a more realistic rendering of the board.

7. Merge 2-layer branch to main

In this chapter, you will merge the "2layer" Git repository branch into "main." The "2layer" branch contains all changes relating to your work in the two-layer PCB. In the next chapter, you will begin work on a four-layer version of the PCB. But first, merging "2layer" into "main" will ensure that "main" contains the latest version of the schematic.

Remember that I found an error in the schematic during the layout design, which I fixed and committed the change in the current working branch ("2layer"). I plan to create a new working branch for the four-layer PCB, which will branch out of "main."

Another benefit of merging "2layer" into "main" is that the merge will also bring all two-layer layout changes into "main." This means that the new "4layer" branch will inherit the layout from the "main" branch, along with the refined outline in the Edge.Cuts layer, placement of the footprints, and even the 3D models. To design the four-layer version of the PCB, you will only need to delete the current copper tracks and fills, change the board layout to four layers, and redraw tracks and fills.

Before working with Git, I suggest one naming convention change. Currently, the name of the folder that contains the two-layer Gerbers is "MCU_Datalogger_Gerbers." Change this to "2_layer_MCU_Datalogger_Gerbers". Soon, you will have a second Gerbers directory called "4_layer_MCU_Datalogger_Gerbers". You will not track these folders in Git, but they will co-exist in your project directory.

Let's continue with Git. Ensure that you have saved all your work. Open your terminal window. Check the current status of the repository with the "status" command:

```
% git status
On branch 2layer
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: MCU Datalogger.kicad_pcb
```

```
    modified: MCU Datalogger.kicad_prl
    modified: MCU Datalogger.kicad_sch

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    2_layer_MCU_Datalogger_Gerbers/

no changes added to commit (use ”git add" and/or "git commit –a")
%
```

You can leave the Gerbers directory untracked. Commit the three changed and tracked files to the working branch:

```
% git commit –am "Completed 2–layer PCB."
[2layer a49la4c] Completed 2–layer PCB.
  3 files changed, 18 insertions(+), 9 deletions(–)
%
```

The 2layer branch contains all changes. Continue to merge the changes in 2layer with the main branch. You can do this in two steps:

1. Switch your working branch to "main" using the "checkout" command.
2. Merge "2layer" into "main" using the "merge" command.

Here is my Git command line session:

```
% git checkout main
Switched to branch 'main'
% git status
On branch main
Untracked files:
  (use "git add <fi1e>..." to include in what will be committed)

    2_layer_MCU_Datalogger_Gerbers/

nothing added to commit but untracked files present (use "git add"
to track)
% git merge 2layer
Updating c1a0d5e..a491a4c
Fast–forward
  MCU Datalogger.kicad_pcb | 15236 ++++++++++++++++++++++++++++++++
++++++++–––––
  MCU Datalogger.kiEad_pr1 | 26 +–
  MCU Datalogger.kicad_pro | 8 +–
  MCU Datalogger.kicad_sch | 5 +–
4 files changed, 14092 insertions(+), 1183 deletions(–)
%
```

From this point, until you switch to a different branch, you will be working in the main Git branch.

To avoid Git reminding me to track the Gerber directories, I will add them to the ".gitignore" file. Below you can see the updated contents of this file:

```
.DS_Store
fp-info-cache
MCU Datalogger-backups/*
2_layer_MCU_Datalogger_Gerbers/*
4_layer_MCU_Datalogger_Gerbers/*
```

Save the updated ".gitignore" file, and commit the change to the ".gitignore" file to the main branch:

```
% git status
On branch main
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: .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Added gerber directories to gitignore."
[main ecf6970] Added gerber directories to gitignore.
1 file changed, 2 insertions(+), 1 deletion(-)
%
```

In the next chapter, you will begin work in the four-layer version of the PCB, and to track changes in the layout, you will create a new working branch.

8. Design 4 Layer PCB in new Git branch

Let's begin work on the four-layer version of the PCB by creating a new branch in the project's Git repository. Currently, the repository contains two branches. The current working branch is "main" (identified as active by the asterisk). The "2layer" branch contains changes relating to the two-layer version of the PCB. The two branches are in sync, and there are no uncommitted changes.

You can use the "branch" command to get information about repository branches:

```
% git branch
  2layer
* main
%
```

Create a new branch to track changes in the four-layer version of the PCB. The name of the new branch is "4layer". Here is my command line session:

```
% git checkout -b 4layer
Switched to a new branch '4layer'
% git status
On branch 4layer

nothing to commit, working tree clean
%
```

You are now working in the "4layer" branch, which is clean (i.e., there are no pending changes" and tracking for changes.

In the next chapter, you will go straight into Pcbnew, change the board configuration to four layers, and remove and redraw all copper tracks and fills.

9. Four-layer PCB routing

Because of the layout work in the two-layer PCB already committed to the Git repository, most of the design work is already complete. Layout workflow steps 2 (outline and constraints), 3 (component placement), and 5 (silkscreen) is complete and does not require any modifications.

This chapter will re-work steps 1 (setup) and 4 (routing and copper fills). You will also do a DRC (step 6). In the next chapter, you will export the new Gerber files, check them for fitness, and upload them to the manufacturer.

Let's begin.

Open Pcbnew. First, remove all tracks and fills using the Global Deletions window from the Edit menu:

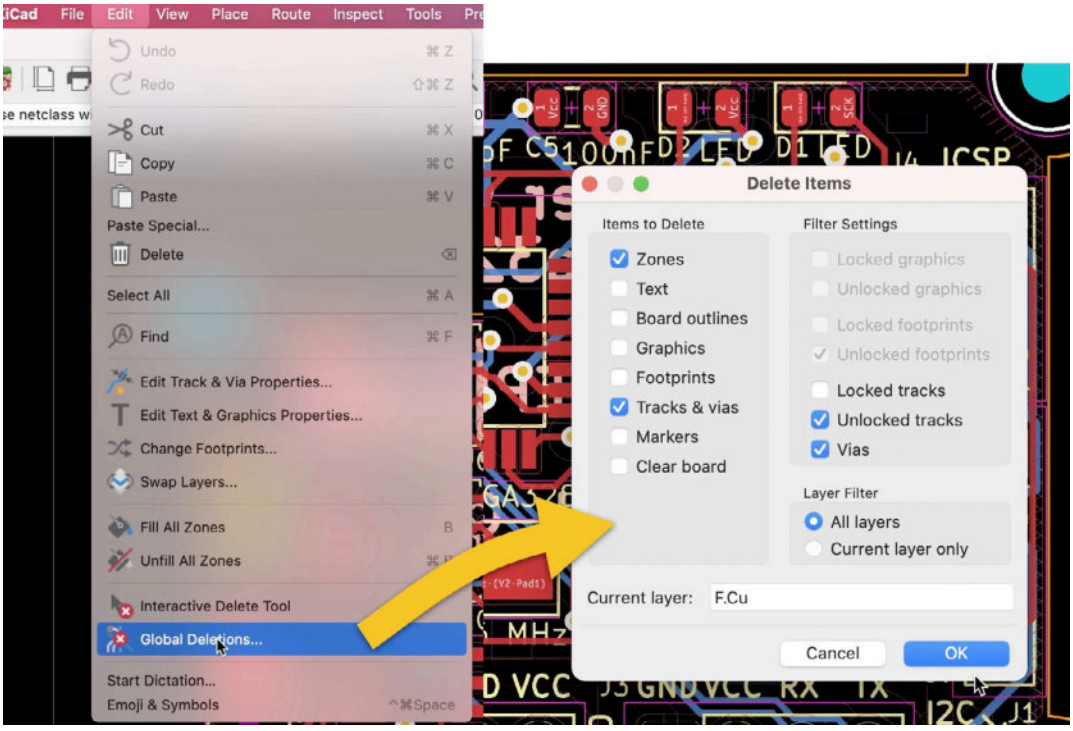


Figure 11.9.1: Delete all tracks, vias and zones.

The layout will retain all other design elements. Here is the current state of the board:

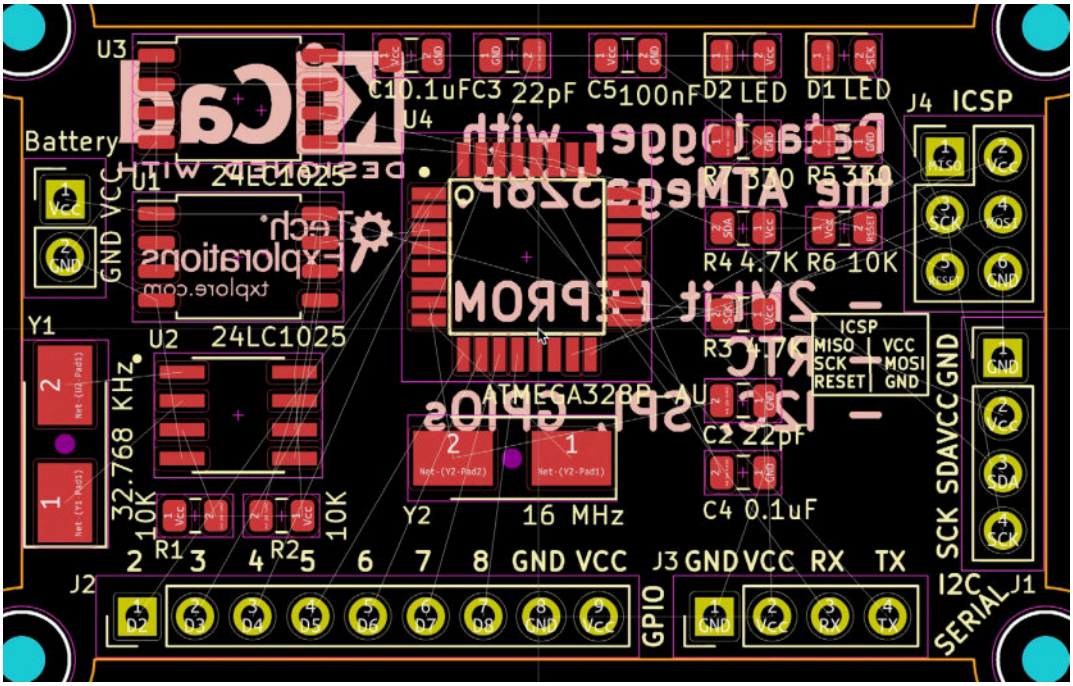


Figure 11.9.2: The board without tracks, vias, and filled zones.

In the Board Setup window, under Board Stackup, Physical Stackup set the number of copper layers to 4:

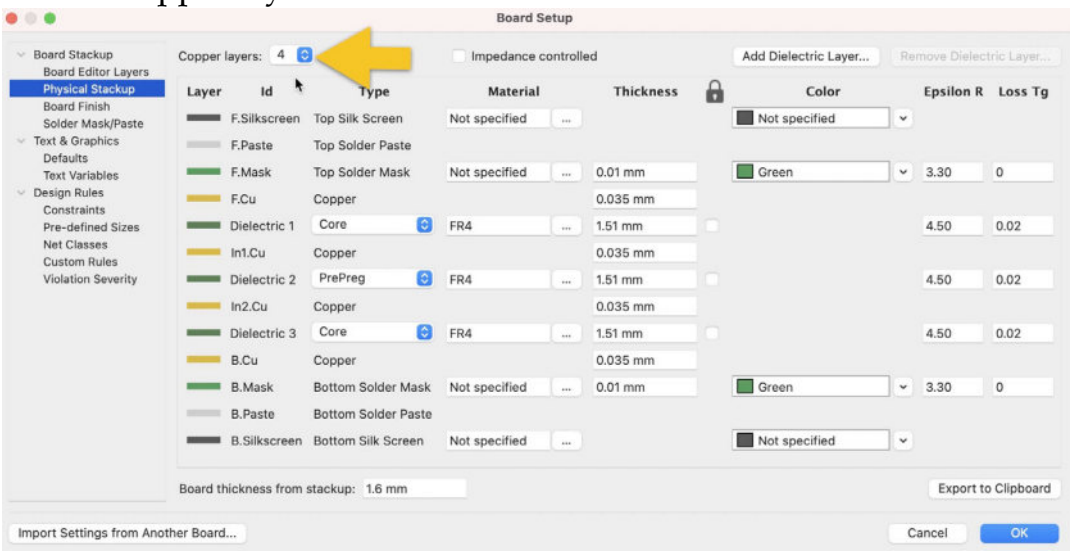


Figure 11.9.3: Switch the number of copper layers to “4”.

My routing strategy for the four-layer board is similar to the two-layer board. I tried to draw the route signal routes (that is, all routes other than those that belong to the Vcc and GND nets) in the top, In1, and In2 copper layers.

I will draw the first one connected to the GND net in the bottom layer and the second connected to the Vcc net in the In1 layer for the copper fills.

You can see the resulting four-layer PCB below:

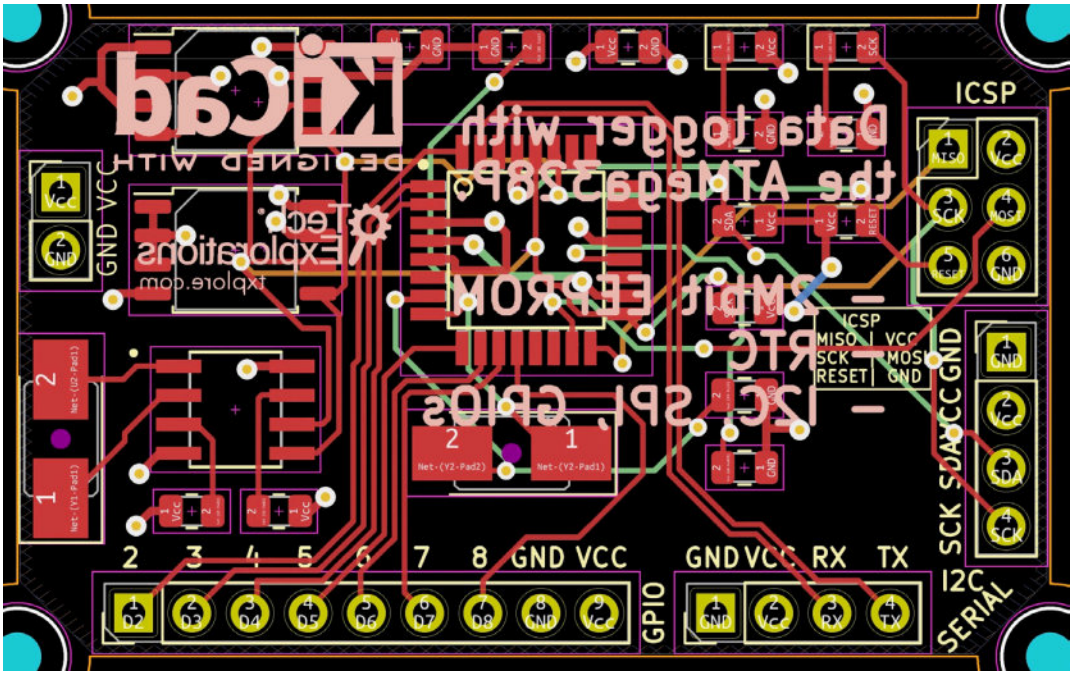


Figure 11.9.4: The completed four-layer PCB.

Do a DRC to ensure that the board does not contain any actionable violations.

Save the new layout, and commit the changes to Git:

```
% git status
On branch 4layer
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: MCU Datalogger.kicad_pcb
    modified: MCU Datalogger.kicad_prl

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Completed 4-layer pcb routing."
[4layer a316bc1] Completed 4-layer pcb routing.
  2 files changed, 12486 insertions(+), 8043 deletions(-)
%
```

Let's continue with the manufacturing of the board.

10. Four-layer PCB manufacturing

Let's manufacture the PCB. I will be using the same settings as the two-layer PCB, except I will now choose four layers instead of 2 in the order form. The board dimensions and tolerances remain unchanged.

Generate the Gerber files:

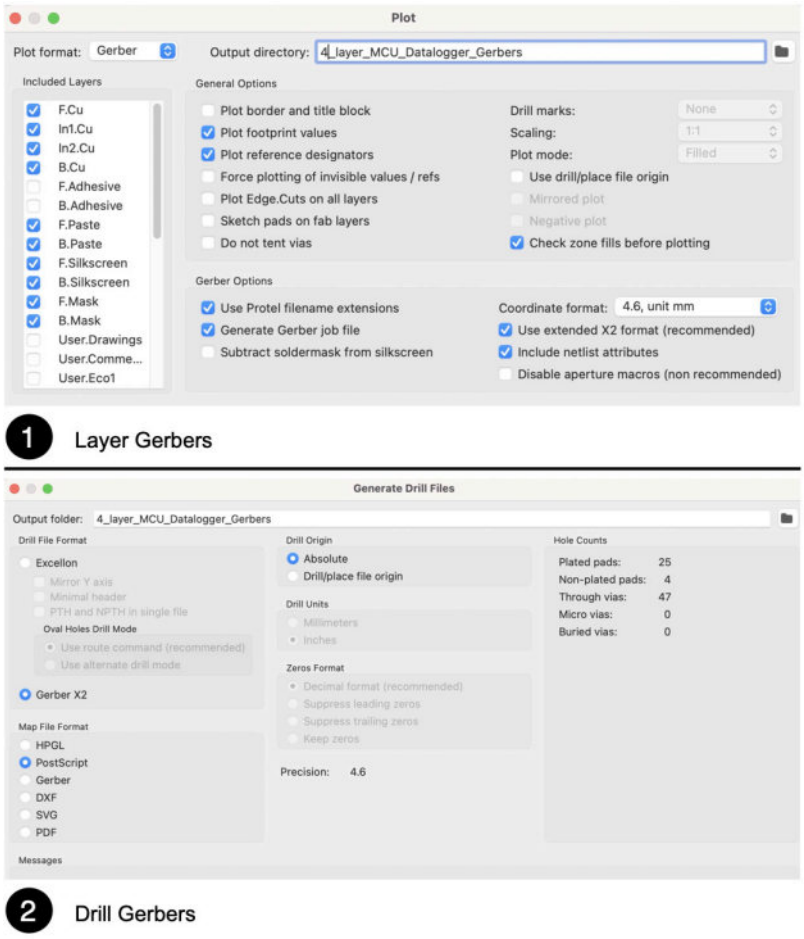


Figure 11.10.1: Generating the Gerber files.

Test the generated Gerber files using the KiCad Gerber Viewer app:

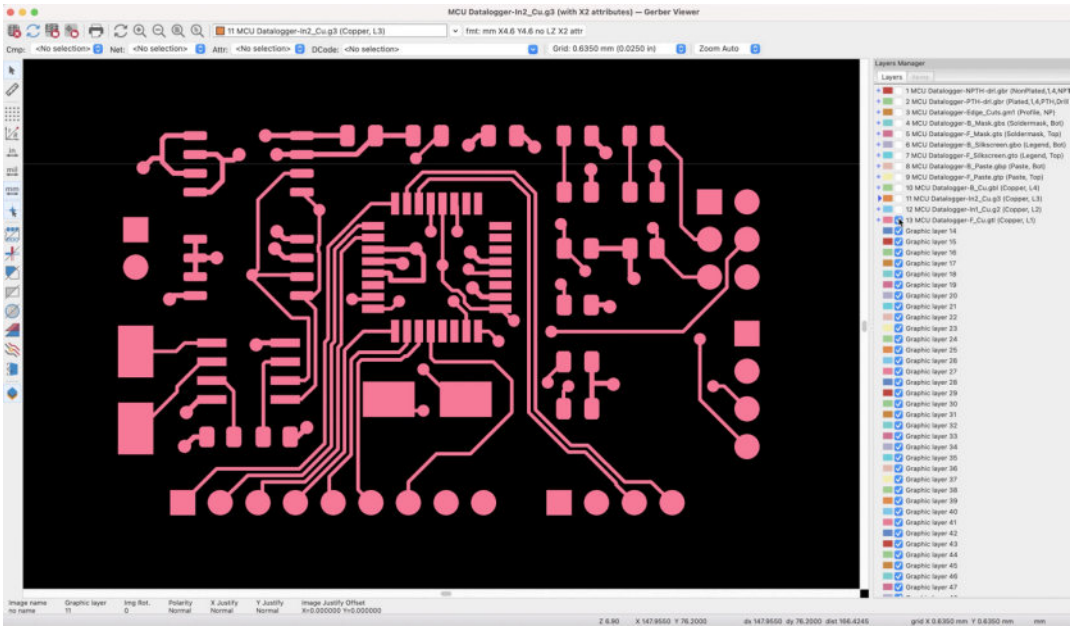


Figure 11.10.2: Testing the Gerber files in Gerber Viewer.

Finally, upload the Gerber files ZIP archive to the manufacturer's website. Below you can see my order settings:

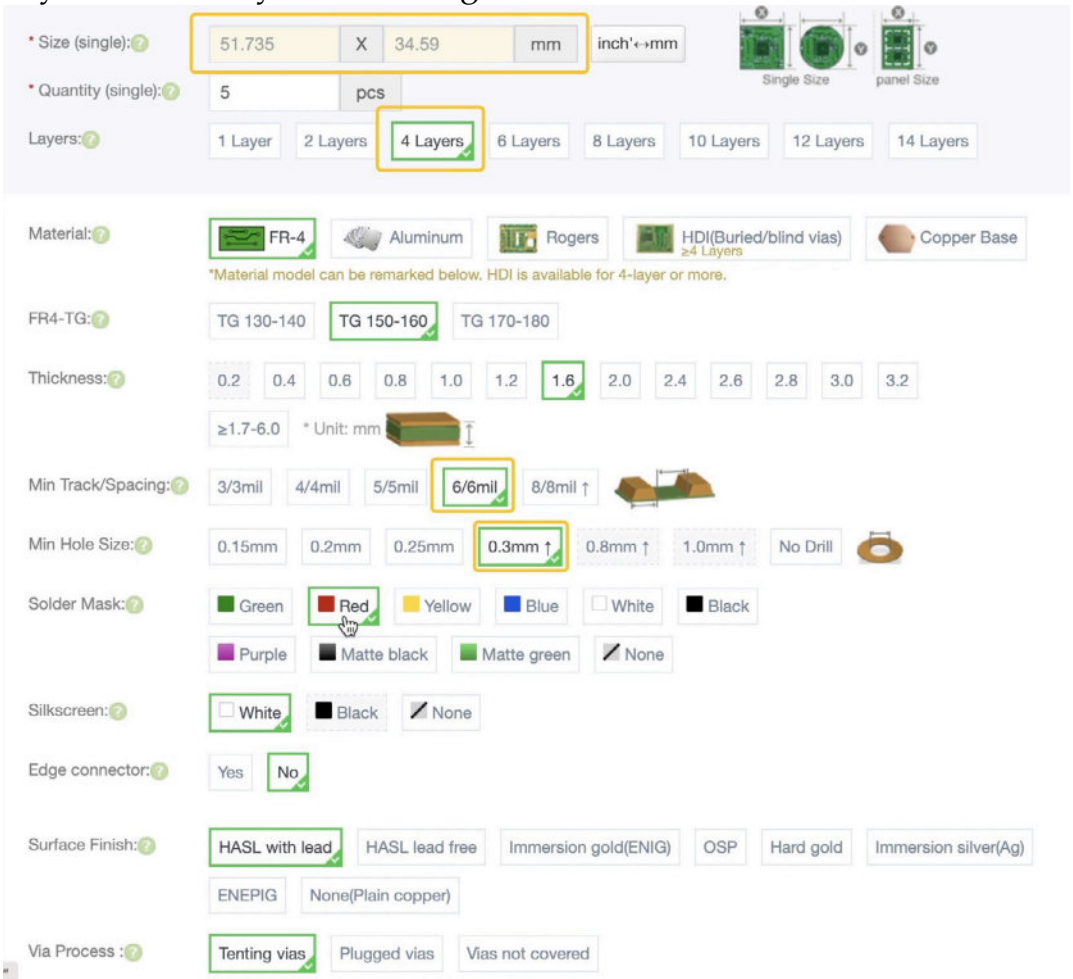


Figure 11.10.3: Four-layer PCB order form.

The only difference between the order form for the four-layer PCB and the one for the two-layer PCB is the number of layers (see above).

Let's finish this project by committing the changes to the "4layer" Git branch (save changes in the editor if you haven't done so already):

```
% git status
On branch 4layer
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: MCU Datalogger.kicad_pcb

no changes added to commit (use "git add" and/or "git commit -a”)
% git commit -am “Completed 4-layer PCB.”

[4layer f5b6b0b] Added board characteristics.
  2 Files changed, 55 insertions(+), SS deletions(—)
%
```

With this commit, this project is complete.

However, there is one more skill that I want to demonstrate in the next chapter. Imagine the situation where you need to change the schematic after you have completed work on the two versions of the PCB layout. At the file level, this change will exist in the schematic file of the repository's main branch. You will then need to sync the changes in this file to the other two branches ("2layer" and "4layer") without modifying the layout files.

How can you do this? I will show you in the next chapter.

11. Updating layout from changes to the schematic with Git

In this chapter, I will demonstrate a solution to the problem of how to use Git to import changes to the schematic into layouts that exist in different branches. This problem is common in situations where you have a KiCad project with a single schematic but more than one layout.

To solve this problem, you will need to use Git.

There is a single schematic tracked in the main branch in the MCU datalogger project that you completed. There are two versions of the layout; a two-layer layout tracked in the "2layer" branch, and a four-layer layout tracked in the "4layer" branch. A slightly different scenario would call for a single schematic informing an SMD and a THT version of a two-layer board.

In either case, you will likely want to make changes to the schematic at some point. For example, you may want to add new components or fix a bug. The individual schematic file will need to be synced between the main branch and the branches of the layouts.

You already know how to merge two branches. You can use the "merge" command. The problem with this option is that the" merge" command would merge the schematic file and all files between the branches involved. This is risky, as it could overwrite the layout files of one version of the PCB with that of another.

There is a safer option by which you can use Git to merge specific files between branches. This gives you maximum flexibility and removes all risks. In the case of the current project, it is possible to use Git to merge changes to the "MCU Datalogger.kicad_sch" file only between branches and ignore everything else. The Git feature that I will use in the example that follows is called “gitattributes”. You can use gitattributes in an extensive range of situations, but in this case, I will use it to set up a merge strategy to allow me to merge a single file between branches.

Let's look at an example using the MCU datalogger project as we left it in the previous chapter. Begin by getting the status and the current list of branches in the repository:

```
% git status
On branch 4layer
nothing to commit, working tree clean
% git branch
  2layer
* 4layer
  main
%
```

I am currently working on the "4layer" branch, and the repository contains three branches. Say that you want to make a change to the schematic. First, switch into the "main" branch:

```
% git checkout main
Switched to branch 'main'
%
```

Next, create a new file with the name ".gitattributes." You can see the content of this file below (a single line):

```
MCU Datalogger.kicad_pcb merge=pcb
```

This line informs Git to ignore files with "MCU Datalogger.kicad_pcb" during a merge operation. You may also use a wildcard character in the name, if you prefer, such as "*.kicad_pcb," which would cause Git to ignore all files with this extension. The name of this merge rule is "PCB." This is an arbitrary name, and you can change it if you wish.
Merge this new file into the main branch:

```
% git add .
% git commit –am "Added gitattributes file to preven PCB Files from
merge."
[main 3Zc16f8] Added gitattributes file to preven PCB files from
merge.
  1 file changed, 1 insertion(+)
  create mode 100644 .gitattributes
%
```

Open Eeschema, and make a change to the schematic. I have added a new net label in one of the un-named nets, as you can see below:

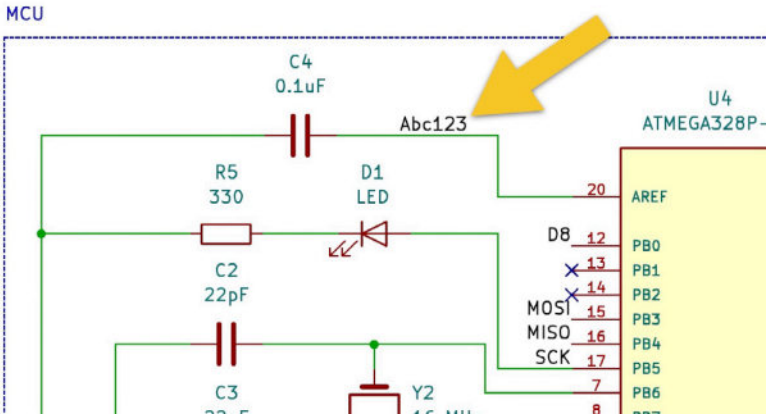


Figure 11.11.1: Made a change to the schematic.

Save the change, and commit it to the "main" branch:

```
% git commit -am "Made small change to schematic."
[main 7e48d78] Made small change to schematic.
1 file changed, 4 insertions(+)
%
```

Say that you would like to update the four-layer PCB with the latest change in the schematic. Switch your working branch to "4layer":

```
% git checkout 4layer
Switched to branch '4layer'
%
```

Next, merge the changes in "main" into "4layer". Git will you to type in a commit message. I typed this: "Merging an update to the schematic" and then saved the text editor to complete the merge operation.

```
% git merge main
Merge made by the 'recursive‘ strategy.
 .gitattributes | 1 +
 MCU Datalogger.kicad_sch | 4 ++++

 2 files changed, 5 insertions(+)
 create mode 100644 .gitattributes
```

Two files were merged. The merge contained the new ".gitattributes" file and the changes in the schematic file.
Confirm that the change in the schematic editor appears in the "4layer" branch. Close Eeschema if it is still open, and then re-open it. Confirm that the new net label appears:

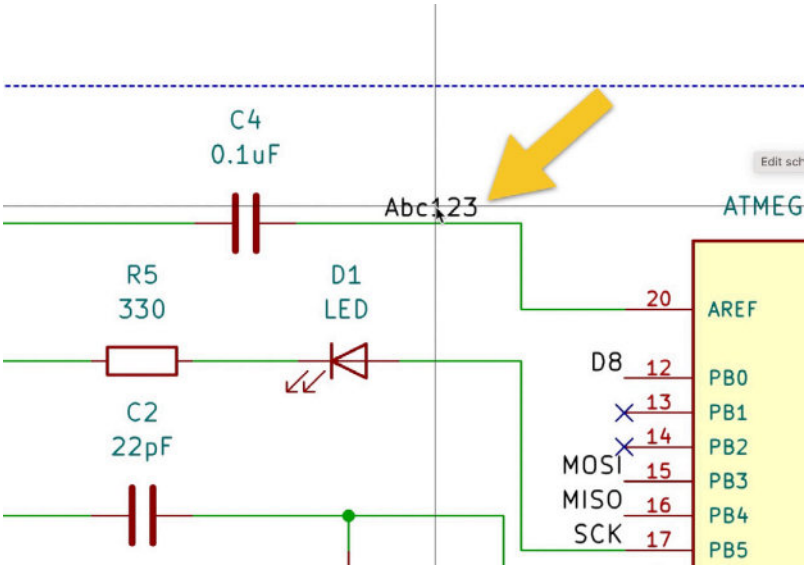


Figure 11.11.2: The new net labels appears in the schematic in the “4layer” branch.

Open the layout editor and update it with the changes from the schematic editor. Below you can see a detail of the layout editor that shows the new net label:

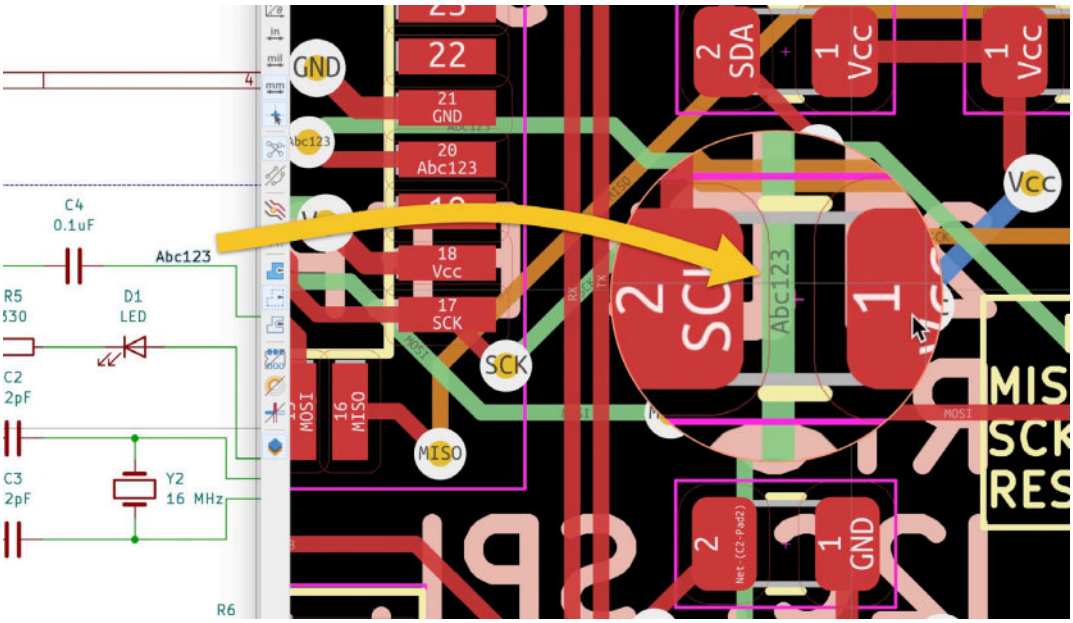


Figure 11.11.3: The new net label appears in the four-layer PCB.

You can repeat the process to update the two-layer PCB with the change in the schematic.

12. Finding and correcting a design defect

As part of the beta program of this book, reader Antti found a bug in the schematic of this project. This bug required corrections in both the schematic and the layout. Instead of re-writing the chapters in this part of the book, I introduced this chapter. I preferred not to obscure the reality that errors are a natural part of the engineering that leads to fixes and improvements.

In this chapter, I will document the bug that Antti found and how I iterated through the design of the project's PCB to fix it.

The primary defect

In step four of the schematic workflow, I introduced the bug when I incorrectly wired the SCL pin (from the I2C interface) together with the SCK pin (from the SPI interface). Both SCL and SCK convey clock signals, but, of course, those signals belong to different communication interfaces and should be kept separate.

In the figure below, I use arrows to highlight the location of the bugs.

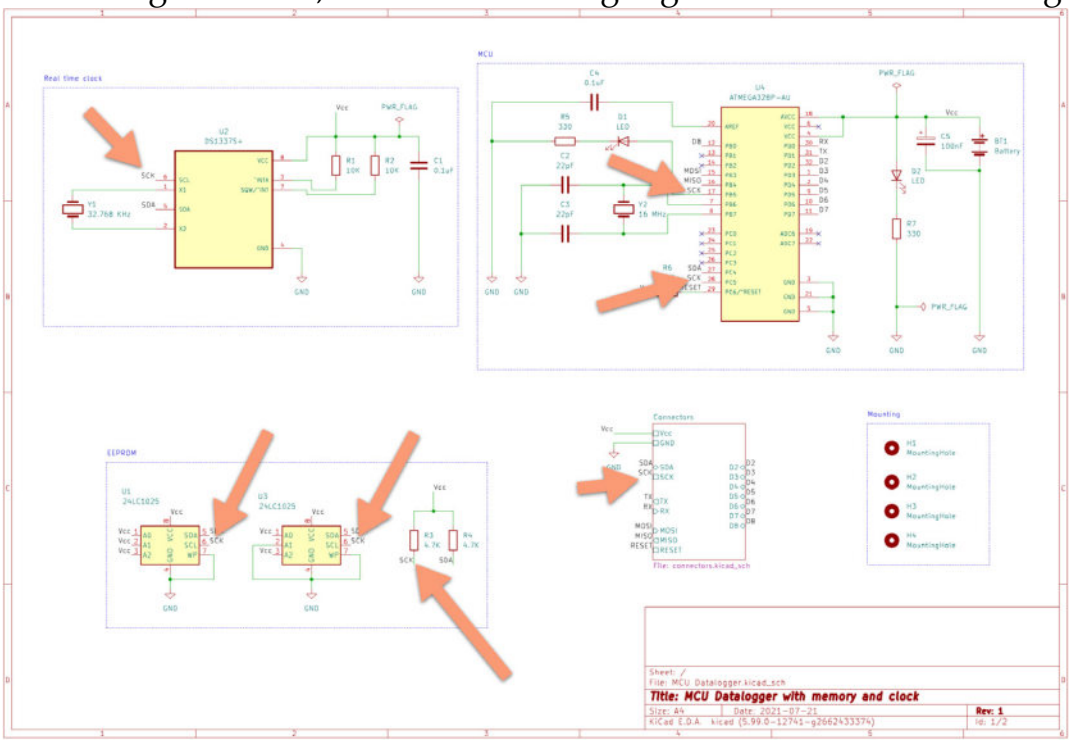


Figure 11.12.1: The location of the defects.

The secondary defect

I worked on the fix for the SCL-SCK bug on KiCad 6.0.0 rc1. I designed the PCB of this project in an earlier version of KiCad. As you can see in the figure above, I have used two PWR_FLAG symbols. I placed one PWR_FLAG in the Vcc net in the MCU group and another in the Vcc net of the "Real time clock" group. While the ERC in the earlier version of KiCad did not complain about the two PWR_FLAG symbols, KiCad 6.0.0 rc1 did. I addressed this issue in the schematic fix.

The fixes

I created a new "SCL" net label to fix the primary defect and attached it to all SCL pins, including pin 28 of U4. I also added a new hierarchical pin label to the connector in the Connectors sheet.

I deleted one of the two PWR_FLAG symbols to fix the secondary defect. While addressing the defects, I keep the complete history of the changes in the project's Git repository. This was an excellent opportunity to see how Git and KiCad can work together in a real-life situation.

12.1. Fix the schematic

I will begin fixing the two defects in the 2layer branch. I plan to:

1. While in the 2layer branch, update the schematic (set the correct net labels and delete the redundant power flag).
2. Merge the updated schematic file into the main branch.
3. Merge the updated schematic into the 4layer branch.
4. Return to the 2layer branch to update the 2-layer board layout.
5. Return to the 4layer branch to update the 4-layer board layout.

An essential element of the process that I outline above is merging changes in the main branch into the 4layer branch while ignoring changes to the PCB layout file. I have described how to set up this capability in the earlier chapter, "11. Updating layout from changes to the schematic with Git". My instructions below will not work without first setting up the ".gitattributes" file in the main branch.

Fix the schematic in the 2layer branch

Open your terminal, switch into the project's KiCad directory, and ensure that you are working in the 2layer branch:

```
% git branch
  2l-experimental
```

```
* 2layer
  4l-experimental
  4layer
  main
 %
```

As you can see in my terminal session above, I am working in the 2layer branch. Continue in the schematic editor. Fix the secondary defect (the redundant power flag) by deleting it. I use the interactive delete tool for this:

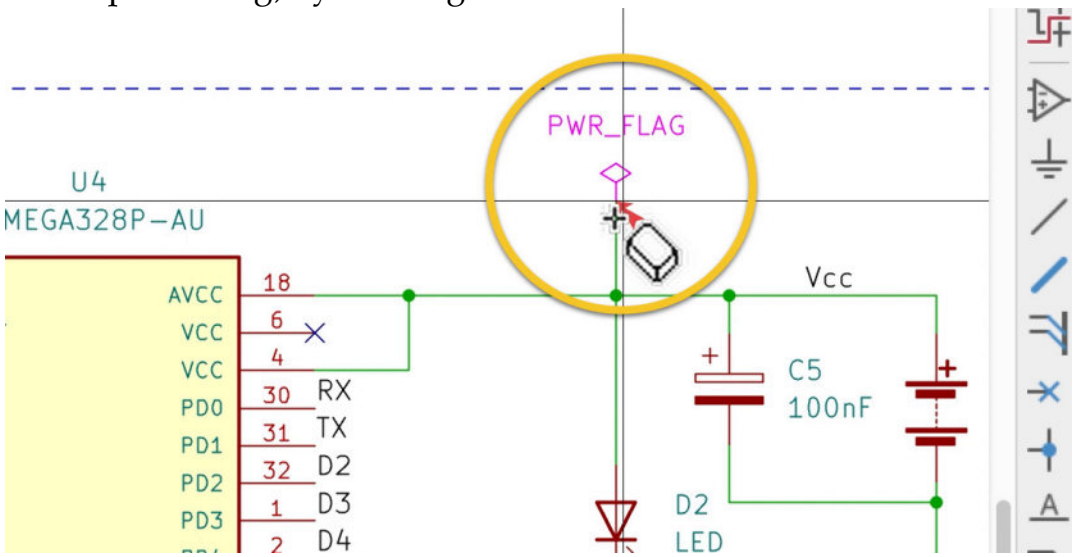


Figure 11.12.1.2: Delete the redundant power flag.

Do an ERC and confirm that the check is no longer complaining about the power flag issue.

Continue to fix the problem with the net labels. In the Atmega328P module, pin PB5 (#17) conveys the SCK signal, the clock for the SPI interface. I have attached the correct net label "SCK" in the schematic. However, pin PC5 (#28) conveys the SCL signal, the clock for the I2C interface. This is the core of the bug (I have incorrectly attached the SCK net label).

To fix this, delete all instances of the SCK net label, except for the one attached to U4 pin 17. Look at Figure 11.12.1 for a guide on the exact location of the SCK labels to delete.

To replace the deleted net label, create a new net label with the name "SCL." Attach instances of this label in all SCL pins (footprints U2, U4, U1, U3). You will also need to create a new hierarchical label named "SCL" for the Connectors sheet and expose it to the root sheet.

You can see the two sheets of this project with the net and hierarchical labels corrected in the two figures below.

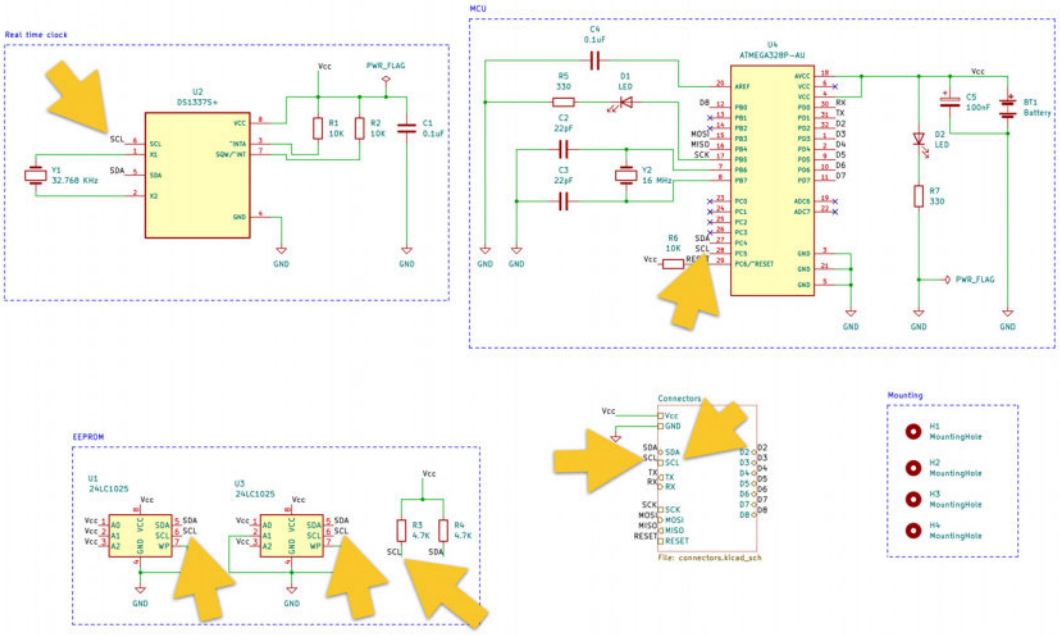


Figure 11.12.1.3: The root sheet with corrected net labels.

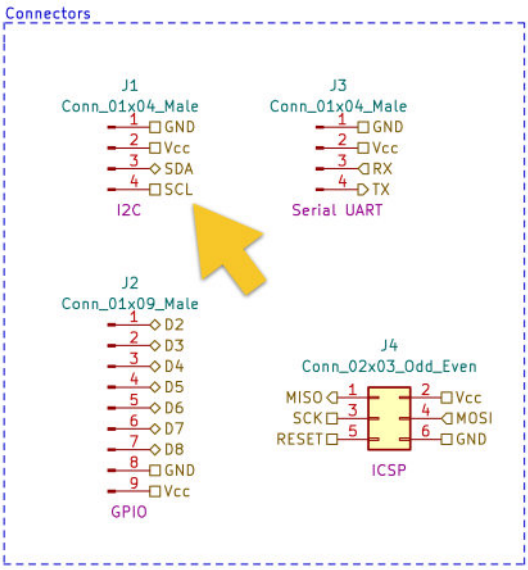


Figure 11.12.1.4: The Connectors sheet with corrected net labels.

Do a visual check to ensure you haven't forgotten anything, and then an ERC. There should be no errors. My ERC showed a couple of symbol violations ("modified in library") which I can safely ignore.

Save the schematic and return to your terminal window. Check the Git status, add the changes to staging, and commit them. Here is my session:

```
 % git status
On branch 2layer
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: MCU Datalogger.kicad_pcb
        modified: MCU Datalogger.kicad_prl
        modified: MCU Datalogger.kicad_pro
        modified: MCU Datalogger.kicad_sch
        modified: connectors.kicad_sch

no changes added to commit (use "git add" and/or "git commit -a")
% git add .
% git commit -am
    "Fixed two defects in the schematic."
[2layer c84e21c] Fixed two defects in the schematic.
    5 files changed, 5141 insertions(+), 4860 deletions(-)
% git status
On branch 2layer
nothing to commit, working tree clean
%
```

At this point, you have fixed the bugs in the schematic and have committed the changes to the 2layer branch in Git. Let's merge these changes in the main branch.

Merge changes to 'main'

To switch your working branch to "main", use the "checkout" command:

```
% git checkout main
```

Next, merge the changes committed in the 2layer branch into main. For this, use the "merge" command:

```
% git merge 2layer
Auto-merging MCU Datalogger.kicad_sch
CONFLICT (content): Merge conflict in MCU Datalogger.kicad_sch
Auto-merging .gitignore
CONFLICT (content): Merge conflict in .gitignore
Automatic merge failed; fix conflicts and then commit the result.
%
```

Unfortunately, Git was unable to merge the changes automatically. There are changes in two files that have conflicts that I had to resolve in my instance manually. Git has inserted markers in those files to help me find the conflicts. Let's look at the first conflicted file, "MCU Datalogger.kicad_sch".
In the main KiCad window, right-click on the schematic editor icon and select "Edit in text a editor".

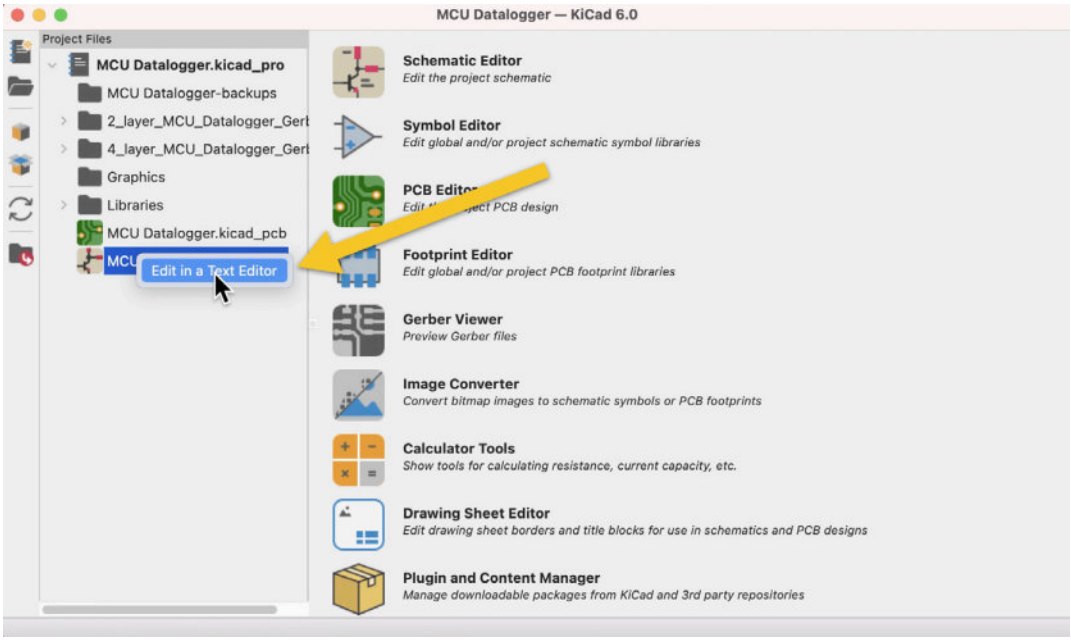


Figure 11.12.1.5: Open file in the text editor.

If you don't see this option, you will need to set up a text editor in the KiCad Preferences window. You will find the "Common" tab (see below).

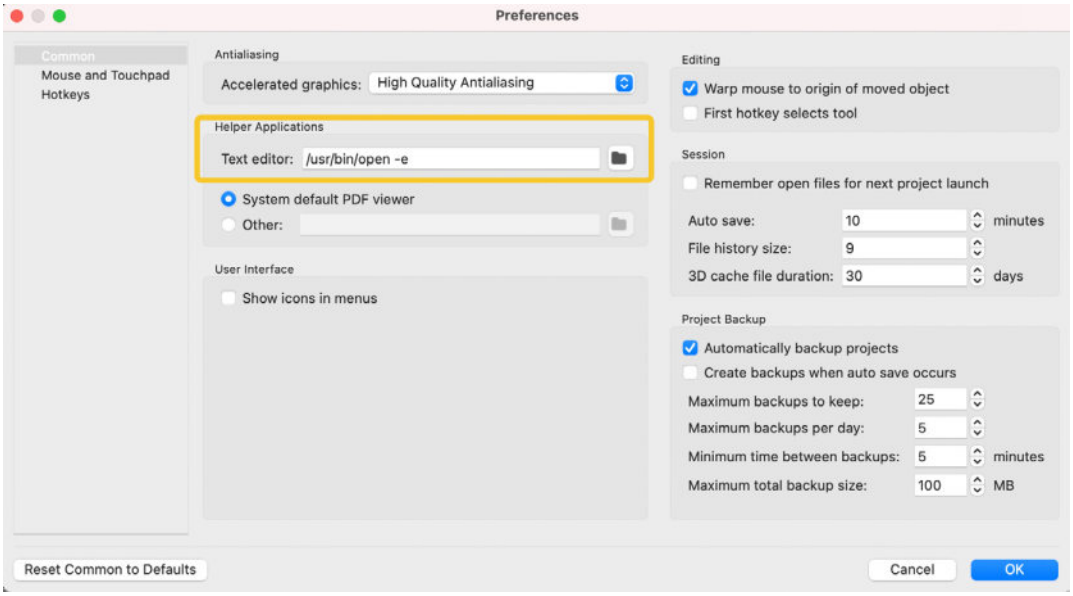


Figure 11.12.1.6: Setup your preferred text editor.

Once the schematic file is loaded in the text editor, scroll down until you find the marker "<<<<<< HEAD". Git marks the conflicted segments of the file with these markers. On top is the text from the main branch, and under the "======" marker, you see the version coming from the 2layer branch. In this case, I will delete the conflicting text from the main branch and keep the one from the 2layer branch. I depict the resolved conflict below:

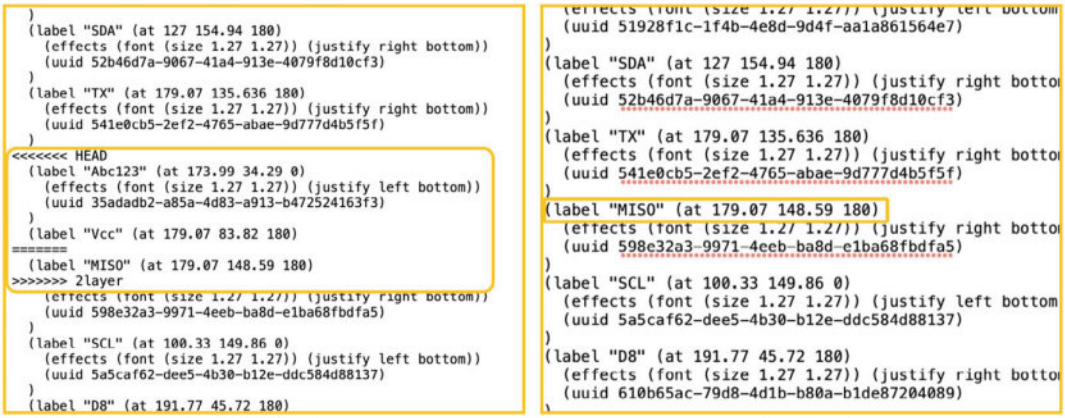


Figure 11.12.1.7: Resolving a Git merge conflict.

Save the text file. Repeat the process for the second conflicting file, ".gitignore". You will not be able to use the KiCad main window to open ".gitignore" in a text editor, so use your editor and manually find and open this file. I used Atom, and again chose the keep the 2layer branch version of the conflicted text:

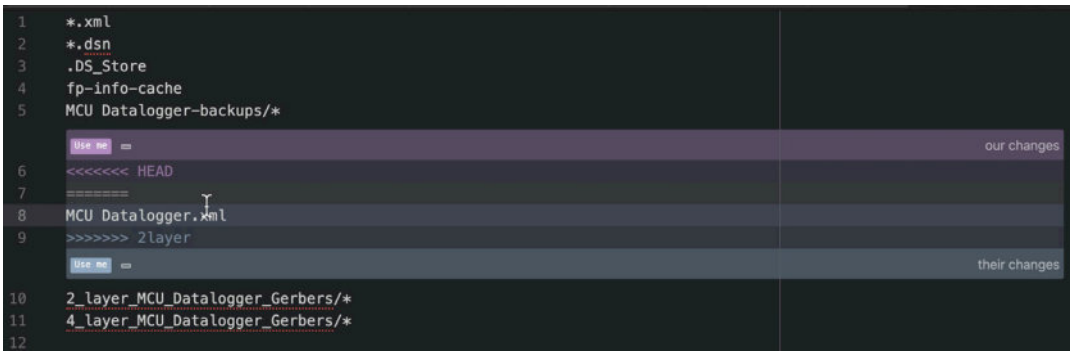


Figure 11.12.1.8: Resolving another Git merge conflict.

Save and close the text editor, and return to the command line terminal. Check the Git status, and commit the resolved conflicts:

```
% git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge ——abort" to abort the merge)

Changes to be committed:

    modified: MCU Datalogger.kicad_pcb
    modified: MCU Datalogger kicad_prl
    modified: MCU Datalogger.kicad_pro
    modified: connectors kicad_sch

Unmerged paths:
  (use "git add <file>..." to mark resolution)
```

```
    both modified: .gitignore
    both modified: MCU Datalogger.kicad_sch

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: MCU Datalogger.kicad_prl

% git add .
% git commit -am "Fixed bugs in schematic."
[main 4f1c810] Fixed bugs in schematic.
```

Let's recap before going any further. At this point:

1. The bugs are fixed in the schematic.
2. The updated schematic exists in the "main" and "2layer" branches.

What remains to be done is to:

1. Merge the schematic file changes into the "4layer" branch.
2. Fix the layouts for the 2-layer and 4-layer PCBs.

Let's do item "1" now and item "2" in the following two segments of this chapter.

Merge schematic changes into '4layer'

To merge the changes of the main branch into the 4layer branch (while ignoring the layout file thanks to the existence of the ".gitattributes" file), you will use the "merge" command while in the 4layer branch.

When you run the "merge" command, Git will probably ask you for a comment by opening up a text editor in your terminal. Type a description of the merge (such as "Bringing fixes in the schematic", and save the editor content to return to the terminal.

Here is my command line session:

```
% git checkout 4layer
Switched to branch '4layer'
% git merge main
Auto-merging MCU Datalogger.kicad_pro
Auto-merging MCU Datalogger.kicad_pr1
Auto-merging MCU Datalogger.kicad_pcb
Merge made by the 'recursive' strategy.
  .gitignore | 3 +
```

```
  MCU Datalogger.kicad_pcb | 7980 ++++++++++++++++++++++++++++++++++
+++++++++------------------
  MCU Datalogger.kicad_pr1 | 9 +-
  MCU Datalogger.kicad_pro | 6 +-
  MCU Datalogger.kicad_sch | 1668 +++++++++----------
  README.md | 5 +
  connectors.kicad_sch | 337 ++--
7 files changed, 5147 insertions(+), 4861 deletions(-)
create mode 100644 README.md
%
```

Git was able to merge automatically, and the 4layer branch contains the latest schematic version.

In preparation for the next step, updating the 2-layer version of the PCB with the fix, check out the 2layer branch:

```
% git checkout 2layer
```

Let's continue with the 2-layer PCB layout.

12.2. Fix the 2 layer PCB layout

You are working in the 2layer Git branch and are about to update the layout file to contain the fixes that you initiated in the previous step.

Open the layout editor and import the changes from the schematic. The layout will show a couple of ratnest lines resulting from the net label changes in the schematic. I have turned off the silkscreen layers to make it easier to work with the cluttered layout. You can use the widgets in the Appearance pane to make the editor work more comfortable.

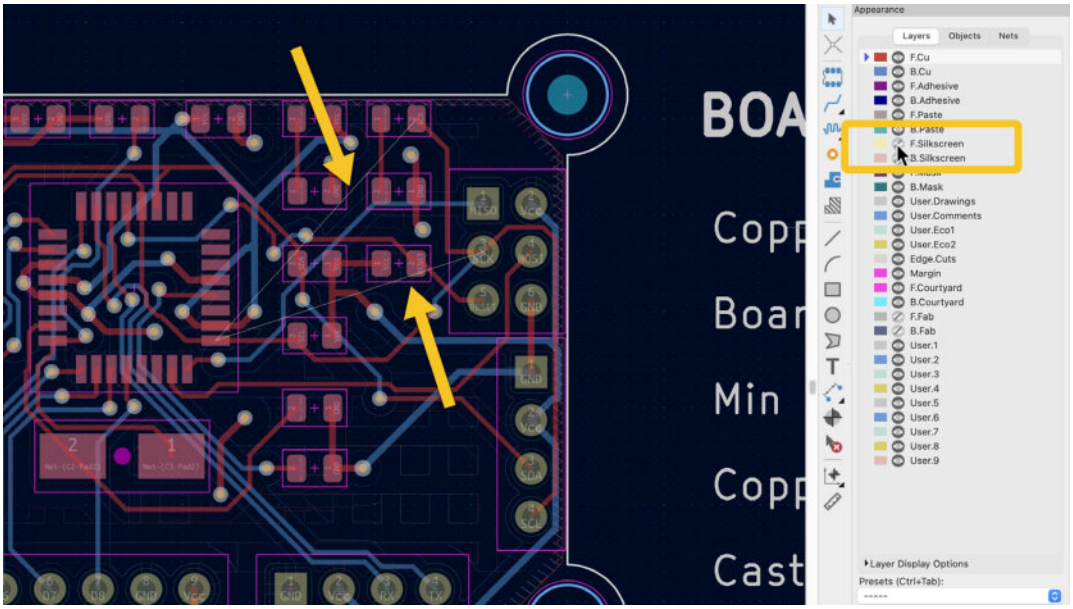


Figure 11.12.2.9: The two ratsnest lines are the result of the changes in the schematic.

When I rework a populated layout that is mostly routed, I use the interactive delete tool to delete tracks that start or finish on pads that I will need to re-wire. For example, pad 2 of D1 (the footprint at the top right corner of the PCB) will need re-wiring. I have deleted several affected copper tracks in the figure below to make re-drawing easier.

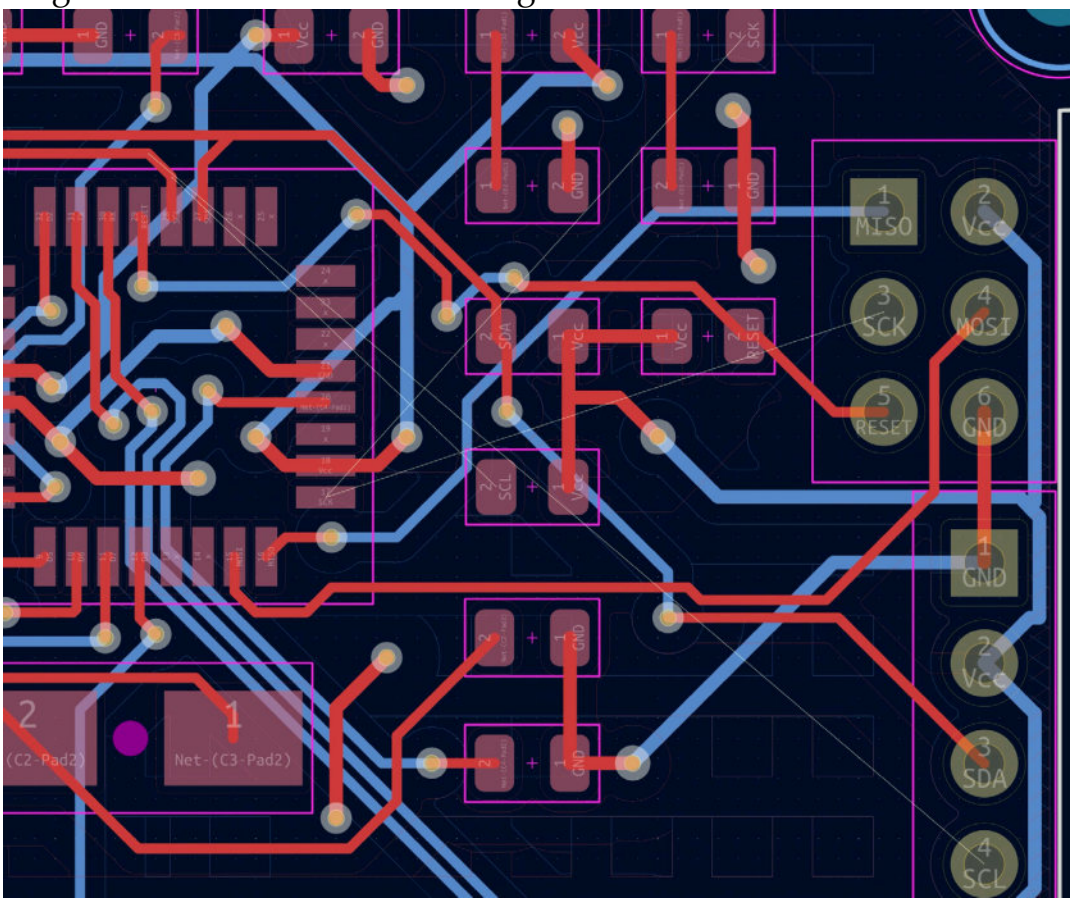


Figure 11.12.2.10: I have deleted several copper tracks.

Draw the new copper traces. Here's the fully-routed instance of my PCB:

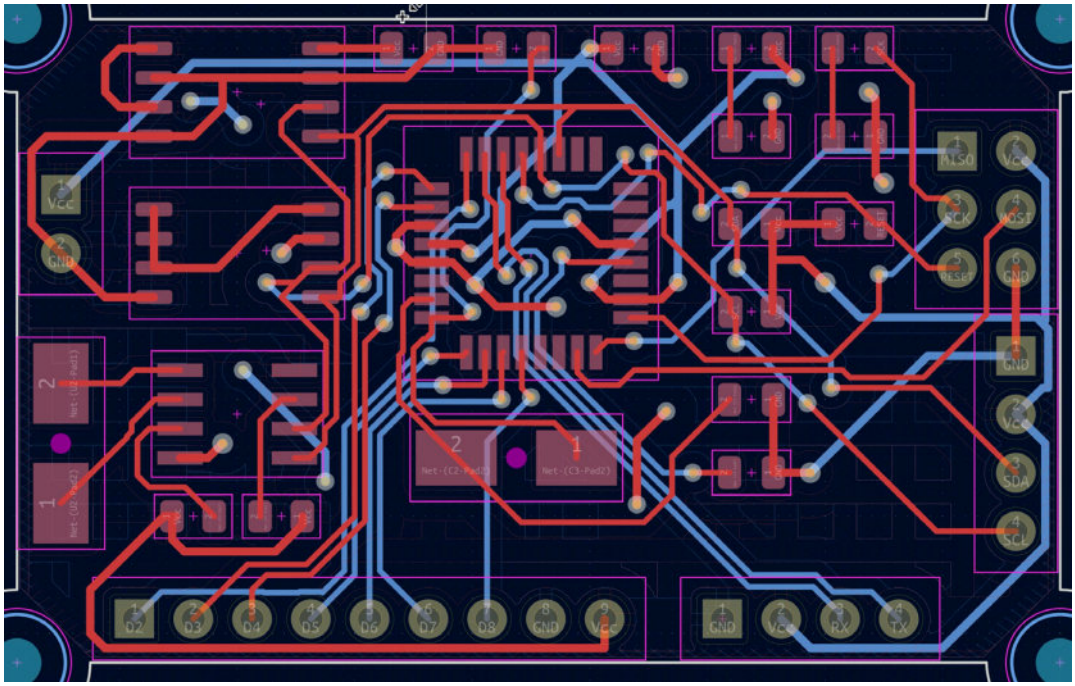


Figure 11.12.2.11: Fully routed, with defects fixed.

The 2-layer layout is now fully routed. Run a DRC to confirm that there are no errors. In my case, the DRC showed two "unconnected end" violations. I chose to address those violations by deleting small segments of copper track that I overlooked in earlier work sessions with this board. You can see one of those below:

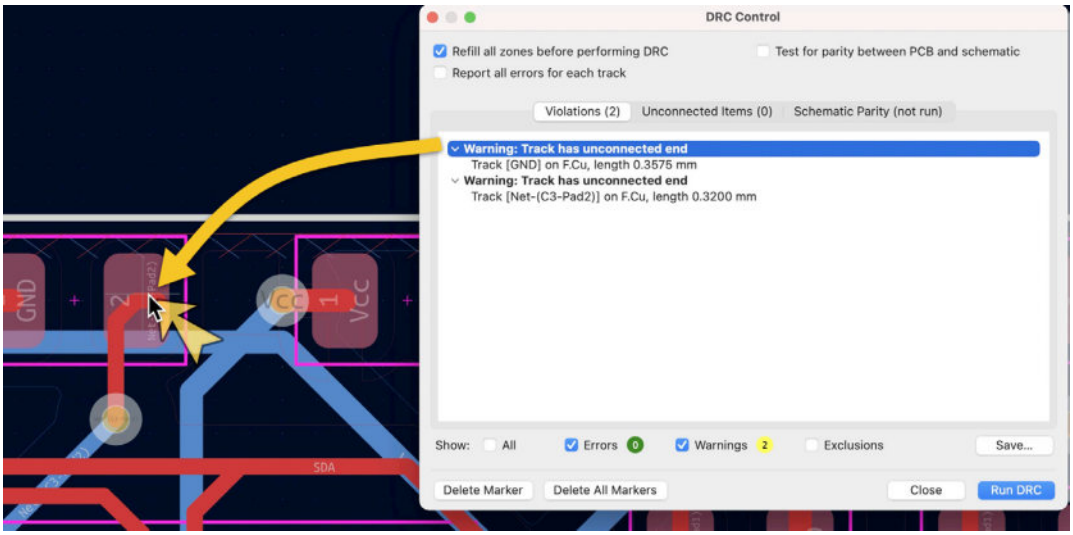


Figure 11.12.2.12: Fixing stray copper trace segments.

After addressing both violations, I repeated the DRC, which shows zero errors and warnings.

Save the layout editor file and return to the terminal when you are ready. Check the Git status, and commit the changes. Here is my terminal session:

```
% git status
```

```
On branch 2layer
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: MCU Datalogger.kicad_pcb
    modified: MCU Datalogger.kicad_prl

no changes added to commit (use "git add" and/or "git commit -a")
% git add .
% git commit -am "Fixed 2 layer PCB."
[2layer 6cc3a84] Fixed 2 layer PCB.
  2 files changed, 2758 insertions(+), 2416 deletions(-)
```

The 2-layer PCB now contains the bug fixes. Let's repeat the process for the 4-layer PCB.

12.3. Fix the 4 layer PCB layout

Let's finish the update process by applying the fix to the 4-layer version of the PCB.

Close the layout editor if you left it open after finishing work in the 2-layer PCB, then go to the terminal and switch your working Git branch to "4layer":

```
% git branch
  2l-experimental
* 2layer
  4l-experimental
  4layer
  main
% git checkout 4layer
Switched to branch '4layer'
%
```

In KiCad, open the schematic editor and double-check that you are working with the latest schematic version. You should have already imported the latest changes, but I find that small "sanity checks" like this can save you a lot of frustration and time if things don't go exactly to plan.

Open the layout editor, and import the schematic changes. As with the 2-layer PCB, you will see a couple of ratsnest lines that give you a clue that the import was successful. I use the same pattern as before: learn redundant or incorrect copper tracks, and replace them with new ones. Make your layout editor comfortable to work with by turning off unnecessary layers. For the sake of brevity, I present my finished and completely routed instance of the 4-layer PCB:

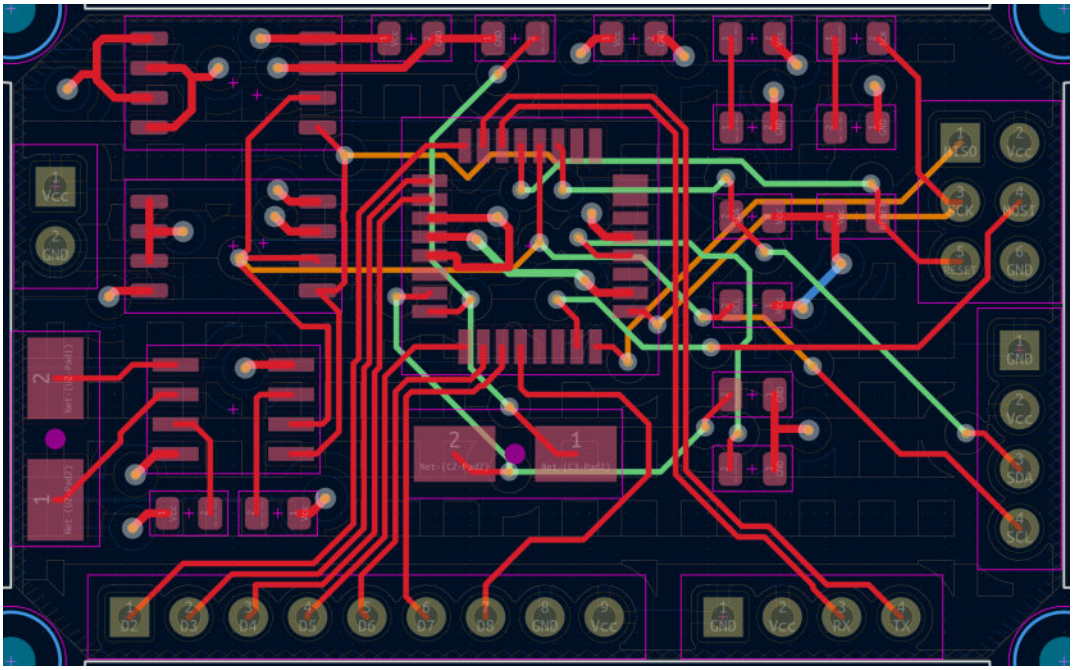


Figure 11.12.3.13: The fully routed 4-layer PCB that contains the fixes.

As always, do a DRC before you commit the changes. My DRC revealed similar warnings to the 2-layer PCB (stray copper track segments), which I fixed. My last DRC returned no errors or warnings.

Let's finish up by committing the changes to Git repository's 4layer branch:

```
% git status
On branch 4layer
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: MCU Datalogger.kicad_pcb

no changes added to commit (use "git add" and/or "git commit -a")
% git add .
% git commit -am "Fixed net bugs in 4 layer version of the PCB."
[4layer 9138eb5] Fixed net bugs in 4 layer version of the PCB.
  1 file changed, 7007 insertions(+), 7101 deletions(-)
% git status
On branch 4layer
nothing to commit, working tree clean
```

This concludes the work needed to fix the bug that Antti found. Both version of the PCB are now up-to-date, with all change history committed to the project Git repository.

1. Project - Introduction

Welcome to Part 12 of this book. In the chapters that follow, you will design an ESP32 development board. To assist you with the design process, you will use the reference schematic design from Espressif, the original designer and manufacturer of the original ESP32 dev kit module.

By working on this project, you will learn how to use and modify reference designs. The ESP32 is a widely used and understood board. It is compact, yet packs significant computing power and capabilities. It's circuit board is more complex compared to those in the previous projects of this book, but not to complex to make this project long and tedious.

Below you can see the layout design as it will look once the project is completed.

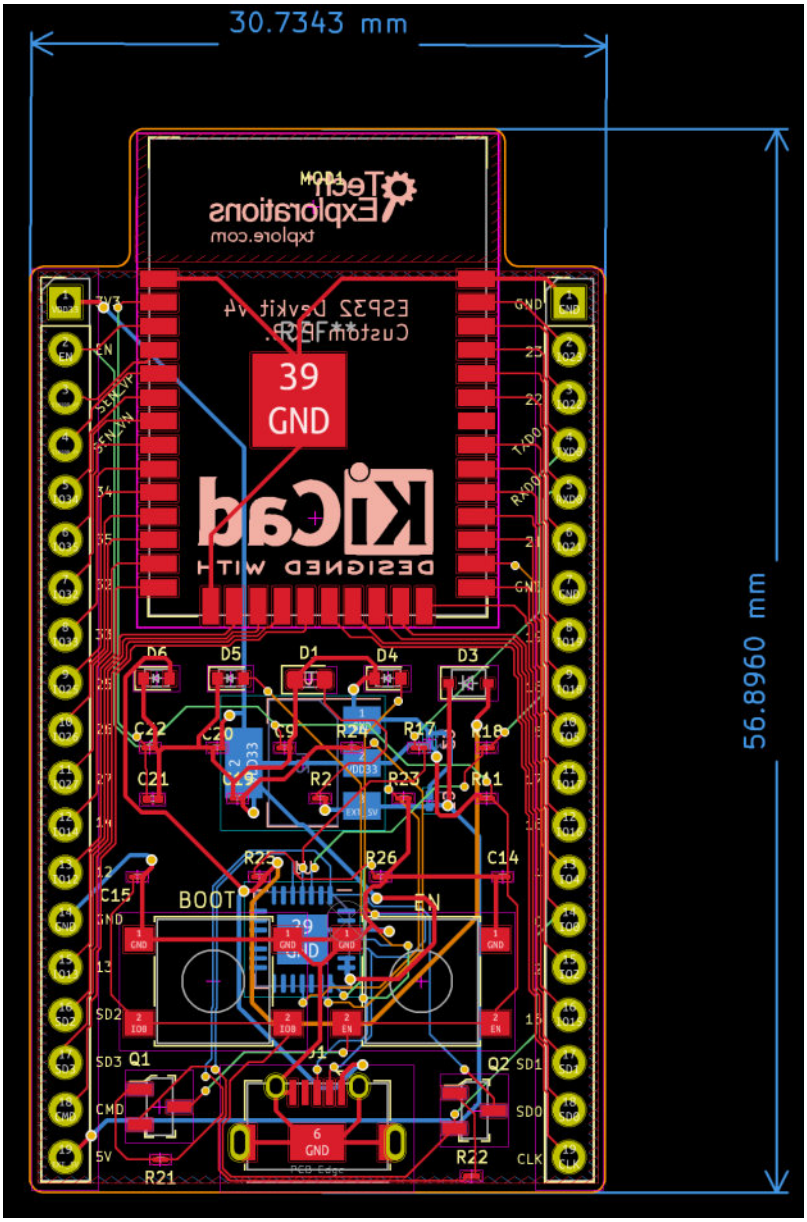


Figure 13.1.1: The ESP32 development kit clone project deliverable.

In this instance of the development kit, I have retained the form factor and shape of the original ESP32 development kit. This is a four-layer board, with the ESP32 module at the top with its integrated antenna extending outside of the board, the power and serial communications port at the bottom, and the pin headers along the sides. The board also features the two control buttons, “BOOT” and “EN”, and uses SMD components on the front and back. Here’s a view of the reference layout:

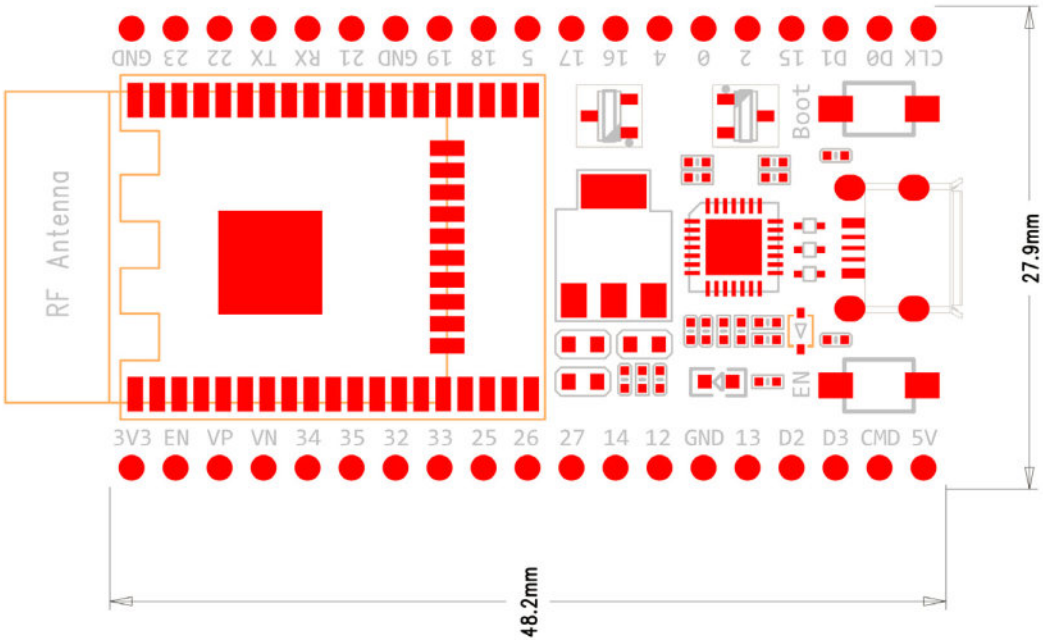


Figure 13.1.2: A view of the ESP32 development kit reference layout.

Once you understand the design elements of this board, you will be able to modify its shape and component placement to match the requirements of your project. For example, in my instance of the board, I have chosen to place the voltage regulator in the back of the PCB, even though in the reference design the regulator in is the front. I did this because I wanted to allow more room in the front for small components, like the resistors, to make assembly by hand easier.

Below you can see the final schematic.

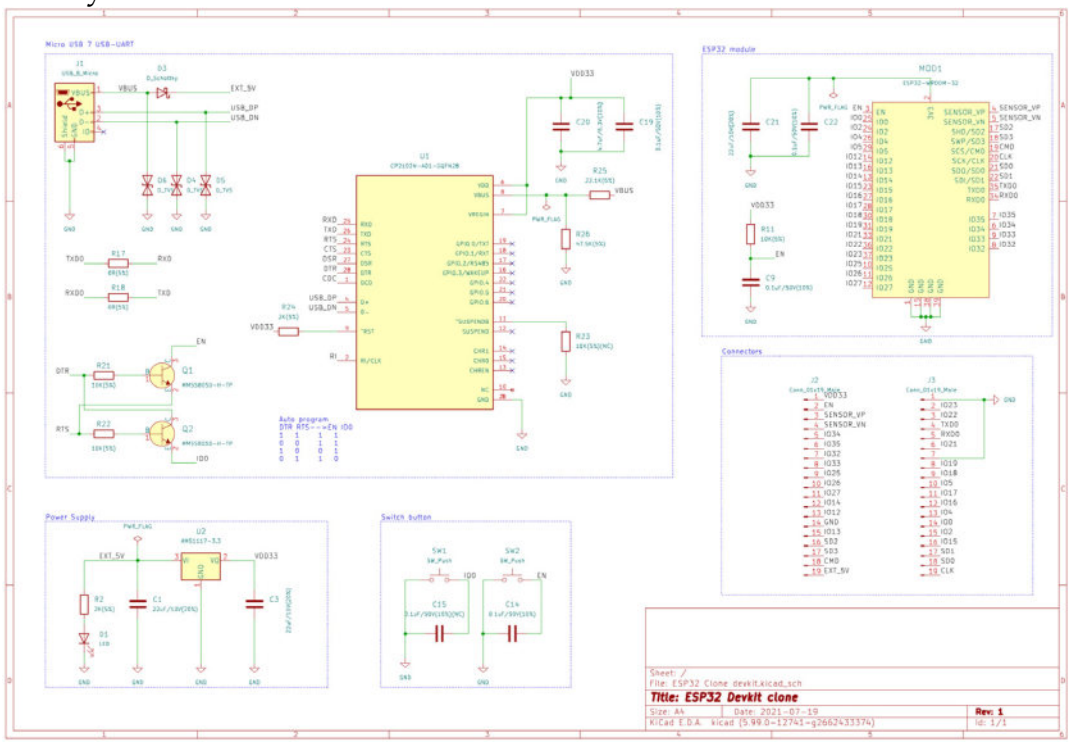


Figure 13.1.3: The ESP32 development kit clone project schematic.

Thanks to the ESP32 module's highly integrated design, the board that hosts the module (i.e. the development kit) is relatively simply. In the instance of the development kit design that you see above, I have arranged the components in five functional groups, and I have drawn all details from the original Espressif reference schematic design. You can see the reference schematic design below:

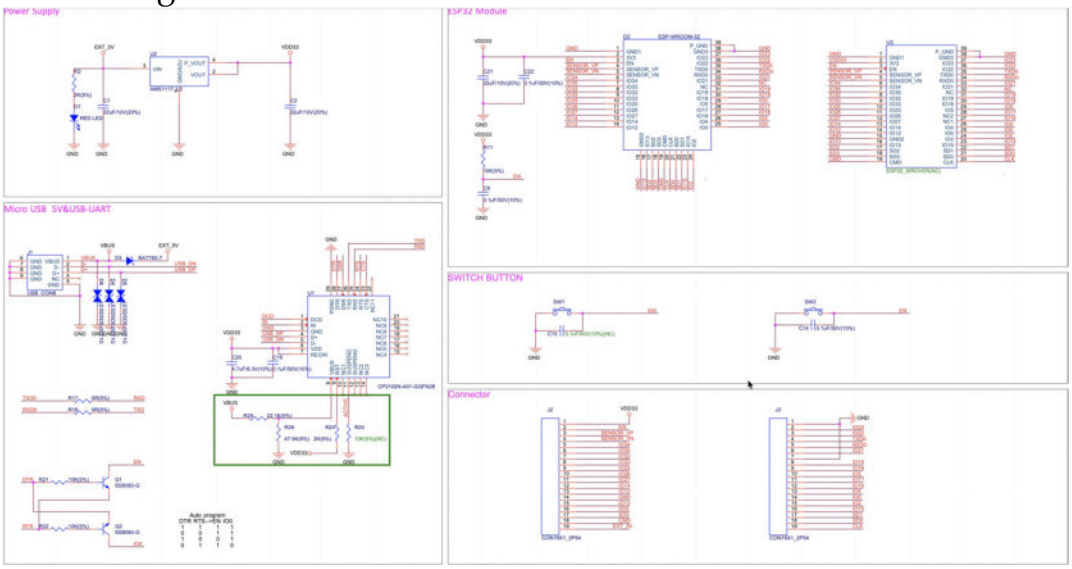


Figure 13.1.4: The ESP32 development kit reference schematic.

You will learn more about this in chapter 2.

Below you can see the 3D rendering of the development kit.

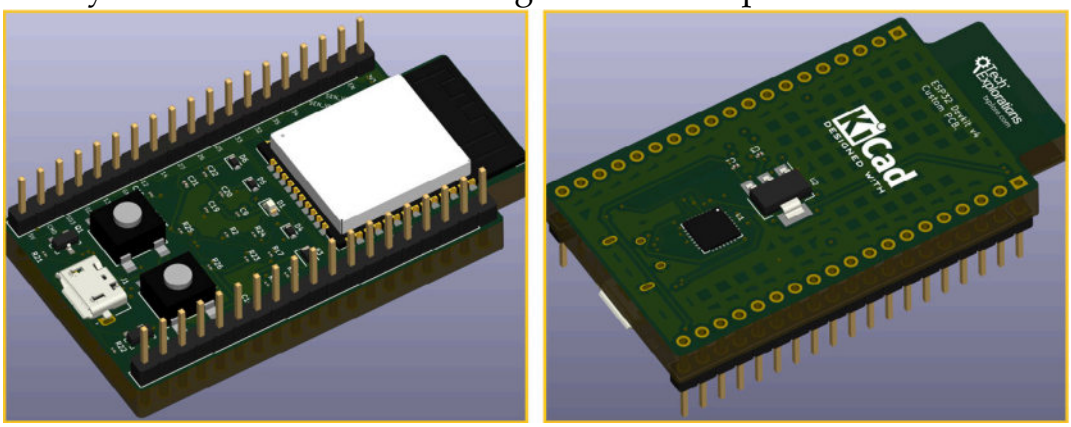


Figure 13.1.5: A 3D rendered view of the project deliverable.

The 3D rendering contains models for all components and is accurate in terms of what the final manufactured board would look like.

The process of customising an existing board begins by finding the original schematic and layout designs. In the case of the ESP32 development kit, you can find the official reference designs in the Espressif website (https://

www.espressif.com/en/support/documents/technical-documents). I have used the references design for the ESP32-DevKitC-v4 board. You can use the search filter to find this design (or try my search URL "https://www.espressif.com/en/support/documents/technical-documents?keys=ESP32-Devkit").

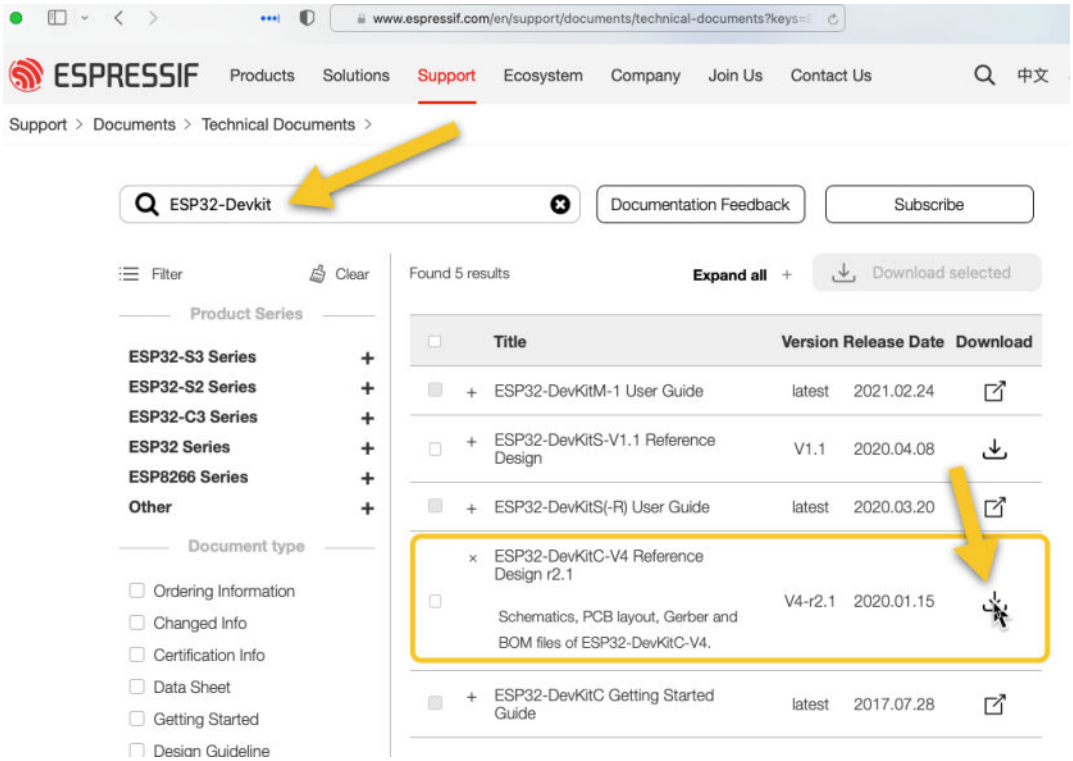


Figure 13.1.6: The source documentation and designs for this project.

Download the ZIP file that contains everything you will need for this project: the schematics, the PCB layout, and the BOM. The ZIP also contains a set of Gerber files which you do not need (you will generate a new set at the end of the project).

The reference BOM is very important because it gives you all the components and their values that you will need for your custom board. These components and values are tested by Espressif and used in millions of manufactured board, so you can be confident that they will work. I used the reference BOM to help me select symbols and footprints for my instance of the board. The project BOM that you see below contains information that I sourced from the reference BOM.

Reference	Value	Footprint
C1, C2, C21	22uF/10V(20%)	Capacitor_SMD:C_0201_0603Metric

C9, C14, C19, C22	0.1uF/50V(10%)	Capacitor_SMD:C_01005_0402Metric
C15	0.1uF/50V(10%) (NC)	Capacitor_SMD:C_01005_0402Metric
C20	4.7uF/6.3V(10%)	Capacitor_SMD:C_01005_0402Metric
D1	LED	LED_SMD:LED_0603_1608Metric
D3	D_Schottky	Diode_SMD:D_SOD-323
D4, D5, D6	D_TVS	Diode_SMD:D_SOD-523
J1	USB_B_Micro	Connector_USB:USB_Micro-AB_Molex_47590-0001
J2, J3	Conn_01x19_Male	Connector_PinHeader_2.54mm:PinHeader_1x19_P2.54mm_Vertical
MOD1	ESP32-WROOM-32	digikey-footprints:ESP32-WROOM-32D
Q1, Q2	MMSS8050-H-TP	digikey-footprints:SOT-23-3
R2, R24	2K(5%)	Resistor_SMD:R_01005_0402Metric
R7, R18	0R(5%)	Resistor_SMD:R_01005_0402Metric
R11, R21, R22	10K(5%)	Resistor_SMD:R_01005_0402Metric
R23	10K(5%)(NC)	Resistor_SMD:R_01005_0402Metric
R25	22.1K(5%)	Resistor_SMD:R_01005_0402Metric
R26	47.5K(5%)	Resistor_SMD:R_01005_0402Metric
SW1, SW2	SW_Push	Button_Switch_SMD:SW_SPST_B3S-1000
U1	CP2102N-A01-GQFN28	Package_DFN_QFN:TQFN-28-1EP_5x5mm_P0.5mm_EP2.7x2.7mm
U2	AMS1117-3.3	Package_TO_SOT_SMD:SOT-223-3_TabPin2

Table 13.1.1: The Bill of Materials for this project.

The BOM shows several SMD components that are very small, such as the 0402 resistors. If you plan to assemble your custom board by hand, you should consider the difficulty of working with such small components. But, since this is your custom board, you can choose to replace those small components with alternatives that you feel more comfortable to work with.

This is just an example of how you can modify a reference design to fit your specific requirements.

The main objectives of this project are:

1. To help you practice skills you acquired in previous projects.
2. To gain experience in the design of a PCB based on an existing reference design.
3. To gain experience in creating dense, four-layer PCBs.

I used Snapeda to find the symbol-footprint pairs for MOD1 (the ESP32 module). You should be able to find all other symbols and footprints in KiCad's libraries.

Let's begin.

2. Schematic design

In this chapter, you will complete the schematic design of this PCB by following the schematic design workflow.

2.1. Schema 1 - New KiCad project and Schematic Setup

Let's begin with the new project. Create a new KiCad project, and store its files in a new directory. I have named my project "ESP32 Clone Devkit". You can see my new project setup below:

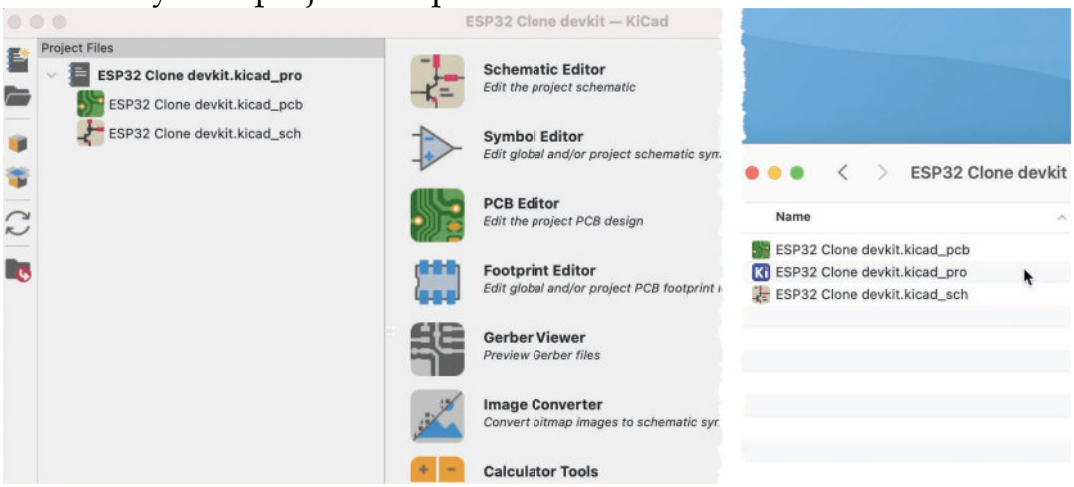


Figure 13.2.1.1: Begun a new project.

Continue to setup the schematic editor. I list my settings below (if a setting is not mentioned, you can assume that I have not changed its default setting):

- Page Settings:
  - Issue Date: copied today's date from the date field.
  - Revision: 1.
  - Title: ESP32 Devkit clone.

Next, review the reference schematic for symbols, footprints and 3D models that you will need to download from an online repository like Snapeda.

There are three components that KiCad does not have suitable candidates for and you will need to source elsewhere. In the figure below, I have circled the symbols that you will need to get from Snapeda or the Digikey libraries for KiCad.

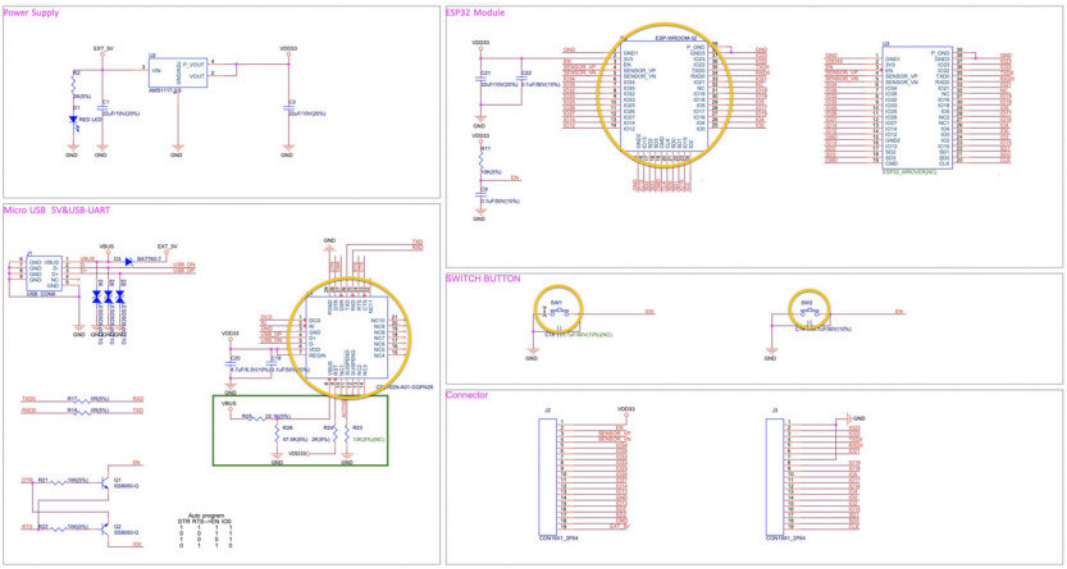


Figure 13.2.1.2: The circled symbols are not available in the KiCad symbol libraries

First, a component for the two push buttons. I looked at the libraries that come with KiCad for SMD push buttons, but I was not able to find the exact model I wish to use. After some research on Snapeda, I found a suitable model, the B3S-1000P that contains a symbol, footprint, and 3D model. Download the ZIP for this component and expand the archive in a new "libraries" folder in your project folder.

Second, a component for the CP2102N chip that implements the USB to UART bridge. While KiCad has a standard QFN28 footprint, it does not have a symbol that matches the one used in the schematic design reference. I could spent time to try and compensate for the discrepancies, but I decided that my time is better spent on finding an exact match for the symbol instead of attempting to rewire the schematic. The component I use in my design is CP2102N-A01-GQFN28. Download it and expand it in the project libraries folder. The archive contains all three files (symbol, footprint, 3D model).

Third, a component for the ESP32 module. I used a symbol and footprint available in the DigiKey library. You will find the symbol in the "dk_RF-Transceiver-Modules" group, by looking for "ESP32WROOM-32".

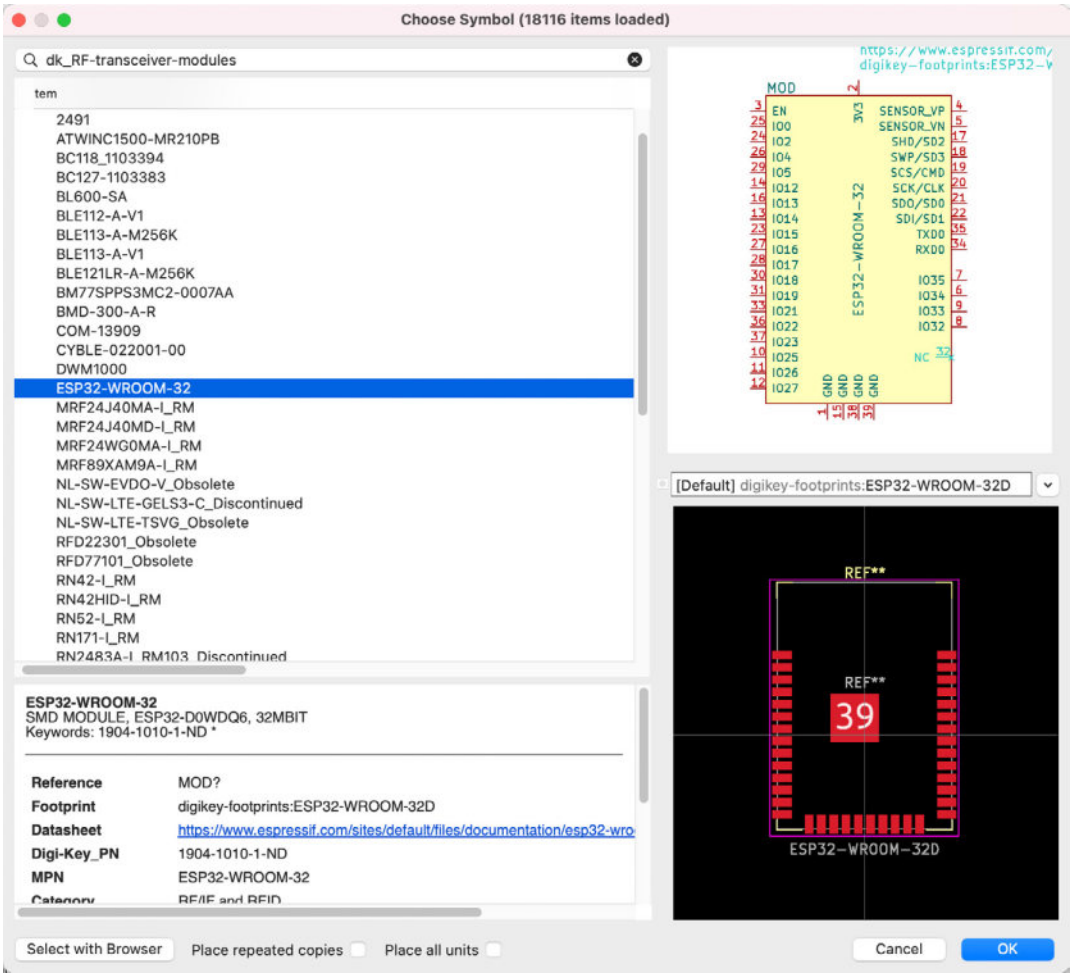


Figure 13.2.1.3: Using the ESP32 Wroom 32 module in the DigiKey library.

You can find the DigiKey library for KiCad on Github. You can learn how to install it in the relevant chapter earlier in this book. I have installed the two Snapeda symbols in my Project Specific Libraries tab, and the Digikey library in my Global Libraries tab (see below).

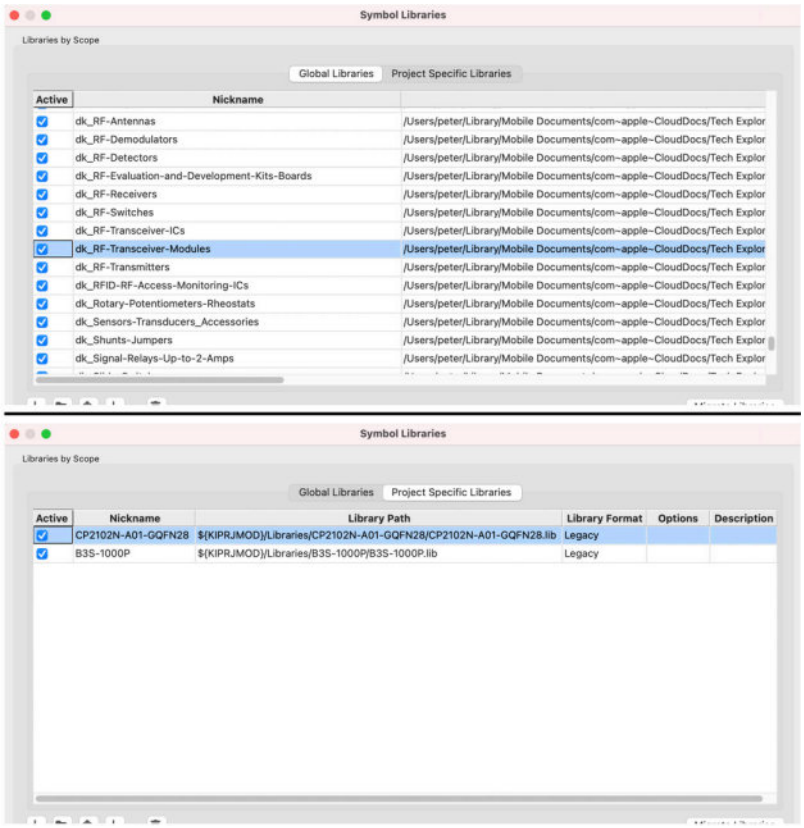


Figure 13.2.1.4: My symbol libraries setup.

Once you have all three of the symbols downloaded and installed continue with step two of the workflow.

2.2. Schema 2 - Symbols

In the previous segment you prepared your schematic editor, and installed the three third-party symbols. It is now time to add the schematic symbols to the design sheet. Use the reference design as an aid during the process of adding the symbols to the sheet. I have arranged the Eeschema and PDF viewer windows side by side, as in the example below:

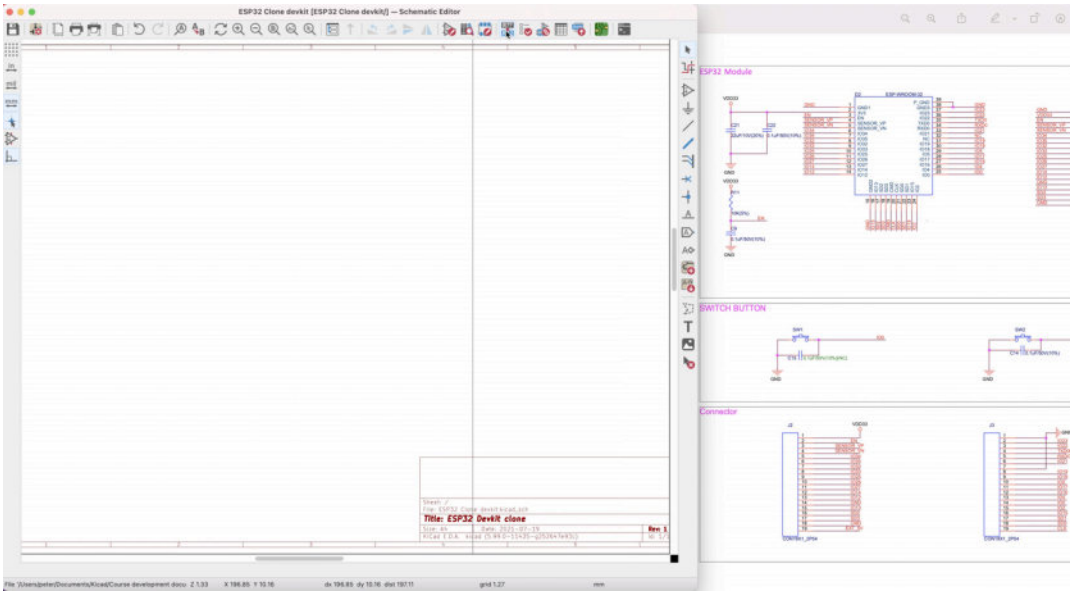


Figure 13.2.2.5: Side-by-side arrangement streamlines work.

For this schematic, I decided to add all large components first (starting with the ESP32 module), and then continue with the smaller ones. Below you can see my schematic after I have added the main components in the sheet:

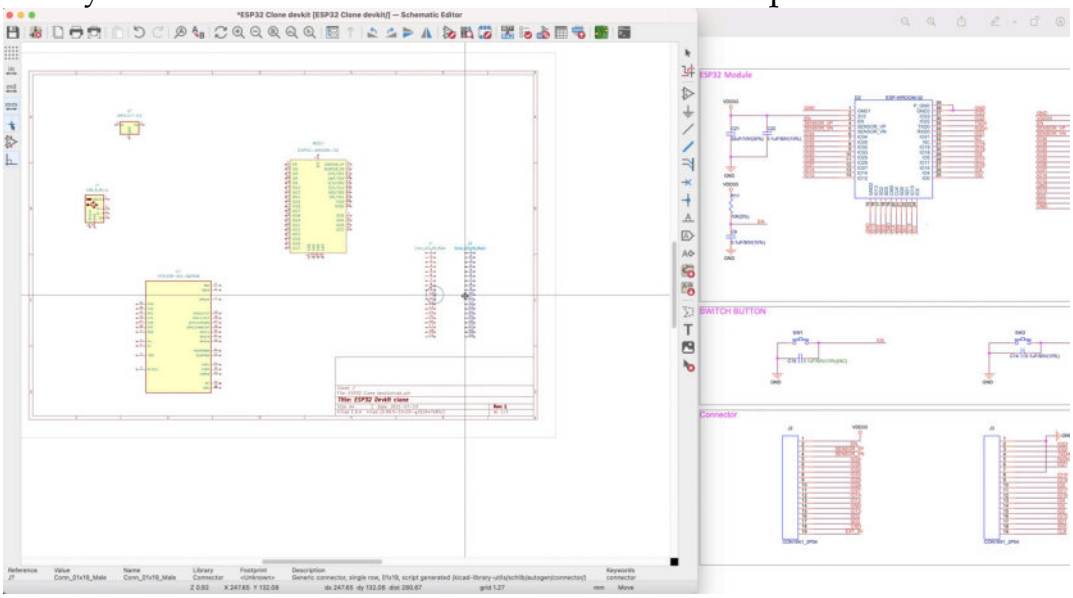


Figure 13.2.2.6: Added main components to the schematic sheet.

Continue with the smaller components, such as the buttons, USB connector, LEDs, diodes and resistors. Here's my schematic with all symbols added to the sheet:

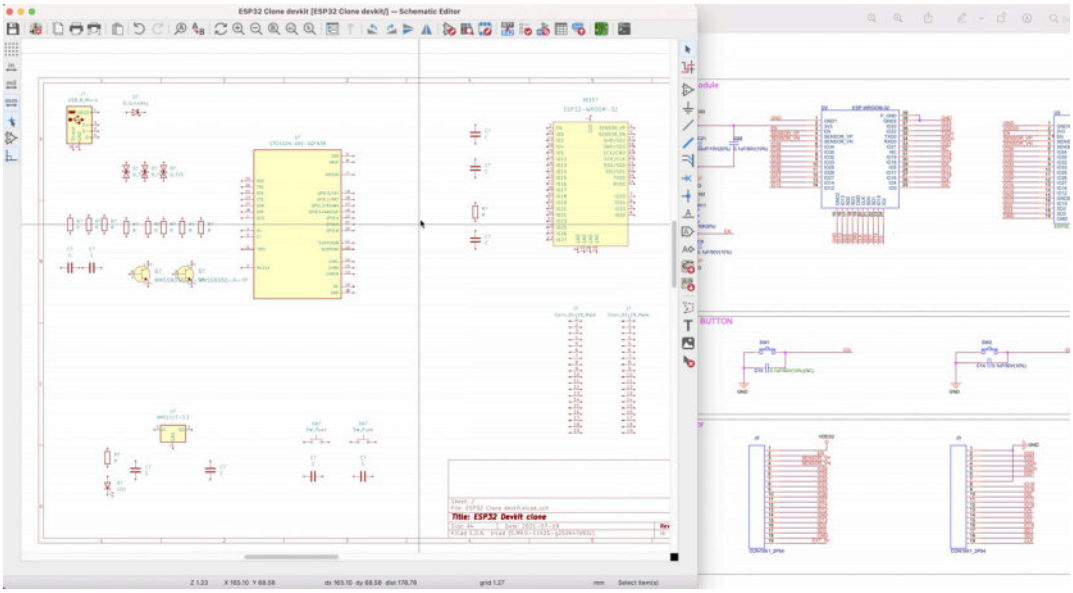


Figure 13.2.2.7: All symbols added to the sheet.

At this point, all schematic symbols should be in the sheet. In the next step of the workflow, you will annotate those symbols. Unlike the practice from previous projects, I recommend that you annotate this schematic manually so that your schematic follows the designators used in the reference design. If you use the automatic annotator, chances are that your schematic will contain designators that don't match those in the reference design. This will make it harder to work in the schematic editor and you will need to be much more cautious as progress through the copy process.

2.3. Schema 3 - Annotate and set component values

In this segment of the chapter, you will manually annotate symbols. While you are at it, you will also add the component values.

In earlier projects in this book, you used the automated annotator tool to quickly generate unique identifiers for each symbol. The annotator tool is quick, and because in prior projects you did not work off a reference design, you did not have to worry about discrepancies between the reference designators in your schematic versus those in the reference design.

In this project, however, you are using a reference schematic design as an aid for your design. Discrepancies in the symbol reference designators can cause you a lot of confusion later in the project. Therefore, it is better to ensure that reference designators match between your schematic and the reference schematic.

There are two ways by which you can set values for each symbol:

1. You can use the Properties window for each symbol individually.
2. You can use the Symbol Fields Table window and complete this task in bulk.

In the figure below, you can see the Symbol Fields Table window as it will look at the end of this segment. I have already populated all reference designators and values:

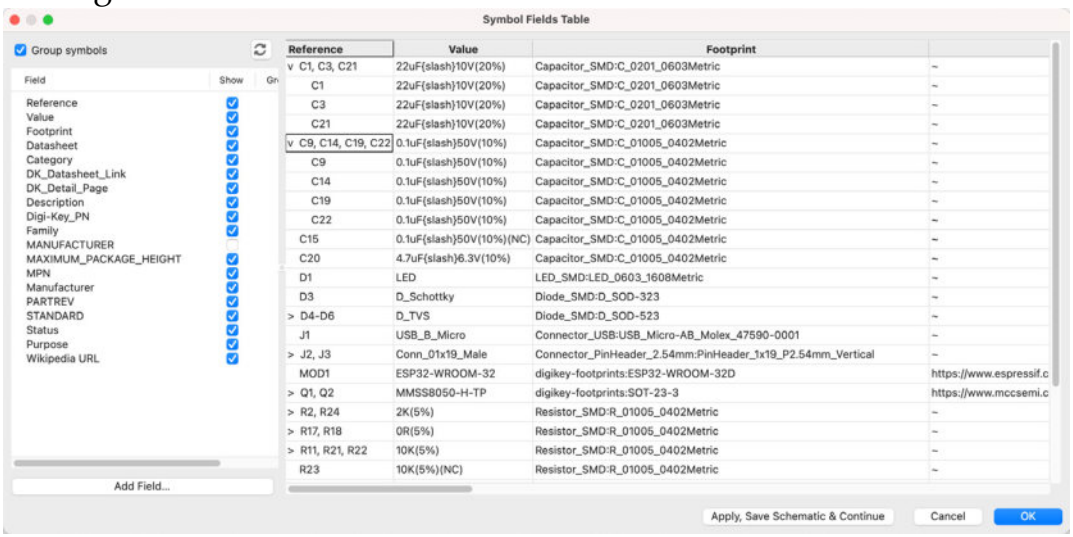


Figure 13.2.3.8: The Symbol Fields Table, populated.

However, at the start of this work, the same window looks like this:

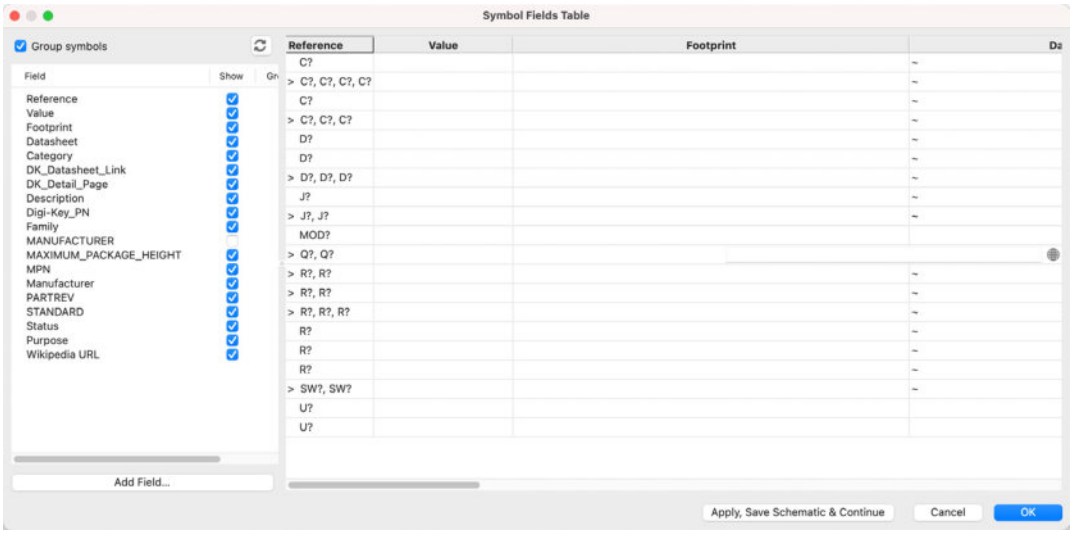


Figure 13.2.3.9: The Symbol Fields Table, not populated.

Unfortunately, can not use the Symbol Fields Table to manually edit the Reference fields. You can use it to edit all other fields. Therefore, the only option that you have for manually editing the reference designator for each symbol is to use the symbol's Properties window. Since you have to open this window manually for each symbol, you can take the opportunity to also set the symbol's value field.

Again, setup your desktop so that Eeschema and the PDF viewer window that contains the reference design are side-by-side. I chose to begin from power supply functional block of schematic and edit each symbol individually. For example, in the figure below, I edit the voltage regulator symbol with the designator “U2”, following the reference design:

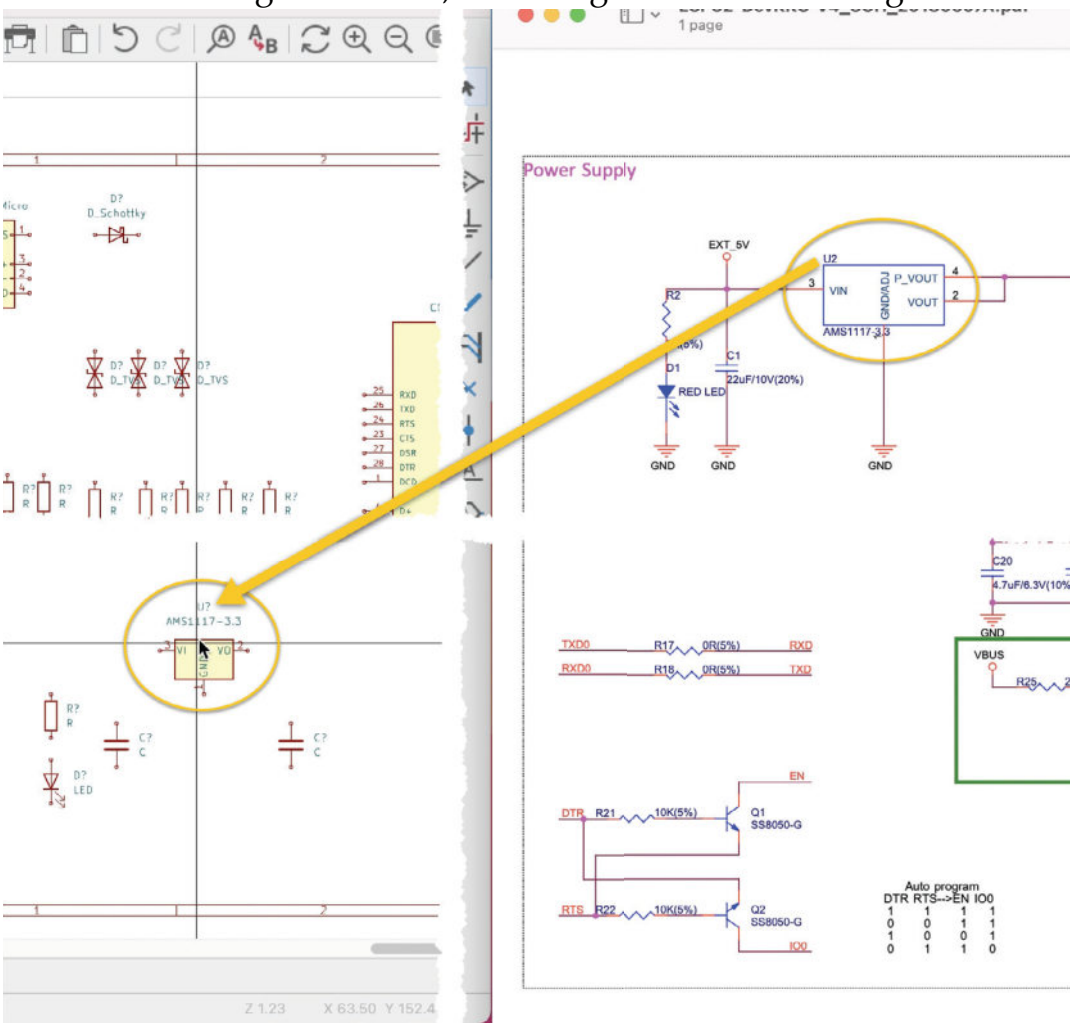


Figure 13.2.3.10: Setting the designator of the voltage regulator to “U2”.

Below you can see the Properties window for the voltage regulator. I have just edited the contents of the Reference and Value fields using information from the reference design.

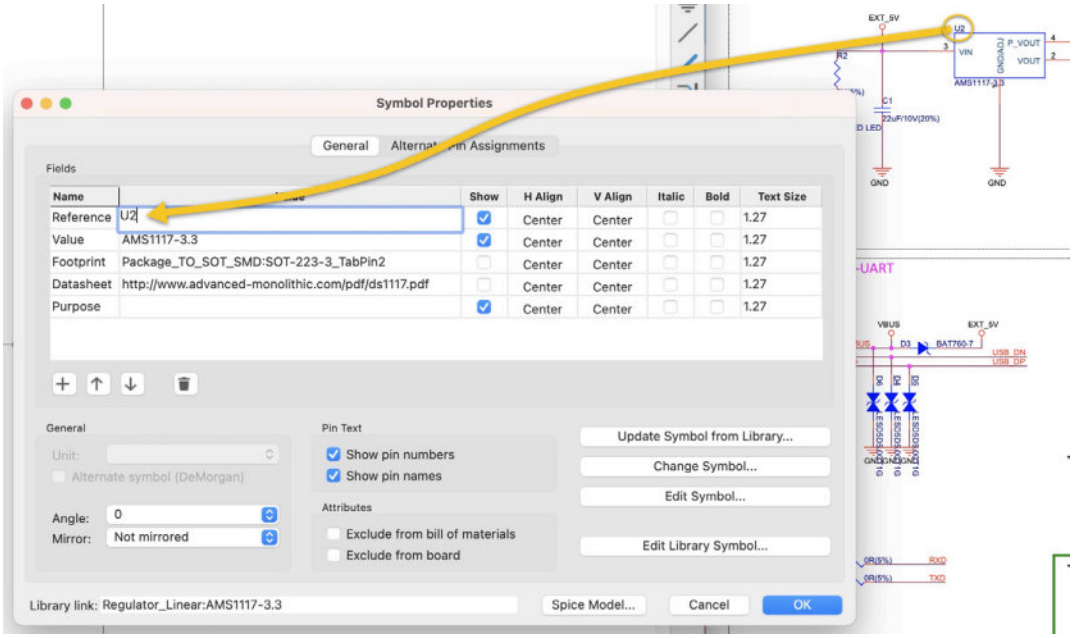


Figure 13.2.3.11: The Properties window for the voltage regulator symbol.

Continue with other symbols in the power supply group (resistor, LED, capacitor). Remember to add the values as you work your way through the symbols. If this matches better your working style, you can choose to leave the values for a later time, and edit them in bulk using the Symbol Fields Table window.

Be methodical, and take you time. This is not something you should rush. It took me approximately 15 minutes to manually annotate the schematic. Below is the result:

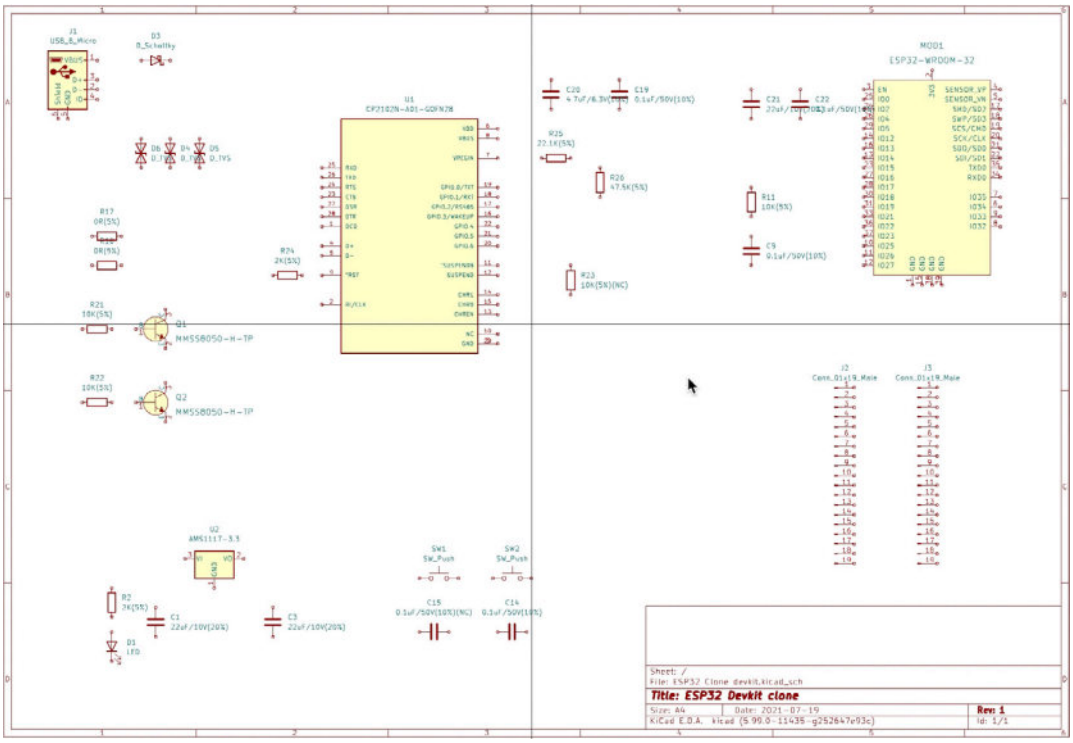


Figure 13.2.3.12: The schematic is fully annotated.

Let's continue in the next step for the workflow where you will set the symbols in their final positions before the wiring in step four of the workflow.

2.4. Schema 3 - Arrange

You are still working in step three of the schematic design workflow. You have added the symbols in the sheet, and manually annotated them. In this segment, you will finalise the position of each symbol in the sheet so that they are in appropriate positions ahead of the wiring step.

Most of the symbols are positions that are close to where I'd like them to be. Most of the work that I had to do here was to make small changes to those positions. For example, I aligned resistors and capacitors so that I can connect them with straight wire lines later. I also took the opportunity to delineate the functional groups with graphical boxes. This is something that I usually leave for step seven of the workflow ("comments") but I decided to do earlier in this project to help me reduce the risk of error as I am working with the reference design.

Below you can see the schematic as it looks before I started changing the position of the symbols:

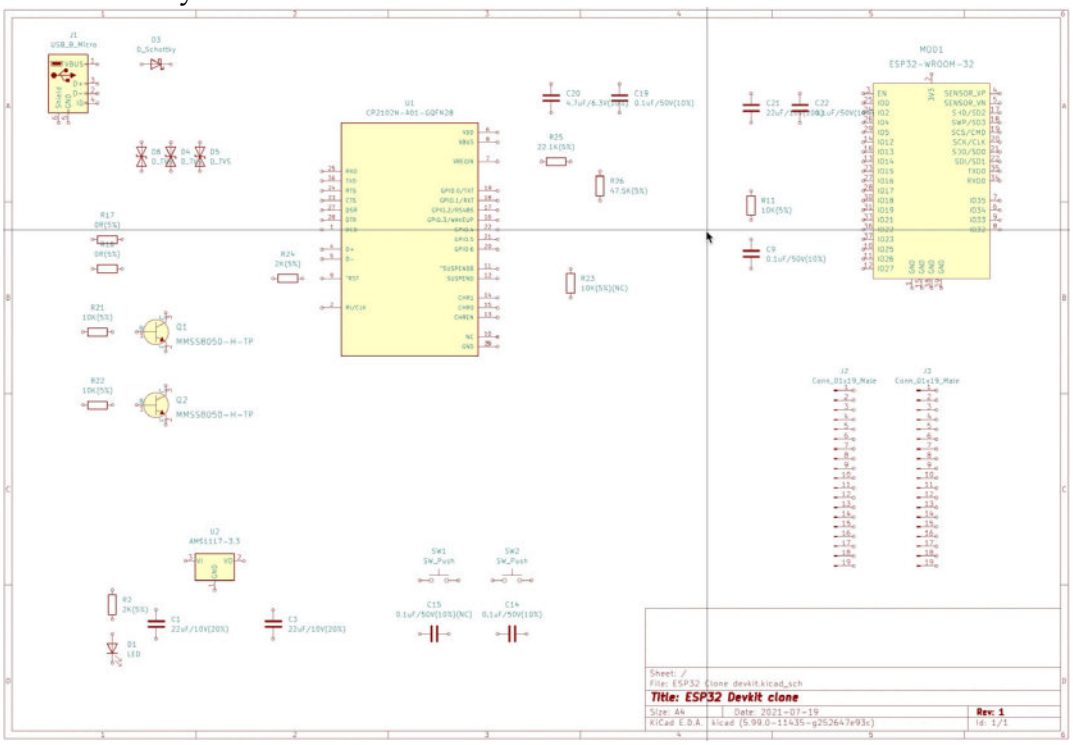


Figure 13.2.4.13: Starting to reposition the symbols.

Below you see see the schematic after I completed this work:

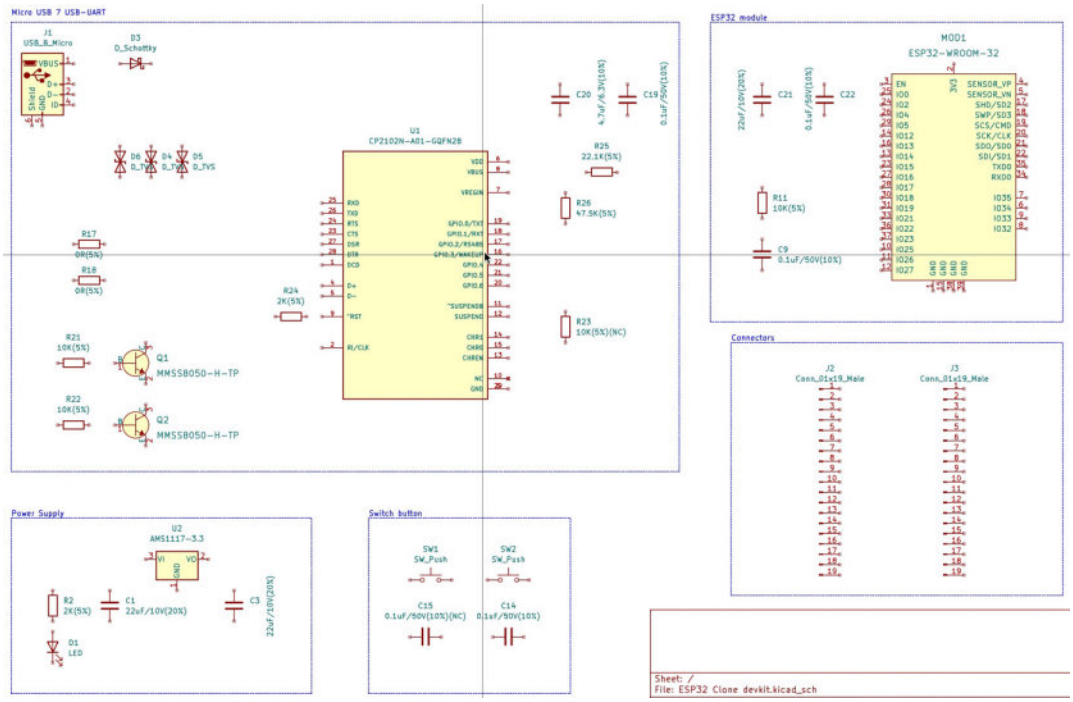


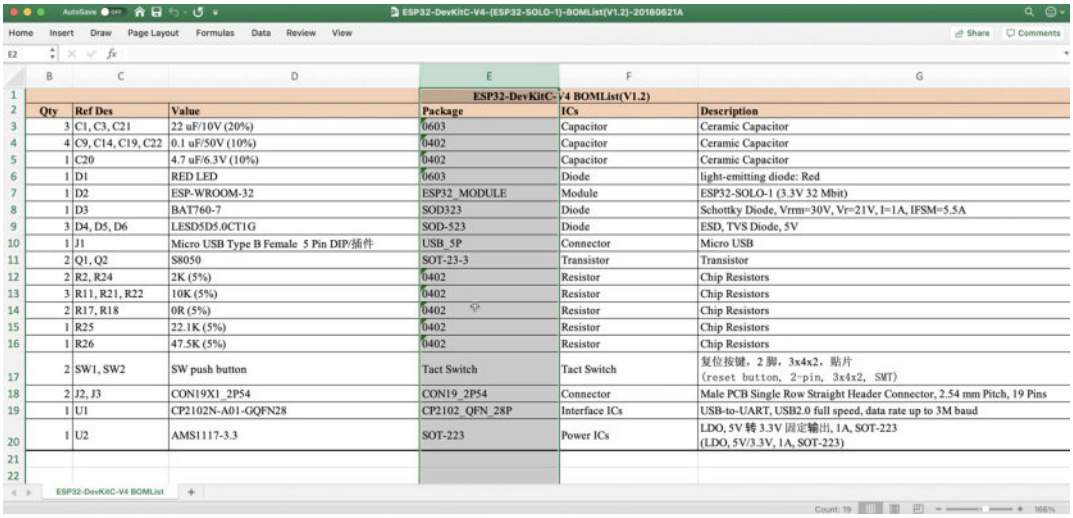
Figure 13.2.4.14: Completed the repositioning of the symbols.

As you can see, the difference between the first and second version of the schematic are small. However, the newer version is more readable.

Let' finish step three of the schematic workflow by associating symbols with their footprint counterpart.

2.5. Schema 3 - Associate

In this segment you will complete step three of the schematic design workflow by associating the symbols with their footprints. To do this work, you will need the BOM file that came with the reference design archive that you downloaded from Espressif. You will find this file inside a folder named "05_BOM List". It's file name is "ESP32-DevKitC-V4-(ESP32-SOLO-1)-BOMList(V1.2)-20180621A.xlsx". Below, you can see this file open in Microsoft Excel:



ESP32-DevKitC-V4-(ESP32-SOLO-1)-BOMList(V1.2)-20180621A

ESP32-DevKitC-V4 BOMList(V1.2)

Qty	Ref Des	Value	Package	ICs	Description
3	C1, C3, C21	22 uF/10V (20%)	0603	Capacitor	Ceramic Capacitor
4	C9, C14, C19, C22	0.1 uF/50V (10%)	0402	Capacitor	Ceramic Capacitor
1	C20	4.7 uF/6.3V (10%)	0402	Capacitor	Ceramic Capacitor
1	D1	RED LED	0603	Diode	light-emitting diode: Red
1	D2	ESP-WROOM-32	ESP32_MODULE	Module	ESP32-SOLO-1 (3.3V 32 Mbit)
1	D3	BAT760-7	SOD323	Diode	Schottky Diode, Vrrm=30V, Vr=21V, I=1A, IFSM=5.5A
3	D4, D5, D6	LESD5D5.0CT1G	SOD-523	Diode	ESD, TVS Diode, 5V
1	J1	Micro USB Type B Female 5 Pin DIP/插件	USB_5P	Connector	Micro USB
2	Q1, Q2	S8050	SOT-23-3	Transistor	Transistor
2	R2, R24	2K (5%)	0402	Resistor	Chip Resistors
3	R11, R21, R22	10K (5%)	0402	Resistor	Chip Resistors
2	R17, R18	0R (5%)	0402	Resistor	Chip Resistors
1	R25	22.1K (5%)	0402	Resistor	Chip Resistors
1	R26	47.5K (5%)	0402	Resistor	Chip Resistors
2	SW1, SW2	SW push button	Tact Switch	Tact Switch	复位按键，2 脚，3x4x2，贴片 (reset button, 2-pin, 3x4x2, SMT)
2	J2, J3	CON19X1_2P54	CON19_2P54	Connector	Male PCB Single Row Straight Header Connector, 2.54 mm Pitch, 19 Pins
1	U1	CP2102N-A01-GQFN28	CP2102_QFN_28P	Interface ICs	USB-to-UART, USB2.0 full speed, data rate up to 3M baud
1	U2	AMS1117-3.3	SOT-223	Power ICs	LDO, 5V 转 3.3V 固定输出, 1A, SOT-223 (LDO, 5V/3.3V, 1A, SOT-223)

Figure 13.2.5.15: The reference design BOM file.

To find the appropriate footprint, use the information in column E of the BOM file. For example, for capacitor C1, the reference design recommends the 0603 package for a ceramic capacitor. In my implementation of this board, I have followed the reference design recommendations.

In Eeschema, use the Footprint assignment window to make the associations in bulk. If you prefer, you can also use the Properties window for each symbol (a more time-consuming option).

I recommend that you take some time to browse through the footprint libraries on your own, until you can find the required footprints. If you need to, you can consult the BOM table in the introduction of this project. There, you will find the full list of footprints that I have used.

Below you can see my completed associations table:

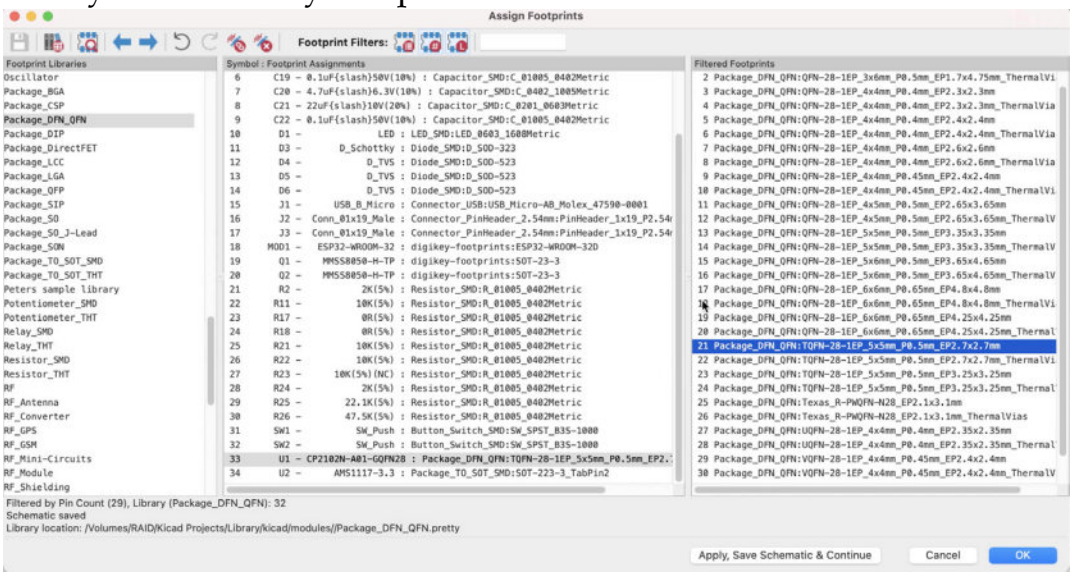


Figure 13.2.5.16: The completed associations table.

If you are unable to find any of the footprints, especially those from Snapeda and Digikey, ensure that you have correctly install the necessary libraries.

Once you have completed the associations, continue with step four of the workflow: wiring.

2.6. Schema 4 - Wiring

With the help of the schematic reference from Espressif, wiring your schematic should be straight-forward. Symbols are already placed in their functional groups. As in previous projects, the main strategy here is to use graphical line wires to connect pins within the same functional group, and net labels for everything else.

Place the Eeschema window next to the PDF viewer with the reference design, and start wiring. Below you can see the completed wiring for the first functional group, the power supply:

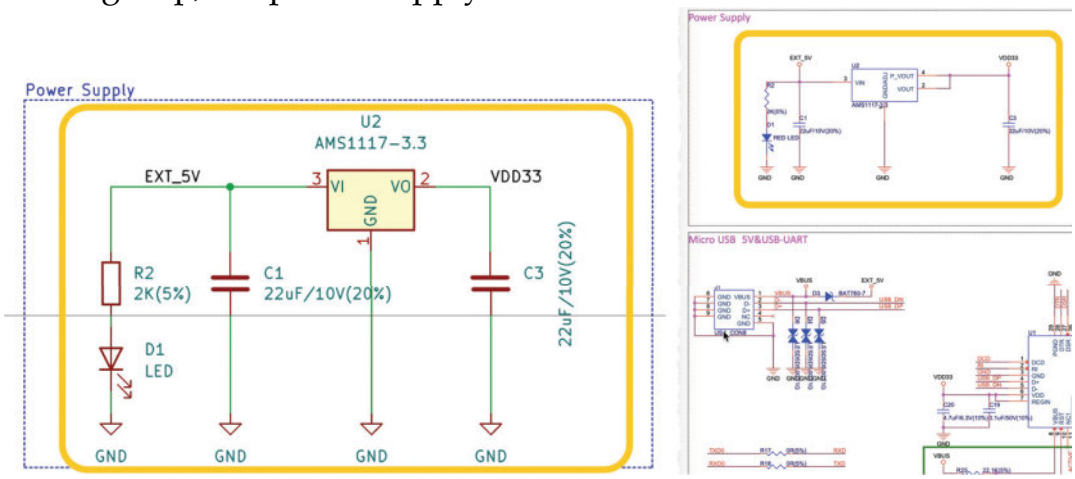


Figure 13.2.6.17: Power supply group wiring is complete. Left: my wiring. Right: reference.

Continue with the switch buttons:

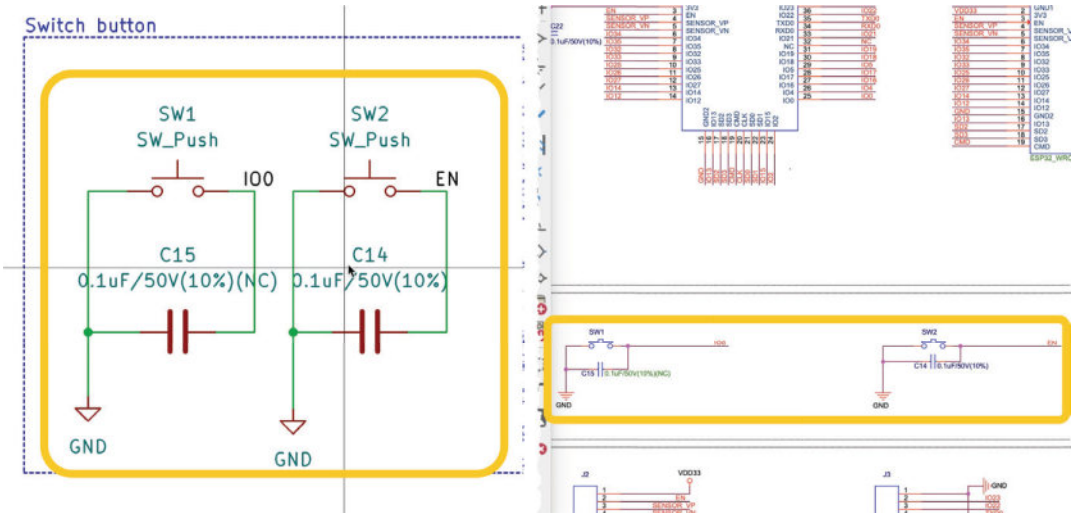


Figure 13.2.6.18: Switch buttons group wiring is complete. Left: my wiring. Right: reference.

Next up, the connectors, where all wirings are done with net labels:

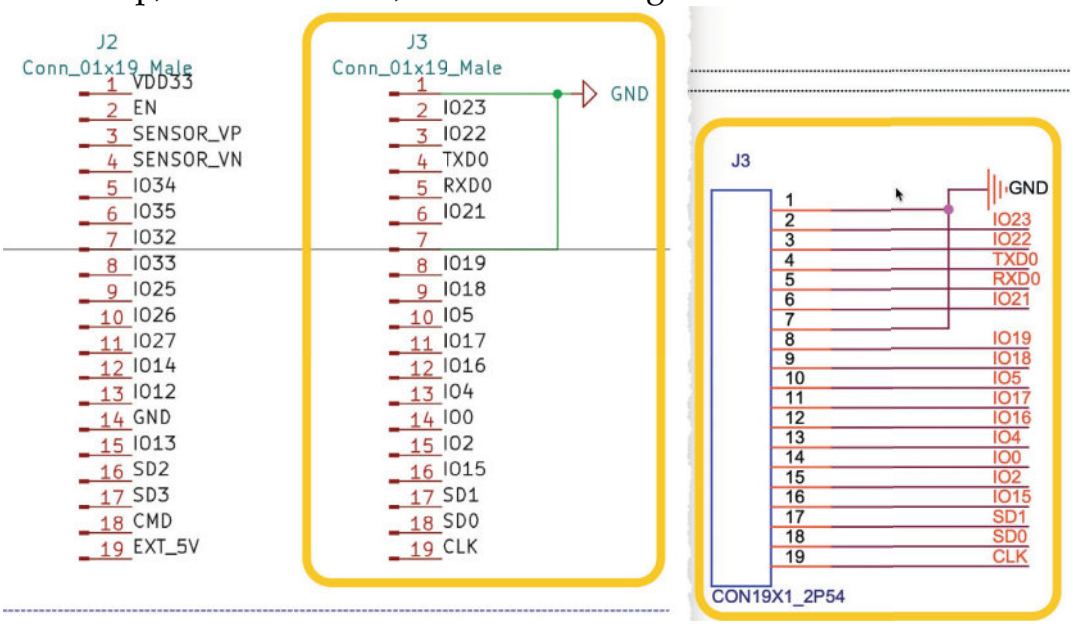


Figure 13.2.6.19: Connectors group wiring is complete. Left: my wiring. Right: reference. Not showing reference for J2.

Next, the ESP32 module group:

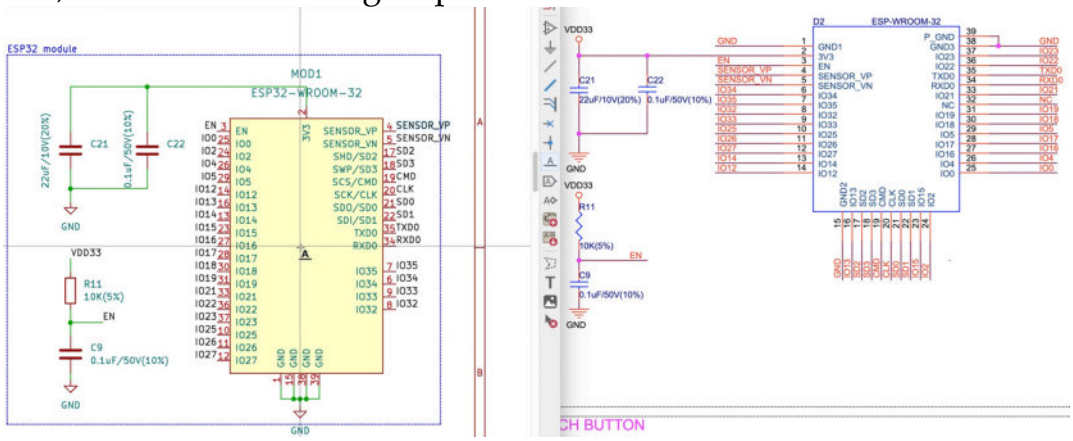


Figure 13.2.6.20: ESP32 group wiring is complete. Left: my wiring. Right: reference.

Finally, the Micro USB group, which is the most complicated part of the schematic:

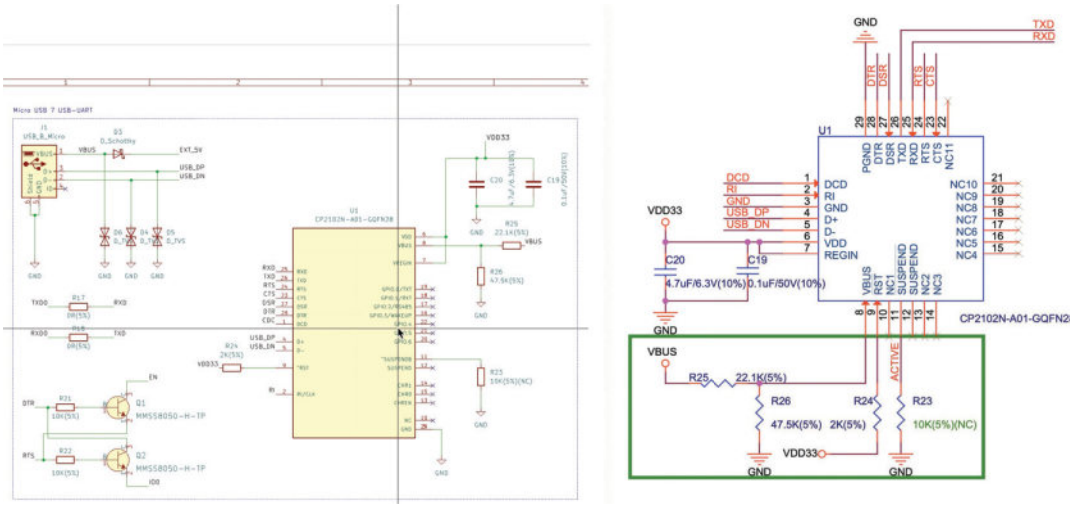


Figure 13.2.6.21: Micro USB group wiring is complete. Left: my wiring. Right: reference.

Here is the schematic, with wiring complete:

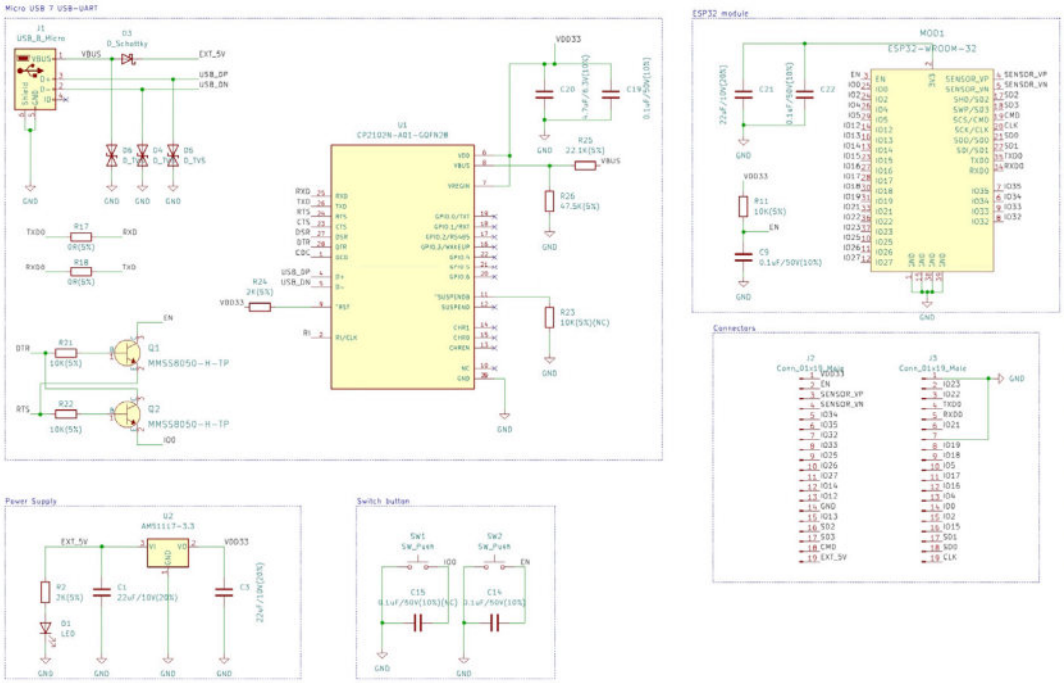


Figure 13.2.6.22: Schematic wiring is complete.

Let's continue with part five of the workflow. You have already created the nets, so what is left to do is to create the next classes and associated then with nets.

2.7. Schema 5 - Nets and Net Classes

In the wiring step that you completed in step four of the workflow, you used net labels alongside graphic wire lines. As a result, you have already created all required named nets. Therefore, to complete step five of the schematic design workflow, you only need to setup the net classes, and associate nets with those classes.

In addition to the default class, go ahead and create three more classes. In each class, assign the nets that I list below:

- Default (already exists): assign all net except those listed below.
- GPIO: All IOxx nets.
- POWER: EXT_5V, VBUS, VDD33, GND.
- USB: RXD, RXD0, TXD, TXD0, USB_DN, USB_DP.

Below is my schematic setup window showing the four net classes with some of the net allocations.

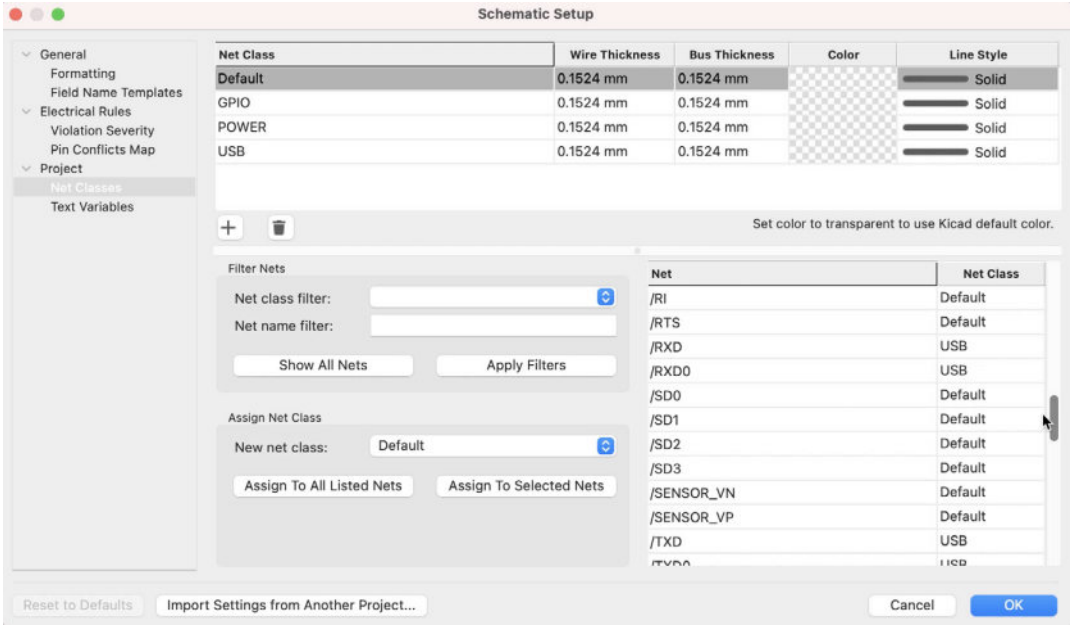


Figure 13.2.7.23: My net class setup.

Let's continue with the Electrical Rules Check.

2.8. Schema 6 - Electrical Rules Check

Time to do the Electrical Rules Check. I usually forget a few simple things, like annotate GND or power labels. My ERC window indicates that the schematic is not fully annotated. The non-annotated symbols are the GND symbols. I don't need to manually annotate those symbols, so I will use the annotator tool and have them automatically annotated.

With the schematic fully annotated, re-run the ERC and click on Run ERC. The checker reports three errors and one warning:

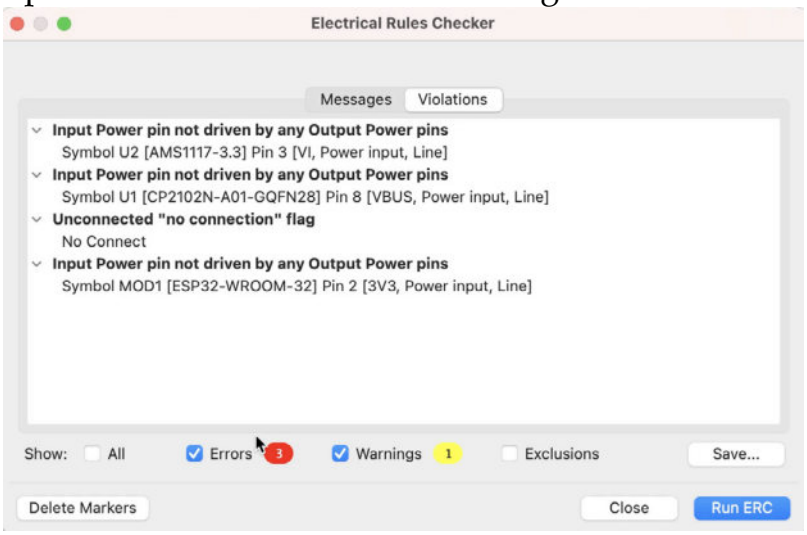


Figure 13.2.8.24: The ERC report.

The first two errors indicate that there are two pins that are marked as "input power pins" that are not actually connected to an "output power pin". To fix this error, attach POWER_FLAG symbols to the affected nets. Below you can see the fix for the first error:

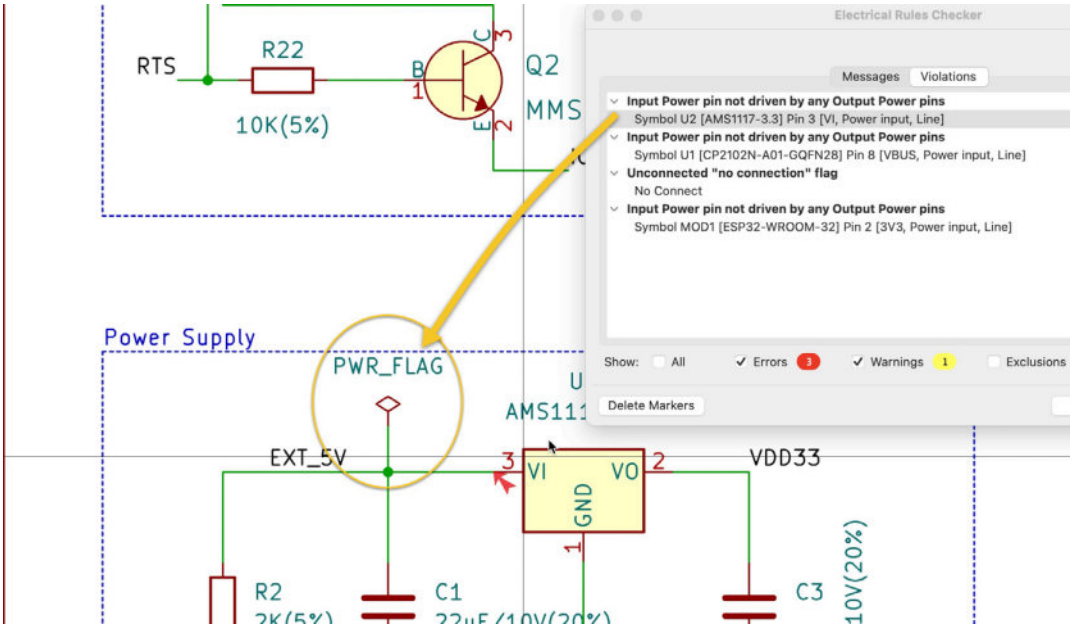


Figure 13.2.8.25: Fixing an "Input Power pin not driven by any Output Power pins" error.

Fix the second and fourth errors in the same way.

The third error (actually, an "Unconnected "no connection" warning) is unusual but easy to fix. See this error below:

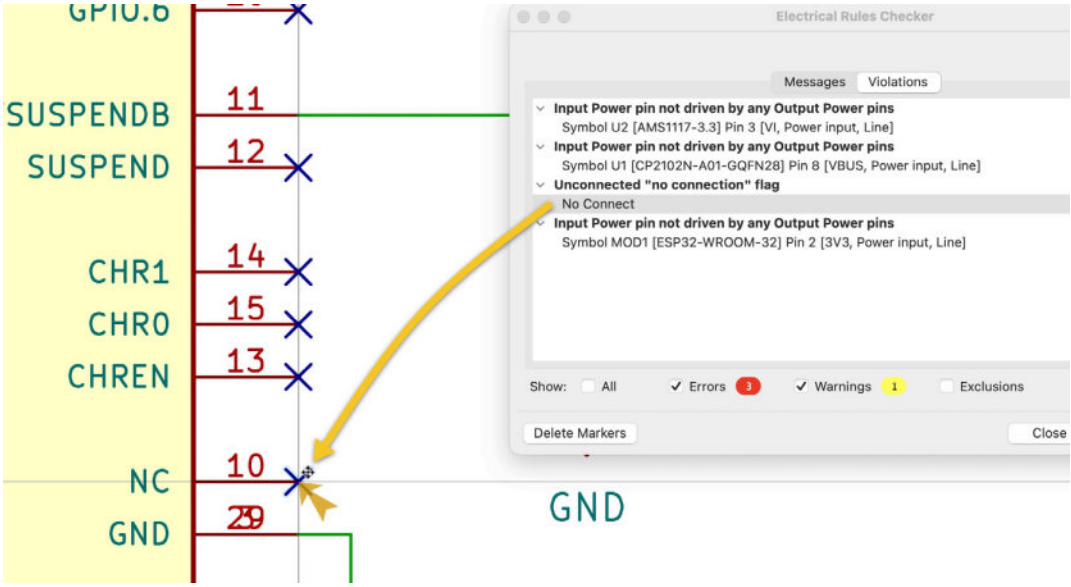


Figure 13.2.8.26: An unusual “Unconnected” warning.

The origin of this warning is that the ESP32 symbol pin 10 (with name “NC”) already has an “unconnected” symbol attached to it. I did not notice this when I added a second unconnected symbol on top of the original. Simply delete the “X” symbol on pin 10, and run the ERC once more. Here is the result:

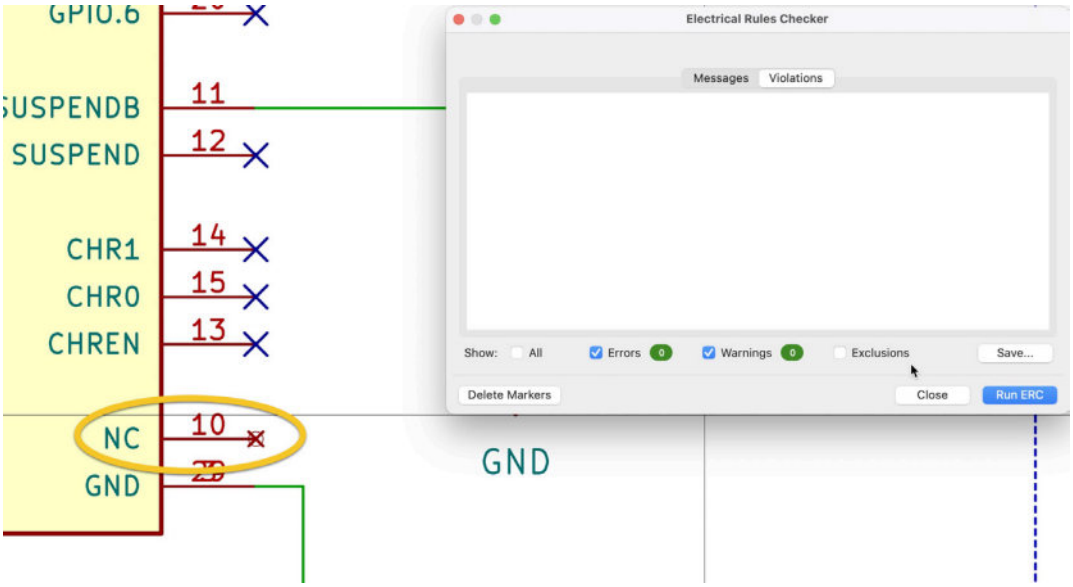


Figure 13.2.8.27: Fixed last warning.

Let’s finish the schematic design workflow in the next segment of this chapter, where you will add additional comments to the schematic.

2.9. Schema 7 - Comments

You have already done most of the commenting work earlier in the process. The schematic design contains boxes to demarcate the functional groups, and those boxes have names.

The reference design contains a truth table for the DTR and RTS signals. I'd like to include that in my schematic with the use of a text box. Below, you can see the truth table in the reference design (right) and the location I'd like to place it in my design:

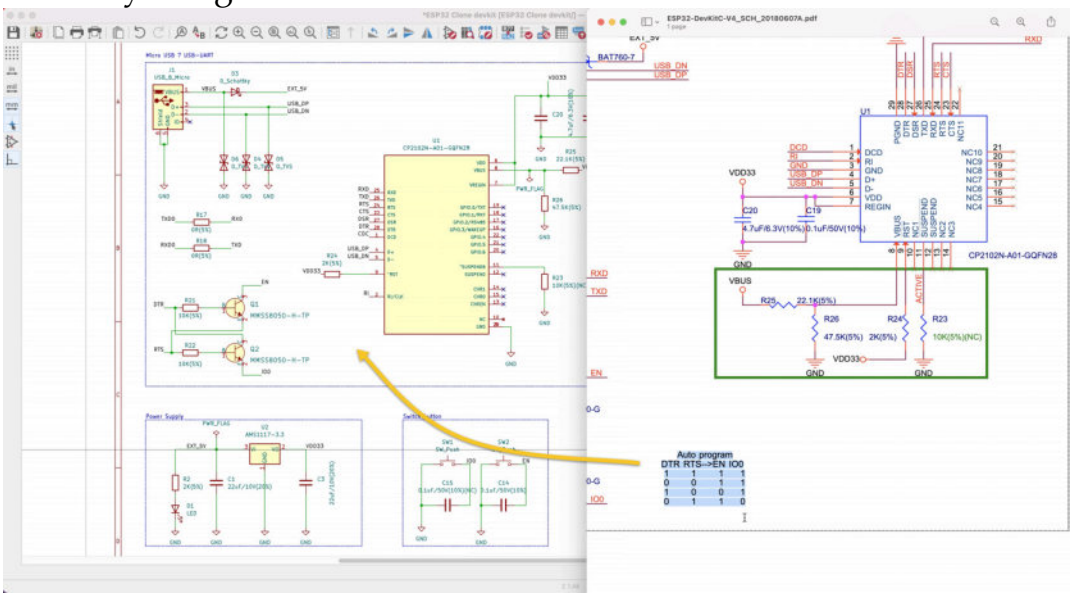


Figure 13.2.10.28: The truth table in the reference schematic.

Below you can see the text label that contains the truth table text, and its location in my schematic design:

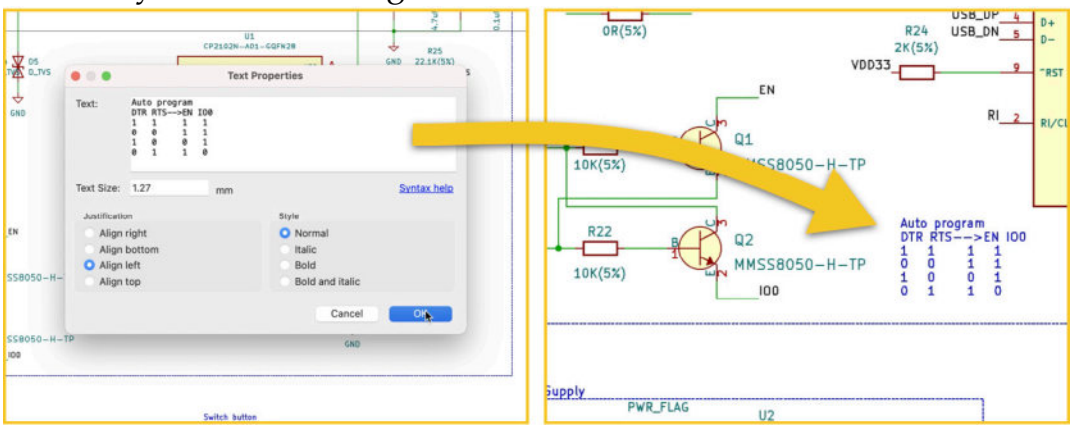


Figure 13.2.10.29: The truth table in my schematic design.

Another improvement I'd like to make is to reduce the size of the component value text. The default size is too big, causing for text to spill over other elements of the schematic, like in the example below:

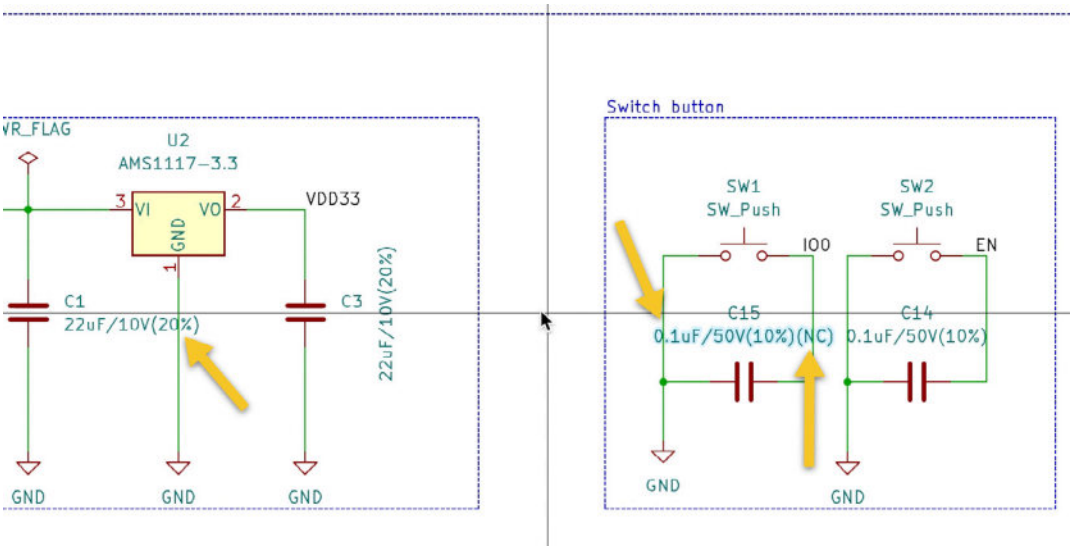


Figure 13.2.10.30: Value text is too big.

To change the size of all value text elements, use the “Edit Text & Graphics Properties…” tool from the “Edit” menu. Set it to change all “Values” to “Text size” “1 mm”:

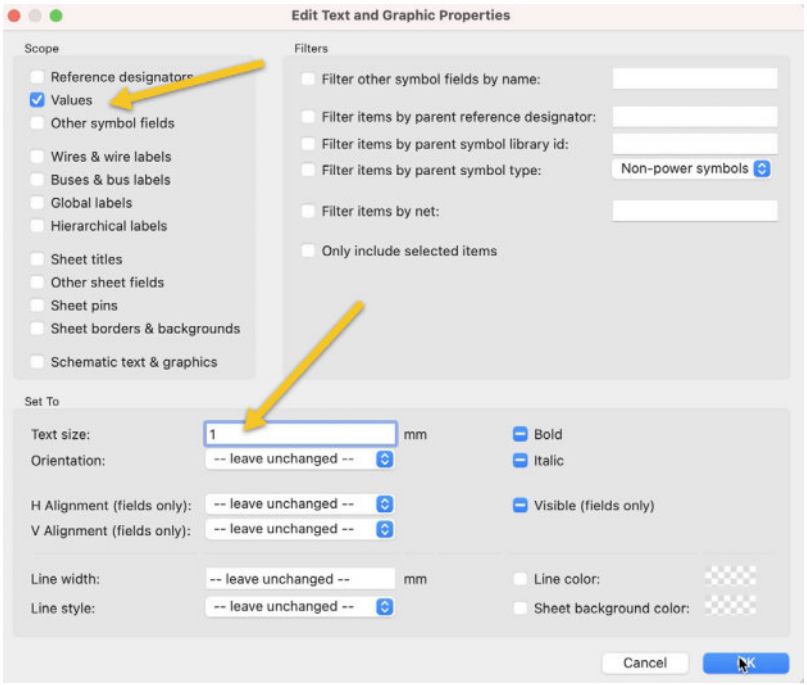


Figure 13.2.10.31: Changing size for all Value text elements.

This change will allow value text elements to better fit in the available space:

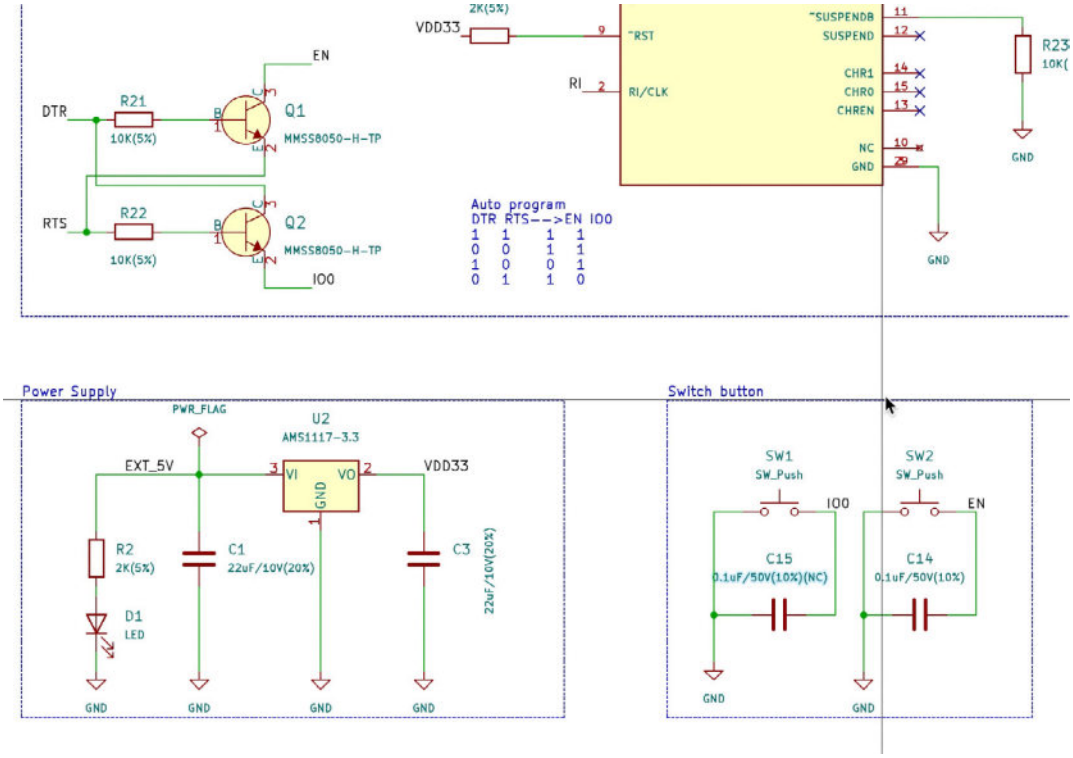


Figure 13.2.10.32: Value text fits better.

This completes the schematic design workflow. Let's continue with the layout.

3. Layout design

In the previous chapter you completed work on the schematic design of the ESP32 development kit clone. You are now ready to begin work on the layout design.

Before you start work in Pcbnew, take a few minutes to review the objectives of the layout design workflow.

The main objective is to design an ESP32 development board that can fit in a mini breadboard. It will have a simple rectangular shape, with two pin rows along the long edges. It will have the USB connector along one of the narrow edges. Below is the 3D rendering of the board that I designed as I worked on this project:

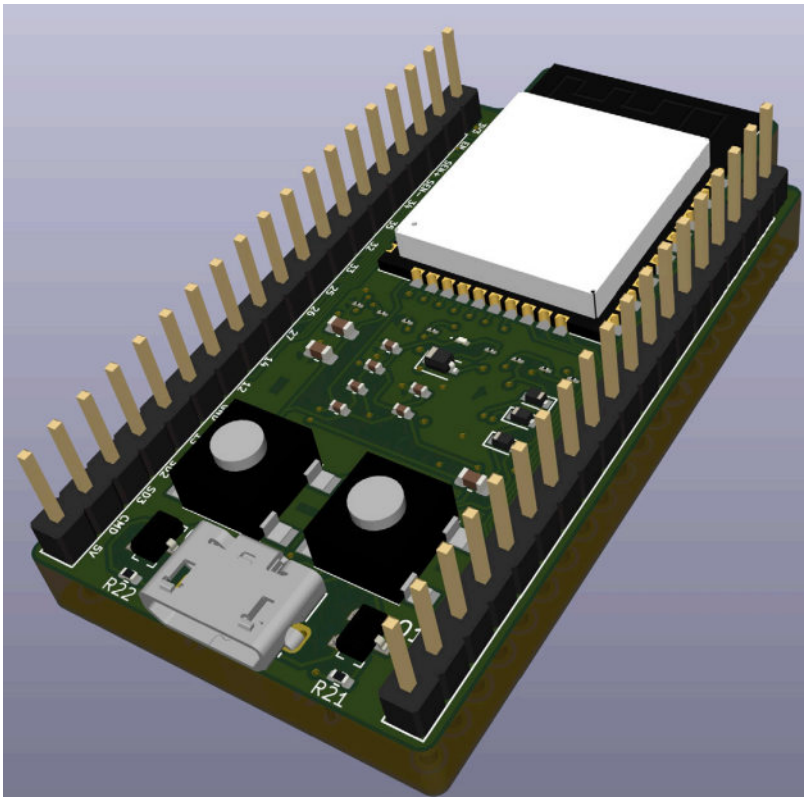


Figure 13.3.1: The main objective of this project.

Below is the layout of the board, as it looked when I finished work on it:

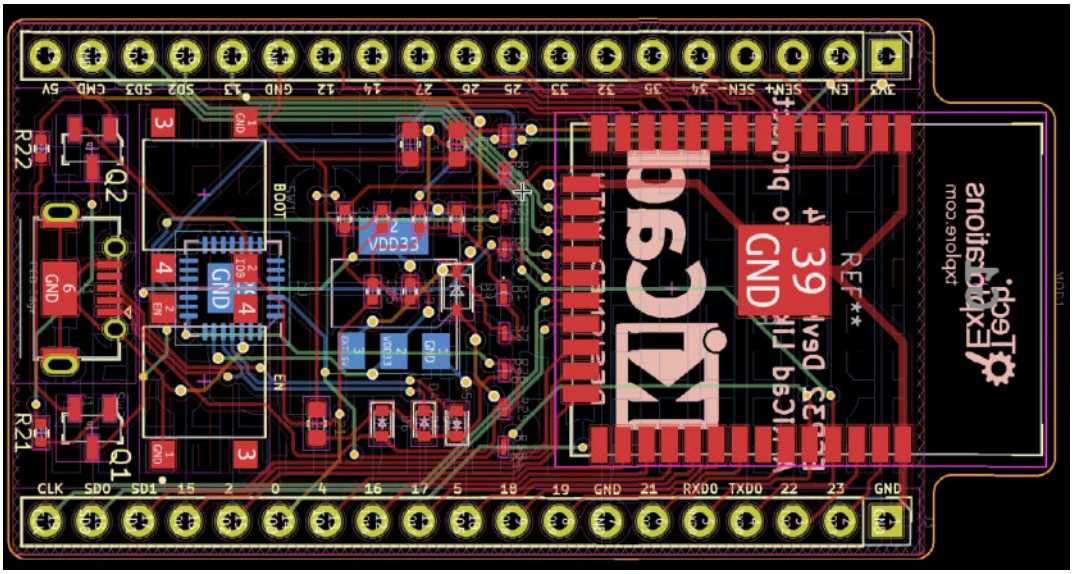


Figure 13.3.2: The PCB as it looks in the layout editor at the end of this project.

There are two main geometrical constraints:

1. Minimise the size of the board in order to minimise its cost.
2. Ensure that it can be plugged into a double mini-breadboard.

In the photograph below I show a measurement that I took that shows the required distance between the pin rows:

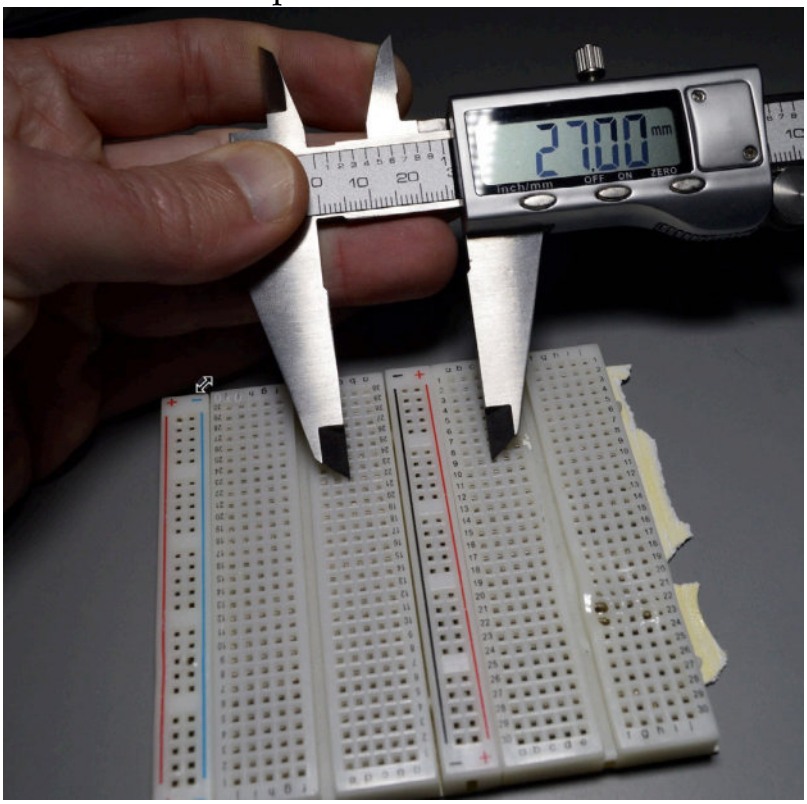


Figure 13.3.3: Measuring the distance between the rows.

Based on this measurement, I will design the PCB so that the distance between the rows is approximately 27 mm. This will allow me to use my

ESP32 development kit with a two mini-breadboards combined, as in the photograph above.

In addition to the above, this project's layout design will give you the opportunity to practice:

1. Working with differential pairs.
2. Working with components on both sides of the board.
3. Adjust the default design rules to allow work with small components that otherwise would cause violations.

Let's begin by setting up the layout design editor.

3.1. Layout 1 - Setup

Let's setup the layout editor to match the requirements of the work ahead.

Open the Board Setup window and edit the following settings (if I have not listed a setting below, leave it at its default value):

- Board Stackup.
  - Physical Stackup.
    - Copper layers: 4.
  - Board Editor Layers:
    - F.Cu: mixed.
    - In1.Cu: mixed.
    - In2.Cu: mixed.
    - B.Cu: mixed.
  - Text & Graphics.
    - Defaults.
      - Silk Layers.
        - Line Thickness: 0.15 mm.
        - Text Width: 0.8 mm.
        - Text Height: 0.8 mm.
        - Text Thickness: 0.1 mm.
  - Design Rules.
    - Minimum track width: 0.1 mm.
  - Net Classes.
    - GPIO and USB.
      - Clearance: 0.1 mm
      - Track Width: 0.1 mm
      - Via Size: 0.5 mm

        - Via Hole: 0.35 mm
        - uVia Size: 0.25 mm
        - uVia Hole: 0.1 mm
        - DP Width: 0.1 mm
        - DP Gap: 0.15 mm
        - Assigned nets GPIO: all IOxx nets.
        - Assigned nets USB: RXD, RDX0, TXD, TXD0, USB_DN, USB_DP
    - POWER.
        - Clearance: 0.1 mm
        - Track Width: 0.15 mm
        - Via Size: 0.6 mm
        - Via Hole: 0.4 mm
        - uVia Size: 0.25 mm
        - uVia Hole: 0.1 mm
        - DP Width: 0.2 mm
        - DP Gap: 0.25 mm
        - Assigned nets: EXT_5V, VBUS, VDD33, GND
    - Default:
        - Clearance: 0.1 mm
        - Track Width: 0.15 mm
        - Via Size: 0.6 mm
        - Via Hole: 0.35 mm
        - uVia Size: 0.25 mm
        - uVia Hole: 0.1 mm
        - DP Width: 0.1 mm
        - DP Gap: 0.15 mm
    - Data: will setup later.

The reason I have reduced the sizes for the GPIO and USB net classes is that some of the components on the board are so small that the default design rules would trigger multiple clearance violations. In the example below, the arrows point to resistors that use the 0402 package. To prevent clearance violation errors from occurring, I have changed the minimum clearance for members of the GPIO class to 0.1 mm.

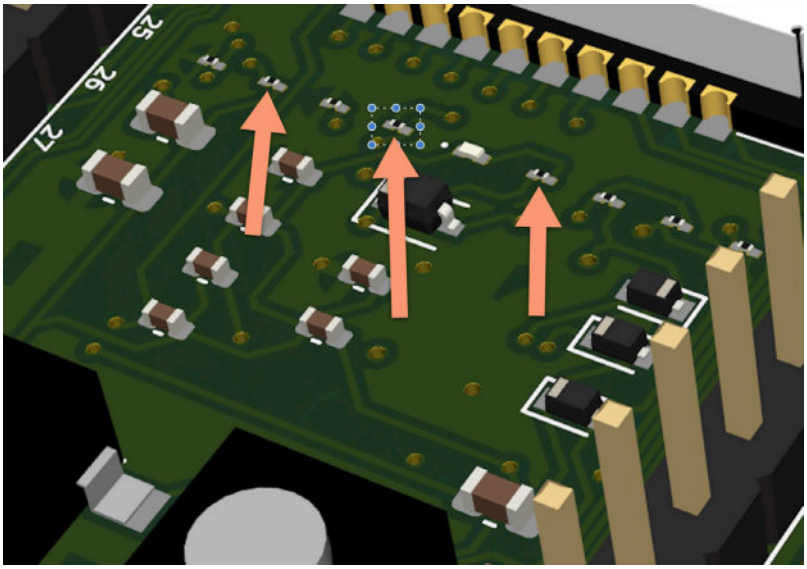


Figure 13.3.1.4: Small components require smaller clearances in the design rules.

Similarly, this board contains a couple of differential pair tracks. Those differential pairs are located in the top left of the board, between the ESP32 module pads and the pins row. Because the distance between the pads is very small, I had to reduce the clearance of the differential pairs in the design rules.

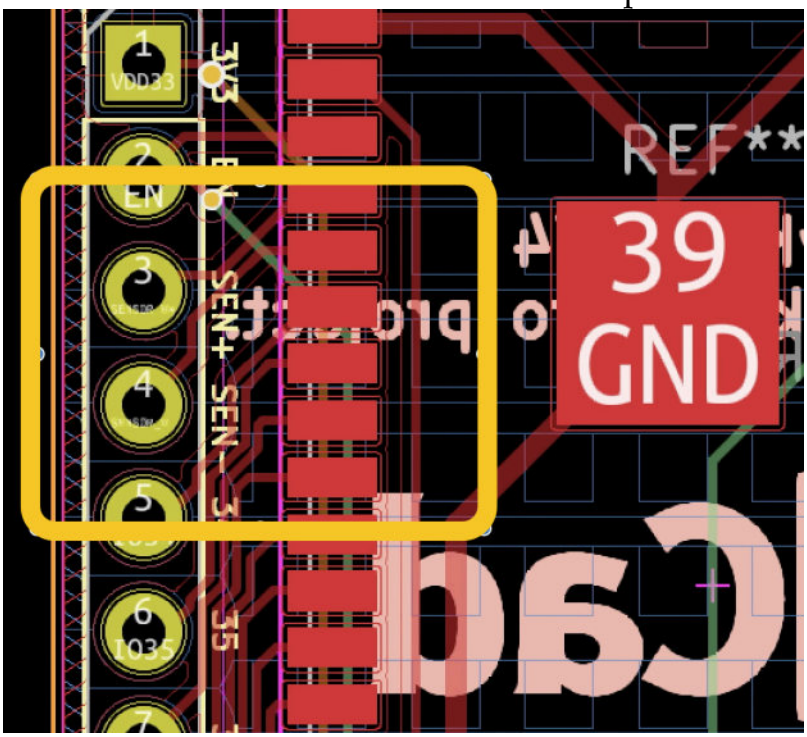


Figure 13.3.1.5: Differential pairs require adjusted clearances.

Finally, setup the editor page in the Page Settings window:

- Page Settings.
    - Issue Date: copy today's date.
    - Revision: 1

- Title: ESP32 Custom Dev Kit

Save your work, and continue to step two of the layout workflow.

3.2. Layout 2 - Outline and constraints

Let's continue with Step two of the layout workflow. Here, you will draw a rough outline of the board in the Edge.Cuts layer based on the constraints I described in the setup step.

The main geometrical constraint is that you want to design the board so that the two pin headers are approximately 27 mm apart. This will allow the board to plug into a double mini-breadboard. In the photo below, I show my caliber measuring the distance between the two rows where I want to place the pin headers:

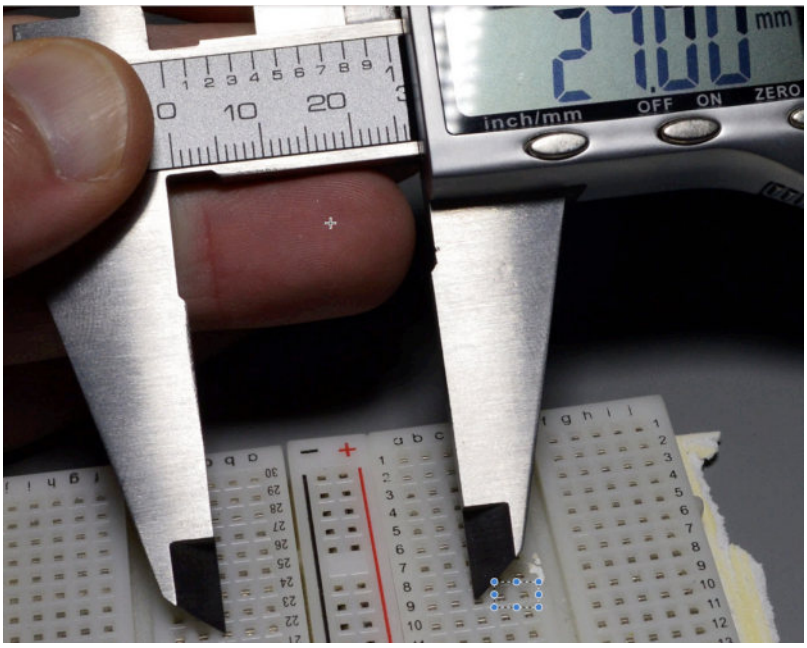


Figure 13.3.2.6: The main geometrical constraint is the distance between the pin headers.

The measurement that I took with my calliper, 27 mm, contains some tolerance in it so I can vary it in the design. From my experimentation, I found that a distance in the range of 26.90 mm to 27.10 mm is acceptable. In my design, I will aim to the middle of this range.

Start by importing the footprint in the editor using the "Update PCB from Schematic" tool. You can see my editor window below with the bundle of the footprints just imported:

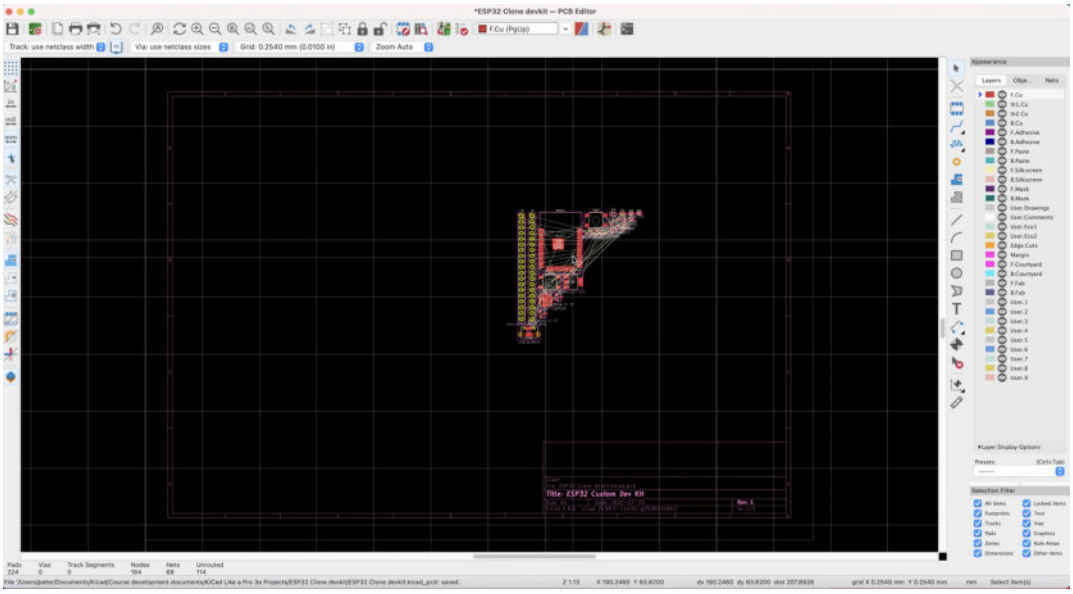


Figure 13.3.2.7: Imported footprints from schematic.

Since the geometrical constraints call for a specific distance between the two pin headers, I will use them to draw the rough outline of the board. Consult the reference design to find out the correct side for each pin header. Header J2 should go to the left side, and J3 to the right side.

Take footprint J2, place it towards the middle-left of the editor, and lock it. You will use J2 as an anchor to measure a distance of approximately 27 mm from the second header, J3. I have set my grid size to 0.127 mm. In the figure below, I have set J3 to be 26.92 mm away from J2. Notice the dx and dy values in the bottom status bar, and the grid size. Reminder: to reset the dx and dy counters, press the space bar.

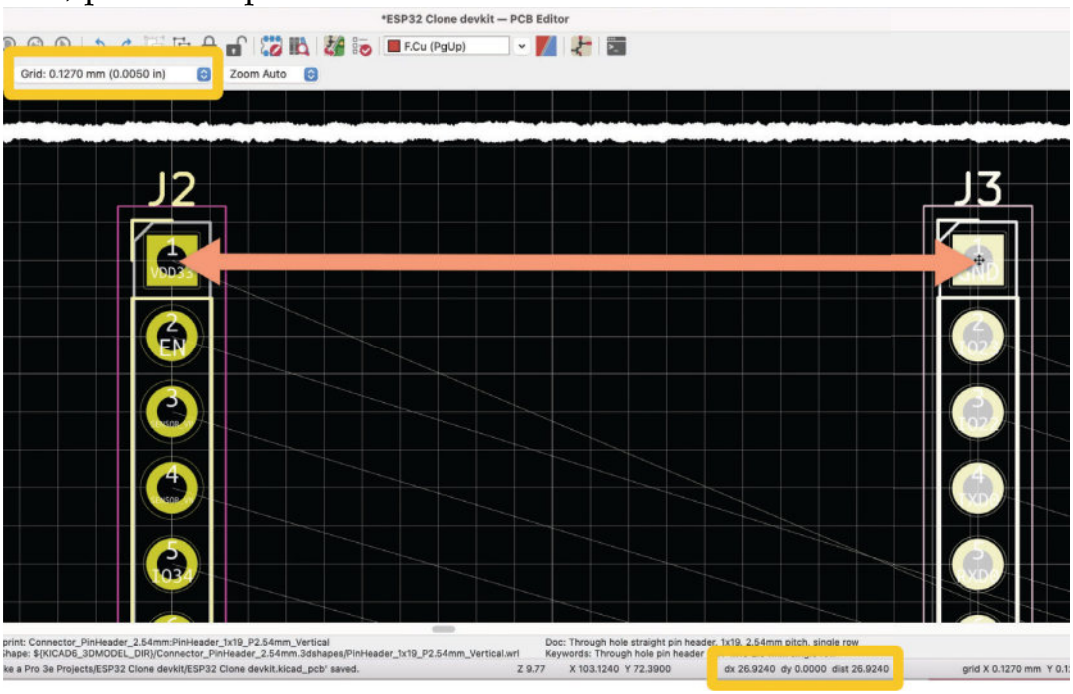


Figure 13.3.2.8: Setting J3 26.92 mm away from J2.

In the figure below, I have completed the placement of J3, and locked it in place. I then used the measurement tool to insert this measurement in the User.1 layer:

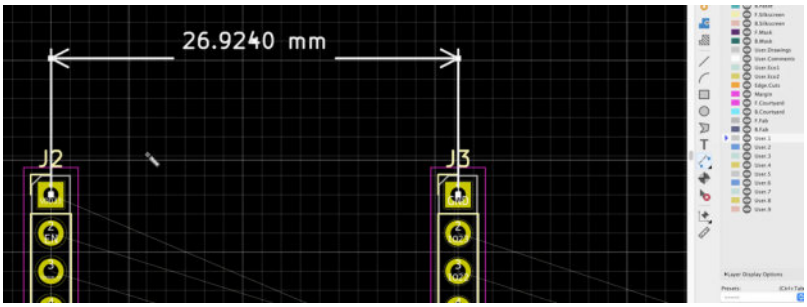


Figure 13.3.2.9: Pin header placement is complete.

Ensure the J2 and J3 are locked. Do a sanity check and confirm that the ESP32 module, which is the board's largest component, will fit between the two headers. As you can see below, it will:

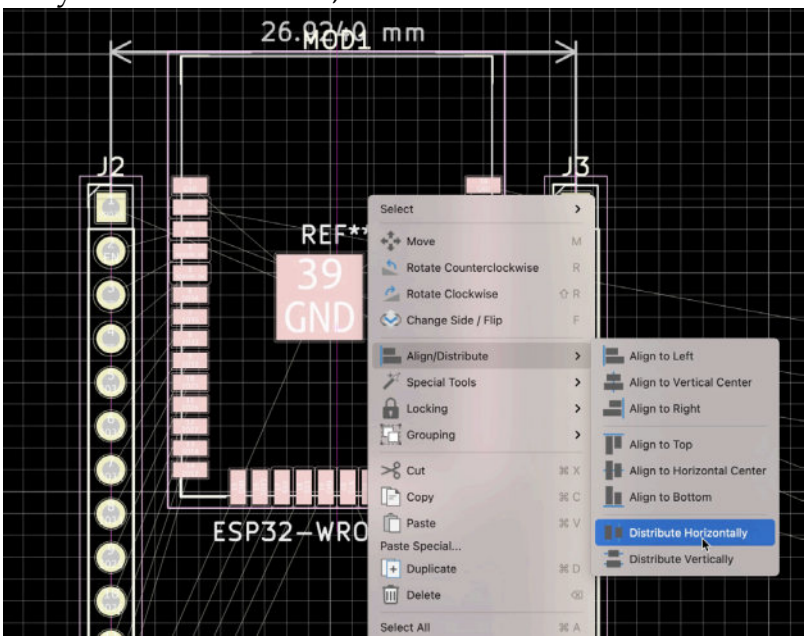


Figure 13.3.2.10: Centering the ESP32 module between the pin headers.

To ensure that the ESP32 module is centered between the pin headers, select all three footprints, right click to open the context menu, and select "Distribute Horizontally" from the "Align/Distribute" menu. Also shift the ESP32 module upwards so that the antenna area extends outside of the main board area. This ensure that the antenna circuitry is away from potential interference from other board electronics.

Lock the ESP32 module.

You can see the current state of the board below:

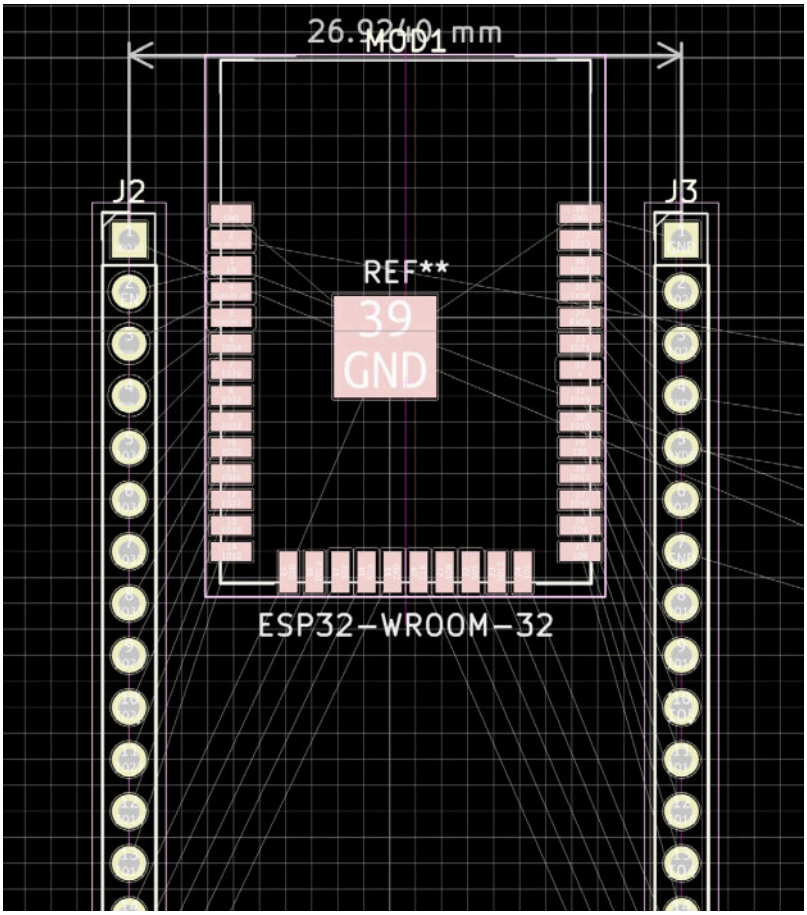


Figure 13.3.2.11: The first three footprints are locked in place.

Now that you have the footprints that define the shape and size of the board, you can proceed to draw the rough outline. Change the grid size to 0.254 mm, enable the Edge.Cuts layer, and choose the rectangle drawing tool. Below you can see the rough outline, in orange:

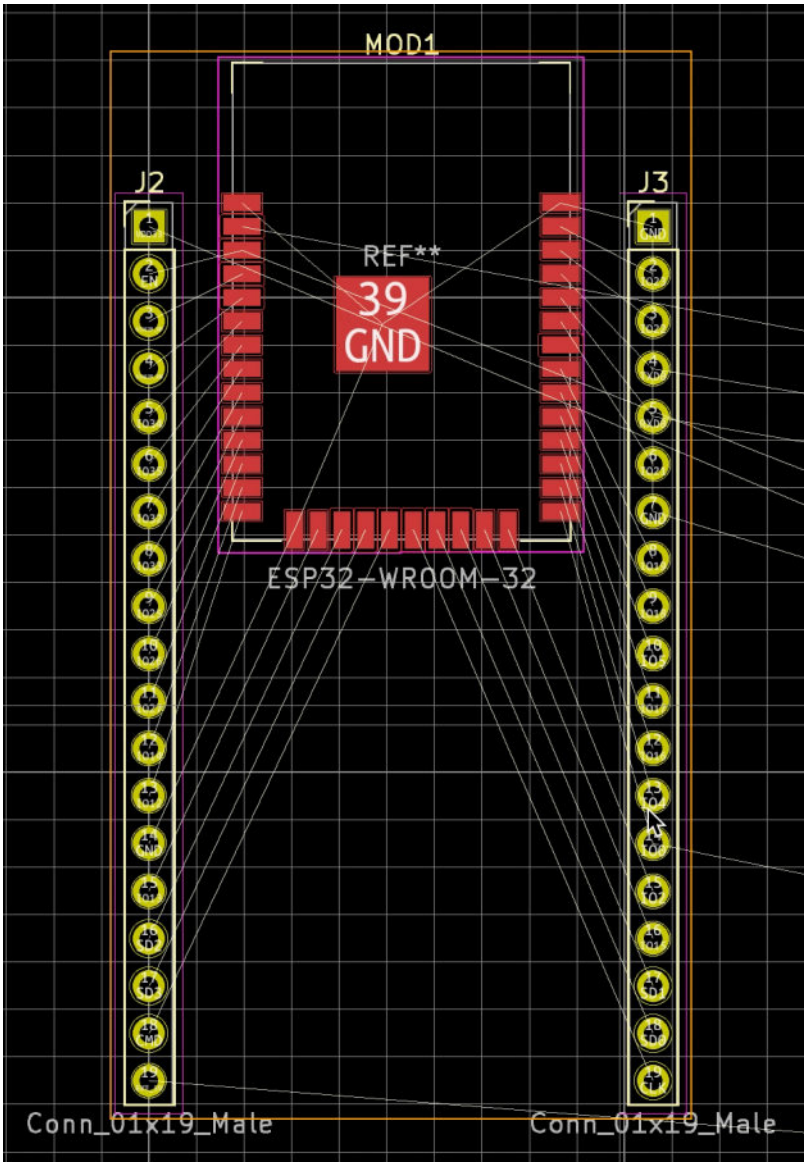


Figure 13.3.2.12: Completed the rough outline of the board.

Continue in the next segment where you will place the rest of the board footprints inside the rough outline.

3.3. Layout 3 - Place components

Continue to move the rest of the footprints within the rough outline. Move the footprints close to the board outline to make them easily accessible. At the start of the placement process, my editor looks like this:

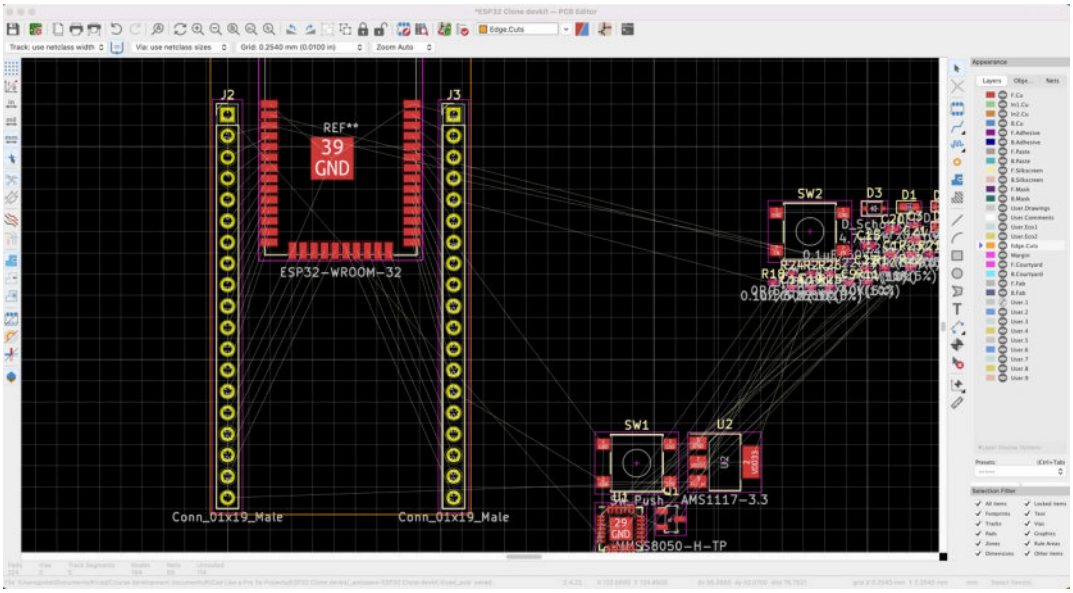


Figure 13.3.3.13: Beginning of the footprint placement process.

You can use the reference design for help with the appropriate position of each footprint, but also remember that you have freedom to vary. You can see the reference design tomorrow:

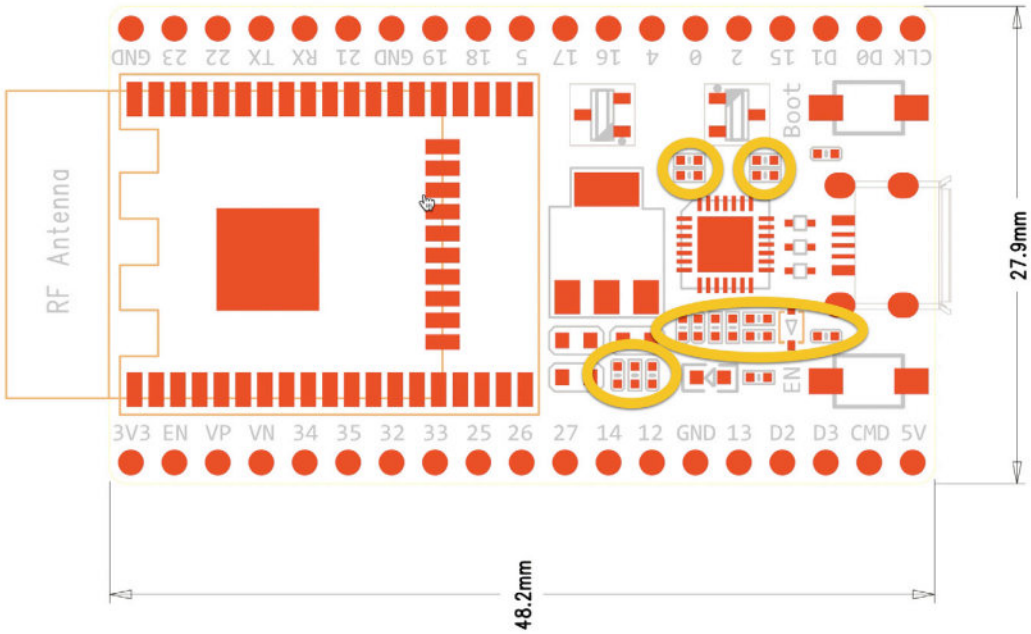


Figure 13.3.3.14: The board reference design.

In the figure above, I have used yellow circles to mark a few components that are very closely positioned. In the reference design this is no a problem, since the board will be manufactured by an assembly machine. However, I plan to assemble my board by hand and such packed positioning will make hand-assembly very difficult.

Instead of following the reference design blindly, I chose to make a variation. I decided to place the voltage regulator and USB-UART bridge in the back of the board. This will release much of the front of the board to space out the small capacitor and resistor footprints, which will make assembly easier.

I have decided to place other components, such as the buttons and the USB connector, as they appear in the reference design.

Continue with the placement of the USB connector (J1) and the buttons (SW1 and SW2). Use KiCad's alignment tools to centre and justify the footprints. Lock J1, SW1 and SW2 in place. Below is my layout at this time:

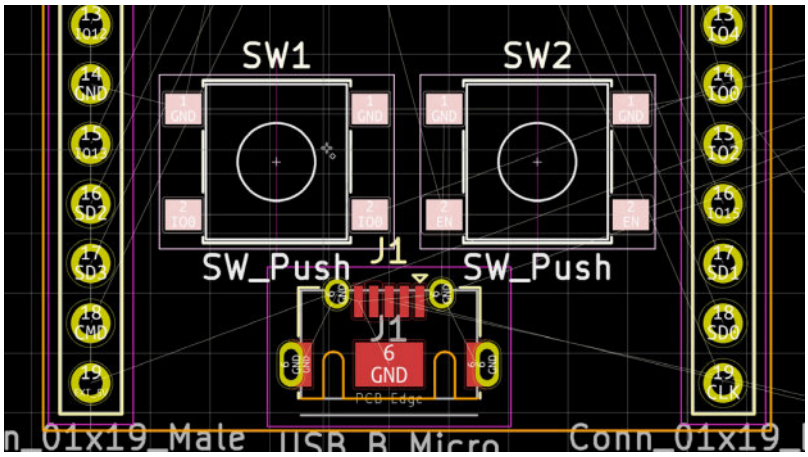


Figure 13.3.3.15: Completed placement for J1, SW1 and SW2.

I have decided to place U1 and U2 in the back of the board. Position them close to the front of the board (where the J1 connector is). To place a footprint in the back of the PCB, select "Back" in the Side field of the footprint's properties window:

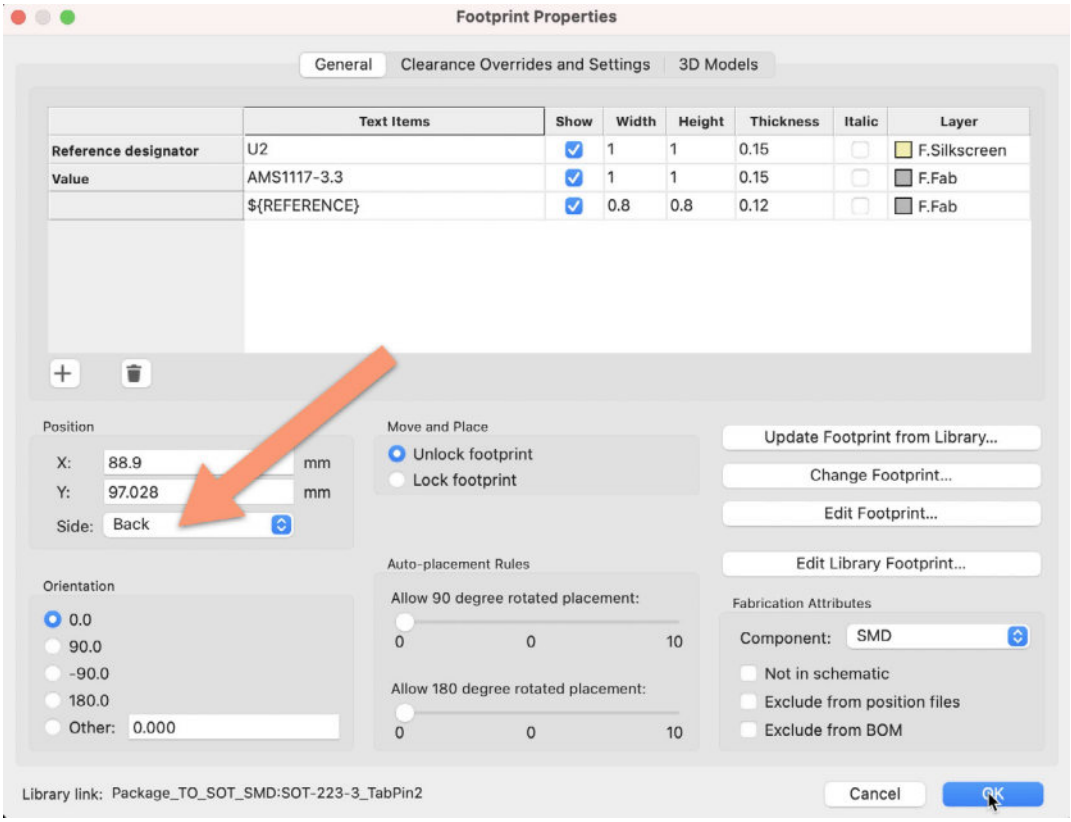


Figure 13.3.3.16: Placing a footprint in the back of the board.

Lock U1 and U2 in place. At this time, my layout looks like this:

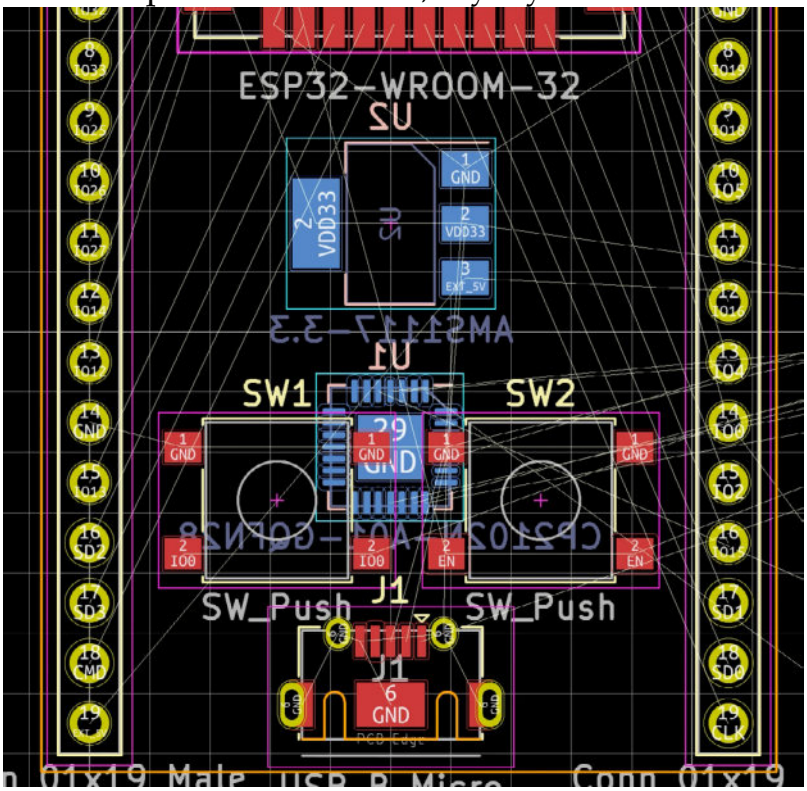


Figure 13.3.3.17: Blue: back layer. Red: front layer.

Ensure that the courtyard lines (purple or light blue lines around the footprints) do not intersect with other lines and are not outside of the orange board outline. Courtyard lines indicate the border of a footprint.

Continue with the transistors Q1 and Q2. Always align and justify, and lock in place when finished.

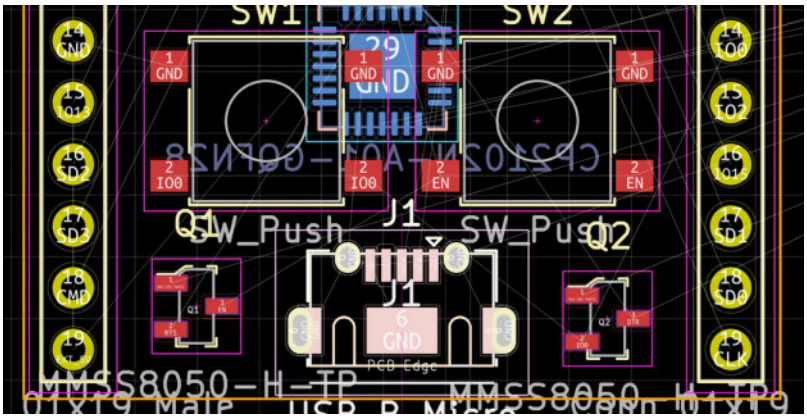


Figure 13.3.3.18: Placed Q1 and Q2.

Finally, continue to place the small footprints: LEDs, resistors, capacitors. These footprints will go in the free space between the buttons and the ESP32 module. To reduce clutter, disable the back layer visibility. Here is what my editor looked like before I begun work:

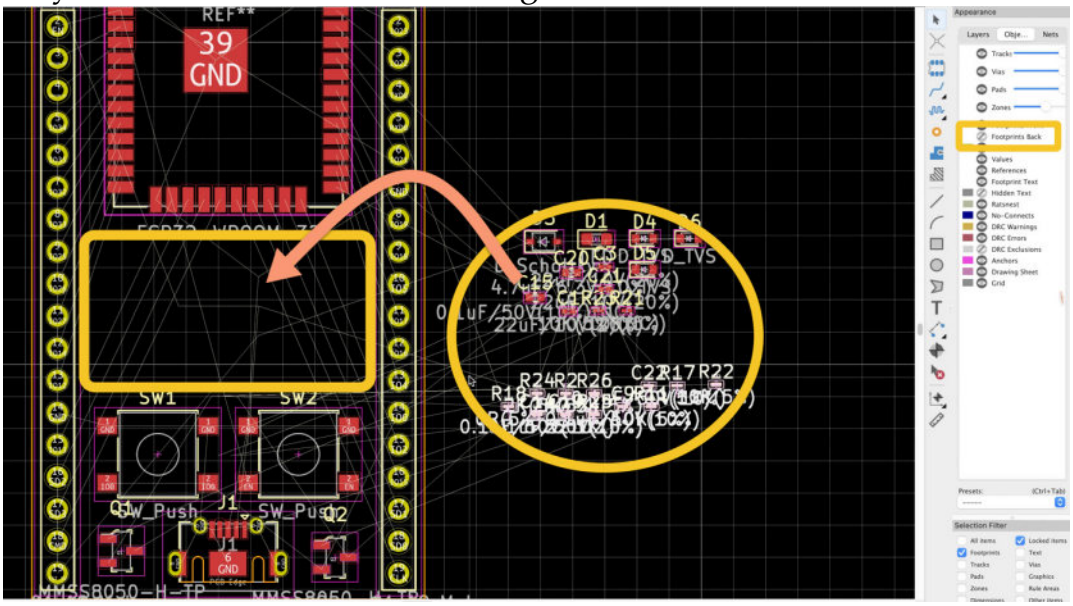


Figure 13.3.3.19: Preparing to place the small footprints.

During placement, observe these guidelines:

- Use ratnest lines and attempt to place footprints with pads near their closest connection.
- Place footprints in groups: resistors should be placed close to other resistors and capacitors close to other capacitors.

- Avoid mixing footprints of different types as this will increase the risk of errors in assembly.
- Justify and align frequently.
- Space out as much as possible so that you make use off all available space. This is very important if you are planning to assemble the board by hand.
- Take as much time as you need to produce a high-quality design. Fixing it later will be costly.

You can see my small footprint placement below:

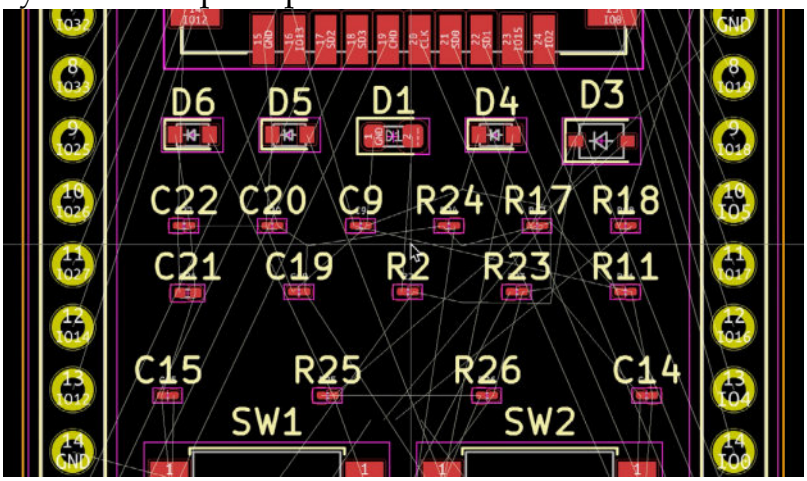


Figure 13.3.3.20: Completed placement of small footprints.

The footprint placement is now complete:

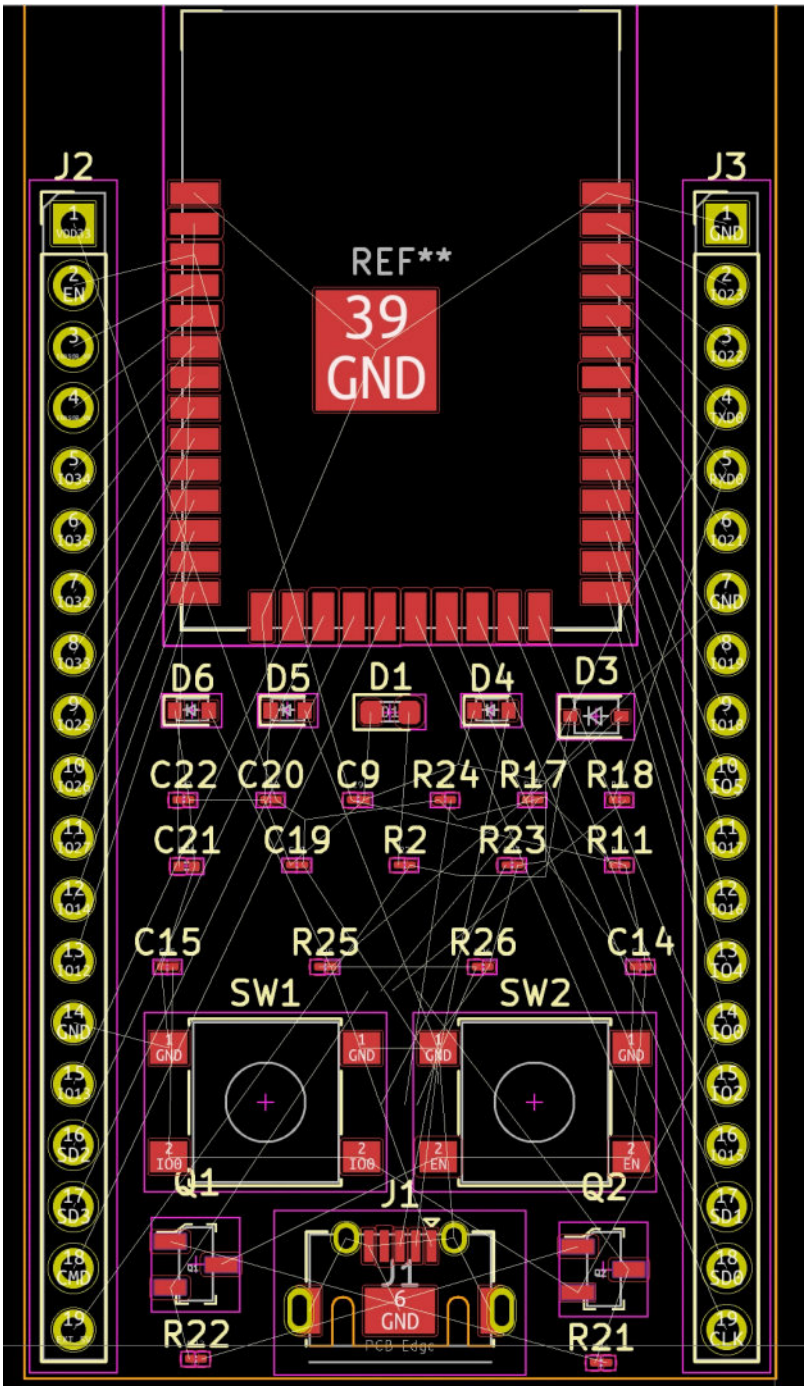


Figure 13.3.3.21: Showing final placement of all footprints in the front of the board.

In the next segment of this chapter, you will return to step two of the layout workflow and refine the board outline.

3.4. Layout 2 supplemental - refine outline

You have completed the placement of all footprints in the front and back of the board. You can now give your board its final refined shape. As you have done in previous projects, the refinement will consist of two changes to the rough outline:

1. Replace all 90-degree corners with aesthetically pleasing rounded corners.
2. Remove substrate material that is not necessary.

This is what the board looks like at the moment:

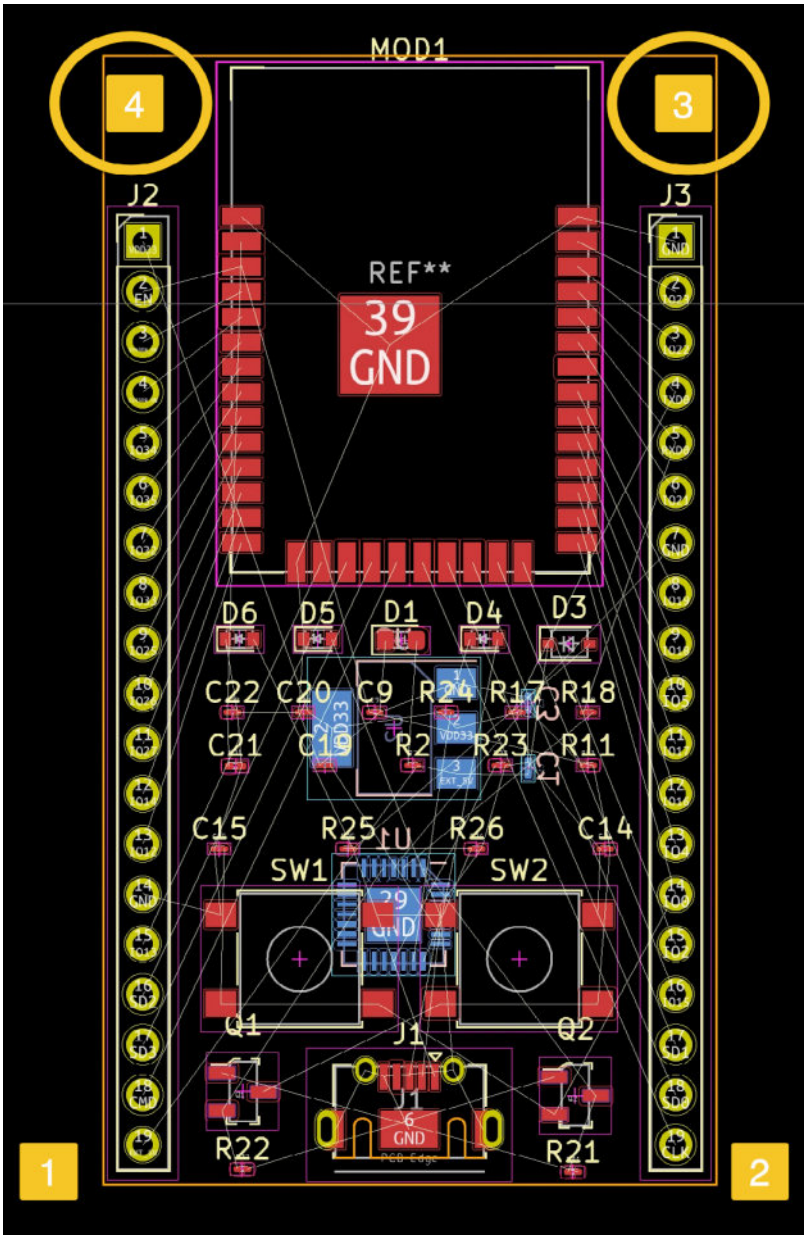


Figure 13.3.4.22: Plan to refine the rough outline.

In the figure above, I will replace the corners “1” and “2” with rounded corners. At the top of the board, there is un-utilised substrate material in corners “3” and “4”. I will remove those. I believe that removing this material will make the board look more modern and streamlined.

Use the process that you learned in the previous projects to do the refinement work.

As I worked to draw the refined outline, I made several errors that caused problems with the 3D viewer, and the DRC. I list some of those errors below.

Text labels

The DRC will complain when a text label in the silkscreen overlaps with other elements or is outside of the board outline. See example below, where the label “MOD1” is outside of the board and J3 is both outside of the board and overlaps with other elements.

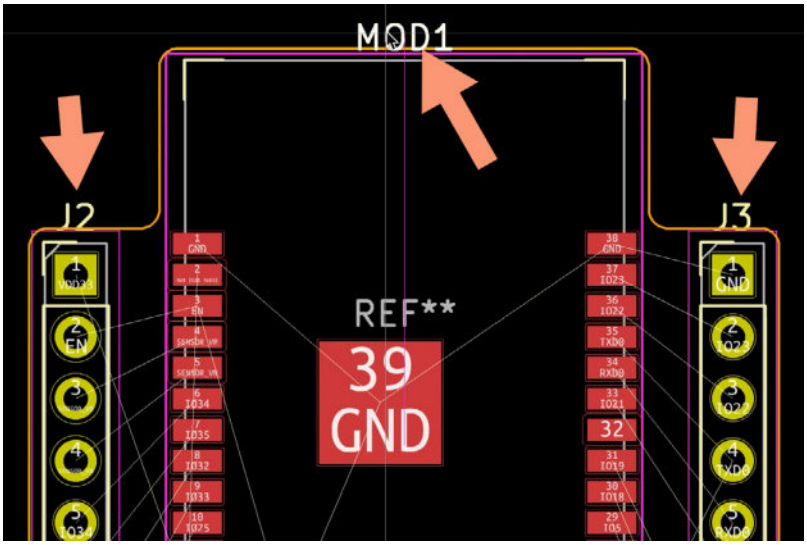


Figure 13.3.4.23: Text label bugs.

To fix such errors, move the labels inside the board outline and ensure that they don’t overlap with other elements (this was my choice for MOD1). You can choose to make them invisible (this was my choice for J2 and J3).

Malformed courtyard

Courtyard lines should form a closed polygon (same applies to lines in the Edge.Cuts layer). In the example below, the courtyard line from the ESP32 module footprint is malformed.

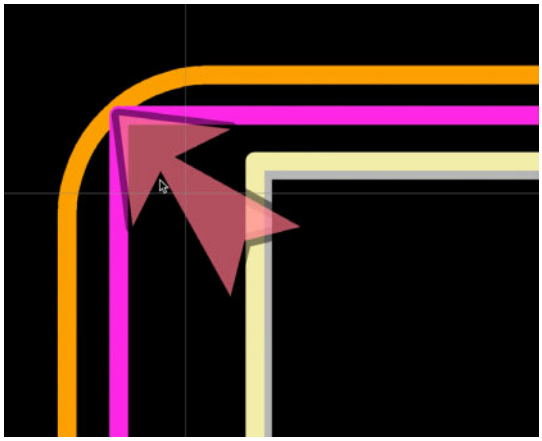


Figure 13.3.4.24: Courtyard line bug.

To fix this, you have two choices:

1. Delete the footprint's courtyard (my choice).
2. Redraw the courtyard to ensure it forms a closed polygon.

I chose option one as it is easier and faster. I am mindful that the DRC will not be able to use the courtyard to let me know of overlap violations. But, as I have already placed all footprints in the board, I don't expect any related problems.

To delete or redraw the courtyard lines, you will need to use the footprint editor.

Edge.Cut errors in footprints

Footprints may contain errors. This is why it is important to be able to recognise them, and fix them. In this project, the footprint I used for the USB port contains an open polygon drawn in the Edge.Cuts layer (see below).

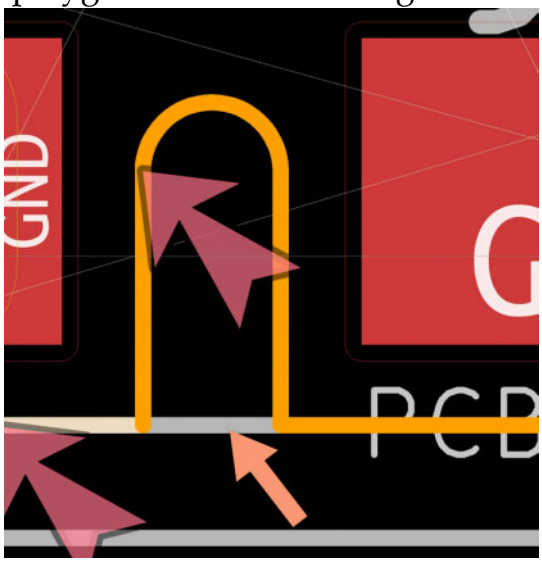


Figure 13.3.4.25: Open polygon graphic in the Edge.Cuts.

The 3D viewer will show an error if it detects an open polygon in the Edge.Cuts layer. Your PCB manufacturer may also have trouble dealing with this, and may ask you to fix the layout and re-submit the Gerbers.

To fix such errors you will need to edit the footprint in the footprint editor. I chose to simply delete the original drawings in the USB connector's footprint.

Clearance violation

In the process of using the DRC to help me fix drawing bugs, I found that there is a clearance violation relating to the pads of the small capacitor and resistor footprints. These footprints belong to the Default net class, and their minimum clearance is set to 0.2 mm.

You can see some of the violations below:

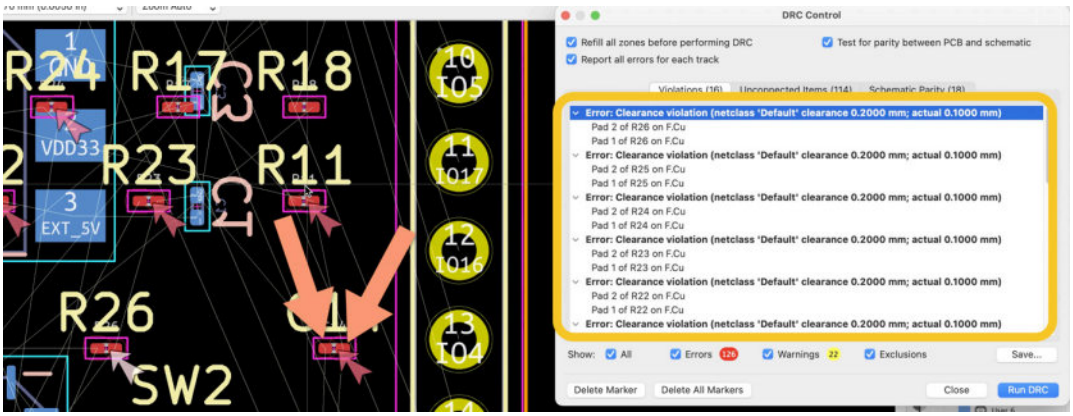


Figure 13.3.4.26: Pad distance violates minimum clearance.

I use two arrows to point to the pads of C14. Those pads are 0.1 mm apart, when the minimum is 0.2 mm. To fix this error, you have a few choices:

1. Change the minimum clearance for the Default class to 0.1 mm.
2. Make the nets to which the affected pads belong members to the GPIO net class.
3. Create a new net class with minimum clearance 0.1 mm, and make the affect pads members of the new class.

I decided to go with the third option, and created a new net class with name "Data".

- Data.
  - Clearance: 0.1 mm
  - Track Width: 0.1 mm
  - Via Size: 0.5 mm
  - Via Hole: 0.35 mm
  - uVia Size: 0.25 mm

- uVia Hole: 0.1 mm
- DP Width: 0.1 mm
- DP Gap: 0.15 mm

Assign these nets to the Data net class:

- CDC
- CLK
- CMD
- CTS
- DSR
- DTR
- EN
- All un-named nets, like “Net-(…)”

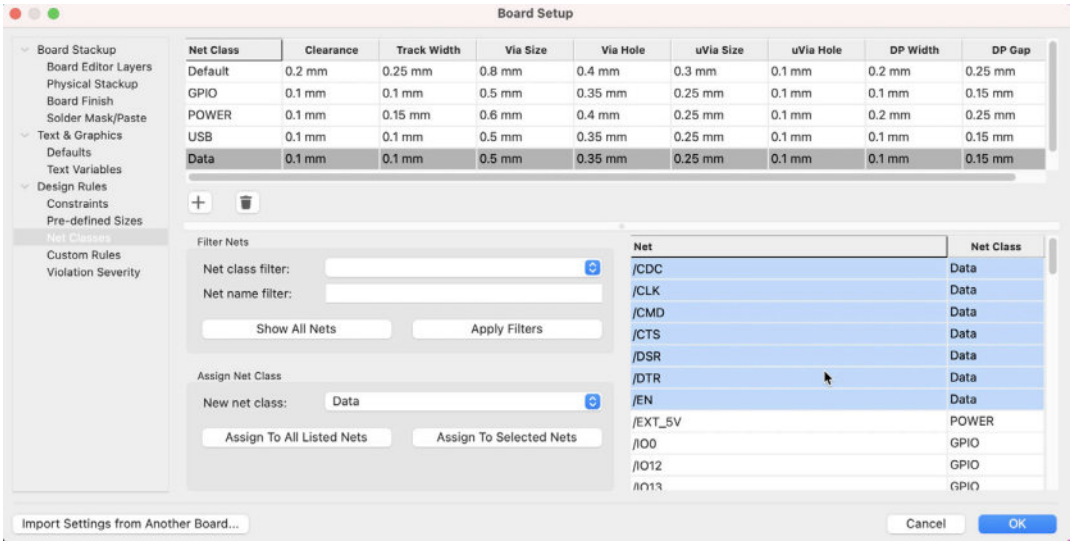


Figure 13.3.4.27: Setting up the Data net class.

Run another DRC to ensure that there are no outstanding violations relating to the Edge.Cuts, courtyard, or clearance issue. Below you can see my board’s refined outline:

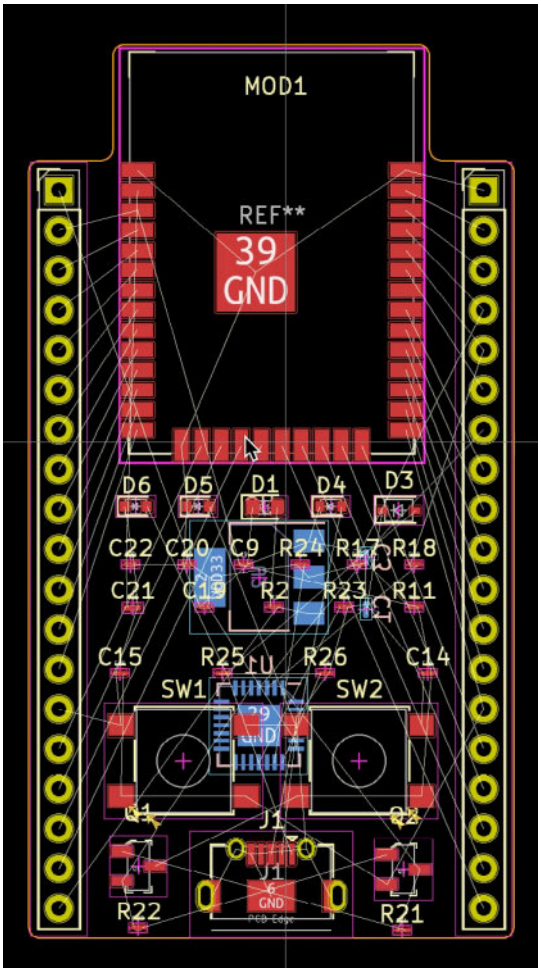


Figure 13.3.4.28: Board refined outline.

And, the board in 3D:

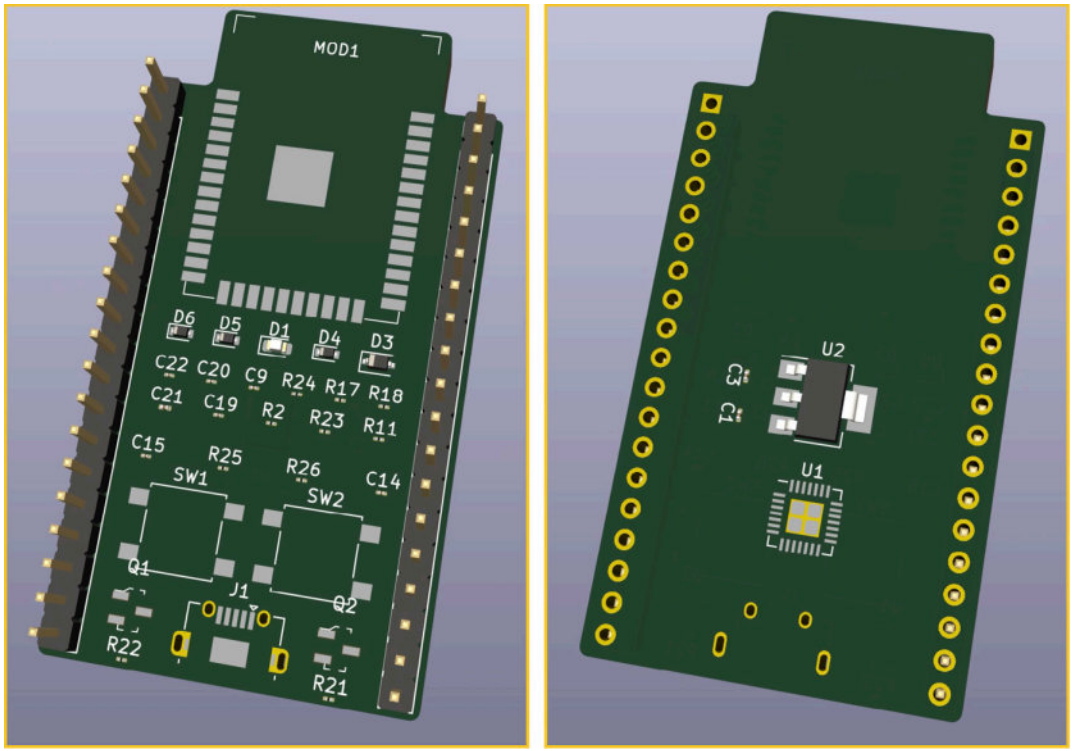


Figure 13.3.4.29: The refined board in 3D.

Let's continue with the routing step.

3.5. Layout 4 - Route

This board is not trivial. It contains a large number of footprints and pads arranged in a relatively small board. The utilisation of four layers will make routing easier, however it will still require significant effort and time.

It took me more than an hour to complete the routing, after I already had practiced in previous iterations. Before you begin, please ensure that you have enough time to complete this step. Follow the general guidelines I listed in the previous project. For the duration of the routing process, I used the interactive router set to "Walk around" mode and "Fix all segments on click" enabled and "Optimise entire track being dragged" disabled.

Generally, I begin work with routing tracks between nearby pads, starting from the central component. For example, below you can see my first three tracks in the front copper layer, connecting the ESP32 GND pads (1, 38 and 39), and the adjacent EN pads (MOD1 pad 3 and J2 pad2):

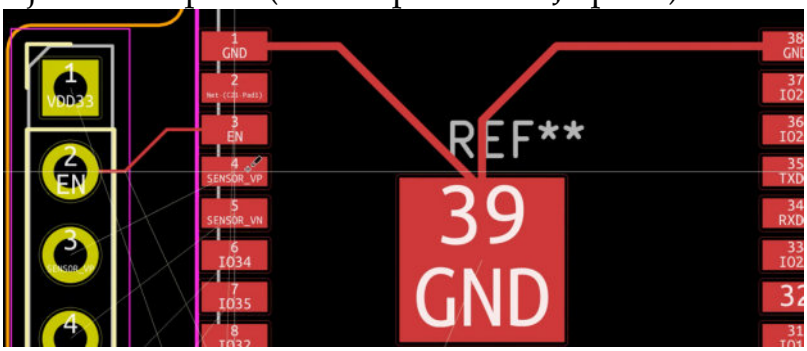


Figure 13.3.5.30: Drawn the first three tracks.

This board contains a differential pair that consist of the SENSOR_VP and SENSOR_VN nets:

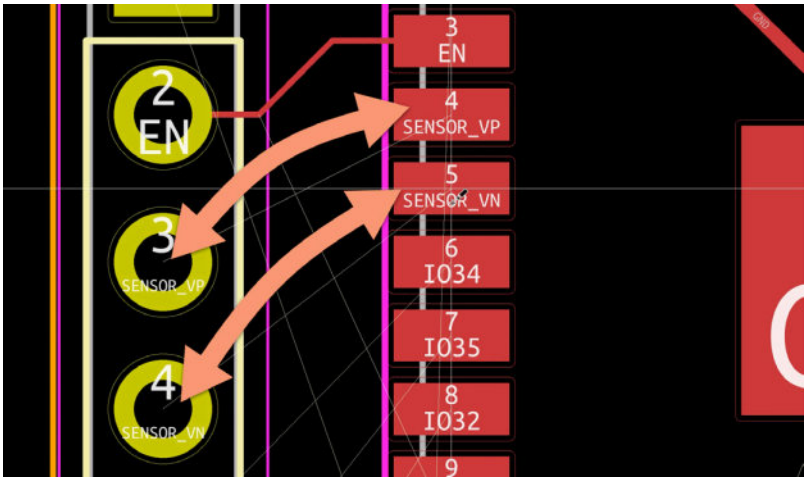


Figure 13.3.5.31: Draw track using differential pair routing.

If you are not familiar with the technique of drawing differential pair routes, please read the relevant chapter in the Recipes part of this book. To draw these tracks as differential pairs, selected the differential pairs tool from the right toolbar, click on one of the pair pads, like 4 in MOD1, and draw towards one of the differential pair pads in J2, like 3.

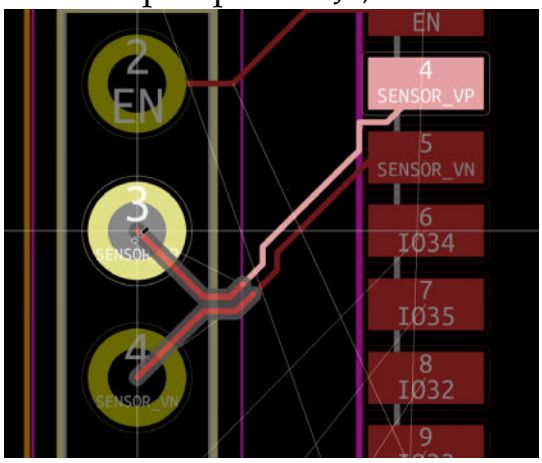


Figure 13.3.5.32: Differential pair routing.

Don't worry if the tracks of thee pair are not perfect, you can improve them later.

Continue with the rest of the pads that belong to MOD1, and then continue with the USB interface and power components.

Here's some guideline reminders:

- Try to arrange tracks close to each other to avoid wasted space.
- Avoid 90-degree angles.
- Try to draw tracks in the top layer since the components are surface-mounted. Use vias to access other layers if needed.

- Use the Shift-< hotkey to open the via menu and select the layer where you want to continue drawing.

In the example below, I have opened the layer selection window to quickly create a via to a new layer. The hotkey for this window is Shift-<:

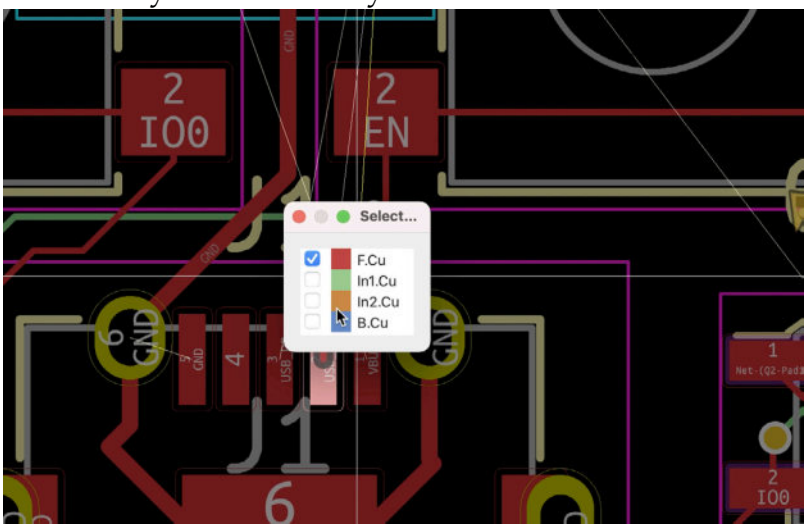


Figure 13.3.5.33: Quickly switching between layers.

To help you with the routing, I provide the following figures that show the routes in each of the four layers:

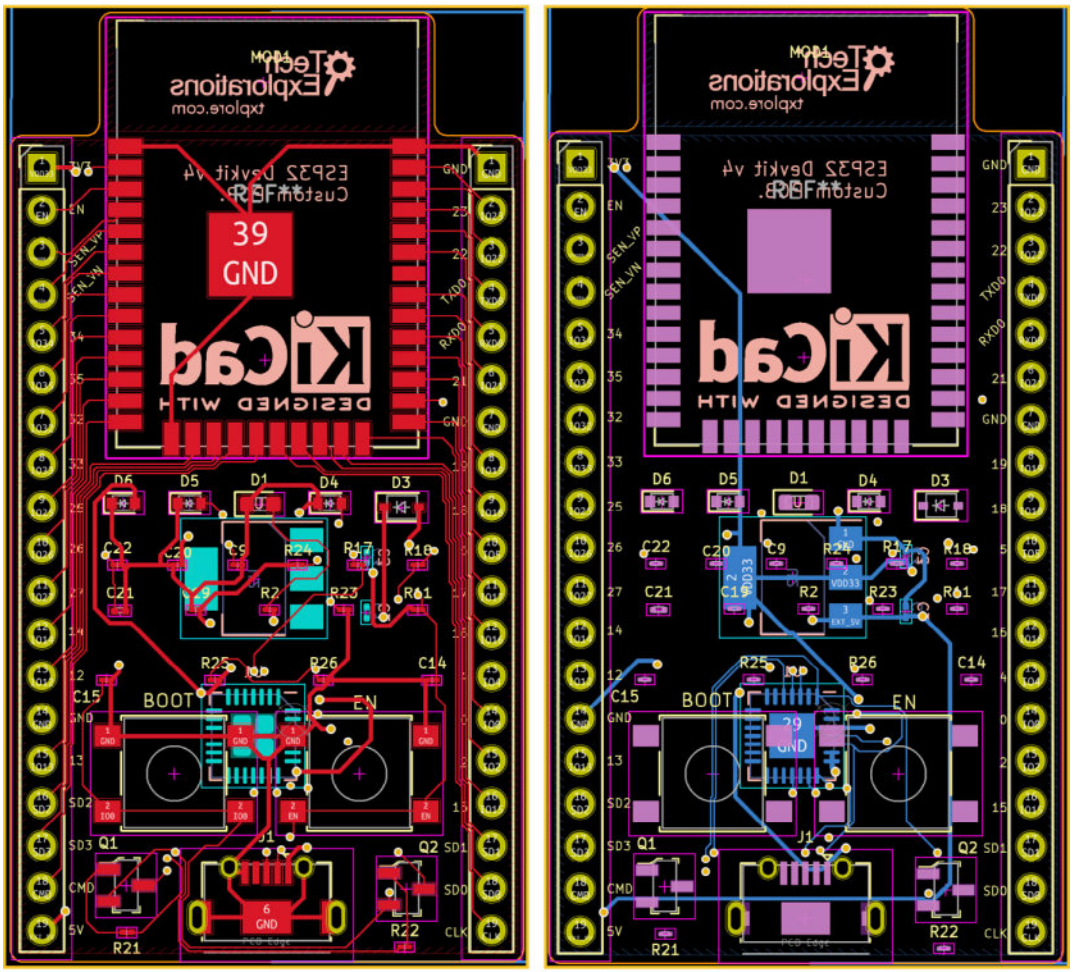


Figure 13.3.5.34: Completed routing, F.Cu and B.Cu.

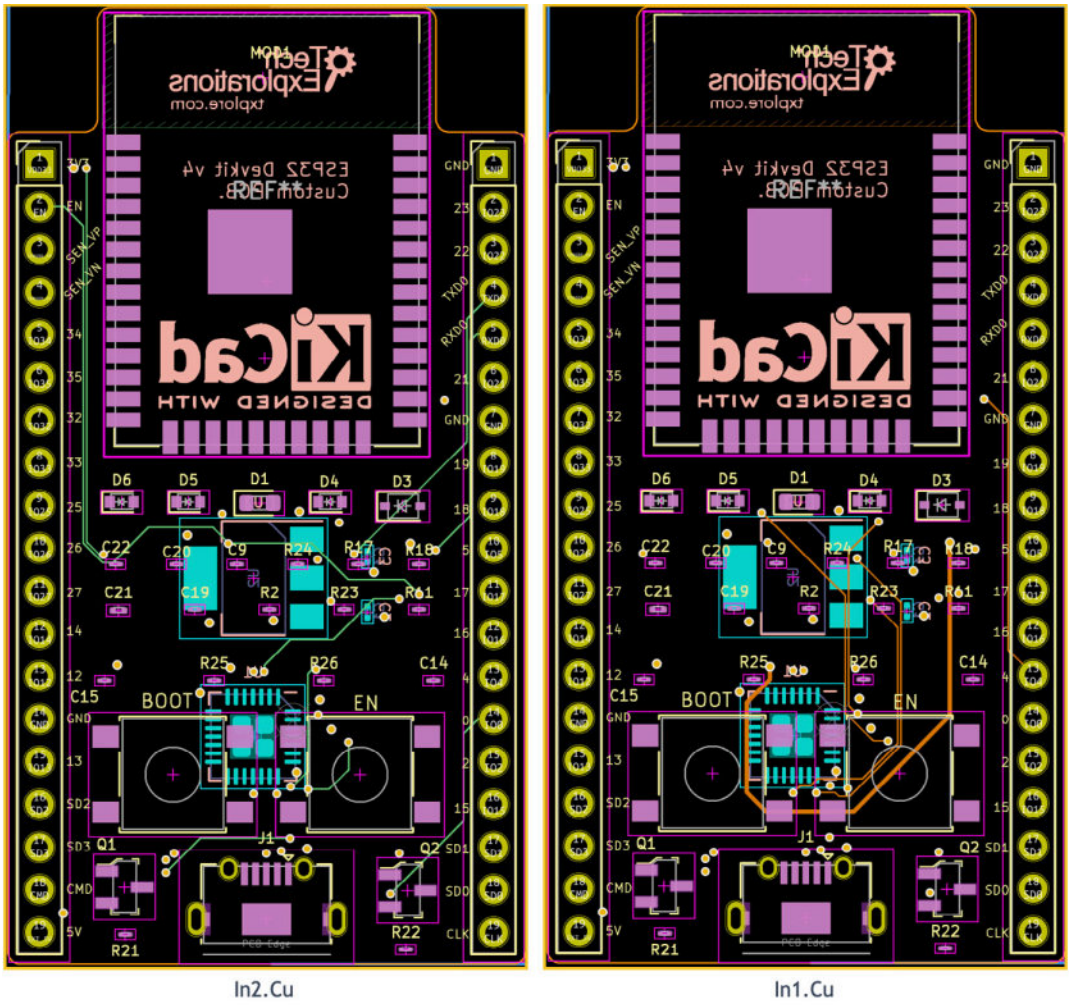


Figure 13.3.5.35: Completed routing, In1.Cu and In2.Cu.

Below you can see my layout at this point. All the routing is complete, except for several GND connections which I will do using copper fills in the next segment of this chapter.

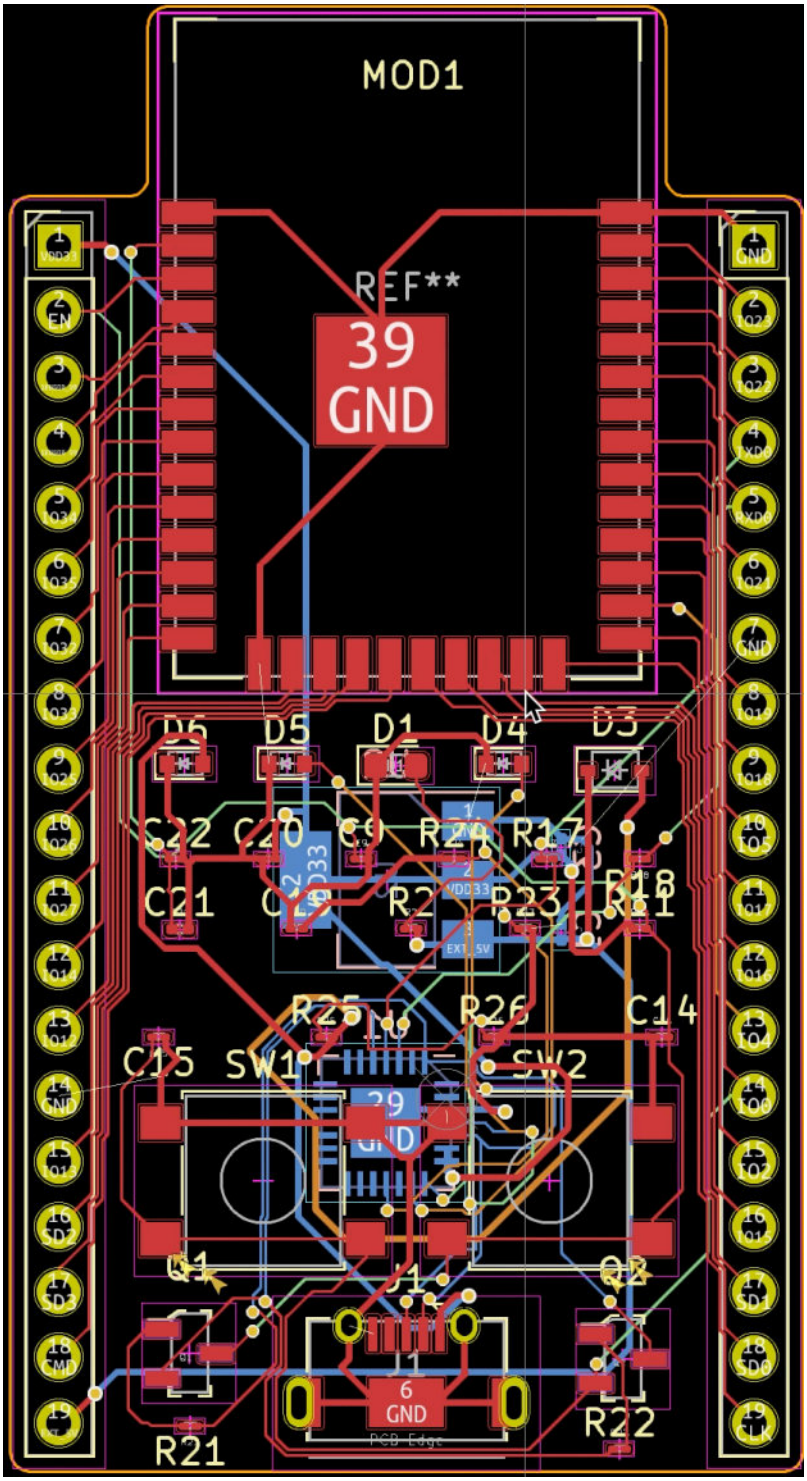


Figure 13.3.5.36: Almost completed routing, pending the copper fills.

Let's finish the routing step in the next segment, where you will draw the copper fills for the GND and VDD33 nets. You will also draw a keep out area for the ESP32 module antenna.

3.6. Layout 4 - Copper fills and keep out areas

Let's finish work in step four of the layout. In this segment, you will complete two tasks:

1. Create a keep-out area to protect the ESP32 module antenna from accidental placement of other footprints.
2. Create a copper fill in the bottom copper layer, and connect it to the GND net. This will also complete any pending GND routes.
3. Create a copper fill in the top copper layer, and connect it to the VDD33 net. This will also complete any pending VDD33 routes.

As I was working on this lecture, I realised that the keep-out area for the ESP32 module antenna is something that I should have done earlier in the workflow. Nevertheless, it is not late to do it now. If you are not familiar with keep-out areas (or "keep-out zones"), please review the relevant chapter in the Recipes part of this book. Select the keep-out zone tool from the right toolbar and click on one of the antenna area corners to begin drawing.

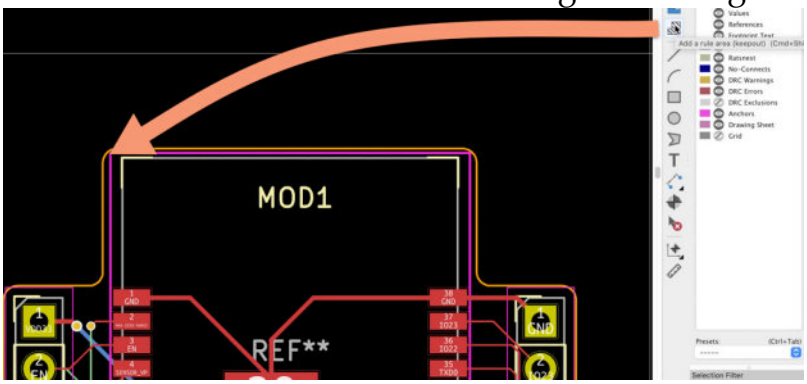


Figure 13.3.6.37: Drawing a keep-out zone.

In the Rule Area Properties window that appears, select all layers (to keep out object from all selected areas), and give the zone a name:

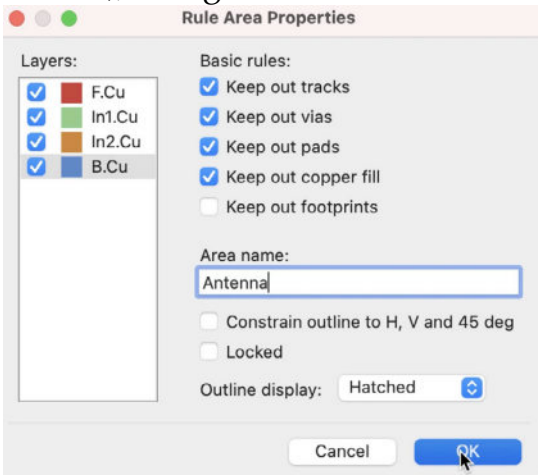


Figure 13.3.6.38: Keep out objects from all layers.

Draw the area so that it looks like this:

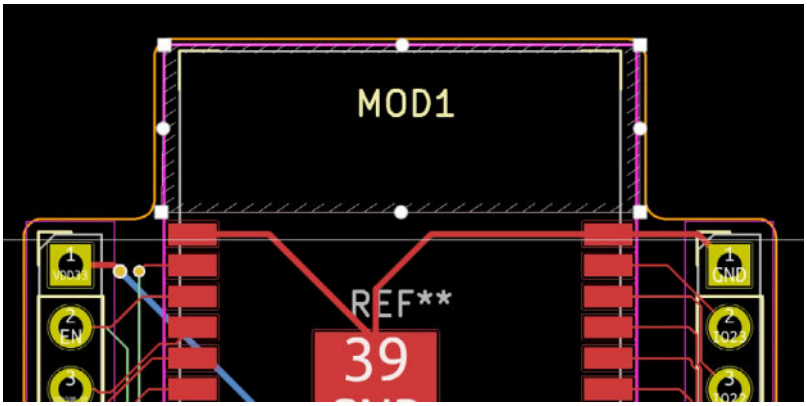


Figure 13.3.6.39: Keep-out zone complete.

Continue to create a copper filled area in the bottom layer, and connect it to the GND net. Here are my setting for this area:

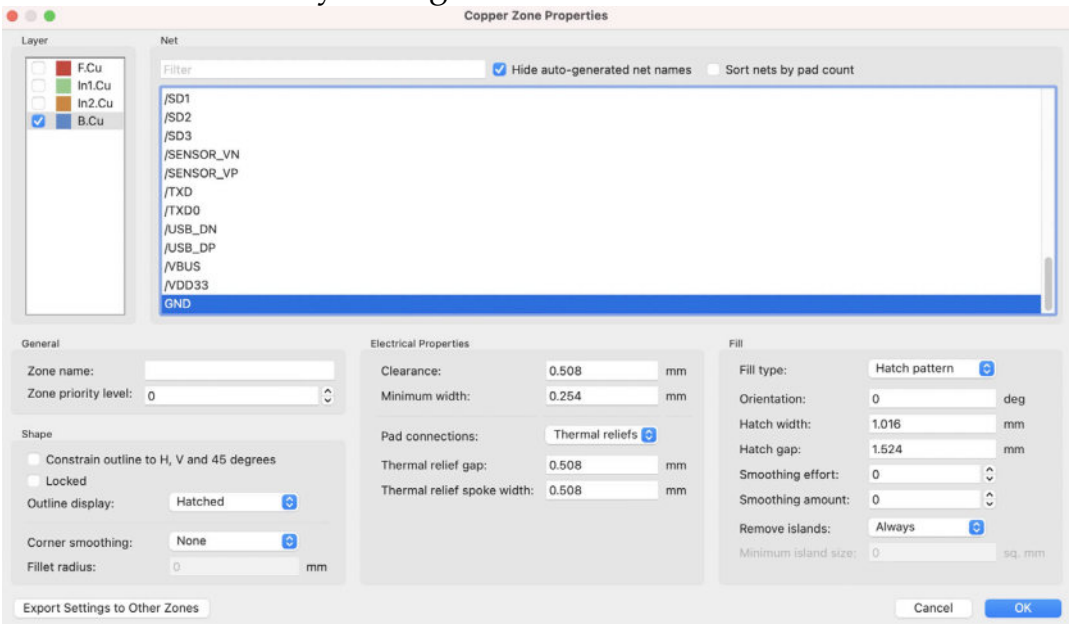


Figure 13.3.6.40: B.Cu copper zone properties.

Finally, create a copper layer in the top layer, and connect it to the VDD33 net:

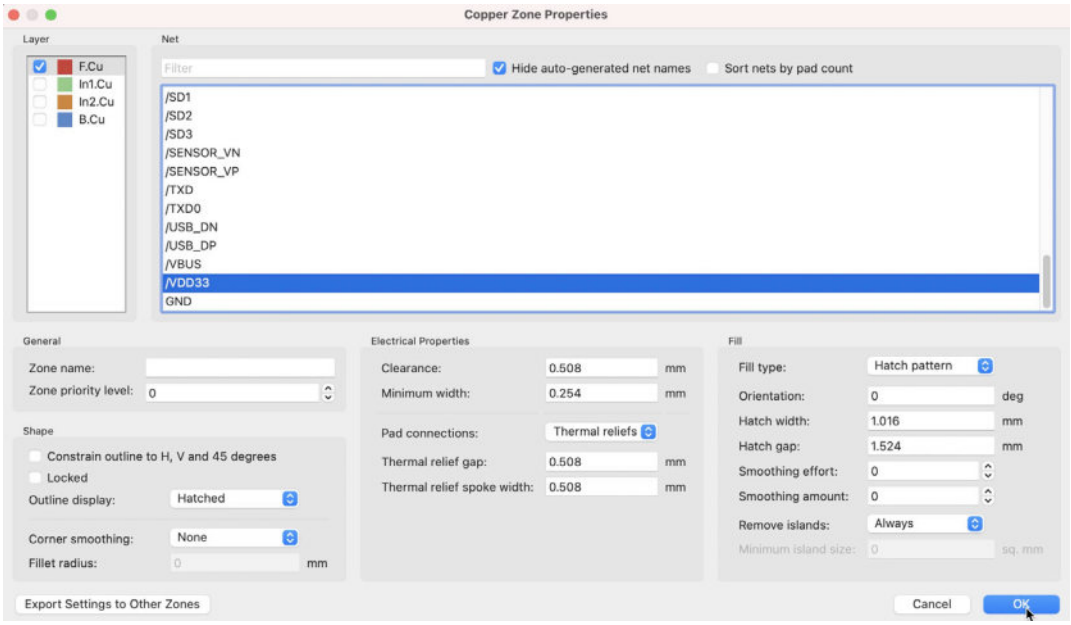


Figure 13.3.6.41: F.Cu copper zone properties.

When you finish drawing the copper fills, fill them and do a DRC. In my instance, the DRC found several unconnected GND and VDD33 pads. Here is an example:

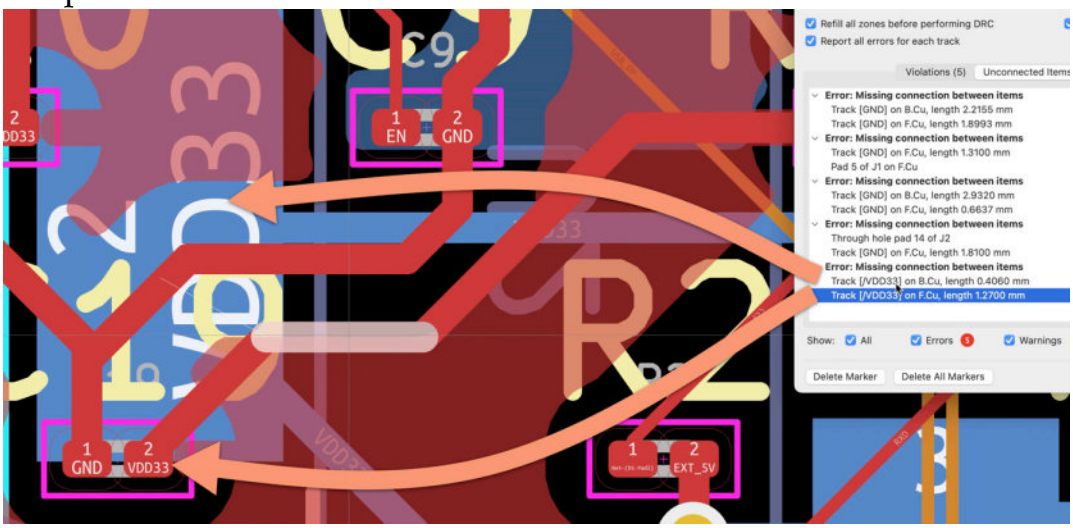


Figure 13.3.6.42: An unconnected VDD33 pad.

The two VDD33 pads, one in the bottom layer, and one in the top, are unconnected after the copper pours. There are similar instances of other VDD33 and GND pads with the same problem.

To fix such bugs, you will need to manually draw tracks and vias. Use the list of violations in the DRC window to guide you, and disable the copper fills so you have better visibility.

The violation in figure 13.3.6.42 (above) is easy to fix with a via and two copper track segments:

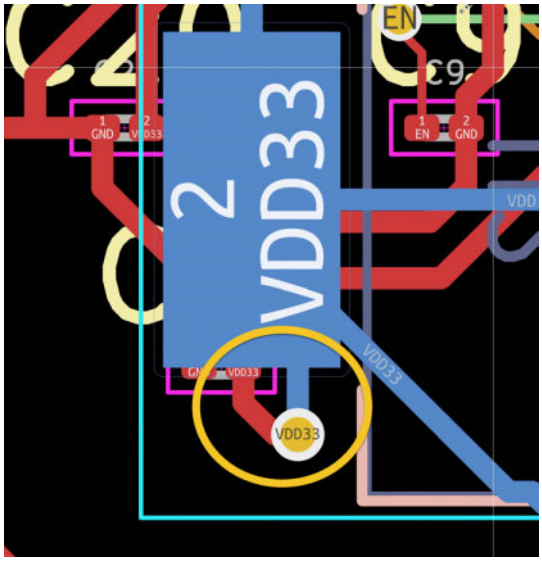


Figure 13.3.6.43: Fixed an “unconnected pad” violation.

Continue in the same manner to fix the rest of the violations. Here is my completed layout, fully routed and filled:

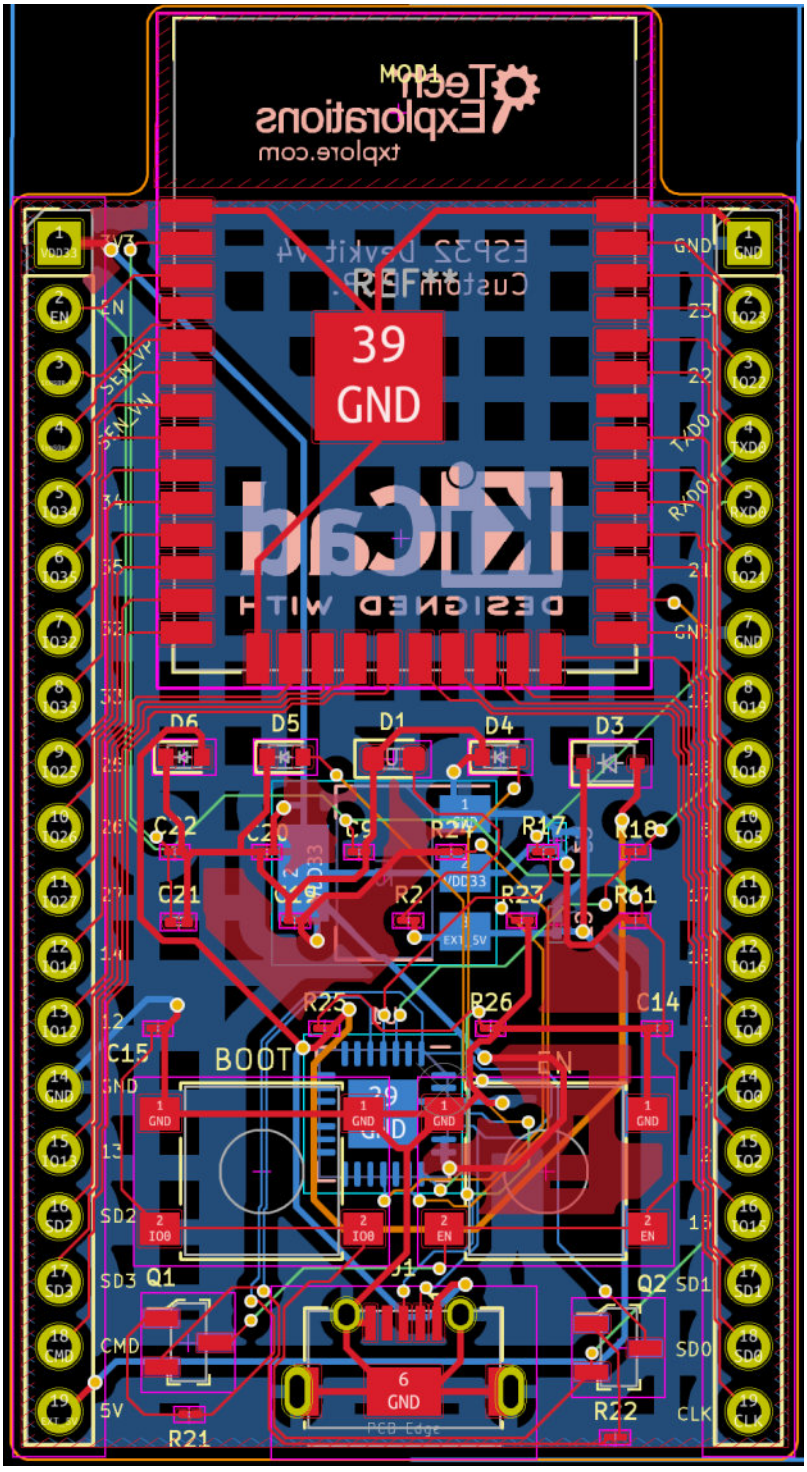


Figure 13.3.6.44: The project PCB, fully routed.

Step four of the workflow is now complete. LEt's continue with the silkscreen.

3.7. Layout 5 - Silkscreen

In this step of the workflow, you will work on text and graphical elements that will exist in the top and bottom side of the board. Here is a list of things to do:

1. Optimise the size and location of text labels, like “D6” and “R18”.
2. Add text labels for the pin headers.
3. Add text labels for the buttons.
4. Add logos in the back of the PCB.
5. Add board information, such as the name, and version, in the back layer.
6. Fix silkscreen-related violations reported by the DRC.

Begin by reducing the size of footprint reference text labels. From the Edit menu, choose “Edit Text and Graphic Properties”. Select “Reference designators” in the Scope, and set 0.1 mm for the line thickness, and 0.7 mm for the text width and height. Click OK, and confirm that the new text size for the values is a better match for the size of the board.

One by one, reposition those text items so that they are near their corresponding footprint, and they don’t overlap. To make this work easier, use the Edit Text and Graphic Properties window to make all items in the F.Fab layer invisible.

Below you can see the text in the front silkscreen:

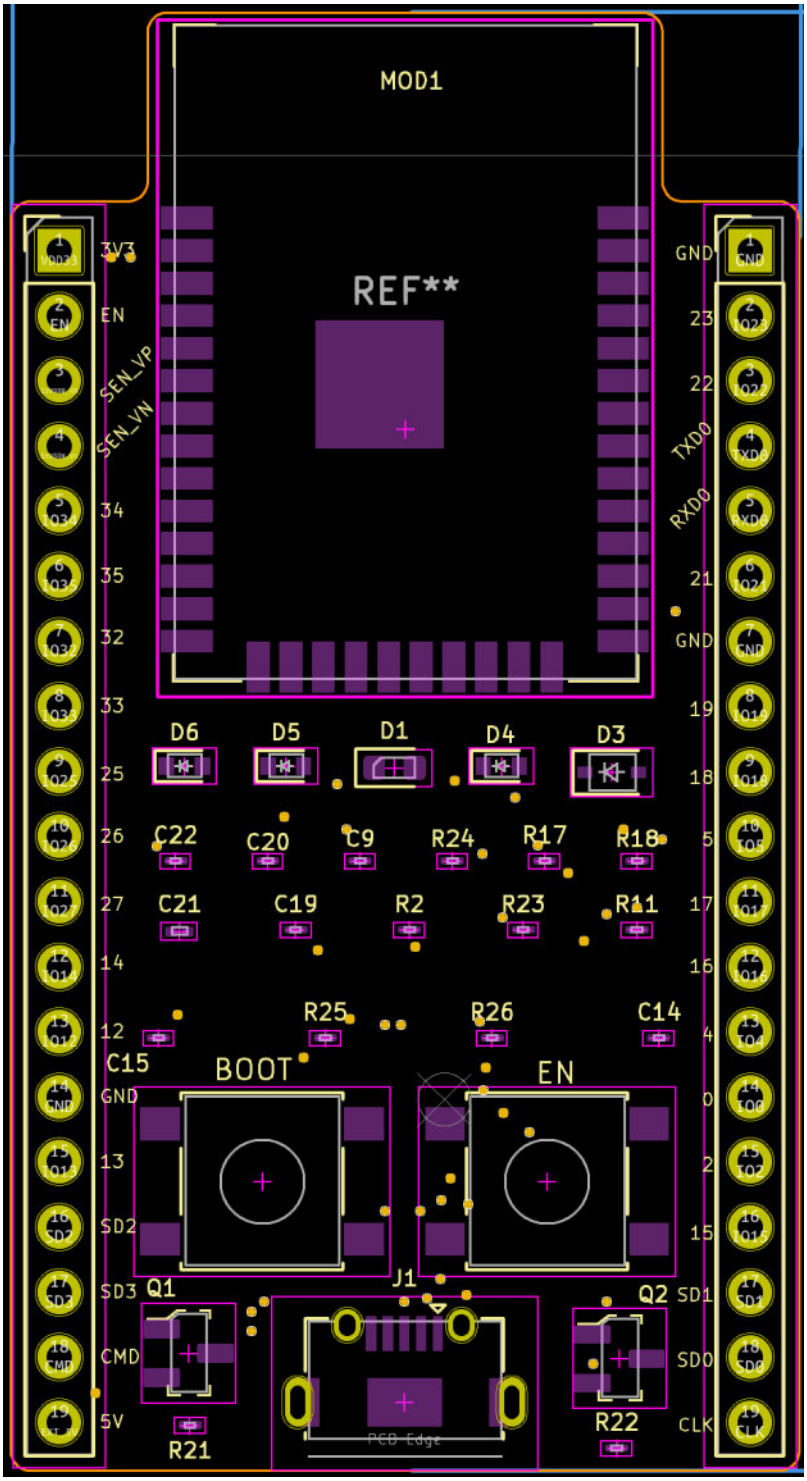


Figure 13.3.7.45: The contents for the front silkscreen layer.

And here are the contents of the bottom silkscreen layer:

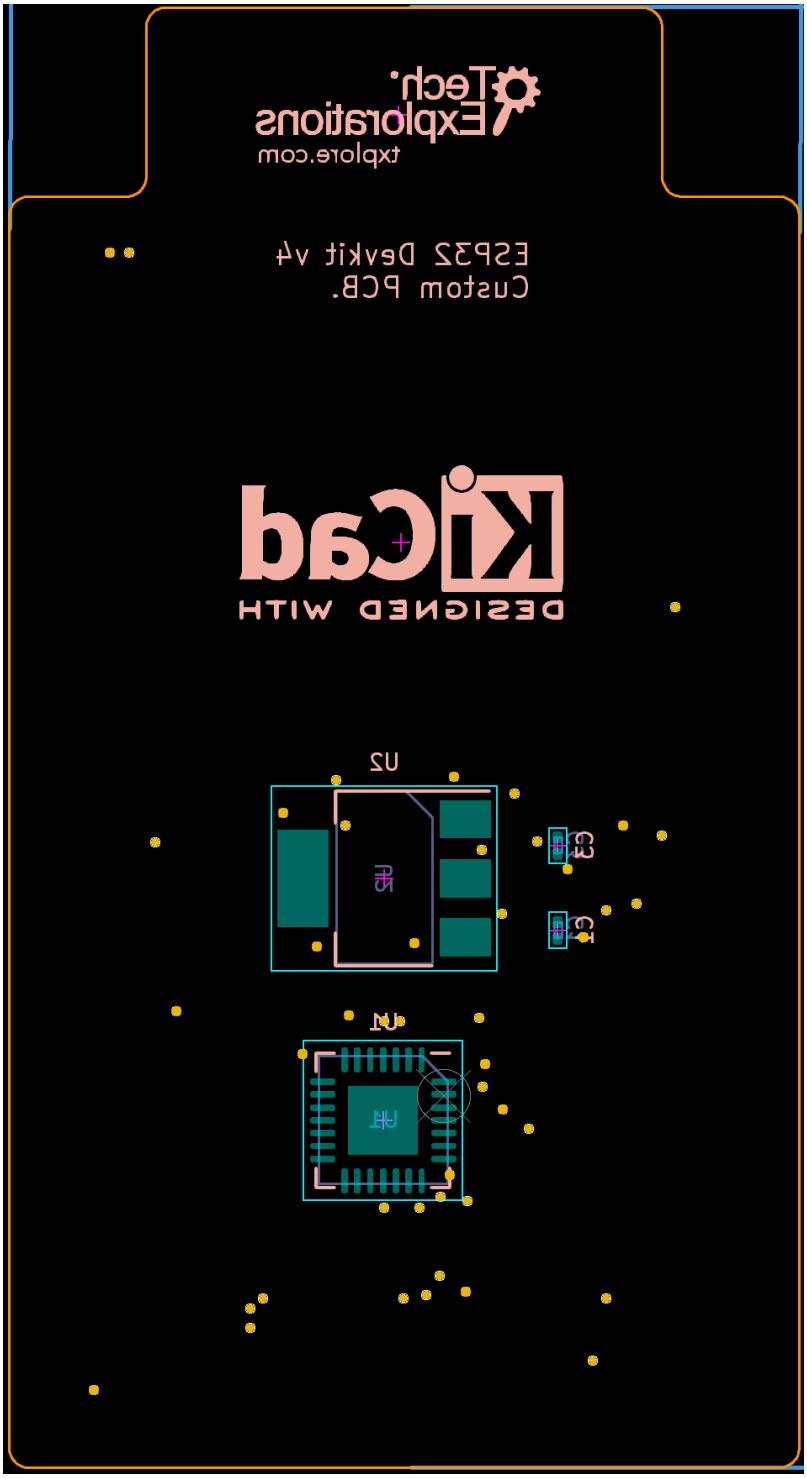


Figure 13.3.7.46: The contents for the back silkscreen layer.

I think that the contents of the silkscreen, back and front, as you see it above, is sufficient.

Before doing the final DRC, I want to do another round of improvements with the routing, and fix a few small issues that I noticed during the work in the silkscreen.

3.8. Layout 4 - Routing improvements

This project is close to completion. In this segment, I want to make a few final improvements in the copper tracks.

For example, I noticed that the gap between those two track in one of the inner layers is too wide:

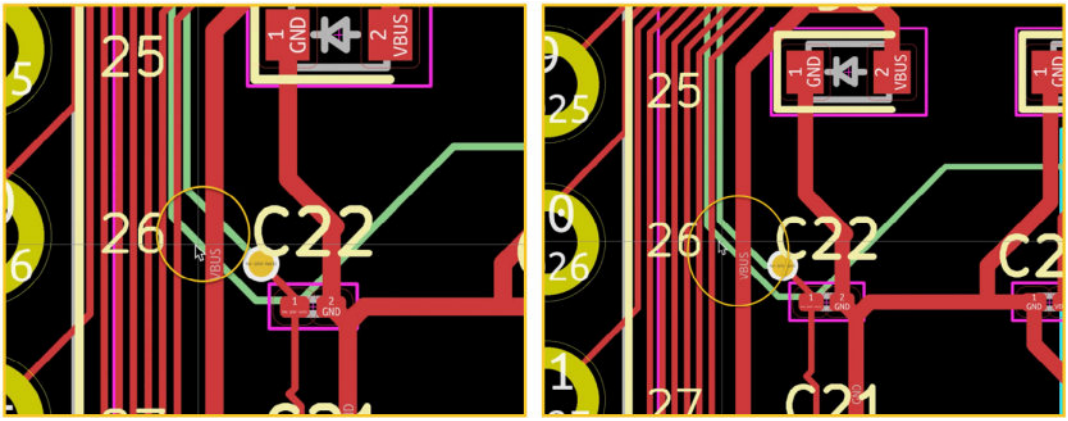


Figure 13.3.8.47: Minimising gaps.

Large gaps can waste valuable space on a PCB. This waste can be important when you need to add elements such as a via to complete a connection.

Here is another example:

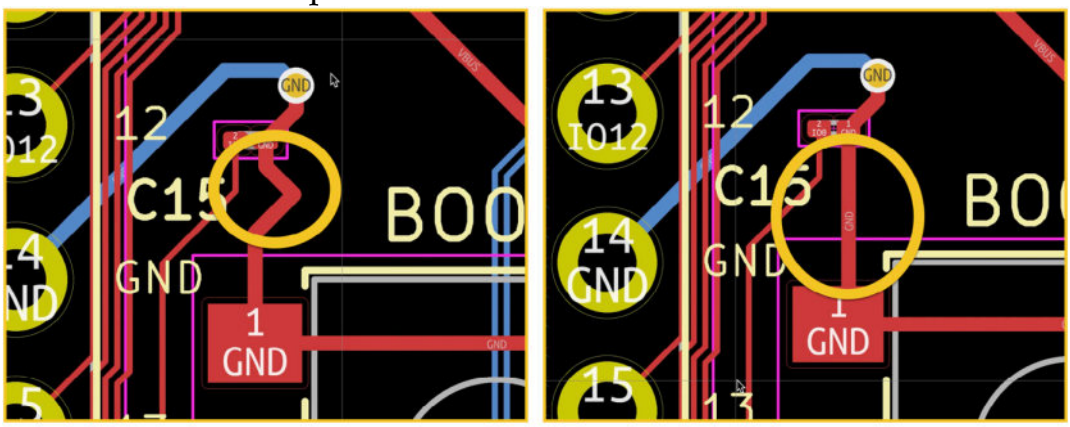


Figure 13.3.8.48: Opt for straight tracks.

The track in the left of the figure above had three bends, even though it connected two near-by pads. Just use a straight track like in the example in the right.

A final example, where I have replaced a 90-degree corner with 45-degree ones:

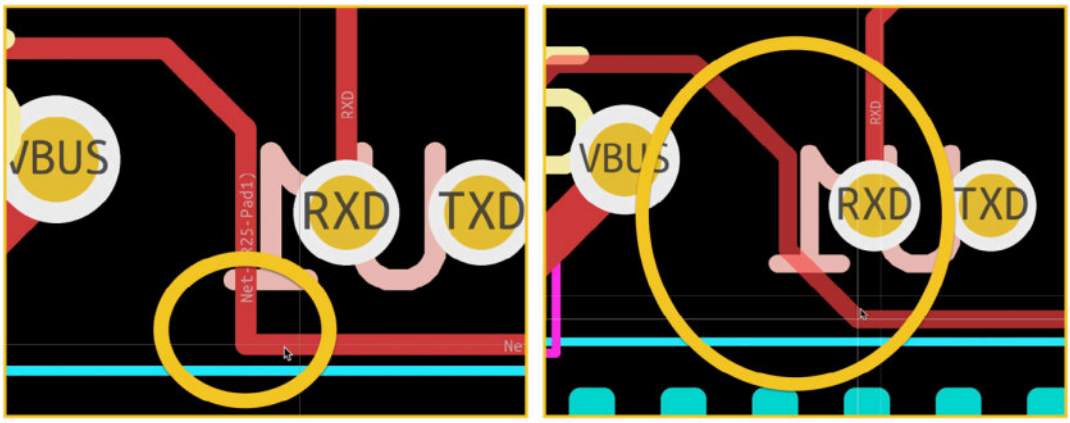


Figure 13.3.8.49: Replace 90-degree corners with 45-degree ones.

Take your time to look for other improvements, and continue to the last DRC in the next step.

3.9. Layout 6 - Design Rules Check

Time for the final design rules check.

My check only shows one violation and twenty schematic parity warnings. The violation is a warning about a silkscreen overlap. You can see this below:

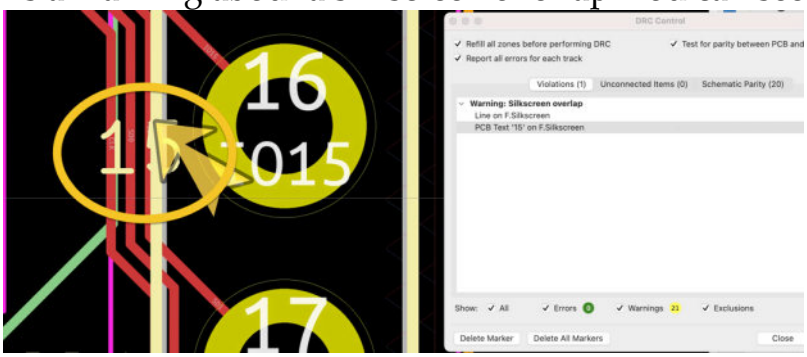


Figure 13.3.9.50: A “silkscreen overlap” warning.

This is easy to fix by moving the text label “15” off the line.

The schematic parity warnings have to do with pads that are missing a net. These warnings are generated for pins that are marked as “unconnected” in the schematic, and they are safe to ignore.

With the last warning fixed, the layout is ready to export as Gerber files, and upload to the manufacturer.

3.10. Layout 7 - Manufacture

The layout is complete and tested. In this segment you will export the Gerber files, check them for defects, and upload them to the manufacturer’s website.

The process is similar to that of the previous projects, with one important difference. In this project, the clearances are smaller than those in previous projects. You must be careful to select the appropriate clearances in the manufacturer's order form to be at least equal or smaller than the clearances in the layout.

Below I provide my settings for the Gerber file export:

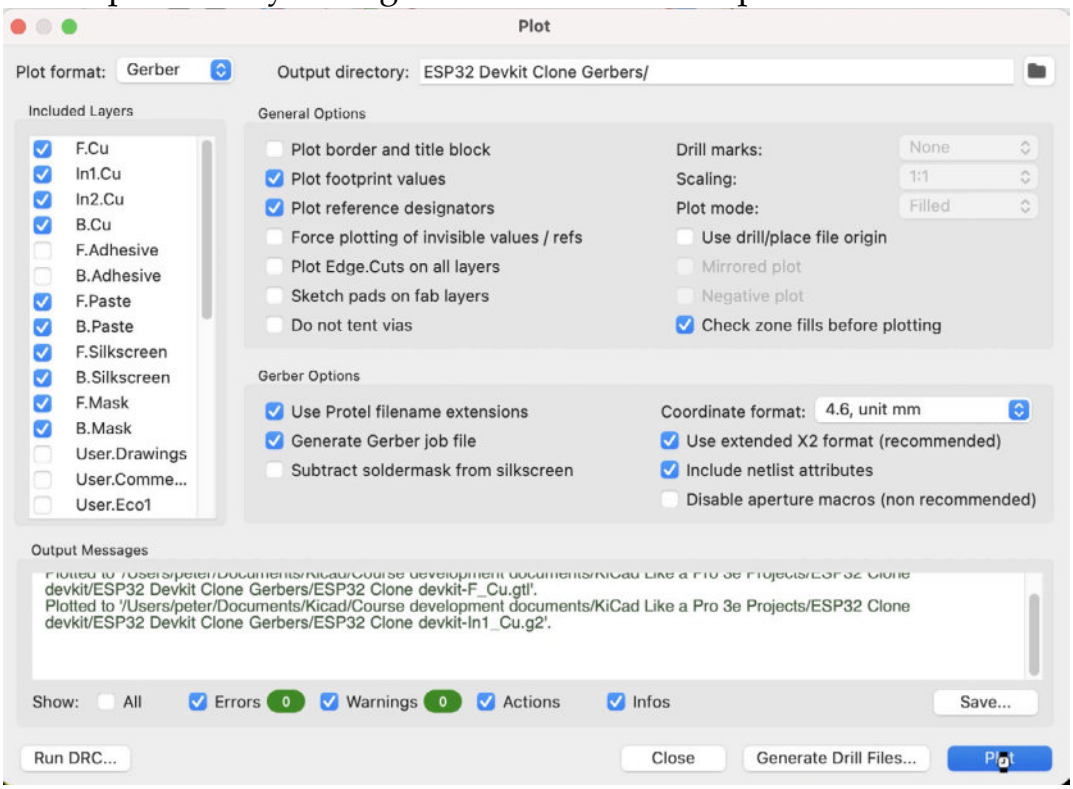


Figure 13.3.10.51: Gerber file export settings.

And the drill files export settings:

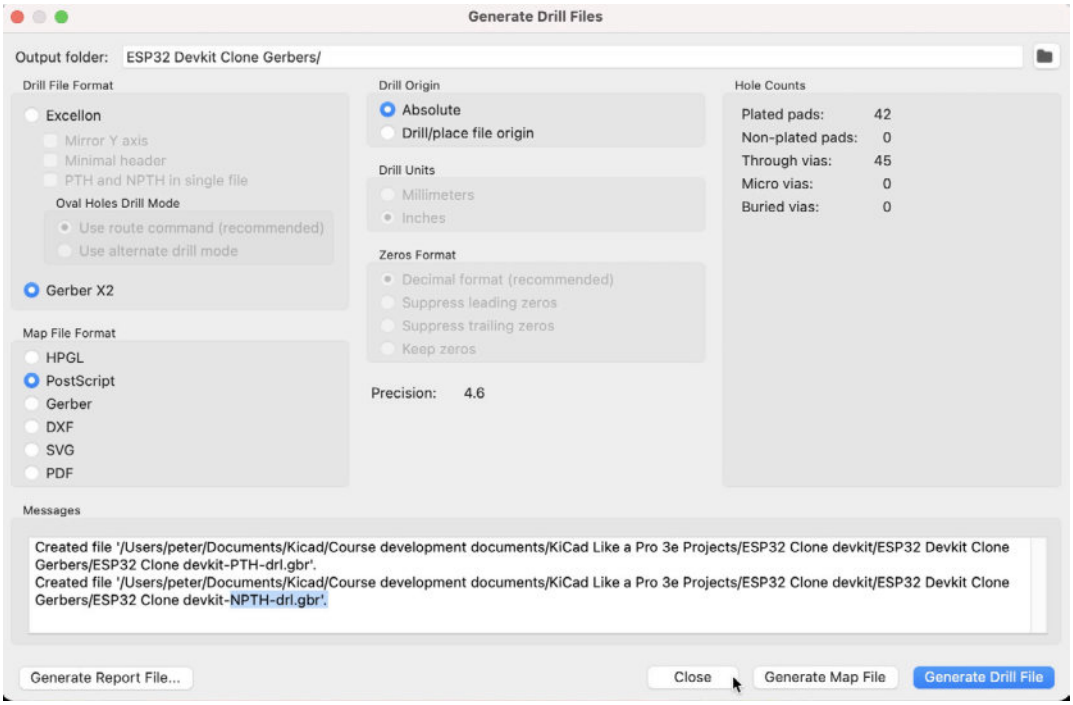


Figure 13.3.10.52: Gerber file export settings for the drills.

Generate the measurements for the board so that you can provide them to the manufacturer's order form:

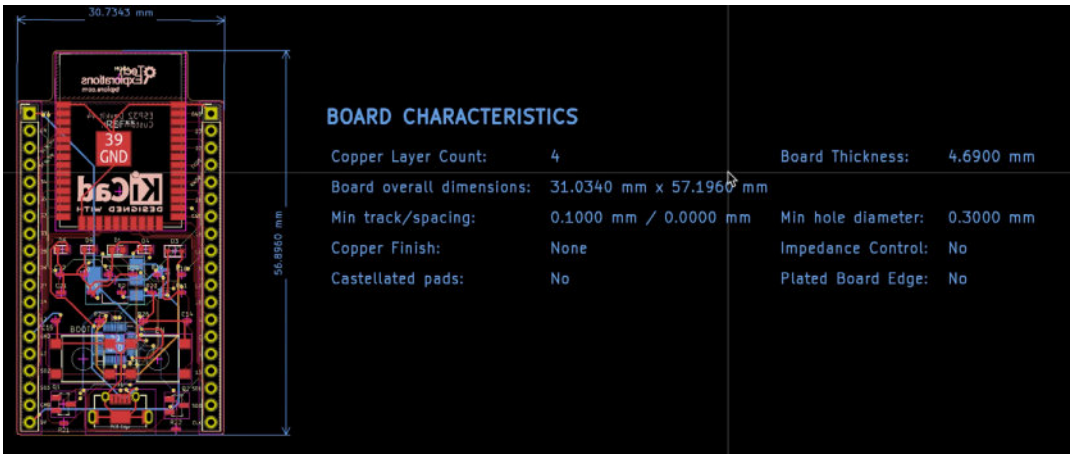


Figure 13.3.10.53: Board size and characteristics.

To add the board characteristics to the editor, choose "Add Board Characteristics" under the Place menu, and select one of the User layers.

Use the Gerber viewer to validate the Gerber files:

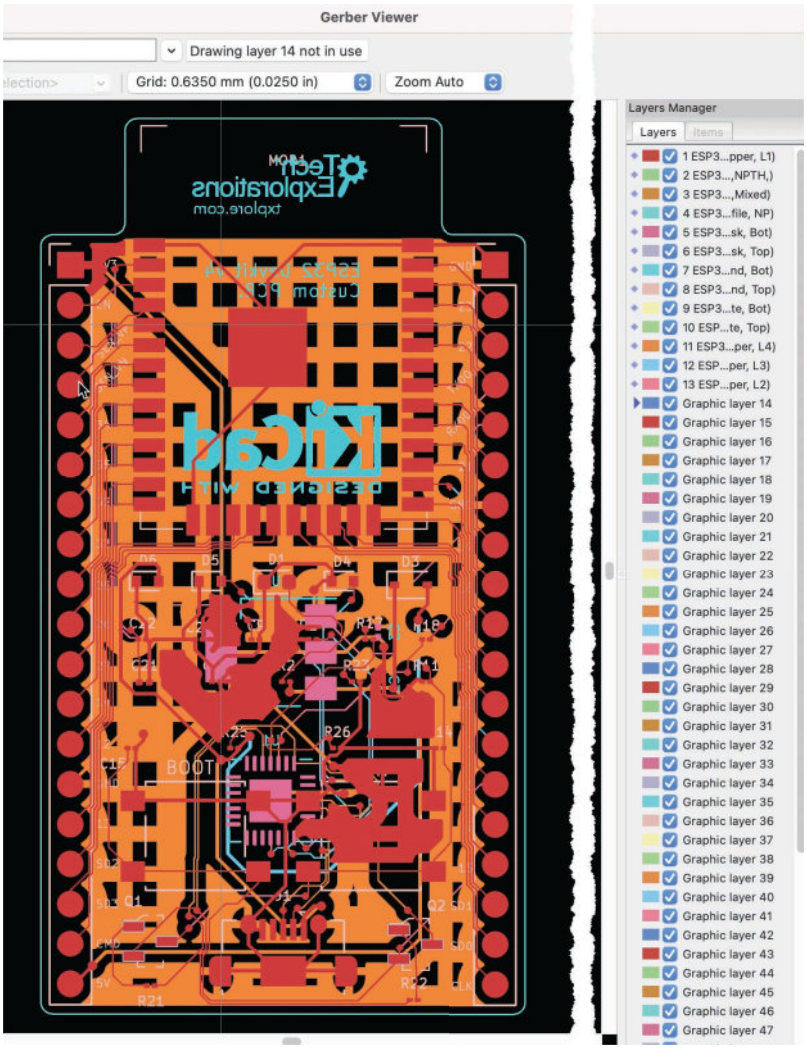


Figure 13.3.10.54: Gerber Viewer showing all layers.

The Gerber Viewer shows no issues of concern, so I can proceed with the order. My order form is below.

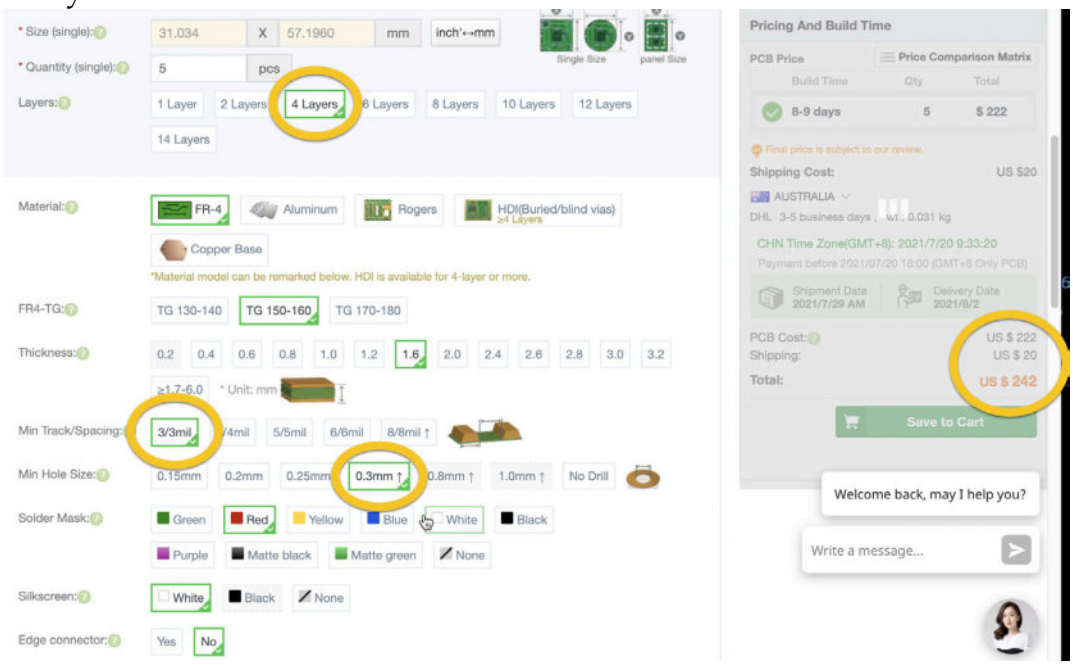


Figure 13.3.10.55: Project PCB order form.

In the order form I show above, I have marked with yellow circles the most important fields. This is a four-layer PCB, and I have selected 3/3mil for the track spacing and 0.3 mm for the minimum hole size. These values match the 0.1 mm track spacing and 0.3 mm minimum hole diameter that I used for the design of the board. With these values, the cost of five copies for this PCB is $222. Obviously, the clearance settings I have chosen (in response to the tiny components listed in the reference BOM) have increased the cost significantly. It is possible to design the layout for this PCB using larger settings, and in the process reduce the cost by a magnitude.

With this, I declare this project complete!

4. 3D shapes

In this chapter I will add several missing 3D models so that the 3D viewer can synthesise a complete 3D rendering of the board.

At the end of the layout workflow, my board looks like this in 3D:

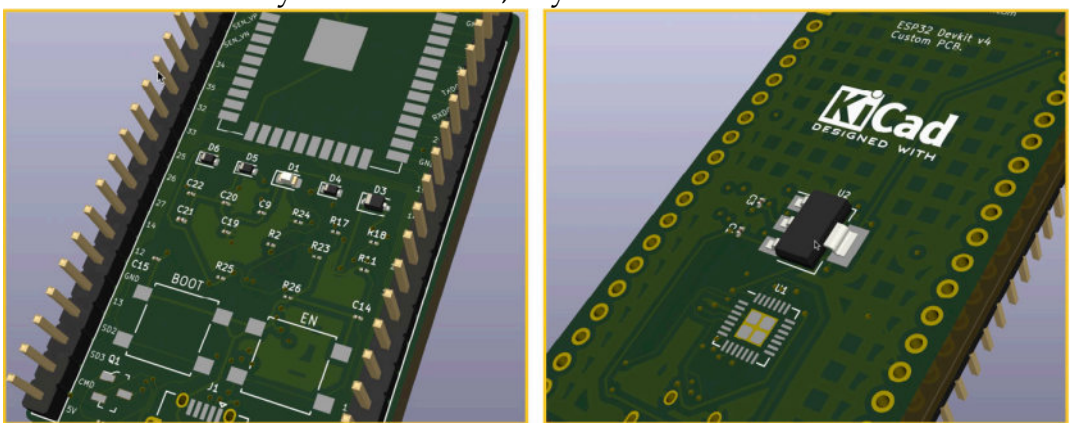


Figure 13.4.1: 3D rendering of my board.

As you can see above, the layout contains 3D models for the voltage regulator in the back, and the headers, capacitors, LEDs, and resistors in the front.

Let's go through the footprints that don't have a 3D model. Here are the sources of the 3D models I used:

- ESP32 module, from Snapeda: ESP32-WROOM-32D.step
- CP2102N chip, from Snapeda: CP2102N-A01-GQFN28.step
- Buttons, from Snapeda: b3s-1000p.stp
- USB connector, from KiCad's library: Connector_USB.3dshapes/USB_Micro-B_Molex_47346-0001.wrl
- Transistors, from KiCad's library: Package_TO_SOT_SMD.3dshapes/SOT-23.wrl

Download, install, and setup these 3D models in your layout. Below you can see my board in 3D, with all 3D models included:

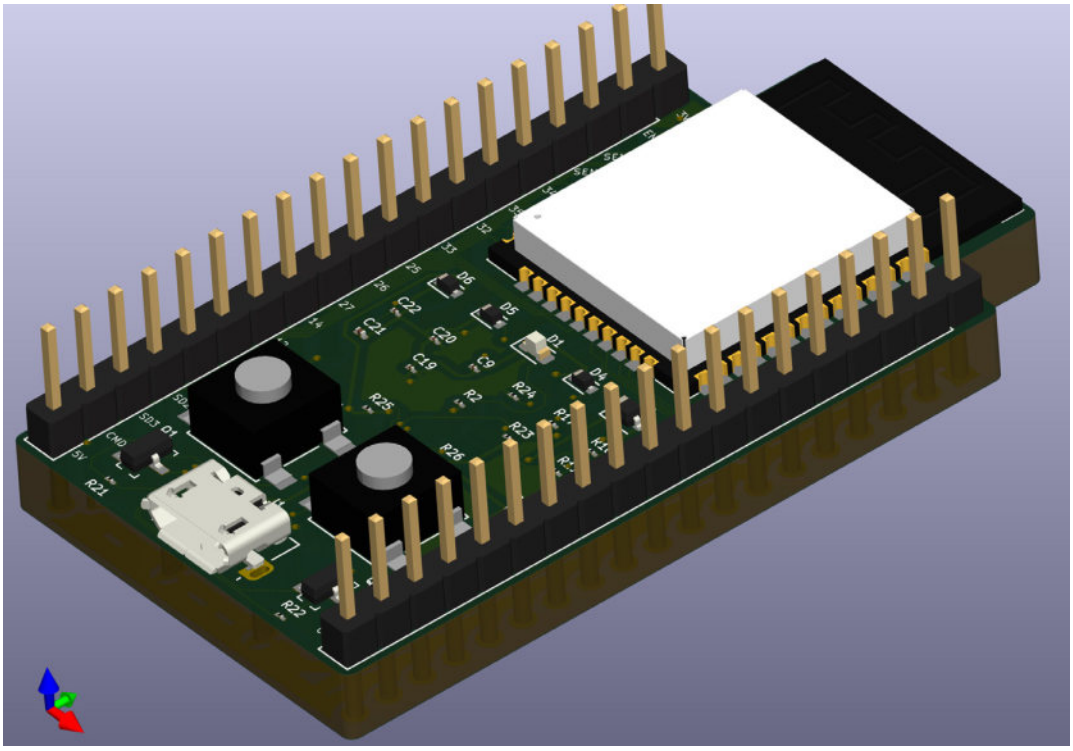


Figure 13.4.2: 3D rendering of my PCB, front, all models included.

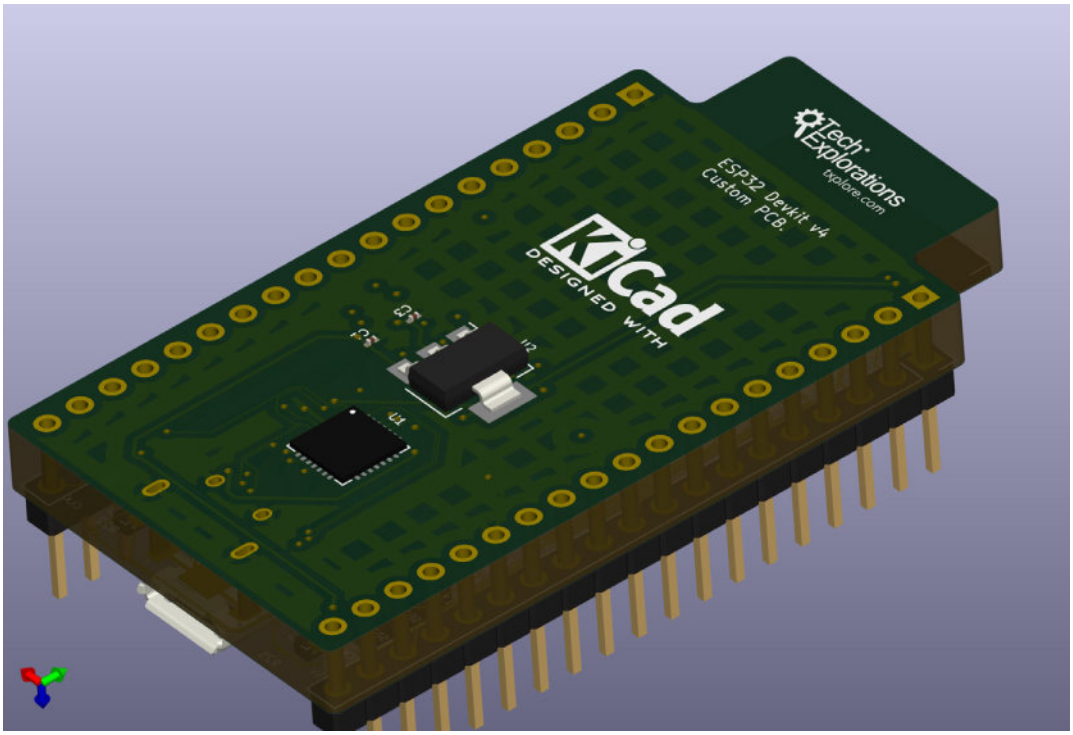


Figure 13.4.3: 3D rendering of my PCB, back, all models included.

Congratulations for completing this project!

5. Finding and correcting a design defect

As part of the beta program of this book, a reader found a defect in the schematic of this project. In this chapter I will show you how to quickly fix this defect.

The defect

During my copying of the elements and wiring from the ESP32 devkit reference design to my KiCad schematic, I did not notice that transistor Q2 was flipped. In the reference design, the emitter of Q2 (pin 2) is connected to DTR, as in the image below (this is a segment of the reference design):

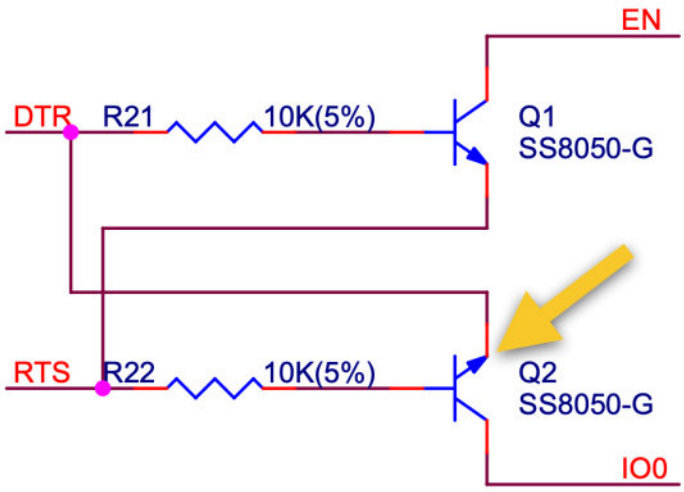


Figure 13.5.1: Segment of the reference design showing the correct wiring for Q2.

When I copied this network to my KiCad schematic, I did not notice the correct orientation of the emitter. The result is that I connected the emitter (pin 2) to IO0 instead of DTR. In the image below, I show the error in the schematic:

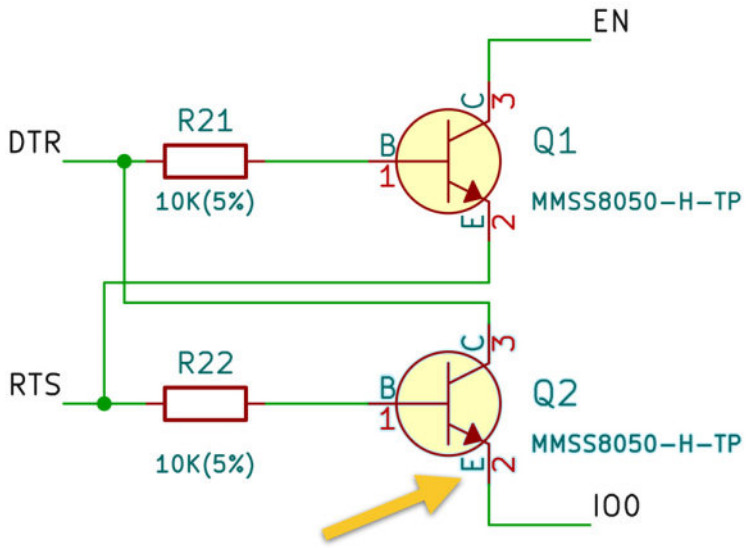


Figure 13.5.2: The arrow shows the error in my instance of the schematic. The emitter (pin 2) of Q2 is connected to IO0 instead of DTR.

Fortunately, this defect is easy to fix.

5.1. Fix the schematic

First, let's fix the defect in the schematic. I have to re-orient the emitter of Q2 so that it points towards Q1. The easiest way to do this is to use the "Mirror Vertically" option of the Q2 symbol properties (see below).

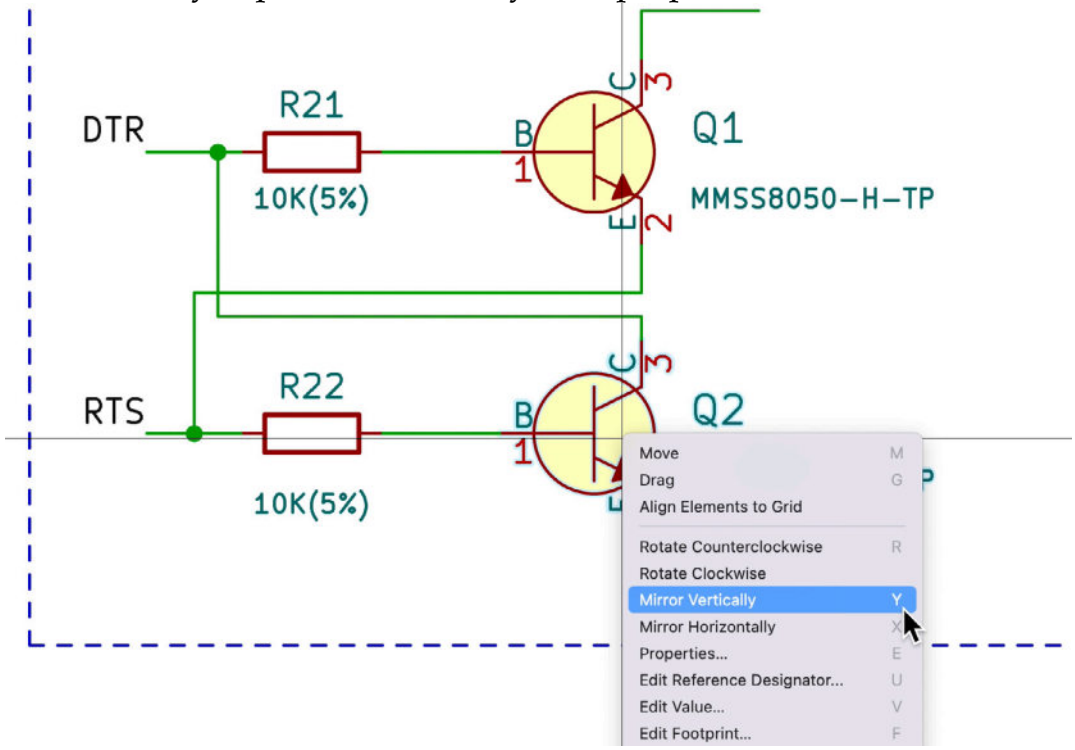


Figure 13.5.1.3: Fix the wiring defect by flipping Q2 vertically.

After the defect is corrected the Q2 and Q1 wiring will look like this:

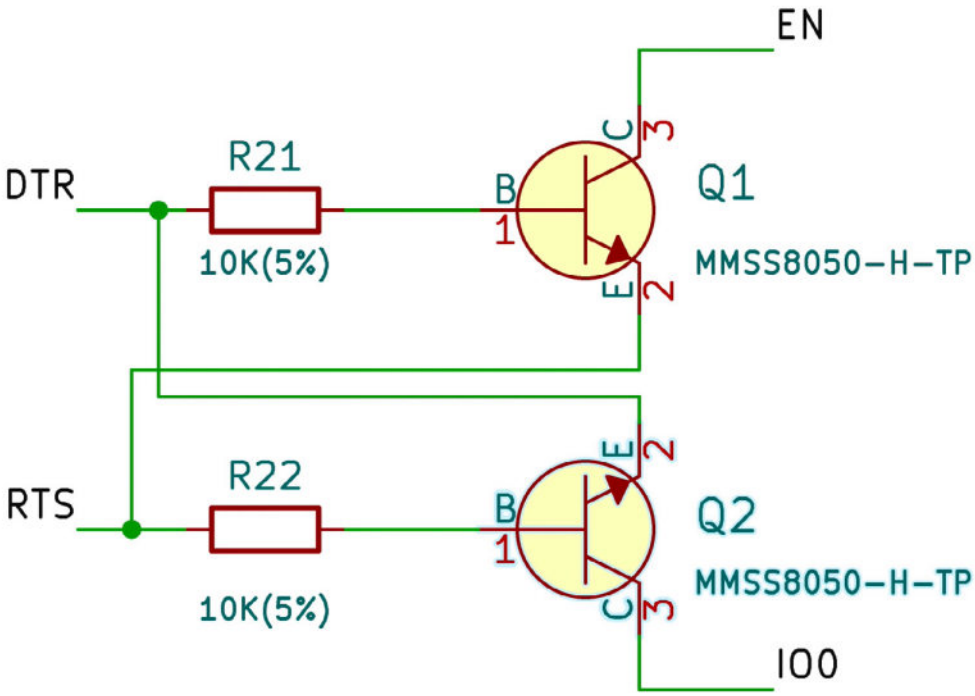


Figure 13.5.1.4: The defect is corrected.

Save the schematic. Let's continue the correction process in the layout editor.

5.2. Fix the layout

In Pcbnew, import the changes in the schematic into the layout, and notice that there are a couple of new ratnest lines in Q2 (see below).

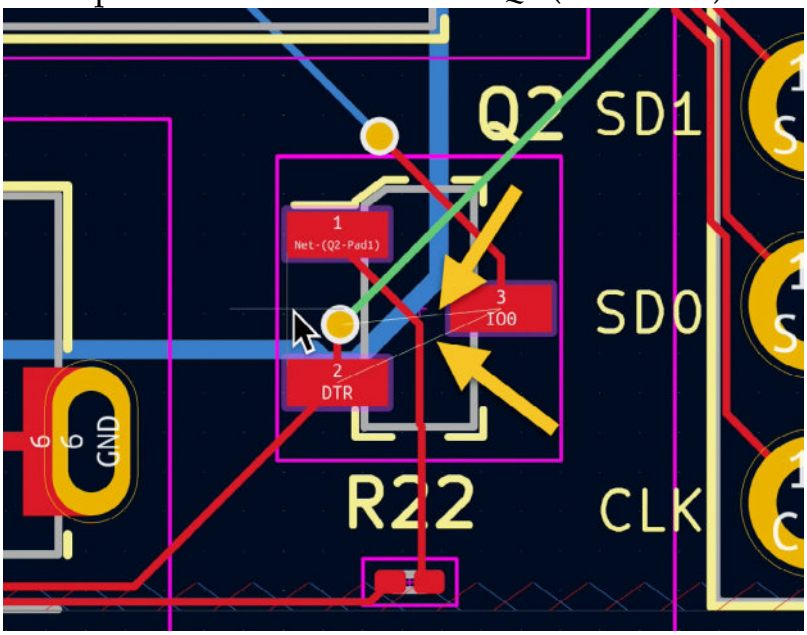


Figure 13.5.2.5: After importing the schematic changes, ratsnest lines appear in Q2.

After the import of the schematic changes, the layout editor not delete the copper traces that connect Q2 pins 2 and 3 to other parts of the PCB. The only clue that something has changed are the two new ratsnest lines. You will need to manually delete the original copper traces, and draw new ones.

In the image below I have already deleted several obsolete copper traces that originally connected Q2 pins 2 and 3 to the IO0 and DTR nets. I also dragged the DTR copper trace and via (marked with the yellow arrow) to the left to make it easier to connect to Q2 pin 2 next.

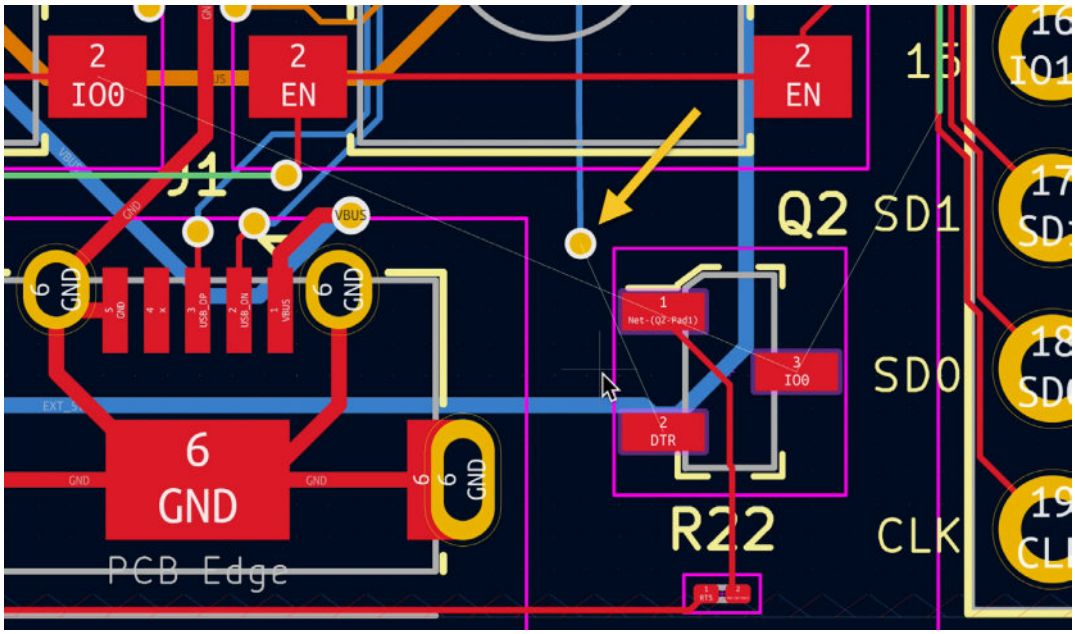


Figure 13.5.2.6: Removed obsolete

copper traces.

Finally, re-draw the copper traces. In the figure below, I use arrows to highlight them:

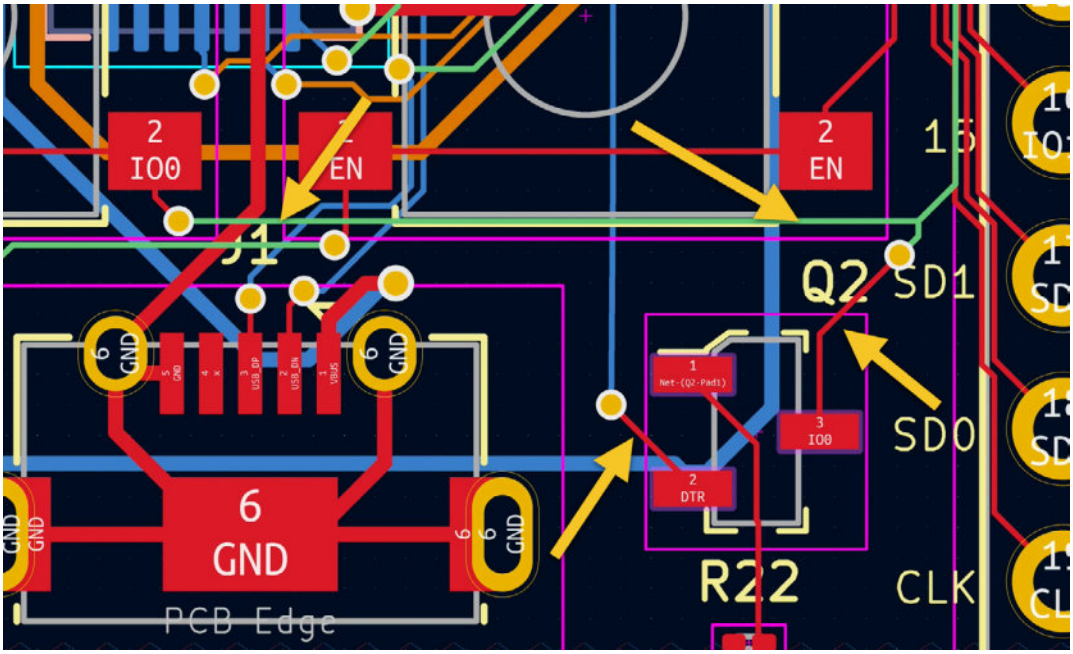


Figure 13.5.2.7: New copper traces from Q2 pins 2 and 3.

With the re-draw complete, the original defect is now corrected. Do a DRC to ensure there are no other issues to address. My DRC shows a problem with the copper track coming from MOD1 pin 26 (see below).

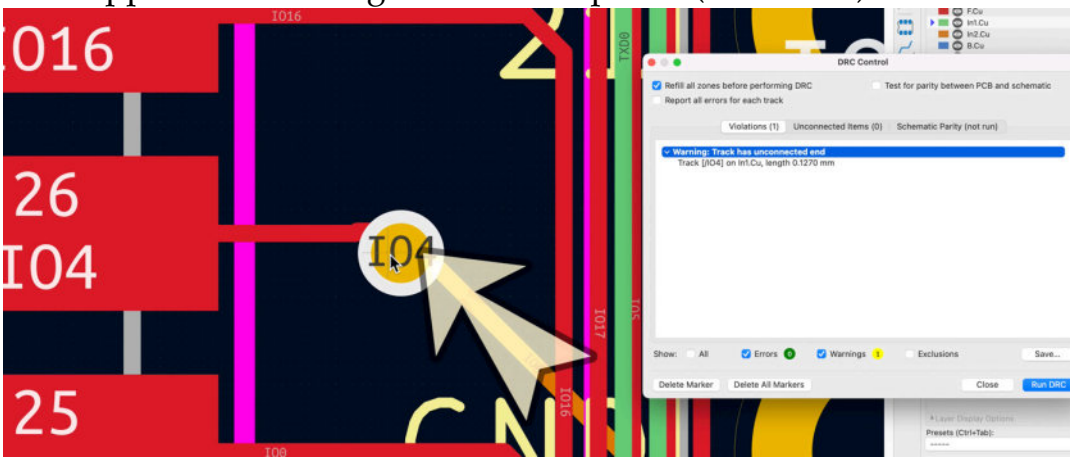


Figure 13.5.2.8: A new violation.

To fix this violation, I opted to delete the via and adjoining traces, and re-draw them. You can see the result below:

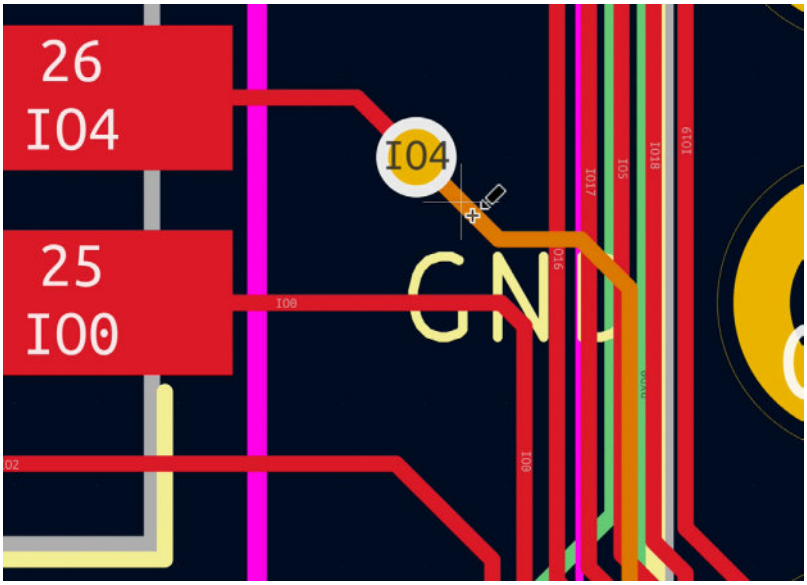


Figure 13.5.2.9: Re-wired and fixed the last violation.

My latest DRC showed no further defects, so I was able to re-export the Gerber files.

Part 13: Recipies

1. Create a custom silkscreen or copper graphic

One of the most common "finishing touches" of PCB design is to add graphics. For example, you could add decorative pictures, a logo, or a stamp representing something important about your new board.

In the projects in this course, I have routinely added two specific graphical elements: The Tech Explorations logo and the KiCad logo. Here's an example (13.1.1):

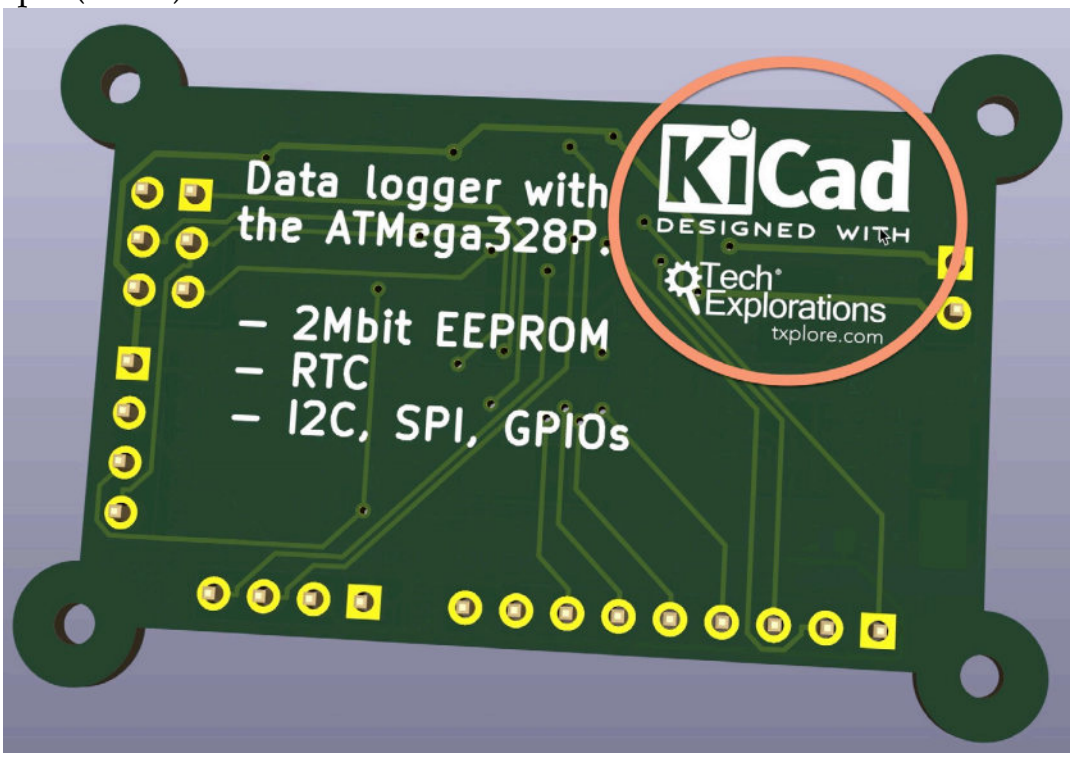


Figure 13.1.1: Two examples of graphical elements, placed in the back silkscreen.

In this example, I have placed the two logos in the back silkscreen layer. The KiCad logo is available from the footprint library marked as "Symbol" that ships with KiCad (Figure 13.1.2).

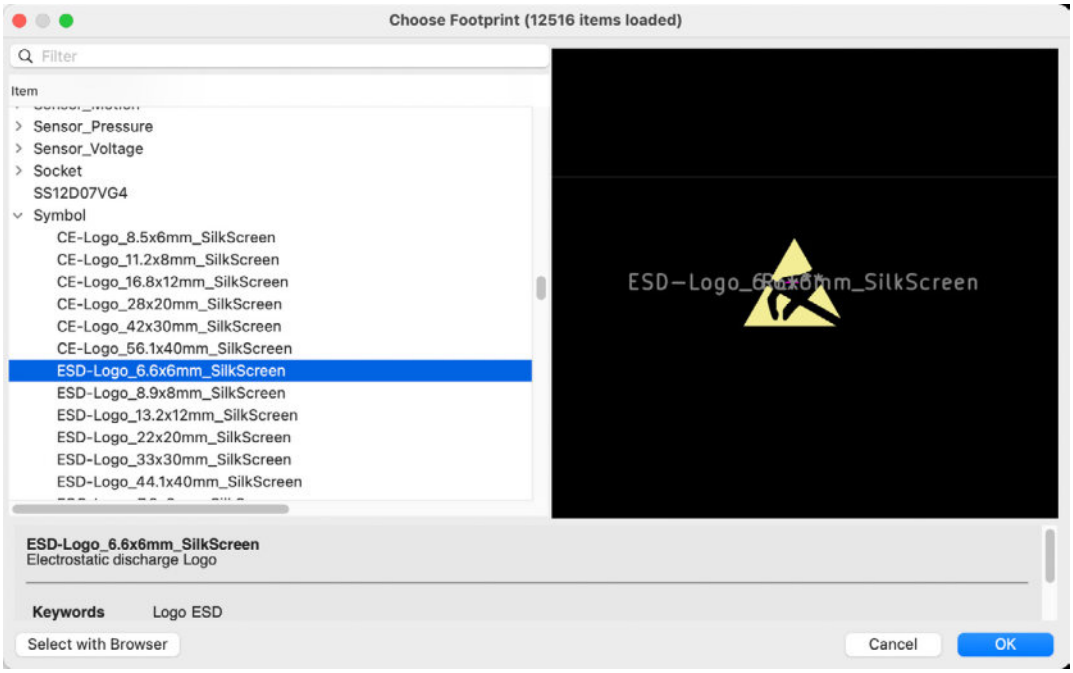


Figure 13.1.2: The Symbol footprint library contains several graphics footprints.

You can select any of the available footprints in this library and drop them on the PCB editor as you would with any other footprint.

If you want to use a graphic not available in the existing libraries (and you can't find it anywhere on the Internet), you can create it.

In this Recipe, I will show you a simple method that yields a high-quality graphical footprint.

First, let's look at the layers that can hold a graphical footprint. You can place graphic footprints in four layers:

- Front silkscreen (F.Silkscreen).
- Back silkscreen (B.Silkscreen).
- Front copper (F.CU).
- Back copper (B.CU).

Regardless of which one you choose, the process for creating custom graphics in KiCad is the same.

Here is the process, in a nutshell (Table 13.1.1):

Step #	Description

1	Find the raw image that you want to use and convert it to a black and white bitmap.
2	Use the KiCad Image Converter app to convert the bitmap image from step one into a footprint.
3	Save the new graphical footprint into a library, and add the library to the scope of your project.
4	Add the new graphical footprint to your PCB in Pcbnew.

Table 13.1.1: The four steps for creating a custom graphical footprint.

KiCad's Image Convert app can convert a bitmap into four different formats:

- Symbol (.lib file)
- Footprint (.kicad_mod file)
- Postscript (.ps file)
- Drawing Sheet (.kicad_wks file)

Since we are interested in creating a graphic, we'll be exporting it to a .kicad_mod file (footprint).

Let's begin the process.

Step 1: A graphic in its original format
For this example, I'll be starting with a color PNG image file. You can also start from a JPEG or other image file format (you'll need to adjust the operation of converting your starting image file to a black and white bitmap) (Figure 13.1.3).

Figure 13.1.3: I'm starting with this PNG color image.

My original image contains elements in black, red, and white (the background). I need to convert this image into a black and white (only) bitmap image. Therefore, I will need to convert any red contents into either black or white. In this case, it makes sense to convert the red content into black.

I'm not a graphics designer and don't want to overthink problems like this. Instead of using a graphics tool like Photoshop to do the job (a clear overkill), you can use one of the many online image converters. A quick Google search will reveal a selection of online (and mostly free) image format converters. I found that the one at https://image.online-convert.com/ is precisely enough for this task. This tool offers a variety of converters. The one you want is "image to BMP". You can find it at https://image.online-convert.com/convert-to-bmp.
Drag and drop the PNG file in the green file area to upload the image (Figure 13.1.4).

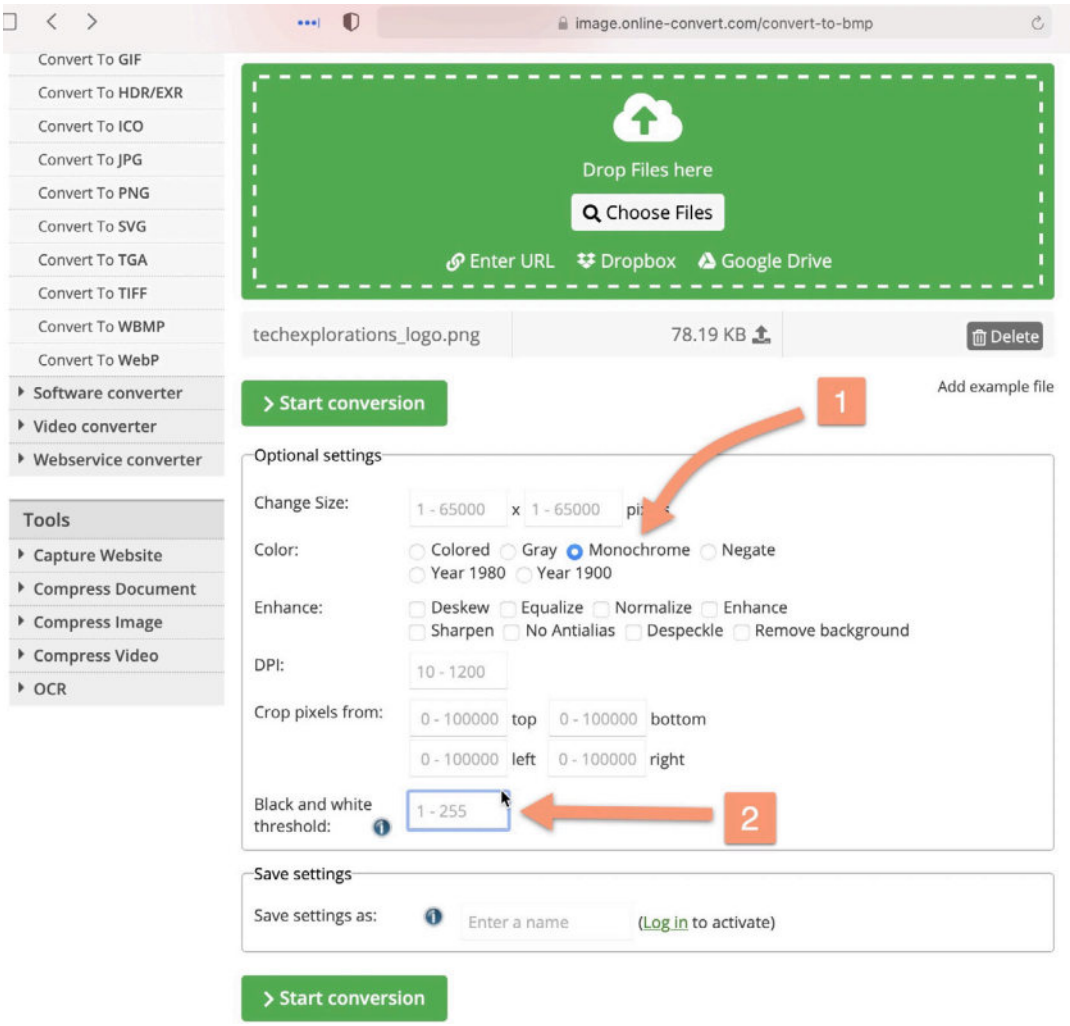


Figure 13.1.4: The image converter form.

Once the file upload is complete, check the "monochrome" option (1) under "Color". You can leave all other settings unchanged.

The "Black and white threshold" (2) accepts a number that helps you control which areas of the original image file will be converted to black and white. For simple graphics, like the Tech Explorations logo, the tool does a good job of automatically working out the conversion threshold. I did not have to make any changes here. However, if you are not getting the desired results, play around with a few numbers in the threshold text field until you find the one that works best.

Click on the green "Start conversion" button to trigger the conversion, and download the new BMP file.

Figure 13.1.5: The image converter form.

Now that you have the new BMP file (Figure 13.1.5), you can continue with Step 2.

Step 2: Use the Image Converter to create the footprint

To start the KiCad Image Converter app, go to the KiCad Project window, and click on the Image Converter icon (Figure 13.1.6).

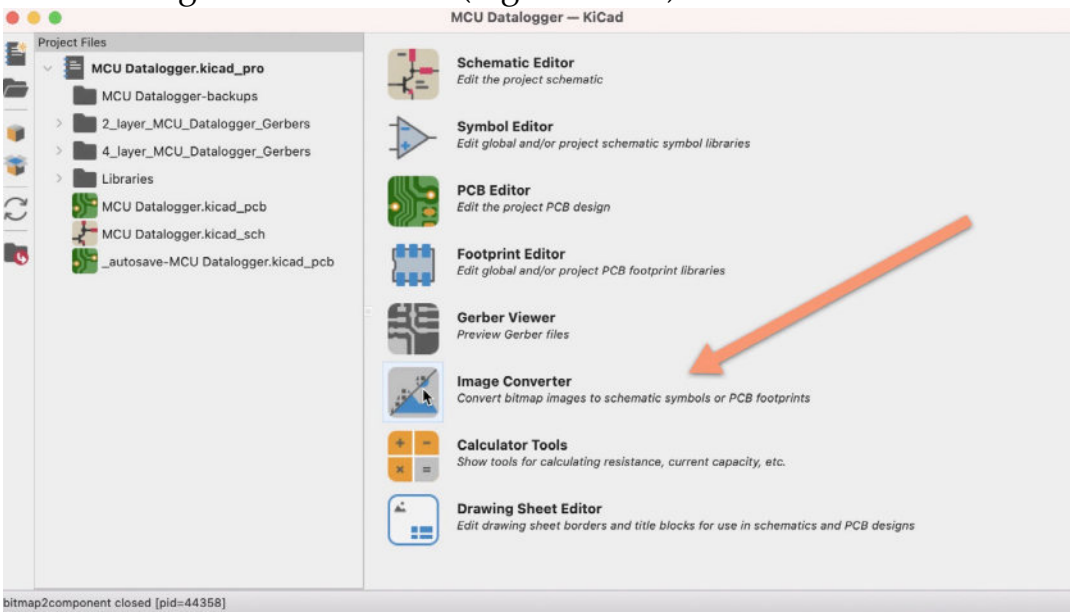


Figure 13.1.6: The image converter form.

In Image Converter, click on the “Load Bitmap” button and then find and open the BMP file you just created. Your Image Converter will look like this (Figure 13.1.7):

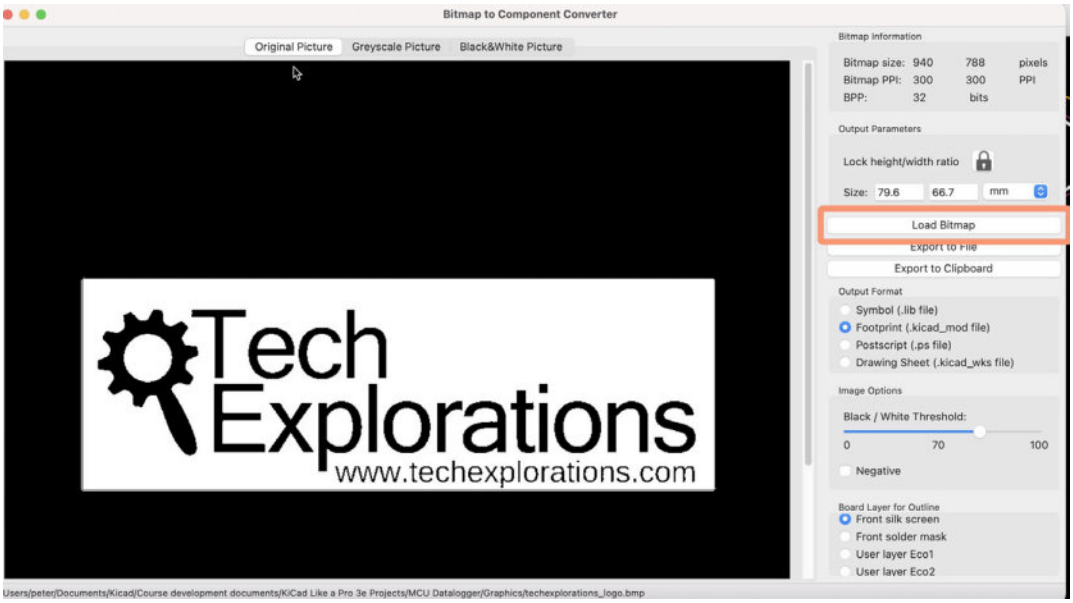


Figure 13.1.7: The image converter form.

You can switch the image view between “Original Picture,” “Greyscale Picture,” and “Black&White Picture.” At this point, ensure that the black and white picture is how you want it. This is where you can confirm that the PNG to BMP conversion was successful. If there is something that you want to improve in the image, should do it now by returning to Step 1 of the process.

If the black and white picture looks good, you can proceed. Ensure that you have selected the “Black&White Picture” tab and focus on the Size widget container. This is where you can control the size of the footprint that you are about to create. As the number of pixels in the original photo is fixed, changes in the size will affect the resolution.

To work out the appropriate size of the graphic, go back to Pcbnew and use the measurement tool to find out the height and width of the graphic (Figure 13.1.8).

Figure 13.1.8: The image converter form.

In my measurement (see above), I confirm that I can place my logo graphic in the approximate center of the PCB, with a width of around nine or ten millimeters. With this information, go back to the Bitmap to Component Converter app and type “10” in the first Size text box. The default state of the height/width ration lock is “locked,” so the tool will automatically calculate the height for the graphic (Figure 13.1.9).

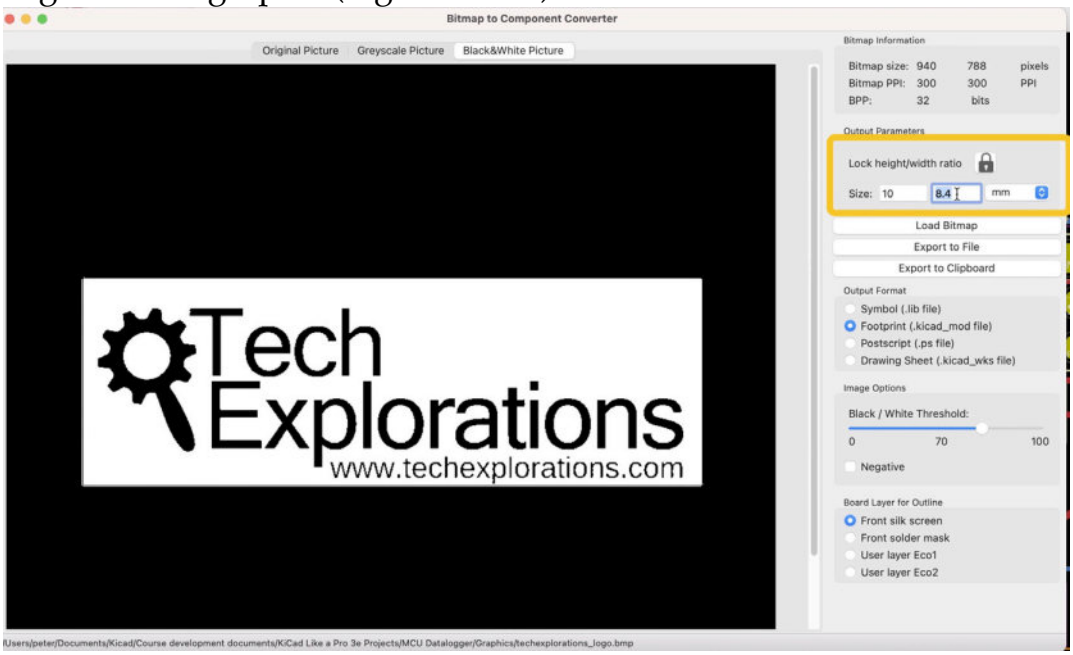


Figure 13.1.9: Set the size of the footprint.

Notice that as you manipulate the size numbers, the figures in the “Bitmap information” group also change. In general, as long as the Bitmap PPI is over 300, your graphic will look good in the final manufactured PCB.

Ensure that you set the output format to "Footprint."

You can also play with the Black/White Threshold and see how it affects the image. For this image, it looks best when the threshold is around 50. In the same widget group, you can check the "Negative" checkbox. This way, you can flip black and white regions. As the graphic appears in Figure 13.1.9, the graphic background is drawn with white ink in the silkscreen. The text and logo are ink-free, so they "inherit" the color of the solder mask. If you check the "Negative" box, the logo text is drawn with the white silkscreen ink, and the background is clear as it inherits the color of the solder mask.

Finally, select the layer for the footprint you are about to create. You can choose between the silkscreen, solder mask, and user layers Eco1 or Eco2. The silkscreen and solder mask will produce a graphic that is visible when the PCB is manufactured. However, the user layers are not manufactured. If you choose a user layer, you will need to do further work in Pcbnew to change one of the manufactured layers if you want this graphic to be visible.

In my example, I have selected the “Front silkscreen.“ You will be able to place the footprint on the back silkscreen in PcbNew.

Finally, click on “Export to File” button, and save the new footprint in your project or custom libraries directory. You will use this file in the next step (Figure 13.1.10). In this example, I am saving the new footprint inside a directory that KiCad and PCBnew are already configured to look for footprints. This means that I will be able to use the new graphic immediately. If you are saving the new footprint in a location that Pcbnew does not know about, you should follow the process I describe earlier in this book to install the library in Pcbnew before you can use it.

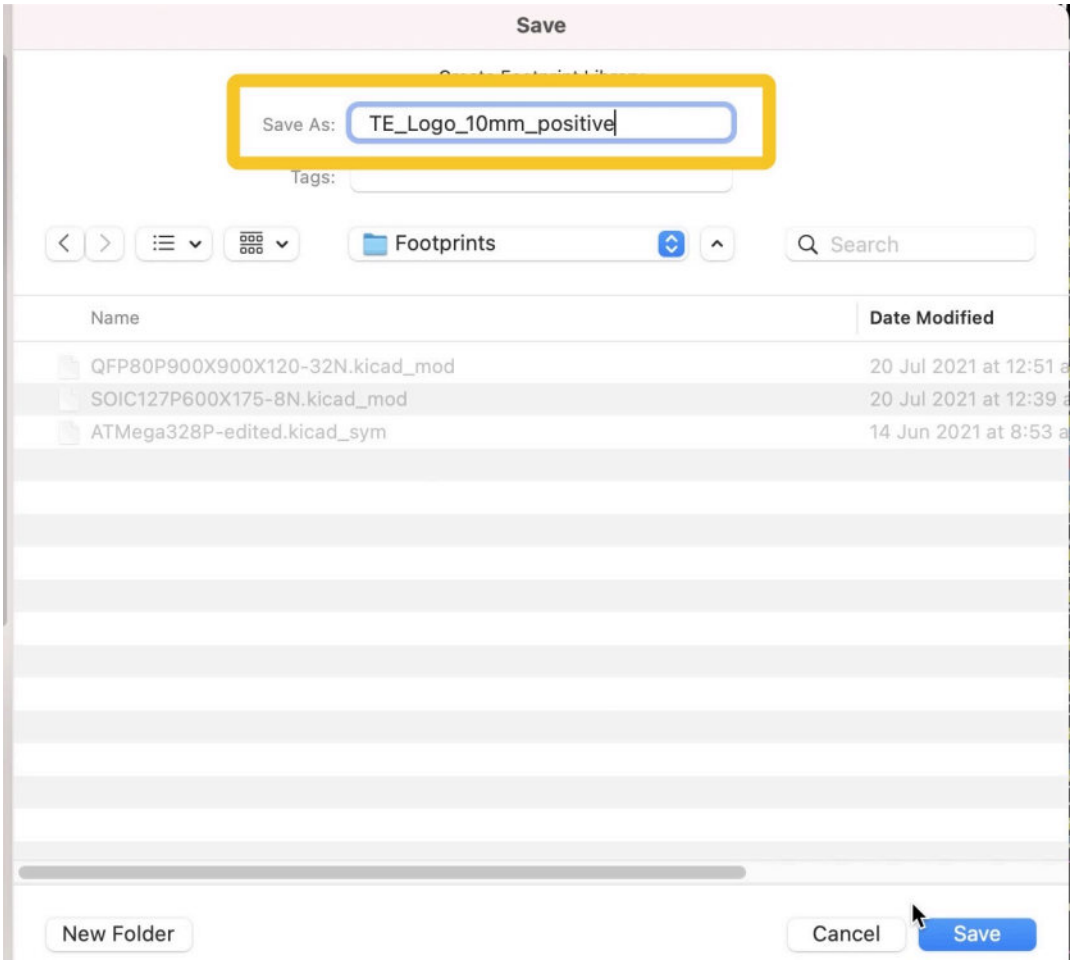


Figure 13.1.10: Set the size of the footprint.

As part of the filename, I take care to include the size ("10mm") and other information that can help me choose between alternatives later on (such as "positive" vs. "negative," "silkscreen" vs. "solder mask" etc.).

Step 4: Use the new graphical footprint in Pcbnew

If you saved your new footprint in one of the footprint libraries already in the KiCad footprint libraries path, you can continue with Pcbnew.

Start Pcbnew, select the "Add footprint" tool from the right toolbar and click anywhere inside the editor to bring up the footprint chooser. Search for the new footprint by typing part of its name in the filter box (Figure 13.1.9).

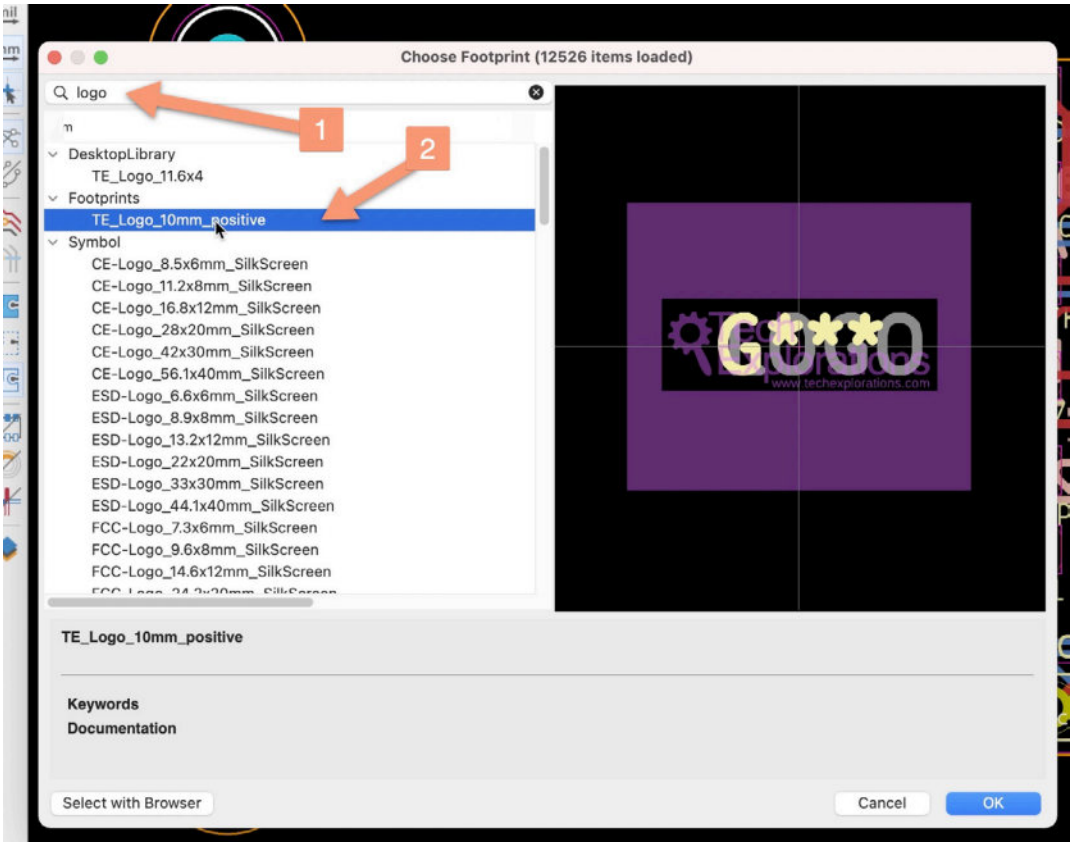


Figure 13.1.11: The new footprint is available in the footprint chooser.

In the example above, I have searched for “logo” (1), and the new footprint is listed under my “Footprints” library (2). Double-click to drop the footprint in the editor.

When the new footprint appears in the editor, double-click on it to edit its properties (Figure 13.1.12).

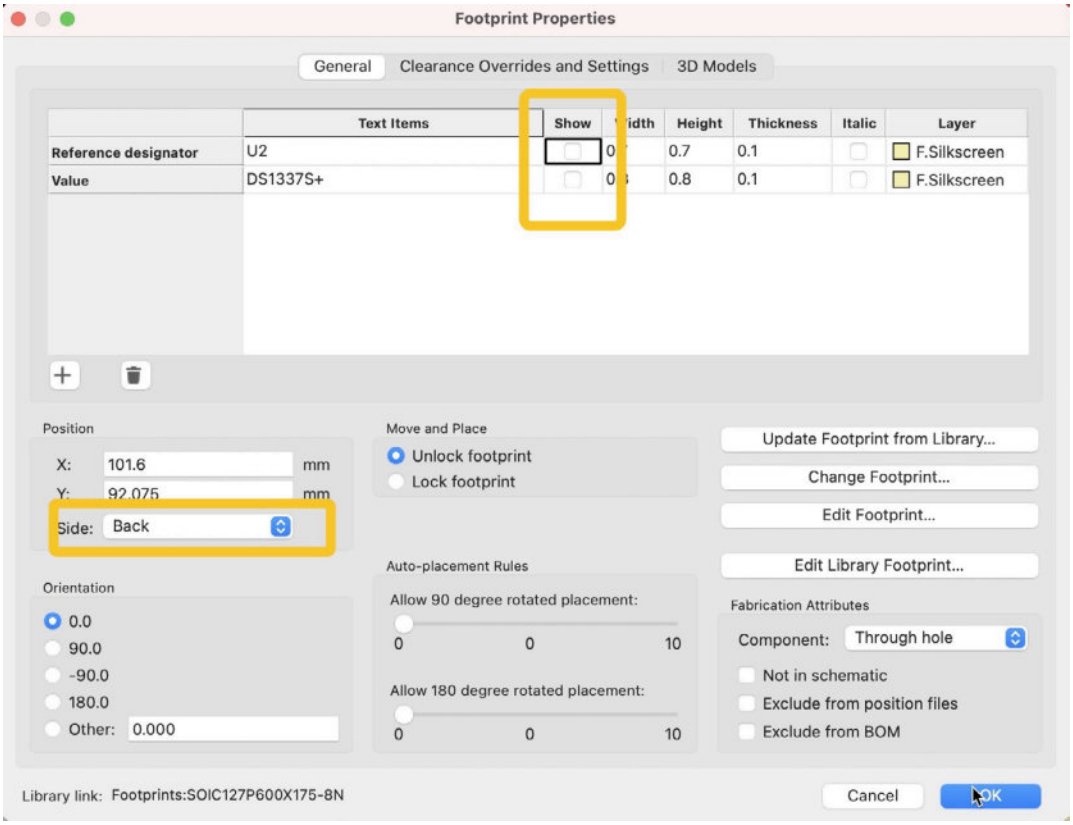


Figure 13.1.12: The new footprint properties.

Since I want to place the footprint in the back of the PCB, I have set the "Side" dropdown to "Back." I have also unchecked the "Show" checkbox for the reference designator.

In the 3D viewer, the new footprint graphic looks like this (Figure 13.1.13):

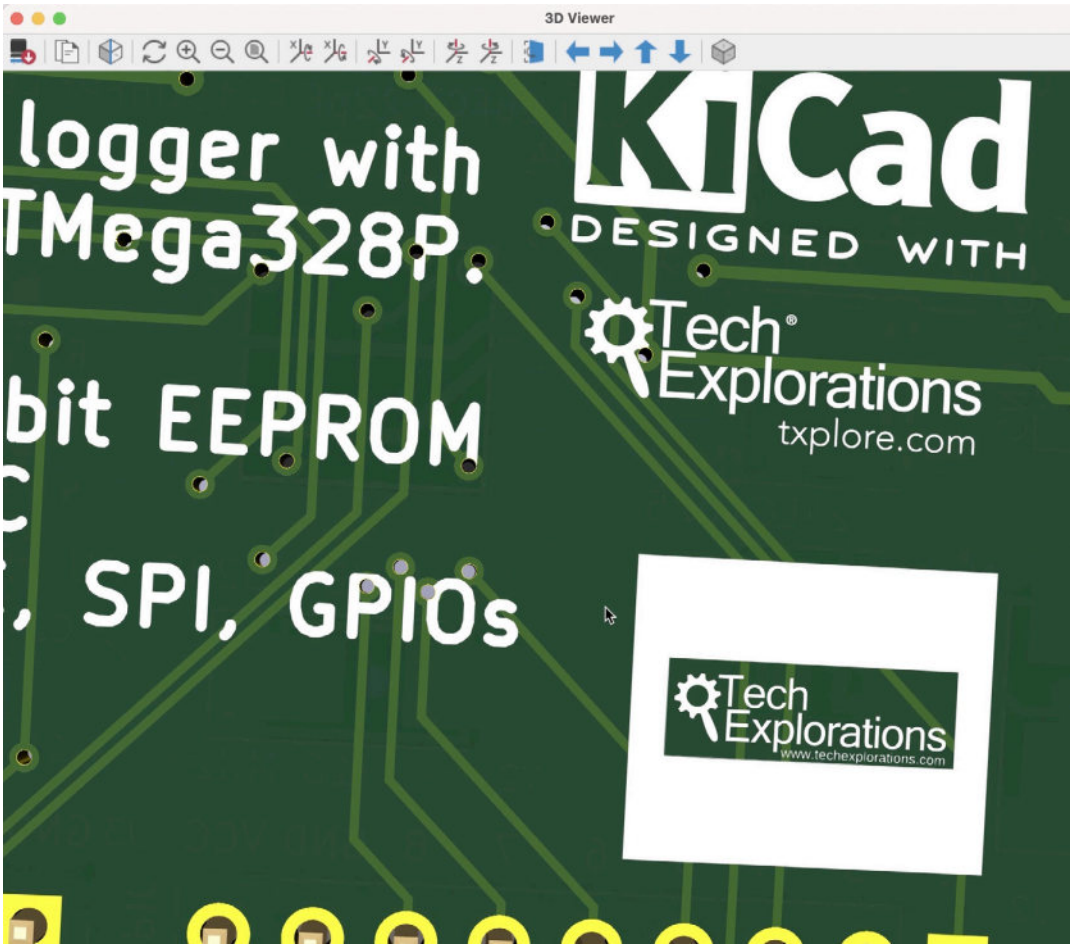


Figure 13.1.13: The new footprint in the back of the PCB.

If you want to experiment with the negative version of the same footprint, go back to the Bitmap to Component Converter app, and check the "Negative" checkbox like this. Export the footprint with a new filename so that the original footprint is preserved. I used "TE_Logo_10mm_negative_silkscreen.kicad_mod". Then, add the new footprint to the editor, and view it in the 3D viewer. Mine looks like this (Figure 13.1.14):

Figure 13.1.14: The new footprint (negative) in the back of the PCB.

You can create as many variations as you wish for the same graphical footprint. There can be variations in size, layer, and black/white threshold ("negative" vs. "positive").

2. Change a symbols and footprints in bulk

It is possible to change symbols and footprints in bulk using the respective tools in the schematic in layout editors. In this chapter you will learn how to use those tools.

2.1. Change a symbol in bulk

This section will teach you how to replace the symbol used for a specific component in a KiCad schematic in bulk.

Here's an example.
Say that you have a schematic that contains several resistors and capacitors, like this (Figure 13.2.1.1):

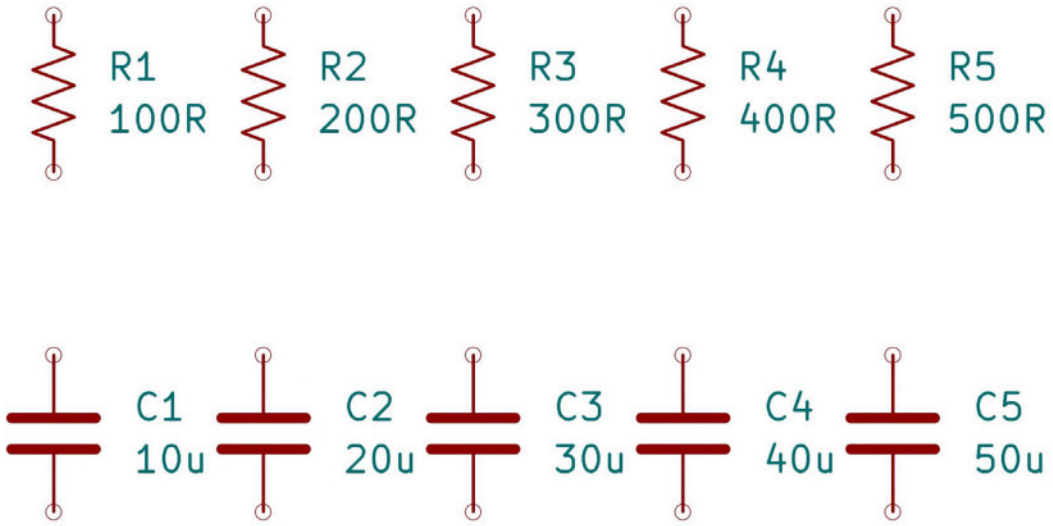


Figure 13.2.1.1: This schematic contains several resistors and capacitor; we'll change these symbols in bulk.

The symbols I have used for the resistors follow the US/IEEE 315-1975 style. I would like to change all resistor symbols to use the IEC 60617 (International Electrotechnical Commission) style. You can see a comparative presentation of symbols in Wikipedia.

Of course, I could make the change by editing the properties of each symbol, one at a time. To do this, double-click on a resistor symbol to reveal its properties window, and then click on the "Change Symbol" button (Figure 13.2.1.2).

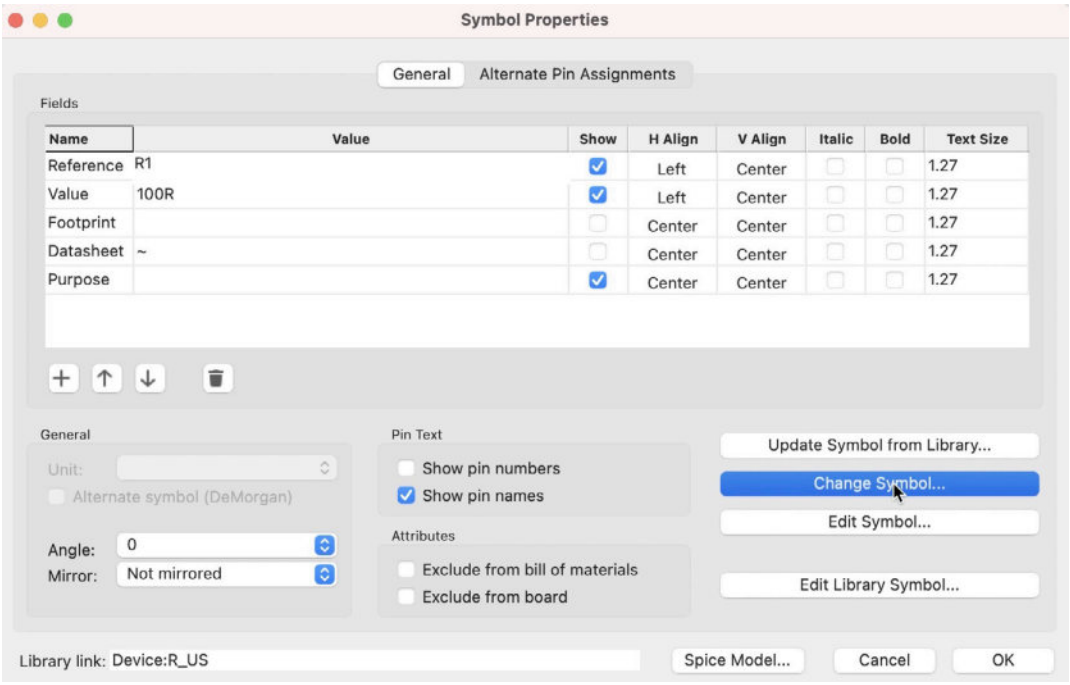


Figure 13.2.1.2: The individual symbol properties window.

This will bring up the "Change Symbols" window, which looks like this (Figure 13.2.1.3):

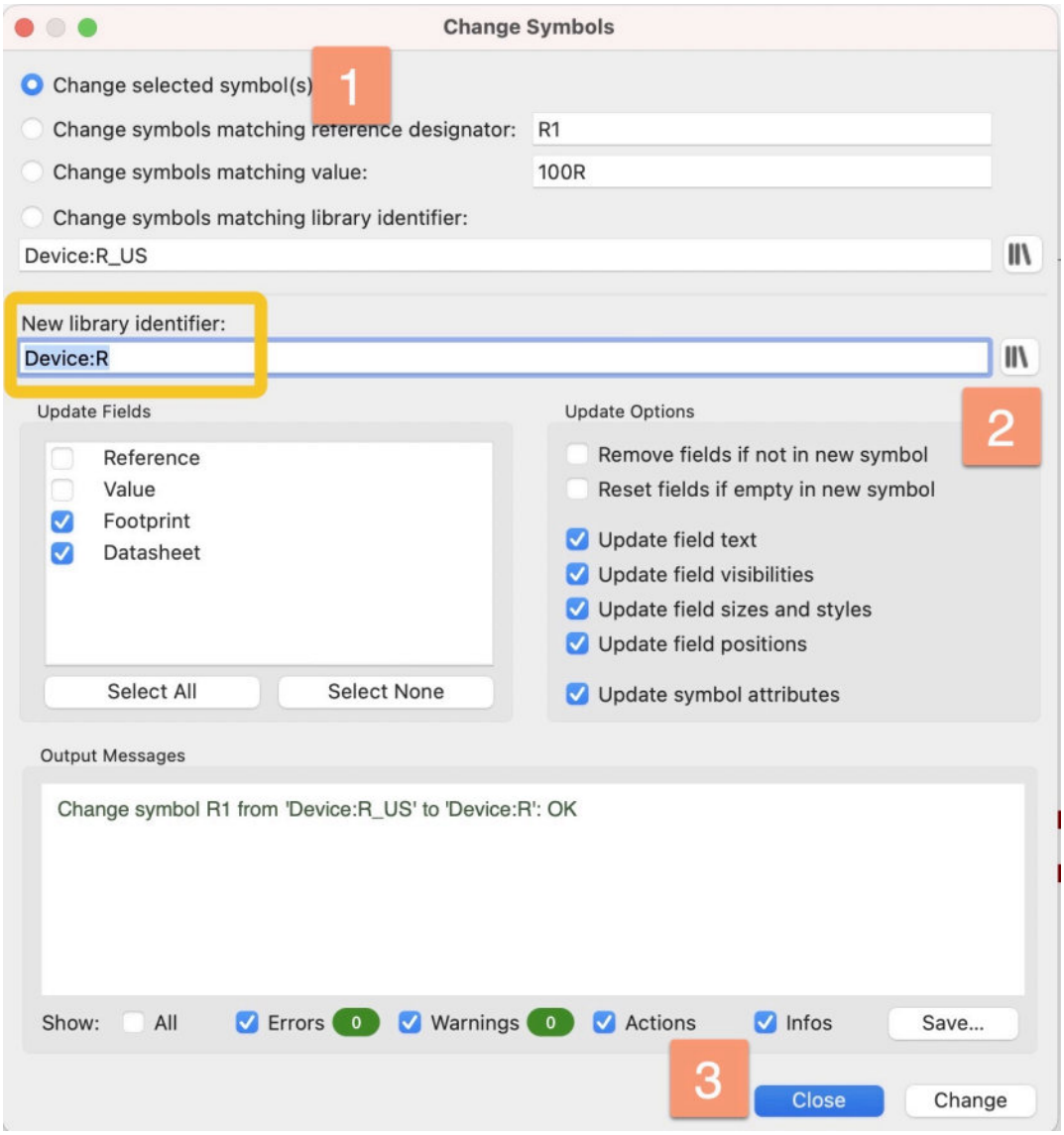


Figure 13.2.1.3: The Change Symbols window.

In the Change Symbols window, notice that the first option is selected, “Change selected symbol(s)” (1). You can choose multiple symbols by holding down the Shift key as you click on each symbol.

Find the new symbol you want to use by clicking on the library button (2). You can see the name of the new symbol in the “New library identifier” text field.

Click on “Change” to finish the process.

You can access the Change Symbols window directly. Click on the Edit menu item, and then Change Symbols (Figure 13.2.1.4).

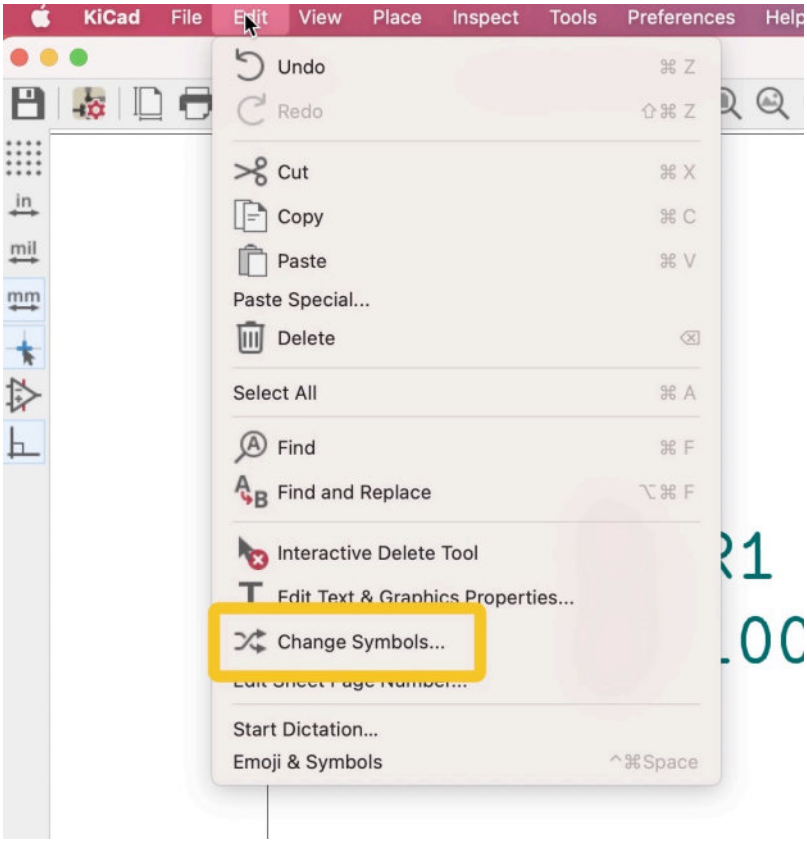


Figure 13.2.1.4: The Change Symbols window.

This makes it possible to change the symbol of more than one component. Using the Change Symbols window, you can choose one of these methods:

1. You can multiple-select all symbols by holding the Shift key down as you click on each symbol. Then, bring up the Change Symbols window and follow the same process I described earlier (i.e., as if you were changing the symbol for a single component). Ensure that the first option, "Change selected symbol(s)," is selected.
2. You can go directly into the Change Symbols window, without first selecting any symbols in the editor, and use one of the other three selection options:
    - Matching reference designator.
    - Matching value.
    - Matching library identifier.

Let's look at an example where I'll use the library identifier option.

To begin with, you will need to know the current symbol library identifier. You can find that by double-clicking on one of the symbols you want to change. This will bring up the Symbol Properties window. You can see the

library identifier in the bottom field of the Properties window (Figure 13.2.1.5).

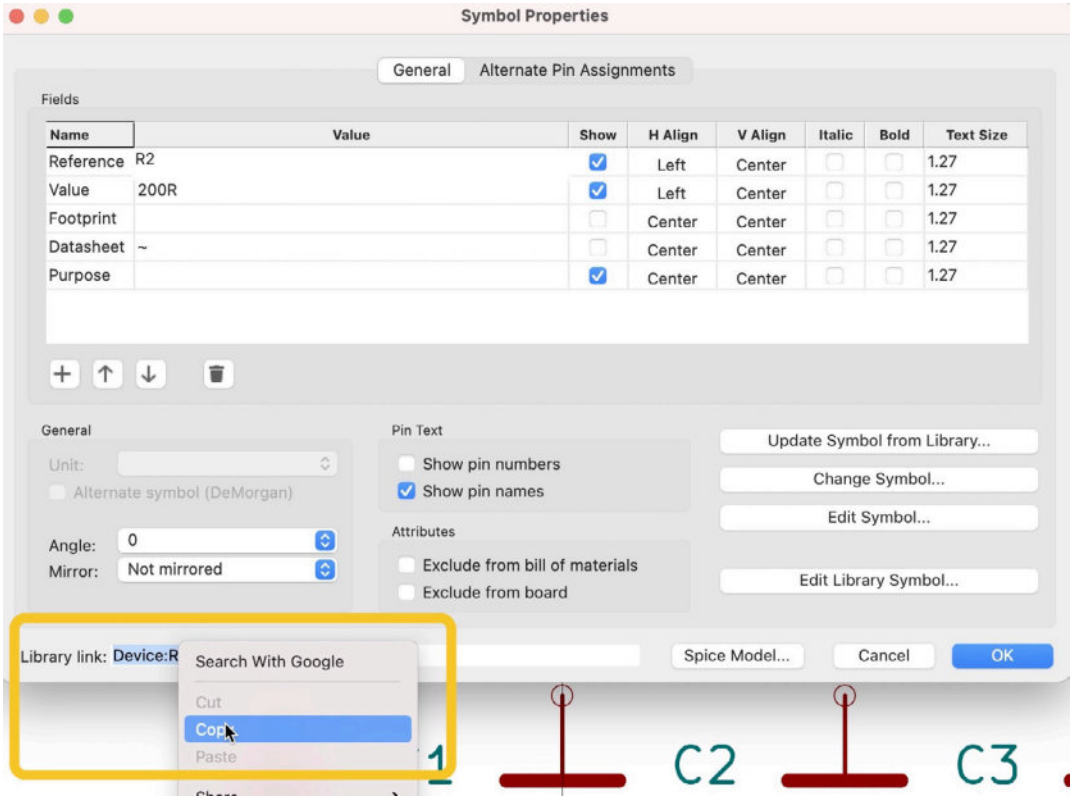


Figure 13.2.1.5: Get the library identifier from the symbol Properties window.

Copy the library identifier, and then bring up the Change Symbols window (Edit —> Change Symbols) (Figure 13.2.1.6). Paste the property identifier in the "Change symbols bathing library identifier" field (1). Of course, ensure that this radio button is selected. You can also click the library button next to this text field to bring up the library browser and find the library identifier instead of copying from another symbol.

Next, fill in the library identifier for the symbol that you want to use (2). If you know what it is, you can either type it in or use the library browser to find it.

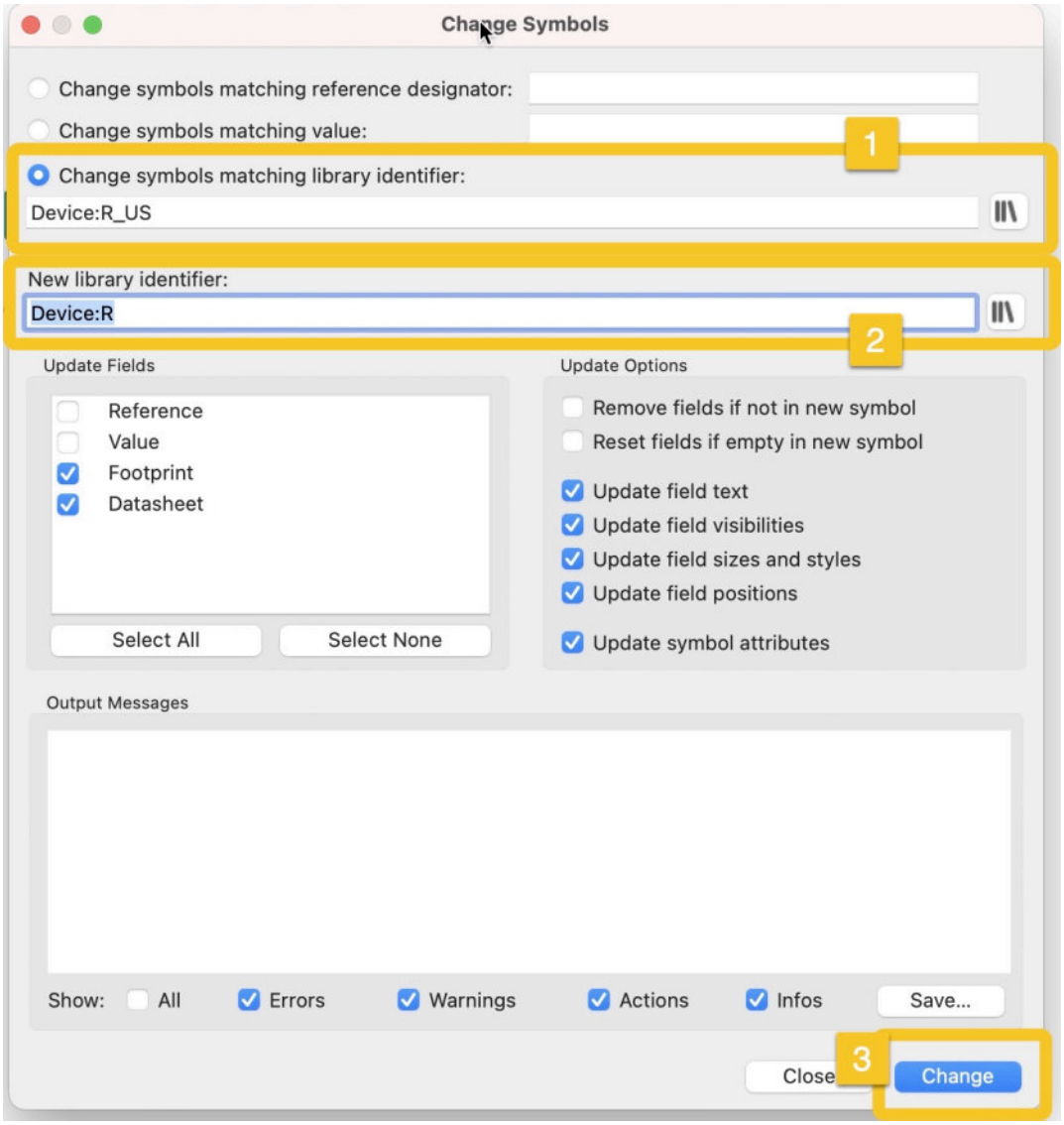


Figure 13.2.1.6: The Change Symbols window setup to change R_US symbols to R.

There are a few more options that you can explore in the "Update Fields" and "Update Options" groups. In most cases, the defaults are appropriate. When you are ready, click on "Change" to finish the process.

With a single operation, all resistor symbols are now changed (Figure 13.2.1.7):

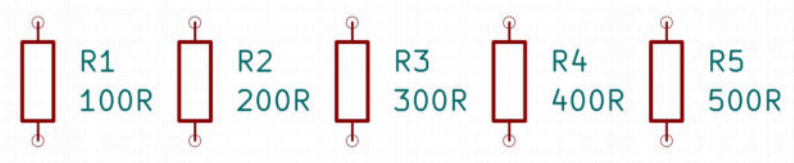


Figure 13.2.1.7: The symbols for the resistors have been changed in bulk.

The Change Symbols window is a powerful time-saver feature in KiCad 6, similar to its "sibling" features "5. Edit Text & Graphics Properties," "Find

and Replace" and the "Interactive Delete Tool". You can learn more about these features in their dedicated chapters.

2.2. Change a footprint in bulk

This section will teach you how to replace the footprint used for a specific component in a KiCad schematic in bulk. I will use an example of the PCB from one of the projects in this book:

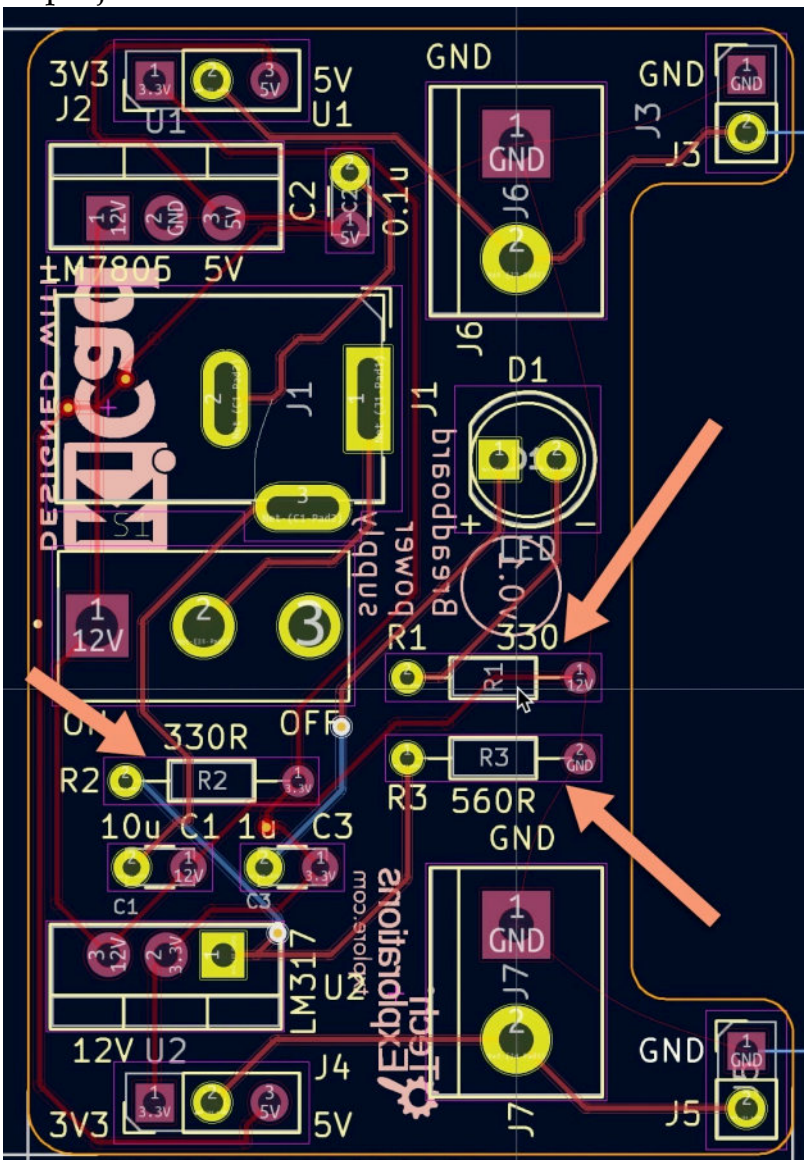


Figure 13.2.2.8: I will replace the TH resistors with SMD equivalents.

In the figure above, the PCB contains three TH resistor footprints. I will use the Change Footprints tool to replace these footprints with an SMD equivalent. To make a bulk change, you must know a property that is common among all footprints. In this example, all resistors use the same

footprint, with the same identifier. You can see this identifier in the footprint's properties window (at the bottom of the window):

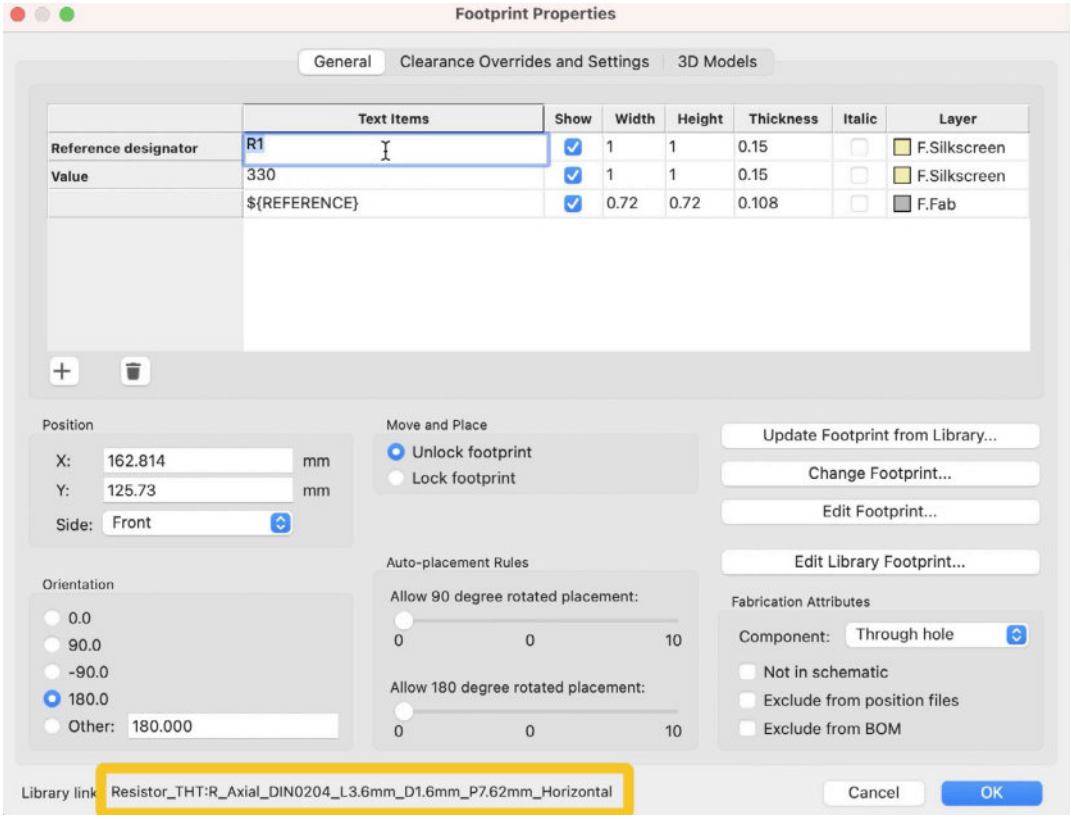


Figure 13.2.2.8: The footprint identifier in the footprint properties window.

With this information at hand, bring up the Change Footprints tool (under the Edit menu). As per the example below, use the fourth options as the footprint selector ("1"), and copy the footprint identifier that is common among the footprints you want this change to apply to. Then, select the new footprint identifier, "2". You can use the footprint chooser by clicking the library button. Finally, click Change. You will see confirmation of the changed footprints in the output messages box.

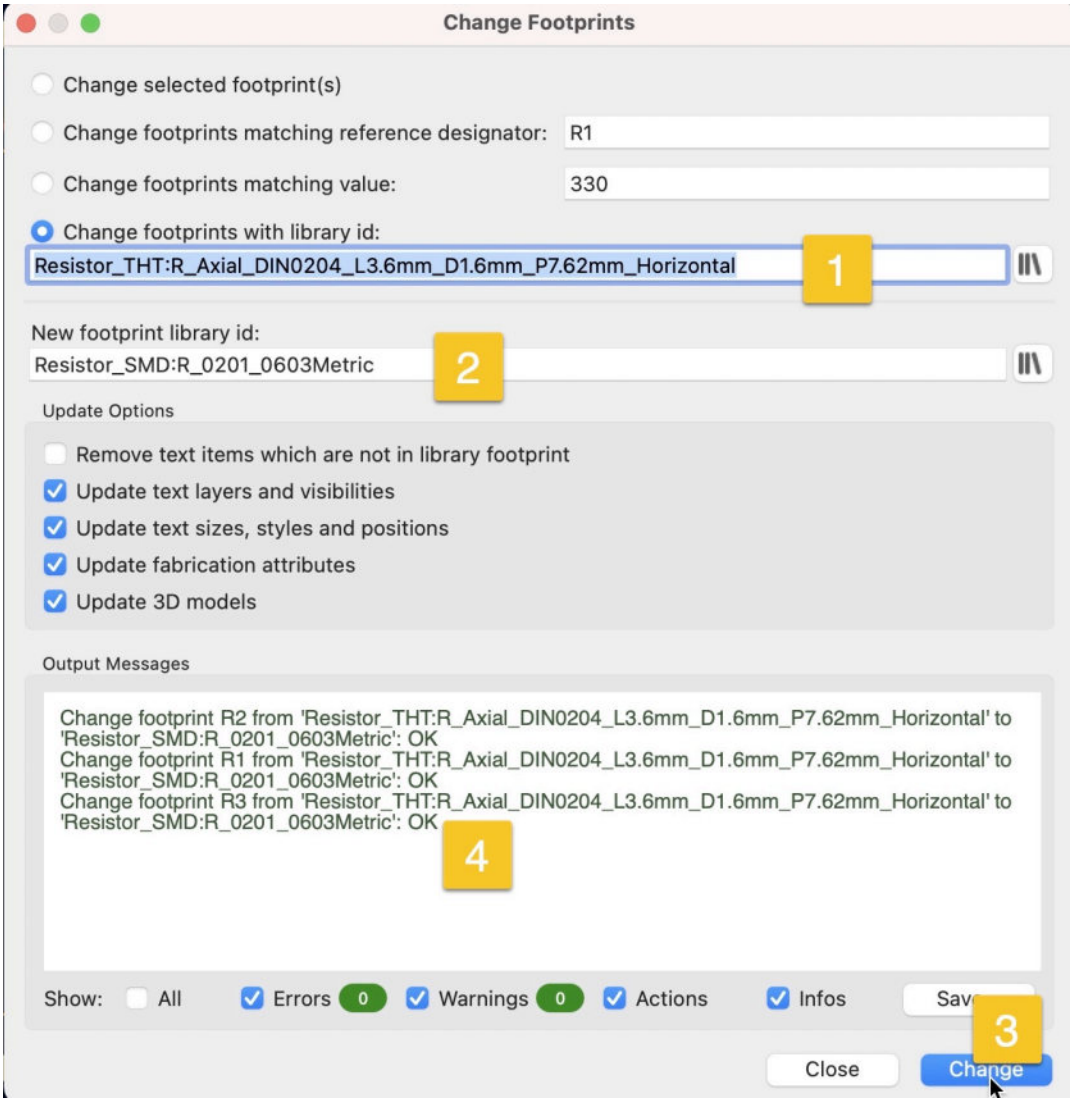


Figure 13.2.2.9: Changing footprints.

Click Close to dismiss the window, and look at the results in the editor:

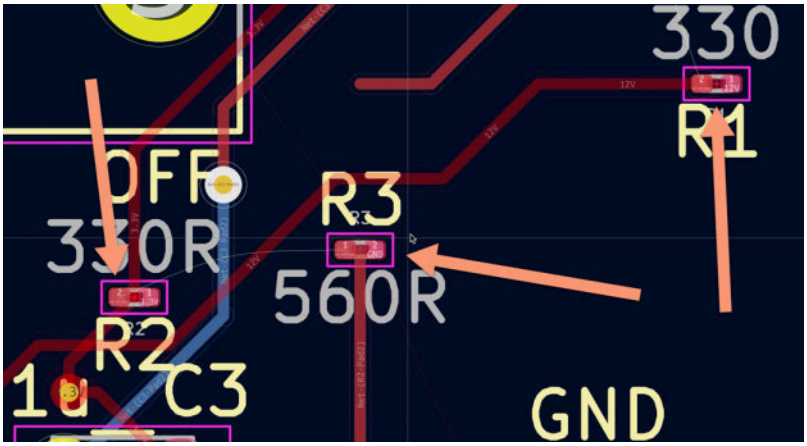


Figure 13.2.2.10: Changed footprints.

The original TH footprints are now SMD. You will need to adjust the copper tracks and finish the electrical connections. Notice that the ratsnest lines have appeared and can help you with the drawing of the tracks.

With the help of the Change Footprints tool you will be able to easily change footprints.

3. Interactive delete

Deleting unwanted elements of a schematic or layout always contains an amount of risk. Yes, you can undo an incorrect deletion or restore from a backup if things have gone wrong. But why make a mistake in the first place? In health and medicine, prevention is better than any cure, and the same applies to computer-aided design.

In KiCad, you can delete an element, such as a text item, a symbol, footprint, or graphic, by hovering the mouse pointer over the component and pressing the Delete key. You can also right-click on an element to bring up the contextual menu and then select Delete.

The problem with these methods is that it is not always obvious what you are deleting when you are working in busy schematics or layouts.

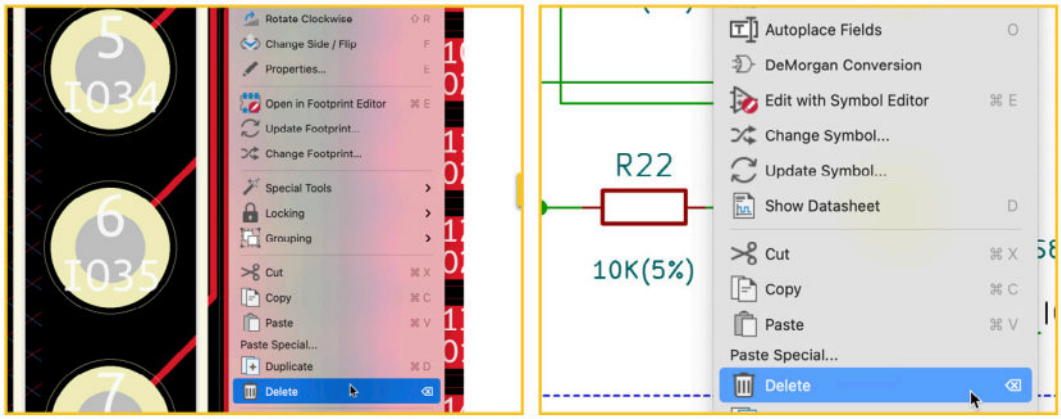


Figure 13.3.1: Deleting something in Pcbnew (left) and Eeschema (right) entails a level of risk.

In Figure 13.3.1 (above), I am using the context menu to delete an unwanted element. In Pcbnew (left), I wanted to delete a track. However, the header footprint was selected instead of the track because of imprecise targeting and the active selection filter settings. You can see that Pcbnew highlighted the header footprint with white instead of regular yellow. In Eeschema, the item with its context menu activated is marked with a fuzzy halo around it. It is easy to make mistakes and delete the wrong thing in both cases, especially in busy editors.

If you use the hover and Delete keypress option, you don't even get an indication of precisely what you are about to delete.

To make it safer to delete schematic or layout elements, KiCad provides the Interactive Delete tool. This tool is available in both Eeschema and Pcbnew. You can find it in the Edit menu and at the bottom end of the right toolbar (Figure 13.3.2).

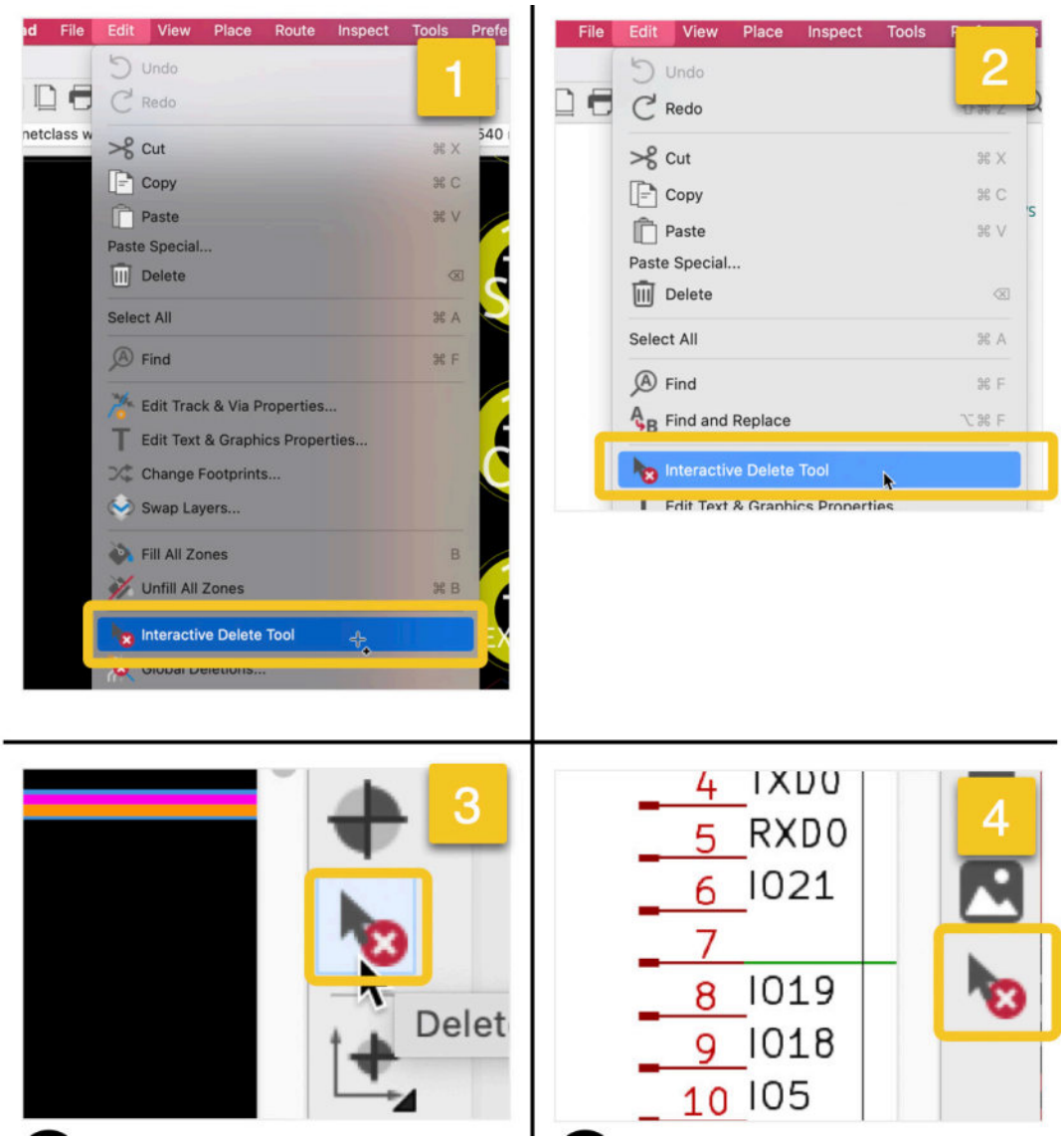


Figure 13.3.2: You can find the Interactive Delete tool in the Edit menu and at the bottom end of the right toolbar. Pcbnew (1) and (3), Eeschema (2) and (4).

When you select the Interactive Delete tool, the cursor changes to a rubber eraser with a small cross. Anything that the cross hovers over is highlighted with a configurable color (you can customize all colors via the Preferences window; then Schematic Editor —> Colors and PCB Editor —> Colors). You can see an example below (Figure 13.3.3).

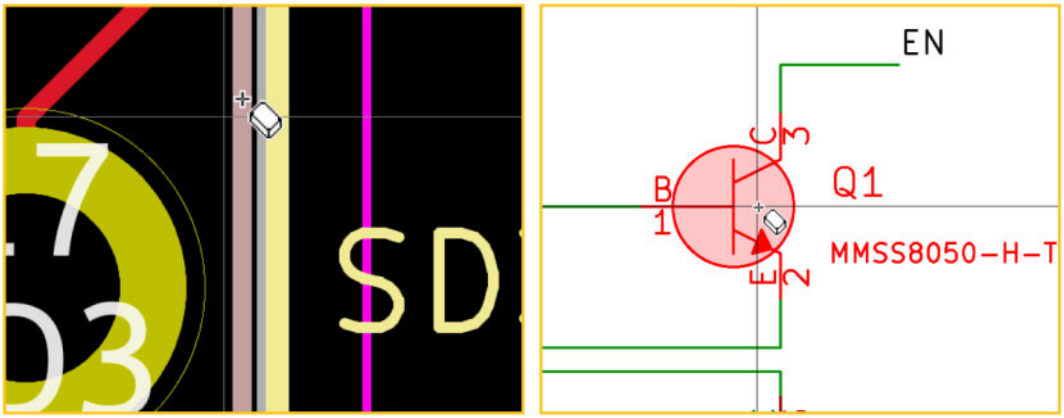


Figure 13.3.3: An example of the Interactive Delete tool. Pcbnew (left), Eeschema (right).

In the layout editor, the Interactive Delete tool will work regardless of the settings of the Selection Filter, but it will not allow you to delete locked items.

4. Find and Replace (Eeschema)

In Recipe “2.1. Change a symbol in bulk”, you learned how to quickly change symbols assigned to a component based on common criteria, like the library designator or the value. KiCad offers more bulk-editing tools and dedicates one of them to text in the Schematic Editor.

You can access this tool from the Edit menu. Click on Edit, then click on Find and Replace.

Let’s look at an example where the Find and Replace tool is useful. I have a schematic that contains several instances of the “RESET” net label. This label exists over two different pages of the schematic. See the images below (Figure 13.4.1).

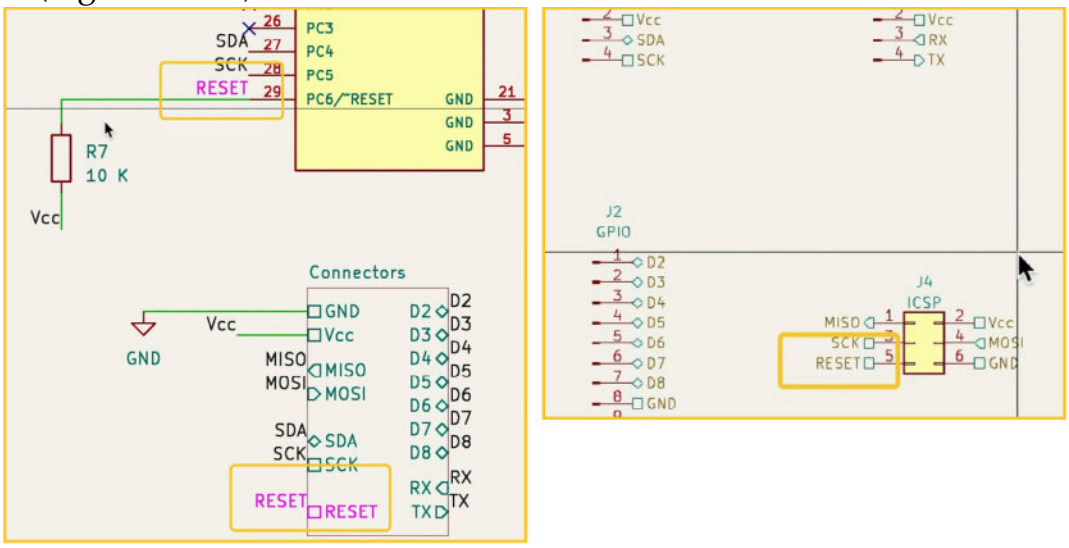


Figure 13.4.1: The net label “RESET.” I’ll change it to “RST.”

I would like to change “RESET” to “RST” using the Find and Replace tool.

Bring up the Find and Replace tool by clicking on Edit —> Find and Replace. In the “Search for” field, type “RESET,” and in the “Replace with” field, type “RST” (Figure 13.4.2).

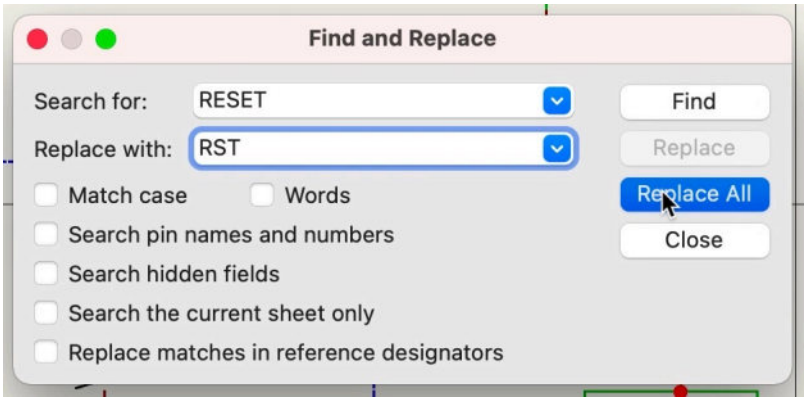


Figure 13.4.2: The Find and Replace window ready to get to work.

Click on "Replace All." The Find and Replace window will not go away so that you can search and replace more text if you want. You can still pan and zoom in on the schematic to confirm the changes. Click the Close button to dismiss the Find and Replace window.

In the schematic, you can now see "RST" in all net labels that previously contained "RESET" (Figure 13.4.3).

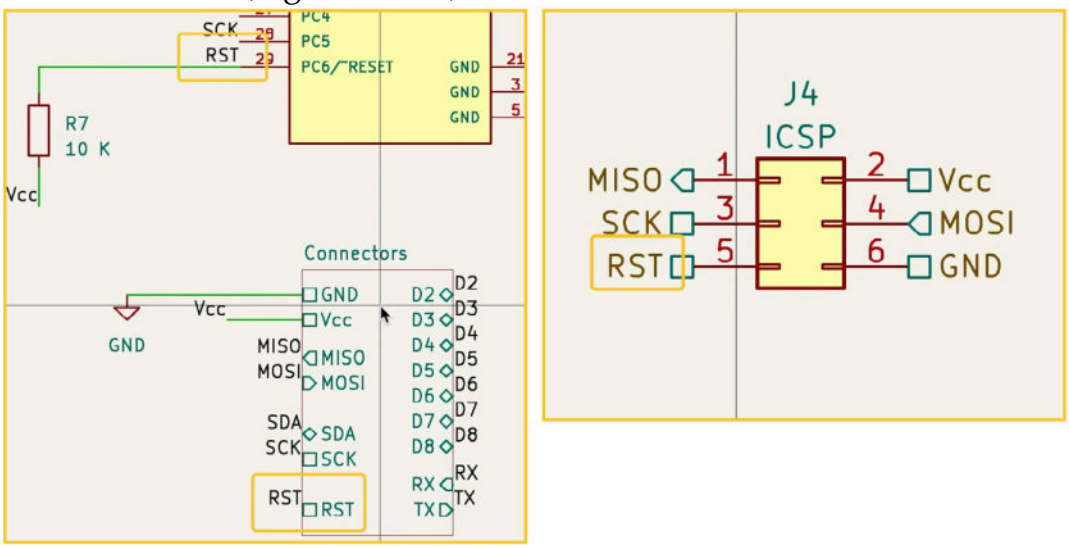


Figure 13.4.3: The net label "RESET" has been changed to "RST."

You can use this tool to find and change any text type, whether a net label or a text item, as long as it appears within the schematic. Text that appears in the schematic page settings area (in the bottom right corner of the page) is not affected.

5. Edit Text & Graphics Properties

If you want to make bulk changes to text or graphic element properties in the schematic or layout editors, you can use the Edit Text & Graphics Properties tool.

You can access this tool via the Edit menu. Click on Edit, then click on Edit Text & Graphics Properties (Figure 13.5.1).

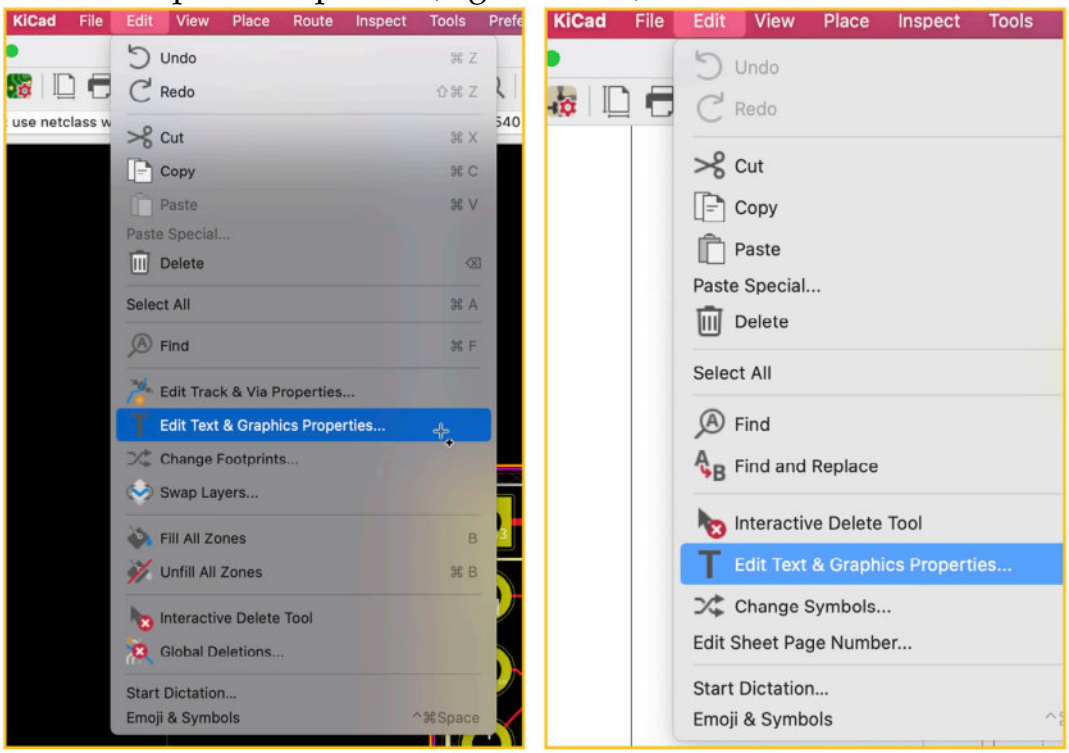


Figure 13.5.1: How to access the Edit Text & Graphics Properties tool in Pcbnew (left) and Eeschema (right).

The tool looks different in Pcbnew compared to Eeschema, but it works in the same way in both variations.

First, set the scope. The scope controls the type of elements that the tool will target. For example, in Eeschema, you can target global labels, wire labels, and values. In Pcbnew, you can target values and reference designators.

Second, set the filters. With the filters, you can define a sub-group within the already set scope. For example, suppose you are working on the layout. You have selected the reference designator scope. Then, in the filter, you can narrow the targets to only include reference designators in the top copper layer.

Third, define the change that you want to insert. Here is an example: Suppose you work in the schematic editor and select the “values” scope. In the “Set to” group (in Pcbnew, the same group is named “action”), you can change the size, orientation, and other aspects of the text used to show component values to 10mm, Right and Bold.

Below you can see what the Edit Text & Graphics Properties looks like for Pcbnew (left) and Eeschema (right) (Figure 13.5.2).

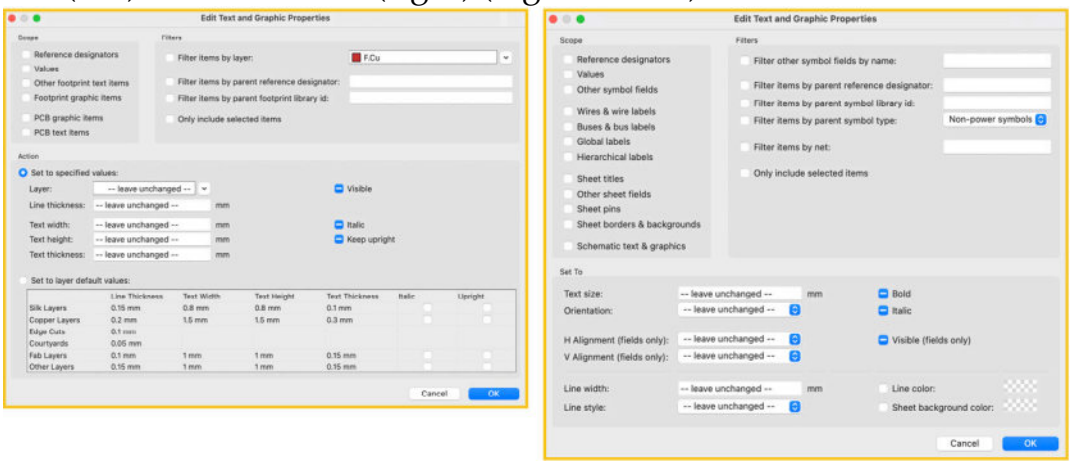


Figure 13.5.2: The Edit Text & Graphics Properties window in Pcbnew (left) and Eeschema (right).

Let’s look at an example. Say that in the schematic below, you wish to change the color and thickness of the wires. At the start, wires have the default thickness and color, and the schematic looks like this (Figure 13.5.3):

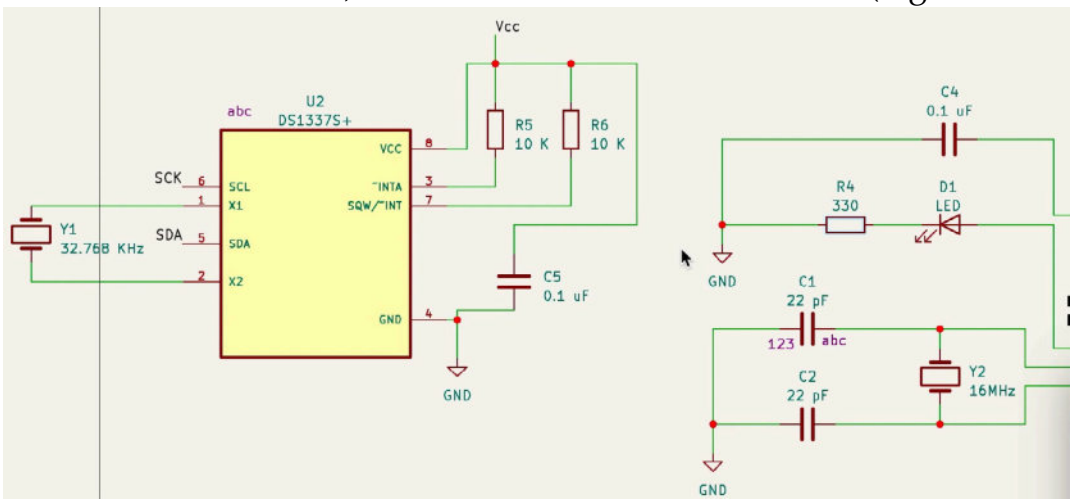


Figure 13.5.3: Wires drawn with their default color and thickness.

Bring up the Edit Text & Graphics Properties window, and set the following:

1. Scope: Wires & wire labels.
2. Filters: leave defaults.
3. Set to:

- Line width: 2mm
- Line style: Dotted
- Line color: red

The window looks like this (Figure 13.5.4):

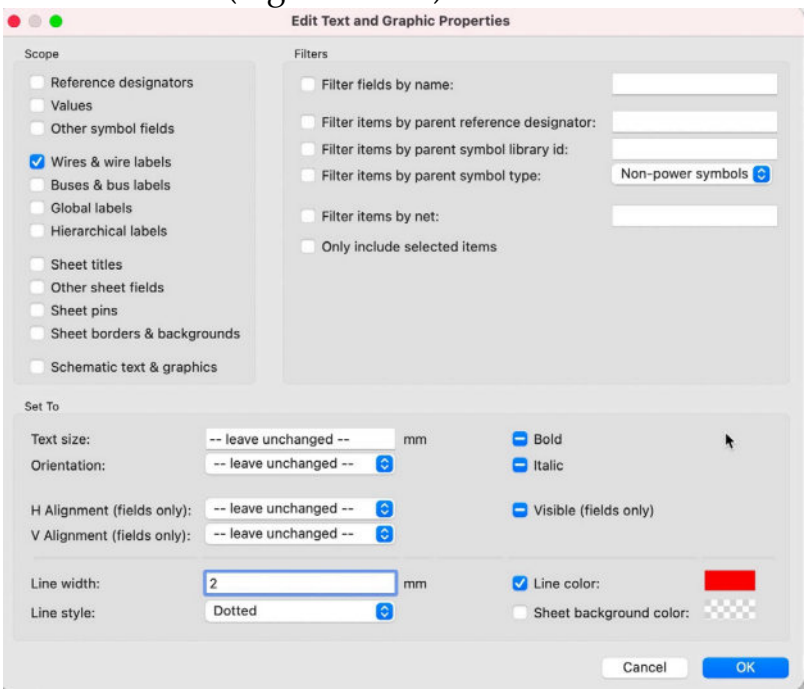


Figure 13.5.4: I'm changing the "look" of the wires in my schematic.

Then, click OK.

The schematic now features huge dotted red lines that represent the wires (Figure 13.5.5):

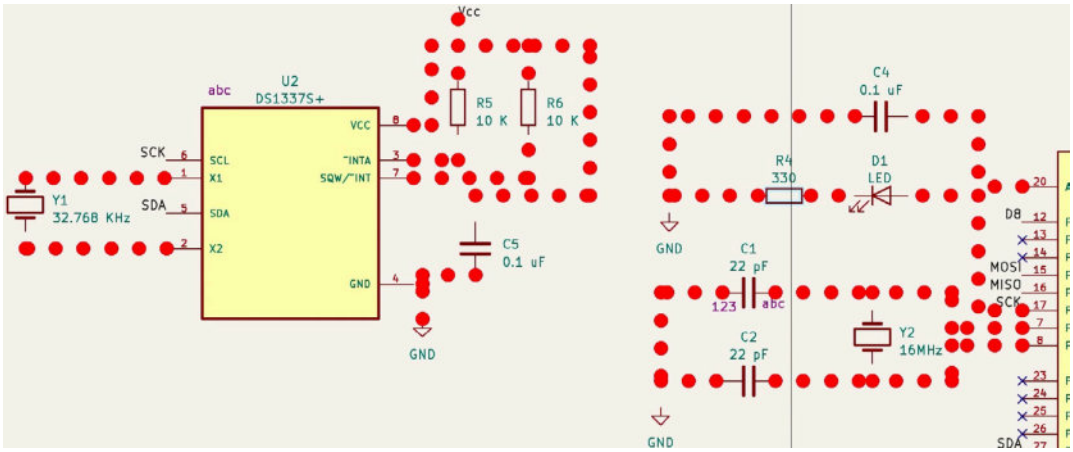


Figure 13.5.5: The default look of all wires is changed to this red dotted style.

Of course, the way that wires now look is not particularly helpful. You can undo this change by typing Cmd-Z (Mac OS) or Ctr-Z (Windows, Linux). A more helpful change of wire characteristics would be using the filter and allocating different colors to important nets using the Filters. One of the filters allows you to do exactly that ("Filter items by net").

6. Edit Track & Via Properties (Pcbnew)

In Pcbnew, you can edit track and via properties, in bulk, similar to how you can edit text and graphics properties in both Pcbnew and Eeschema (see recipe "5. Edit Text & Graphics Properties").

You can open the Edit Track & Via Properties window via the Edit menu item in Pcbnew (Figure 13.6.1).

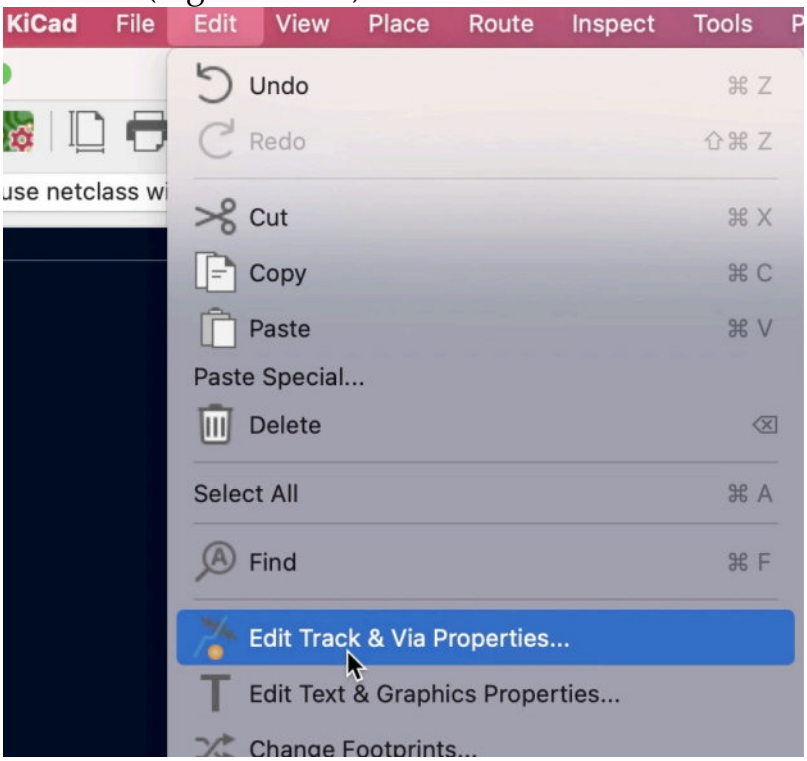


Figure 13.6.1: Bring up the Edit Track & Via Properties window.

It looks like this (Figure 13.6.2):

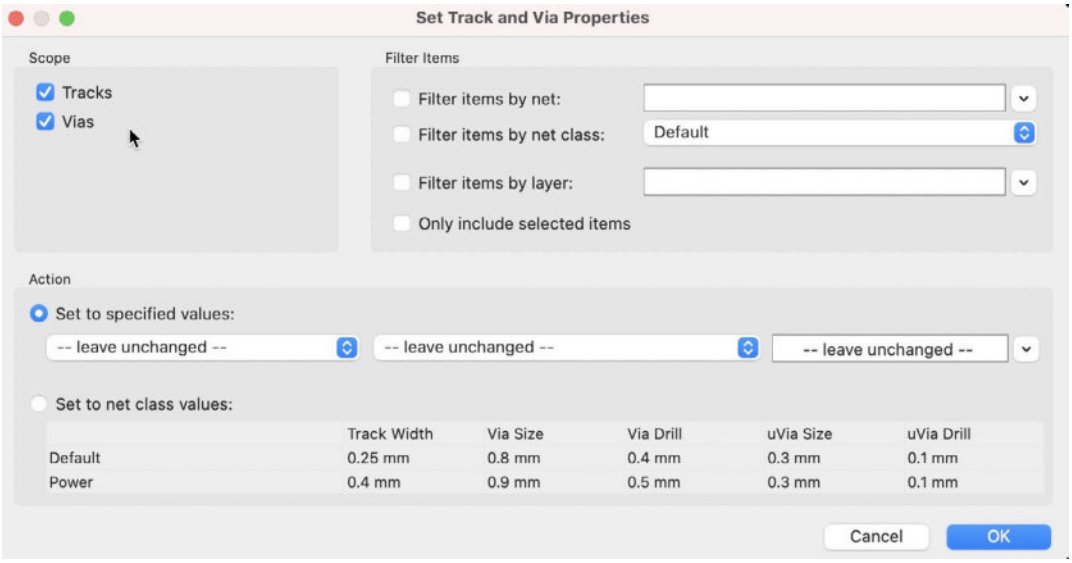


Figure 13.6.2: The Edit Track & Via Properties window.

If you have used the Edit Text & Graphics Properties tool, you will find Set Track and Via Properties familiar. There are three groups of widgets: Scope, Filter Items, and Action.

In Scope, you set the kind of item(s) you want to change. In Filter Items, you can narrow down the items that you want to change based on their net, class, or layer memberships. You can also use your mouse to click and select items manually. Finally, in Action, you can specify the changes to be implemented.

Let's work through a simple example.

Say you have a layout where tracks that belong to the GND net have a default width of 0.25mm. Below, you can see a segment of the layout (Figure 13.6.3):

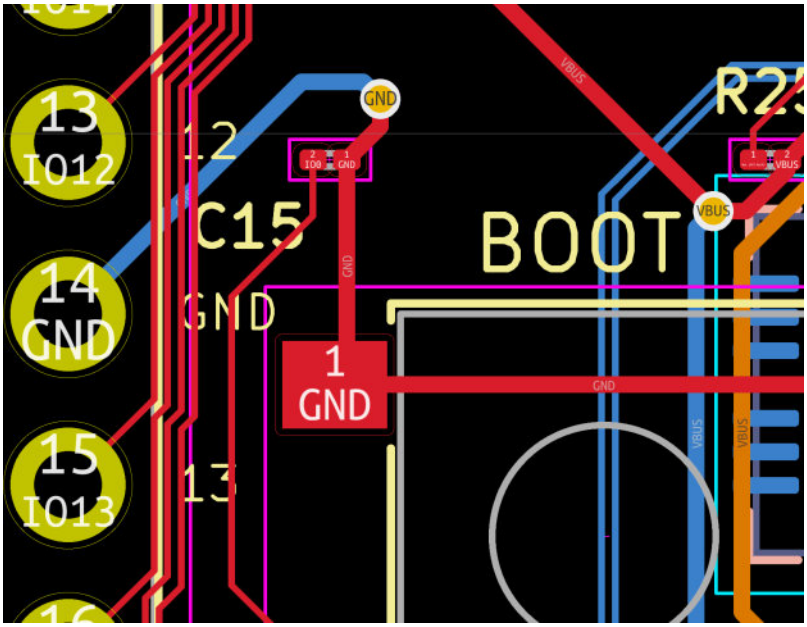


Figure 13.6.3: The default width of GND tracks is 0.25mm.

I want to change all GND tracks so that they are 0.50 mm. I know this is excessive, but I'd like the change to be visually apparent for the sake of this example.

To make this happen, we'll use the "Set Track and Via Properties" tool.

Before that, however, I need to set up the custom track sizes that you'd like to use in the Board Setup window. I show this in a dedicated chapter (see "How to create custom sizes for tracks and vias"). For this example, I have set two pre-defined track widths: 0.25mm and 0.50 mm (Figure 13.6.4):

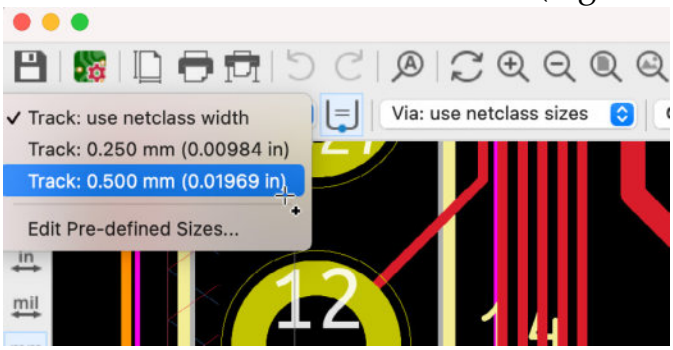


Figure 13.6.4: I have set a couple of pre-defined track widths; I will need them in the Set Track and Via Properties window.

Continue by bringing up the Set Track and Via Properties window. Set the options like this:

- Scope: Tracks.
- Filter items: Filter items by net: GND.
- Action: Set to specified values: Track: 0.500 mm (0.01969 in).
- Everything else should be unchecked or "leave unchanged."

Here's what the window should look like (Figure 13.6.5):

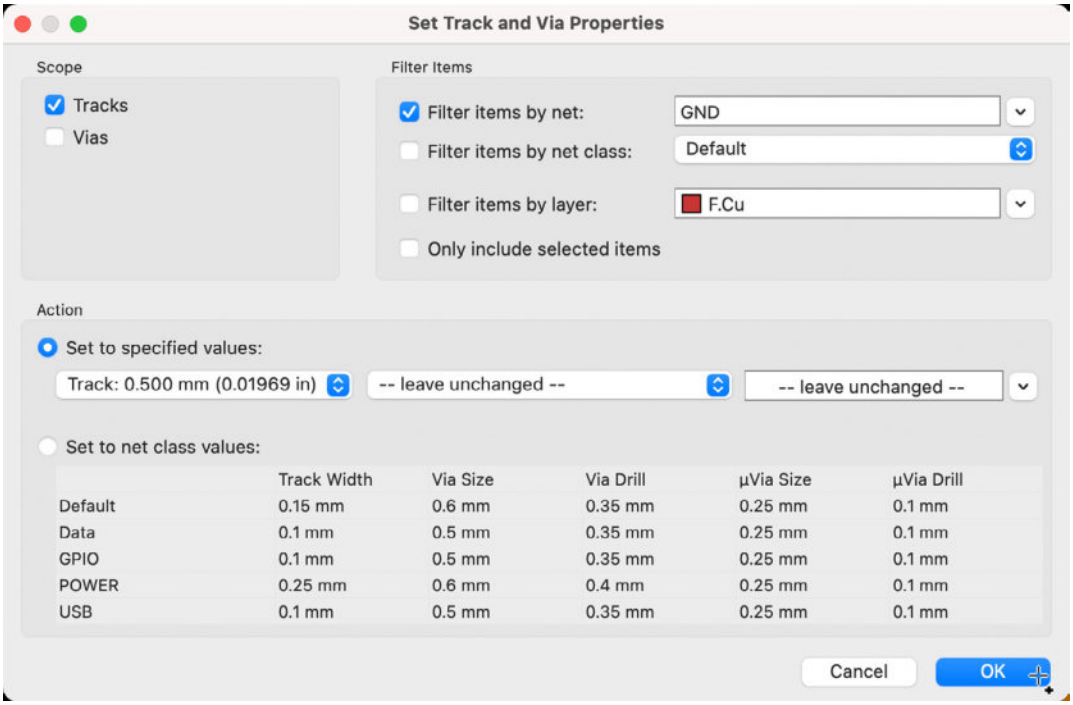


Figure 13.6.5: I'm changing the width of all tracks that belong to the GND net to 0.50 mm.

KiCad will take a moment to apply the change and will redraw all tracks that belong to the GND net to be double their original width (Figure 13.6.6):

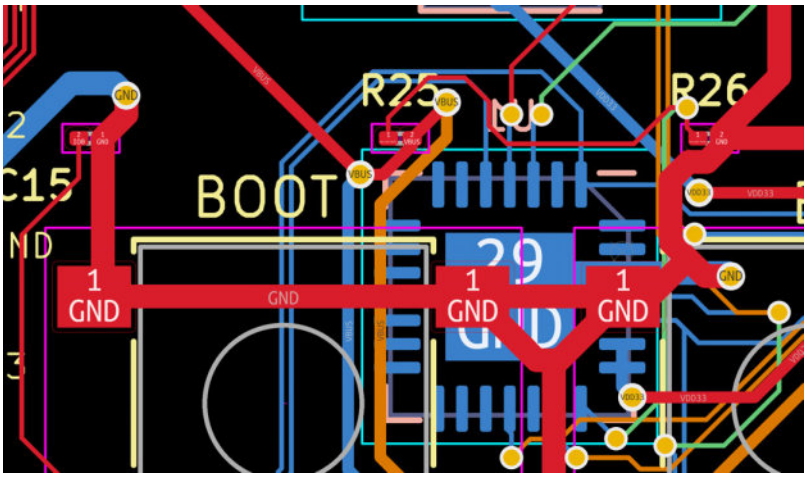


Figure 13.6.6: All tracks that belong to the GND net have a new width of 0.50 mm.

All tracks that belong to the GND net now have a width of 0.50 mm.

You can use the same tool to change properties of vias and update via and tracks with the values of a revised or added net class.

7. Text variables

A new feature in KiCad 6 is Text Variables. With Text Variables, you can create variables that contain any text string, which you can use in a schematic or layout to parametrize anything "text."

For example, you can set a Text Variable to hold a version number. In your Eeschema and Pcbnew designs, you can reference this variable so that it always shows its current value. As you are working through new versions of your PCB, you will be able to update the version number at a single location (the text variables window) instead of looking for the various locations in the schematic and layout editors when the version number may appear.

Let's look at an example.

Start by creating a Text Variable. There are two places where you can do this. One is via the Schematic Setup dialog box in Eeschema, and the other is in the Board Setup in Pcbnew. They are under differently-named groups, but they are the same thing. When you create a Text Variable in Eeschema, it will also appear in Pcbnew and vice-versa.

In Eeschema, click on File, then click on Schematic Setup. Under Project, click on Text Variables (Figure 13.7.1). At the bottom of the Text Variables list, click on the "+" button to create a new row (1). For the variable name, enter "project_name" (2), and for the text substitution, enter "Breadboard power supply" (3). When ready, click on OK to finish (4).

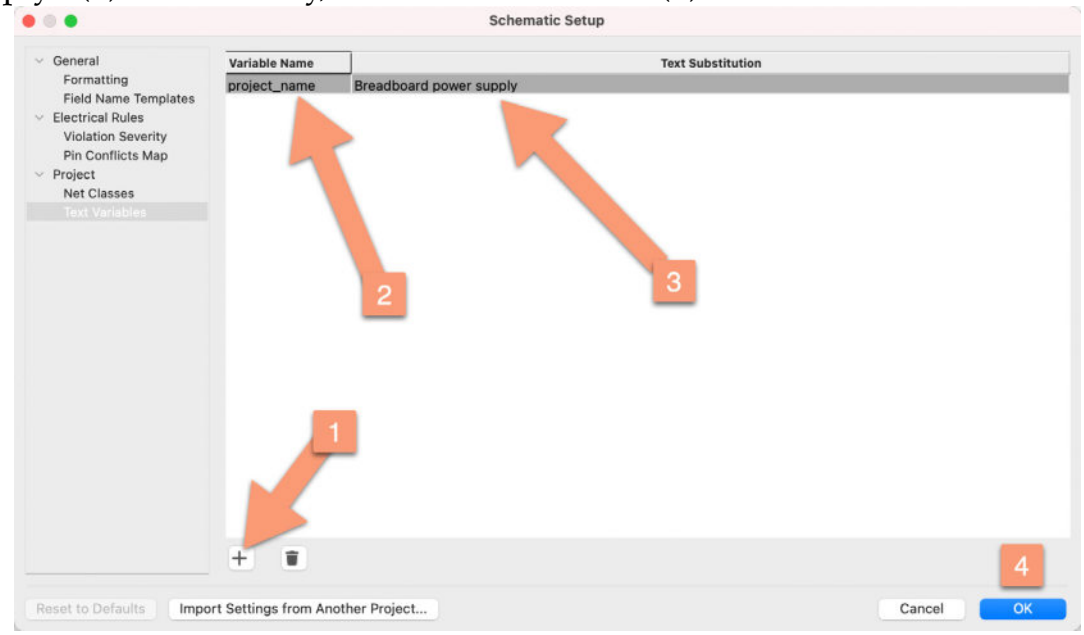


Figure 13.7.1: A new Text Variable.

Try the same in Pcbnew. Click on File, then Board Setup. Under "Text & Graphics," you'll see "Text Variables." Click on that to show the variables list. Notice that the variable you created in Eeschema also appears here (Figure 13.7.2).

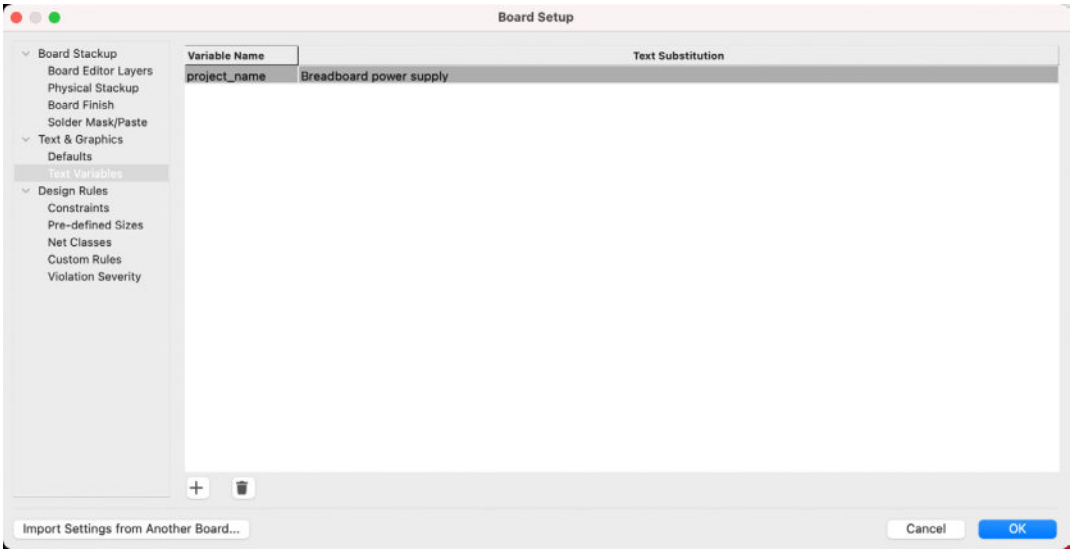


Figure 13.7.2: The new text variable that was created in Eeschema also appears in Pcbnew.

Next, create a few references to the new variable using its name. For example, let's use this variable in the Eeschema Page Settings, which appear in the editor sheet's bottom right corner. Bring up the Page Setting window (File —> Page Settings), and enter the variable reference "${project_name}" (without the double quotes) in the Title field (Figure 13.7.3):

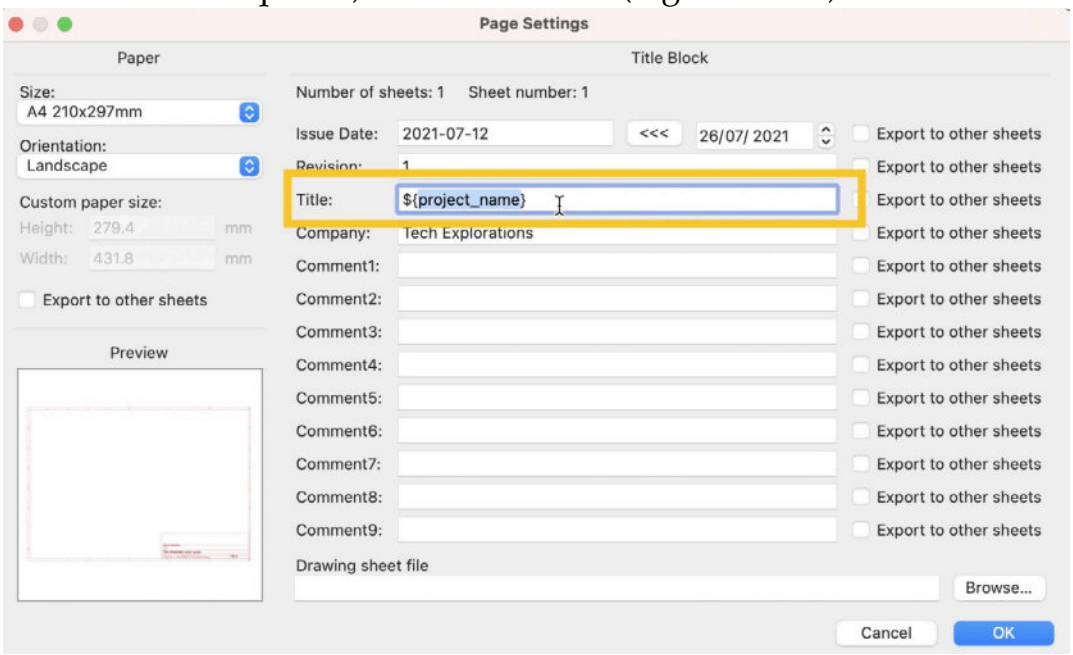


Figure 13.7.3: The variable reference used in the Page Settings.

Click OK to exit the Page Settings window. Pan the editor to the bottom right corner so you can see the Sheet title. It should contain the value of the "project_name" text variable (Figure 13.7.4):

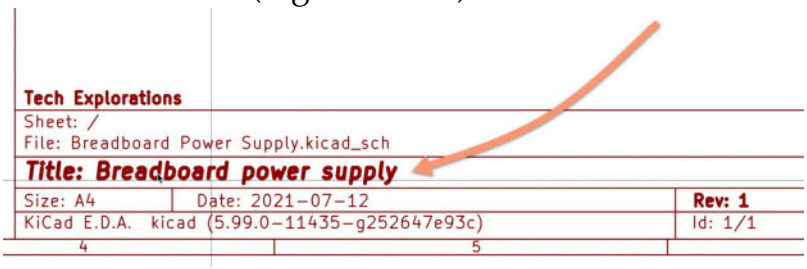


Figure 13.7.4: The variable reference is substituted for its value.

As you can see, the variable reference was substituted by its value.

Let's use the same text variable in Pcbnew. Go to Pcbnew, and click on File —> Page Settings. In the Title field, type the text variable reference "${project_name}" (without the double quotes) (Figure 13.7.5).

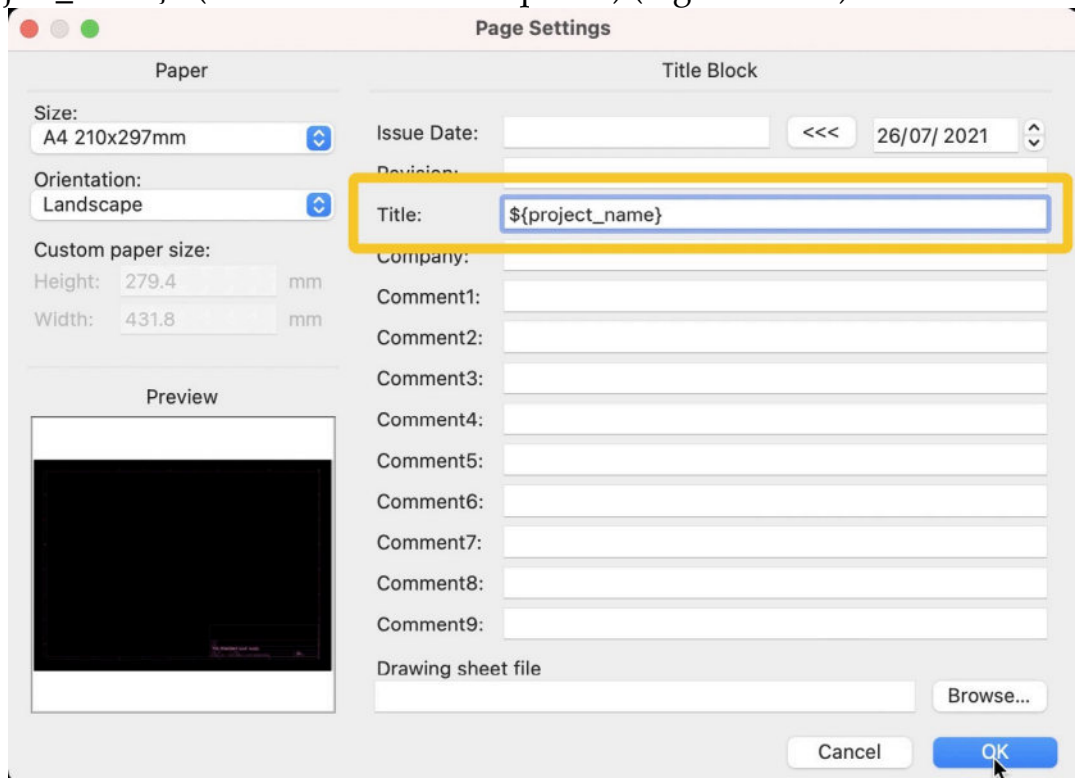


Figure 13.7.5: The variable reference used in the Page Settings in Pcbnew.

Click OK to exit the Page Settings window. Pan the editor to the bottom right corner so you can see the layout Sheet title. It should contain the value of the "project_name" text variable (Figure 13.7.6):

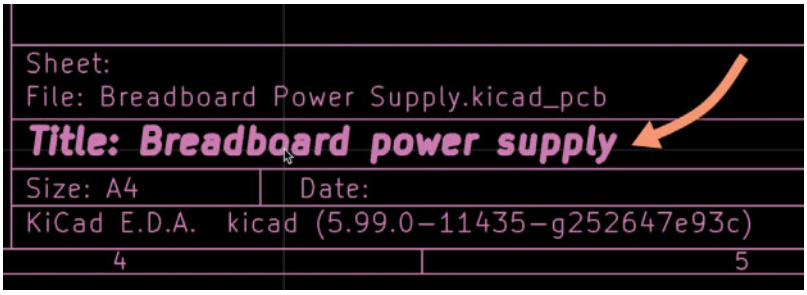


Figure 13.7.6: The variable reference is substituted for its value.

Let's try something else. Let's create a new text variable and use it in a text item in Eeschema.

First, open the Schematic Setup window and create a new variable named "input_circuit_name". For the value, enter "Input power and 5Volt subcircuit" (Figure 13.7.7).

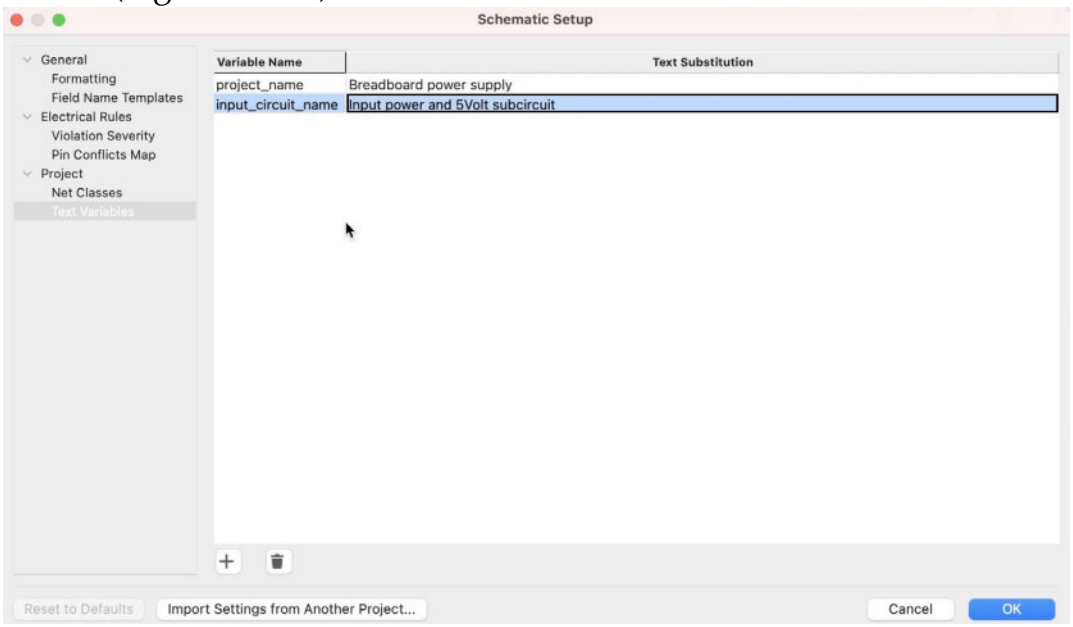


Figure 13.7.7: A new text variable.

Click OK and return to the editor.

Create a new text item. In the Text properties window, enter a text string containing both the variable reference and a fixed string. Notice that as you type the "{"to start typing in the text variable name, KiCad will give you a list of valid variable names. You can continue typing or choose one of the offered names (Figure 13.7.8).

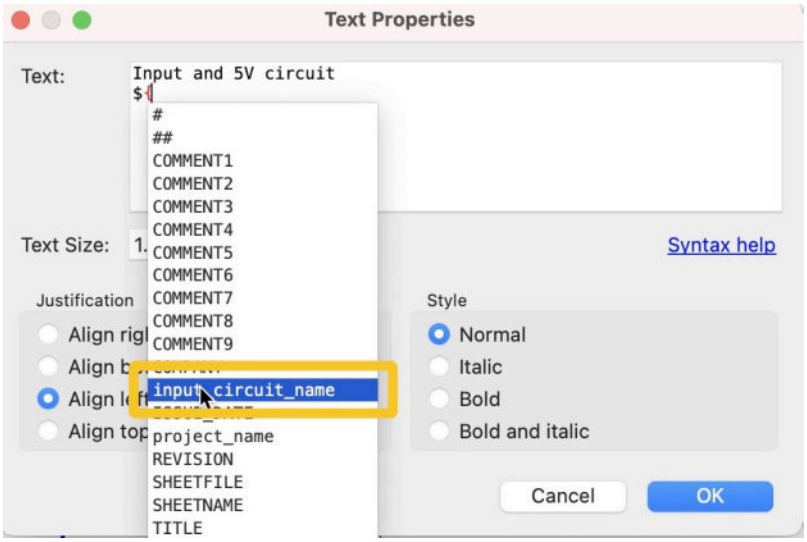


Figure 13.7.8: Code completion works with text variables.

You can see the contents of the Text field below (Figure 13.7.9):

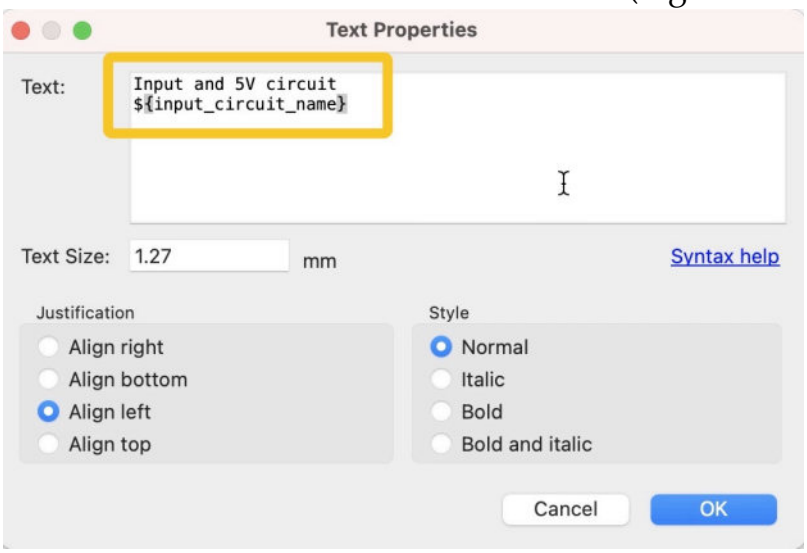


Figure 13.7.9: Example that combines fixed text with a text variable.

Click OK, and see the result of the text variable substitution in the schematic editor (Figure 13.7.10):

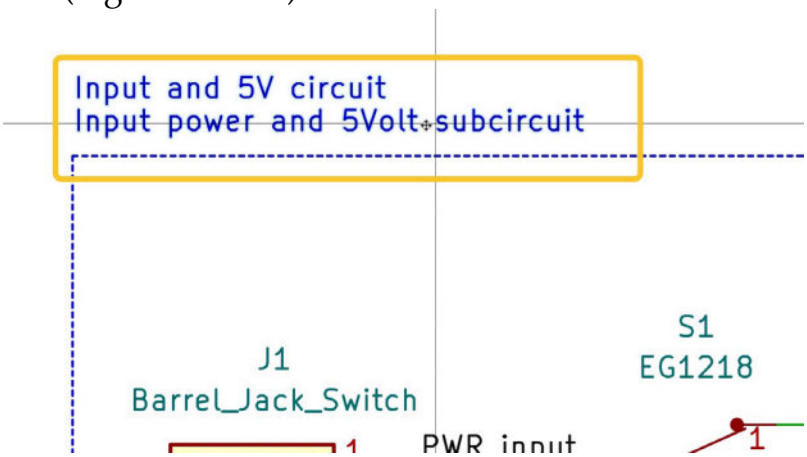


Figure 13.7.10: A text item with a text variable.

Let's see what happens if you make a change to the value of an existing text variable. Change the value of the "input_circuit_name" to something else, like this (Figure 13.7.11):

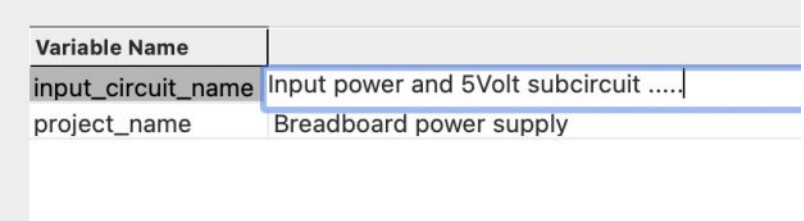


Figure 13.7.11: I have added a few dots in the text variable.

Click OK. The text item updates its content accordingly (Figure 13.7.12):

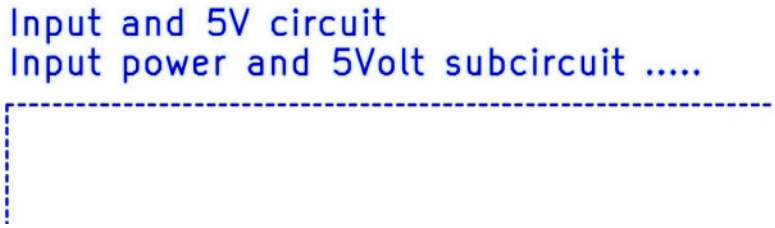


Figure 13.7.12: The changed text variable value in the text item.

You can expect the same text variables behavior in Pcbnew.

Here are some ideas of things that you can do with text variables:

- Better control of PCB version numbers that appear in the silkscreen. Instead of using fixed text version numbers, parametrize them.
- Standard pull-up or pull-down resistor values (or any other value you have a lot of in a design). Use a single text variable that makes it easy to adjust if necessary.
- URLs for PCB usage or specification information.

This is one of those features that once accustomed to; it is hard to think of KiCad without it.

8. Board Setup - pre-defined sizes for tracks and vias

In the layout editor, you can set the physical dimensions of tracks via and differential pairs in two ways:

1. Automatically, based on the net class to which these items belong.
2. Manually.

In most cases, you will want to use the automatic method. You can learn more about it in the dedicated chapter “6.3. Board Setup - Design Rules and net classes”.

The manual method gives you more freedom for tracks that do not neatly fit within an existing net class or for when you want to experiment.

When you start Pcbnew for the first time in a project, there are no pre-defined (custom) track widths or via sizes. At the top menu, you can confirm this by clicking on the two dropdown widgets on the right side (Figure 13.8.1):

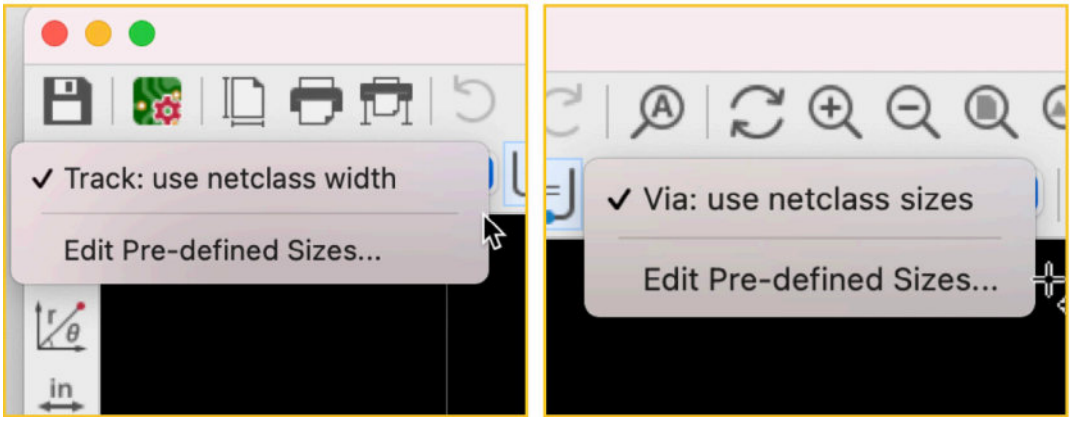


Figure 13.8.1: The track width and via sizes menus in a new Pcbnew project.

The default method for setting track and via width and size attributes is by adopting those specified in the item’s net class. As you can see in the screenshots above, the net class option is the default selection in both cases. There is no other pre-defined option. But, it is possible to create arbitrary ones that will be available through the same menu.

To create pre-defined track widths and via sizes, click on the last option of either dropdown widget, “Edit Pre-defined Sizes…”. You can also click ”File,” “Board Setup, “and then ”Pre-defined Classes.”

You will see the empty “Pre-defined track and via dimensions” list (Figure 13.8.2):

Figure 13.8.2: The pre-defined sizes tables in a new project are empty.

You can click on the "+" button at the bottom of each of the "Tracks", "Vias," and "Differential Pairs" lists.

Let's create a few pre-defined track widths. Each time, click on the "+" button at the bottom of the Tracks list. A new row will appear in the list. Click inside the new cell to set the cursor, and type the custom values, such as "0.15", "0.20," and "0.25". While I was at it, I also created custom via and differential pair sizes. Here's an example from one of my boards (Figure 13.8.3):

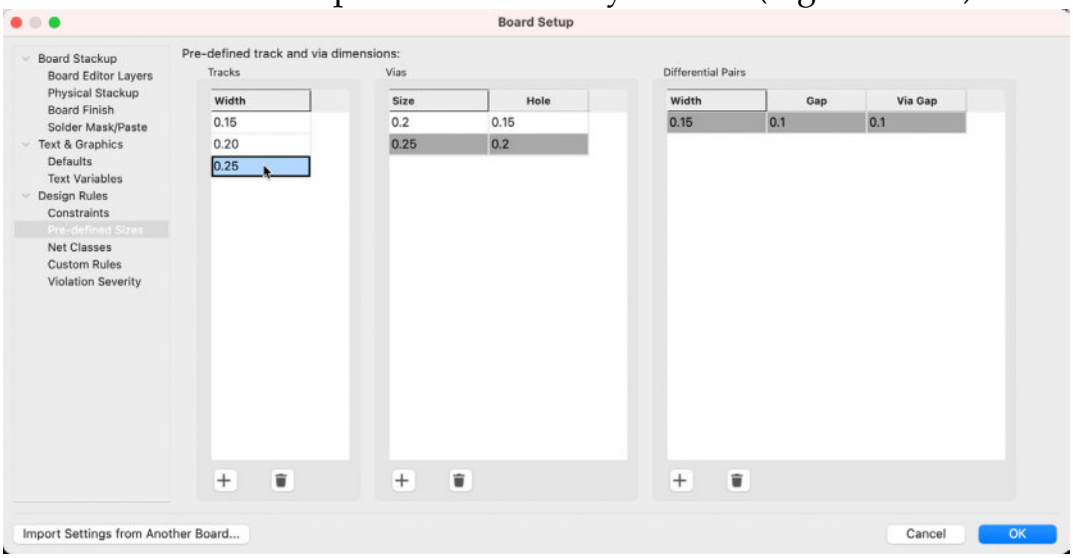


Figure 13.8.3: Pre-defined sizes for tracks, vias, and differential pairs.

When finished, click on "OK." Click on the track width and via sizes dropdown in the top menu, and notice that the new pre-defined sizes are now available for use (Figure 13.8.4):

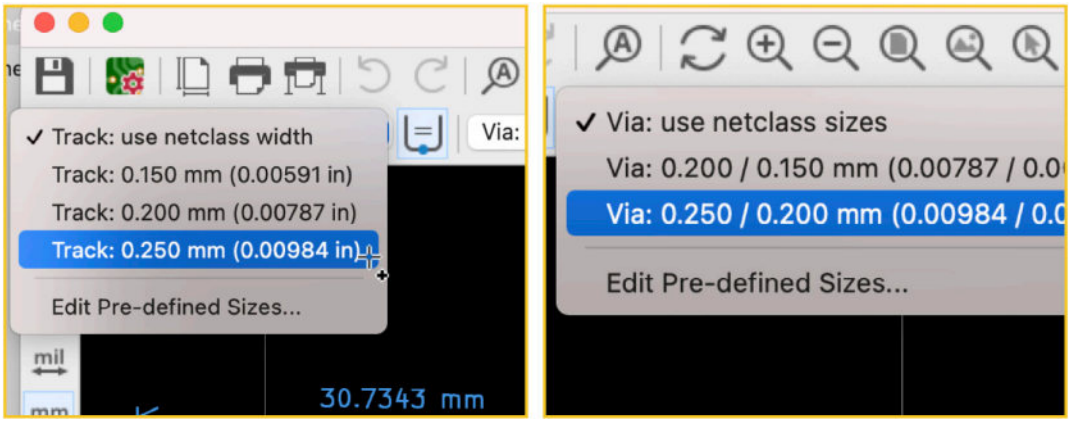


Figure 13.8.4: Pre-defined sizes for tracks and vias are now available for use.

To use the pre-defined sizes, you can use one of the following methods:

1. Select the size first, then create a new track via or differential pair.
2. With the track, via, or differential pair tool already selected, you can right-click in the editor to bring up the context menu and then select one of the available sizes from the Select Track/Via Width menu (Figure 13.8.5).

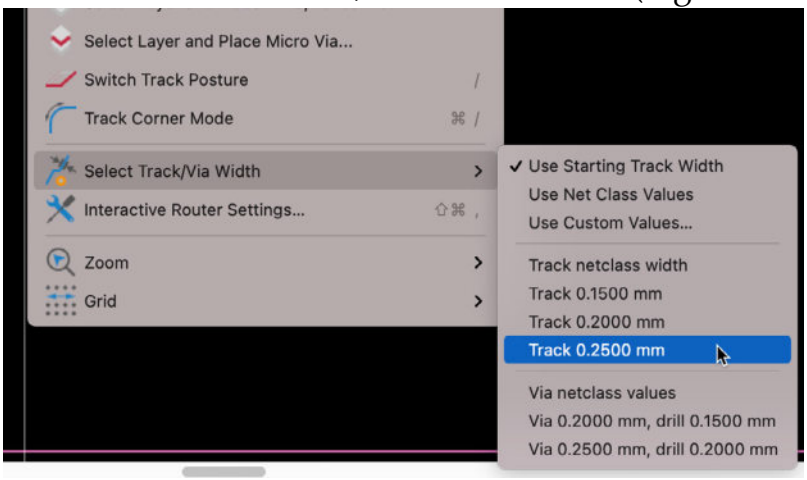


Figure 13.8.5: You can select pre-defined sizes through the context menu; this requires the track, via, or differential pairs tool to be enabled.

I used the second method to draw a single track with three different track widths (Figure 13.8.6):

Figure 13.8.6: A single track composed of segments with different widths.

A track like this would not be possible to draw using the automated net class method.

In summary, while the preferred method for defining the sizes of track and vias is to have them inherited from the net classes (I call this the "automated" method), you have the option of setting custom sizes, as you learned in this chapter.

9. Board Setup - Design rules violation severity

Pcbnew contains a checker tool that can detect defects in your layout and then classify those defects into three categories: Errors, Warnings, and Ignore. While the default settings for the classifier are reasonable, you may want to experiment with alternative settings. Pcbnew has a tool that allows you to modify the violation severity classifier. Eeschema also has its own violation severity and classifier, which you can also customize (learn about it in the dedicated chapter “Board Setup - Design Rules and custom rules”).

To access the Violation Severity settings, click on File, Board Setup, Violation Severity (Figure 13.9.1).

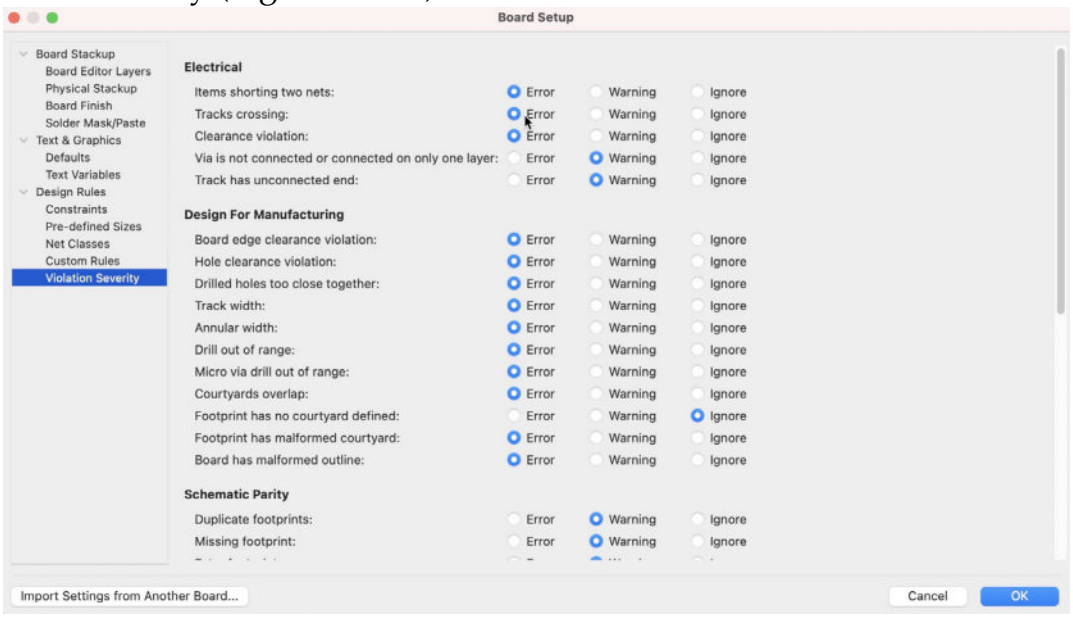


Figure 13.9.1: The Violation Severity settings in the Board Setup window.

In the Violation Severity settings, you can customize the classification of any of the conditions that the Design Rules Checker can identify. For example, you can classify “Hole clearance violation” to be an Error or a Warning.

I will explain how the Violation Severity classifier works with the help of an example. Consider the layout below (Figure 13.9.2):

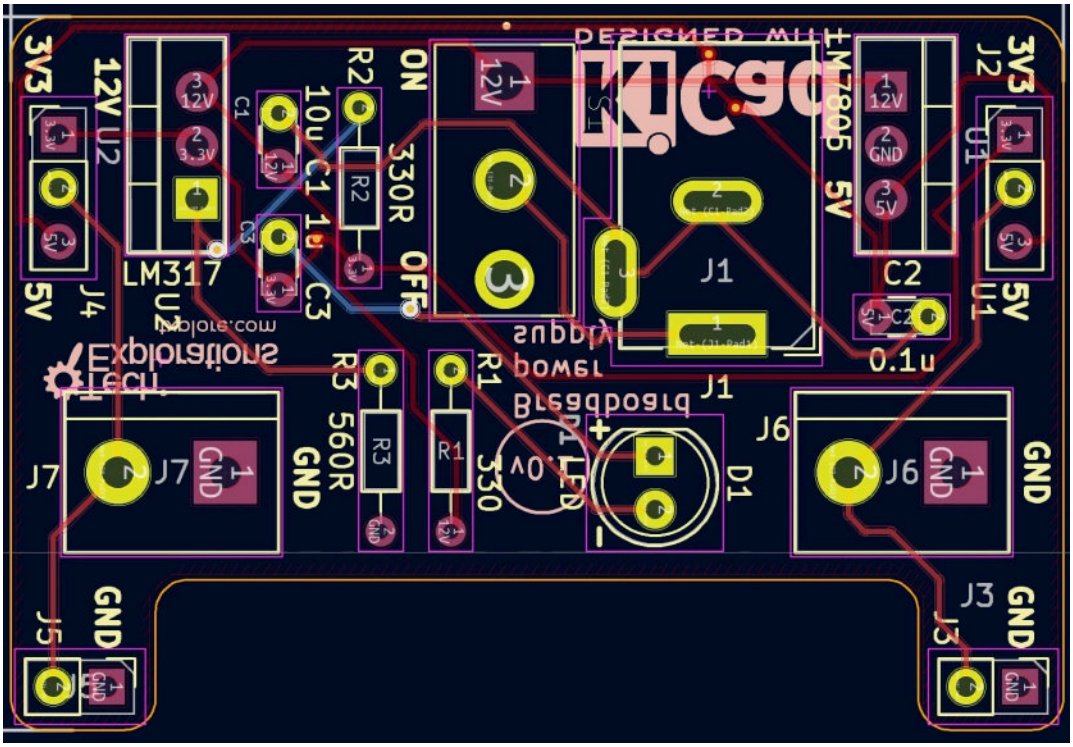


Figure 13.9.2: I'll run the Design Rules Checker on this layout.

The layout shown above comes from one of the projects in this book. The Design Rules Checker, using the default Violation Severity settings, returns three warnings, as you can see below (Figure 13.9.3):

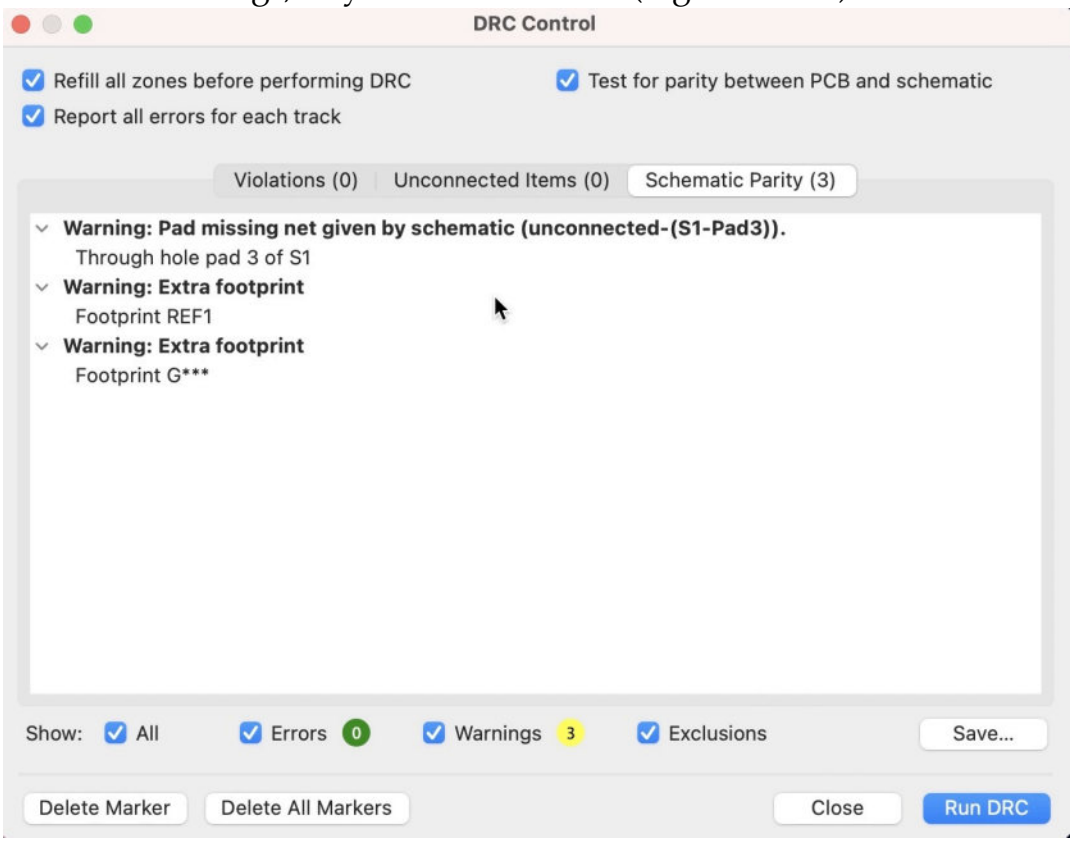


Figure 13.9.3: The DRC output using the default Violation Severity settings.

There are two warnings titled “Extra footprint” that relate to the two back silkscreen layer logos (“Tech Explorations” and “KiCad”). These logo graphics are footprints that only exist in the layout, with not schematic symbol counterpart. This warning alerts me to that fact. I can safely ignore this warning.

The second warning is about pad 3 of the slide switch footprint S1. I have intentionally not connected this pad to any nets, and the DRC warns me that it should be (connected to a net).

By default, both issues are classified as Warnings. I can change this in the Violation Severity rules so that, for example, the “Extra footprint” issue is classified as an Error.

Bring up the Violation Severity dialog, and look for the “Extra footprint” setting under “Schematic Parity.” Change the severity type to “Error.” The dialog should look like this (Figure 13.9.4):

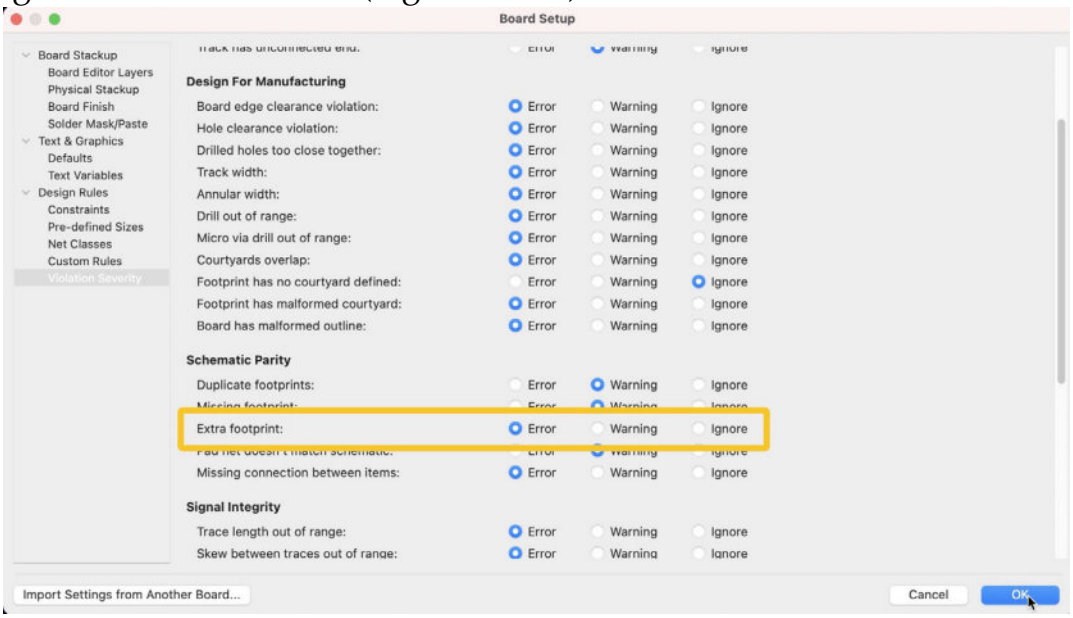


Figure 13.9.4: I have changed the violation severity type to “error” for the “Extra footprint” setting.

Next, perform a new DRC. The result is below (Figure 13.9.5):

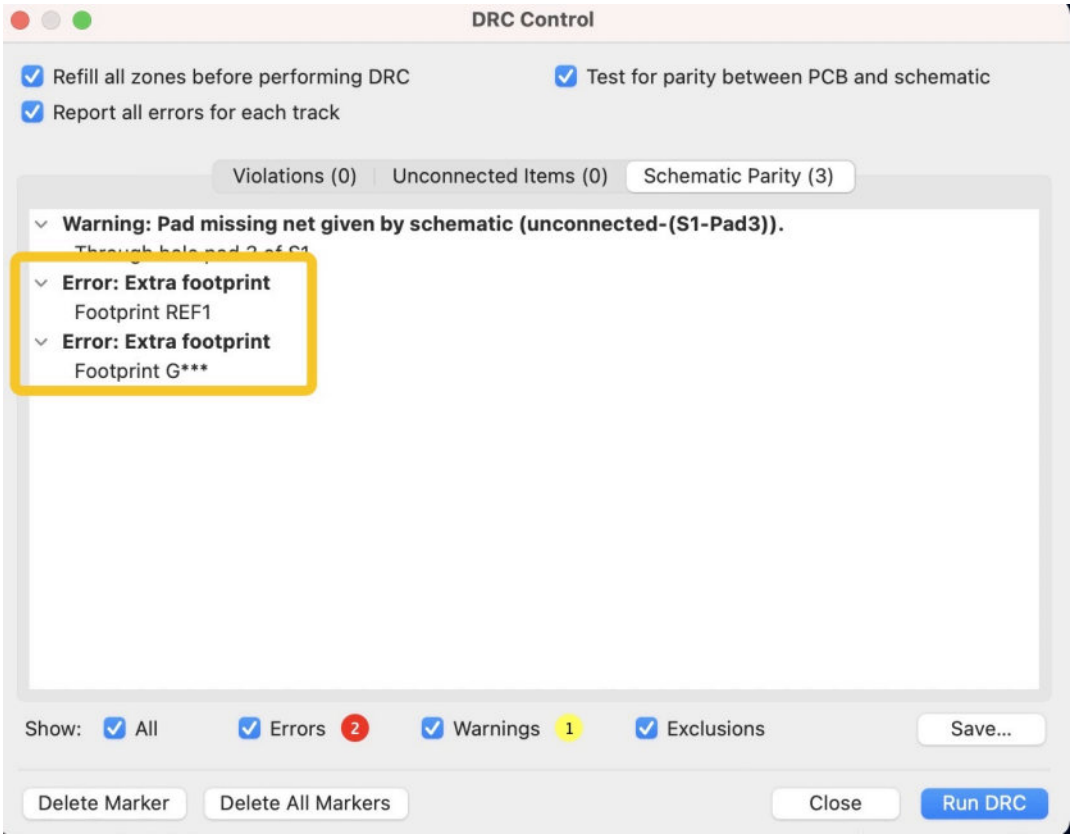


Figure 13.9.5: The two “Extra footprint” issues are now classified as “Error.”

The DRC now classifies the two “Extra footprint” condition as errors.

You can make similar changes to any of the conditions that the DRC can recognize. You can see a listing of the recognizable conditions in the Violation Severity dialog box. They are numerous, and they offer a basic level of flexibility for discovering and classifying violations.

If you need more customization power, you may consider using the Custom Rules feature under Design Rules. You can learn more about this in a dedicated chapter.

10. Board Setup - Custom design rules

Pcbnew allows you to customise the classification of design rules violations. You can learn about this in a dedicated chapter in this book. The Violation Severity dialog offer a simple way to change how the violation classifier works.

If you want a much higher level of customisability in the way that the design rules checker work, you can use the powerful "Custom Rules" feature.

Custom Rules allows you to write rules using a programmatic language. This language derives from S-Expressions, which invented for use in the Lisp programming language.

Here is an example of a custom DRC rule:

```
(version 1)

# This is an example DRC rule.
# The name of the rule is ExampleMinPowerNetClearance
# The rule sets a minimum of 1mm clearance between a
# track that belongs to the Power net and other elements.

(rule ExampleMinPowerNetClearance
   (constraint clearance (min 1.0mm))
   (condition "A.NetClass == 'POWER'"))
```

And here is how this code looks like in the Custom Rules dialog box. You can bring up this box by clicking on File —> Board Setup —> Custom Rules (Figure 13.10.1):

Figure 13.10.1: An example custom DRC rule.

The rule starts with a version number (1). You can add comments by prepending a comments line with the "#" character (2). The rule starts with the "rule" command, followed by a name. In separate lines, you can define the constraint and conditions (3). There is a custom rule validator at the bottom of the dialog box that looks for bugs in the code. You can invoke the validator by clicking on the "checklist" button on the right side of the validator output text box. In this example, my custom rule has no errors (4).

You can find detailed syntax information and rule code examples in a pop-up window that appears when you click on "Syntax help" at the top right of the Custom Rules dialog box.

When you write rules, take care to enclose each clause in parentheses, as per the S-Expressions notation.

You can define multiple conditions or constraints using booleans, or placing them in separate lines. Here is a modified example:

```
(rule ExampleMinPowerNetClearance
   (constraint clearance (min 1.0mm))
   (constraint track_width (max 2.0mm))
   (condition "A.NetClass == 'POWER'")
   (condition "A.NetClass == 'Data"))
```

This rule contains two constraints and two conditions, each in its line, which helps with clarity.

Of course, you can create more than one custom rule. As in a program in Python or Ruby, you can include multiple functions in a single program file.

In Pcbnew, you can add multiple custom rules in the DRC rules window. Here is an example:

```
(version 1)

# This is an example DRC rule.
# The name of the rule is ExampleMinPowerNetClearance
# The rule sets a minimum of 1mm clearance between a
# track that belogns to the Power net and other elements.

(rule ExampleMinPowerNetClearance
   (constraint clearance (min 1.0mm))
   (constraint track_width (max 2.0mm))
   (condition "A.NetClass == 'POWER'")
   (condition "A.NetClass == 'Data"))

# This is a second DRC rule.
# The name of the rule is ExampleMinPowerNetClearance2

(rule ExampleMinPowerNetClearance2
   (constraint clearance (min 0.5mm))
   (condition "A.NetClass == 'GPIO"))
```

The second rule is named ExampleMinPowerNetClearance2, and it sets a minimum clearance for tracks that belong to the GPIO net class.

The rule editor can do code-completion. It will automatically present a list of valid keywords as you type. Below are some examples (Figure 13.10.2):

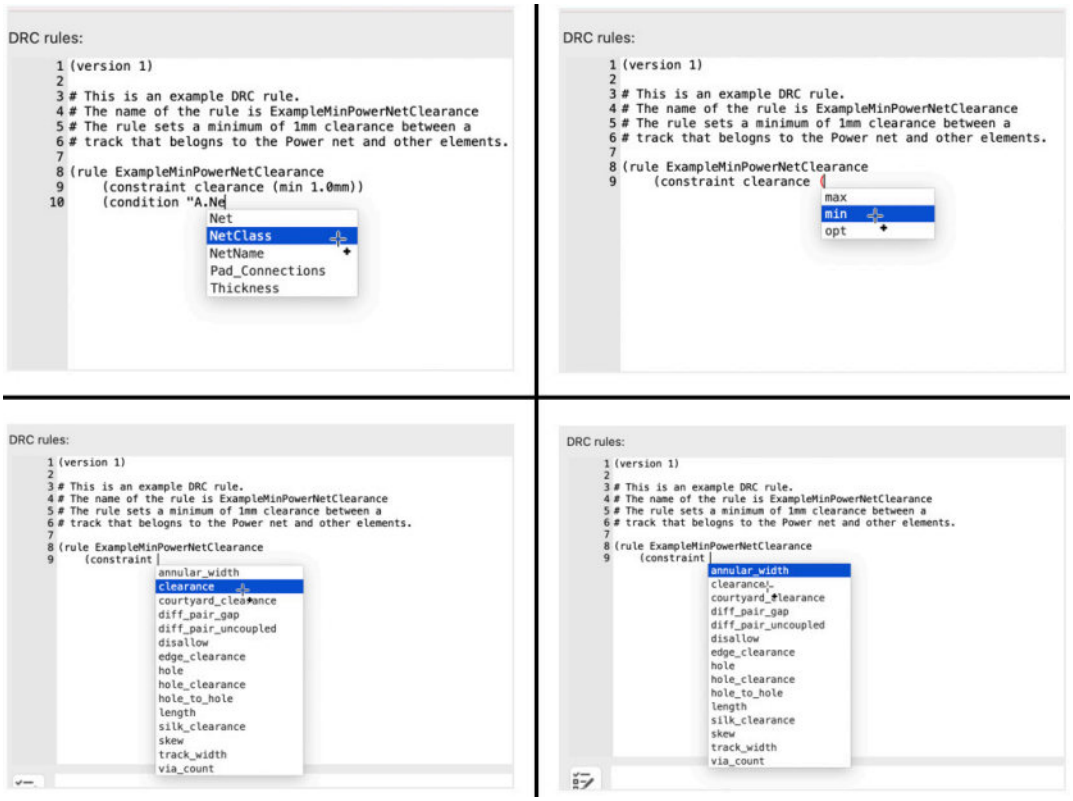


Figure 13.10.2: Examples of custom rules code completion.

I find that I can create custom rules quickly and with (usually) no errors with code completion.

Once you have a custom rule ready and validated, click the OK button to enable the rule(s). Then, test by running the DRC. Here is the output of the DRC with the custom rule that I listed at the start of this chapter (Figure 13.10.3):

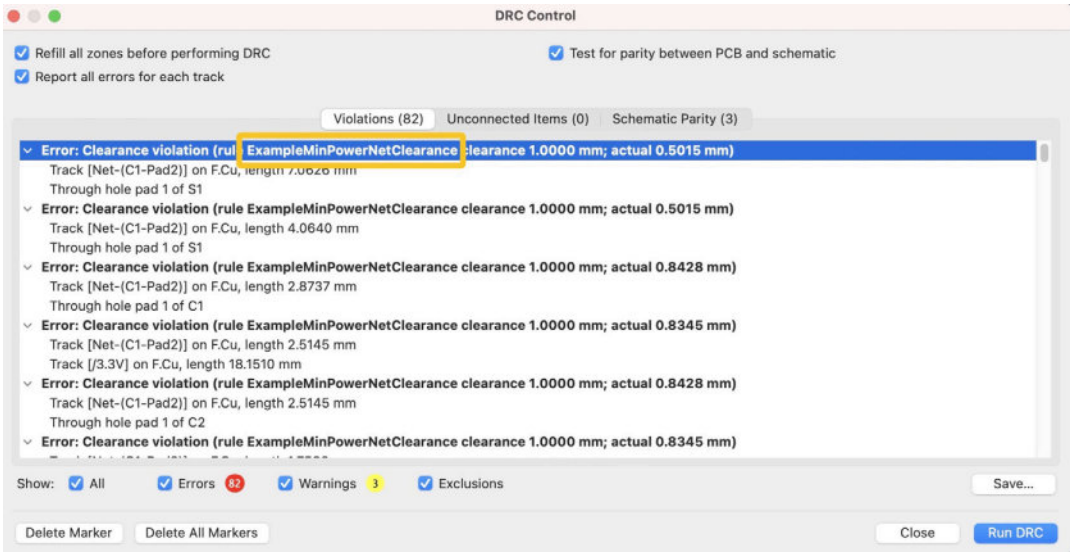


Figure 13.10.3: The custom rule has generated a lot of errors

As you can see, the DRC will use the name of the custom rule in its output so that you can identify the custom rule that fired. In this example, because the constraint that I set was so large (minimum clearance 1mm), the DRC generated 82 errors. Most of those errors are created by the custom rule.

The Custom Rule feature is powerful, and you should use it with care. Start by looking at the syntax help, and become familiar with S-Expressions. Continue by writing simple rules for your layouts so that you can quickly become comfortable with this powerful feature.

11. Schematic Setup - Electrical Rules and violation severity

Eeschema, as Pcbnew, provides a tool that checks for violations with your design. This is the Electrical Rules Checker (ERC) tool, which I have used extensively in all of the projects in this book.

Similar to the Design Rules Check (DRC) in Pcbnew, it is possible to change the default classification settings of the ERC. You can do this via the Violation Severity dialog in the Schematic Setup window. To learn about the Violation Severity configuration in Pcbnew, please look at the relevant chapter. You can access the Violation Severity dialog in Eeschema, click File —> Schematic Setup —> Violation Severity (in the Electrical Rule group). The dialog box looks like this (Figure 13.11.1):

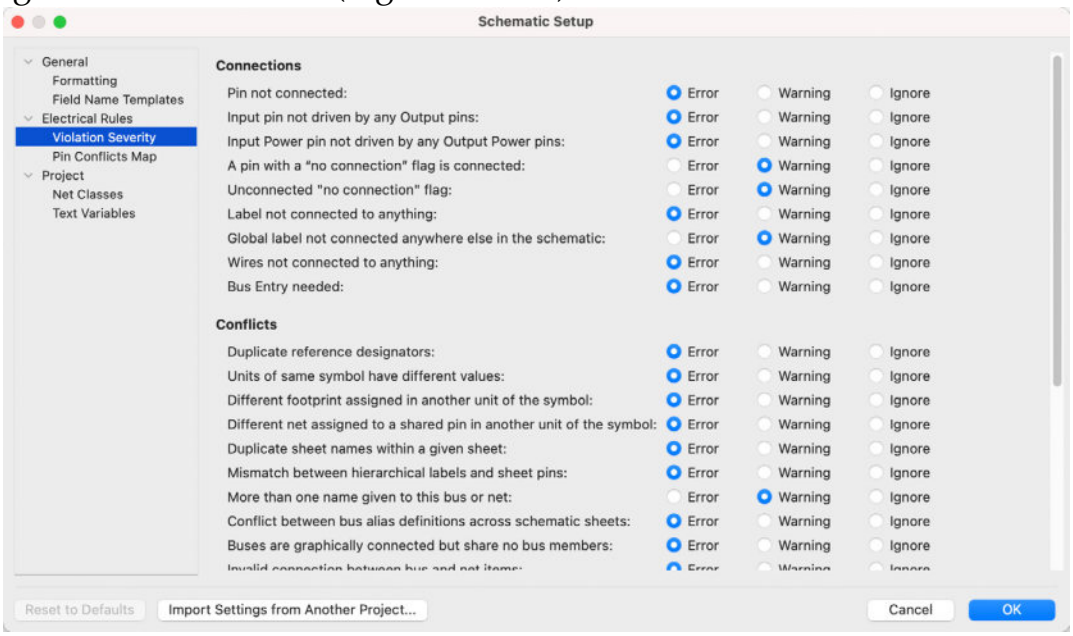


Figure 13.11.1: The Violation Severity dialog box in Pcbnew.

The default settings are reasonable, and you will rarely need to make any changes.

Let's look at an example to help you understand how you can change the classification of a violation during an ERC. In my test schematic below, I have introduced a defect by deleting a wire segment (Figure 13.11.2).

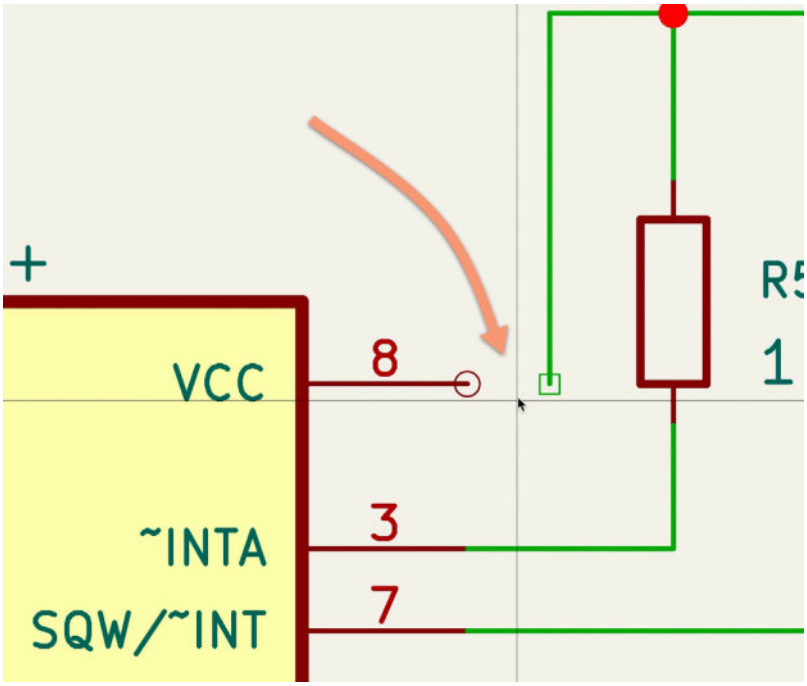


Figure 13.11.2: An evident defect: a wire segment is missing.

Under " Connections, " the violation at the very top of the list, under “Connections,” is “Pin not connected.” The default classification for this violation is “Error.” Go ahead and change this to “Warning” (Figure 13.11.3):

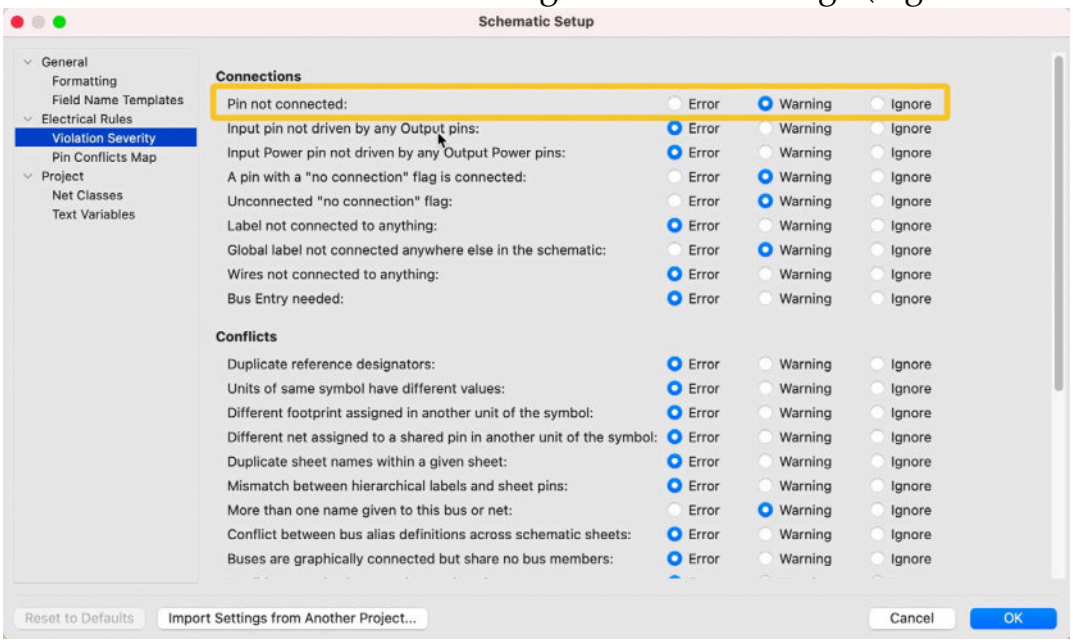


Figure 13.11.3: Changed classification of “Pin not connected” to “Warning.”

Let's see the impact of this change on the way that the ERC classifies schematic violations. Bring up the ERC window (Inspect —> Electrical Rules Checker) and click on "Run ERC." You can see the result below (Figure 13.11.4):

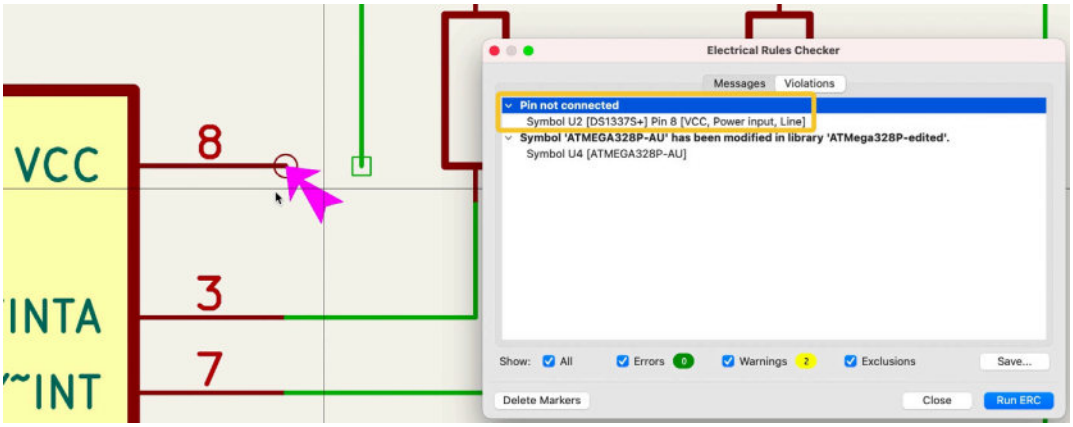


Figure 13.11.4: The ERC lists the missing wire segment as “Pin not connected” and a warning.

The ERC lists the missing wire segment as "Pin not connected," classified as a "Warning." Notice that in this ERC, the total number of errors is zero.

Change the "Pin not connected" classification back to its original "Error," and repeat the ERC. The result is below (Figure 13.11.5).

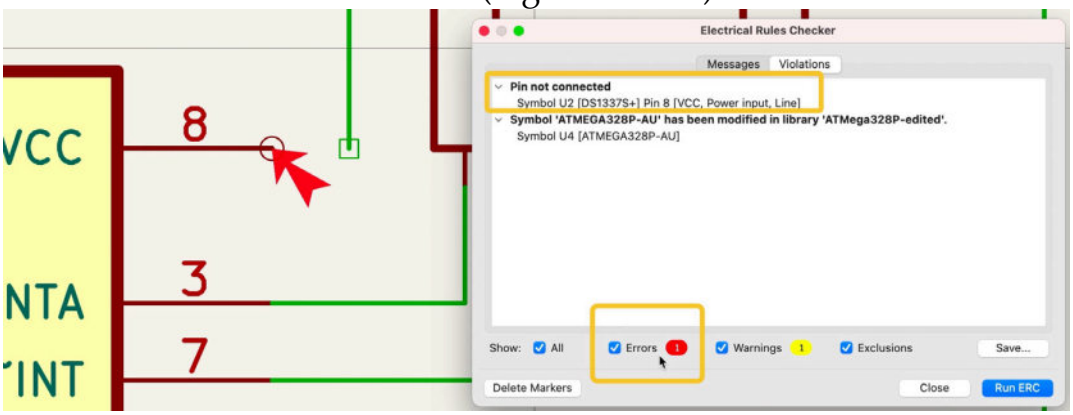


Figure 13.11.5: The ERC lists the missing wire segment as “Pin not connected” and an error.

The same violation that was previously listed as a Warning is now listed as an Error.

Notice that unlike Pcbnew and the DRC, the ERC does not prepend a violation with its classification. For example, the DRC reports a "Warning" like this:

```
Warning: Missing footprint
```

However, an ERC violation looks like this:

```
Pin not connected
```

In the ERC, there is no explicit mention of the kind of violation that it is. You can derive this information by looking at the accumulated numbers of Errors, Warnings, and Exclusions at the bottom of the ERC window.
You can further customize the ERC by looking at the Pin Conflicts Map dialog box. This allows you to set violation classifications for the various pin to pin conditions. You can learn about this in a dedicated chapter.

12. Schematic Setup - Electrical Rules and Pin conflicts map

The Pin Conflicts Map dialog box (found in the Schematic Setup window, under Electrical Rules) allows you to set violation classifications for the various pin to pin conditions.

To access this dialog go to File, Schematic Setup, and click on "Pin Conflicts Map" under "Electrical Rules." The user interface of this tool resembles an assignment triangle. The intersecting points between columns and rows are buttons. Each time you click on a button, you can change its value into one of three possible values:

-  No error or warning.
-  Generate warning.
-  Generate error.

Let's look at an example.

Bring up the Pin Conflicts Map dialog box. It looks like this (Figure 13.12.1):

Figure 13.12.1: The Pin Conflicts Map

In this example, I have highlighted the conflict classification for a connection between two pins designated as outputs. The classification of this type of connection is "Generate error."

Similarly, a connection between an open connector pin and an output pin will generate an error.

Also, notice that the last row of the triangle is set to generate errors for all not connected pins. In my test schematic, I have left pin 23 unconnected, and the ERC lists this as an error

(Figure 13.12.2):

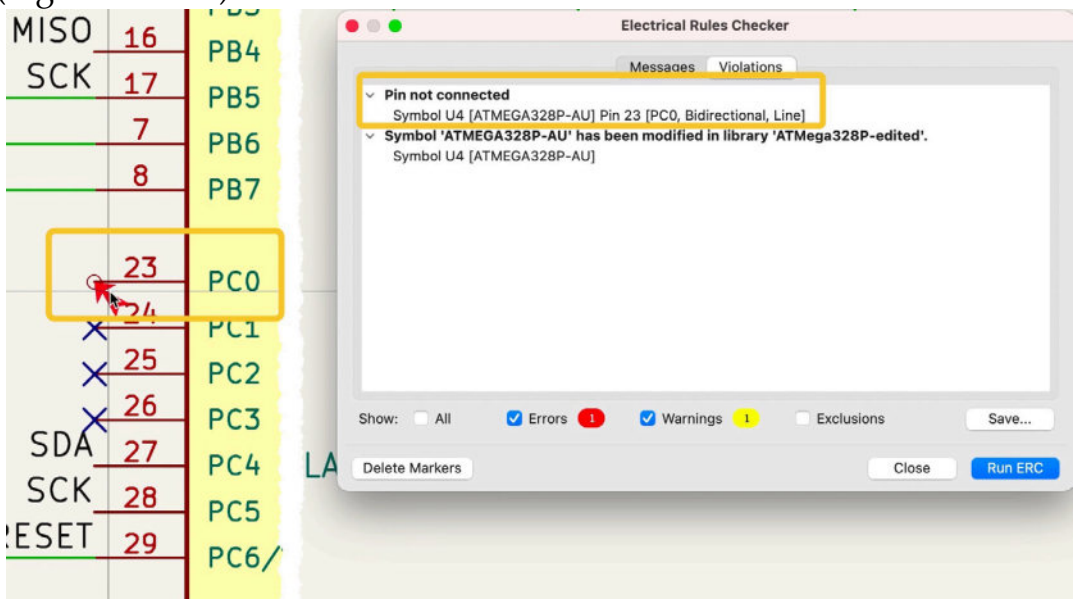


Figure 13.12.2: ERC reports an unconnected pin error.

Let's do a quick experiment that uses this test schematic (Figure 13.12.2):

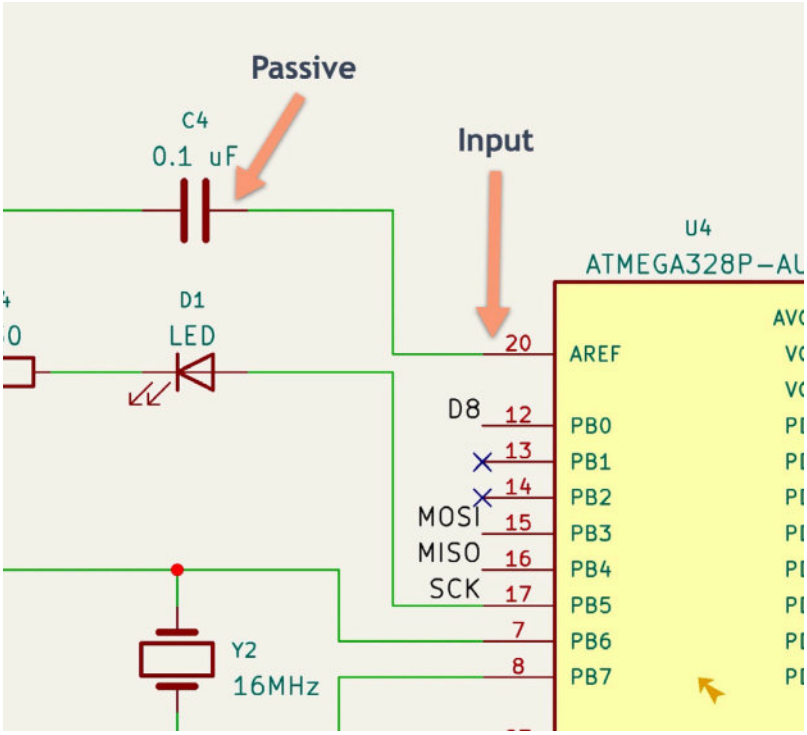


Figure 13.12.2: A connection between a passive and an input pin.

In this example, there is a connection between a capacitor pin (set as "passive" in the symbol properties) and pin 20 of the Atmega328P symbol, which is designated as "input." You can check the pin type by double-clicking on the symbol to bring up its properties window and click on Edit Symbol (Figure 13.12.3).

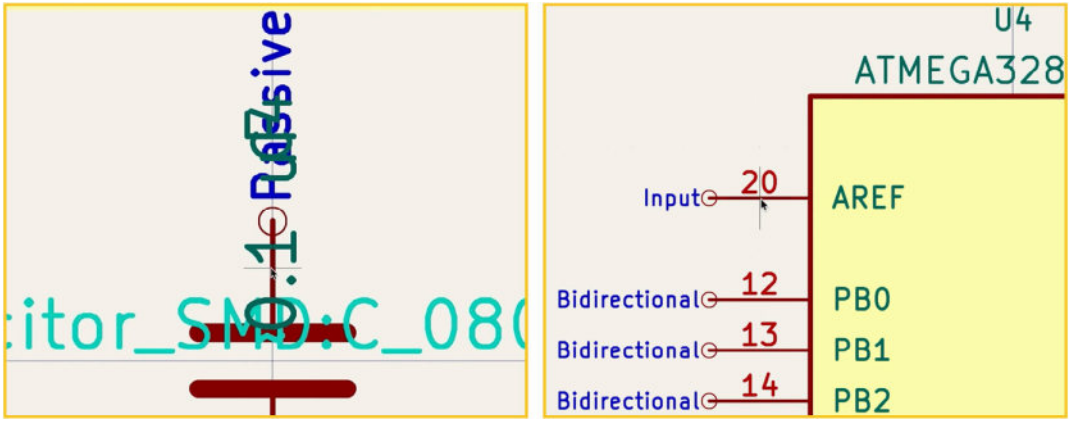


Figure 13.12.3: The type of pins that are connected in my test schematic.

As you can see in figure 13.12.1, the classification for connection between passive and input pins is "green"; therefore, no error or warning is generated. Truthfully, when I ran the ERC earlier (see Figure 13.12.2), the ERC did not list any relevant violations.

Let's change this classification. Open the Pin Conflicts Map, and click on the button in the junction of the Passive Pin row and Input Pin column to show the red "!" mark

(see Figure 13.12.4).

Figure 13.12.4: Passive to input pin connections are now classified as “error”.

Click OK, and rerun the ERC. Here is the result (see Figure 13.12.5):

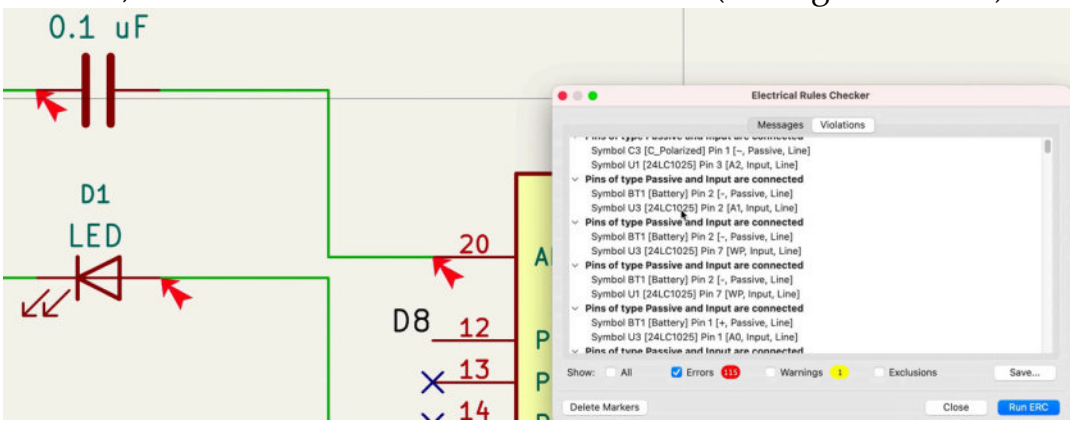


Figure 13.12.5: Passive to input pin connection errors are detected by the ERC.

The ERC has detected a large number of passive to input pin errors. One of them is the one between the capacitor in pin 20 of the Atmega328P. I don't want this kind of connection to be classified as an error, so I have restored the violation classification as "green."

The Pin Conflicts Map gives you another way to customize the operation of the ERC. When used with care and consideration, it can be another way to ensure that KiCad fits well with the specific requirements of your project.

13. Field name templates

"Field name templates" is a new feature in KiCad 6. With this feature, you can add custom fields to the list of symbol field properties and complement the pre-defined ones.

Here is an example of the properties for one of the symbols in my test schematic, the DS1337S+ real-time clock (Figure 13.13.1):

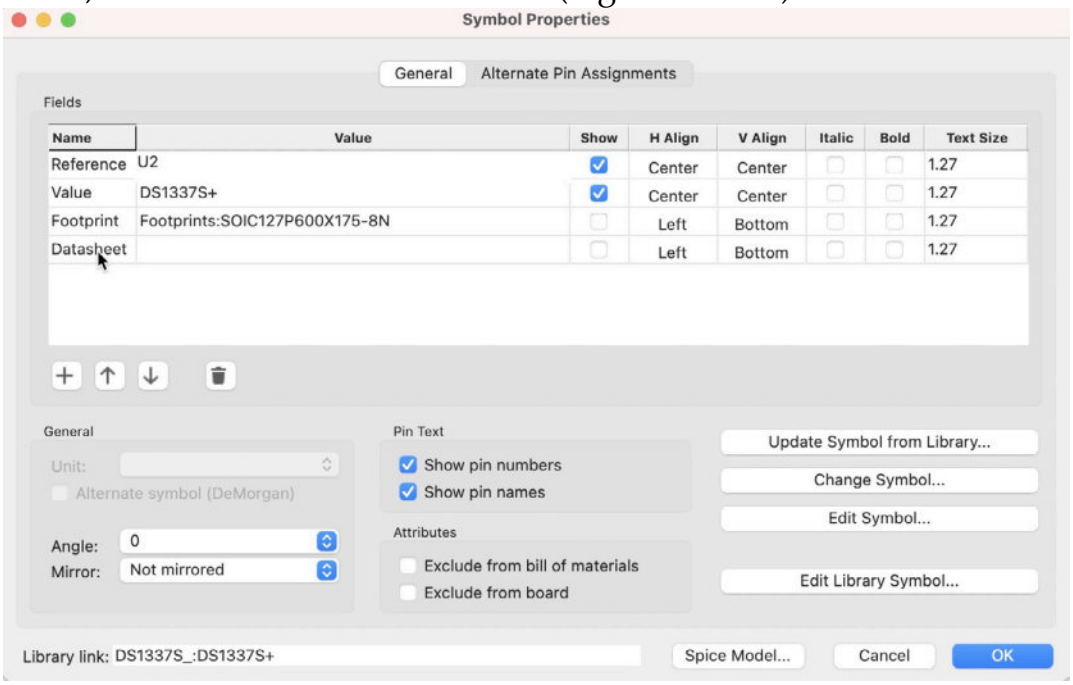


Figure 13.13.1: The Symbol Properties window showing the pre-defined fields.

If you wish to add other fields to this list, you can use the Field Name Templates feature. For example, you may want to add:

- a "Description" field, where you can type a short description for the function of the component,

Your custom field name templates will be shared among all symbols in the project.

- a "source" field so that you add the URL for an online retailer from where you can purchase the component,
- or a "Wikipedia" field where you can copy the URL of a Wikipedia article relevant to the component.

Your custom field name templates are shared among all symbols in the project.

There are two locations where you can define custom field name templates.

- Local project level, via File —> Schematic Setup —> Field Name Templates
- Global project level, via KiCad —> Preferences —> Field Name Templates

Both work in the same way. The only difference is that any field names created in the Schematic Setup window will be available only in the current project. In contrast, those made in the Preferences window will be available to all projects.

Let's look at an example using the local project level option.

Bring up the Schematic Setup window (File —> Schematic Setup) and click on Field Name Templates. Click on the "+" button twice at the bottom of the window to create two new rows. Then type the names for the new fields (Figure 13.13.2):

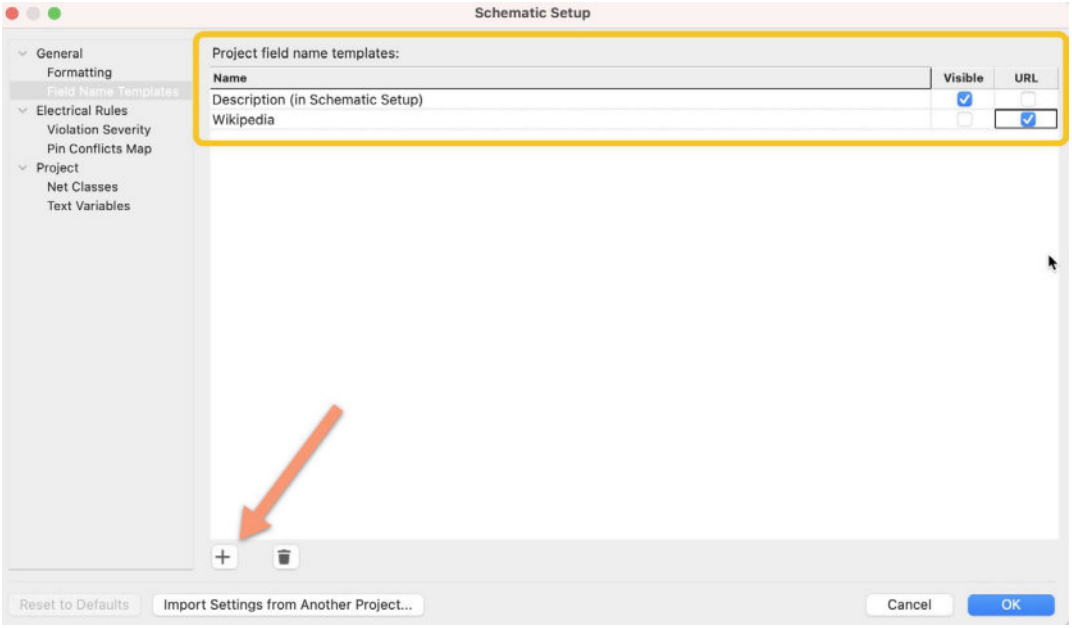


Figure 13.13.2: Two new field name templates.

The two new fields names are "Description (in Schematic Setup)" and "Wikipedia."

Two checkboxes control the new field's value visibility in the schematic editor ("Visible") and set the value as a URL so that when you click on it, the URL is copied to a browser ("URL"). I have enabled visibility for the first field and the URL checkbox for the second.

Click OK to commit the new fields, and return to the editor. In the editor, double-click on any of your symbols to bring up its properties. Below, I show the properties of the DS1337S+ IC. I have added some text in the new fields (Figure 13.13.3):

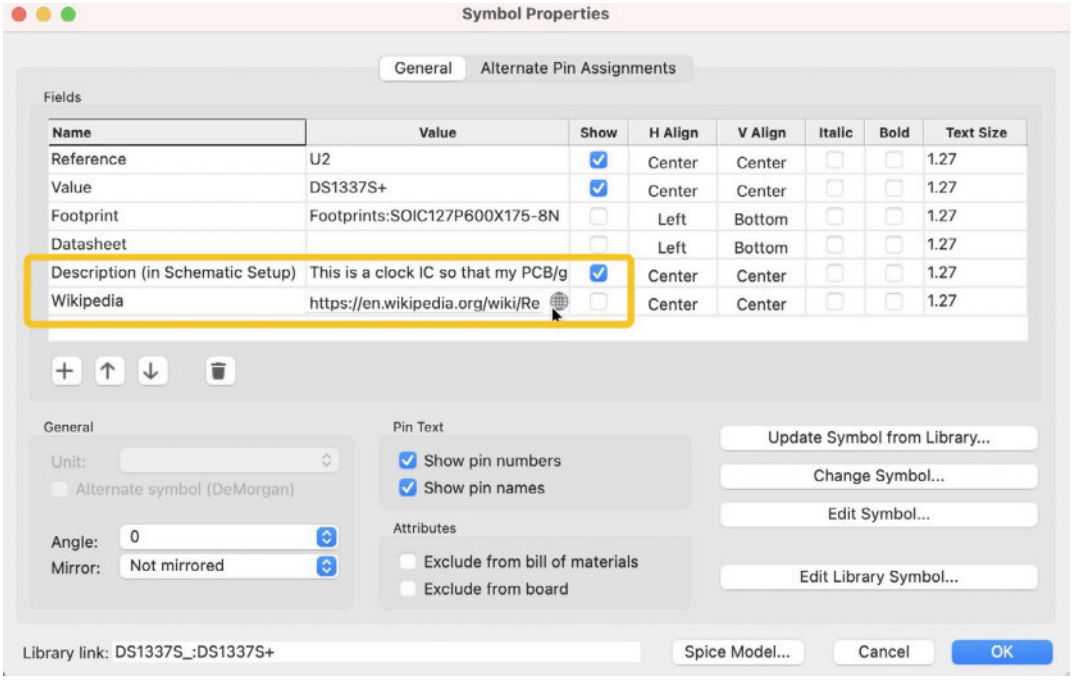


Figure 13.13.3: Two new fields appear in the Symbol Properties field.

The value in the Wikipedia field includes a globe icon to indicate that it is clickable. When you click on this globe, the URL opens up a web browser to the page specified. The "Show" checkbox for the Description is checked to display the text value in the schematic. To show you how the URL looks in the schematic, I have also enabled the Wikipedia "Show" checkbox (though I did not capture this in the screenshot of Figure 13.13.3). Click OK to dismiss the Symbol Properties window and return to the editor. Below you can see how the two new field values look like (Figure 13.13.4):

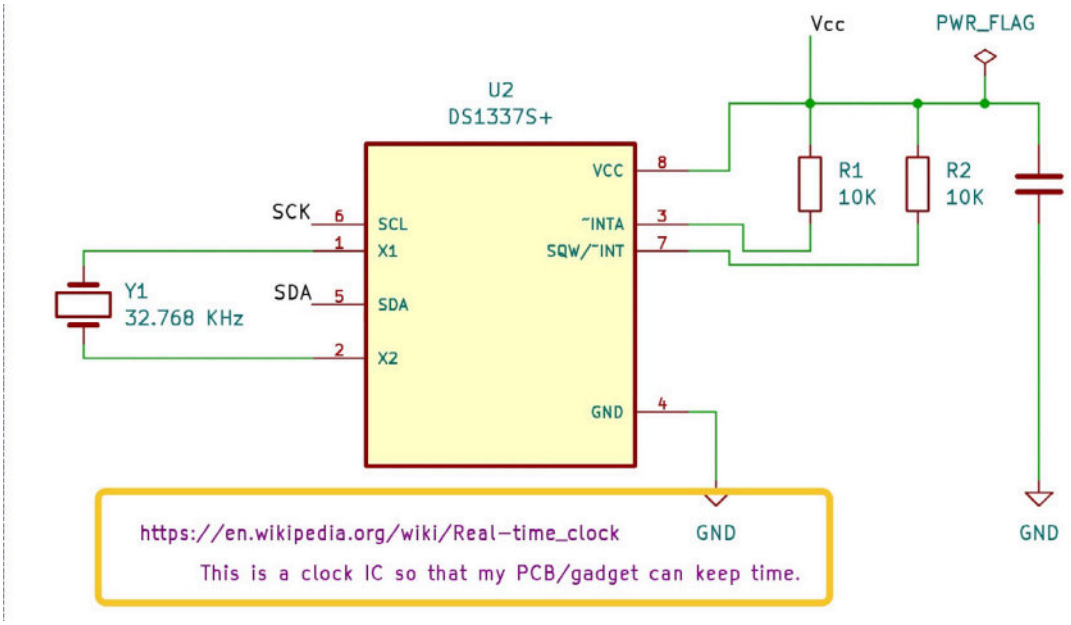


Figure 13.13.4: The values of the new fields in the schematic editor.

Let's try out the second option at the global project level. Click on KiCad —> Preferences —> Field Name Templates. Click on the "+" button once to create a new field, and give it the name "Description (in KiCad Preferences)." I have added the qualification to differentiate between this field and the one I created earlier at the project level. The window looks like this (Figure 13.13.5):

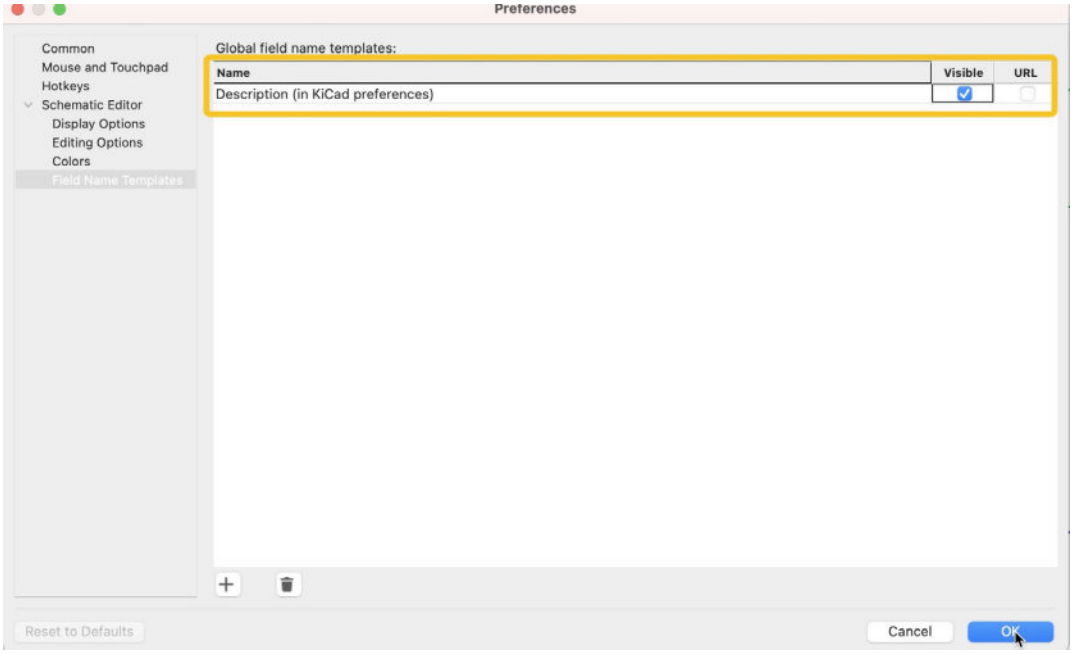


Figure 13.13.5: A new field template at the global level.

Click OK to return to the editor, then double-click on a symbol to bring up its properties. You will notice that the new field appears below the one you created earlier at the project level. Enter some text in the new field. Here is

mine, in the Symbol Properties field for the DS1337S+ IC component (Figure 13.13.6):

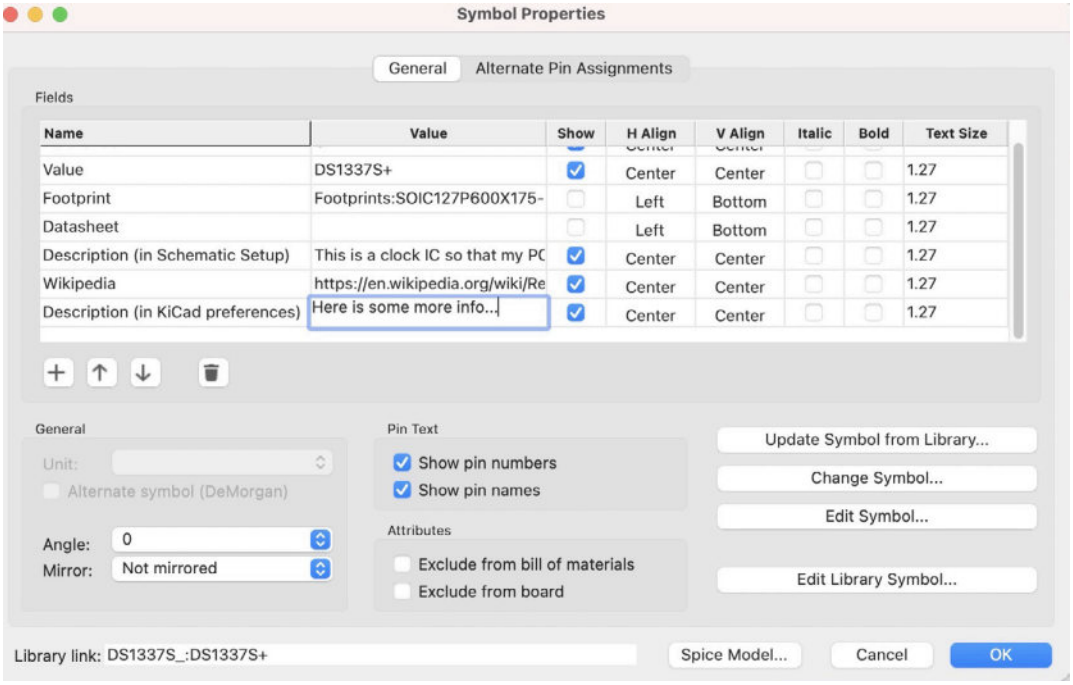


Figure 13.13.6: The new global field name template.

Click OK to return to the editor, and see the global "description" field value in the new field next to its symbol (Figure 13.13.7):

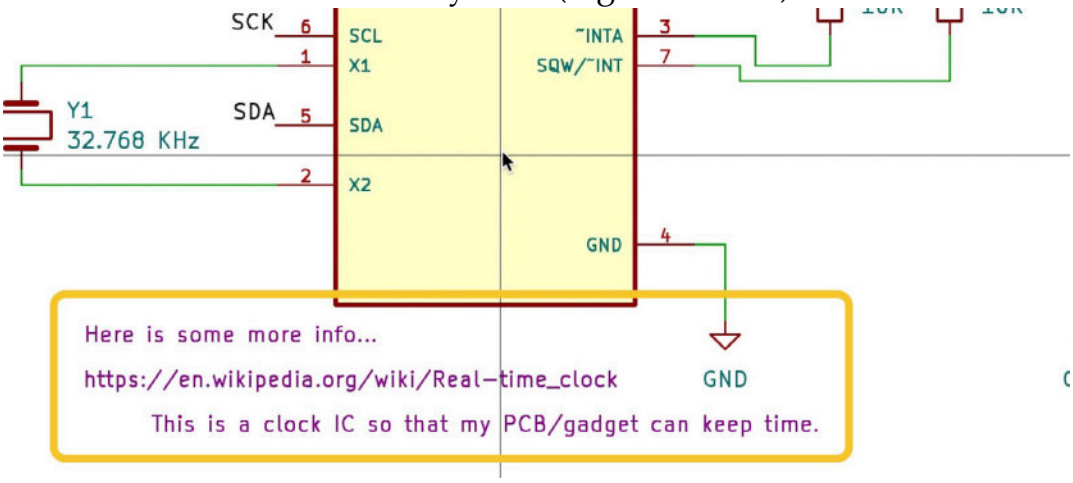


Figure 13.13.7: The new global field value.

You can add any number of project or global field name templates to incorporate helpful information. A handy application for those custom fields is that you can included them in the bill of materials (BOM). To learn more about BOM and how you can generate one in KiCad, please read the dedicated chapter.

14. Bill of Materials

Once you have completed the design of your printed circuit board and you are ready to manufacture it, you may want to consider exporting a bill of materials. A bill of materials ("BOM") is a list of the raw components, along with meta-data like their quantity, values, designators, and sources, that an end-user of your PCB will need to complete the assembly process and deliver a finished product.

KiCad can generate a BOM "out of the box" without a need for external plugins. It is also possible to use third-party plugins to generate a BOM. In this chapter, I will show you both options.

In the first instance, there are two build-in methods for generating a BOM. I will also show you the same using a plugin.

14.1. Build-in BOM in Pcbnew

Pcbnew offers the most straightforward method for generating a BOM. It is available through the File menu. Click on File —> Fabrication Outputs —> BOM (Figure 13.14.1.1).

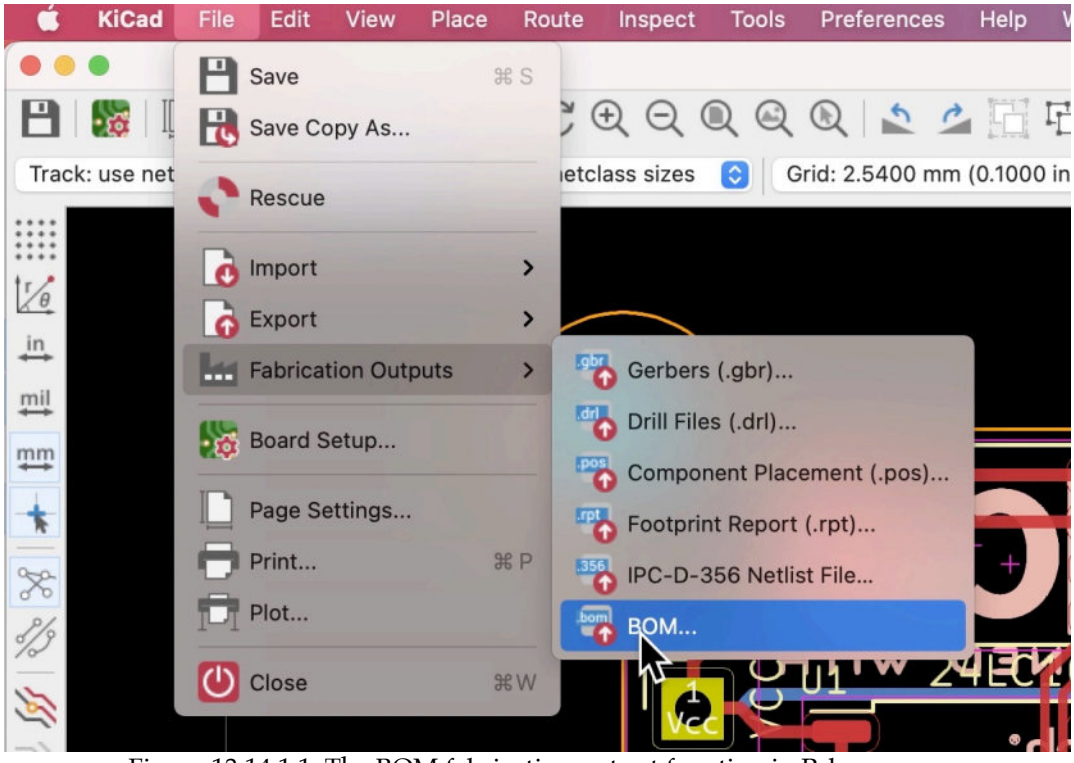


Figure 13.14.1.1: The BOM fabrication output function in Pcbnew.

This function will ask you for a location and name for the new CSV text BOM file and save this file in your computer's file system. In my example below, I have this file the name "MCUCataloggerBOM.csv" (Figure 13.14.1.2).

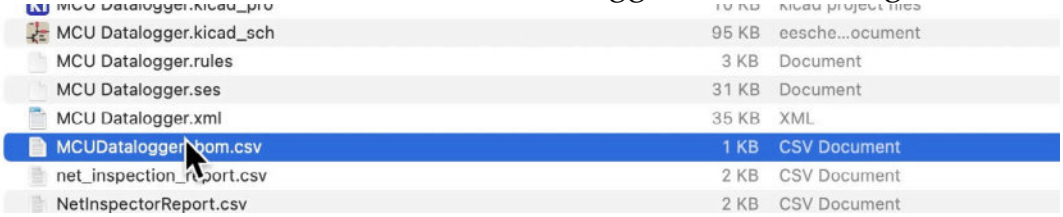


Figure 13.14.1.2: The new CSV BOM file.

Open it with a text editor to see its data. You can see it below in the Atom editor (Figure 13.14.1.3):

Figure 13.14.1.3: The contents of the CSV BOM file from Pcbnew.

This BOM file contains the fundamental bits of information about the components of the PCB circuit:

- ID.
- Designator.
- Package.
- Quantity.
- Designation.
- Supplier and ref.

The BOM does not contain any additional build-in or custom component properties. I remind you that a symbol in Eeschema can hold any number of fields, as you can see in this example (Figure 13.14.1.4):

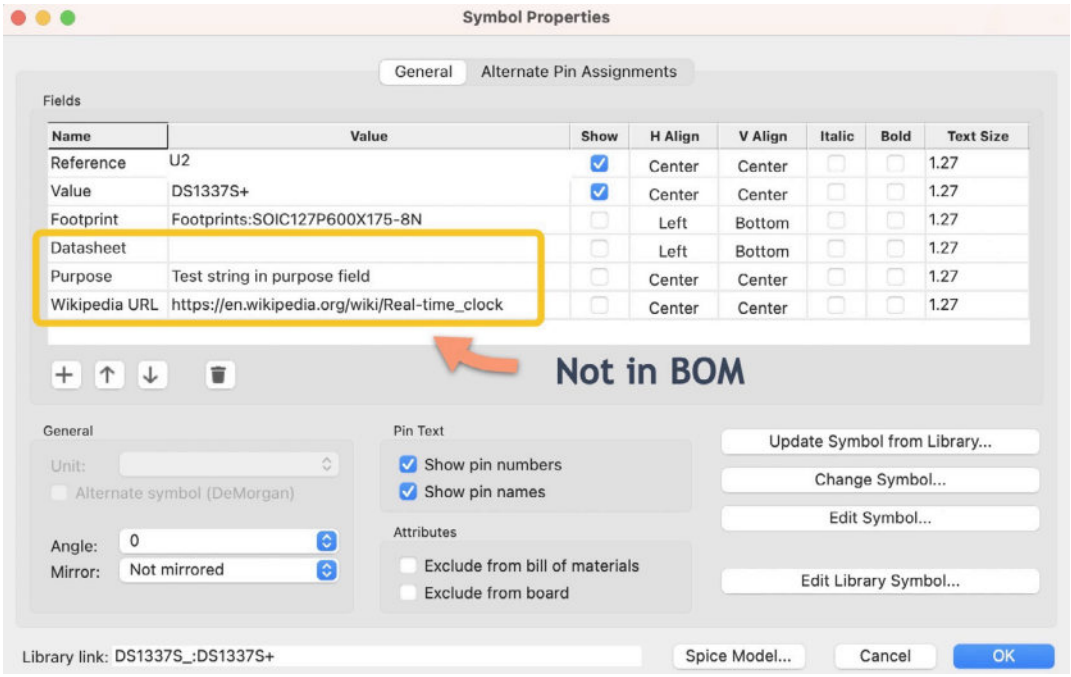
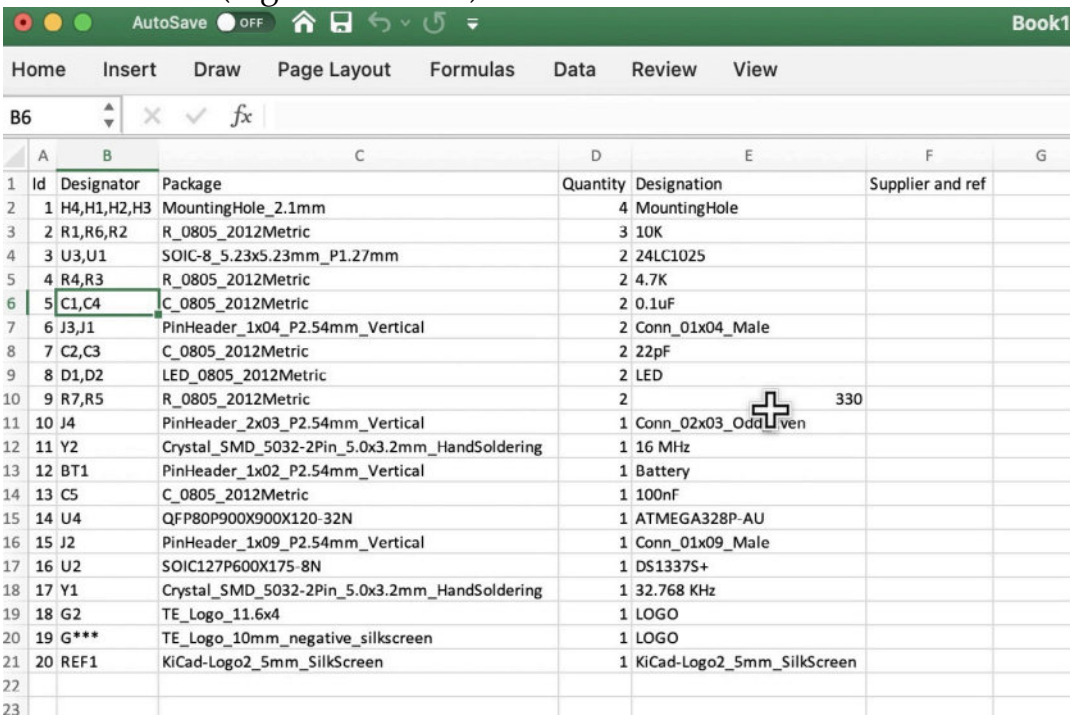


Figure 13.14.1.4: These three fields do not appear in the BOM CSV file from Pcbnew.

In the example above, the three fields at the bottom of the Fields list do not appear in the BOM file I just exported. One of the missing fields ("Datasheet") is built-in, and the other two are custom. I will show you how to include any number of fields in the BOM file using the new two BOM methods.

The exported BOM file is a simple text CSV file. You can open it using a spreadsheet application like Microsoft Excel or Google Sheets. This how it looks like in Excel (Figure 13.14.1.5):



	A	B	C	D	E	F	G
1	Id	Designator	Package	Quantity	Designation	Supplier and ref	
2	1	H4,H1,H2,H3	MountingHole_2.1mm	4	MountingHole		
3	2	R1,R6,R2	R_0805_2012Metric	3	10K		
4	3	U3,U1	SOIC-8_5.23x5.23mm_P1.27mm	2	24LC1025		
5	4	R4,R3	R_0805_2012Metric	2	4.7K		
6	5	C1,C4	C_0805_2012Metric	2	0.1uF		
7	6	J3,J1	PinHeader_1x04_P2.54mm_Vertical	2	Conn_01x04_Male		
8	7	C2,C3	C_0805_2012Metric	2	22pF		
9	8	D1,D2	LED_0805_2012Metric	2	LED		
10	9	R7,R5	R_0805_2012Metric	2	330		
11	10	J4	PinHeader_2x03_P2.54mm_Vertical	1	Conn_02x03_Odd_even		
12	11	Y2	Crystal_SMD_5032-2Pin_5.0x3.2mm_HandSoldering	1	16 MHz		
13	12	BT1	PinHeader_1x02_P2.54mm_Vertical	1	Battery		
14	13	C5	C_0805_2012Metric	1	100nF		
15	14	U4	QFP80P900X900X120-32N	1	ATMEGA328P-AU		
16	15	J2	PinHeader_1x09_P2.54mm_Vertical	1	Conn_01x09_Male		
17	16	U2	SOIC127P600X175-8N	1	DS1337S+		
18	17	Y1	Crystal_SMD_5032-2Pin_5.0x3.2mm_HandSoldering	1	32.768 KHz		
19	18	G2	TE_Logo_11.6x4	1	LOGO		
20	19	G***	TE_Logo_10mm_negative_silkscreen	1	LOGO		
21	20	REF1	KiCad-Logo2_5mm_SilkScreen	1	KiCad-Logo2_5mm_SilkScreen		
22							
23							

Figure 13.14.1.5: The BOM file in Excel.

You can use this spreadsheet instance of the BOM to add or edit the information before you share it with other people or upload it to a manufacturing facility.

This method is quick and straightforward to use, but as I mentioned above, it lacks flexibility. Next, I'll show you the second build-in BOM generator that KiCad offers that provides more flexibility while being somewhat more complicated to use.

14.2. Build-in BOM in Eeschema

The second method for generating a BOM that KiCad offer is available in Eeschema. The functionality we need is "hidden" inside the Edit Symbol Fields window. In Eeschema, click on Tools —> Edit Symbol Fields (Figure 13.14.2.6). You can ignore the "Generate BOM" option in the same menu. Even though it looks like the one we want, it requires a plugin to work. I will show you an equivalent (but better) option later.

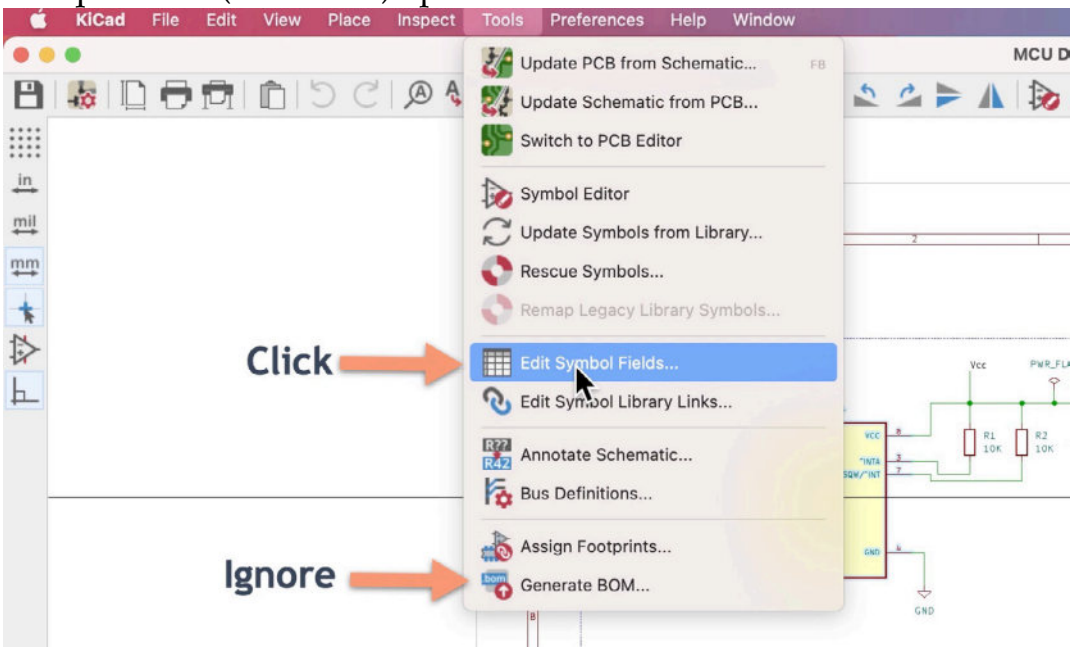


Figure 13.14.2.6: In Eeschema, the BOM generator hides in the Edit Symbol Fields window.

This will bring up the Edit Symbol Fields window. This window offers a convenient way to edit the information of any of the symbol property fields, for any symbol, in bulk. You can edit the values of any field or even create custom fields. This editor can save you a lot of time from going into each symbol properties window and then editing the field values one at a time.

In this scenario, I want to generate a BOM that contains a combination of built-in and custom fields. In the left pane is a list of the available fields with radio buttons (1). The "Show" radio button controls if a given field will appear

in the table in the main pane (2). In this example, I have enabled all "show" fields except for "MANUFACTURER." As I enable and disable the Show checkboxes, the main pane updates its contents.

In the main pane, you can also re-arrange the columns in any order you like. Simply click on the column's header and hold the button down while dragging the entire column to a new position.

When you are ready to export the data, click on the top-most field, hold down the shift key to enable multiple-select, and click again on the bottom right corner field. This works in the same way as when you do a multiple-select operation in a spreadsheet. When all fields are highlighted with a blue background, type Ctr-C or Cmd-C (Windows, Mac), or do a right-click to reveal the context menu and select Copy. You can see all this in the example screenshot below (Figure 13.14.2.7):

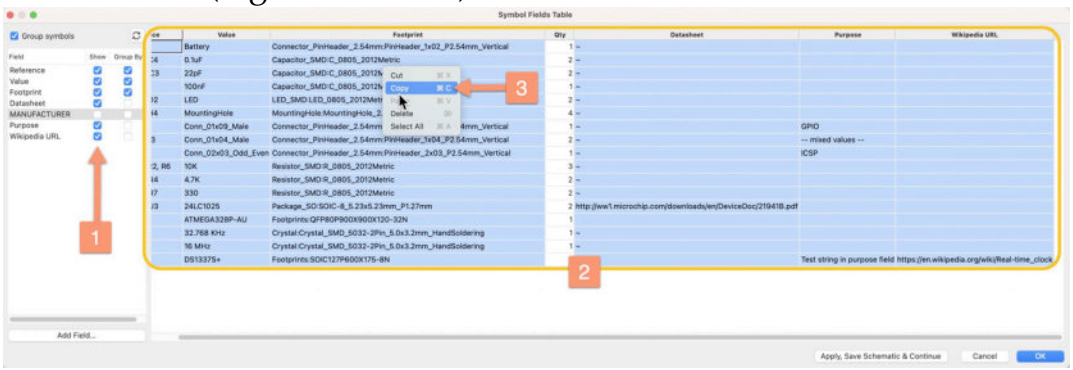


Figure 13.14.2.7: In Eeschema you can export BOM data from the Edit Symbol Fields window via a copy operation.

At this point, you have copied the BOM data in a comma-delimited format in your computer's clipboard. Now you want to paste the data into a spreadsheet or text editor. Below, I have pasted (Ctrl-V or Cmd-V) the copied data to Excel (Figure 13.14.2.8):

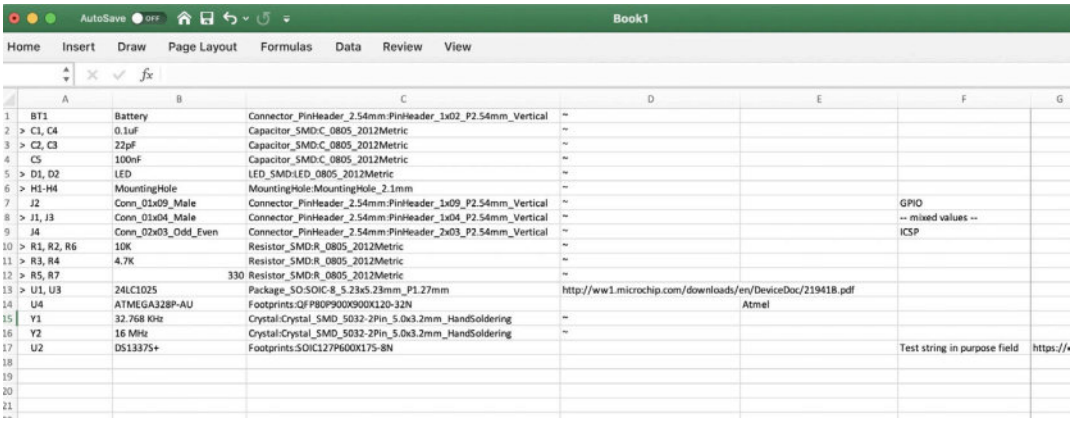


Figure 13.14.2.8: Edit Symbol Field data copied into Excel.

Notice from the screenshot above that the header of the BOM is missing. The header is not included in the data copied from the Edit Symbol

Field window. You will need to add the header manually in Excel. Just insert a new row at the top of the table, and type in the appropriate column title.
The data are still in the clipboard, so you can try paying in a text editor. Below is a screenshot from my paste operation in the Atom text editor (Figure 13.14.2.9):

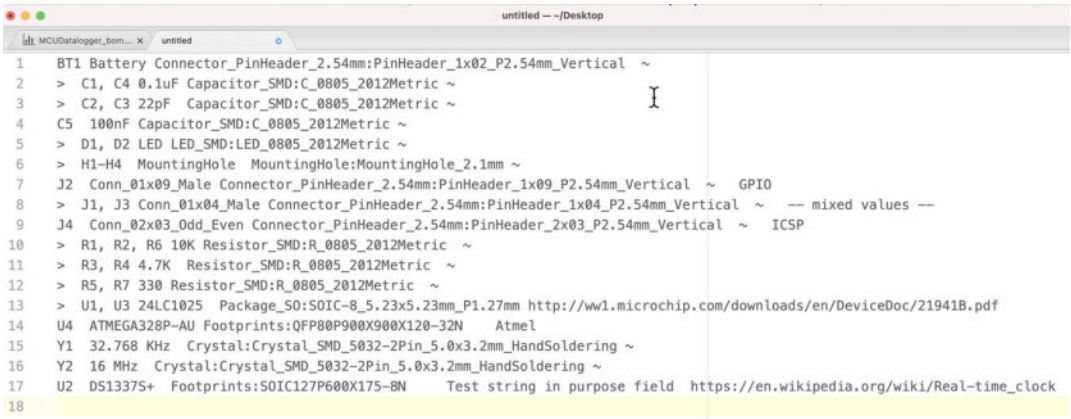


Figure 13.14.2.9: Edit Symbol Field data copied into a text editor.

From there, you can use this spreadsheet or text editor instance of the BOM to add or edit the information before you share it with other people or upload it to a manufacturing facility.

There's one more BOM generator method that I will show that involves using a third-party plugin. The plugin itself provides a powerful way of BOM generation and an opportunity to learn how to install and use a plugin in KiCad 6.

14.3. A plug-in for BOM

KiCad build-in functionality can be extended with the use of external plugins. I will show you how to use one such plugin to generate a BOM for your project in this segment. The plugin I will use is "Interactive HTML BOM," and you can find it in Github at https://github.com/openscopeproject/InteractiveHtmlBom (Figure 13.14.3.10).

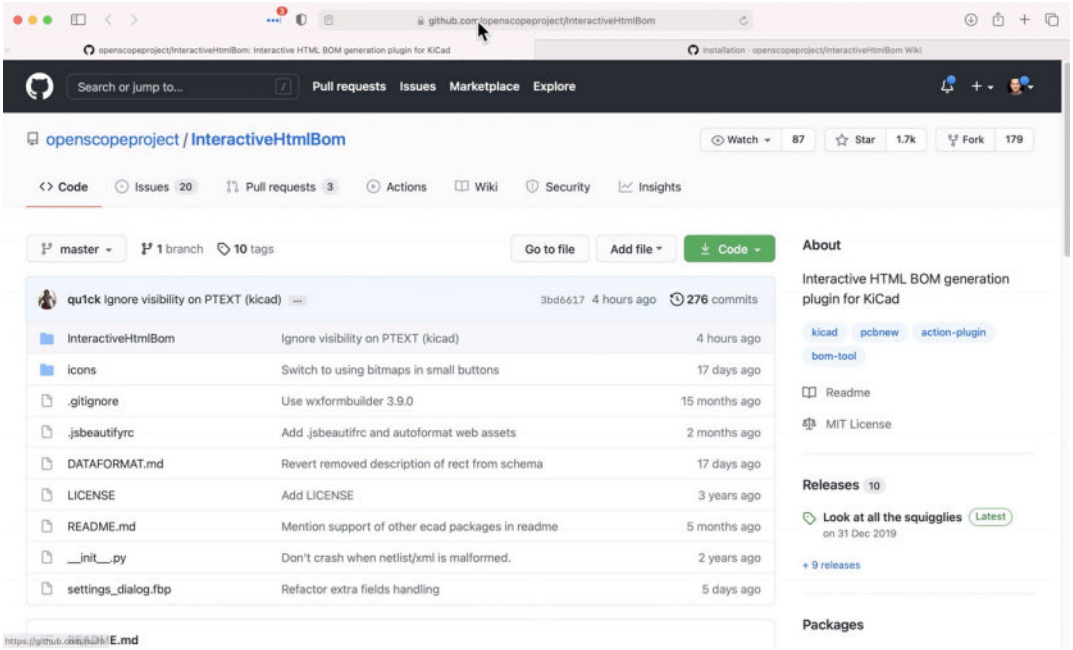


Figure 13.14.3.10: The “Interactive HTML BOM” plugin project on Github.

This plugin contains excellent documentation and also supports the EasyEDA and Fusion360 CAD applications. I encourage you to take a few minutes to review the information in the project wiki (https://github.com/openscopeproject/InteractiveHtmlBom/wiki). This wiki is the source of the information I used to learn how to install the plugin to my KiCad instance and use it.

KiCad can look for plugins in several folders, and the exact locations can vary between operating systems. They also depend on how you installed KiCad on your computer. The most reliable way to find out the correct plugin folder(s) for your KiCad instance is to use the Python scripting console and run a couple of lines of code.

First, bring up the Python scripting console. To do that, in Pcbnew, click on Tools —> Scripting Console (Figure 13.14.3.11).

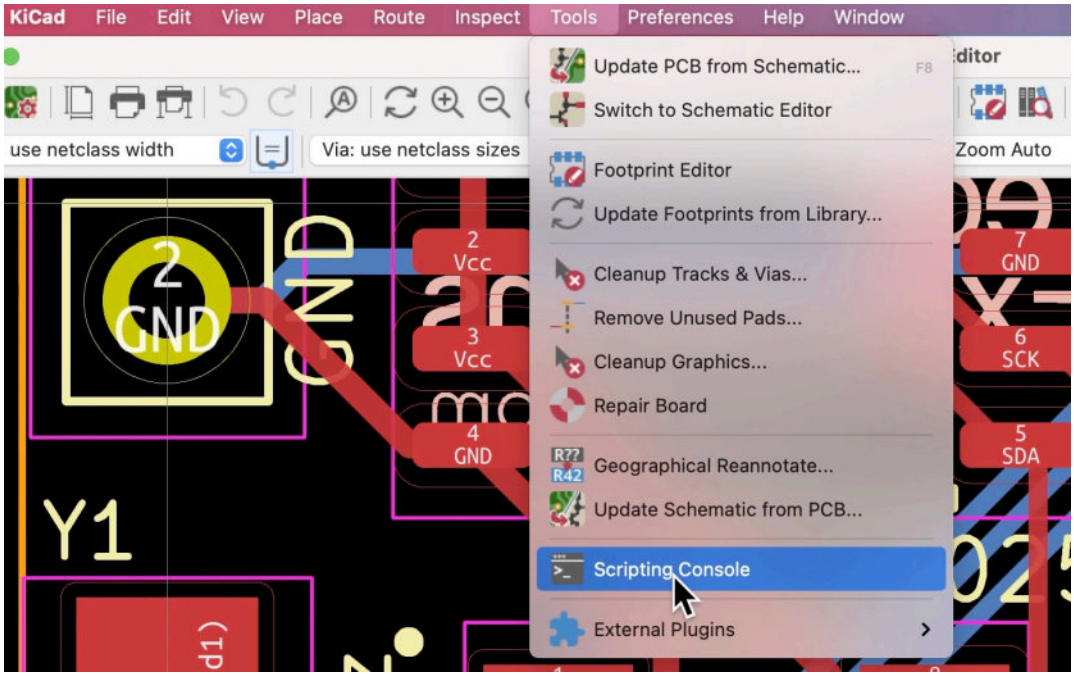


Figure 13.14.3.11: The Python Scripting Console option in the Tools menu in Pcbnew.

This will bring up the console. The console contains the main pane on the top side ("shell"), where you can enter Python code and see the interpreter responses. You can ignore the tabs in the bottom pane as you don't need them for this operation. In the shell, type these two command lines:

```
   >>> import pcbnew
   >>> print(pcbnew.PLUGIN_DIRECTORIES_SEARCH)
   When you commit the second line, the following text will appear
in the shell:
   /Applications/KiCad/KiCad.app/Contents/SharedSupport/scripting
   /Applications/KiCad/KiCad.app/Contents/SharedSupport/scripting/
plugins
   /Users/peter/Library/Preferences/kicad/5.99/scripting
   /Users/peter/Library/Preferences/kicad/5.99/scripting/plugins
   /Users/peter/Documents/KiCad/5.99/scripting
   /Users/peter/Documents/KiCad/5.99/scripting/plugins
   >>>
```

The output you see in your shell will be different from mine. In the example above, there are more than one folders that I can use to store plugins. I prefer to use the last one because it is located in my Documents folder, which is one of the locations of my filesystem that are automatically backed up. I also have shortcuts so that I can navigate there quickly. Below is what the Python scripting console looks for me at this point: (1) the two lines of Python code I have typed in, and (2) the response from the interpreter (Figure 13.14.3.12):

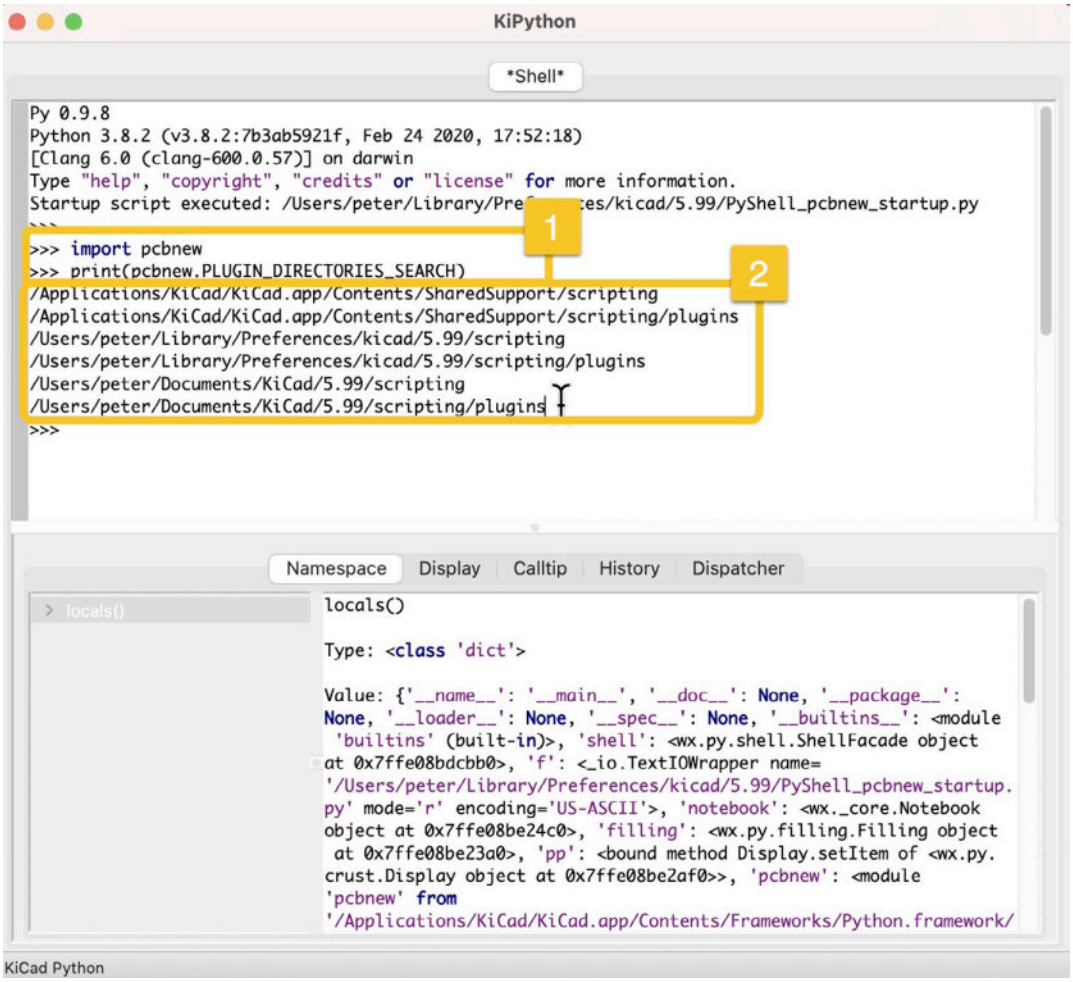


Figure 13.14.3.12: The Python scripting console helps me find the folders where I can store KiCad plugins.

If you are wondering why the path of my KiCad instance plugin folders contains the version "5.99", it is because at the time I am writing this book, I am using the KiCad version 5.99 nightly builds.

There is a shortcut for the KiCad plugins directory. Click on Tools —> External Plugins —> Reveal Plugin Folder, and KiCad will take you straight there.

It is OK to close the Python scripting console. Now that we know where to store the plugin let's focus on the plugin itself. In the plugin's Github repository, click on the "Download ZIP" link (under the green "Code" button) to download it (Figure 13.14.3.13).

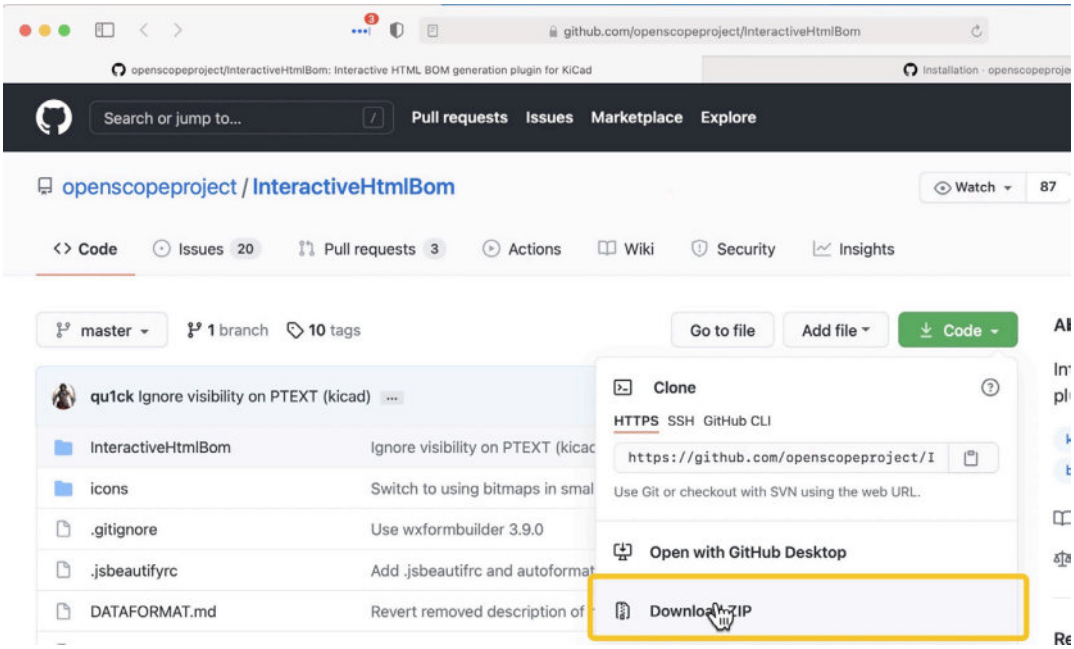


Figure 13.14.3.13: Download the ZIP archive of the plugin.

Download the archive and expand it. Then navigate to the plugin folder that you chose, and copy the plugin folder in it. For my setup, it looks like this (Figure 13.14.3.14):

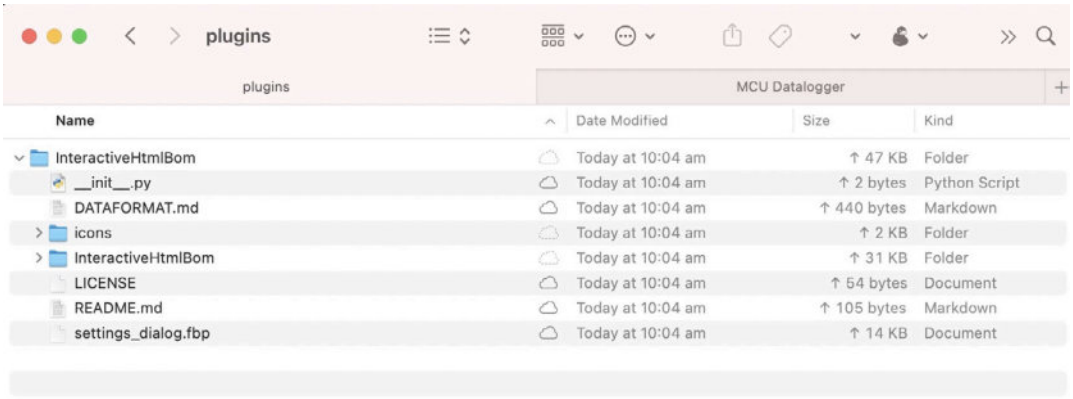


Figure 13.14.3.14: The “Interactive HTML BOM”plugin folder copied in the KiCad plugins folder.

Before you can use the new plugin, you must refresh the KiCad plugin's register. Go to Tools —> External Plugins, and click on Refresh Plugins (Figure 13.14.3.15):

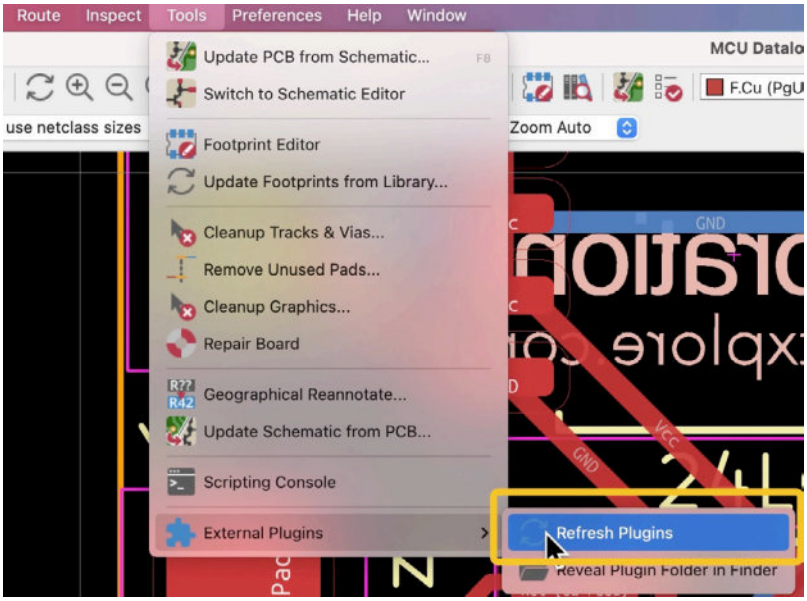


Figure 13.14.3.15: Click on “Refresh Plugins” to activate new plugins.

After about a second, a new icon will appear in the top toolbar (Figure 13.14.3.16):

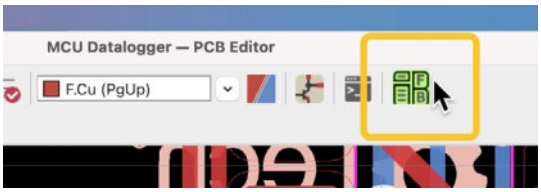


Figure 13.14.3.16: The new button belongs to the Interactive HTML BOM plugin.

Let's try this out. Click on the green HTML BOM button on the top toolbar to get to the plugin's BOM generator window. Below I show how I have set up my BOM generator's three tabs (Figure 13.14.3.17).

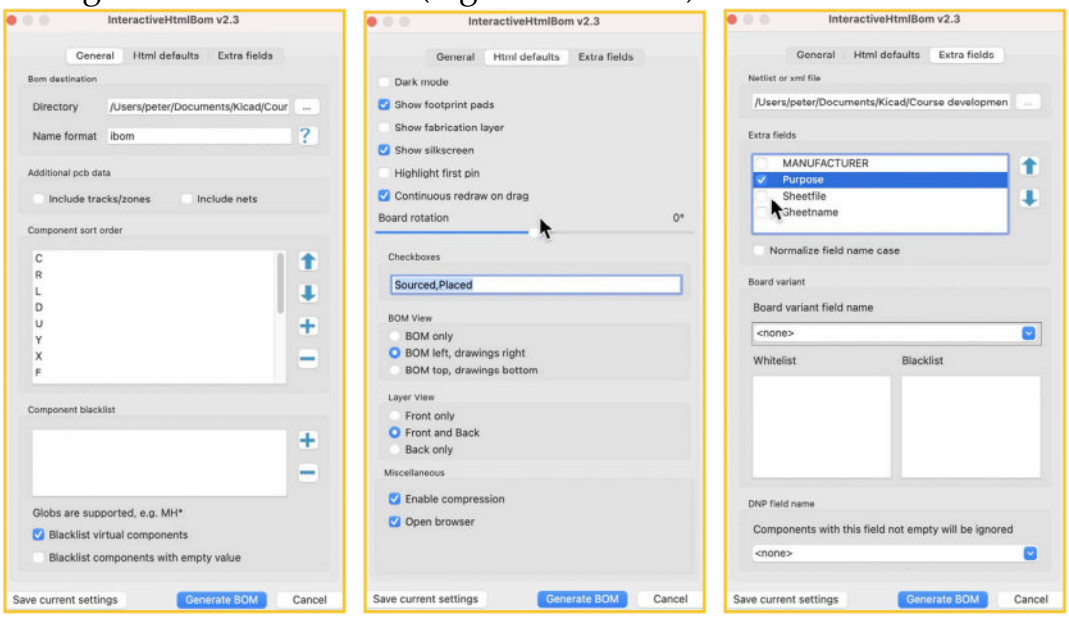


Figure 13.14.3.17: The three tabs of the HTML BOM plugin.

In the General tab, you can set the destination of the BOM files that this plugin will generate. You can also set the ordering of the components based on their type (such as capacitors, resistors, etc.)

In the Html defaults tab, you can control various visual elements that affect how the BOM HTML page will look in your browser.

In the Extra fields tab, you can enable or disable the extra (custom) fields that you'd like to including the BOM export.

Feel free to explore these options. Anything you set will only affect the BOM export, not your KiCad project.
When you are ready, click on "Generate BOM." The plugin will generate several HTML and supporting files and invoke your default web browser to display the BOM. Mine looks like this (Figure 13.14.3.18):

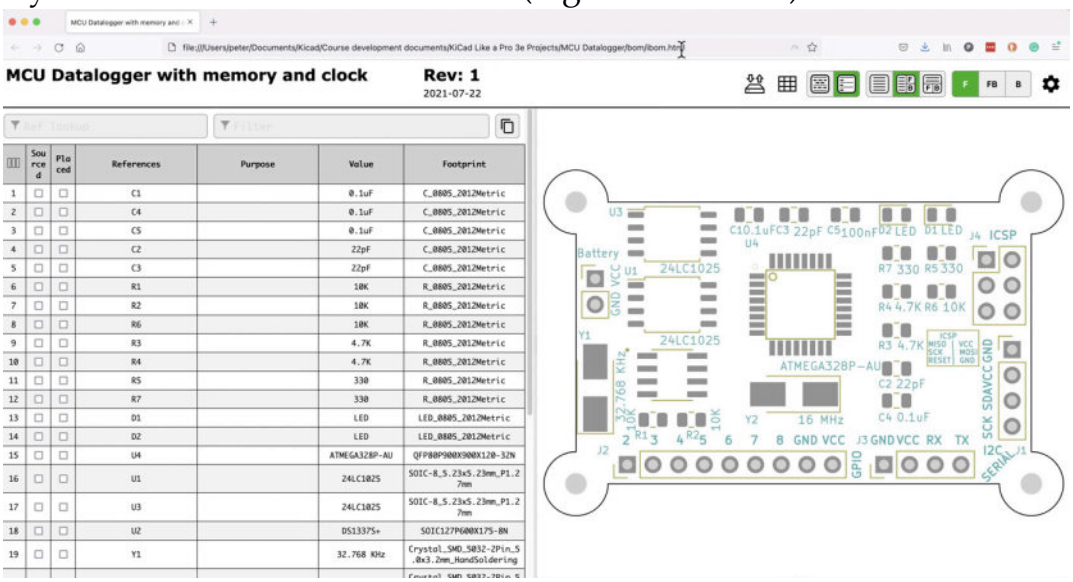
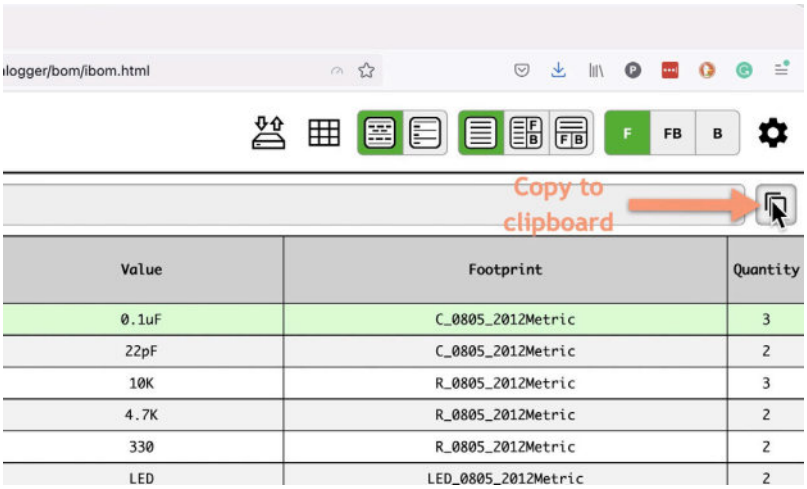


Figure 13.14.3.18: The HTML BOM in the browser.

As you move your mouse over various elements of the BOM page, you will notice that the page reacts. For example, try hovering the mouse pointer over one of the rows in the BOM table (on the left side of the page). The row is highlighted, and the corresponding footprint is also highlighted on the PCB rendering on the right side of the page.

Notice that the BOM table contains columns for each field you selected in the configuration window.

You can use the layout controls at the top-right of the page to change the configuration of the page, get a statistics summary of the board, export a board image, and switch between the front, back, or both sides of the PCB.
To export the BOM so that you can process it in a spreadsheet, click on the copy button located on the top right side of the page(Figure 13.14.3.18):



Value	Footprint	Quantity
0.1uF	C_0805_2012Metric	3
22pF	C_0805_2012Metric	2
10K	R_0805_2012Metric	3
4.7K	R_0805_2012Metric	2
330	R_0805_2012Metric	2
LED	LED_0805_2012Metric	2

Figure 13.14.3.18: Copy the BOM data to the clipboard so you can paste it into a spreadsheet or text editor.

You can then copy the BOM to a spreadsheet, as you did with the previous two methods (Figure 13.14.3.19):

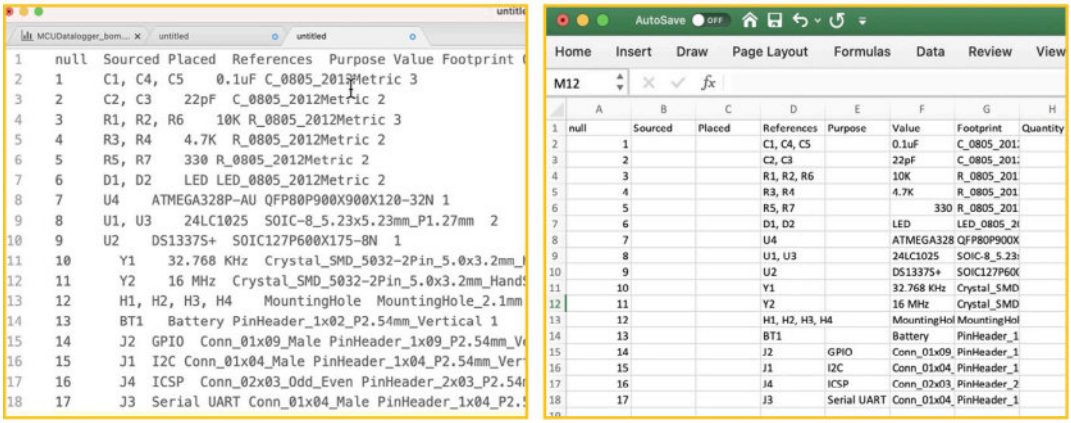


Figure 13.14.3.19: The HTML BOM data in a text editor and spreadsheet.

In summary, to export a Bill of Materials of your PCB project, you can use three methods. The first two are available in KiCad "out of the box, "while the third method requires installing a third-party plugin. The last two methods (the Edit Symbol Fields in Eeschema, and the Interactive HTML BOM plugin in Pcbnew) are equivalent in their ability to export custom fields. The last method, the Interactive HTML BOM plugin, is most suitable if you wish to publish complete BOM documentation on a website that includes full component information and renderings of the front and back of the PCB.

15. Import components from Snapeda

In this chapter, you will learn how to import symbols, footprints, and 3D models (3D shapes) in KiCad from Snapeda.com. To demonstrate the process, I have set up a new KiCad project. In this project, I want to use a component that does not exist in the Kicad libraries: the Texas Instruments MSC1212Y5PAGT central processing unit.

Snapeda is an online repository of electronics CAD components. In my day-to-day work with KiCad, I turn to Snapeda to look for symbols, footprints, and 3D shapes every time I need one that does not exist in the libraries that come with KiCad. Often, I also look at Snapeda for better versions of components that already exist in KiCad. Because it is such a helpful resource, I decided having a dedicated Recipe chapter in this book is worth it.

If you wish to follow along, go to https://www.snapeda.com and create a new free account, or log in to your account if you already have one. Below is the homepage of snapeda.com (Figure 13.15.1):

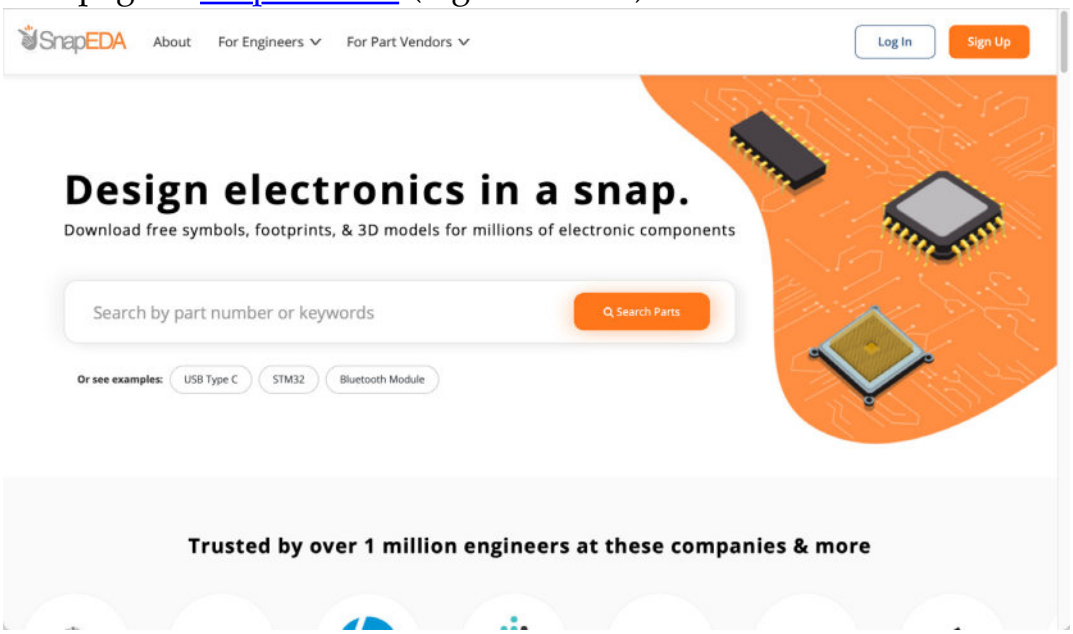


Figure 13.15.1: The Snapeda homepage.

Once you have logged in to Snapeda, use the search bar at the top of the page to search for the component you need. You can search with the model number, like "MSC1212Y5PAGT" or a keyword such as "Texas Instruments 8051 CPU". Snapeda will give you a list of hits. Select the one

that matches your requirements by clicking on it. In my case, I will be working with this component[7] (Figure 13.15.1):

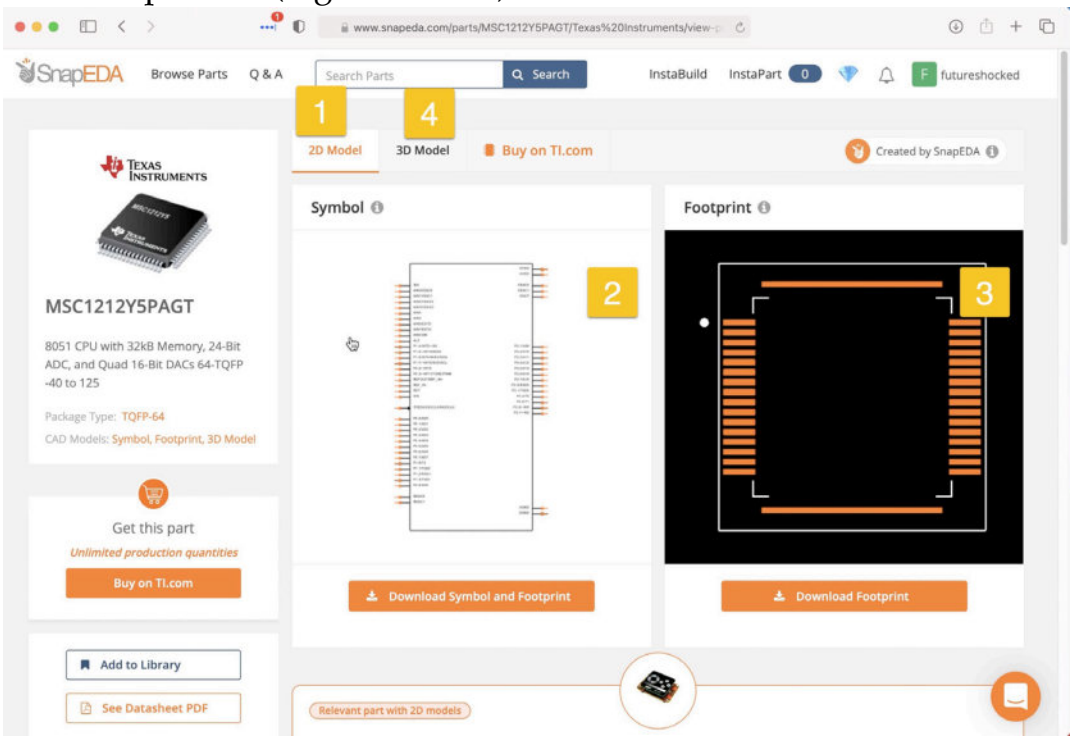


Figure 13.15.1: The TI MSC1212Y5PAGT CPU component.

The component page contains two main tabs: 2D model (1) and 3D model (4). Unde the 2D Model tab you will find the Symbol (2) and the footprint (3). To see the 3D model, click on the 3D Model tab. It looks like this (Figure 13.15.2):

[7] MSC1212Y5PAGT
8051 CPU with 32kB Memory, 24-Bit ADC, and Quad 16-Bit DACs 64-TQFP -40 to 125

https://www.snapeda.com/parts/MSC1212Y5PAGT/Texas%20Instruments/view-part/41461/?ref=search&t=MSC1212Y5PAGT

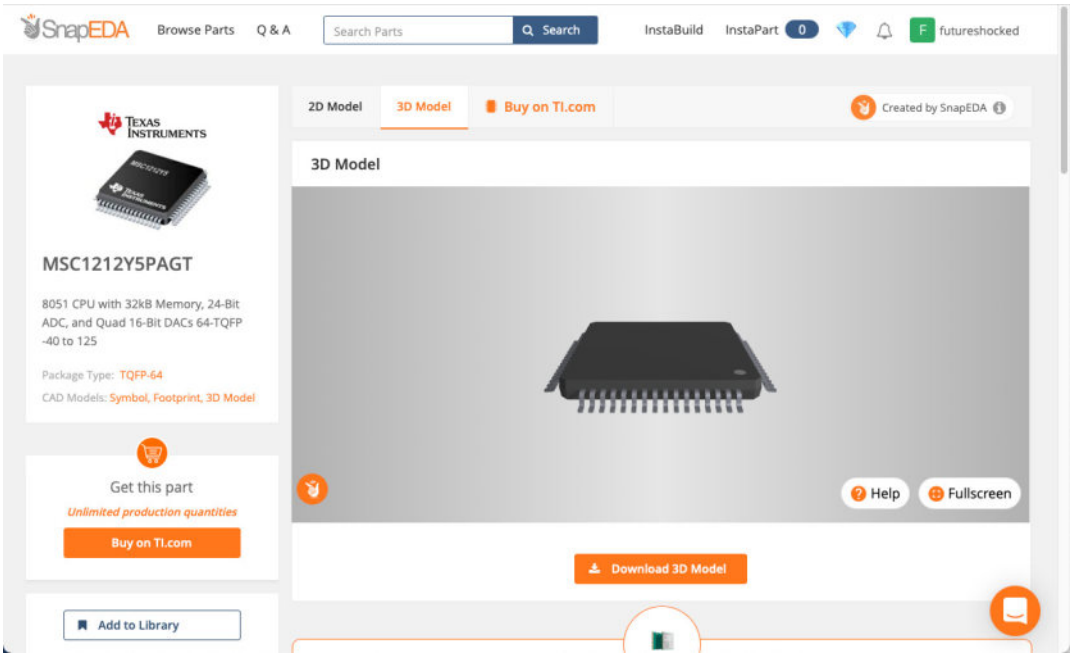


Figure 13.15.2: The 3D model of the component in question.

Available" column will indicate whether the 3D model and other parts of the component are available. In the example in Figure 13.15.3 (below), notice the icons under "Data Available":

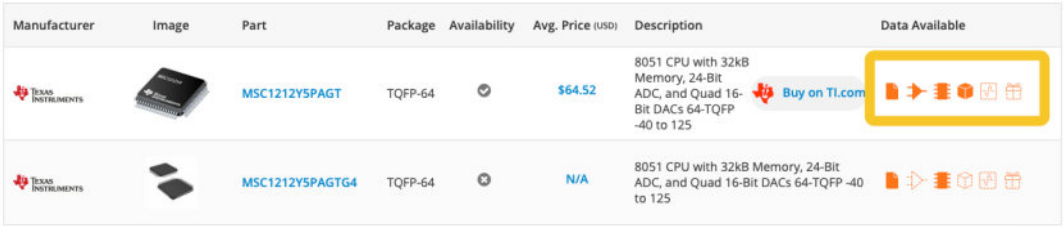


Figure 13.15.3: The available data for the search results.

A solid-colored icon indicates that data is available. From left to right:

- Datasheet.
- Symbol.
- Footprint.
- 3D model.
- Simulator model.
- Sample.

The MSC1212Y5PAGT CPU component that I use in this example provides the datasheet, symbol, footprint, and 3D model.

Let's download the symbol, footprint, and 3D model. Click on the 2D Model tab. Then, click on the "Download Symbol and Footprint" button. Snapeda will ask you to select the CAD software you will be using, so click on "KiCad." As you probably expect, this will download the symbol and the

footprint, but it will also download the 3D model. Snapeda will archive these files in a single ZIP file. Extract them from the ZIP file (Figure 13.15.4).

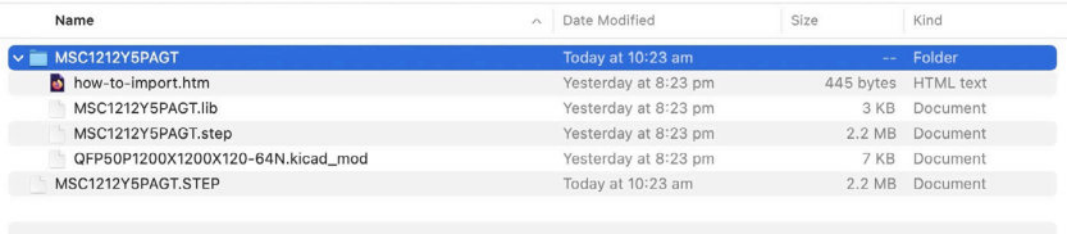


Figure 13.15.4: The symbol, footprint, and 3D model files.

The folder I extracted from the ZIP file contains:

- Symbol file, with the ".lib" extension.
- Footprint file, with the ".kicad_mod" extension.
- 3D model file, with the ".step" extension.

Now that you have these files on your computer import them into KiCad. You can learn how to do this in earlier chapters in this book. See here for symbols, here for footprints, and here for 3D shapes.
I have imported the files to my instance of KiCad, and I'll go ahead to use them. I'll start with the symbol. In my schematic editor, I type the "A" hotkey to add a new symbol. This brings up the Symbol Chooser. I type a few of the first letters for the new component to locate it in the library quickly. The Chooser promptly finds the symbol. Click on it to select it and show its preview in the right-side pane (Figure 13.15.5):

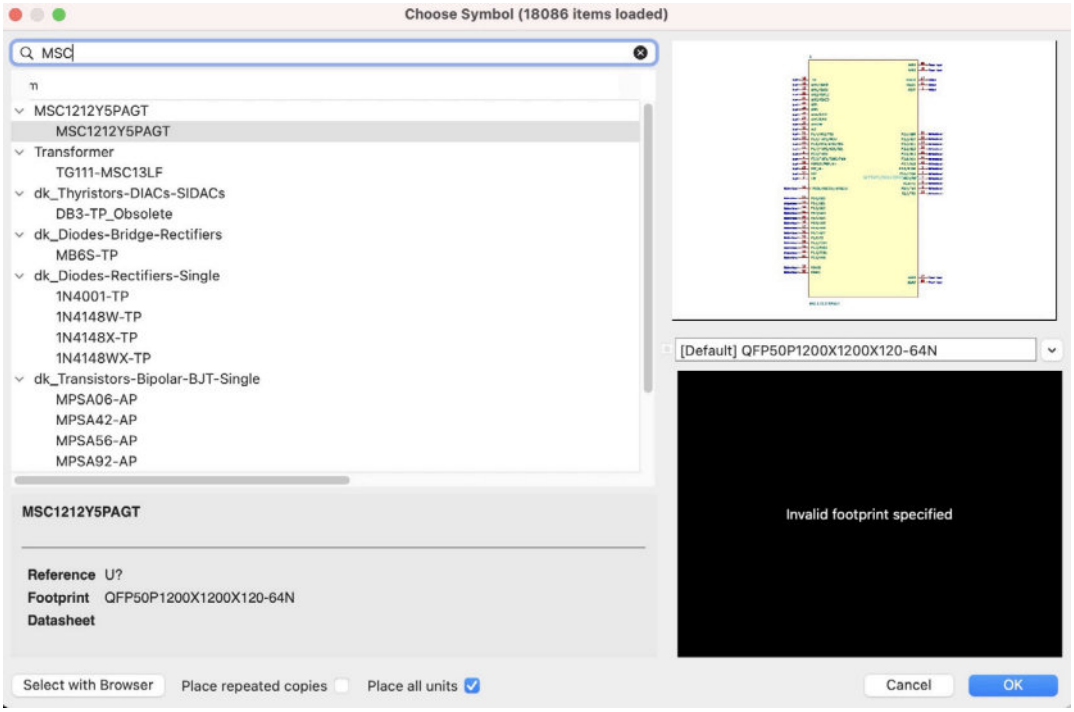


Figure 13.15.5: The new symbol.

Double-click on the symbol to add it to the schematic editor (Figure 13.15.6):

Figure 13.15.6: The new symbol in the editor.

Once the new symbol is in your schematic editor, you can use it as you would with any other symbol. Let's continue with the footprint for the same component. I assume that you have already imported the footprint to KiCad. Double-click on the symbol to open up the symbol properties window. It looks like this (Figure 13.15.7):

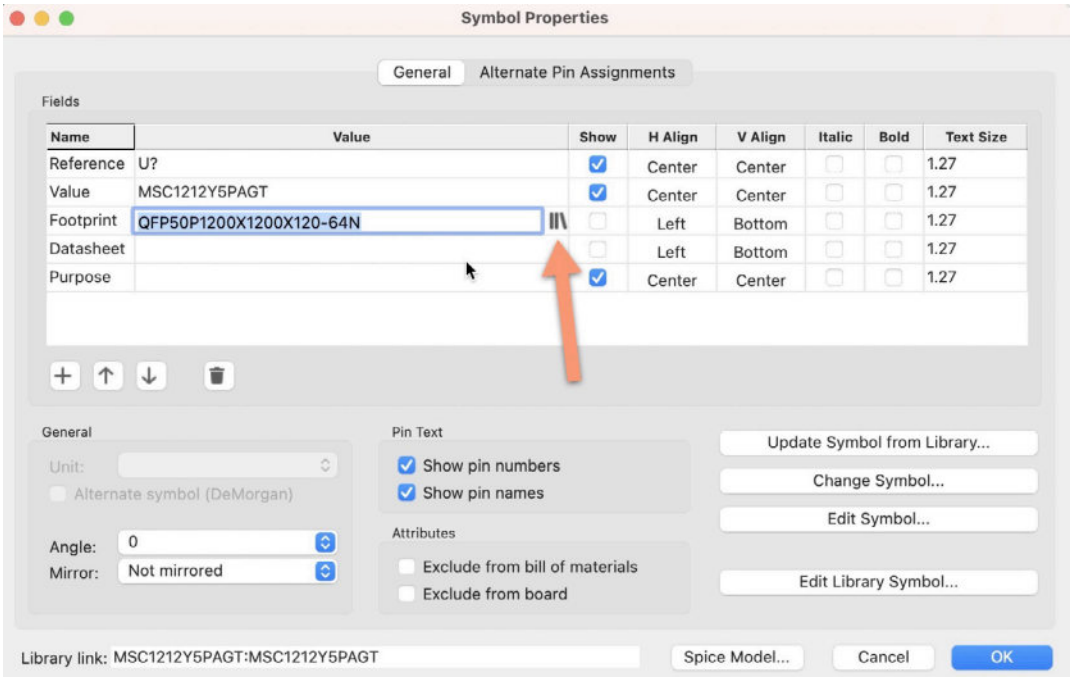


Figure 13.15.7: The new component symbol properties window.

The "Footprint" field contains a default footprint that exists in the KiCad footprints library, the QFP50. I prefer to use the one that I downloaded from Snapeda, so click on the library button (marked by the arrow in the screenshot above). This will bring up the library footprint chooser, which you can see below (Figure 13.15.8):

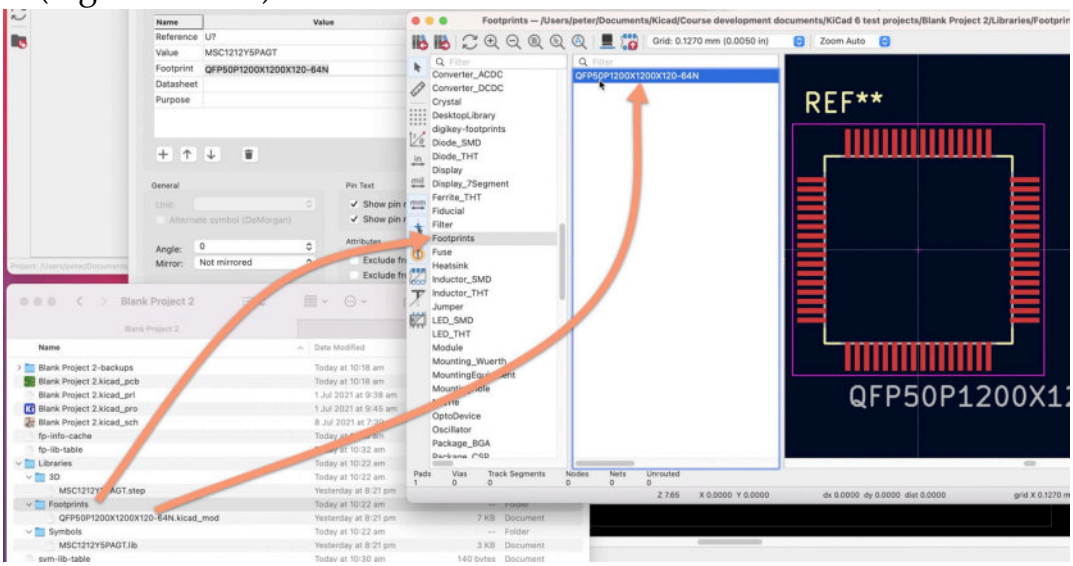


Figure 13.15.8: The new footprint library.

I imported the new footprint using the name "Footprints" as the library name. For this reason, the footprints chooser window shows the footprint file under "Footprints". You should adjust your search by taking into account the name of the new footprint library that you used during the import process.

Double-click on the footprint to complete the association. The footprint chooser window will close, and you will be back at the symbol properties window (Figure 13.15.8).

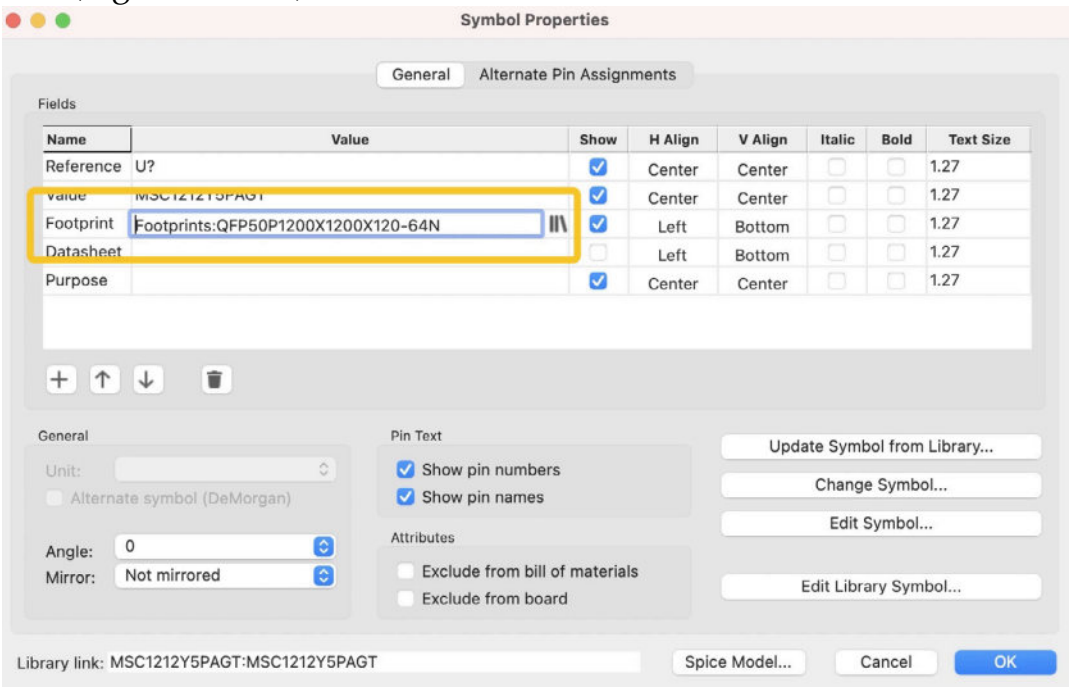


Figure 13.15.8: The new footprint and symbols are now associated.

I have now associated the new symbol with its matching footprint. In the screenshot above, notice how the associated footprint belongs to the "Footprints" library.

Next: let's make use of the new 3D shape. To do that, open Pcbnew, update the layout editor with the schematic data, and draw a rectangle graphic around the only footprint in the Edge.cuts layer (Figure 13.15.9).

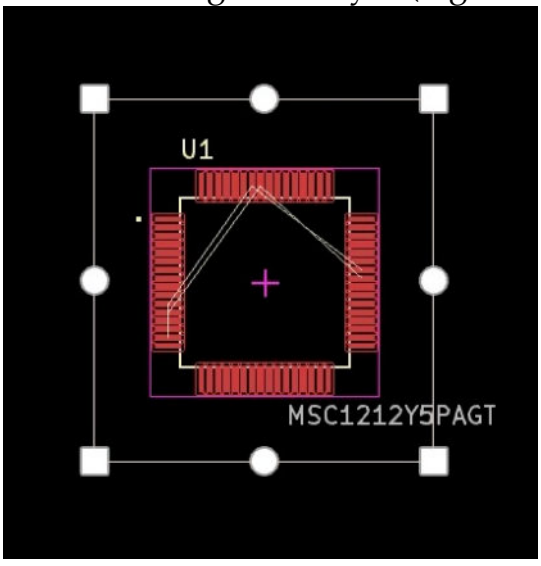


Figure 13.15.9: The new footprint in Pcbnew with an Edge.cuts graphic.

I will set a 3D shape for the new footprint. Double-click on the footprint, and click on the 3D Models tab. By default, this footprint does not have a 3D model defined, so there are no rows in the 3D Model(s) list (Figure 13.15.10).

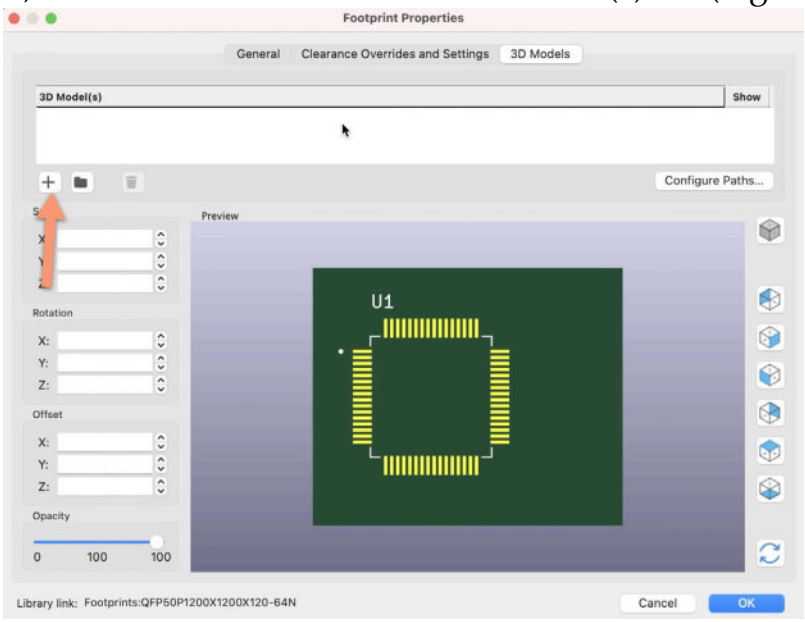


Figure 13.15.10: The new footprint's 3D Models tab.

To add a new 3D symbol, click on the "+" button to add a new row (Figure 13.15.11):

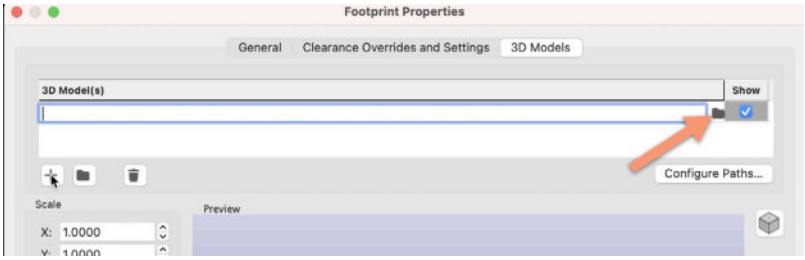


Figure 13.15.11: A new row to hold the path for the 3D model.

Click on the folder button (right side of the new 3D model row). This will bring up a file browser to locate and select the 3D shape (".step") file. In my example, it looks like this (Figure 13.15.12):

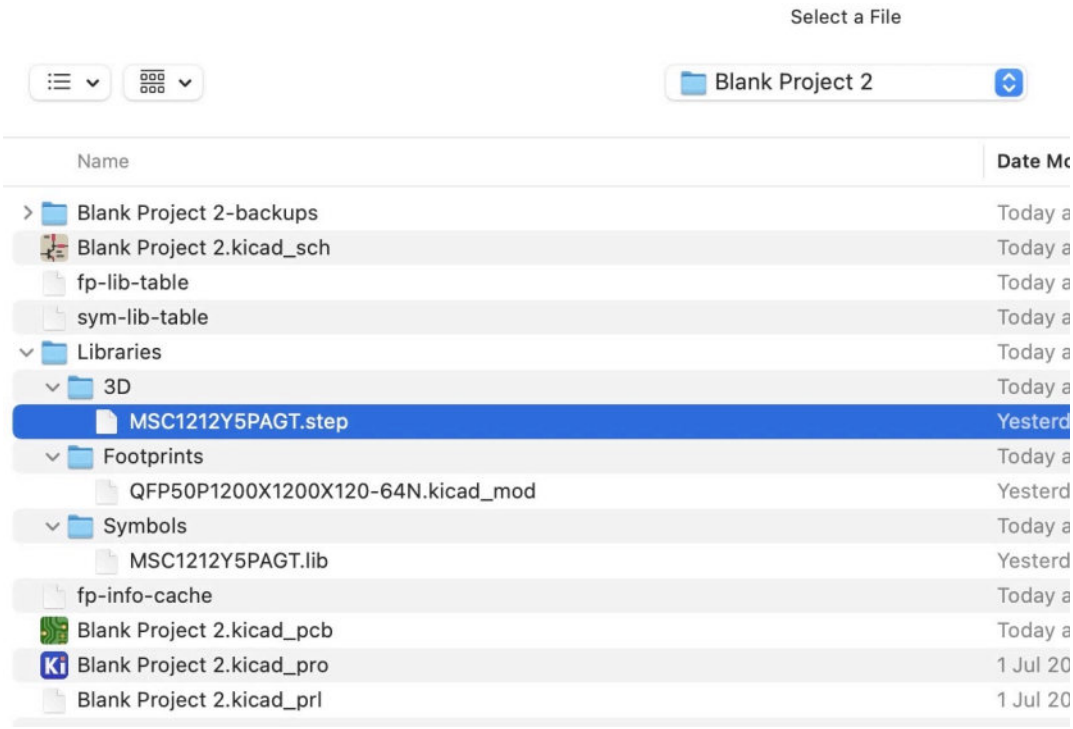


Figure 13.15.12: Double-click on the 3D model ".step" file to add it to the footprint properties.

Double-click on the ".step" file to add it to the footprint properties. The Preview pane will super-impose the model over the footprint, but it may be out of place at first. You will need to adjust some of the placement parameters (scale, rotation, and offset) to make the model align with the footprint. Below you can see my settings (I only changed the X rotation to -90°) (Figure 13.15.13):

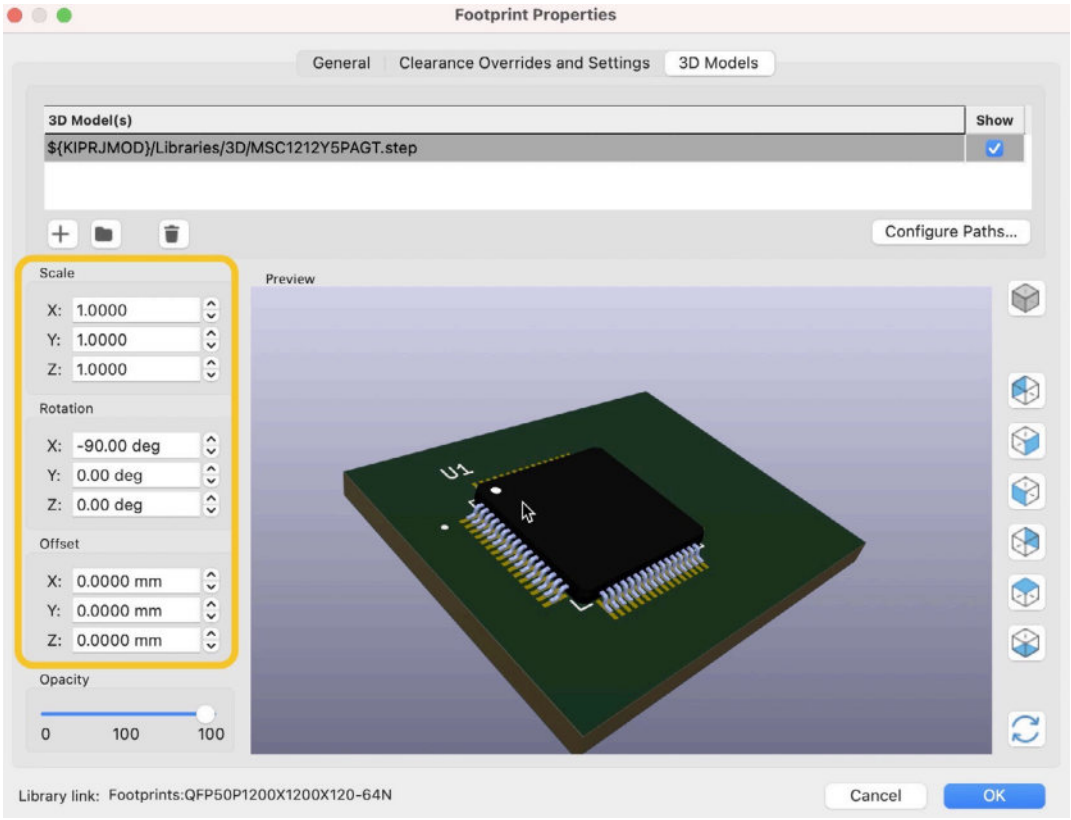


Figure 13.15.13: With a bit of tweaking, the 3D model aligns with the footprint.

Click OK to dismiss the window. Back in Pcbnew, bring up the 3D Viewer from the View menu to see the new 3D model (Figure 13.15.14):

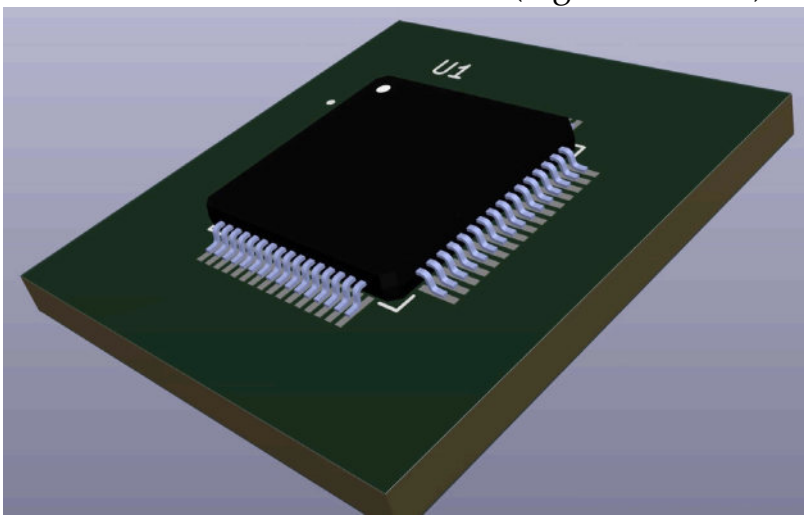


Figure 13.15.14: The new 3D model as it appears in the 3D Viewer.

Snapeda is an excellent source for symbols, footprints, and 3D shapes, but not the only one. I remind you that in previous chapters (see here and here) I have discussed others sources that you can consider when you are looking for components for your projects.

16. The Freerouting autorouter

The routing process of a new PCB is often the most time-consuming one. Especially in PCBs with many pins and confined space, routing can be tedious and frustrating. I find that an autoroute can help in such cases. An autorouter is software that can automatically create tracks using various algorithms optimized for the minimization of the total track length or vias used. Most modern autoroutes offer extensive customizability.

Should you use an autorouter?

Opinions differ, and appropriate use cases for autoroutes vary. Autorouters tend to be "naive." While the designer may customize an autorouter to some extent, autorouters do not "understand" the specific peculiarities of the various circuit segments on which they operate. For example, an autoroute will not be able to "understand" the importance of the shape and length of a differential track that connects a high-speed CPU with a memory chip or the track that connects an RF component to an antenna. Autorouted boards also tend to look less refined, with tracks drawn in a mechanical way distinctively different to tracks drained by a skilled designer.

Professional PCB designers tend to rely on manual routing for all but the most trivial PCBs. Often, autoroutes are used extensively during the prototyping process. The designer will often edit an autorouted board to improve it.

With all that in mind, auto-routing is a tool that is worth knowing and used when appropriate and with care. This chapter will show you how to use an external auto-router that works well with KiCad 6, Freerouting.

Below is an example of what an autoroute board looks like using Freerouting (left side). The autorouted completed the operation in approximately 3 minutes. On the right side of the screenshot is the same board, routed by myself. It took me around 90 minutes to complete this work (Figure 13.16.1).

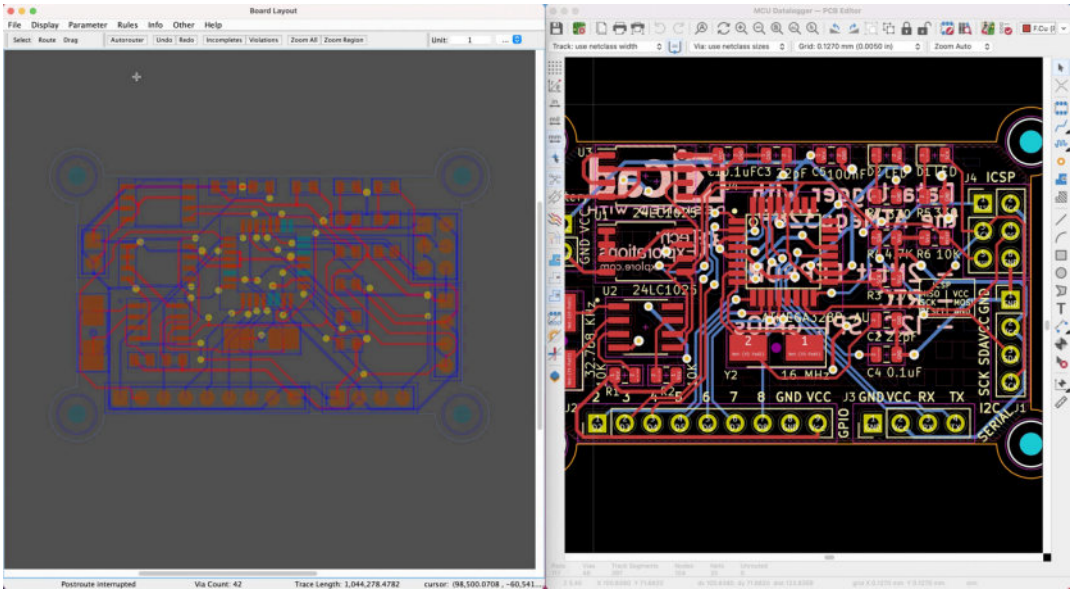


Figure 13.16.1: The same board routed automatically (left) and manually (right).

"Freerouting" is not only an autorouter. You can use it to manually route a board, similar to how you can use Pcbnew.

Below is an example (Figure 13.16.2). I have deleted some of the autorouted tracks, enabled the Route functions, and used the mouse to draw a new route that connects the two pads. Freerouting "knows" which pads and pins can be connected because I have imported a file that I generated earlier in KiCad (you will learn how to use Freerouting with KiCad later in this chapter).

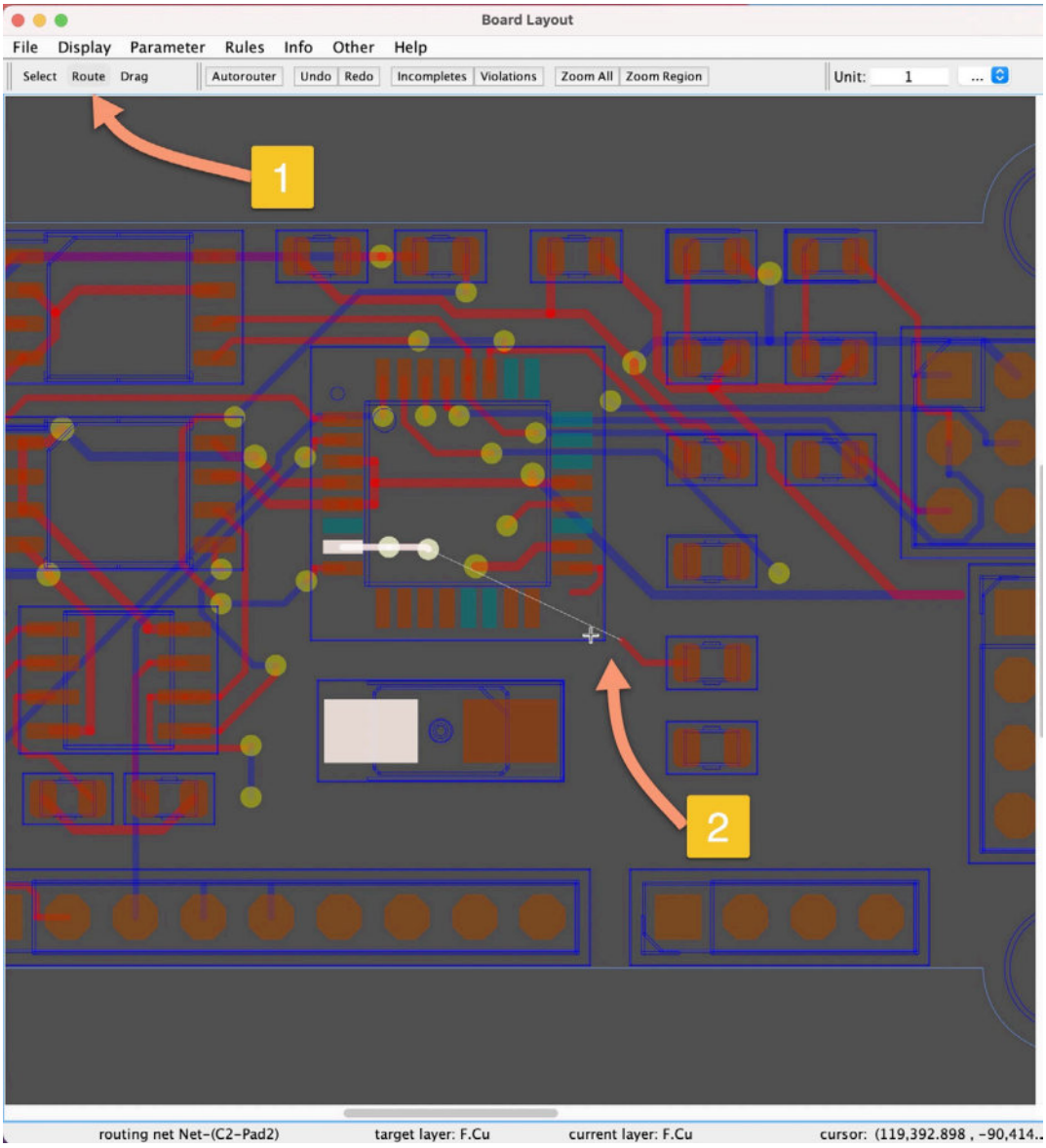


Figure 13.16.2: Freerouting is also useful as a manual router.

Freerouting is an open-source Java application that works with the KiCad, Eagle, and LayoutEditor CAD packages.

It works as a separate and distinct application; that is, it does not integrate with KiCad. It can exchange information with KiCad utilizing import and export files. It is a bit rough in the edges, as it can be challenging to learn and get used to its user interface.

Freerouting can automatically route a PCB in multiple layers, and it is very configurable. You can configure many aspects of the autoroutes operation, such as the preferred direction of the traces on each layer, whether to use vias or not, and whether to do a post-route pass to try and reduce the number of vias or total trace length.

In the following few segments in this chapter, I will show you how to install Freerouting on MacOS, Linux Kubuntu, and Windows 10 and then use

it to route a test board in two and four layers. Freerouting, being a Java application, can potentially work on any operating system with the Java runtime environment installed. In practice, I have noticed problems that seem to be related to the application's graphical libraries. After trial and error, I have found that Kubuntu is a good choice for working with KiCad and Freerouting on Linux, especially compared to Ubuntu Linux.

16.1. Install and start FreeRouting on MacOS

In this segment, you will learn how to install Freerouting on MacOS and start the application. Let's begin.
Freerouting is a Java application, so you will need to install the Java runtime environment on your Mac. MacOS ships with the JRE, so there is a good chance that you don't need to do anything else at this point. You can find the JRE downloads page here: https://www.java.com/en/download/manual.jsp or https://jdk.java.net/16/ .

To confirm that your Mac is ready to run Java applications, bring up a terminal window and type this command:

```
% java --version
```

Below is the response that I see on my iMac, OpenJDK 16.0.1 (Figure 13.16.1.3):

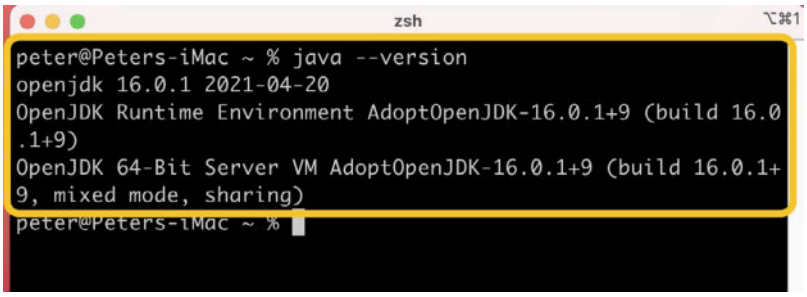


Figure 13.16.1.3: The JRE version that is running on my iMac.

With the JRE sorted, continue to Github to download the compiled version of the Freerouting application. The download page is at https://github.com/freerouting/freerouting/releases (Figure 13.16.1.4).

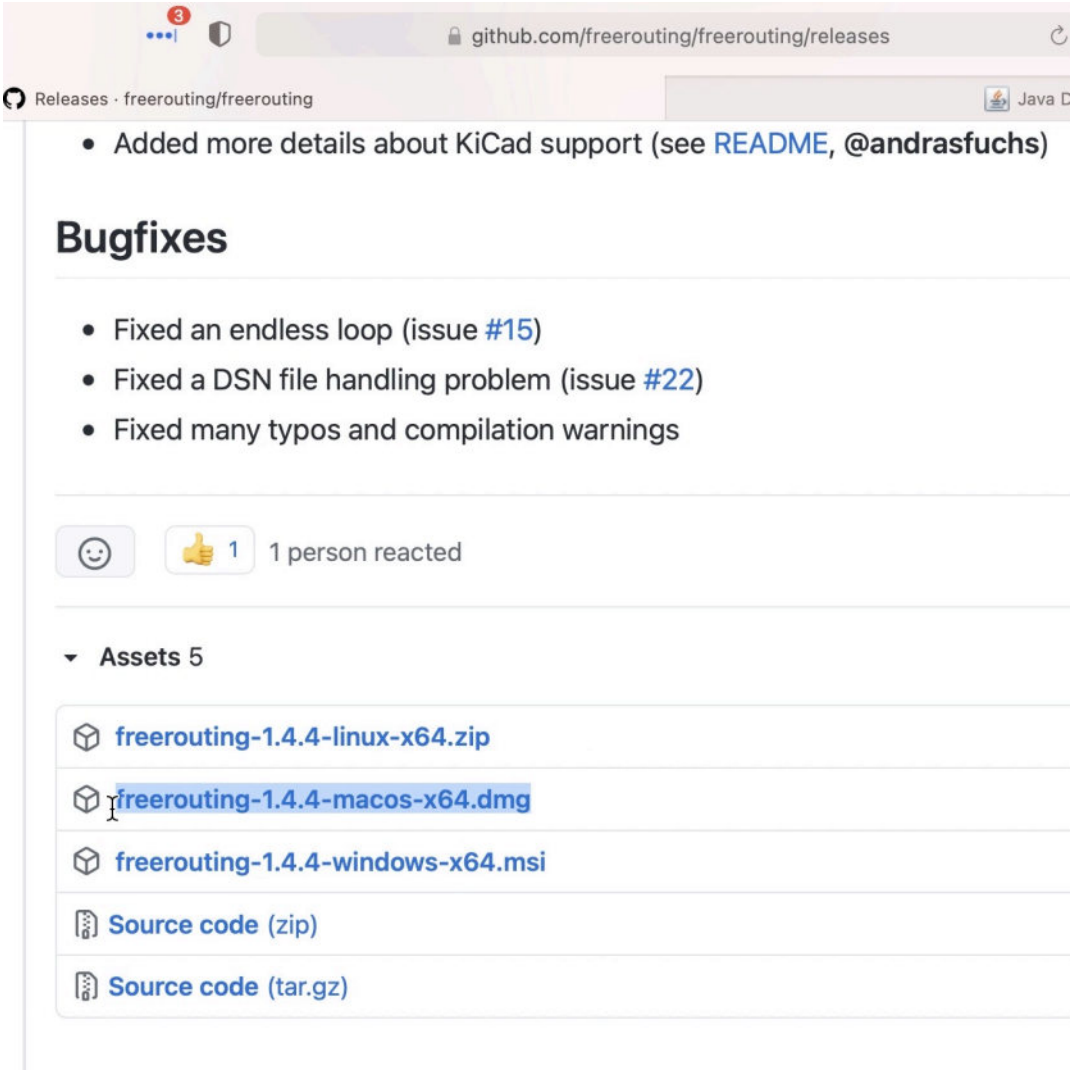


Figure 13.16.1.4: The Freerouting downloads page.

At the time I am writing these lines, the latest available version of Freerouting is 1.4.4. As you can see above, the project offers compiled versions for Linux, MacOS, and Windows.

Download the DMG file for MacOS, then double-click on it to mount the virtual disk in your Mac's file system (1) and click "Agree" (2) in the dialog box that appears (Figure 13.16.1.5).

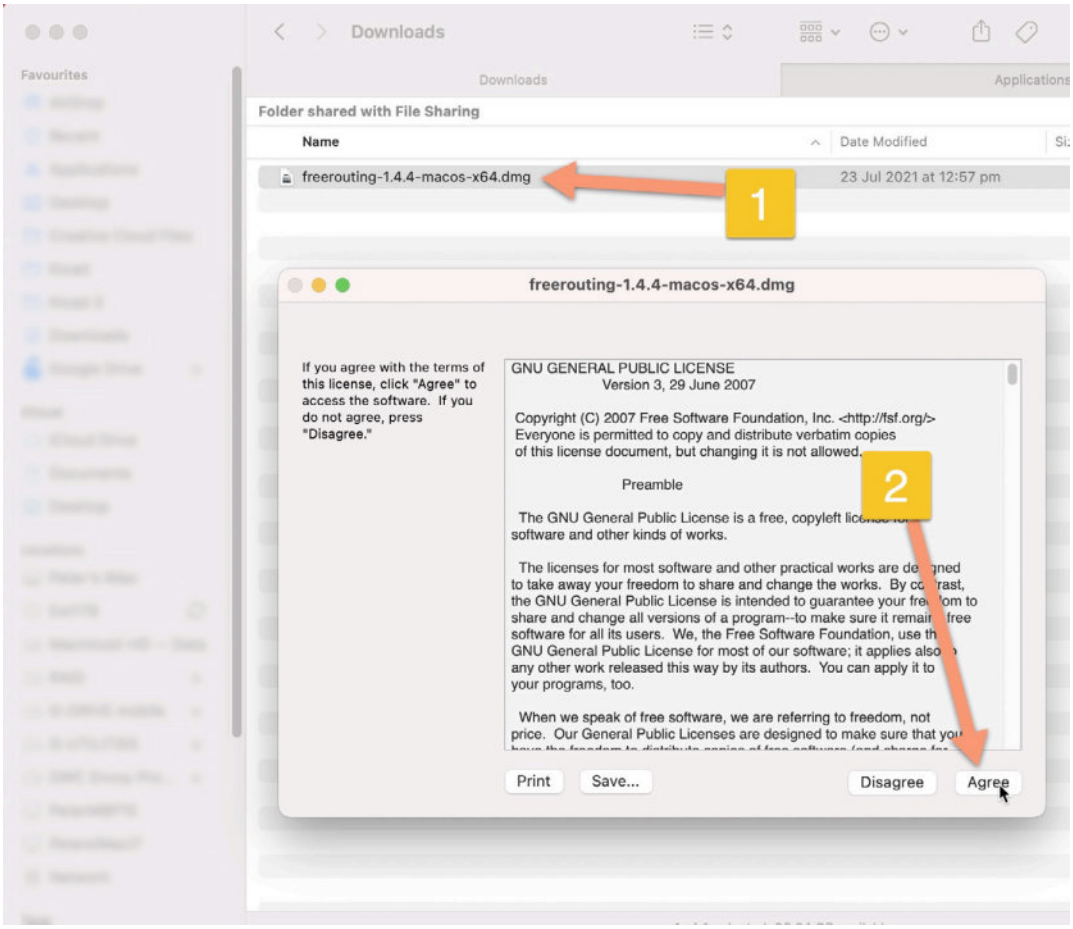


Figure 13.16.1.5: Mount the Freerouting virtual disk on the MacOS file system.

The Freerouting virtual disk is now mounted on your Mac's file system. Click on it (1), and you will see a single application file. Double-click on the application file (2) (Figure 13.16.1.6):

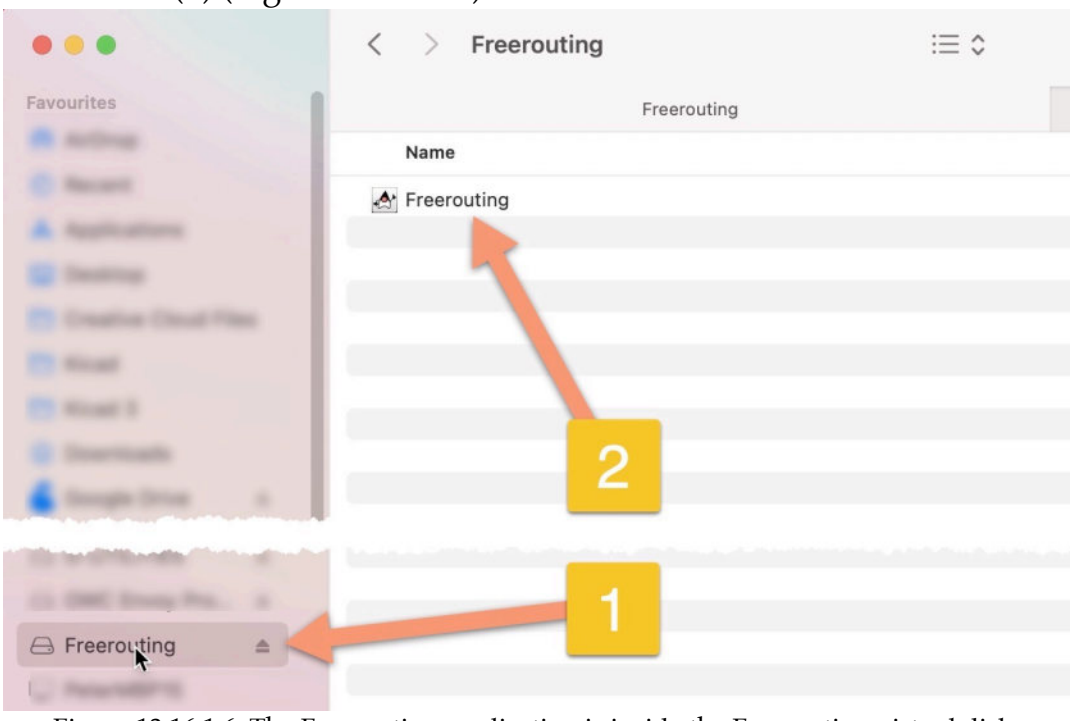


Figure 13.16.1.6: The Freerouting application is inside the Freerouting virtual disk.

Unfortunately, there seems to be a bug in the file that prevents the Freerouting application from starting. You should see an error message like this (Figure 13.16.1.7):

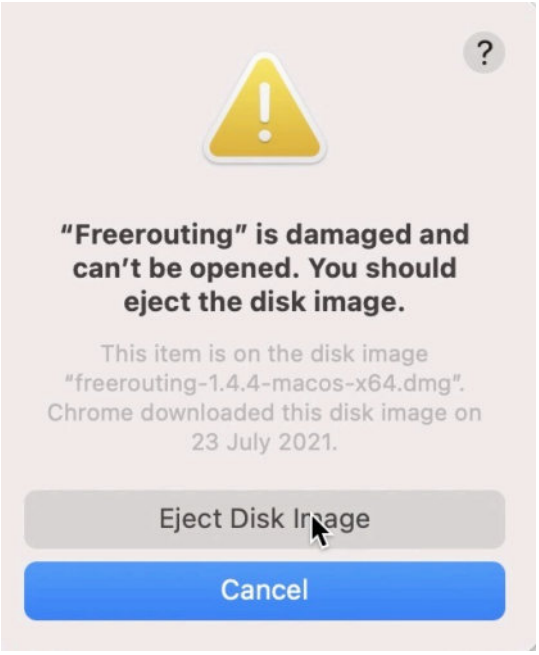


Figure 13.16.1.7: No, Freerouting is not damaged. Don't eject the disk!

This message is misleading. Freerouting is not damaged. However, its launcher seems to contain a bug that prevents the application from starting. Thankfully, this is something that we can quickly fix.

Right-click on the application file to show the context menu, and click on "Show Package Contents" (Figure 13.16.1.8).

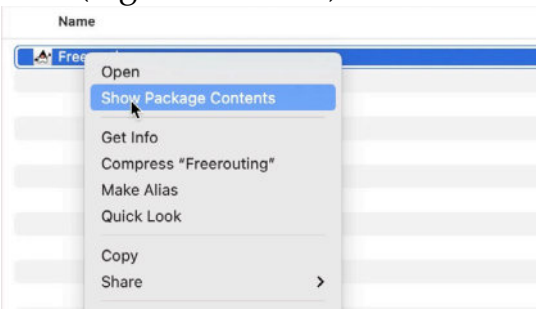


Figure 13.16.1.8: The application package contains the actual Freerouting application file.

This will reveal the files contained within the package. You can browse these files. Go in the "Contents" and the "app" directories until you see the "freerouting-executable.jar" file (Figure 13.16.1.9).

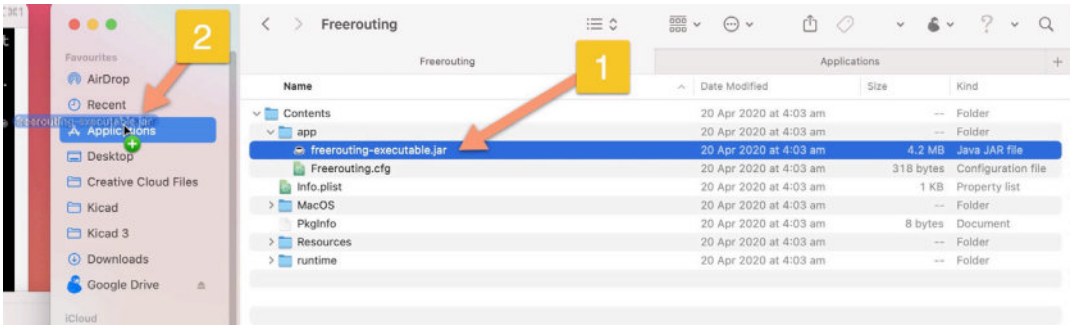


Figure 13.16.1.9: Drag "freerouting-executable.jar" in the Applications folder.

The file with the ".jar" extension is the actual Java application. You can start the application at its present location. But, if you plan to use Freerouting in the future, it is better to copy it into the Application directory. Drag and drop "freerouting-executable.jar" into the Applications directory (Figure 13.16.1.10).

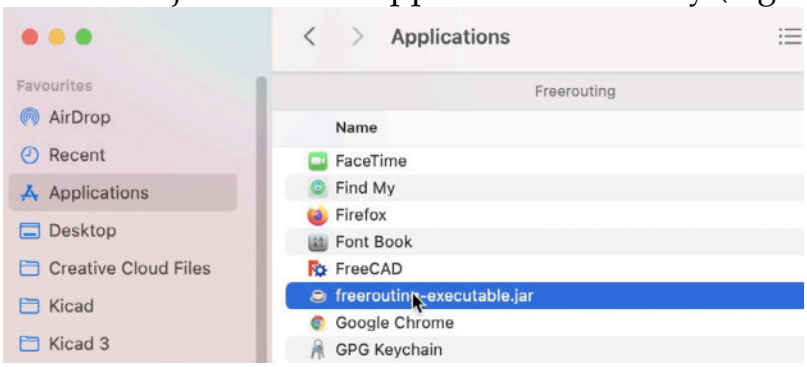


Figure 13.16.1.10: The Freerouting application is in the Application directory.

Because of MacOS security, the first time you attempt to start Freerouting you will see a message like this (Figure 13.16.1.11):

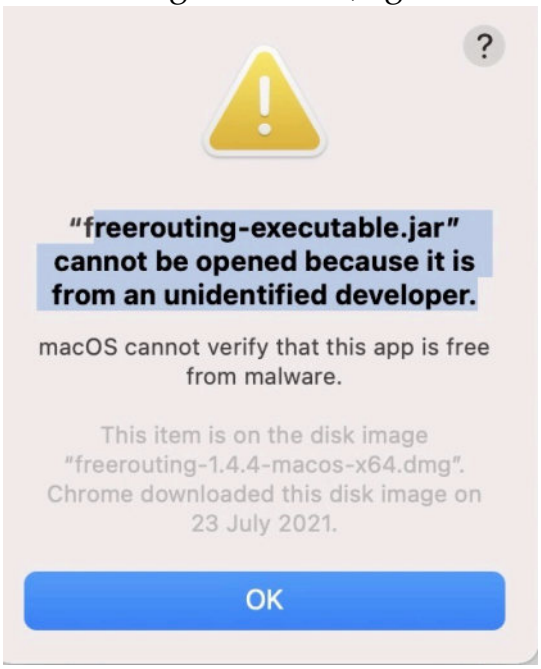


Figure 13.16.1.11: Freerouting is from an unidentified developer and MacOS will block it from starting.

You will need to permit MacOS to start Freerouting. Click on the Apple menu (top left of the screen) and then click on System Preferences. In the Preferences window, select the “Security & Privacy” widget (1) (Figure 13.16.1.12).

Figure 13.16.1.12: Allow Freerouting to start in the Preferences, Security & Privacy dialog.

In the Security & Privacy dialog box, click on General (2). Ensure that the padlock is open (3) and that “freerouting-executable.jar” is listed in the “Allow apps downloaded from:” segment. Then, click on “Open Anyway” (4). You’ll see a confirmation window; click OK to dismiss it.
After this process, Freerouting will start (Figure 13.16.1.13):

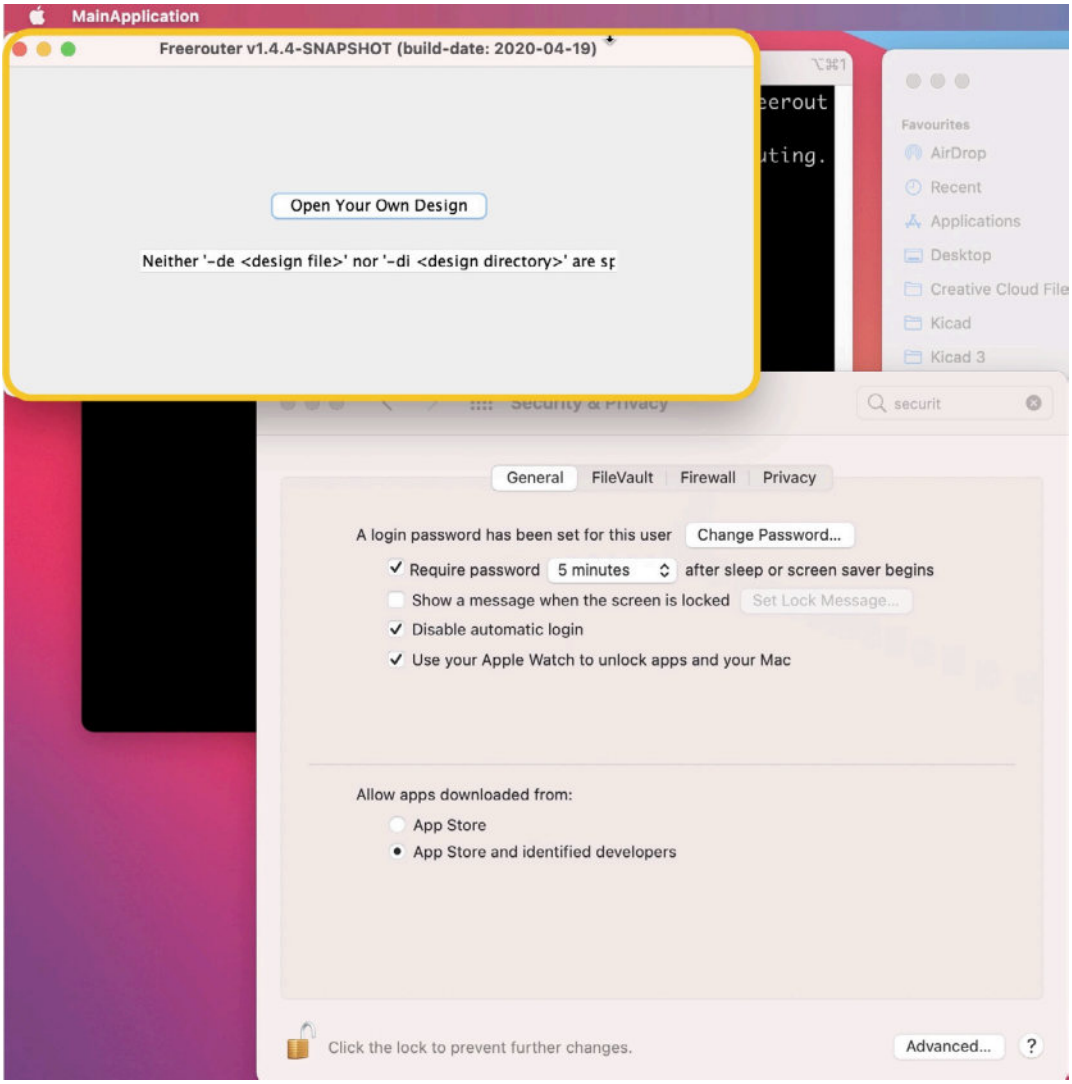


Figure 13.16.1.13: Freerouting starts for the first time.

The Freerouting window that appears contains a single button. When you click this button, you will see a file system browser that you can use to import a compatible file from KiCad. I'll show you how this works later in this chapter.

An alternative way to start Freerouter is to use the command line. Bring up a terminal window, and type this command:

```
% java -jar /Applications/freerouting-executable.jar
```

Below is a screenshot of my terminal (Figure 13.16.1.14):

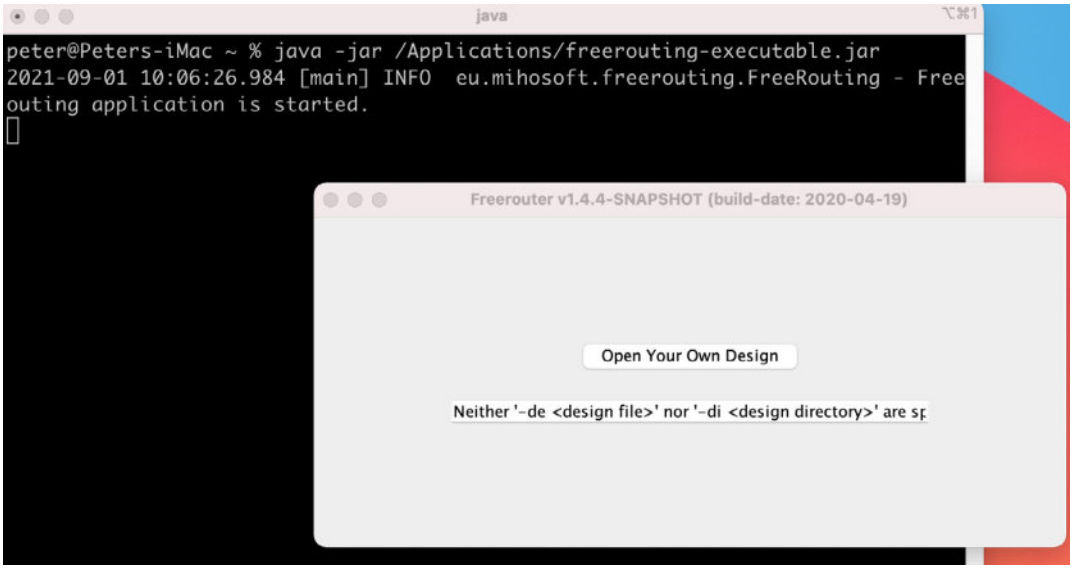


Figure 13.16.1.14: Starting Freerouter via the command line.

At this point, you have installed Freerouting on MacOS, and started the application. To learn how to use it alongside KiCad and perform auto-routing, jump to the 2-layer example. If you want to learn how to install Freerouting on Linux or Windows, continue reading.

16.2. Install and start FreeRouting on Linux Kubuntu

Freerouting works well with Linux Kubuntu, as does KiCad 6. Perhaps this is because Kubuntu uses the KDE desktop which seems to be more compatible with the GUI elements of both KiCad and Freerouting. Below you can see my Kubuntu desktop with the system information window (Figure 13.16.2.15):

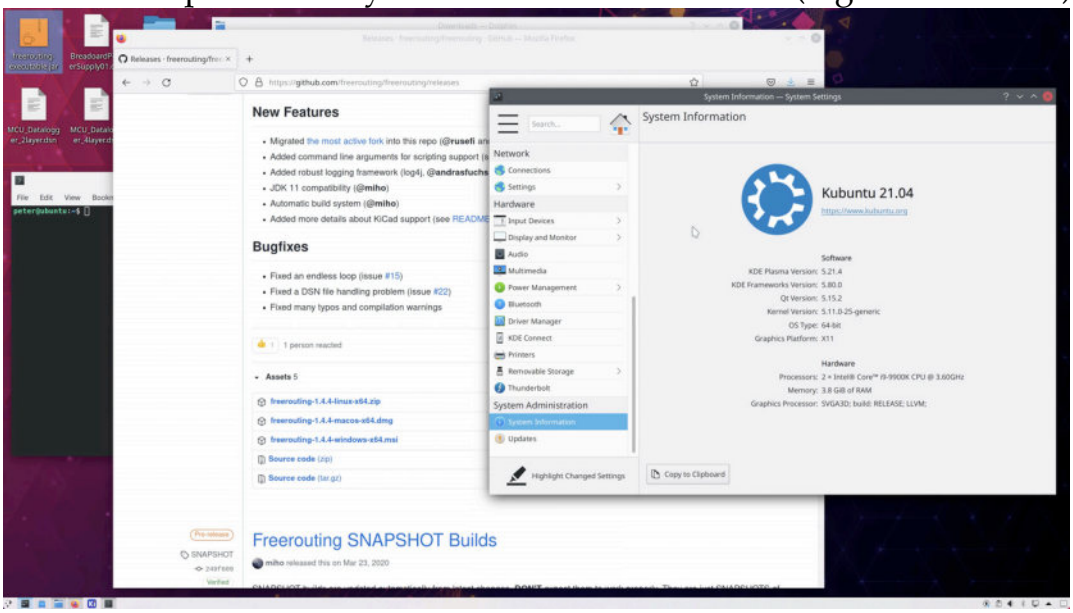


Figure 13.16.2.15: My Linux Kubuntu desktop.

Kubuntu contains a suitable Java runtime environment so that you can proceed with the download of the Freerouting application. The download

page is at https://github.com/freerouting/freerouting/releases (Figure 13.16.2.16).

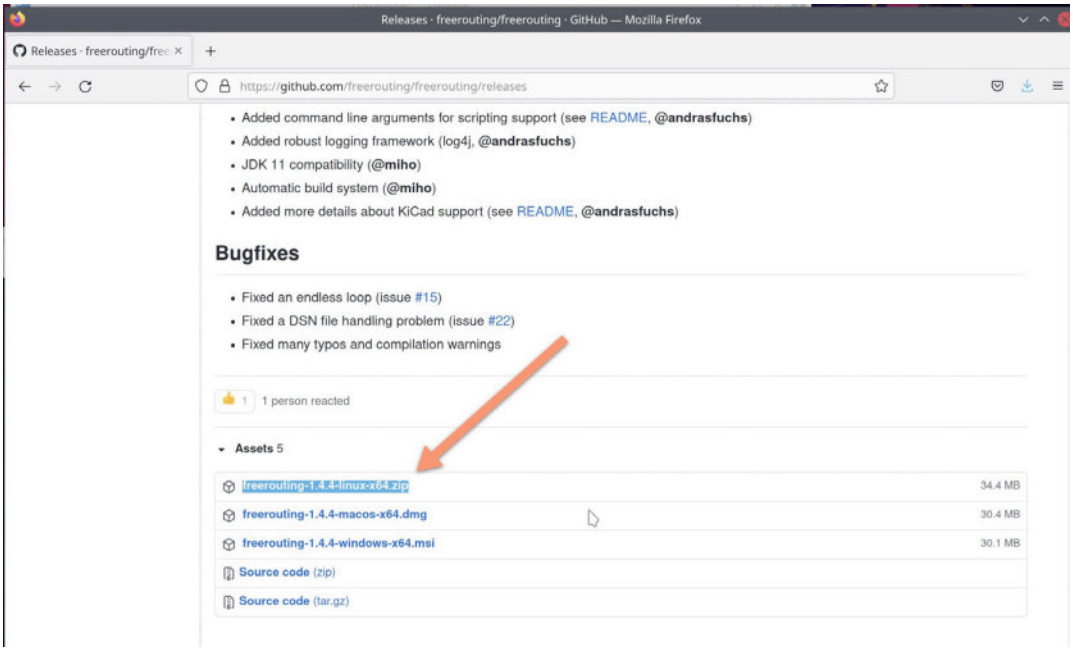


Figure 13.16.2.16: Download the Freerouting executable for Linux.

Extract the contents of the ZIP file, and browse the files (Figure 13.16.2.17).

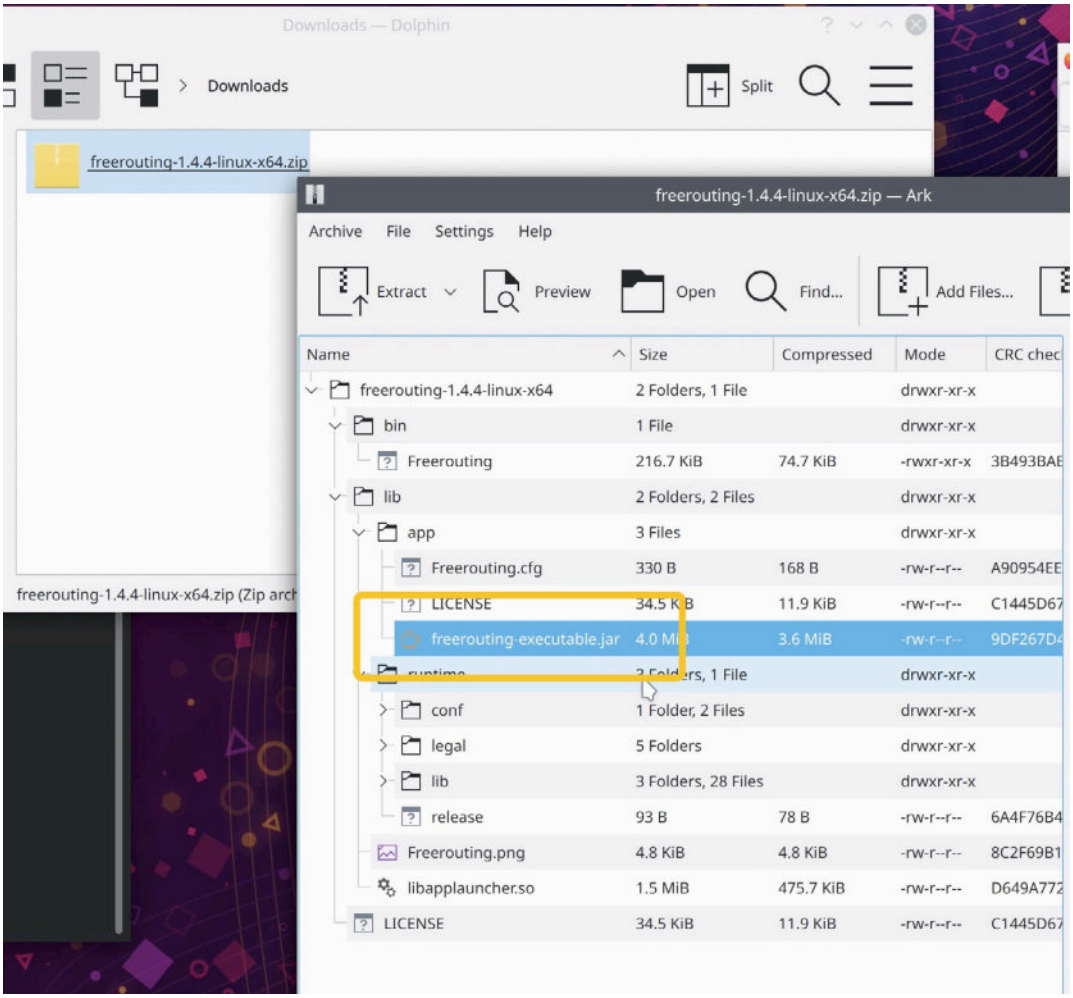


Figure 13.16.2.17: The contents of the Freerouting ZIP archive.

The only file that you need from the archive is the one with the ".jar" extension, under "freerouting-1.4.4-linux-x64", "bin," "lib," "app." You can copy this file at a convenient location on your Linux file system. I copied my instance on the desktop (Figure 13.16.2.18).

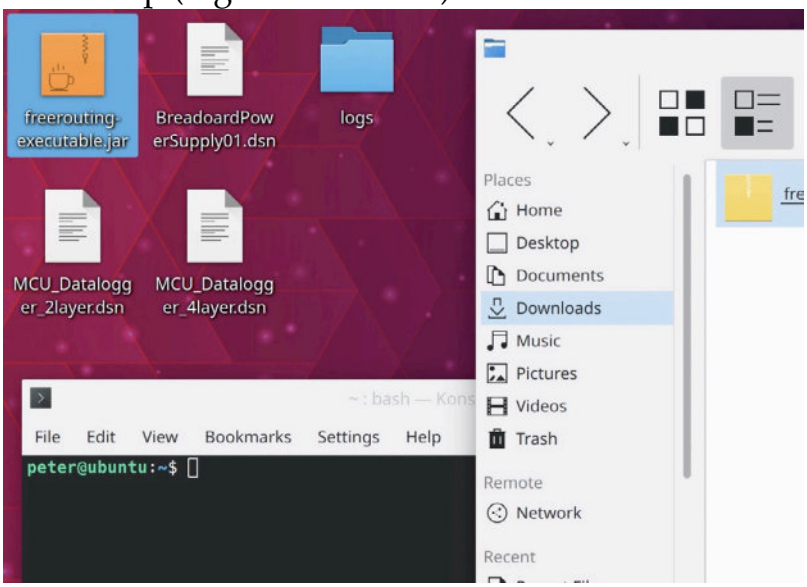


Figure 13.16.2.18: The Freerouting executable on my Kubuntu desktop.

To start Freerouting, you must use this command line instruction (I assume that your Freerouting executable is on the desktop) (Figure 13.16.2.19):

```
$ java -jar ~/Desktop/freerouting-executable.jar
```

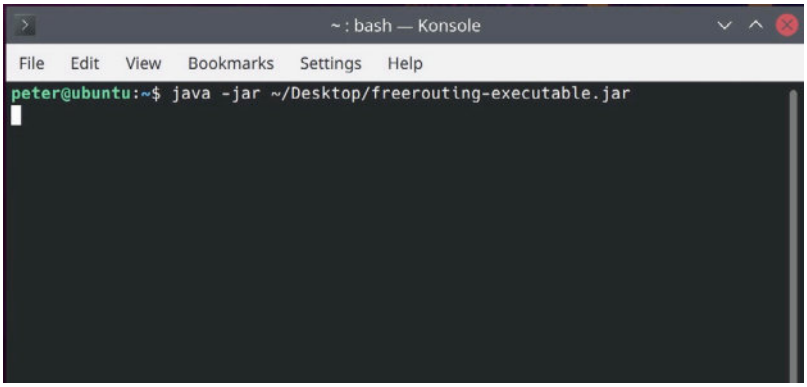


Figure 13.16.2.19: Starting Freerouting on the Kubuntu command line.

This will start the application. You should see the Freerouting window, like this (Figure 13.16.2.20):

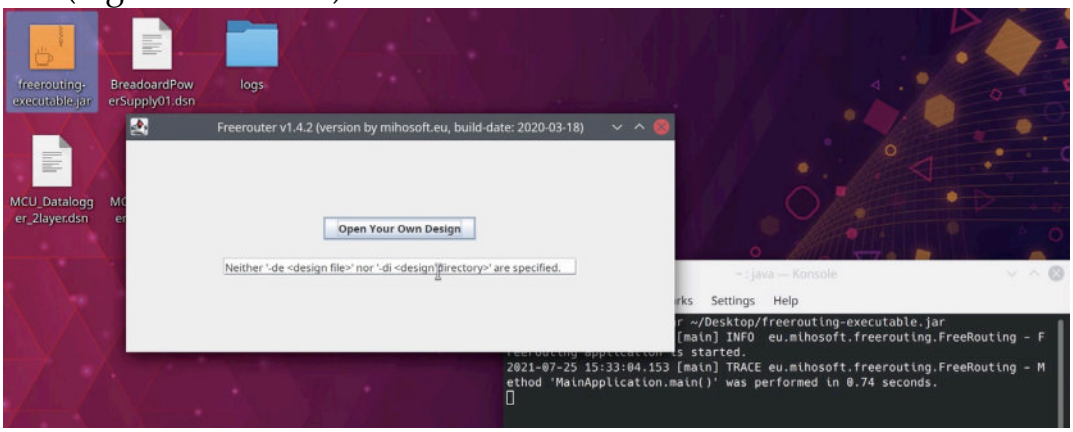


Figure 13.16.2.20: Freerouting has started and waiting for an import file from KiCad.

At this point, you have installed Freerouting on Linux Kubuntu and started the application. To learn how to use it alongside KiCad and perform auto-routing, jump to the 2-layer example.

16.3. Install and start FreeRouting on Windows

In this segment, you will learn how to install Freerouting in Windows 10. The process is simpler than in Linux or MacOS. The download page is at https://github.com/freerouting/freerouting/releases (Figure 13.16.3.21).

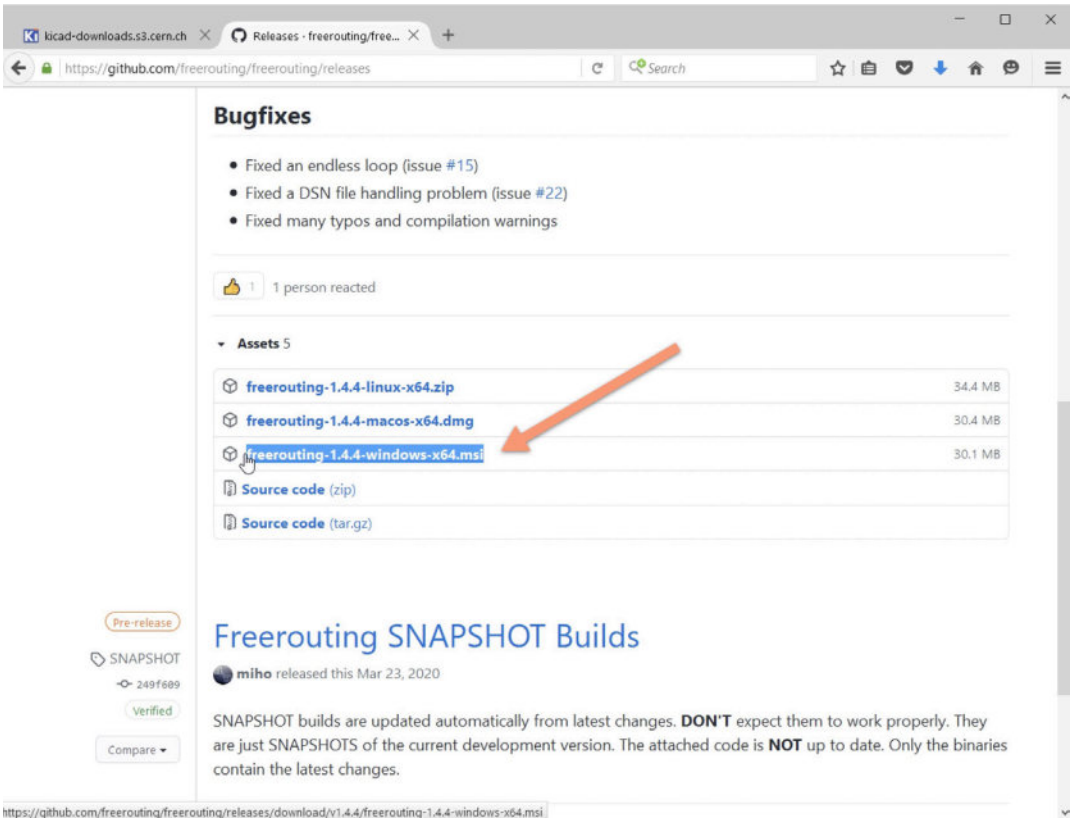


Figure 13.16.3.21: Download the Freerouting executable for Linux.

Click on the Windows ".msi" archive to download it. Double-click on the ".msi" file to start the installation process (Figure 13.16.3.22):

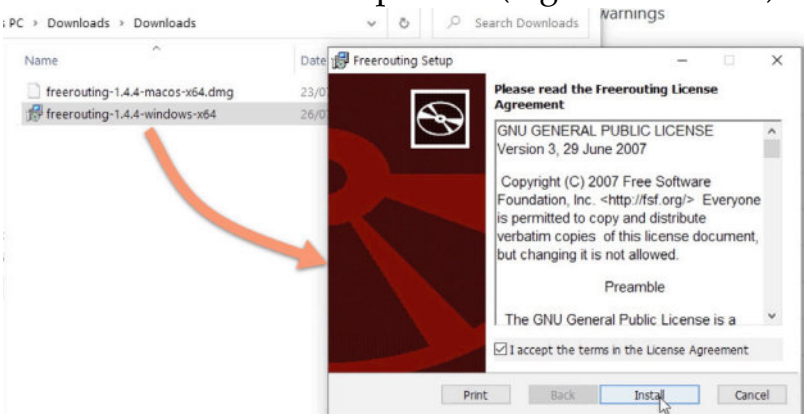


Figure 13.16.3.22: The Windows installer for Freerouting.

The installation process includes only a couple of steps (Figure 13.16.3.23):

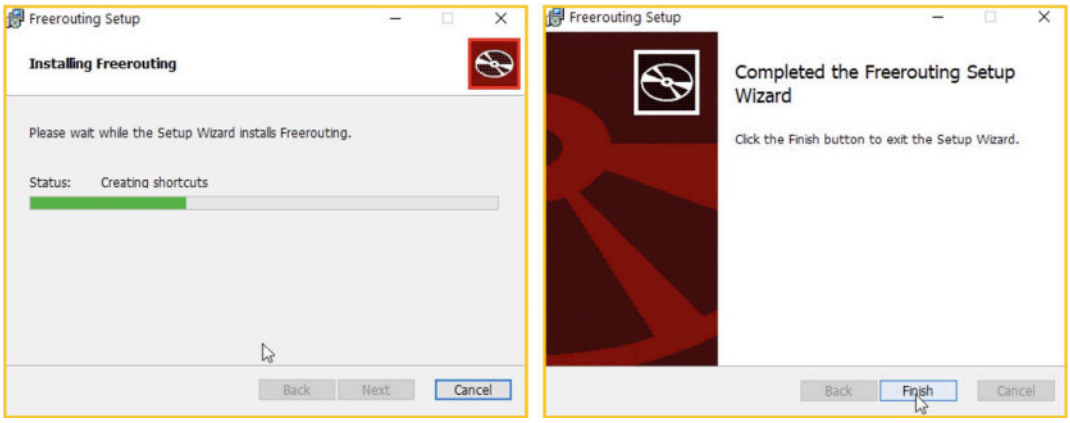


Figure 13.16.3.23: The installation process has two steps.

Click “Finish” (see above) to complete the installation. That is. Your instance of the Freerouting application is now installed. Start the application by clicking on the Windows search box (bottom left of the screen) and type “freerouting” (1). Windows will find the application. It may need a few seconds as the Windows search index is working in the background. Double-click on the application to start it (2) (Figure 13.16.3.24).

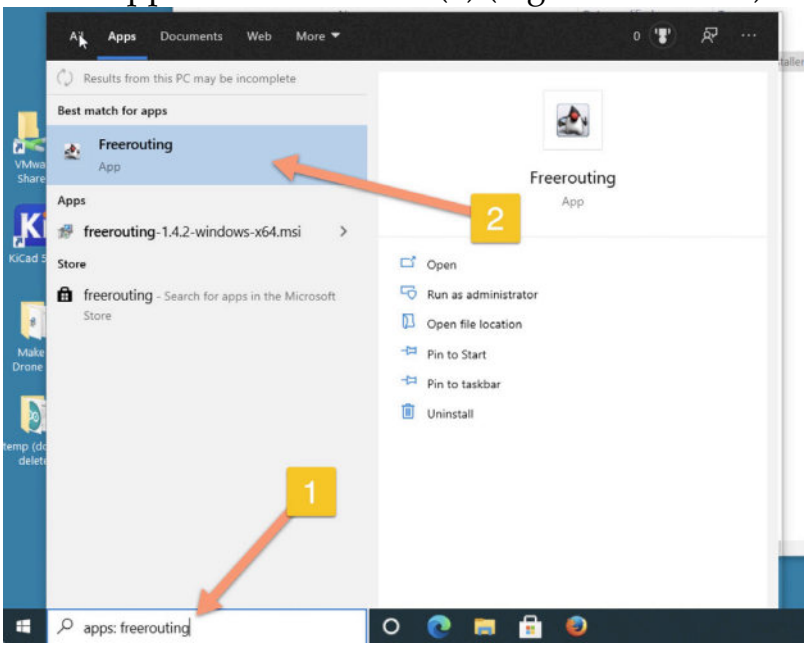


Figure 13.16.3.24: The Freerouting application is ready to use.

The application will start and wait for you to indicate the import file to open (Figure 13.16.3.25):

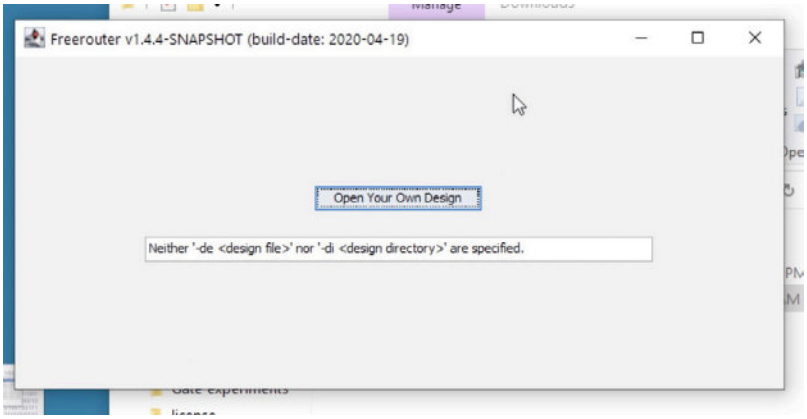


Figure 13.16.3.25: The Freerouting application has started.

At this point, you have installed Freerouting on Windows 10 and started the application. To learn how to use it alongside KiCad and perform auto-routing, continue with the 2-layer example in the next segment of this chapter.

16.4. How to use the Freerouting autorouter 2-layer example

You now have Freerouting installed on your computer. It is time to use it. In this segment of the chapter, you will learn how to route a 2-layer PCB automatically. For this demonstration, I will use the layout from one of the projects you completed earlier in this book.

Below is the example PCB that I routed manually (the MCU cataloguer).

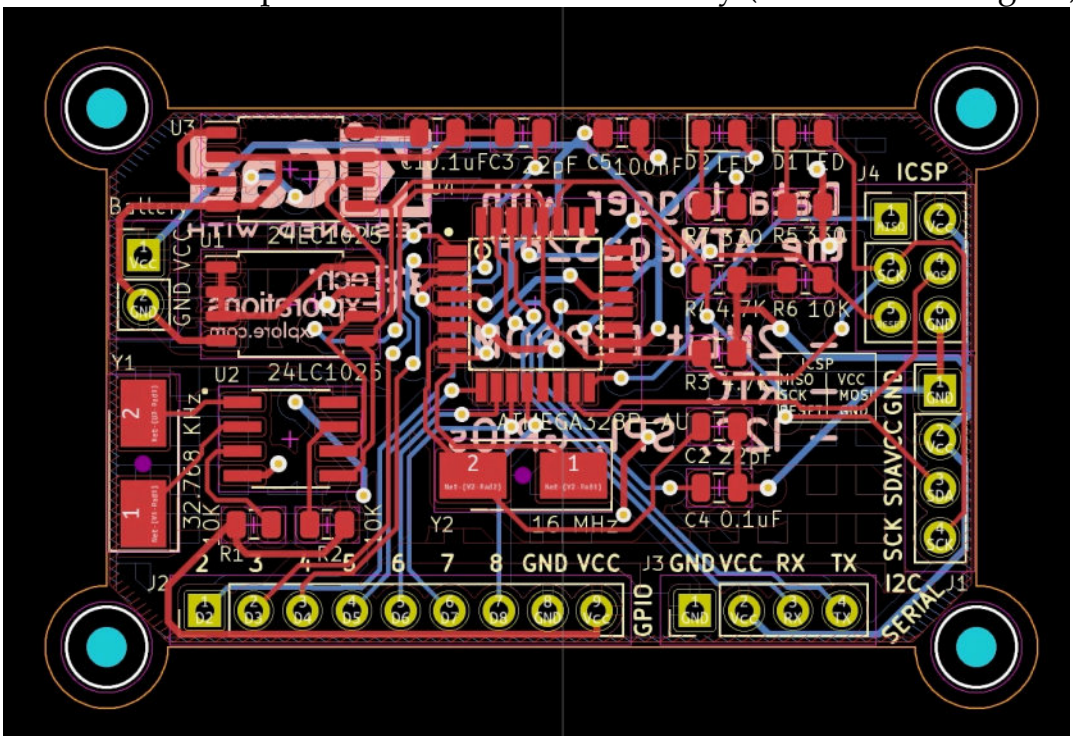


Figure 13.16.4.26: This PCB is routed manually in Pcbnew.

The PCB in the screenshot above is a 2-layer board, and you can confirm by looking at the number of copper layers in Board Setup —> Physical Stackup (Figure 13.16.4.27):

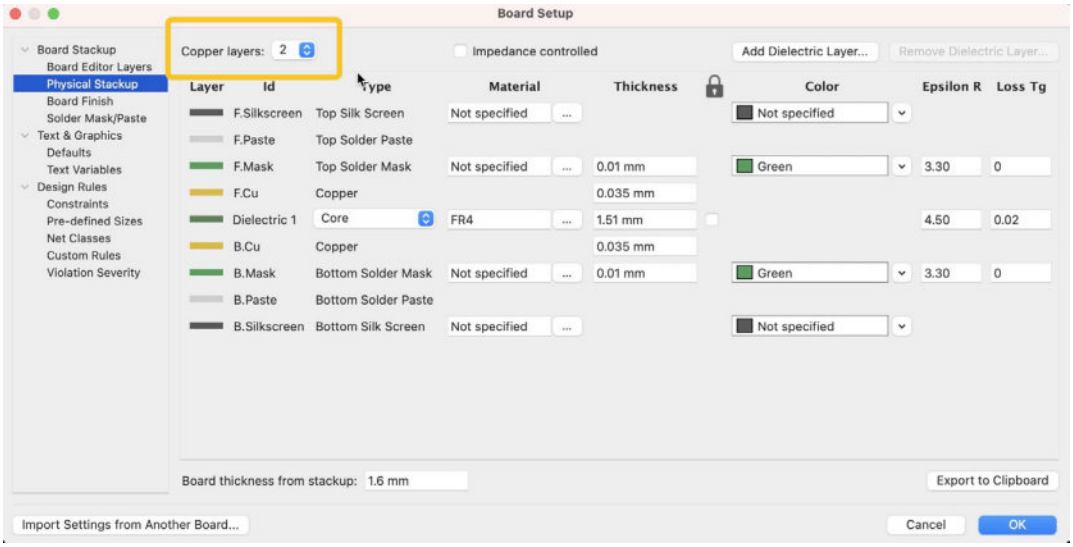


Figure 13.16.4.27: This is a 2-layer PCB.

I will remove all zones, traces and vias by using Pcbnew's Global Deletions tool (Edit —> Global Deletions):

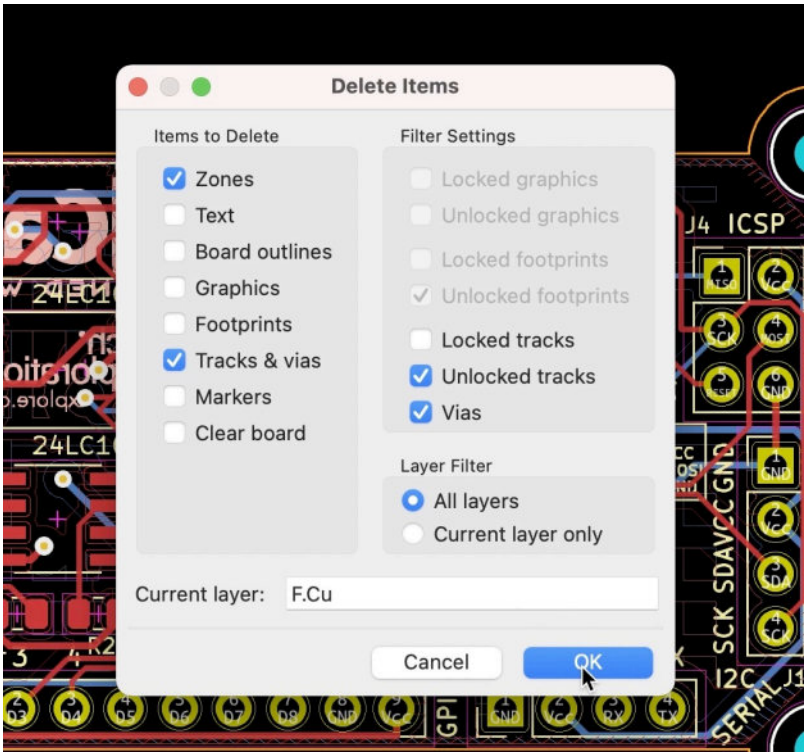


Figure 13.16.4.28: Delete all zones, tracks, and vias.

I now have a PCB without routes, ready to be automatically routed.

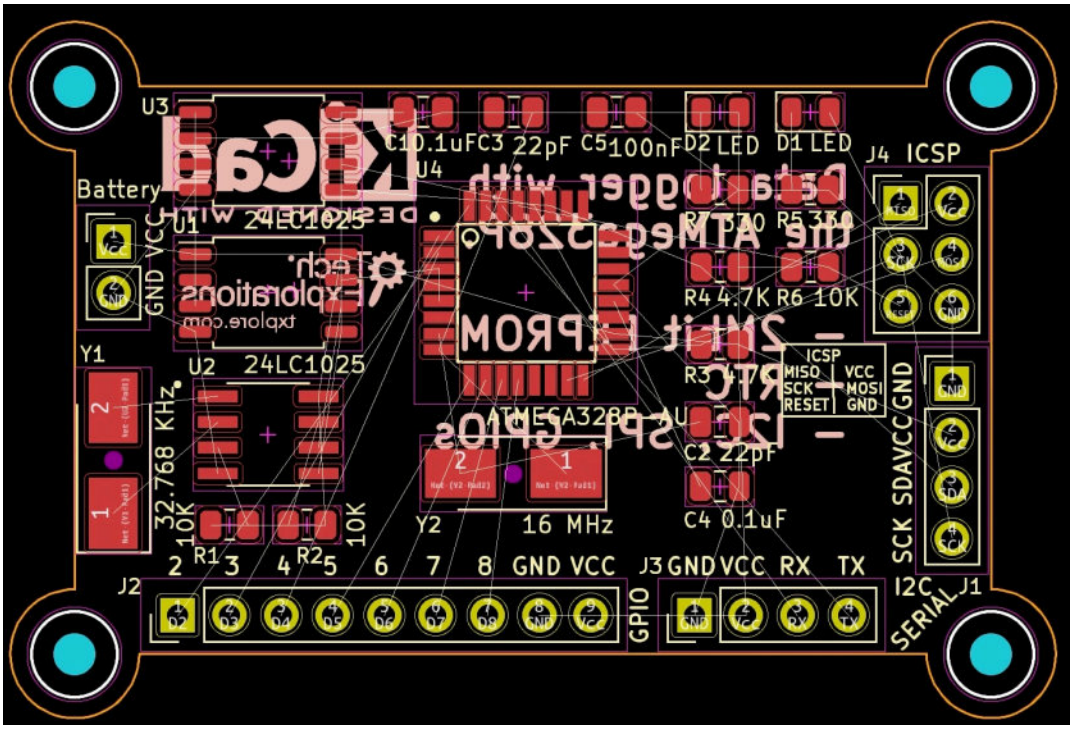


Figure 13.16.4.29: This PCB has no routes.

KiCad can work with Freerouting using a Specctra DSN file. Export the DSN file from the File menu: File —> Export —> Specctra DSN…:

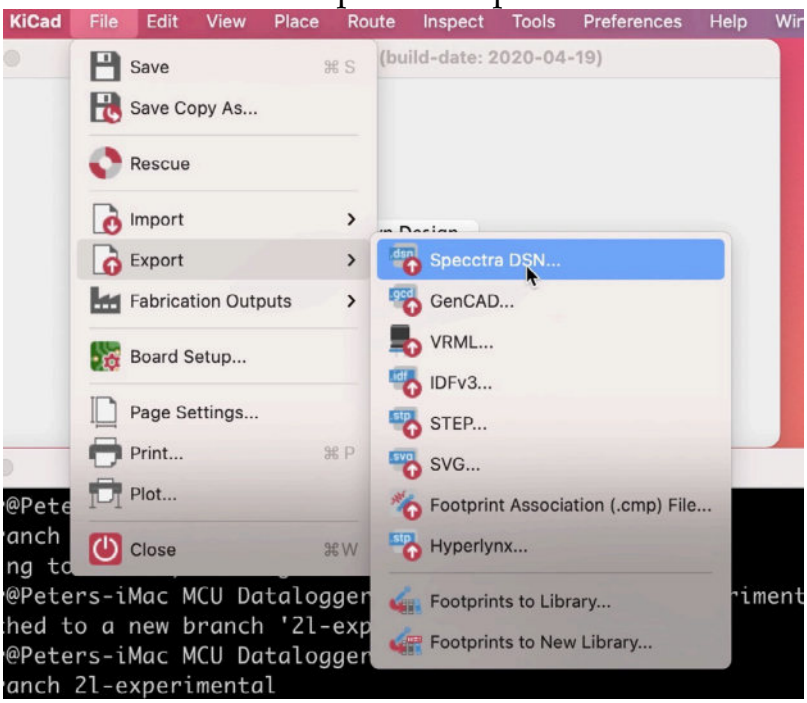


Figure 13.16.4.30: Export the DSN file.

KiCad will ask you for a location for the new file. I choose to save it in the project folder:

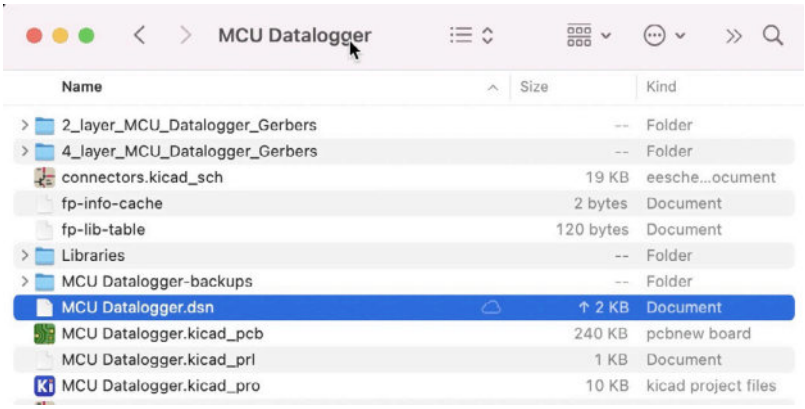


Figure 13.16.4.31: The new DSN file.

It is time to import the DSN file in Freerouting. Start Freerouting if not already running (see below), and click on the button "Open Your Own Design" (1). Navigate to the location where you saved the DSN file, select it (2), and click Open (3).

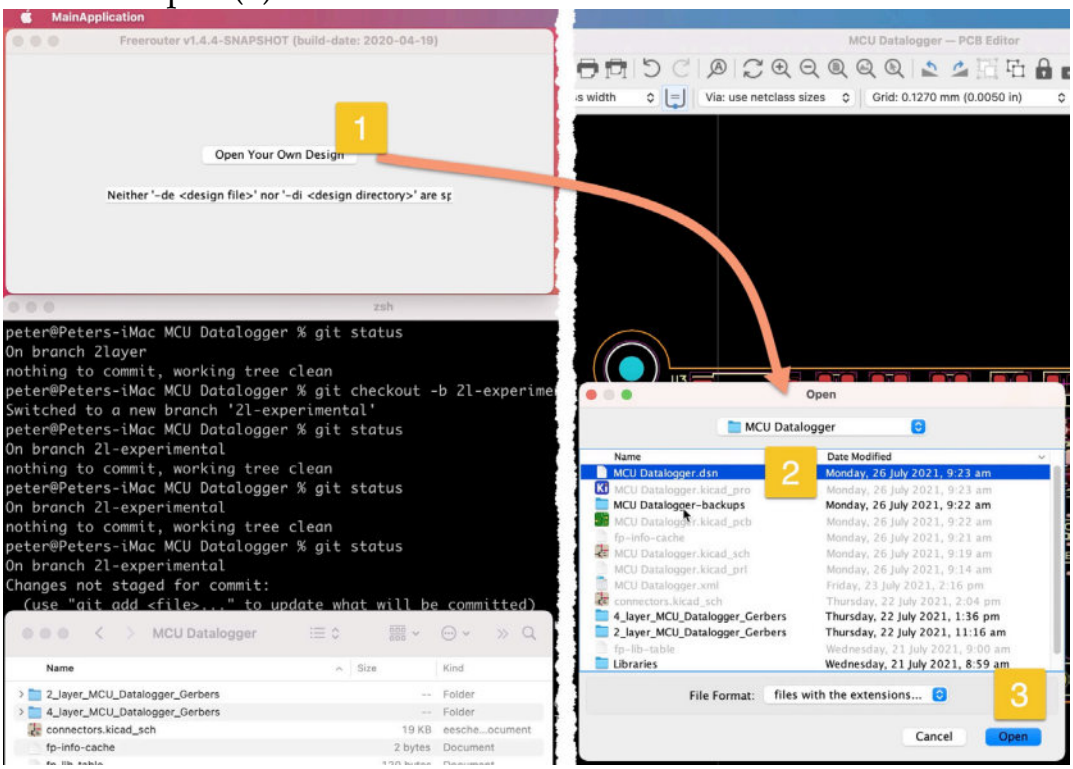


Figure 13.16.4.32: Import the DSN file to Freerouting.

Freerouter's Board Layout window will appear with the PCB and ratnest lines indicating the required tracks (Figure 13.16.4.33):

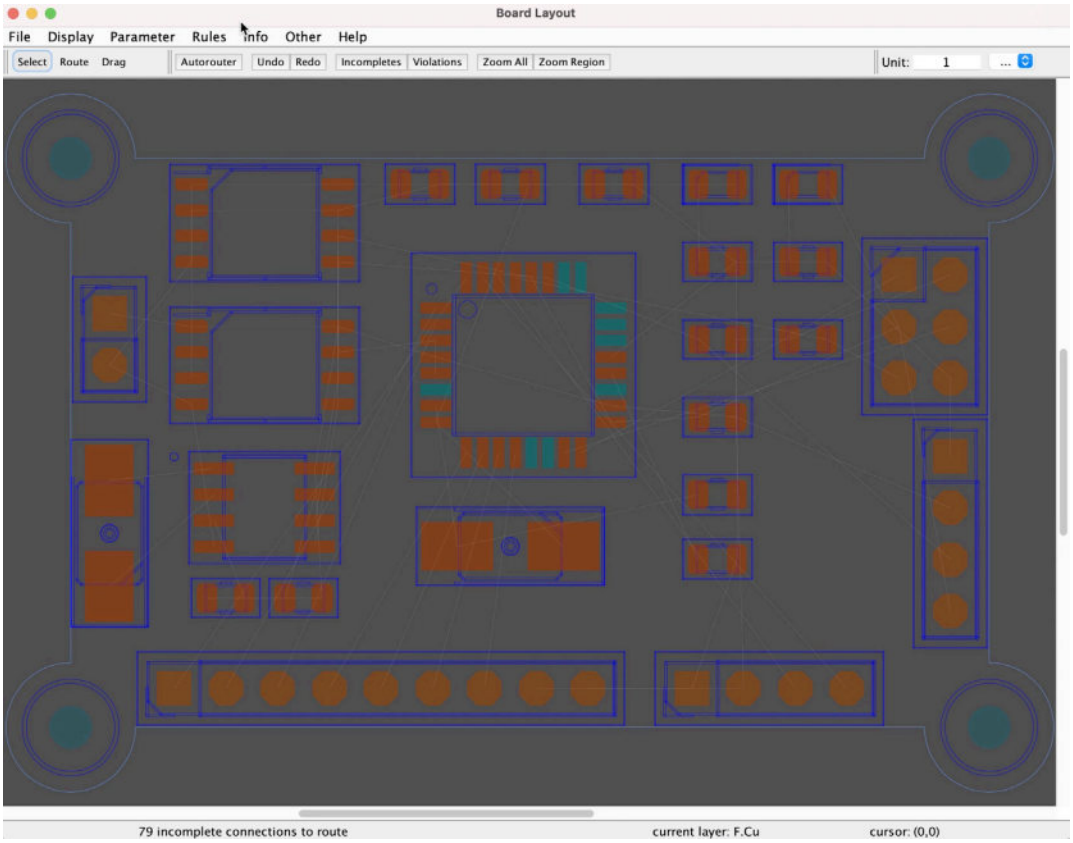


Figure 13.16.4.33: The unrolled PCB in Freerouting.

If this is the first time you are using Freerouter, take a few minutes to click around the various menus and become familiar with the interface. There are helpful options under the Help menu, including a full eBook (Help —> Help Contents). Notice the three buttons at the top-left corner of the window: Select, Route, Drag. You can experiment with manual routing by clicking on the Route button and then click on a pad to start drawing, move the mouse to draw the route, and click again to finish the drawing. You can move footprints or tracks by clicking on the Drag button. With the Select button, you can select a board item, such as a pad, and act on it using one of the available commands that appear on the menu bar.

To delete multiple tracks, click on the "Select" button and create a rectangle that contains the routes you want to delete with your mouse. Then, hit the Delete key on the keyboard to delete the selected tracks. You can delete an individual trace by clicking to select it and then hit the Delete key.
To start the autorouter with its default settings, click on the Autorouter button (Figure 13.16.4.34).

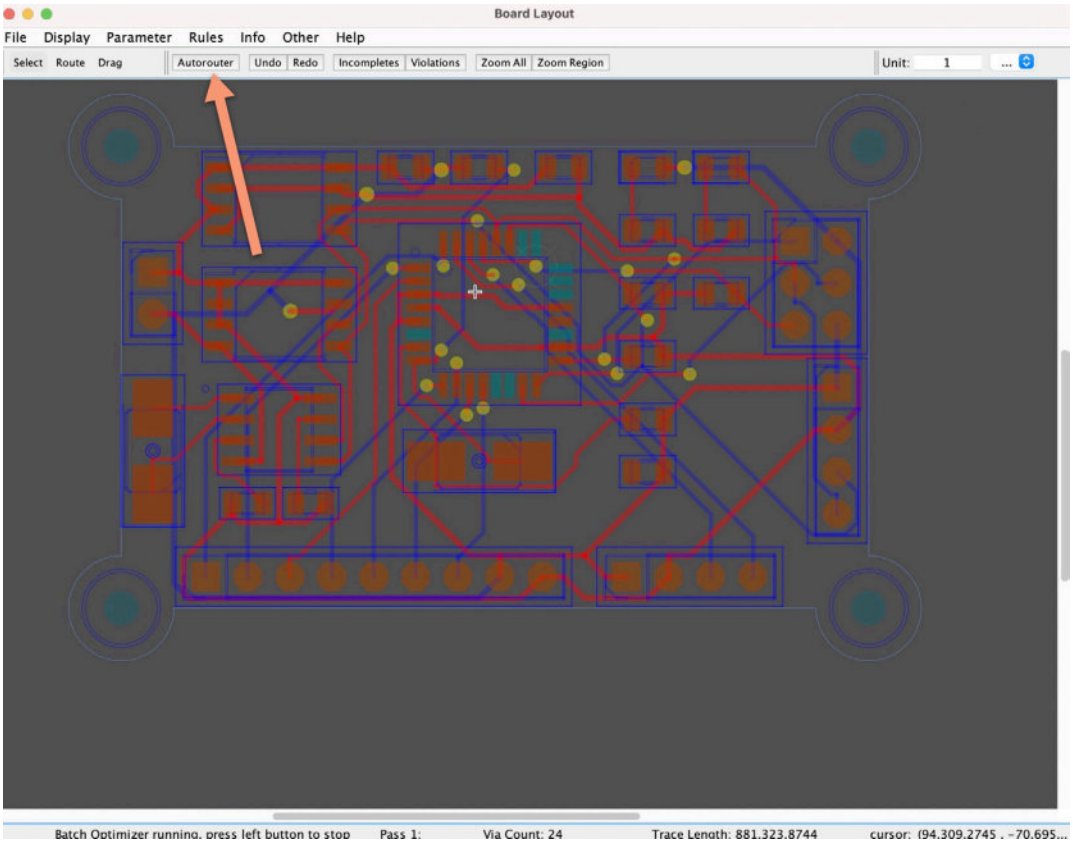


Figure 13.16.4.34: Start the autorouter by clicking on the Autorouter button.

The autorouter will get to work immediately and report its progress in the status bar at the bottom of the window. To stop the autorouter, click anywhere inside the window.

To customize the auto-routing algorithm, use the various options inside the Parameter and Rules menus. For example, click on Parameter —> Autoroute to bring up the Autoroute Parameter dialog (Figure 13.16.4.35):

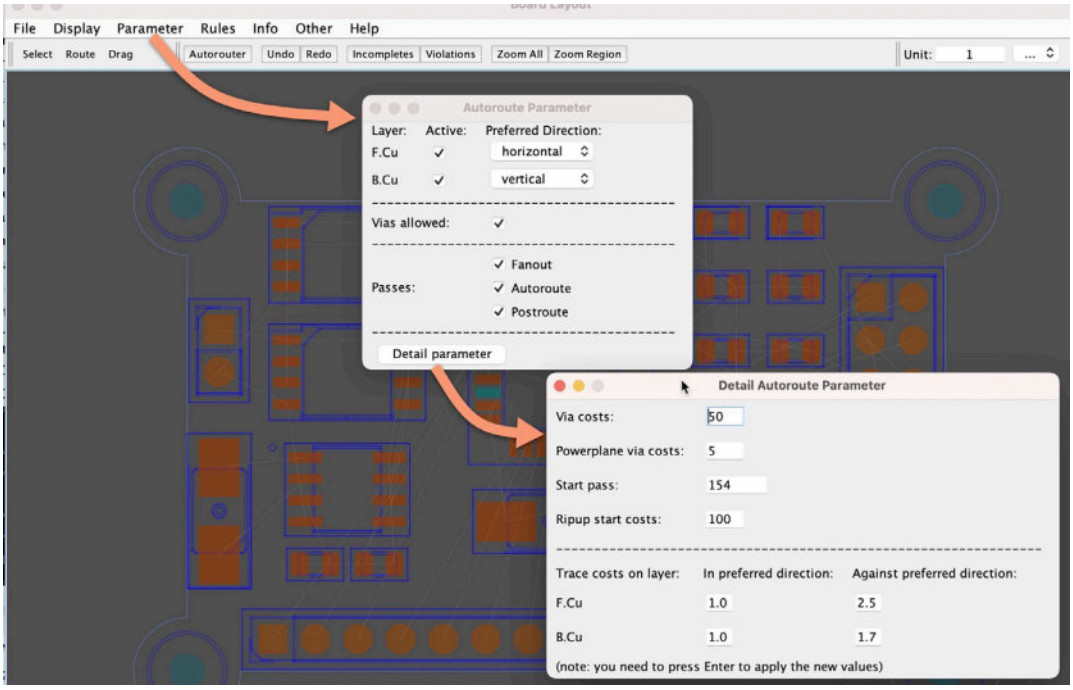


Figure 13.16.4.35: The Autorouter Parameter dialog box.

The Autoroute Parameter dialog allows you to choose the preferred direction of drawing for the enabled layers and configure the pass mode. Click on the Detail Parameter button to reveal a secondary dialog box with further options.

Another example of a way to fine-tune the autorouter algorithm with the Net Classes dialog box (Rule —> Net Classes):

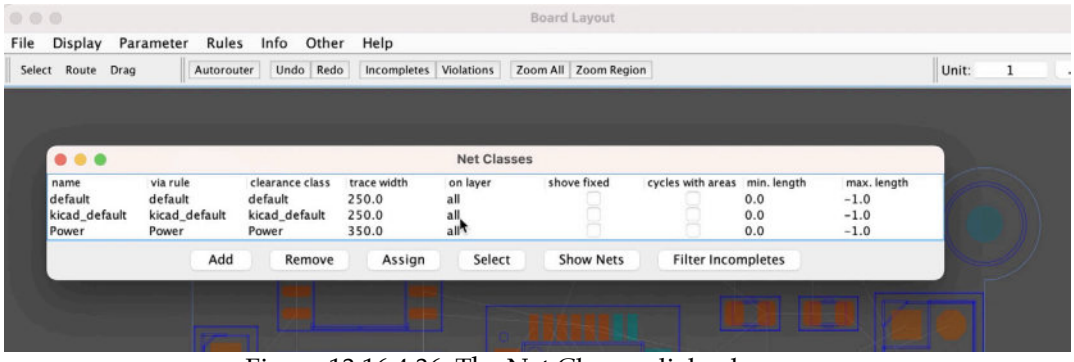


Figure 13.16.4.36: The Net Classes dialog box.

In the Net Classes dialog box, you can see the net classes as I set them in the KiCad project, imported into Freerouter. You can configure individual properties for each, such as whether tracks that belong to a net class are allowed to shove another track.

It is worth taking a bit of time to experiment with the various options and menus.

Let's continue with the auto-routing and, this time, take it to completion. Click on the Autorouter button, and allow enough time for the routing to complete.

While the autorouter is working, you will see new routes appearing and others disappearing and replaced with optimized ones. Also, notice the progress figures in the status bar. In the screenshot below, the autorouter is in its second pass (1), with 73 incomplete routes (2) and 31 complete routes (3):

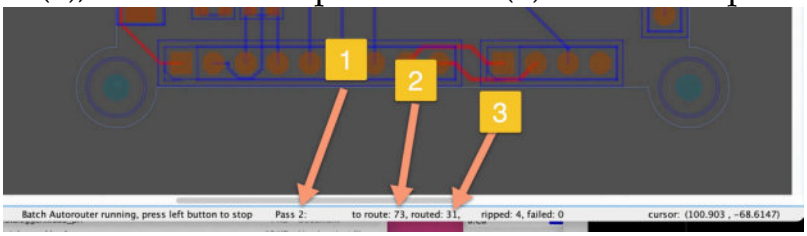


Figure 13.16.4.37: Autorouter progress showing incomplete routes.

A few seconds later, I took another screenshot showing that all routes were complete after 13 passes (1), using 23 vias (2) and with a total trace length of 888.17 mm):

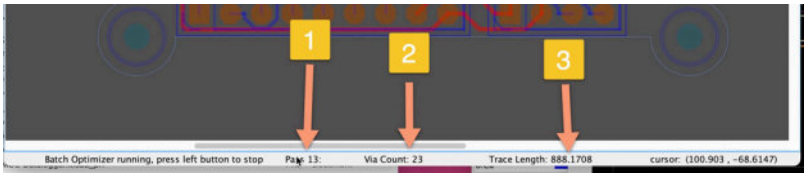


Figure 13.16.4.38: Autorouter progress showing NO incomplete routes.

At this point, the autorouter will continue its work, optimizing the routing by reducing the total trace length and number of vias. You can stop it at any time by clicking inside the router window.

With the PCB now fully routed, you can export the Specctra Session File so that you can continue the process in KiCad. Go to File, then click on Export Specctra Session File:

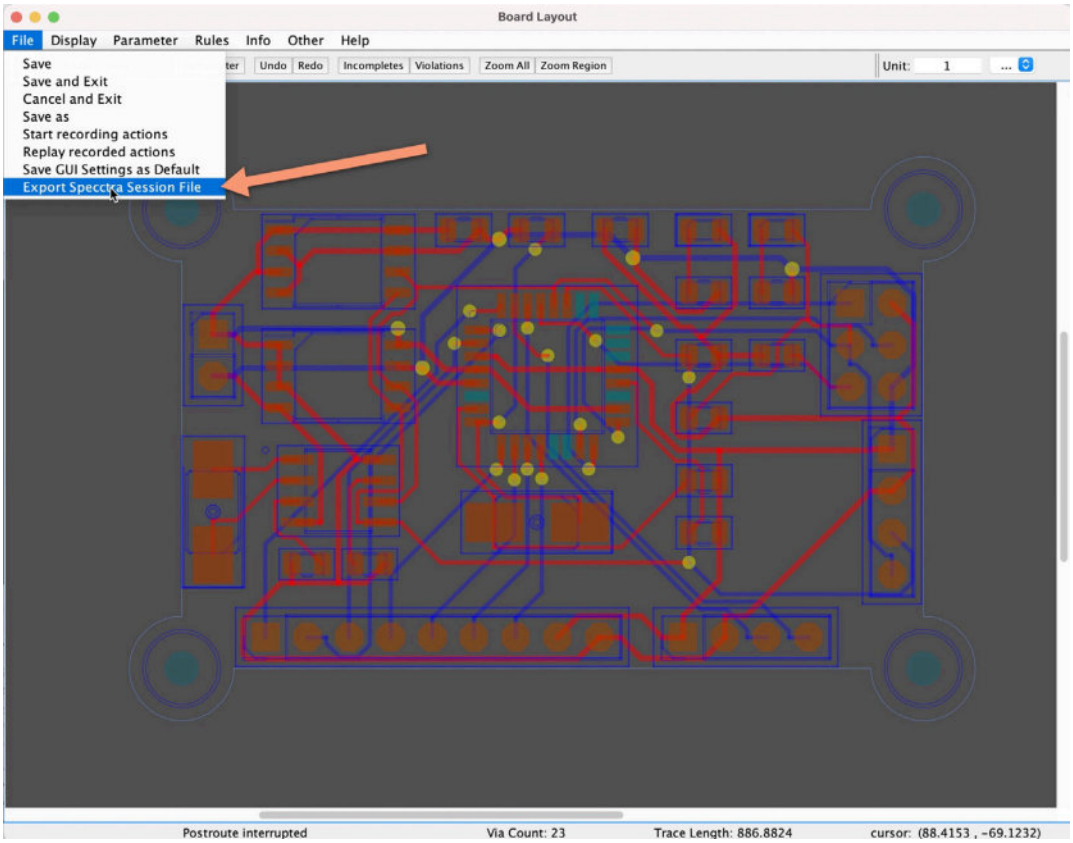


Figure 13.16.4.39: Export Specctra Session File.

The new file will have the ".ses" extension and is automatically saved in the location of the DSN file:

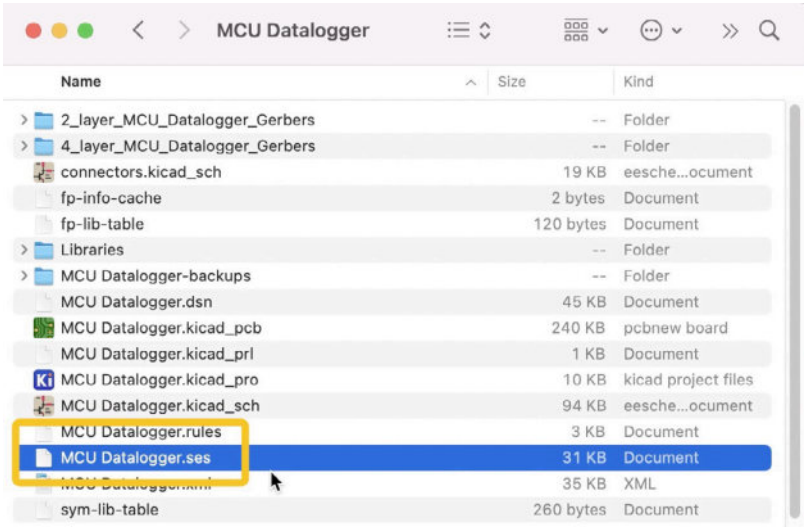


Figure 13.16.4.40: The ".ses" file in my project directory.

Time to import the ".ses" file into Pcbnew. Go to Pcbnew and click on File —> Import —> Specctra Session… :

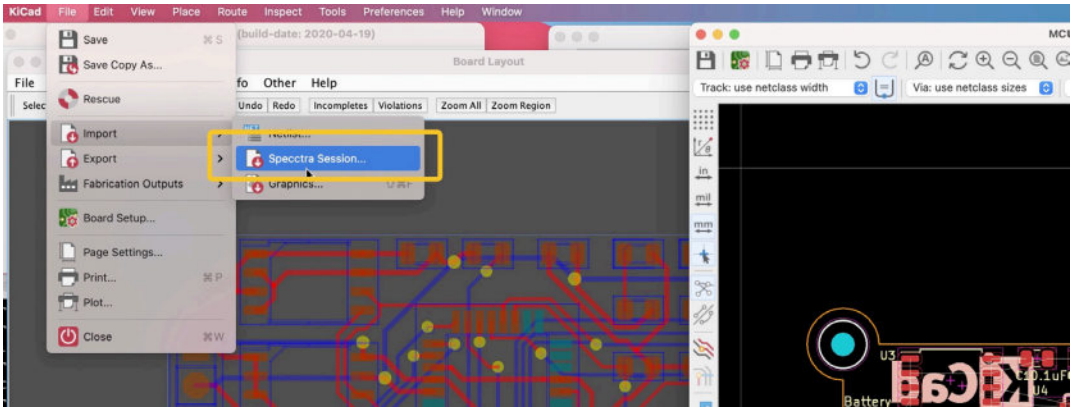


Figure 13.16.4.41: Import the Specctra Session file (SES) generated by Freerouting.

The imported will ask you for the location of the file. Find the file, select it, and click “Open” (Figure 13.16.4.42).

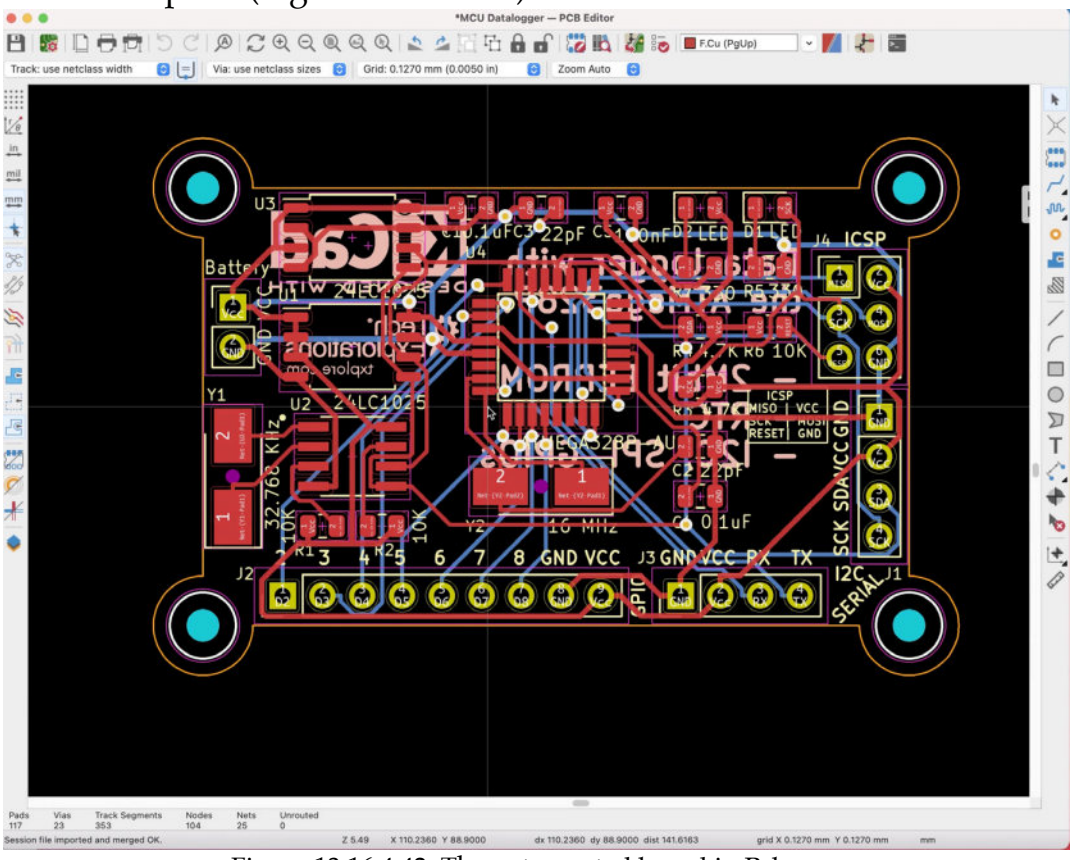


Figure 13.16.4.42: The auto-routed board in Pcbnew.

Pcbnew will replace the rattiest lines with the tracks that Freerouting drew. Compare the PCB in Figure 13.16.4.42 against that of Figure 13.16.4.39 to confirm that the routes are identical.

In summary, it only took the autorouter a few seconds to fully route this 2-layer board. It had taken me over 60 minutes to do the same.

In the next segment of this chapter, I will show you how to route the same PCB automatically but this time using four layers instead of two.

16.5. How to use the Freerouting autorouter 4-layer example

In the previous segment of this chapter, you learned how to use the Freerouting application to route a 2-layer PCB automatically. In this segment, you will automatically redo the routing, but with four layers.

To do this demonstration, I will start in Pcbnew and configure the board to use four layers. In Pcbnew, open the Board Setup window, and go to Physical Stackup:

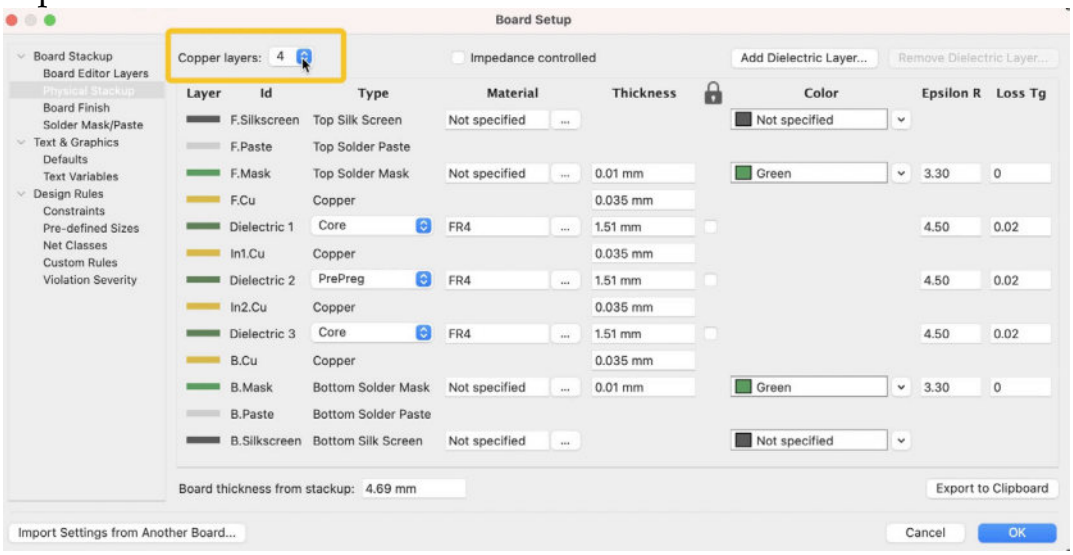


Figure 13.16.5.43: This is a four-layer PCB.

Also, click on the Board Editor Layer tab to confirm the role of each of the copper layers:

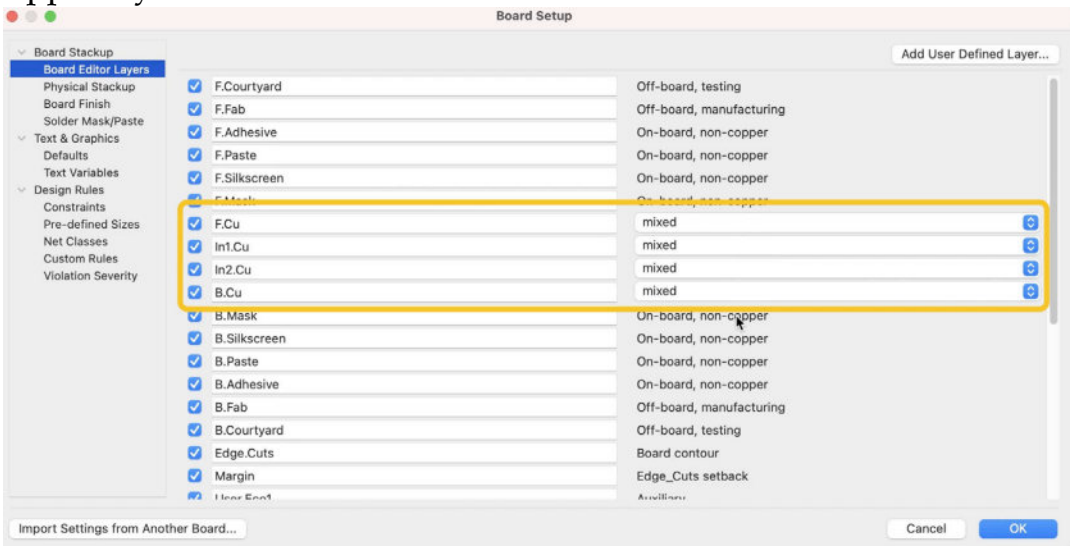


Figure 13.16.5.44: I have configured the four layers of this PCB as “mixed.”

I have set all four layers of this PCB as “mixed” to give the autorouter maximum freedom in routing tracks that belong to signal or power classes.

Click “OK” and continue to export the DSN file with File —> Export —> Specctra DSN:

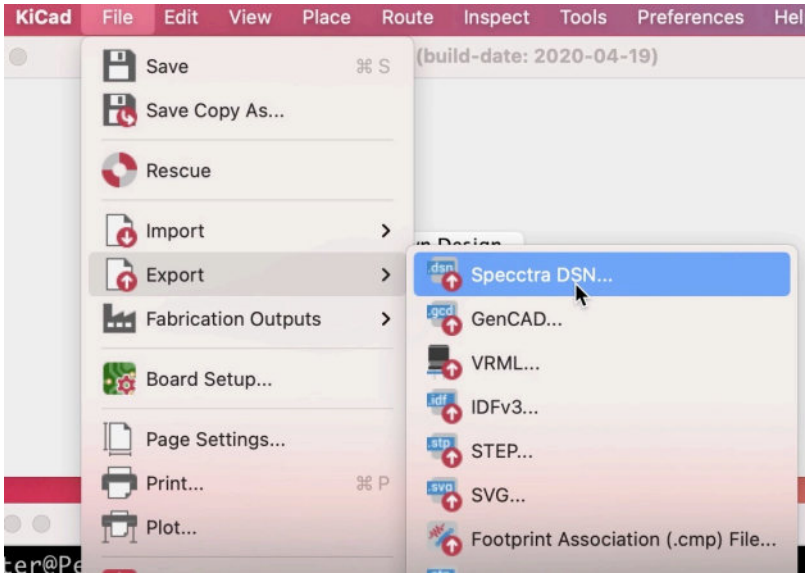


Figure 13.16.5.45: Exporting the DSN file for the four-layer board.

As you learned in the previous segment of this chapter, start the Freerouter application and import the DSN file. The four-layer un-routed board should look like this:

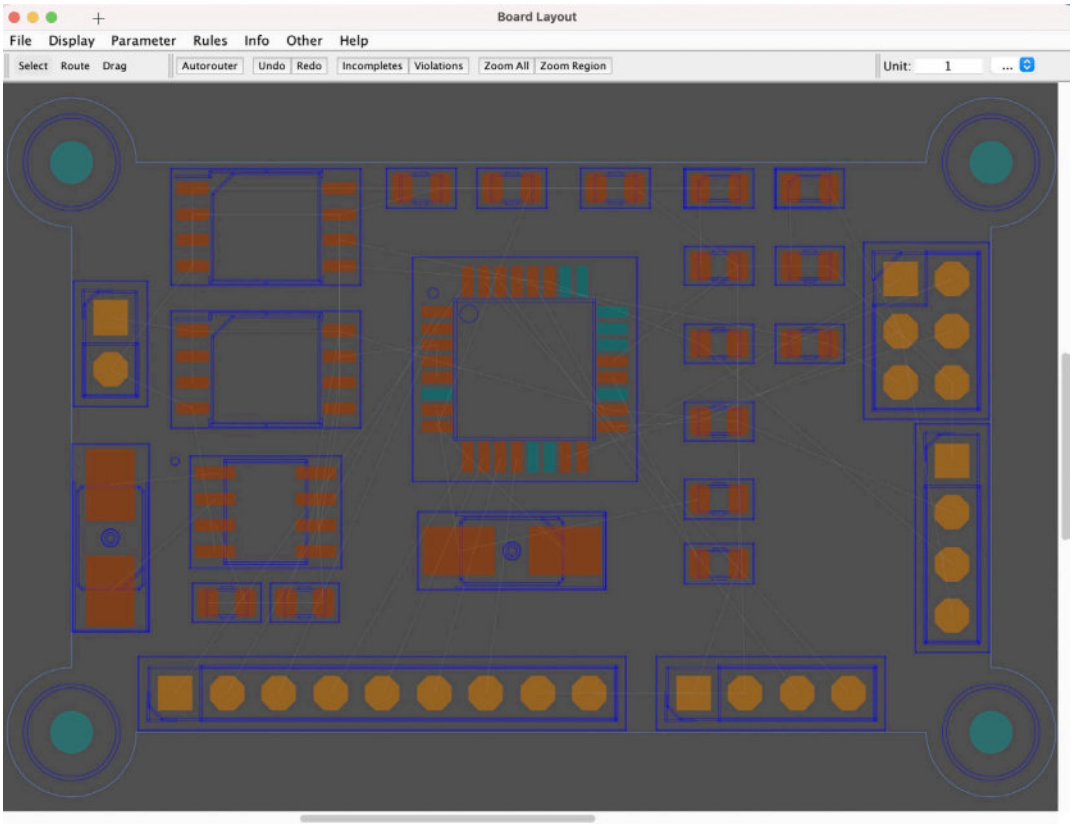


Figure 13.16.5.46: The un-routed four-layer PCB in Freerouting.

In Freerouter, click on the Display menu and the “Layer Visibility.” This will reveal a small window that contains four widgets, one for each layer. You can use those widgets to change the visibility of each layer as the autorouter is

working. You can keep this window active on the side of the Board Layout window to be available to use at any time.

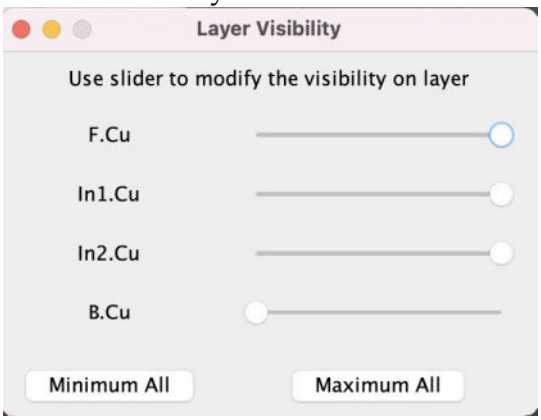


Figure 13.16.5.47: The layer visibility window.

Before starting the autorouter, take a few moments to look at the Autorouter Parameter window (where you can select the preferred track direction for each layer) and the Clearance Matrix window (under Rule), where you can set minimum clearances for each class and layer.

Start the autorouter by clicking the Autorouter button. A few seconds later, the board is fully routed and looks like this:

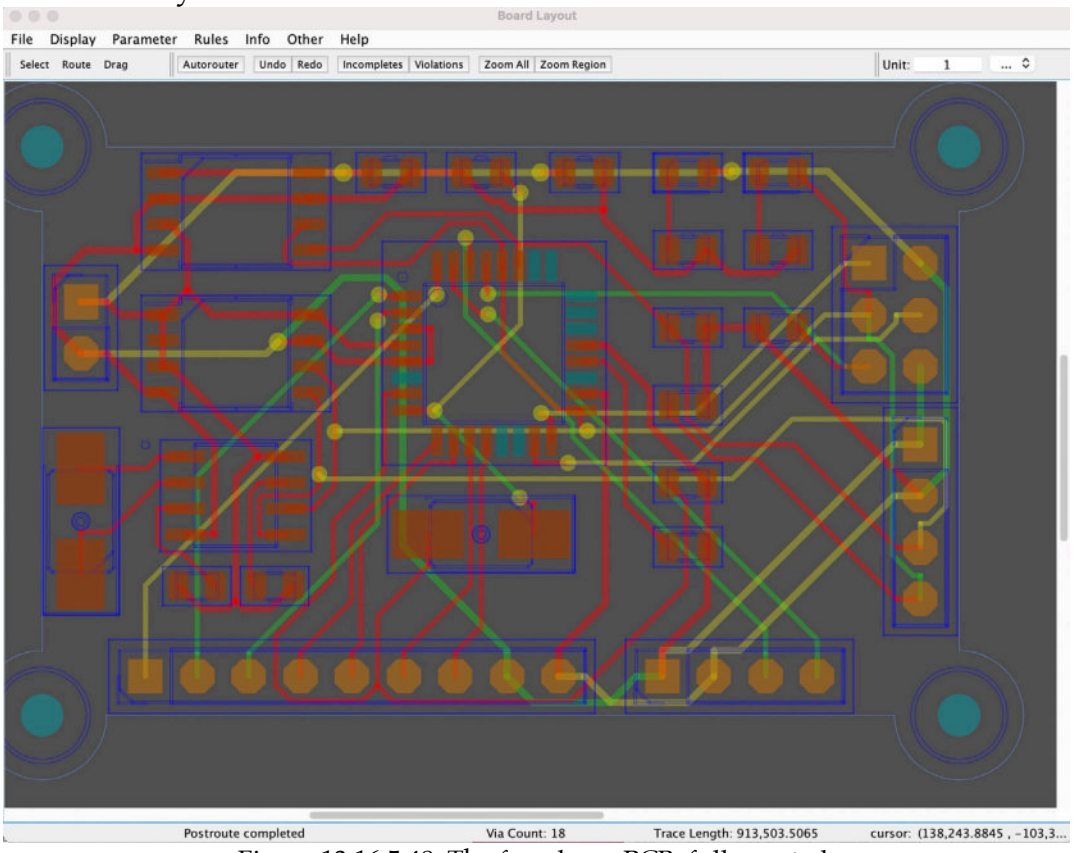


Figure 13.16.5.48: The four-layer PCB, fully routed.

The PCB is now fully routed and ready to export. Export the Specctra Session file, and import it to Pcbnew. In Pcbnew, the board looks like this:

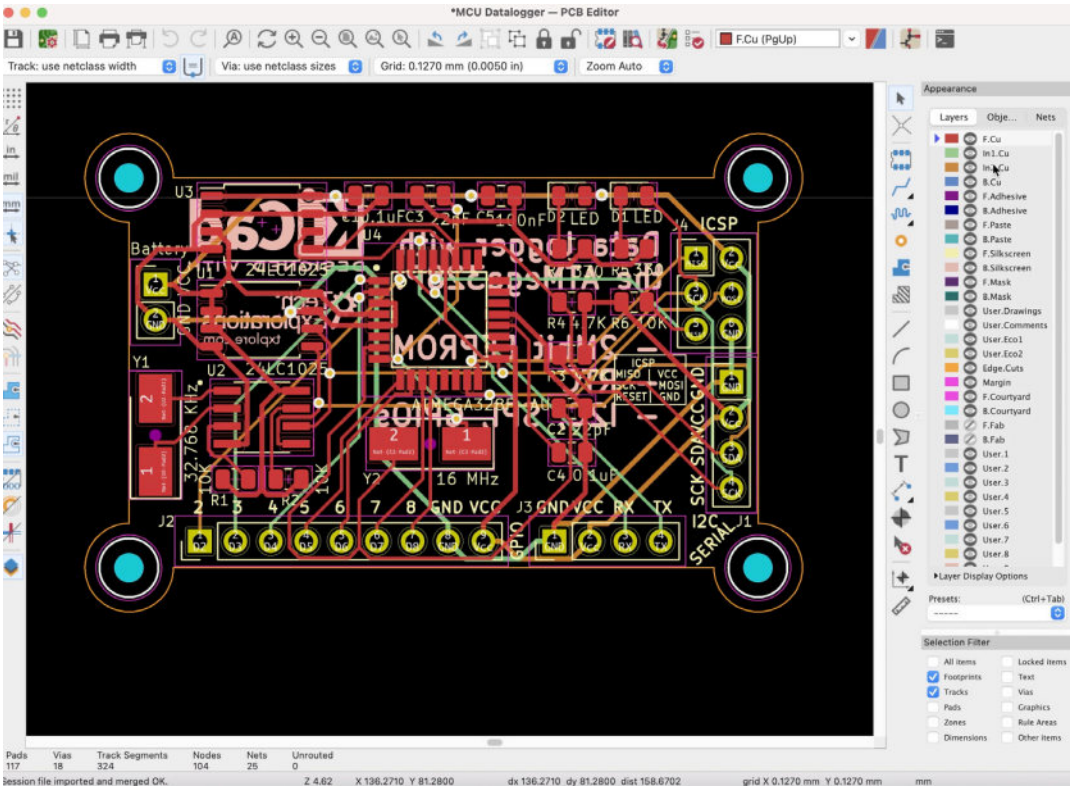


Figure 13.16.5.49: The four-layer PCB fully routed in Pcbnew.

You can compare the routes in Figure 13.16.5.48 and Figure 13.16.5.49 to confirm that they are the same. You should also perform a Design Rules Check to ensure that there are no violations. In my case, the DRC revealed no unconnected items but three violations.

Here is one example:

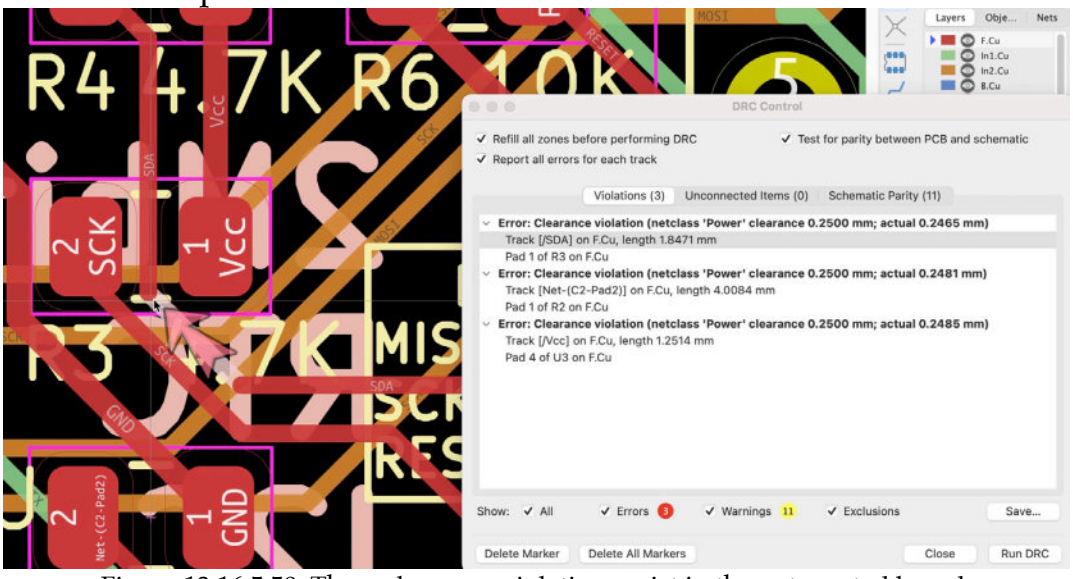


Figure 13.16.5.50: Three clearance violations exist in the autorouted board.

Violations such as the one in Figure 13.16.5.50 are simple to fix with manual editing.

In summary, auto-routing is an important tool for any PCB designer and should be used with an appreciation of its limits. Autoroutes, in general, are

blunt tools that can save you a lot of time, as long as you understand their limitations. For any specialized tracks, you will still need to rely upon manual drawing.

17. Pcbnew Inspection menu

In this chapter, you will learn about the functionalities available under the Pcbnew “Inspect” menu (Figure 13.17.1).

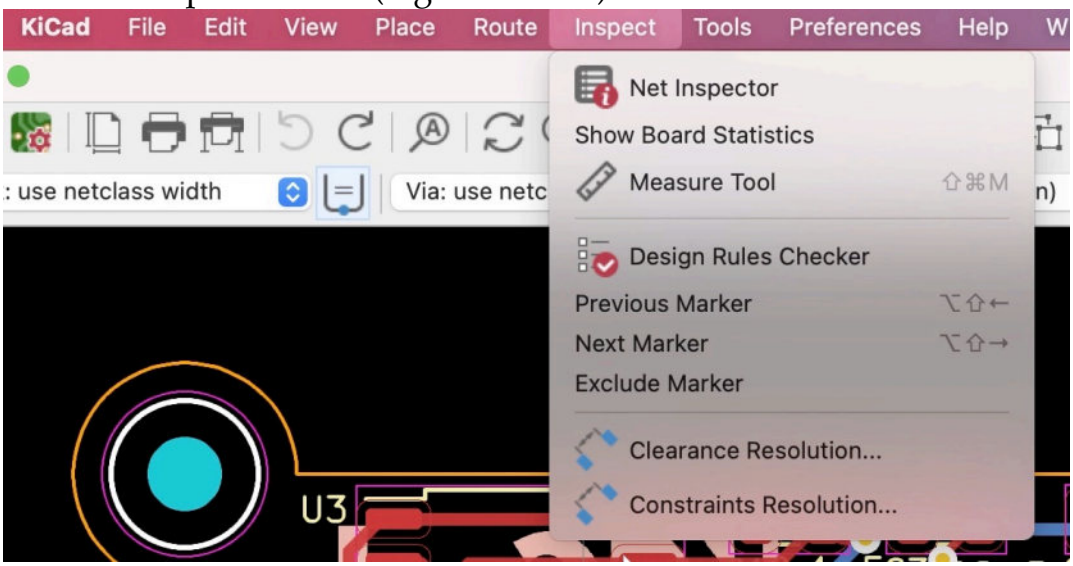


Figure 13.17.1: The “Inspect” menu in Pcbnew.

We’ll begin with the Net Inspector.

Net Inspector

Start the Net Inspector by clicking on Inspect —> Net Inspector. The Net Inspector window will appear, and it looks like this (Figure 13.17.2):

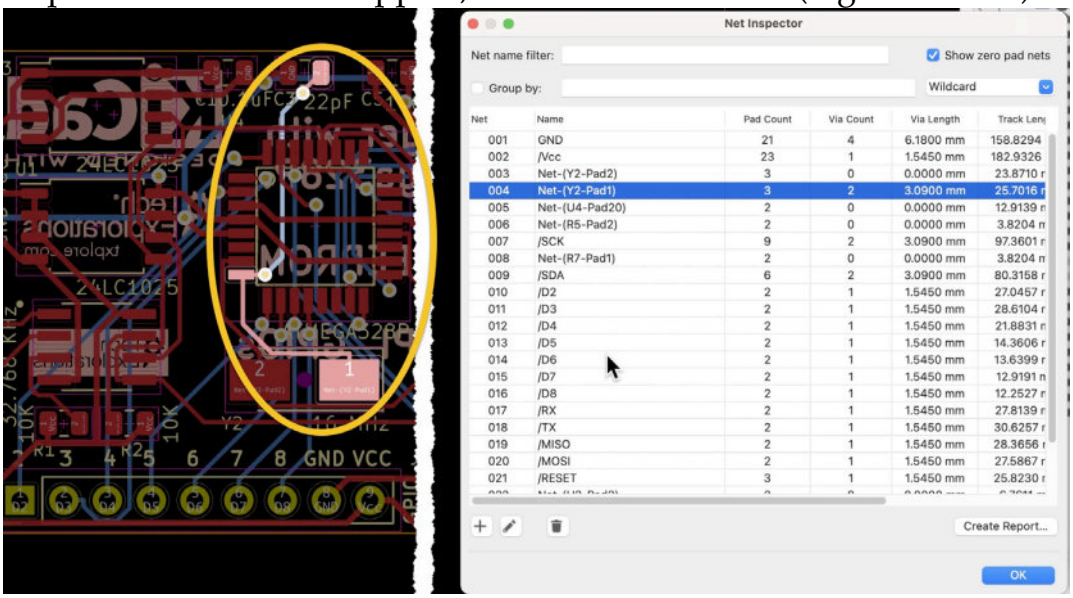


Figure 13.17.2: The Net Inspector window with a net selected and its members highlighted in the editor.

The Net Inspector window contains a list of the nets in the layout and statistics such as the total number of pads, vias, and length of each net. You can click on a net row, and the editor will highlight the net members in the layout. You can use the widgets at the top of the window to filter and group the nets that the inspector shows in the list. For example, type "/D" in the name filter text box, and the inspector will only show nets with names that start with "/D."

At the bottom right corner of the Inspector window is the "Crete Report…" button. Click on this button and save the resulting CSV file on your computer. Below you can see an example of this report (Figure 13.17.3):

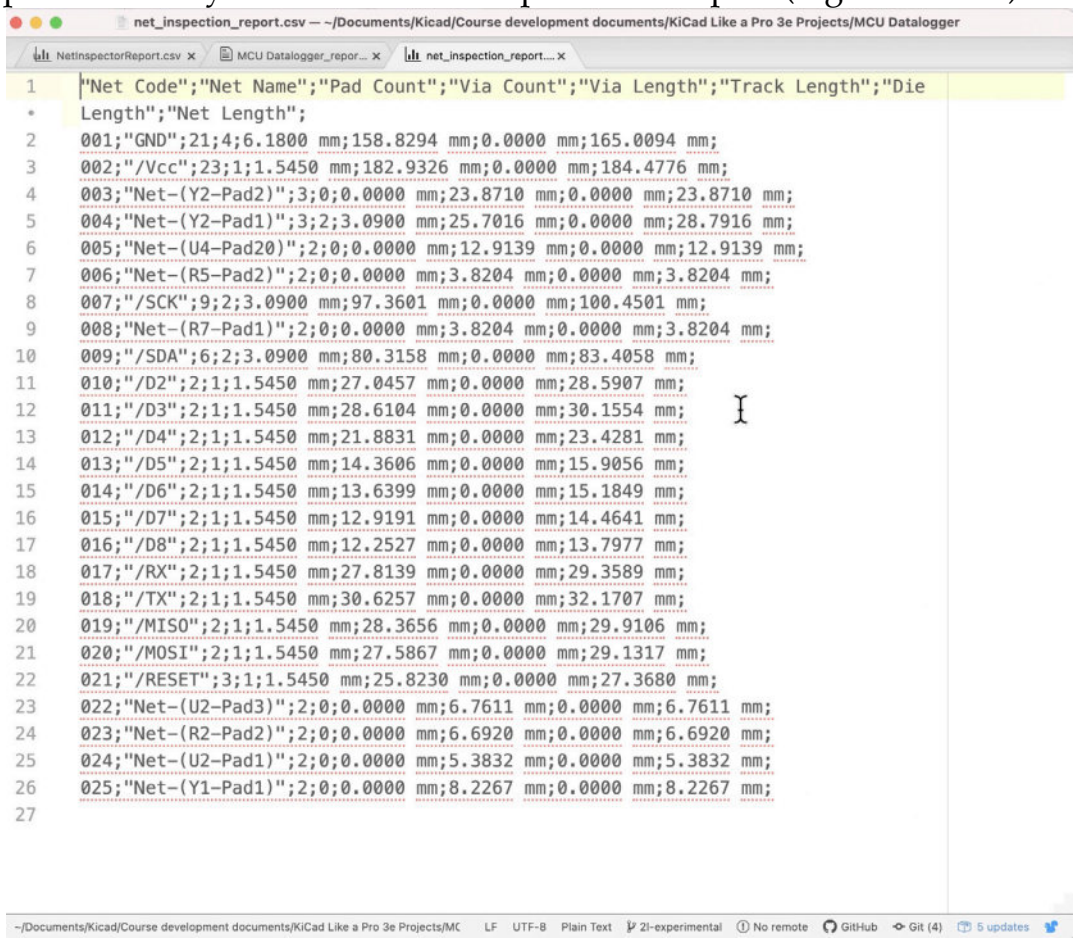


Figure 13.17.3: The Net Inspector exported report.

The report contains the same information that you see in the inspector window in a text CSV format that you can use with spreadsheets for further processing.

Board Statistics

Start the Board Statistics by clicking on Inspect —> Show Board Statistics. The Board Statistics window will appear, and it looks like this (Figure 13.17.4):

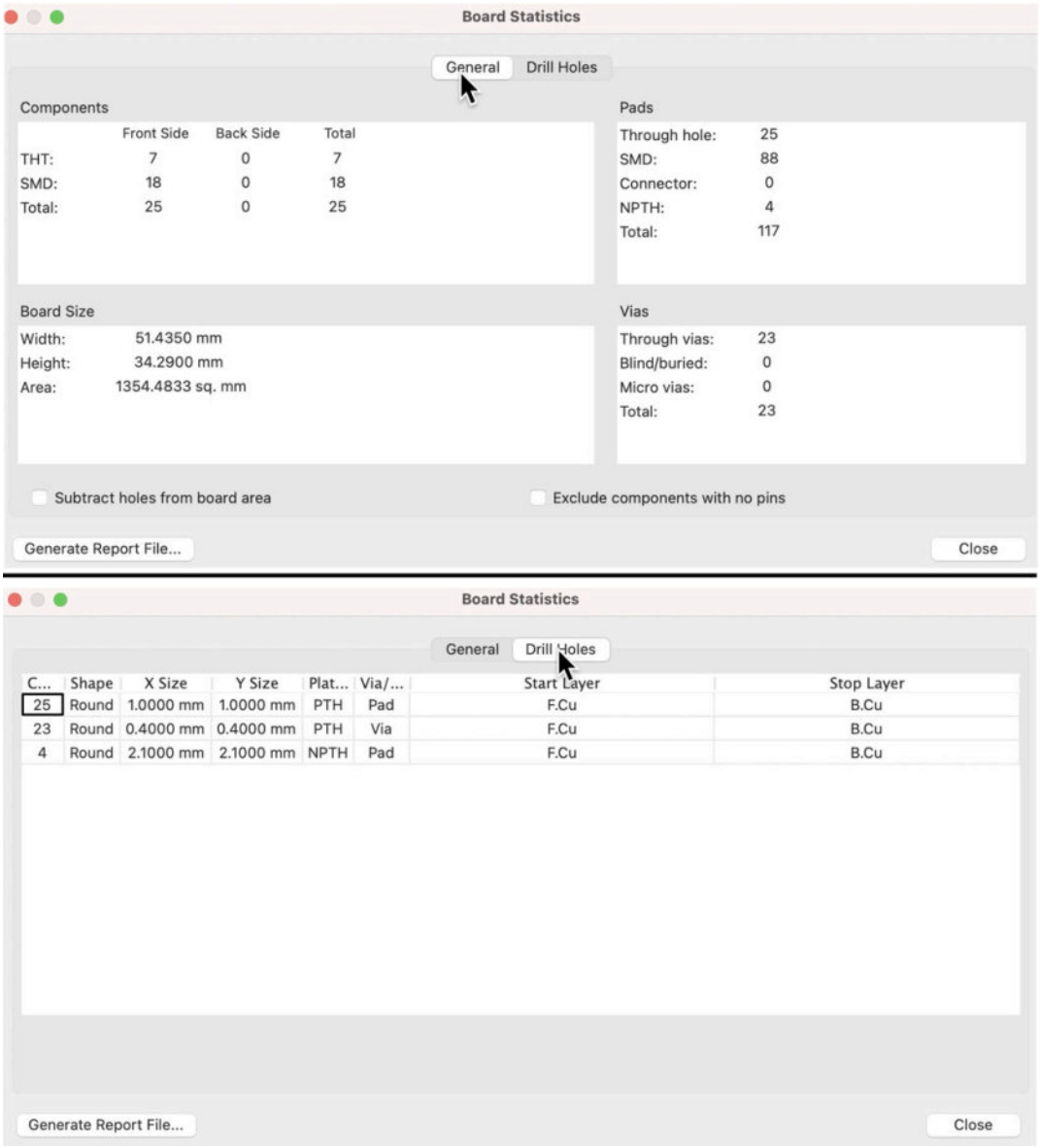


Figure 13.17.4: Board Statistics. General (top), Drill Holes (bottom).

The Board Statistics window contains two tabs. The General tab provides information about the types of components used in the board, pads, vias, and the board dimensions. The Drill Holes tab offers information about the drills, how many are present in the PCB, their dimensions, and start and stop layers.

You can generate a text report that contains all of this information, which looks like this (showing a segment only):

board_stats_report.txt — ~/Documents/Kicad/Course development documents/KiCad Like a Pro 3e Projects/MCU Datalogger

NetInspectorReport.csv | MCU Datalogger_repor... | net_inspection_report.... | board_stats_report.txt

```
PCB statistics report
=====================
- Date: Wednesday, 28 July 2021 at 08:43:23 am
- Project: MCU Datalogger
- Board name: MCU Datalogger

Board
-----
- Width: 51.4350 mm
- Height: 34.2900 mm
- Area: 1354.4833 sq. mm

Pads
----
- Through hole: 25
- SMD: 88
- Connector: 0
- NPTH: 4

Vias
----
- Through vias: 23
- Blind/buried: 0
- Micro vias: 0

Components
----------

|        | Front Side | Back Side | Total |
|--------|------------|-----------|-------|
|THT:    |          7 |         0 |     7 |
```

Figure 13.17.5: The board statistics text report.

Measure tool

You can invoke the Measure tool from the Inspect menu (“Measure tool”), or from the right toolbar:

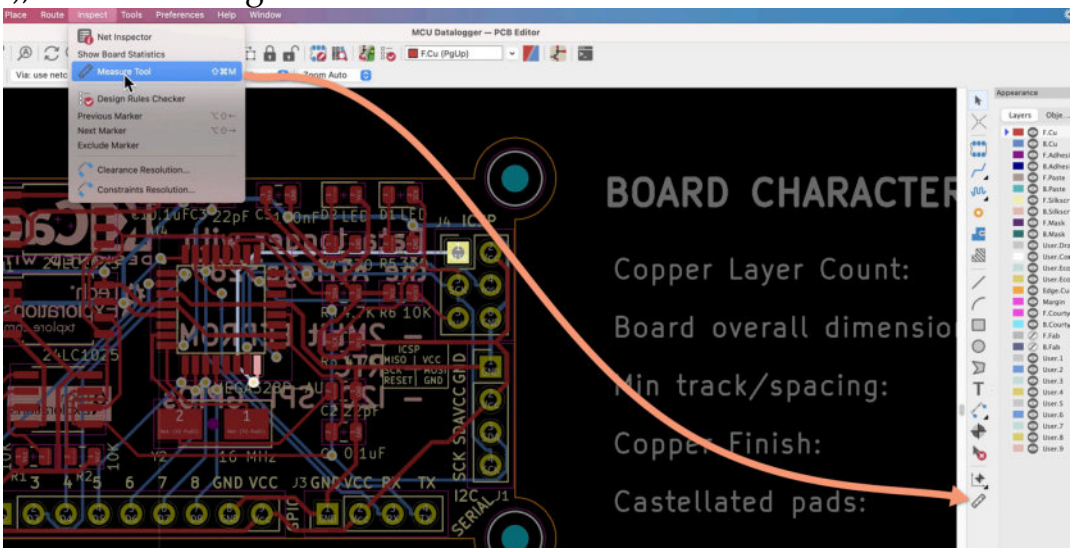


Figure 13.17.6: The Measure tool.

You can use the Measure Tool to make quick distance measurements between any two points in your PCB. Enable the tool, and then click on the layout editor to start measuring, then click again to finish:

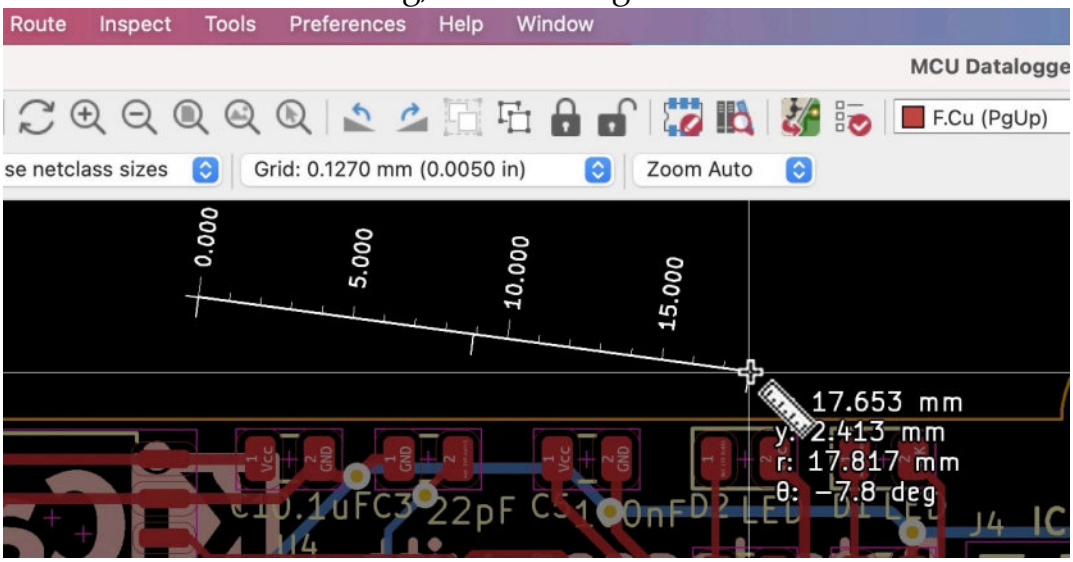


Figure 13.17.7: The Measure tool in action.

Design Rules Checker & Markers

I discuss the Design Rules Checker in detail in dedicated chapters (see here and here).

The Inspect menu include three related options:

- Previous Marker.
- Next Marker.
- Exclude Marker.

The options allow you to navigate through the results of the DRC by clicking on the menu items, or typing the set hotkey. For example, suppose that the DRC contains these results:

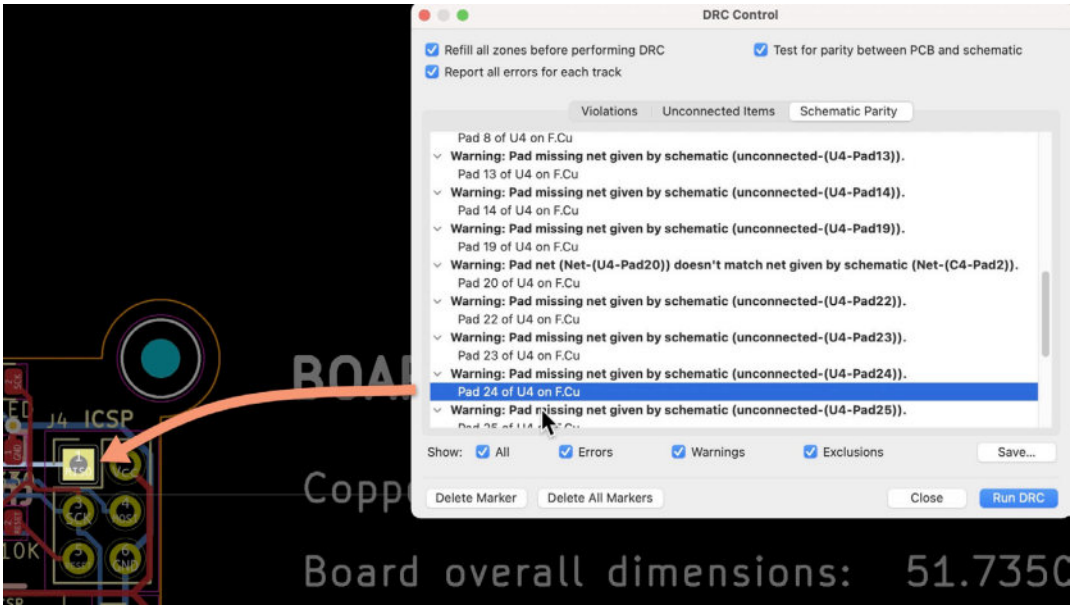


Figure 13.17.7: The DRC results. Click on a row to highlight the source.

You can navigate through the DRC rows by

1. Clicking on a row.
2. Clicking on the marker options inside the Inspect menu.
3. Typing the corresponding hotkeys (for my setup, Option-Shift-Left for Previous Marker, and Option-Shift-Right for Next Marker).
   You can assign custom hotkeys in the Preferences window, Hotkeys.

Clearance Resolution

This option will generate a report about the clearance resolution between two selected items on the board.

For example, in my test board below I have selected two adjacent capacitors (hold the Command or Control key down, then click on the items you can to select). Then, click Inspect and “Clearance Resolution”:

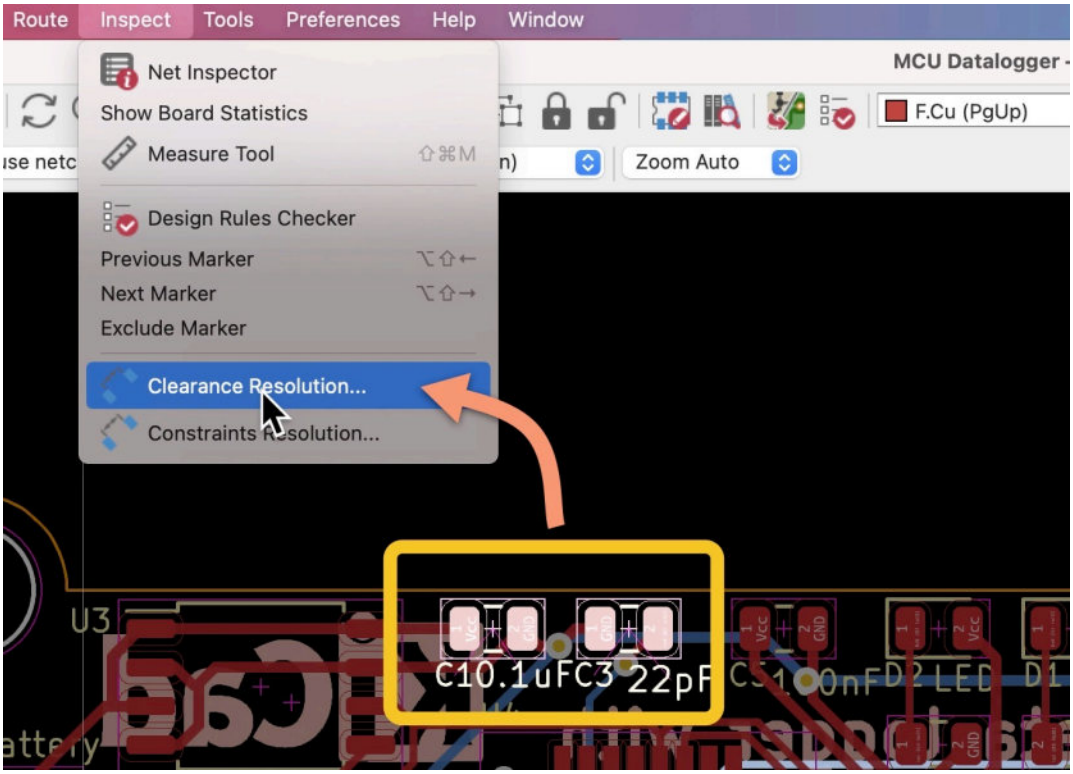


Figure 13.17.8: Select two items from the board, then click on Clearance Resolution.

Pcbnew will open up a new window with the report:

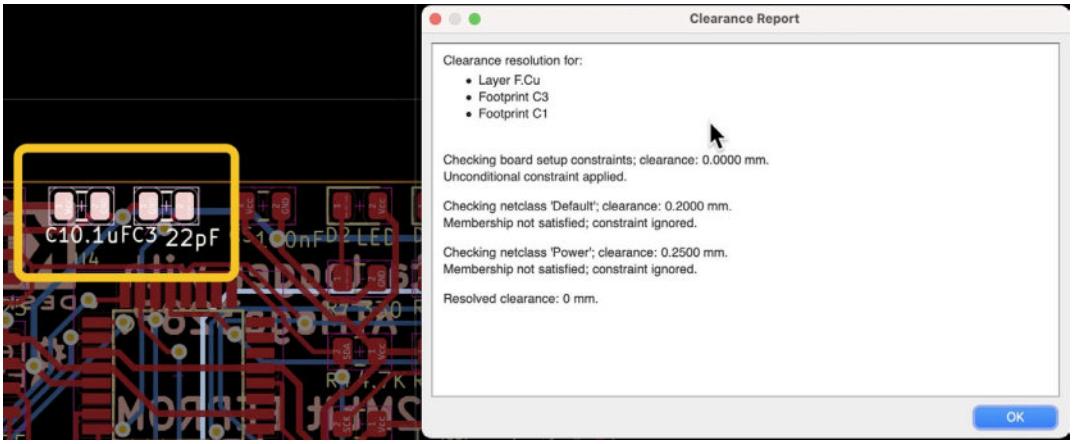


Figure 13.17.9: The clearance resolution report for the two capacitors.

This report will tell you if two items violate any clearance rules. In this example, there are no violations.

Constraints Resolution

Select one item from the board to use the Constraints Resolution tool, then click on Inspect and Constraints Resolution. In the example below, I have selected the MCU footprint:

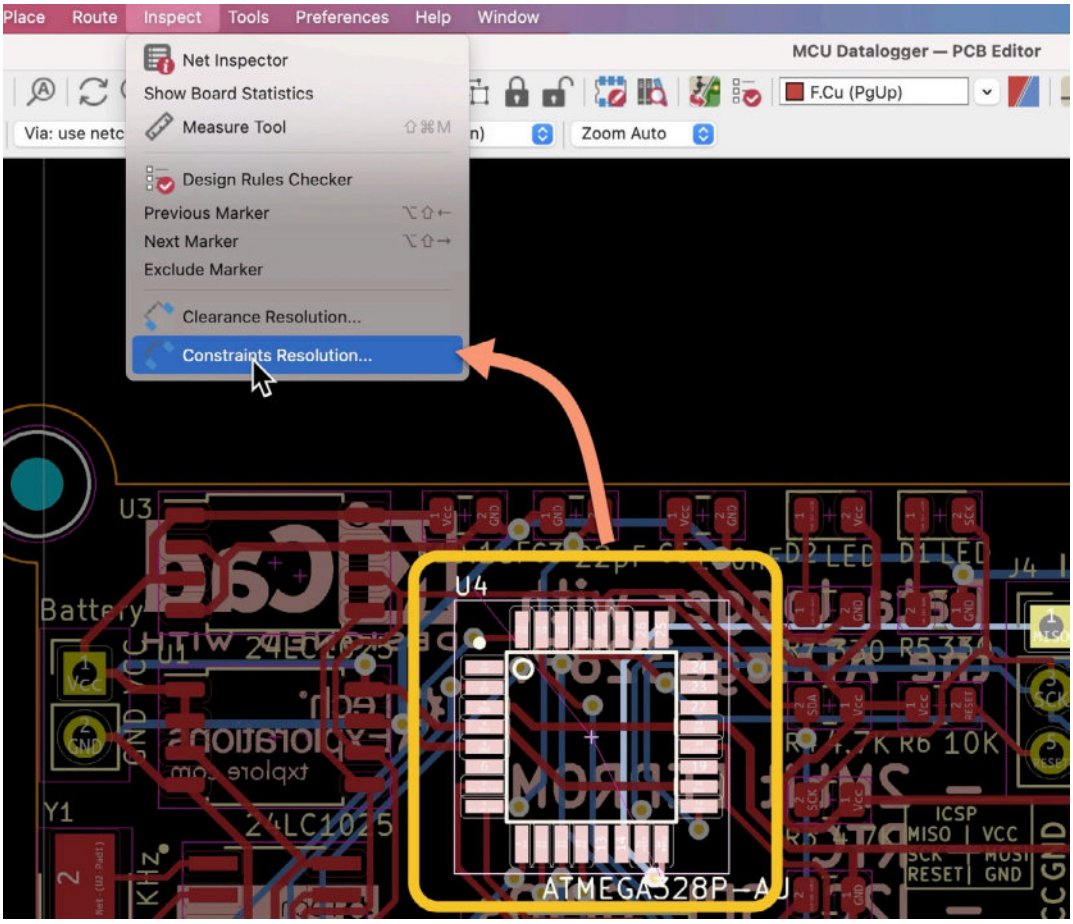


Figure 13.17.10: Select one items from the board, then click on Constraints Resolution.

Pcbnew will open up a new window with the report:

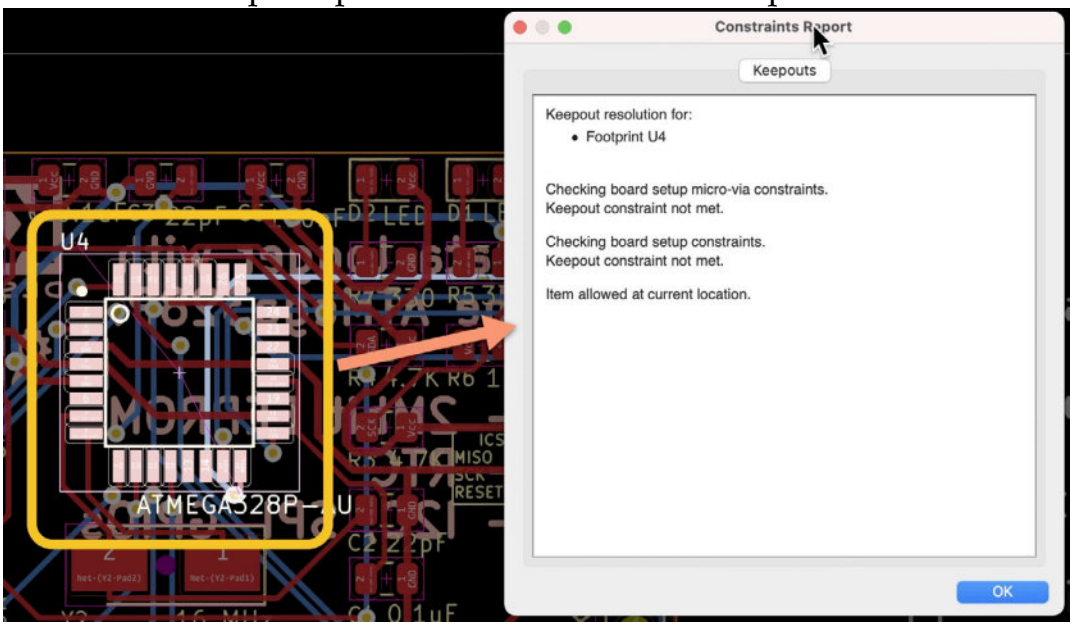


Figure 13.17.11: The Constraints report for U4.

The report provides information about vias, keep outs, location, and perhaps other constraints that I have not noticed up to the time I am writing these lines.

The report may indicate that certain constraints are not met (such as the keep out constraint in the example above). In my experience, the report may contain cases of possible violations that may or may not cause problems in your layout, and it supplements but does not substitute the Design Rules Check.

18. Single track and differential pair routing

Pcbnew offers you two modes to do routing (that is, to create copper tracks): single track and differential pair tracking. You can select the mode via the buttons in the right toolbar or the top menu (Figure 13.18.1).

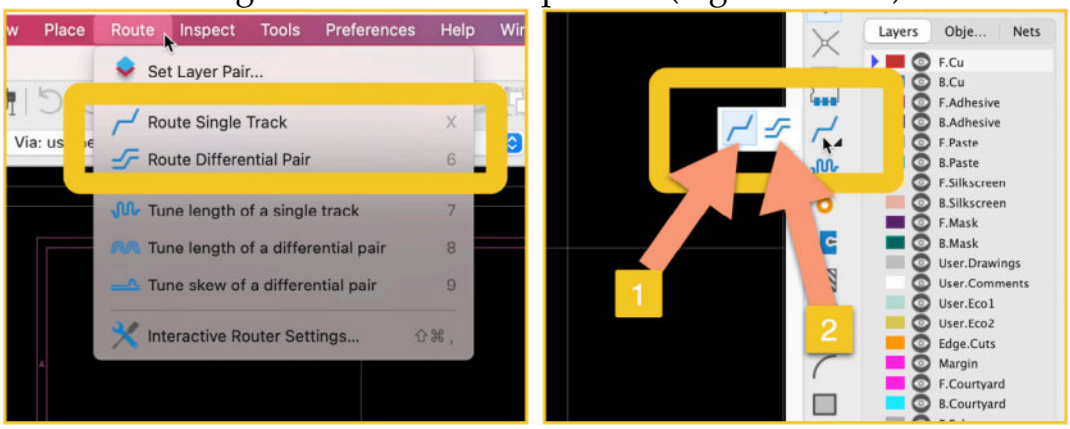


Figure 13.18.1: The two routing modes and how to enable them.

To choose a routing mode from the right toolbar, click and hold on the routing icon to expand it, and then click one of the two options.

By default, the single-track routing mode is selected. You probably have already used this mode as I have used it extensively throughout this book. In this chapter, I will formalize this knowledge and show you how to do differential pair routing.

I will begin this demonstration in Eeschema and design a simple schematic that consists of two 9-pin headers. I have used net labels that contain a single letter (like “A” or “B”), or a letter followed by “+” and “-“, or “_P” and “_N”. Here is my demo schematic (Figure 13.18.2):

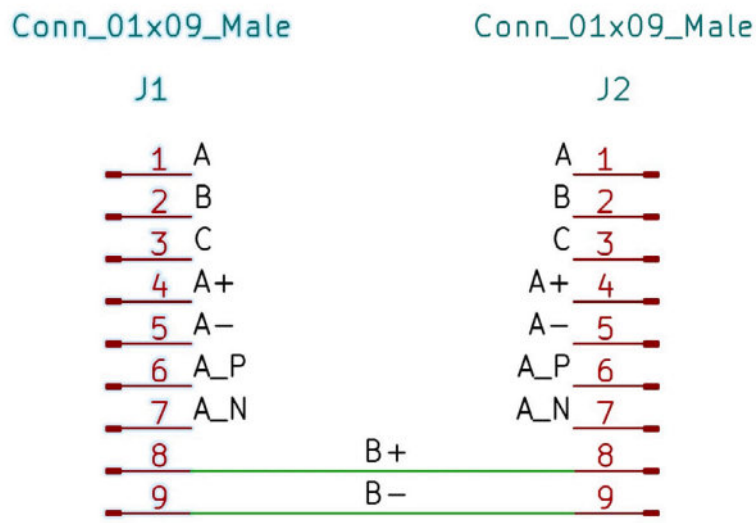


Figure 13.18.2: This schematic contains single nets and differential pair nets.

In the schematic above, I have used wires to connect pins 8 and 9 of J1 and J2. I have made the rest of the connections via the net labels.

A net label that ends with "+" and "-" or "_P" and "_N" represent differential pairs. In a differential pair, information is transmitted between the transmitter and the receiver using signals that travel through two wires. The signal levels (i.e., their voltage) are the same, but their polarity is the opposite. Examples of technologies that utilize differential pairs include DDR SDRAM, PCI Express, Serial ATA.

In KiCad, we can identify differential pair pins using the "+"/"-" or "_P"/"_N" notation. For example, a pin labeled "data+" and "data1-"consists of a differential pair. Pins with labels "data1_P" and "data1_N" also consist of a differential pair. You can use either postfix, but do not mix them. For example, "data1+" and "data1_N" do not consist of a differential pair.

Let's continue with this example. Import the schematic to Pcbnew so that you have this un-routed layout (Figure 13.18.3):

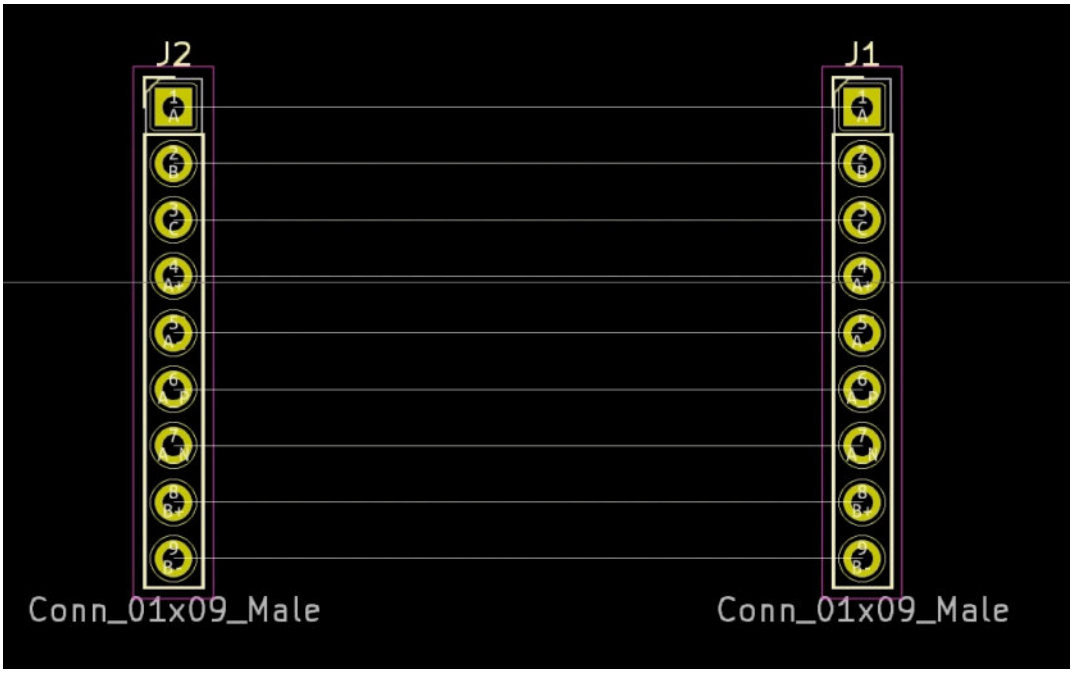


Figure 13.18.3: The un-routed layout.

Notice that the net names are inherited from the schematic in Eeschema and appear on the pads. We'll use this information to help distinguish between single tracks and differential pairs.

For pads "A", "B", and "C", we'll use single track routing since they don't have the "+"/"-" or "_P"/"_N" postfix are single tracks. Click on the single-track routing mode button from the right toolbar (or select "Route Single Track" from the "Route" menu, or type the "X" hotkey on your keyboard). With the single-routing mode selected, click on a pad to start drawing, then click again on the closing matching pad to finish drawing (Figure 13.18.4).

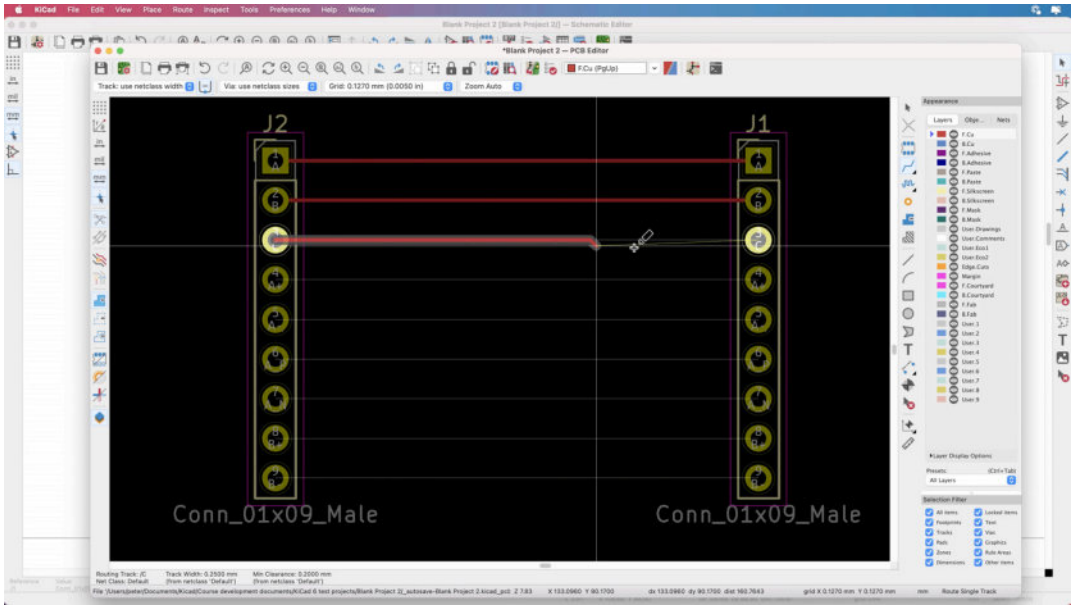


Figure 13.18.4: Single-track drawing.

You can create angled track segments by right-clicking. To complete the track, click on the corresponding pad. The thin rattiest lines can help you find and see the matching pad by linking it with the line and highlighting it so that it stands out from the rest.

Let's continue with the differential pars. I'll start with the pair "A+" and "A-". You can route a differential pair as if it was a single pair. Try this out, and confirm that it works. But a better way to route differential pairs is to select the differential pair routing mode.

Select the differential pair routing mode by clicking on the button with the two lines from the right menu bar (see "2" in Figure 13.18.1), or click on "Route Differential Pair" under the Route top menu, or just type the "6" hotkey on your keyboard.

With the differential pair routing tool selected, start by clicking on either of the two pads that make up the pair to start drawing ("1" in Figure 13.18.5 below). Continue to draw the two tracks as you navigate the pair closer to the target pads ("2", below). You can click to change directions as if you were drawing a single track. To finish drawing, click on either pair's target pads ("3", below).

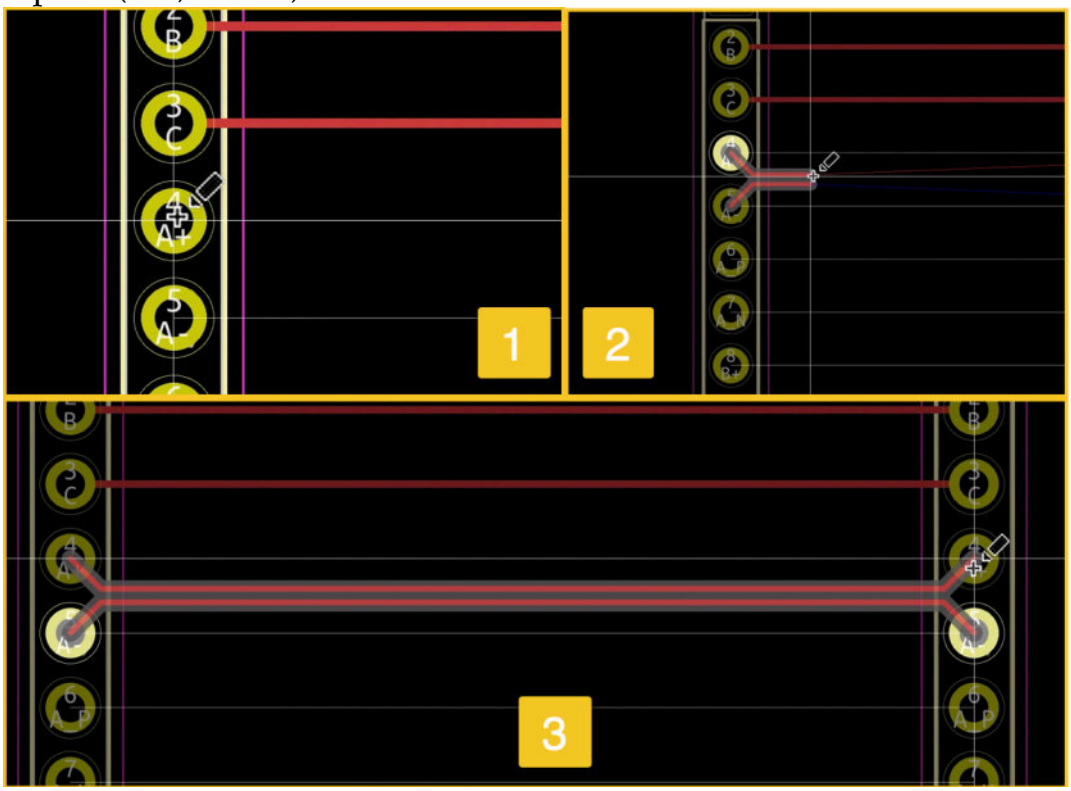


Figure 13.18.5: Drawing differential pair tracks.

Your differential pair tracks is complete. Drawing a differential pair track is very similar to drawing a single track when it comes to elements such as adding segments to make the track go around other elements, and vias. You

can see examples of this in Figure 13.18.6 below. The pairs “A_P”/“A_N” and “B+”/“B-“ contain multiple segments and vias. To create a via, the easiest way is to use the “V” hotkey while you draw.

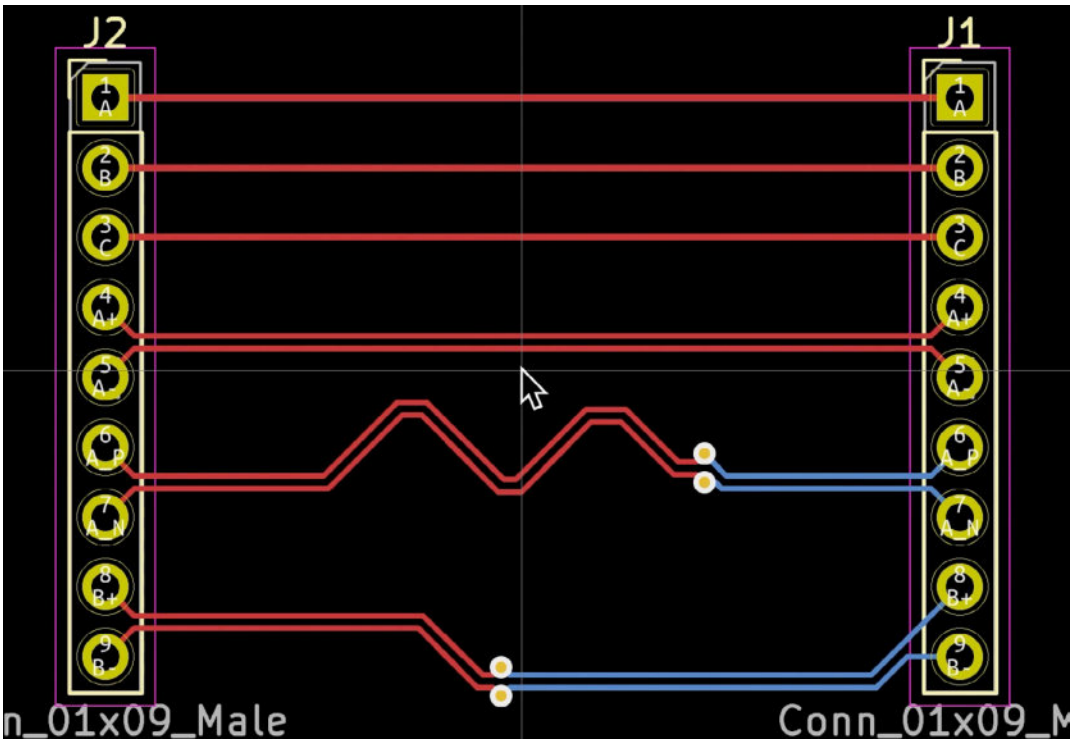


Figure 13.18.6: Differential pair tracks with multiple segments and vias.

19. Track length tuning

In this chapter, you will learn how to tune the length of a single track or a differential pair. Track length tuning is a function that allows you to tune the length of a track or differential pair so that it is an exact value. By tuning the length of a track, you can ensure that a signal can propagate from origin to destination in a specific amount of time. Track length tuning is important for applications such as in memory data or address buses, where signals that belong to the same bus must arrive at the destination within specific timeframes.

The two tools that you will learn about in this chapter can increase the total length by introducing new and repeated segments (meanders). You can set the target length, and the tool will draw enough new meanders to achieve that goal.

You can access the single track length tuning and differential pair length tuning tools via the top menu, under Route, or via the right-side toolbar (Figure 13.19.1):

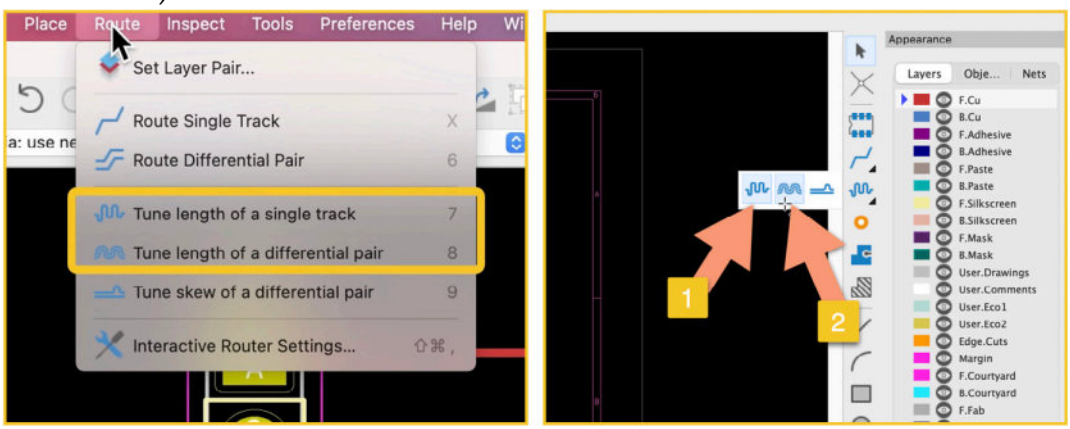


Figure 13.19.1: The two-track length tuning tools.

Let's look at an example of how length tuning works. We'll use the layout from the previous chapter, which looks like this (only showing the top two tracks):

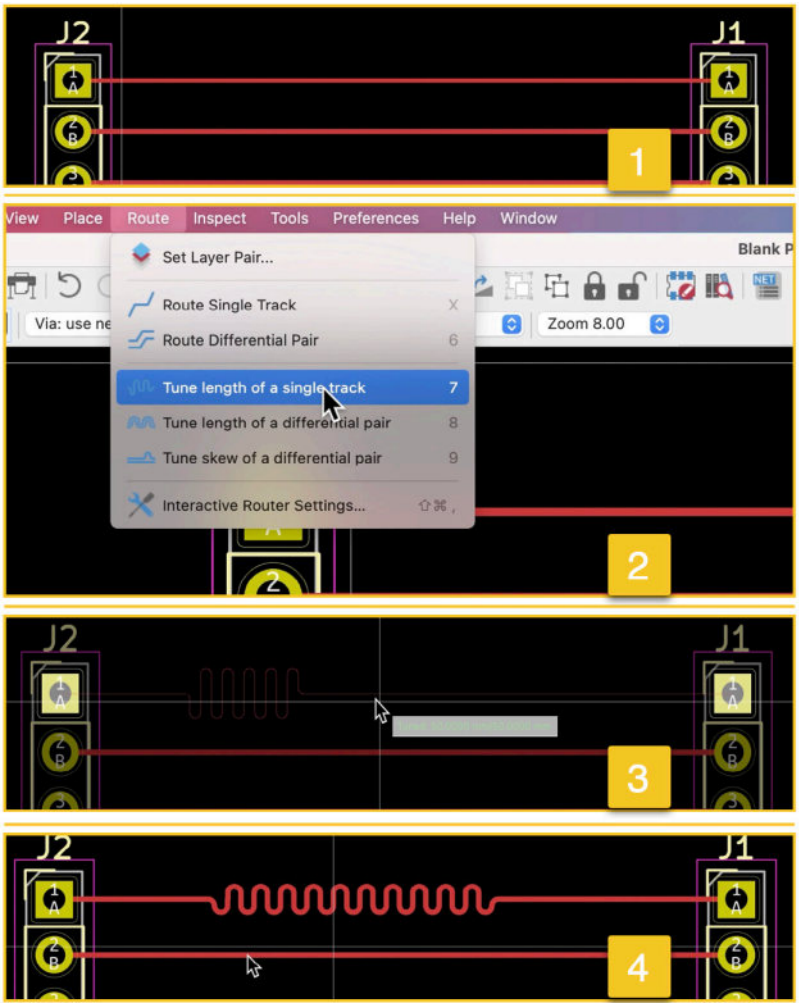


Figure 13.19.2: Fine-tuning the length of a single track.

In Figure 13.19.2 (above),I will increase the length of the track that connects pad 1 of J1 and pad 1 of J2 (“1”). Start by clicking on “Tune length of a single track” from the Route menu (“2”). I prefer to use the hotkey for this tool, which is “7”. Click anywhere on the track, but preferably towards the left or right end so that you have enough space to draw the new segments. I’m starting towards the left end of the track (“3”). Start moving the mouse towards the right end of the track (or whichever way most of the track lies), and notice the new segments have the shape of a meander. When the label that shows the current and target length becomes green, click again to finish the drawing (“4”). The length of the track is now tuned.

It is possible to configure how the length tuner tool works, including the target length. To do this, enable the single (or differential pair) length tuner from the menus or using the hotkeys (“7” for the single length tuner, and “8” for the differential pair length tuner), and then click on “Length Tuning Settings.” Below is the tuning settings window:

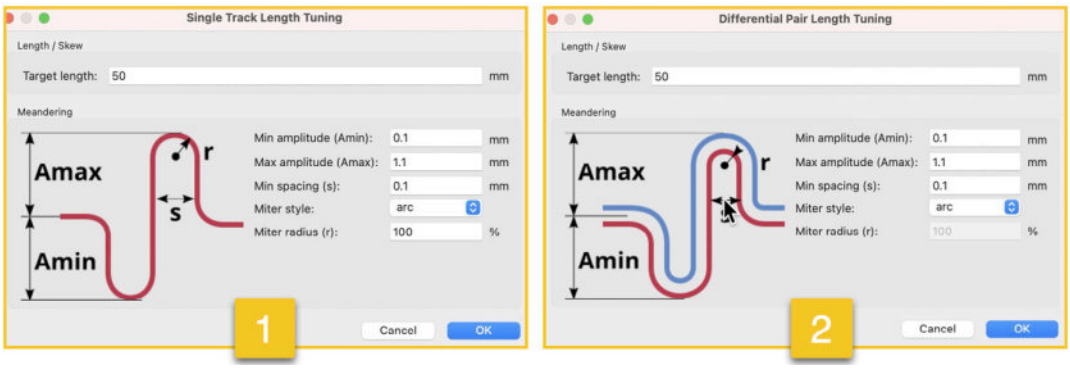


Figure 13.19.3: Length tuning settings for single track and differential pair.

The length tuning settings for both single track and differential pair are very similar. You can control the minimum and maximum amplitude, the spacing between the adjoining segments of a meander, the style (arc or 45 degrees), and the radius. Below you can see the difference between the arc and 45-degree styles:

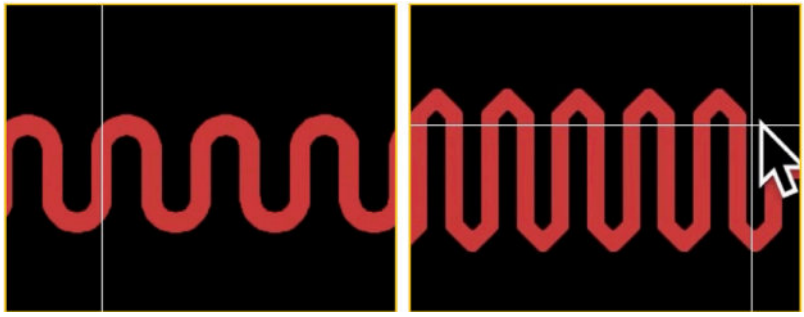


Figure 13.19.4: Meander styles. Arc (left) and 45 degree (right).

Figure 13.19.4: Meander styles. Arc (left) and 45 degree (right).

You can tune the length of a differential pair by selecting the tool “Tune length of a differential pair” from the Route menu or typing the hotkey “8”. Use the same drawing process as with the single track. A length-tuned differential pair looks like this (Figure 13.19.5):

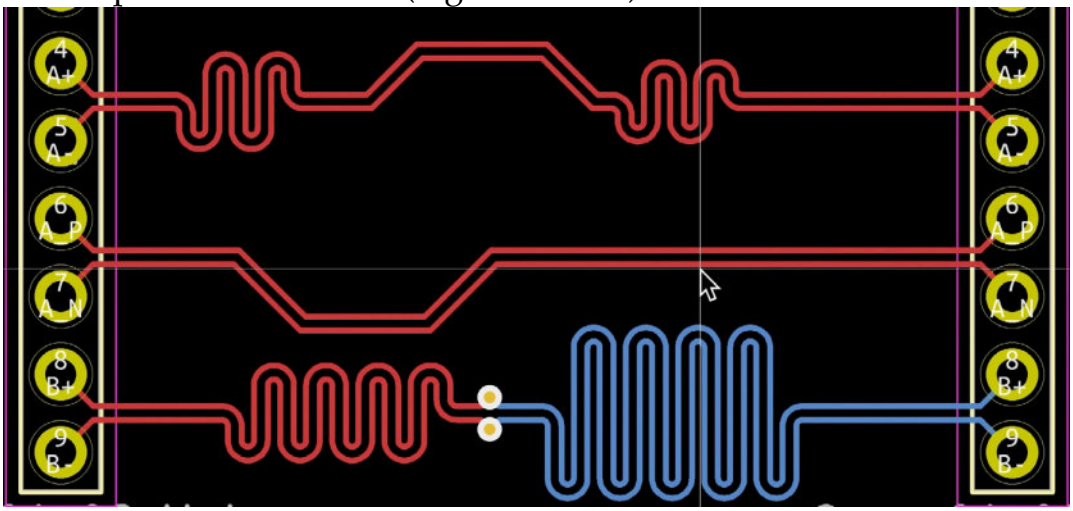


Figure 13.19.5: Length-tuned differential pairs.

In the examples above, I have created two separate meandering segments in the top one. The bottom differential pair includes two vias in the middle. The length tuning tool cannot continue drawing past a via or pad, so I had to stop the drawing and then create a new meandering segment on the other side to achieve the required length goal.

It is also possible to change the spacing and amplitude of the meander during drawing. You can do this using the hotkeys 1, 2, 3, 4 (active only during length-tuning drawing) or via the context menu (Figure 13.19.6).

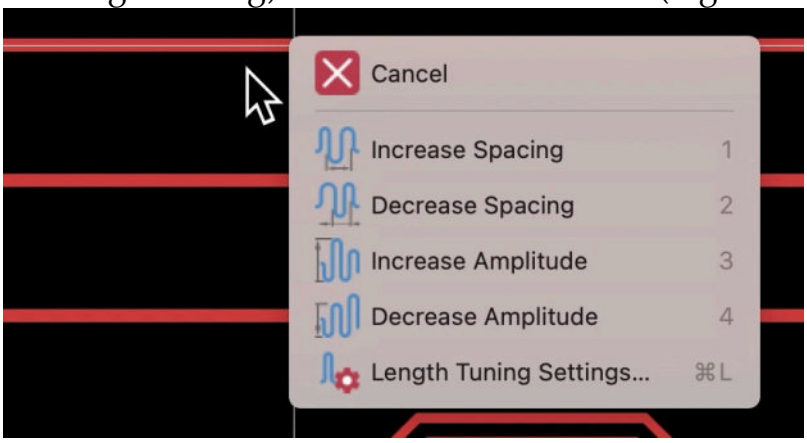


Figure 13.19.6: Space and amplitude control in the context menu.

By increasing or decreasing the amplitude and spacing of the meander, you can change the total footprint on the PCB required by the meander. Of course, as the amplitude gets bigger, the track will need more space around it.

The Route menu contains one more tuner, which you can use to tune the skew of a differential pair. You can learn more about this in the next chapter.

20. Differential pair skew tuning

In the previous chapter you learned how to tune the length of a single track or a differential pair. In this chapter, I'll show you how to use another tool in the Route menu, "Tune skew of a differential pair."

Differential skew[8] is a phenomenon that appears in high-speed, high-frequency applications. The phenomenon refers to the time difference between the two signals in a differential pair. In high-speed applications, a tiny difference in the time it takes for the two differential pair signals to travel to their destination can cause significant processing problems. Such a system would need to have the capability to detect data losses caused by the differential skew and recover. If we can minimize the differential skew as much as possible, then the system would spend less time recovering lost or corrupt data, and its overall performance would increase.

A common cause of differential skew is the difference in length of the two tracks that make up the pair. The differential pair skew tuner allows you to tune one pair of tracks independently of the other to minimize or eliminate the differential skew caused by the track length difference.

To use the tool, choose "Tune skew of a differential pair" from the Route menu, or type the "9" hotkey, or select it from the right toolbar (Figure 13.20.1):

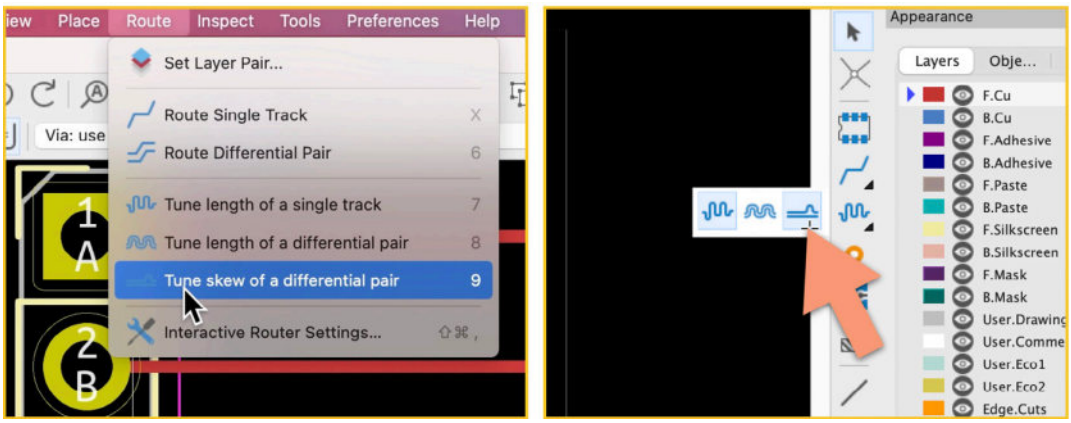


Figure 13.20.1: The differential pair skew tuner tool.

[8] For a discussion on differential skew, see https://www.edn.com/handling-differential-skew-in-high-speed-serial-buses/

Let's look at an example of how to use the differential pair skew tuner tool. My starting point is a simple straight differential pair track, as you can see below:

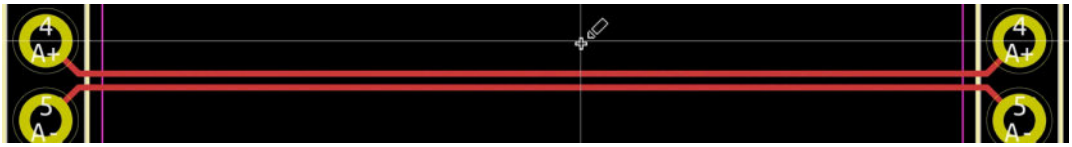


Figure 13.20.2: We'll tune the skew of this differential pair.

Imagine that the signal that flows through the top track ("A+" net) arrives at its destination slightly sooner than the bottom track. Maybe the material used is different, or perhaps the geometry of the track for track of net "A-" is such that it causes a delay. Whatever the reason, I would like to equalize the signal's propagation time by introducing a slight delay in the top track.

First, I will find out how long are each of the tracks makes up this differential pair. You can use the Net Inspector for this under the Inspect menu. Invoke the Net Inspector and click on the "A+" and "A-" nets to see their rows.

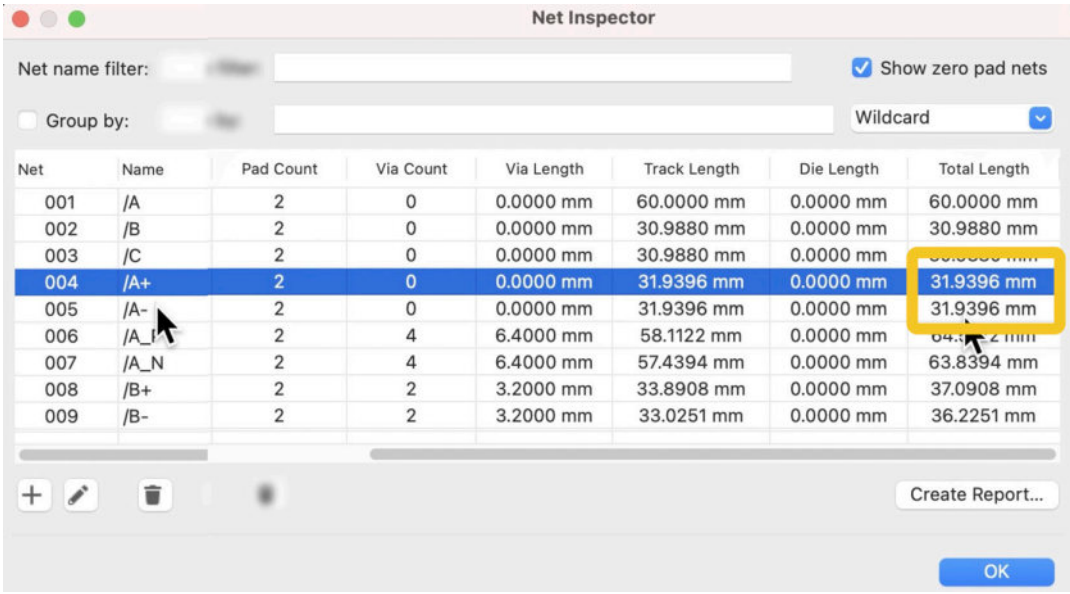


Figure 13.20.3: The current length of the two tracks in the example differential pair.

The two tracks in the example differential pair have an equal length of 31.93 mm. I will tune the length for the "A+" track to 35 mm.

Select the "Tune skew of a differential pair" tool from the Route menu (I prefer to type the "9" hotkey). Then, right-click on the top track of the differential pair to show the context menu and select Length Tuning Settings. The window that appears looks like this:

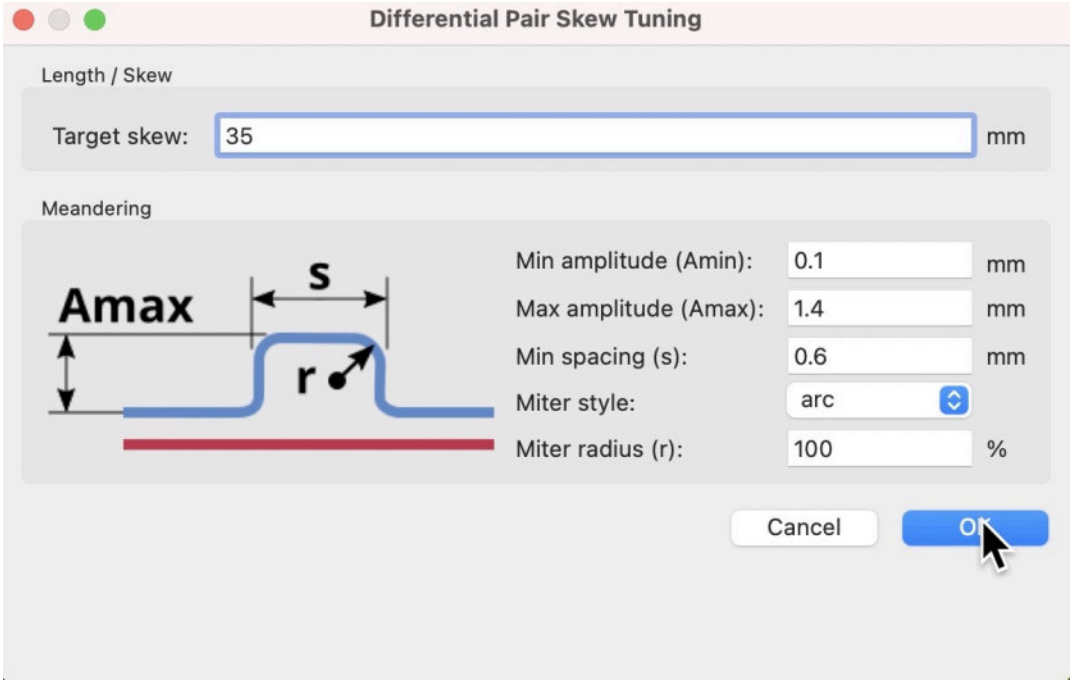


Figure 13.20.4: The Differential Pair Skew Tuning window.

Change the Target skew to 35 mm. The rest of the settings are fine as they are. Click OK.

Concerning figure 13.20.5, move the mouse towards the right to start drawing the new meandering segment and notice how the values in the label change (1). When the length reaches 35 mm, the label becomes green as I have attained the set target. Click again to finish drawing (2).

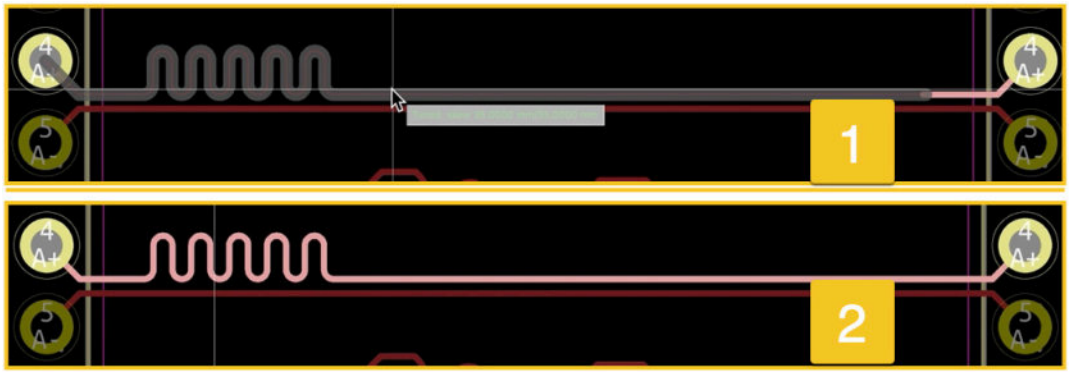


Figure 13.20.5: Increasing the length of one of the tracks in a differential pair.

The differential pair track now has a slightly different length. The track that belongs to the “A+” net is somewhat longer than the track of “A-. “

21. Interactive router modes

Kicad 6 contains an advanced interactive router. This interactive router assists you as you draw traces. For example, the router can do things such as:

- Highlight violations, such as trying to draw a trace over a pad or another route.
- Automatically push traces aside to make room for the trace you are currently drawing.
- Find a route around an obstacle when the obstacle is immovable.
- Optimize the geometry of the track as you draw it.
- Remove redundant tracks as you draw new tracks between the same endpoints.

The interactive router in KiCad 6 has three modes:

- Highlight collisions.
- Shove.
- Walk around.

You can find the interactive router settings under Route —> Interactive Router Settings:

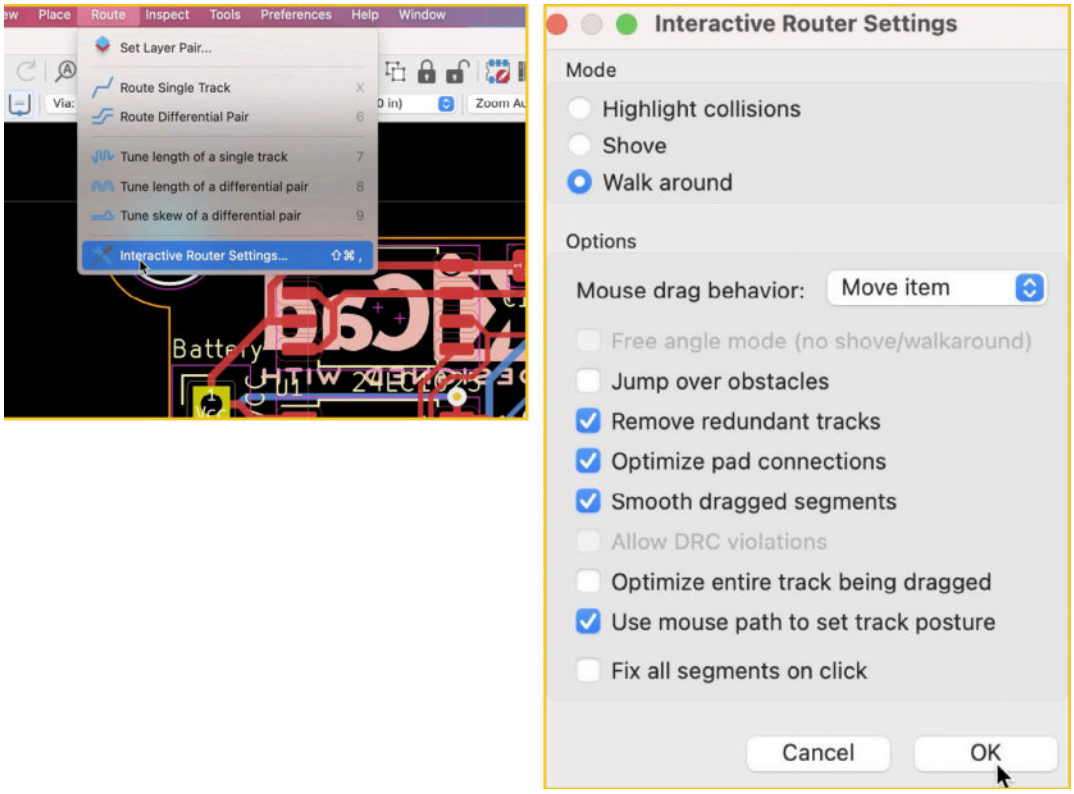


Figure 13.21.1: The Interactive Router Settings window.

You can choose a mode by selecting one of the radio buttons in the Mode group of the settings window. You can configure the specific features of each by selecting them from the Options group of widgets.

In this chapter, you will learn how to use the three interactive router modes.

Walk around

First, let's look at "Walk around." This is the mode I use most often. Ensure that "Walk around" is selected in the Mode group, and click "OK."

To test this routing mode, I am using a layout from one of the projects in this book. You can see my starting layout in Figure 13.21.2, frame 1. I have deleted the trace between the two Vcc pads. Then, I typed "X" to enable the single trace drawing tool. To understand what is happening in these screenshots, I use a green circle to mark the track origin and an orange circle for the mouse pointer's position, and trace ends.

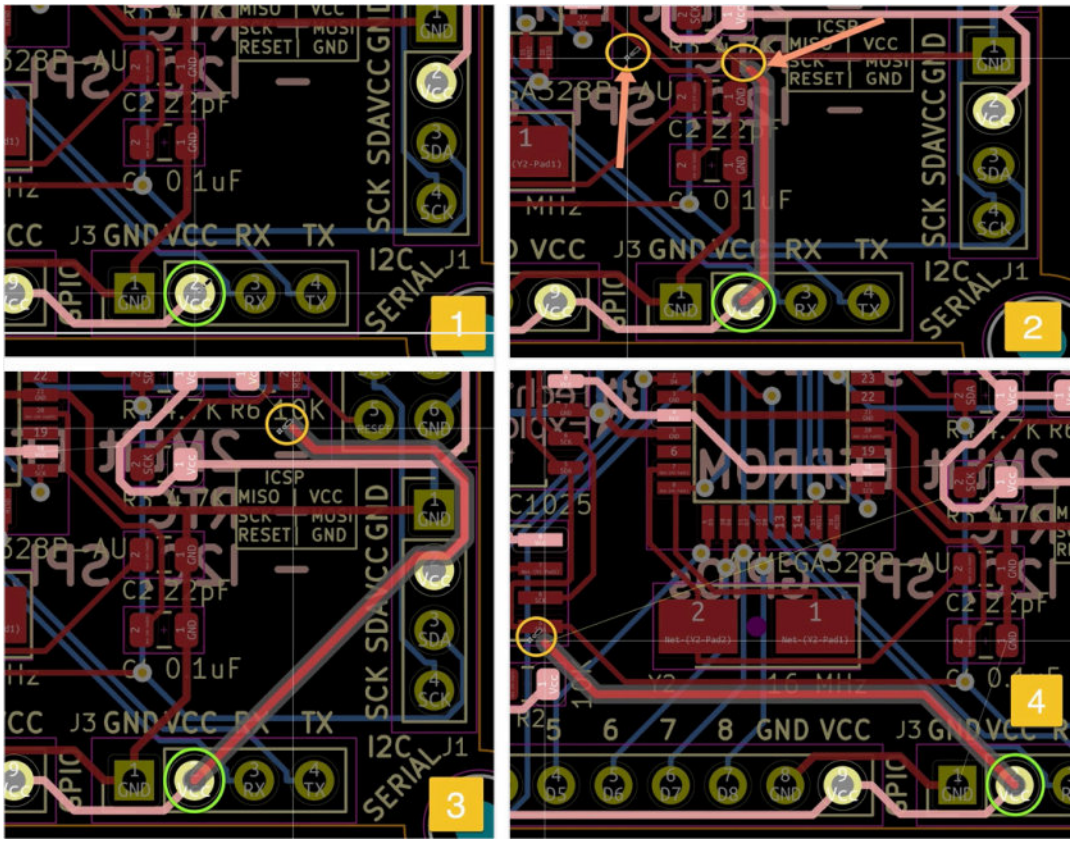


Figure 13.21.2: The interactive router in “Walk around” mode.

In frame 2, I start drawing a new track from pad 2 of the Serial connector. There are several tracks and pads that block the path of the new trace. Notice that the mouse pointer is towards the left of the screenshot, beyond the blocking pads and traces of the two capacitor footprints. My new trace cannot find a path through those obstacles and remains separated from the mouse pointer. There is no path through.

In frame 3, I could navigate the new trace around the right side of the I2C connector, in between the VCC and GND pads. The new trace path took it over an existing trace that also belongs to the Vcc net. Because both traces belong to the same net, there is no violation, and the interactive router did not block the drawing.

In frame 4, I deleted an existing trace between the Serial connector pad 1 (GND) and pad 1 of C1. I did this to draw a new trace towards the left side of the PCB. You can see the new trace starting from the Vcc pad (in the green circle) until it reaches the left edge of the screenshot.

In “Walk around” mode, the Interactive router will not make any automatic changes to the layout. It considers all elements as fixed. The router will try to find a path around existing elements taking a queue from the movements of the mouse pointer.

In my experience, the “Walk around” mode is the safest one to use, and I use it as my default.

Shove

Let’s continue with Shove. Bring up the Interactive Router Settings window from the Route menu, choose the Shove mode, and click OK. I’ll use the same layout as in the “Walk around” demo to demonstrate how Shove works. You can see my starting layout in Figure 13.21.3, frame 1.

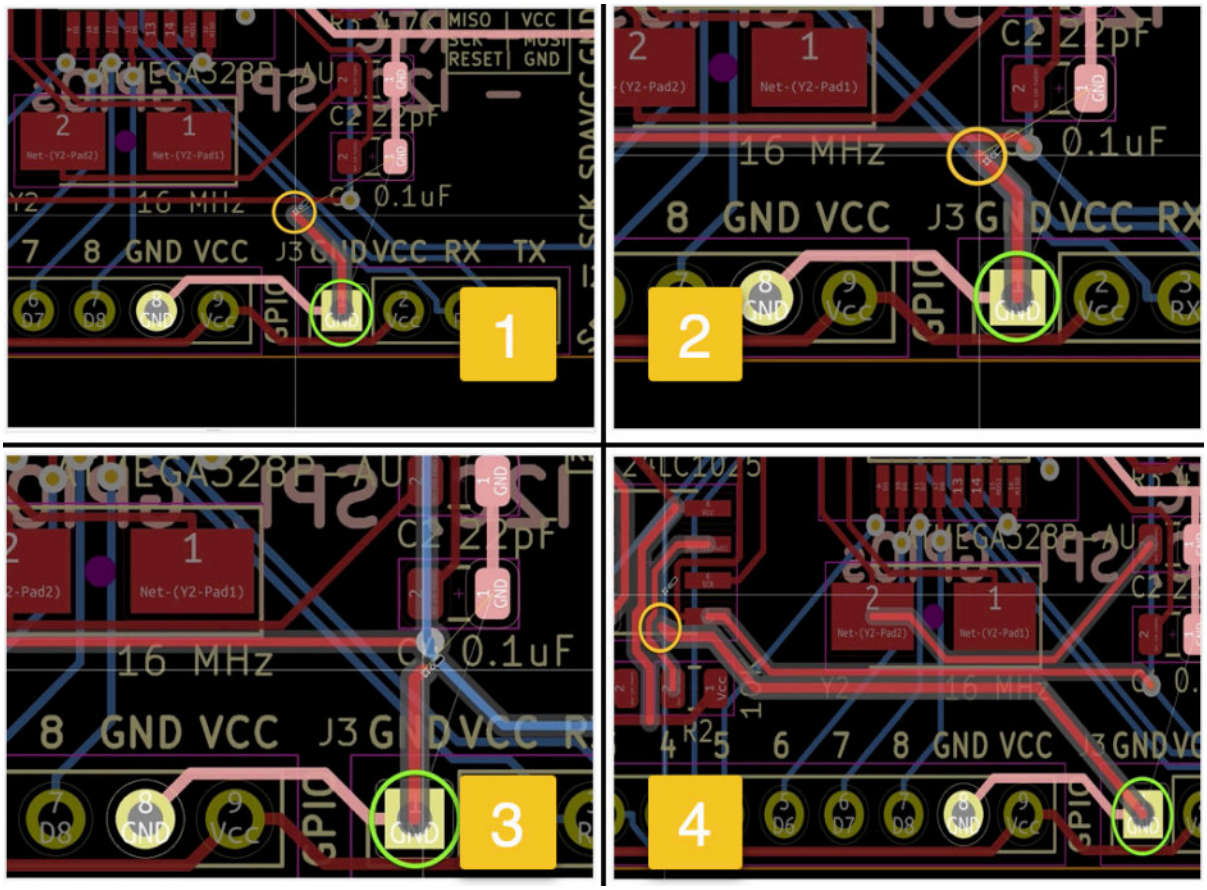


Figure 13.21.3: The interactive router in Shove mode.

In frame 1, notice how the new trace and mouse pointer are a few pixels below an existing trace in the front copper layer. In frame 2, I have moved my mouse pointer a few pixels higher in the area that earlier was occupied by the existing trace. The interactive router has highlighted the collision by giving a bright “halo” to the affected track, and it has used the affected track upwards to make room for the new track. Even though a collision is detected, the interactive router resolves the violation by moving the existing trace out of the way.

In frame 3, I move the mouse pointer towards the via, which occupied the space below pad 2 of the capacitor footprint. Again, the interactive router detected the collision and resolved the violation by moving the via (and the traces connected to it) out of the way.

In frame 4, I have drawn a new trace towards the left side of the board. The interactive router re-positioned any existing traces that were in the way.

The interactive router will try to push existing items out of the way if the conditions are favorable for this to happen. If an element, like a via, or trace, is marked as "locked," the router will keep them in place. It is also possible for the geometry around a trace of via to be such that moving is impossible; perhaps there is no space available.

Highlight collisions

Let's look at the last mode, Highlight Collisions.

Bring up the Interactive Router Settings window from the Route menu, choose the Highlight Collisions mode, and click OK. To demonstrate how Highlight Collisions works, I'll use the same layout as in the Walk around demo. This mode only provides violation feedback. It will not attempt to make any changes to the layout, and will not make any effort to go around existing elements trying to optimise its geometry. This mode gives you maximum freedom to draw a trace. It will even allow you to violate design rules.

Refer to Figure 13.21.3 (below).

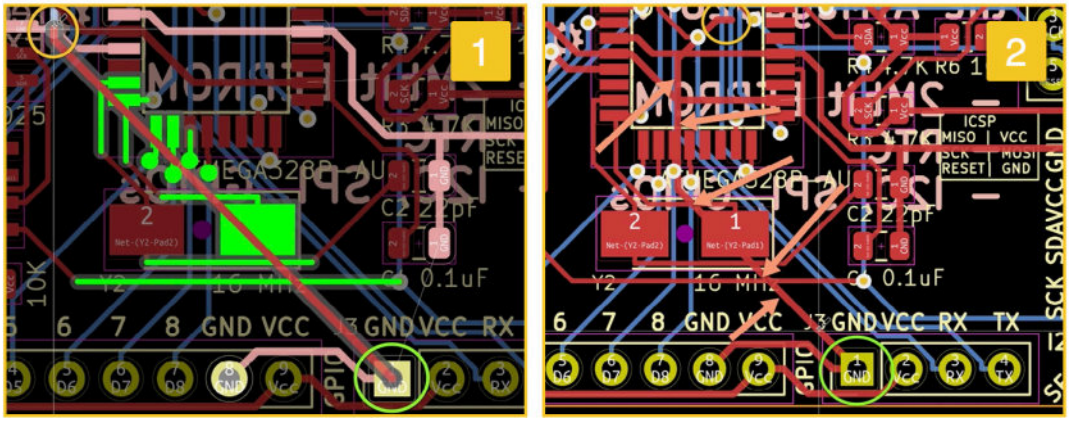


Figure 13.21.3: The interactive router in Highlight Collisions mode. Green indicates a violation.

In frame 1, I am drawing a new trace over existing elements. The interactive router will not enforce the design rules or find a legitimate path for the trace. It will simply highlight any violations in green.

Further, the router will not block me from committing a new trace that violates the design rules. In frame 2, I have drawn a new trace with a path that goes over tracks and pads that belong to other nets.

As you can see, the interactive router in Highlight Collisions mode is permissive. Use with care.

Typically, I will do the bulk of my work in “walk around” mode. Occasionally I will switch to Highlight Collisions mode to deal with difficult situations where “walk around” mode just gets in the way. I rarely use Shove mode because I prefer not to have my carefully drawn tracks altered by KiCad.

22. The footprint wizard

KiCad ships with an extensive library of footprints. If you need a footprint that is not available in those libraries, there is a good chance that you will be able to find it in repositories like Snapeda. But even if that fails, you can create custom footprints using one of two tools that KiCad provides: the Footprint Editor and the Footprint Wizard. You can learn how to use the footprint editor in a dedicated chapter elsewhere in this book.

In this chapter, you will learn how to use the Footprint Wizard. With the footprint wizard, you can create a custom footprint for a standard set of components quickly and easily.

The wizard is part of the footprint editor. To use the wizard, start the footprint editor from the main KiCad project window (see frame 1 in Figure 13.22.1).

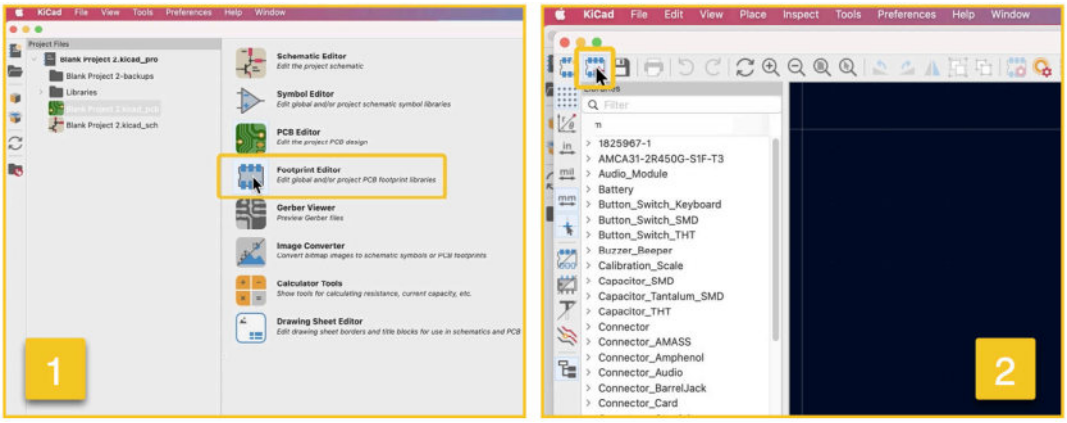


Figure 13.22.1: Starting the footprint wizard.

Once in the footprint editor, click on the Wizard button in the top menu (2).

The wizard will take you through a series of steps. In the first step, you will select the footprint generator that is most appropriate for the type of footprint that you want to create. There are generators for BGA, QFN, DIP, SOIC, and many more types of components.

Create a DIP footprint

For this demonstration, I will create a new footprint for a DIP component. From the list of available generators, I will choose row 8, “S-DIP” (Figure 13.22.2), and click OK.

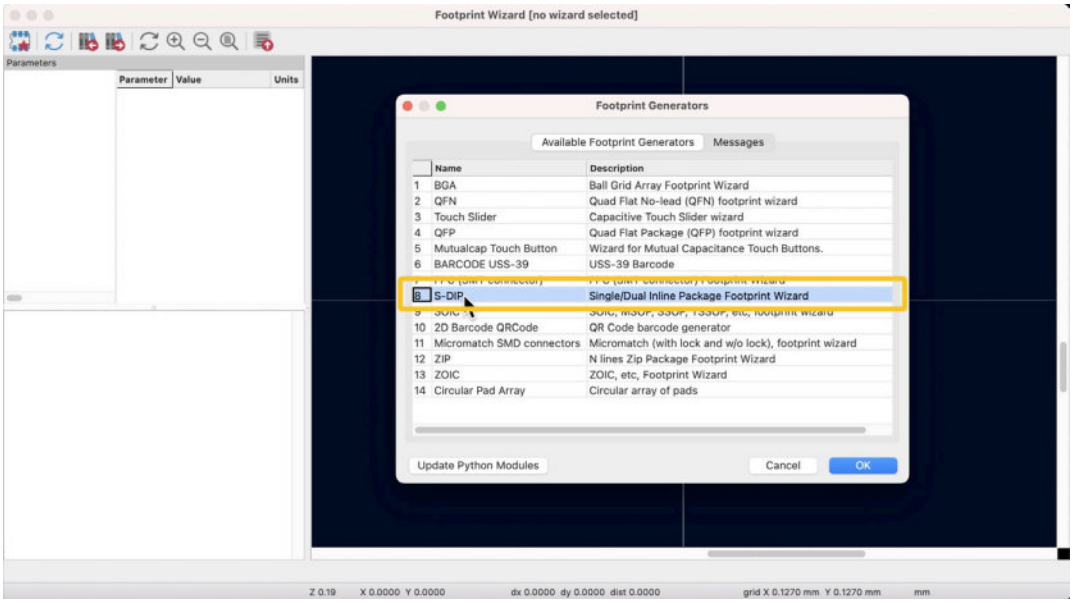


Figure 13.22.2: The footprint wizard, footprint generators window.

I remind you that earlier in this book, I used the footprint editor to create a DIP footprint manually. This is an opportunity to experience how the wizard can speed up the work for standardized footprints.

When you dismiss the footprint generators window, the footprint wizard will display its default settings in the right design pane. On the left side is a list of parameters that you can edit and customize the footprint (Figure 13.22.3).

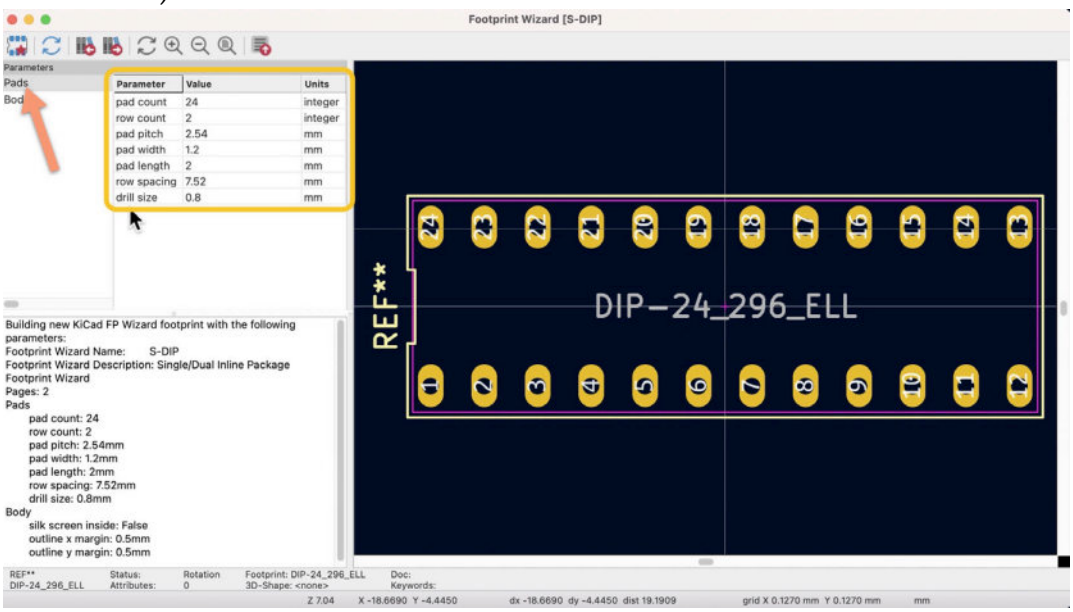


Figure 13.22.3: The Footprint Wizard with the default Pad settings from the DIP generator.

As you change the settings in the parameters pane, the wizard updated the footprint in the right pane.

I made these changes:

- Pad count to 32.
- Pad pitch to 1.54 mm.

In the left pane, click on Body. This will show body parameters in the middle pane (Figure 13.22.4):

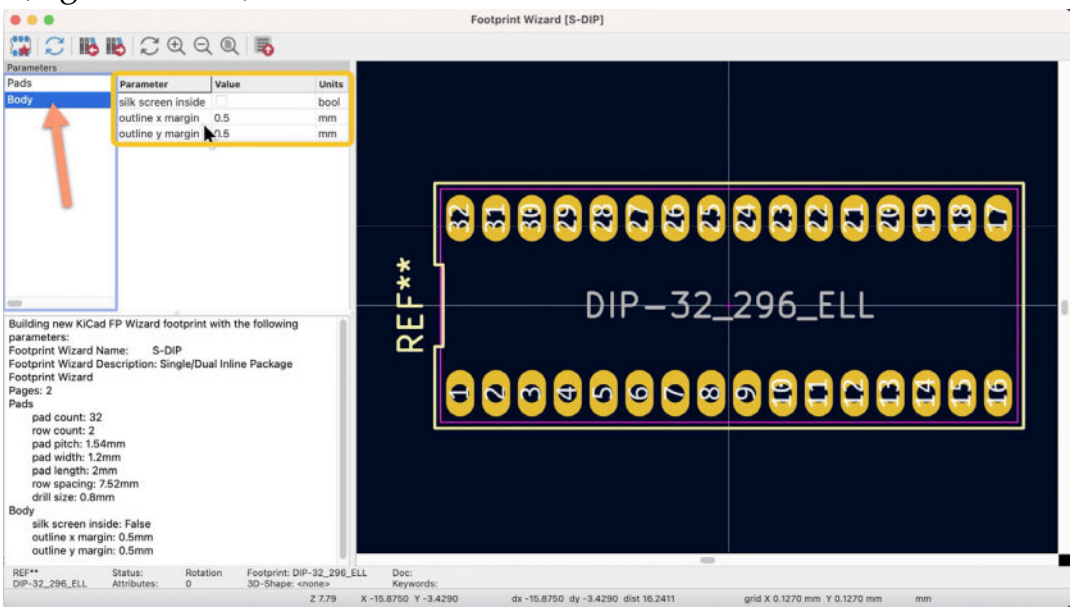


Figure 13.22.4: The Body parameters in the footprints wizard.

I have made a couple of changes in the body parameters:

- Outline x margin to 1.5 mm.
- Outline y margin to 1.5 mm.

Below you can see my new footprint. It took me around 60 seconds to design:

Figure 13.22.5: Done in 60 seconds.

When you finish work on your new footprint, you can export it from the wizard to the footprint editor by clicking on the import button:

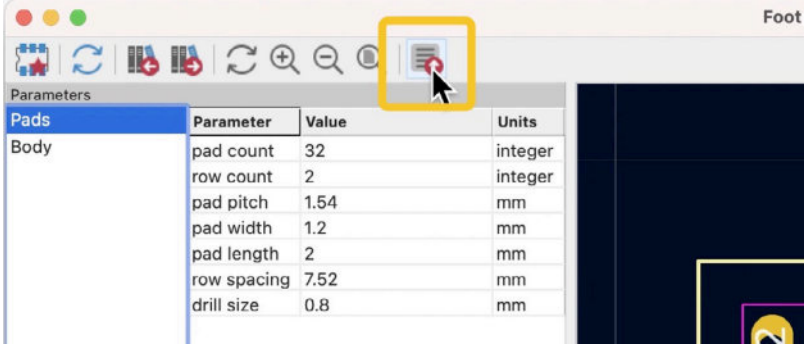


Figure 13.22.6: Import the footprint into the footprint editor.

In the footprint editor, you can continue to customize your new footprint or save it into an existing or new library so you can use it in a project. If you don't know how to do this or need a refresher, see Create a custom footprint, 4, Silkscreen layer from earlier in this book.

Create a Barcode

The footprint wizard contains a 2D Barcode and QRCode generator. You can use this generator to create bar-code graphics that encode information such as a webpage URL with information about your board so that the end-user will not have to type the URL.

I'll show you how that works.

Start the footprint wizard, select the "2D Barcode QRCode" row, and click OK. The wizard will show you this default QR code:

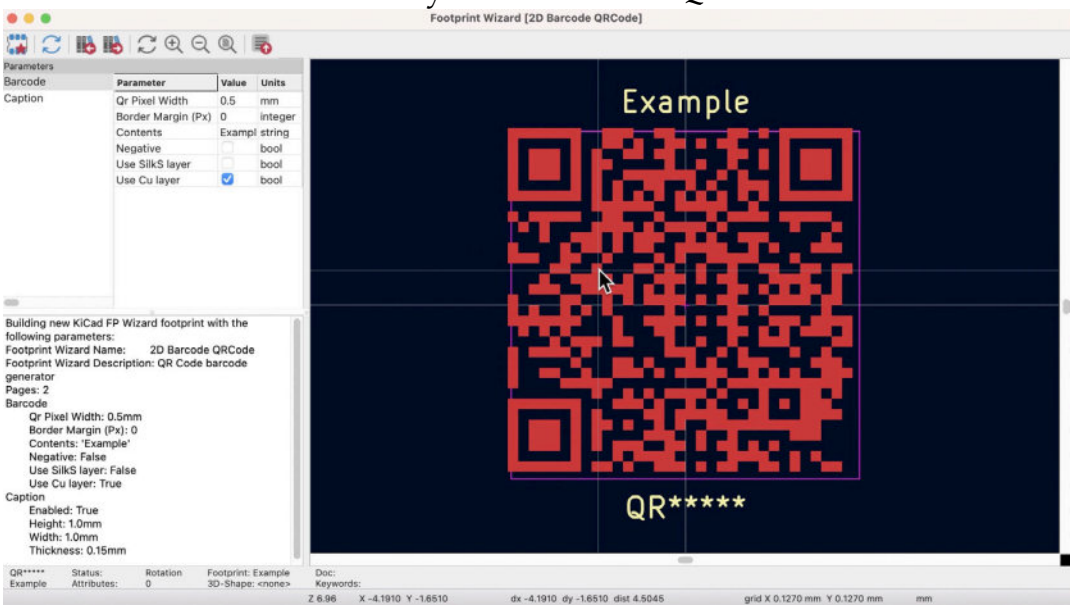


Figure 13.22.7: The default QR code in the footprint wizard.

In the Barcode parameters, I have changed the contents field to "techexplorations.com".
In the Caption parameters, I changed heigh and width to 1.5 mm.
Continue as you would with any other wizard-generated footprint:

1. Export the footprint to the footprint wizard.
2. Save the new footprint to a new or existing footprint library.
3. Use the new footprint in your project.

Below you can see my new QR code footprint on a PCB, in the 3D viewer:

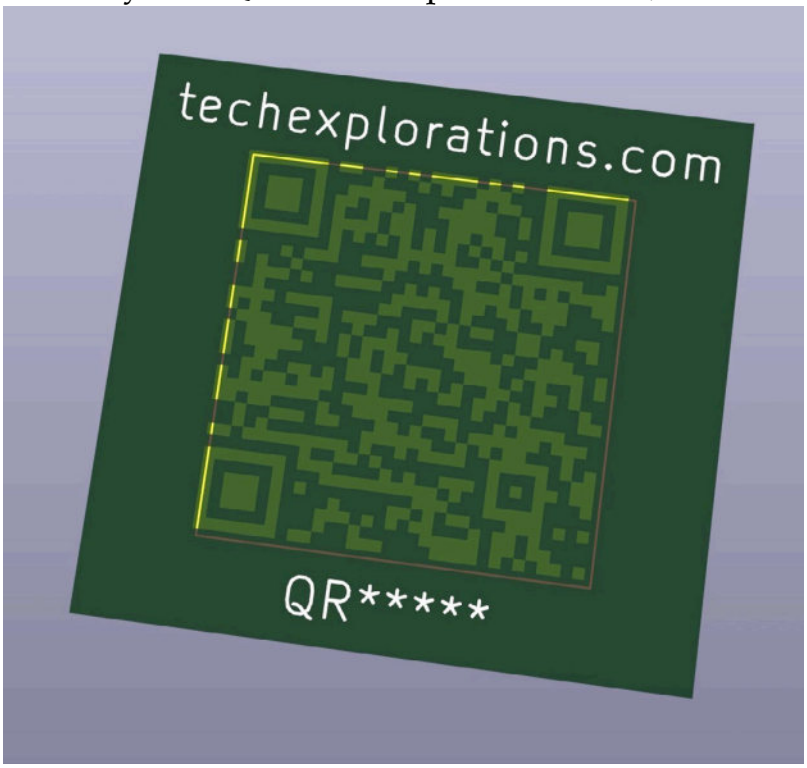


Figure 13.22.8: A QR code footprint generated by the footprint wizard.

23. Pin and wire highlighter tool

It is often difficult to see which pins are connected to other pins in busy schematics and quickly determine which wires belong to the same net. To help in such situations, KiCad provides a tool called “Highlight wires and pins of a net.” You can find it on the top of the right toolbar in Eeschema:

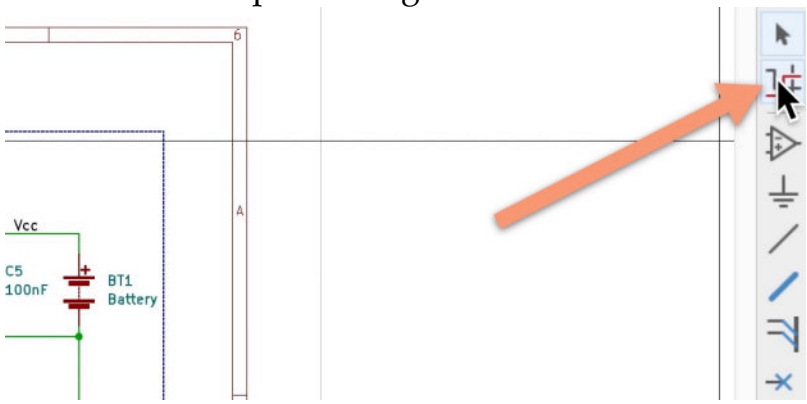


Figure 13.23.1: The ”Highlight wires and pins of a net” tool.

I’ll show you how it works.

Click on the highlight button in the right toolbar to enable the tool. In the example below, I have clicked on one of the wires or pins that belong to the GND net. The highlighter marked all GND net member wires and pins in pink:

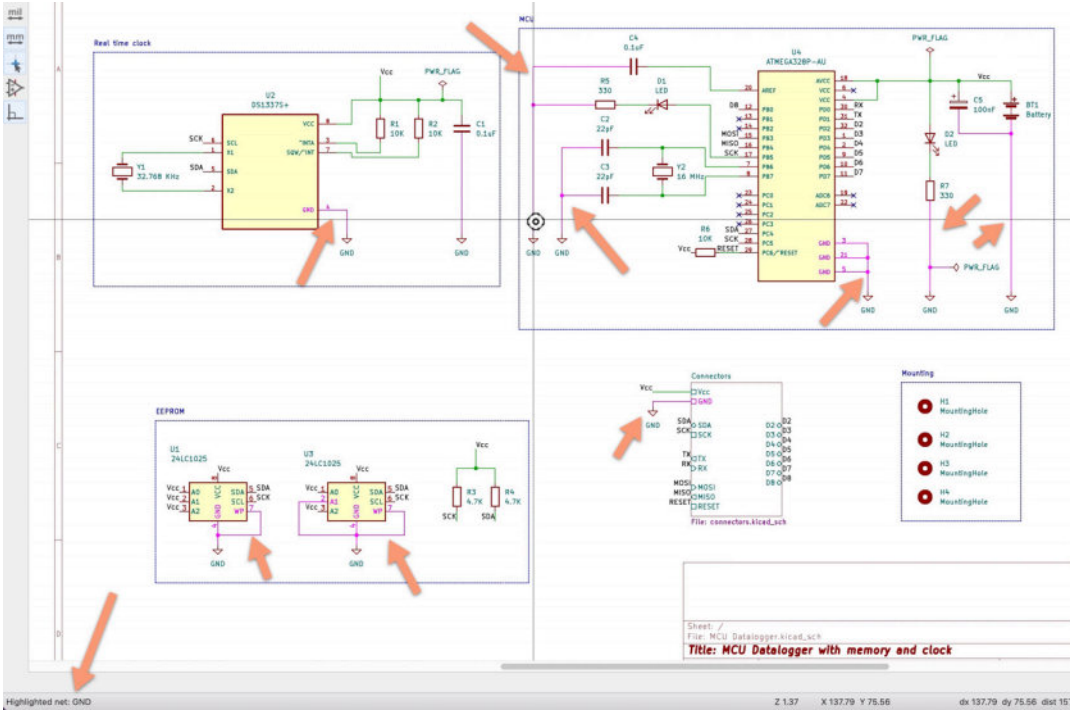


Figure 13.23.2: All members of the GND net are highlighted in pink.

Also, notice the message that appears in the status bar (bottom left corner), indicating the highlighted net is “GND.”

Unfortunately, if your schematic spans multiple pages, the net selection will not carry through from one to the other.

By default, the highlighter color is light pink. If you’d like to change it to something else, like red, you can do so via the preferences window. Go to Preferences, Schematic Editor, Colors, and look for “Highlighted items.” Click on the color box to bring up the color picker and choose a different color. Click OK to exit the Preferences window.

Below you can see the members of the GND net highlighted in red.

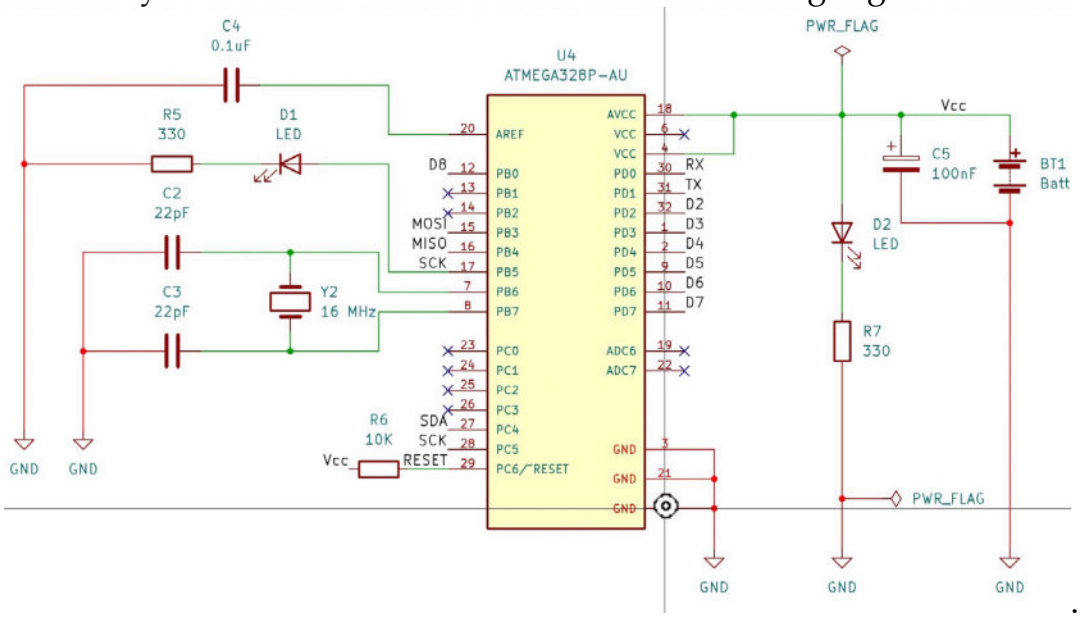


Figure 13.23.3: All members of the GND net are highlighted in red.

24. Pcbnew Origins

KiCad, as with any other CAD application, has a coordinate system. In Eeschema and Pcbnew, you can see the coordinates of your mouse cursor at the status bar. By default, the origin of the coordinate system is located at the top left corner of the editor. The X (vertical axis) and Y (horizontal axis) coordinates increase as you move your mouse down and left of the origin.

In Eeschema, you can see the X and Y values at three sample positions:

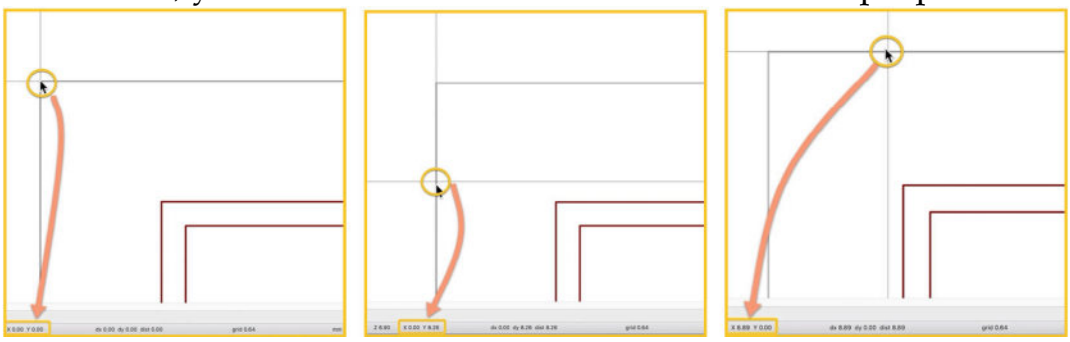


Figure 13.24.1: The default coordinate origin in Eeschema.

Similarly, you can see the default coordinate origin in Pcbnew below:

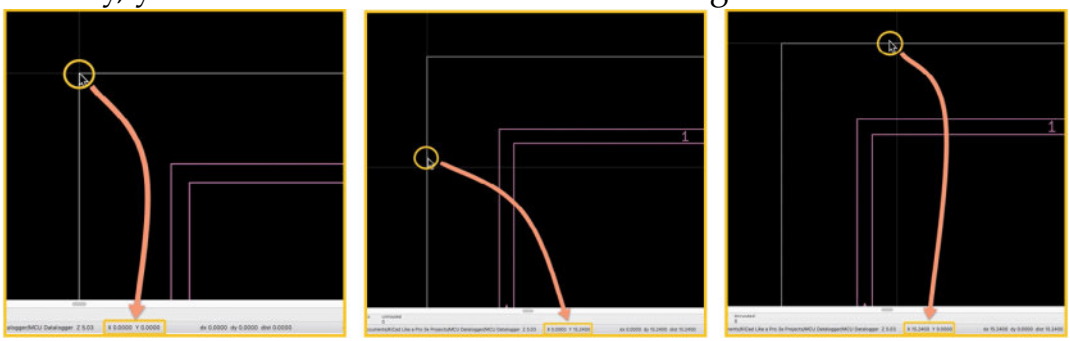


Figure 13.24.2: The default coordinate origin in Pcbnew.

In Eeschema, it is not possible to change how the coordinate system works. This is not a problem since Eeschema is electrical design software, and the coordinate system does not need to relate to “real world” coordinates.

Pcbnew, however, is different. Pcbnew produces real-world output because the PCB that you design in the layout editor will yield files containing coordinates that the PCB manufacturing equipment must use. It is also possible that part of your PCB design toolchain has other mechanical CAD software. All of those systems must share a common coordinate system so that coordinate references are compatible.

In Pcbnew, it is possible to both customize the coordinate system and to change its origin.

Grid origin to bottom left

Repositioning the grid origin to the bottom left of the editor page is a common question by people coming to KiCad from other CAD applications.

Many CAD applications have a coordinate system that originates from the bottom left corner of the editor. In Pcbnew, the default origin is at the top left. To change the default origin in Pcbnew, open the KiCad preferences window, and click on “Origins & Axes” under “PCB Editor.”

You can see the default settings for the Origins & axes below:

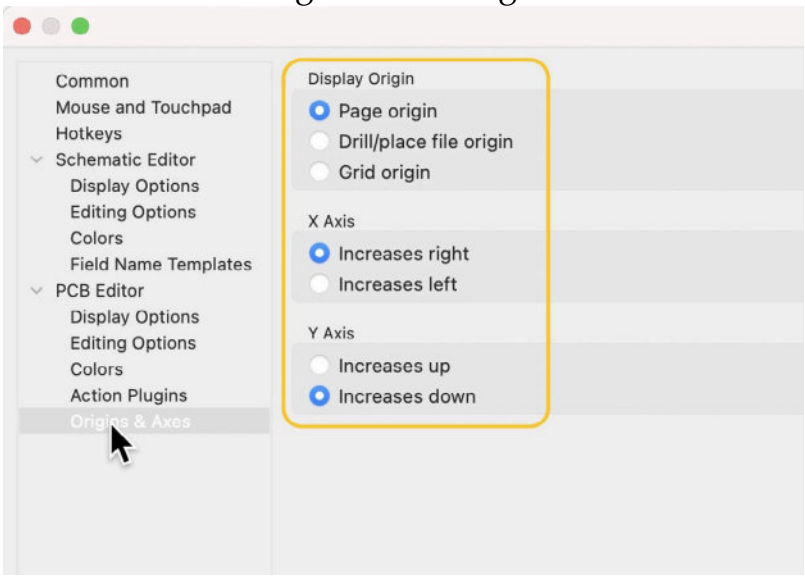


Figure 13.24.3: The default settings for the Origins & Axes.

The default setting, as you see them in the screenshot above, produces a coordinate system with its origin at the top left of the editor. The page origin, in Pcbnew, is always at the top left of the editor. To change it to the bottom left, you must first place a grid origin marker at the location where you want the new coordinate system to originate and then change the display origin to use the new grid origin.

In Pcbnew, click on the grid origin button from the bottom of the right toolbar:

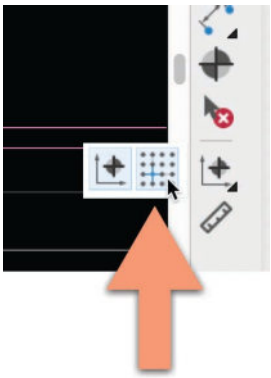


Figure 13.24.4: The Grid Origin button.

Then, create the new origin by clicking at the bottom left corner of the editor:

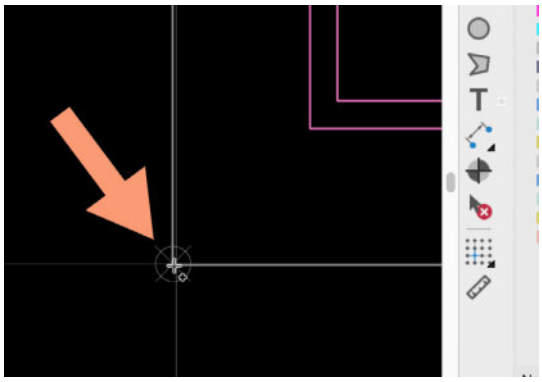


Figure 13.24.5: A new grid origin at the bottom left corner of the editor page.

You now have a new origin. It is not yet enabled, so Pcbnew is still using the default page origin. To switch the coordinate system to the new grid origin, open the KiCad Preferences, and click on Origins & Axes. Change the Display Origin to “Grid origin” and the Y-Axis to “Increases up” (Figure 13.24.3):

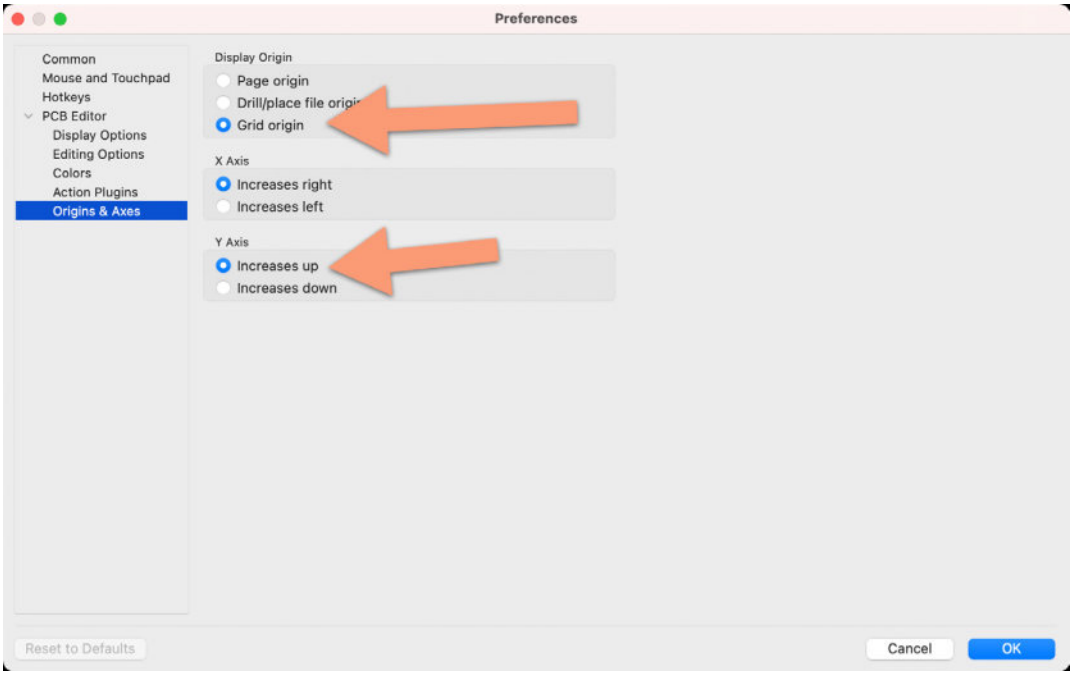


Figure 13.24.3: These settings for the X and Y axes change the coordinate origins to the bottom left of the editor.

Click OK, and return to Pcbnew. Inspect the coordinate system by placing your mouse pointer in the new origin (bottom left of the editor page). In the status bar, the X and Y value should be zero. Below I show the X and Y values for three positions using the new coordinate system.

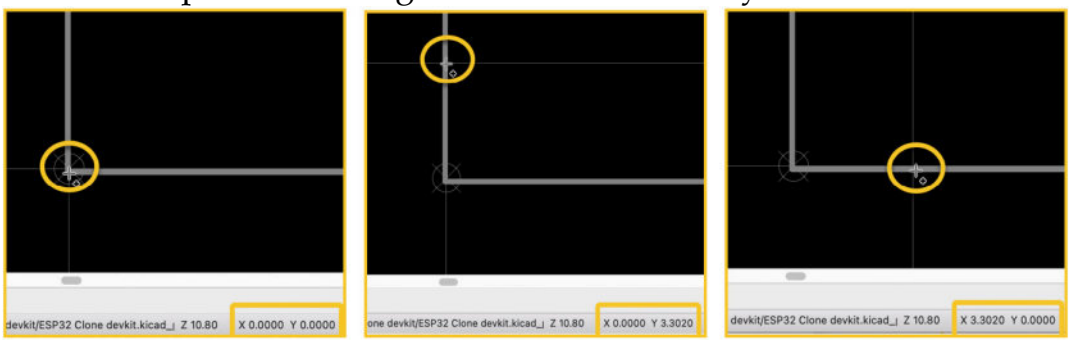


Figure 13.24.6: The coordinates of three positions in the new coordinate system.

Grid origin anywhere

In Pcbnew, you can place the grid origin anywhere, not just on a corner of the editor page. For example, you can place the grid origin marker in the bottom left corner of a PCB, like this:

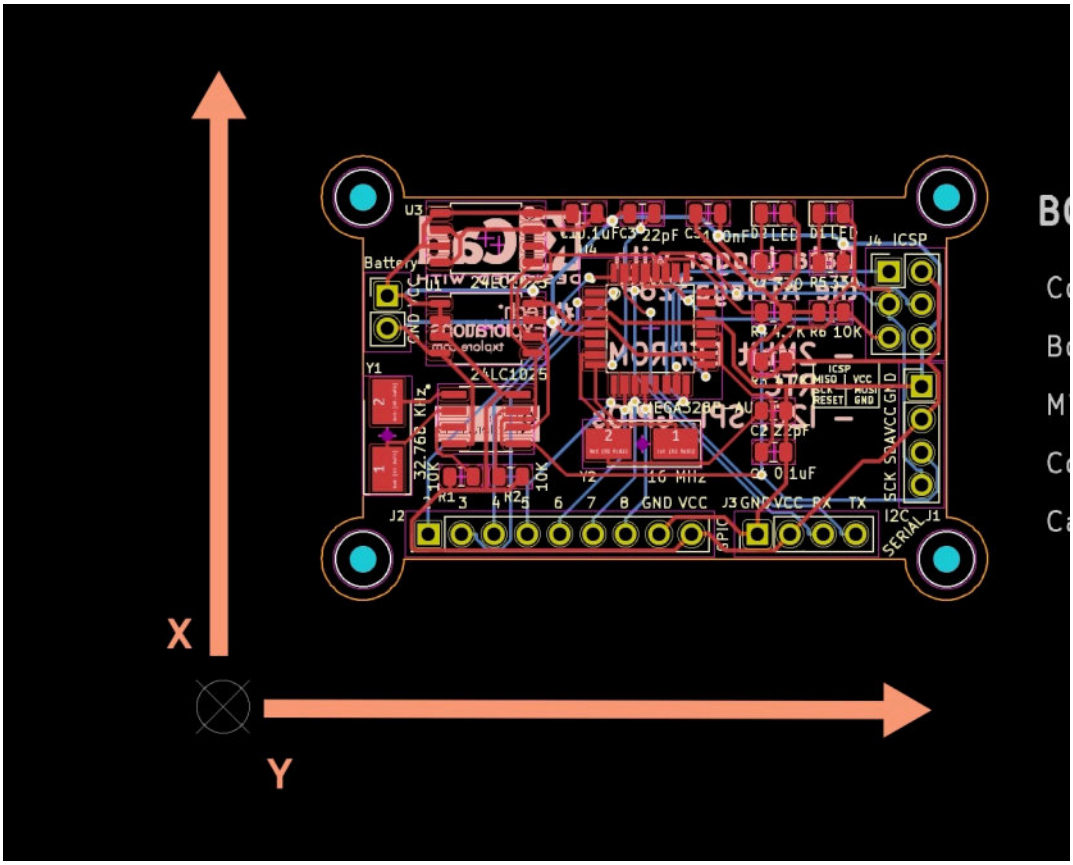


Figure 13.24.7: The grid origin can be anywhere.

I have retained the settings, as seen in Figure 13.24.7. The coordinates now are all in reference to the new origin.

You can even place the grid origin inside a PCB. This will result in Y coordinates on the left side of the central X-axis being negative and positive on the right side.

Drill/place file origin

Pcbnew offers another coordinate system that you can use, the “Drill/place file origin.” It works in the same way as the grid origin.

Click on the ”Drill/place file origin” button from the right toolbar:

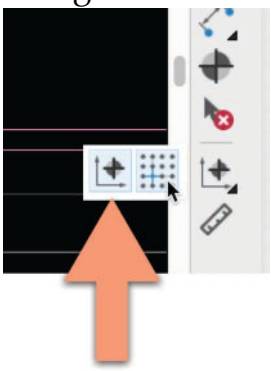


Figure 13.24.8: The ”Drill/place file origin” button.

Then, click anywhere in the Pcbnew editor where you want to place the new origin. In the example below, I'm setting it near an existing drill origin marker:

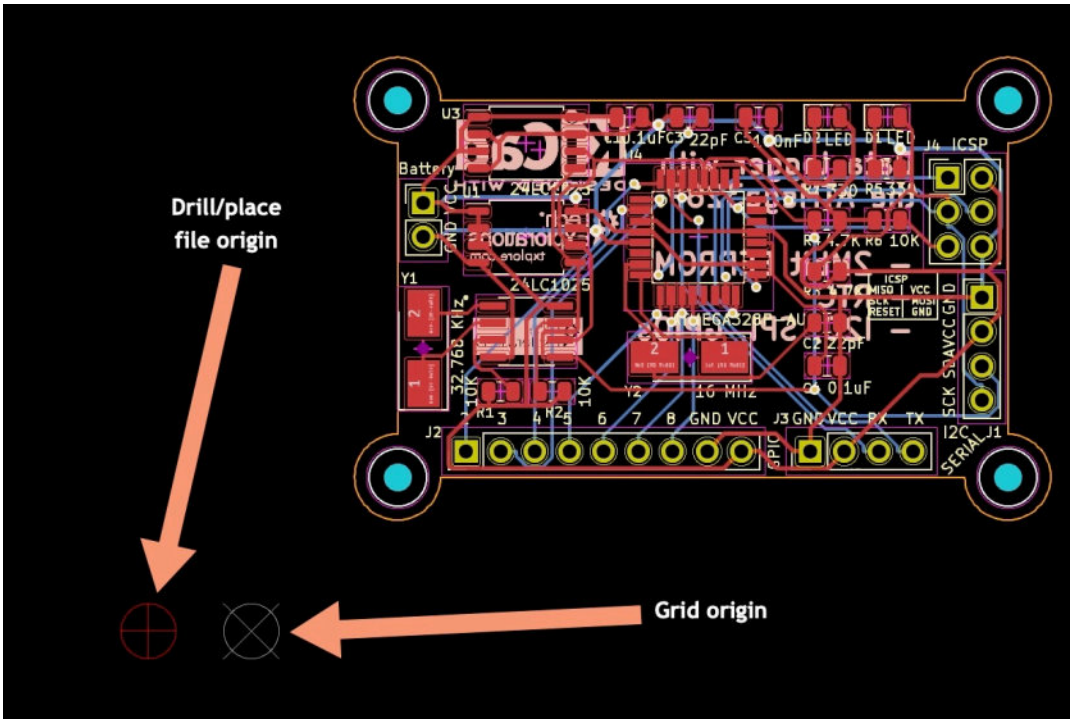


Figure 13.24.9: The "Drill/place file origin" marker next to the Grid origin marker.

Before the new origin is active, you must select it from the Preferences window. Bring up the KiCad Preference window, and click on Origins & Axes. Change the Display Origin to "Drill/place file origin," and click OK.

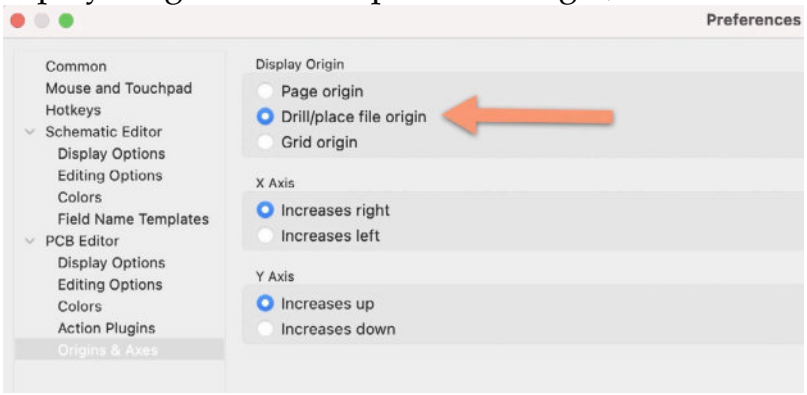


Figure 13.24.10: The "Drill/place file origin" selected.

With this, the new origin is on the red marker. My mouse pointer is on the red marker, and the X and Y coordinate values confirm this:

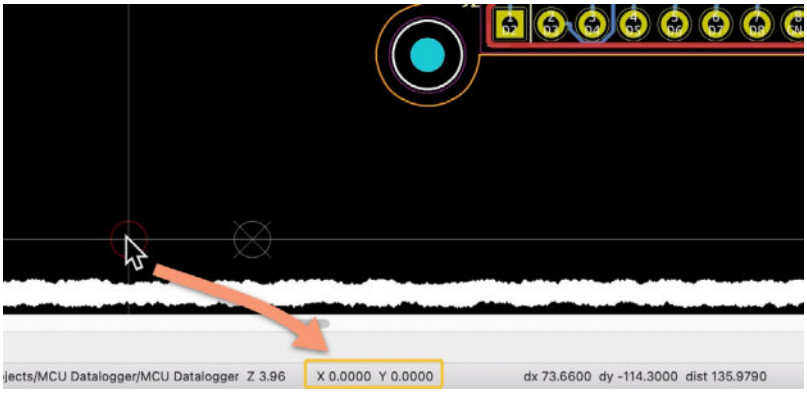


Figure 13.24.11: Testing the "Drill/place file origin" coordinate system.

When to use an alternate coordinate system?

In my experience, I have not needed to change the coordinate system in Pcbnew away from the default Page origin. This is because all the PCB manufacturers that I have worked with have been able to use the default coordinate system, and because I have not had a need to work with other CAD applications.

If you encounter one or both of these cases, Pcbnew provides you with a full set of alternative coordinate systems that you can try.

25. KiCad project management with Git

In this chapter, you will learn how to set up the Git version control system on your computer and use it to manage your KiCad projects.

Git is a free and open-source version control system. With Git, you can create repositories of any project that contains text files (and, with some limitations, binary files). A Git repository provides you with powerful capabilities that are useful to solo developers and teams.

Once you commit your project to a Git repository, you will record a complete history of its development and explore new features without risking the work you have already done. You can also share your work with others and incorporate changes made by team members. You can quickly return to any point in your project's history (no need to keep typing Ctr-Z). You can use tags to mark important milestones in your project with a name that is easy to remember.

Once you become familiar with Git and a simple Git + KiCad workflow, you will find it too risky to work without Git.

Why would you want to use Git with KiCad?

A KiCad consists of several text files. The most important of those files are:

- The project ".kicad_pro" file.
- The layout ".kicad_pcb" file.
- The schematic ".kicad_sch" file.

All of these files contain simple text. Here's an example of each (only showing a few lines for this example):

.kicad_pro:

```
{
  "board": {
    "design_settings": {
      "defaults": {
        "board_outline_line_width": 0.09999999999999999
```

```
            "copper_line_width": 0.19999999999999998,
            "copper_text_italic": false,
            "copper_text_size_h": 1.5,
            "copper_text_size_v": 1.5,
            "copper_text_thickness": 0.3,
            "copper_text_upright": false,
            "courtyard_line_width": 0.049999999999999996,
```

.....

.kicad_pcb:

```
(kicad_pcb (version 20210623) (generator pcbnew)

  (general
    (thickness 1.6)
  )

  (paper "A4")
  (layers
    (0 "F.Cu" mixed)
    (31 "B.Cu" mixed)
    (32 "B.Adhes" user "B.Adhesive")
    (33 "F.Adhes" user "F.Adhesive")
    (34 "B.Paste" user)
    (35 "F.Paste" user)
    (36 "B.SilkS" user "B.Silkscreen")
    (37 "F.SilkS" user "F.Silkscreen")
....
```

.kicad_sch:

```
(kicad_sch (version 20210621) (generator eeschema)

  (uuid 6b311d3d-9ae1-4ec9-af71-d4c3a43f0d08)

  (paper "A4")

  (title_block
    (title "A 4x8x8 LED Matrix Display Clock")
    (date "2021-07-14")
    (rev "${design_version}")
  )

  (lib_symbols
```

```
    (symbol
"ArduinoProMiniSimple:ArduinoProMiniSimple" (in_bom yes)
(on_board yes)
…..
```

KiCad's projects are composed of text files, making it easy to use a versioning system like Git to 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, widely used, and highly versatile. As you will see in this recipe, it is also straightforward 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 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 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.

Using Git alongside an online repository, like Github, you will 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 13.25.1, you can see part of the most recent history of one of the projects in this book, as it appears in the publicly accessible repository on Github.

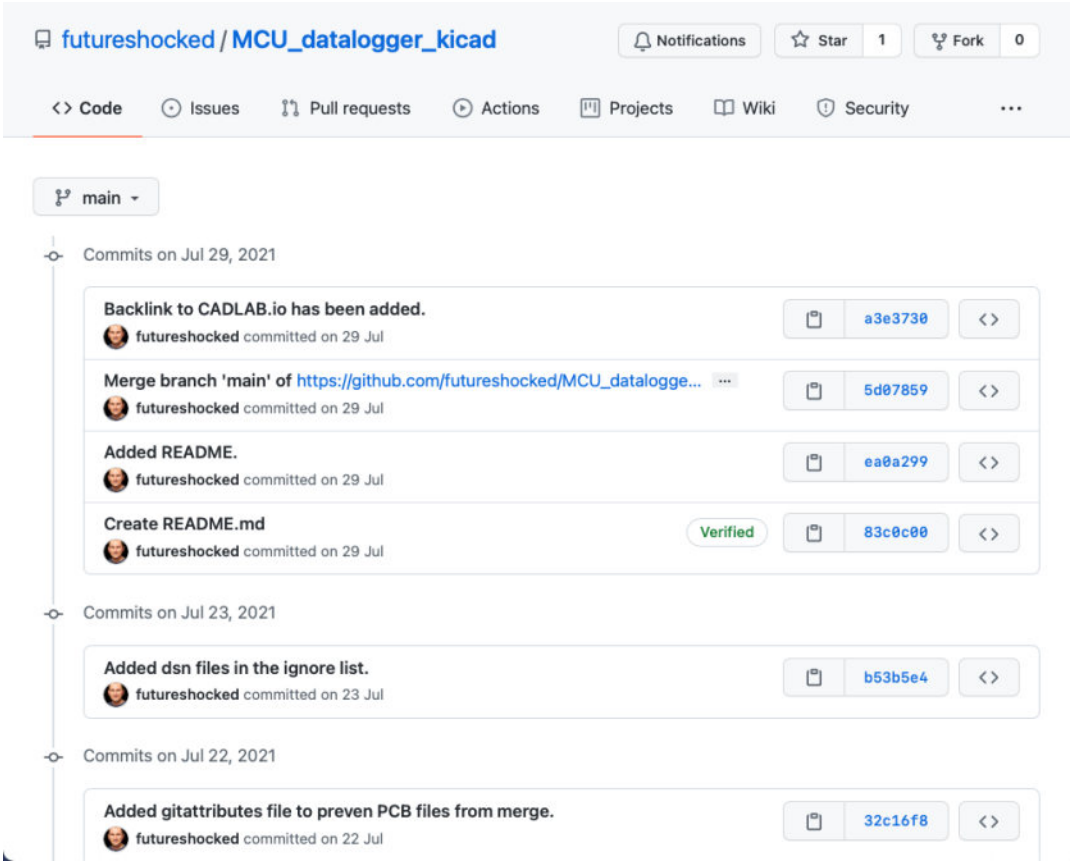


Figure 13.25.1: One of my KiCad projects on Github, showing the recent project history.

In Figure 13.25.2, you can see the history of the schematic file for the same project.

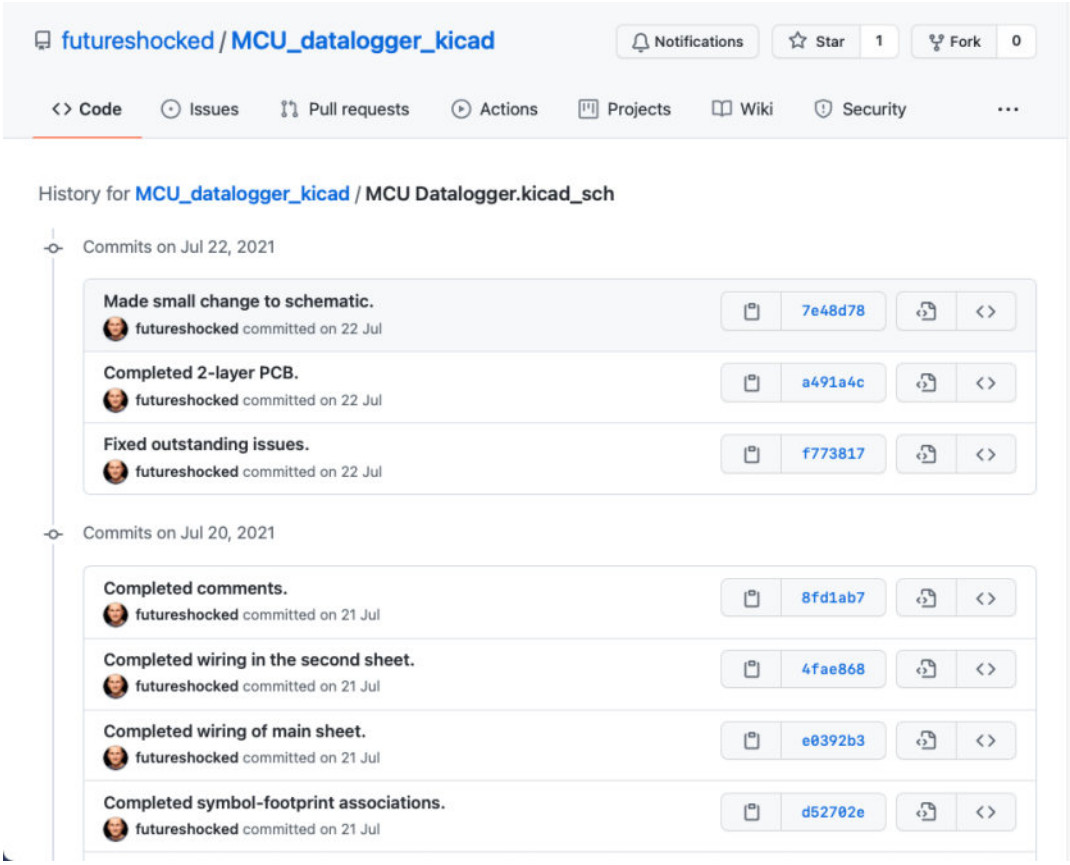


Figure 13.25.2: Part of the history of the schematic file on Github.

The numbers on 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 '7e48d7843cbe041f04e2126f5f75c9c94cafce23'. What you see in Figure 13.25.2 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 13.25.3.

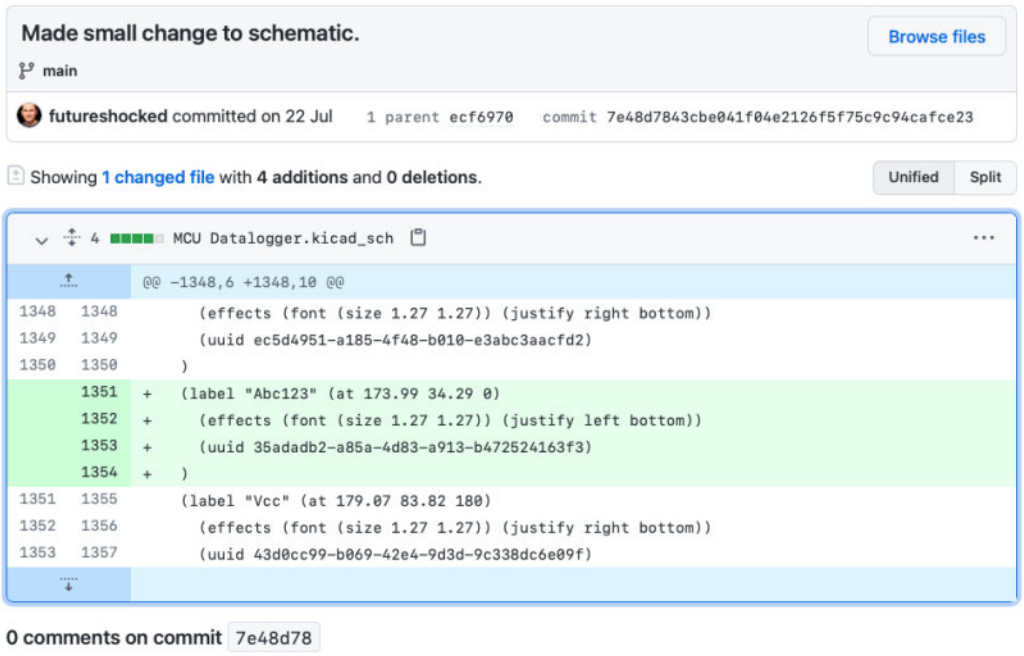


Figure 13.25.3: The changes in this commit of file "MCU Datalogger.kicad_sch".

In this chapter, I will give you a brief introduction to Git and Github. Git is a big topic, and I encourage you to learn more about it by using a specialized source like this Getting Started guide from Github. You may also consider our comprehensive course on the Tech Explorations website.

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

- Install and configure Git on your computer.
- Create a new Git repository for your KiCad project.
- How to exclude files that you do not want to track in Git.
- Commit changes to your project to the repository.
- See those changes in the log.
- Checkout past commits.
- Create branches.
- How to merge branches.
- How to prevent the merging of specific files from different branches.
- Upload your project to Github so you can share it with other people.

25.1. Install Git

Git has installers for MacOS, Windows, and Linux.

You can download the installed for your operating system at https://git-scm.com/downloads (Figure 13.25.1.4).

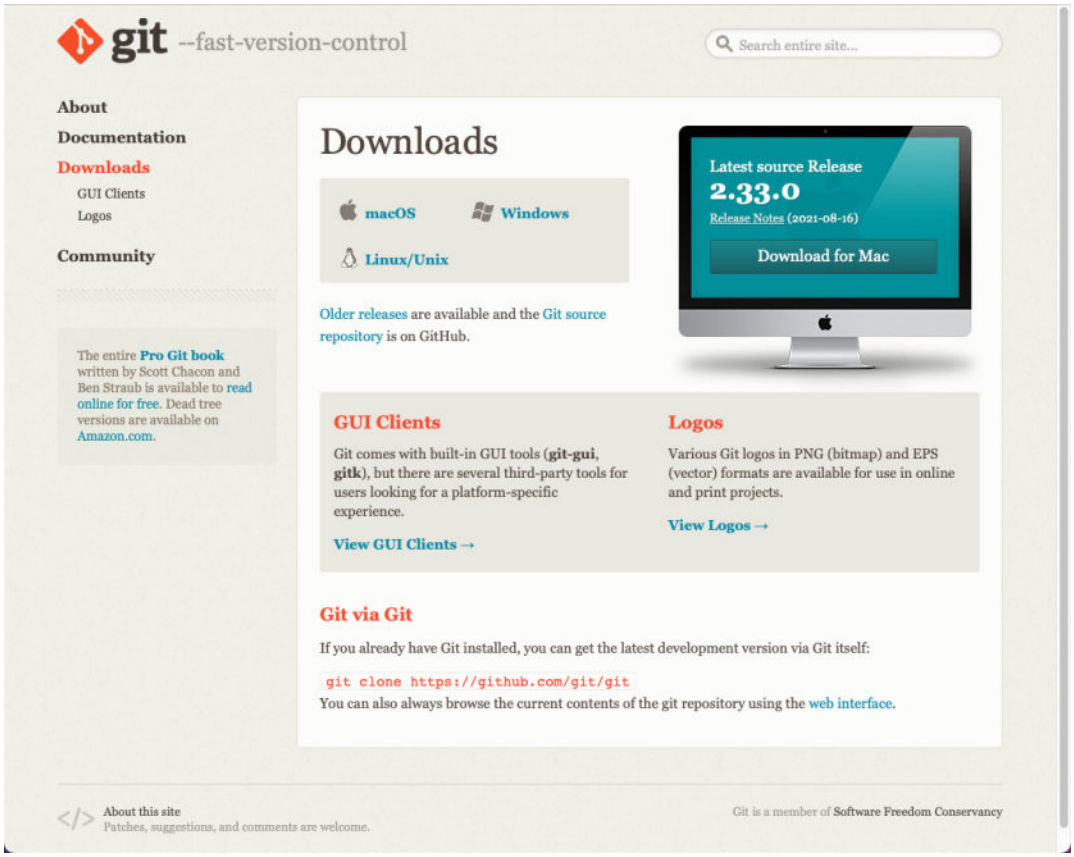


Figure 13.25.1.4: The download page for the Git installer.

The installation process is easy. I suggest you use the binary installer for your OS instead of trying to compile from the source.

You can also find clear installation instructions and the binary installer downloads:

- MacOS.
- Windows.
- Linux.

Go ahead and install Git on your computer. After installing Git, continue to the next segment, showing you how to configure it.

25.2. Git configuration

Now that you have installed Git on your computer, you need to do some simple configurations before using it. Git works best when it knows who you are (i.e., your user name) and your email address. It will use this information to attribute operations such as commits and deletes to the user that has performed them.

Bring up a terminal window.

At the command prompt, configure your git user settings:

```
$ git config --global user.name 'testuser'
$ git config --global user.email 'testuser@example.com'
$ git config --global init.defaultBranch main
$ git config --global credential.helper “cache --
timeout=86400”
```

Below you can see my configuration session on a terminal window in MacOS (Figure 13.25.2.5):

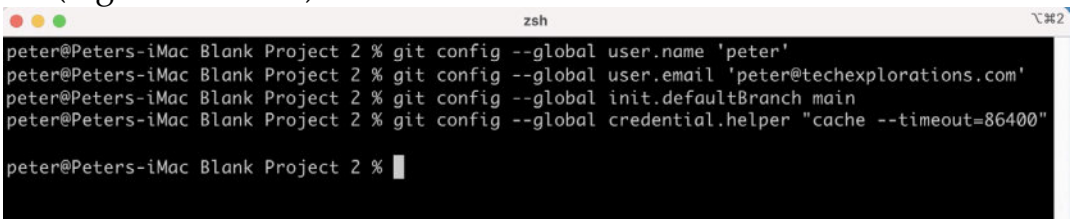


Figure 13.25.2.5: My Git configuration session.

By default, the main branch of Git is called “master.” In recent years there has been a change in the traditional terms “master” and “slave” across computer science disciplines and systems. Github, as an example, has changed its default Git branch from “master” to “main.” This is why in my Git configuration session above, I have used the “init.defaultBranch” option to set “main” as the name of the main branch. This way, the main branch name on my computer will be the same as on Github.

The last configuration where I use the “credential.helper” option sets Git to cache my Github credentials so that I don’t have to authenticate each time I interact with Github on the command line.

You can find details about all the available configuration options in the documentation.

That’s it; the configuration is complete. Let’s continue to the next segment, where we’ll create a new Git repository for a simple demonstration KiCad project.

25.3. Create a new KiCad project Git repository

Git is now ready to help manage a new (or existing) KiCad project. . I’ll show you how to create a new repository, do the first few commits, and check the repository status in this segment.

To demonstrate, I will use a small KiCad demonstration project from earlier in this book. The project contains a small number of files, as you can see in Figure 13.25.3.6:

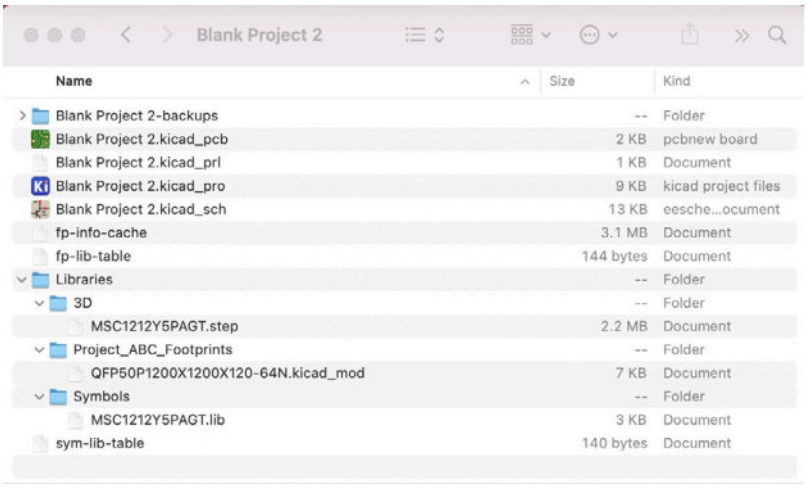


Figure 13.25.3.6: I'll create a Git repository on this KiCad project.

This project has a schematic that contains a few resistors and capacitors. It is very simple, and for this reason appropriate to demonstrate how to use KiCad to track changes between commits (Figure 13.25.3.7).

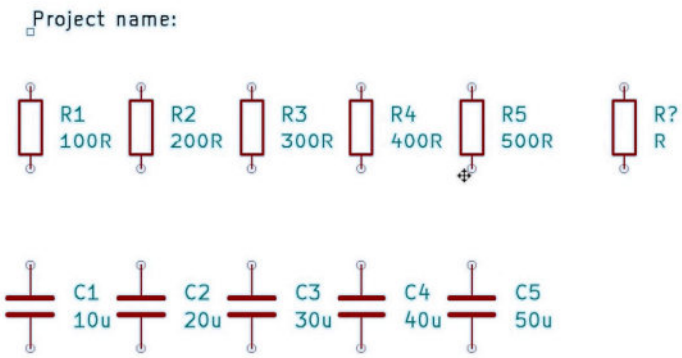


Figure 13.25.3.7: The project schematic before the first commit.

The layout editor is blank.

Start by bringing up the terminal. To confirm that there is no Git repository setup yet, use the "status" command (Figure 13.25.3.8):

```
% git status
Fatal: not a git repository (or any of the parent
directories): .git
```

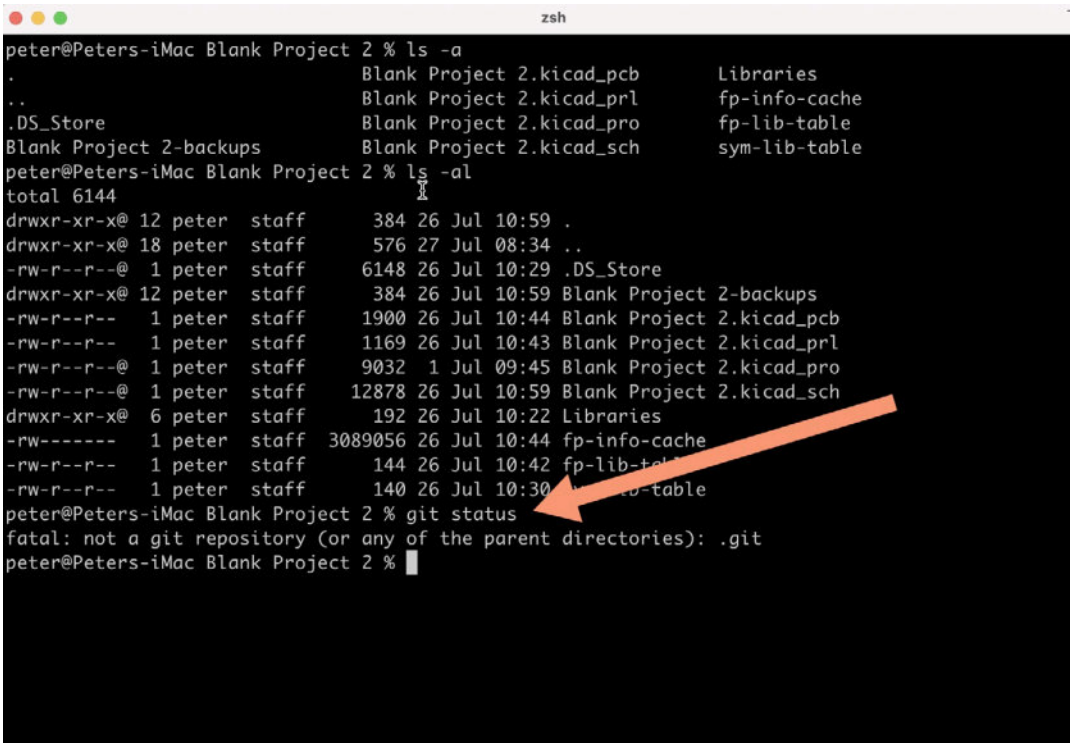


Figure 13.25.3.8: The "git status" command and its output.

The "status" command complained about the absence of the ".git" repository. Git keeps all repository data and configuration inside a directory named ".git," and the absence of this directory indicates that there is no Git repository present (Figure 13.25.3.9).

Let's create a new repository. The command is "init":

```
% git init
Initialized empty Git repository in /Users/peter/Documents/Kicad/
Course development documents/KiCad 6 test projects/Blank Project
2/.git/
%
```

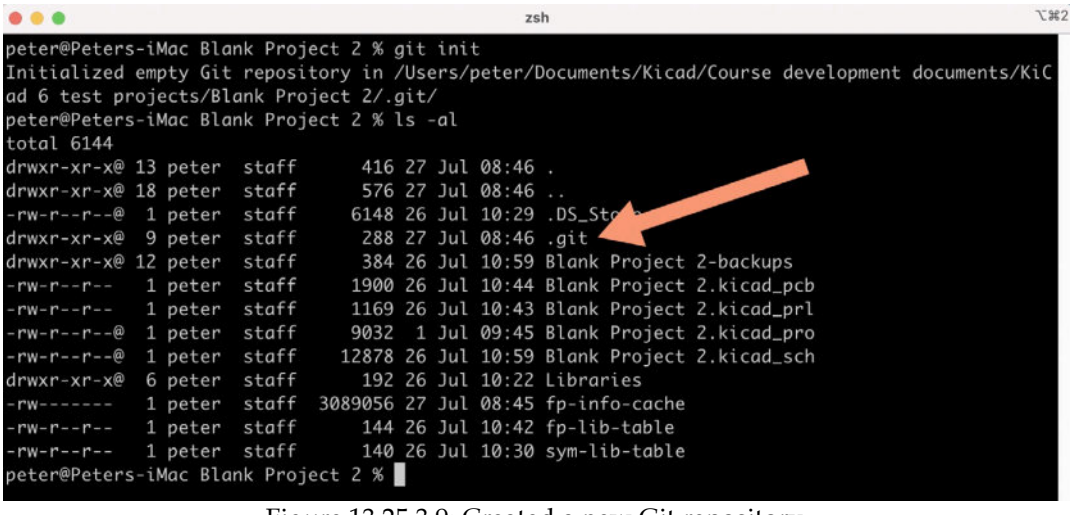


Figure 13.25.3.9: Created a new Git repository.

In the screenshot above, you can see that the ".git" is now present. This means that the KiCad project directory now contains a Git repository. The repository is empty, as you can confirm by using the "status" command once again (Figure 13.25.3.10):

```
% git status
On branch master

No commits yet

Untracked files:
    (use "git add <file>..." to include in what will be committed)
        .DS_Store
        Blank Project Z-backups/
        Blank Project Z.kicad_pcb
        Blank Project Z.kicad_prl
        Blank Project Z.kicad_pro
        Blank Project Z.kicad_sch
        Libraries/
        fp-info-cache
        fp-lib-table
        sym-lib-table E

nothing added to commit but untracked files present (use "git add"
to track)
%
```

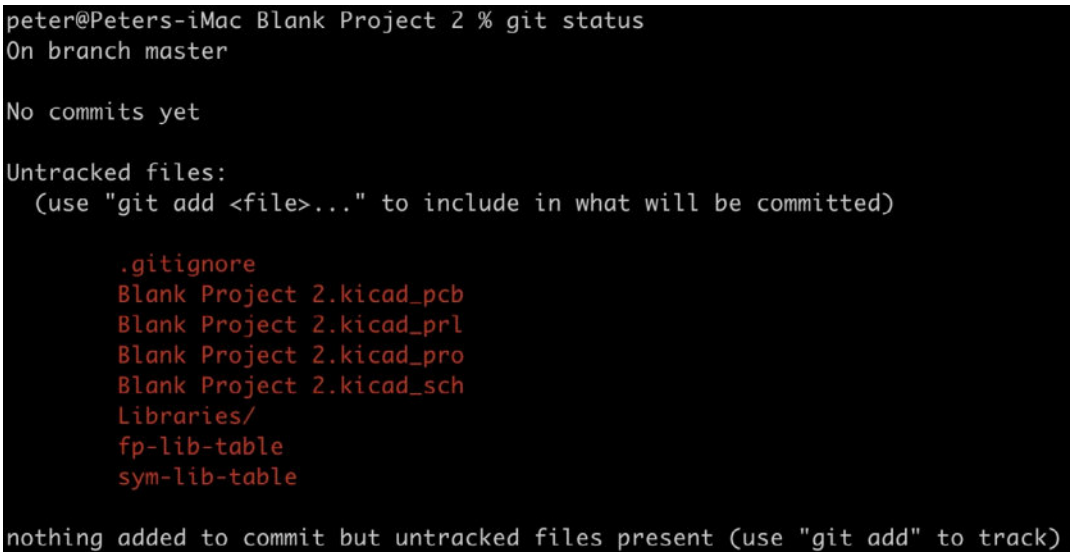


Figure 13.25.3.10: No commits, several un-tracked files.

The response from the "status" command is that there are no commits to the repository and several files in the working directory that are not being tracked. Git even suggests that I use the "add" command to start tracking these files.

I'll show you how to do this soon, but first, I want to show you how to not track (or ignore") files that are not necessary. Ignoring such files keeps the repository tidy. An example of such file is ".DS_Store", which is created by MacOS and is not a project file. Ignoring this file is advisable. I'll show you how in the next segment.

25.4. How to ignore files

Not all files that are present in a project repository should be tracked. In the previous segment, I created a new Git repository for my simple KiCad project. One of the files in the project repository is ".DS_Store", which is a MacOS system file. I want to exclude this file since it is not relevant to the project. It will also benefit my project collaborators who might be working on Linux or Windows computers.

To instruct Git to ignore one or more files, or even a full directory, create a new file named ".gitignore". Use any text editor for this. Below you can see the contents of my ".gitignore" file:

```
*.xml
*.dsn
.DS_Store
Fp-info-cache
*Gerber/*
*backups/*
```

In my .gitignore file, I am exlcuding several other files and directories in addition to the ".DS_Store" file. I use wild-card characters like "*" to match multiple files. For example, I am ignoring all files with extensions ".xml" and ".dsn", and all files insider any directory that contains the string "Gerber".

You can learn more about the syntax used in gitignore files in the Git documentation.

Save this file, and the try the "status" command once again. Here is the new output:

```
% git status
On branch master

No commits yet

Untracked files:
    (use "git add <file>..." to include in what will be committed)
        .gitignore
        Blank Project Z.kicad_pcb
        Blank Project Z.kicad_prl
```

```
        Blank Project 2.kicad_pro
        Blank Project Z.kicad_sch
        Libraries/
        Fp-lib-table
        sym-lib-table
nothing added to commit but untracked files present (use "git add"
to track)
%
```

As you can see, the ".DS_Store" file is not included in the listing of untracked files. It is now ignored.

It is time to do the first commit, and I'll show you how in the next segment.

25.5. Basic Git commands: add, commit

I have configured Git, created a new repository, and set the gitignore file. It is time to do the first commit for this demo KiCad project.

Remember that the last time I issued the "status" command, Git indicated that several files are untracked. Git will only commit tracked files. Therefore, before I commit, I must instruct Git to track one or more files.

The command for this is "add", as in "add a file(s) to the repository and begin tracking them." For the first commit, I can use the "." operator with the "add" command so that Git will start tracking all non-ignored files. Here is my session:

```
% git add .
%
```

That's it. I'll use "status" once again to see the changes (Figure 13.25.5.11):

```
% git add .
% git status
On branch master

No commits yet

Changes to be committed:
    (use "git rm --cached <File>..." to unstage)
        new File: .gitignore
        new File: Blank Project 2 kicad_pcb
        new File: Blank Project Z.kicad_prl
        new File: Blank Project 2 kicad_pro
        new File: Blank Project Z.kicad_sch
        new File: Libraries/3D/MSC121ZY5PAGT step
```

```
        new File: Libraries/Project_ABC_Footprints/
QFP50P1200X1208X120-64N.kicad_mod
        new File: Libraries/Symbols/MSClZlZYSPAGT.lib
        new File: Fp-lib-table
        new File: sym-lib-table
```

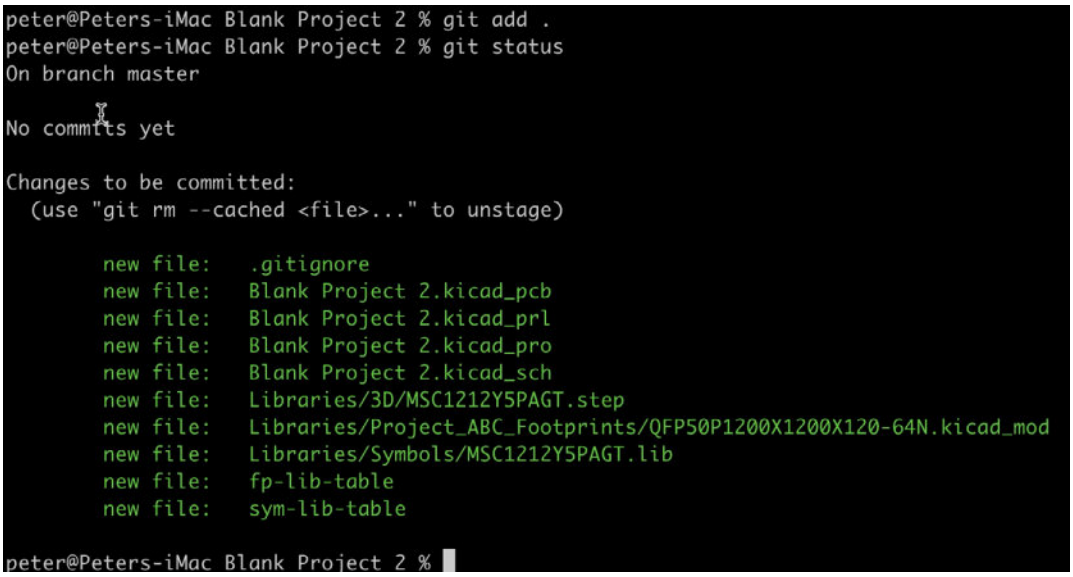


Figure 13.25.5.11: No commits, several tracked files.

The status command reports that several files are now being tracked but not committed. Notice that Git shows the tracked files in green. In Figure 13.25.5.10, untracked files are in red.

Files that are tracked but not committed can change, but Git will not capture the changes in the repository until the files are committed.

To commit a file (or files) to the repository, use the "commit" command. This command can receive various modifiers and parameters. In most cases, you will want to use the "-am" flags. With "a," you tell Git to commit all files that are waiting in the index (that is, being tracked). With "m," you can include a commit message in the same command (otherwise, Git will open a text editor and ask for one).

Here is the first commit command for this repository:

```
% git commit -am "First commit."
[master (root-commit) 16a0e89] First commit.
 10 Files changed, 30490 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Blank Project 2.kicad_pcb
 create mode 100644 Blank Project Z.kicad_prl
 create mode 100644 Blank Project Z.kicad_pro
 create mode 100644 Blank Project Z.kicad_sch
 create mode 100644 Libraries/3D/MSC1212Y5PAGT.step
```

```
 create mode 100644 Libraries/Project_ABC_Footprints/
QFPSOP1200X1Z00X1Z0-64N.kicad_mod
 create mode 100644 Libraries/Symbols/MSC1212Y5PAGT.lib
 create mode 100644 Fp-lib-table
 create mode 100644 sym-lib-table
%
```

The commit command response confirms that ten files were committed into the master branch, and the hash of the commit ID is 16a0e89.

I'll do another "git status" to see the current status of my repository:

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

My repository tree is clean, and there are no changes to commit.

One thing that bothers me is that the name of the only branch that exists in my new repository is "master." As I mentioned earlier in this chapter, there is a recent trend to use "main" instead of "master" in IT and computer science. So, before I continue, I will rename my existing "master" branch into "main."

The command for this is "branch". If you issue "git branch" without any parameters, Git will give you a listing of branches in the repository. Then, you can use the "-m" switch to rename a branch.

Here are the relevant commands for the above:

```
% git branch

* master

% git branch -m master main
% git branch

* main

%
```

Another command that I use frequently is "log." With the log command, you can see the recent history of your project. Try it now:

```
% git log
commit 16a0e89fc22f74dad95F4f20eccdee81e583e084 (m -> main)
Author: peter <peter@txplore.com>
Date: Tue Jul 27 08:53:18 2021 +1000
First commit.
```

```
%
```

The log command responds with a list of commits, their IDs, author and date information, plus the commit message. I have only done a single commit in this example, so the response contains a single record.

Let's make a small change to the schematic to learn how to deal with change. Open the schematic, and add a new resistor (Figure 13.25.5.12):

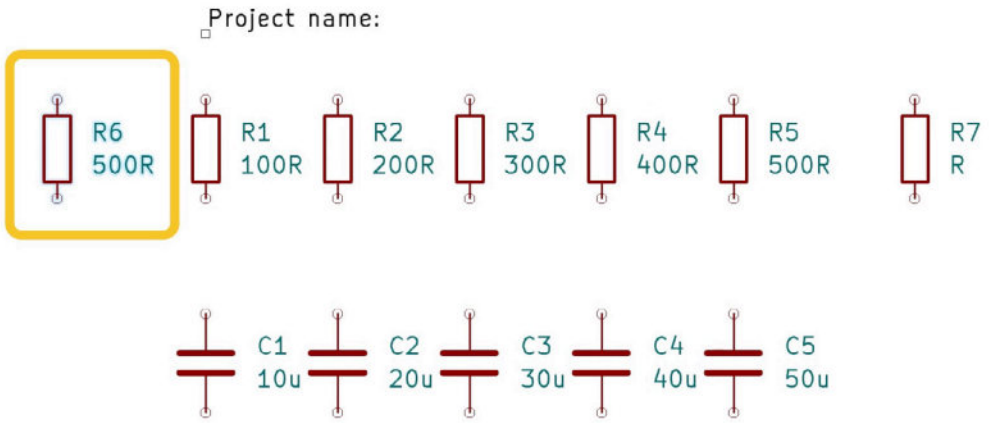


Figure 13.25.5.12: I have added a new resistor (R5) to the schematic.

Save the schematic file. Back in the command line, use the "status" command to see if Git has detected the change:

```
% git status
On branch main
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: Blank Project Z.kicad_sch
no changes added to commit (use "git add" and/or "git commit -a")
%
```

Yes, of course, it did. Git has detected a change in "Blank Project Z.kicad_sch."

You can see the exact changes with the "git diff" command (Figure 13.25.5.13):

```
% git diff
```

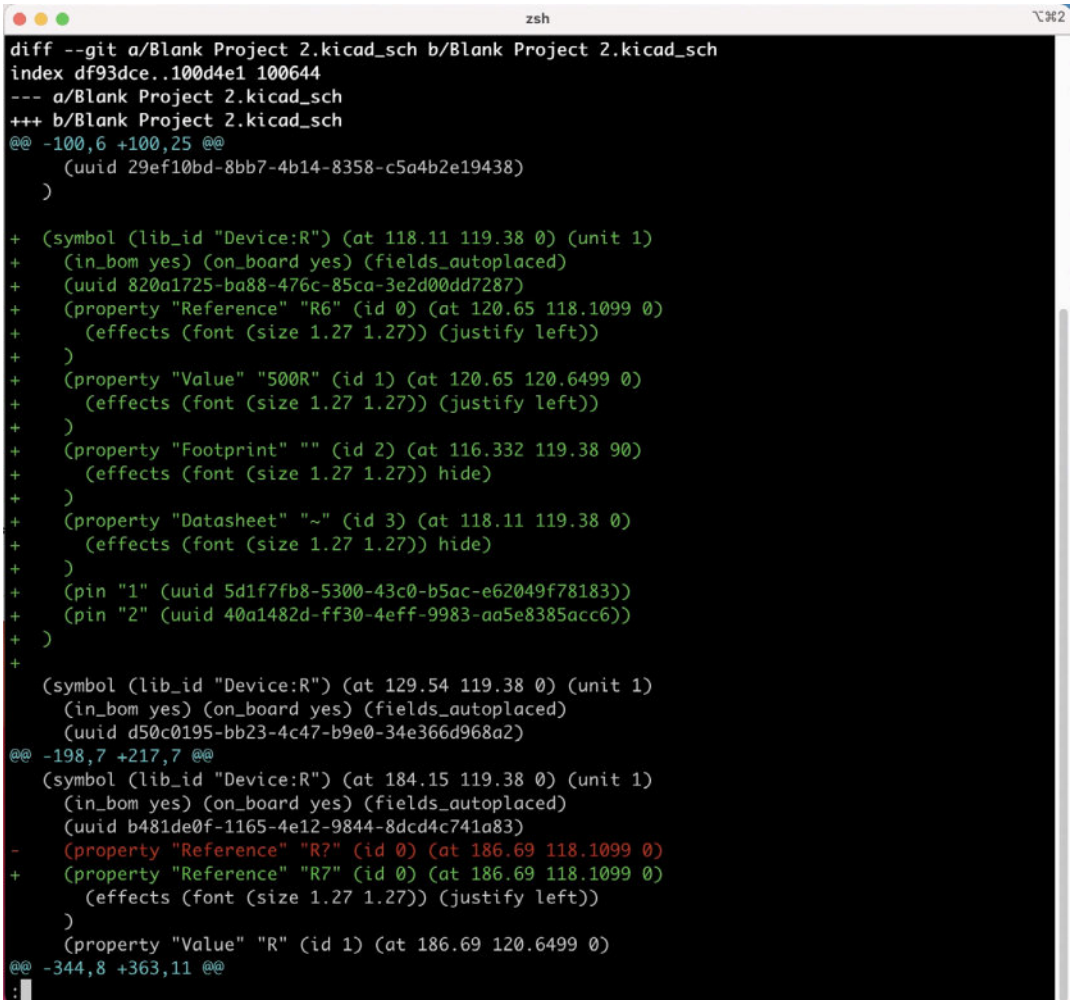


Figure 13.25.5.13: “Git diff“ shows the current changes.

The “diff” command will show the new content in green and removed content in red.

I will proceed with the new commit:

```
% git commit –am "Added a resistor."
[main 7e4bf0e] Added a resistor.
 1 file changed, 24 insertions(+), Z deletions(–)
peter@Peters–iMac Blank Project 2 % git status
On branch main
nothing to commit, working tree clean
%
```

The hash from the new commit ID is 7e4bf0e. You can use this hash later to checkout a specific commit. Your repository now contains two commits.

Imagine that you would like to experiment with your design. It may not work out, and you may need to retract to the current state of the project. To do this safely, you can create a new Git branch and do the necessary work there. In the next segment, I’ll show you how to work with branches.

25.6. Basic Git commands: branch

In Git convention, the "main" or "master" branch contains the "official" state of the project. This may be the project instance you want to share with others, publish on your website, compile for production, or send to the PCB manufacturer. Work that involves, for example, adding new features, experimenting, or testing, is typically done in dedicated branches.

For example, in my KiCad projects, I often maintain a main branch with the PCB that I am comfortable sharing with students and other branches to experiment with different layout configurations or components. In one of the projects in this course, I maintain separate branches for the 2-layer and 4-layer versions of the same board.

In this segment of the chapter, I will show you how to work with branches.

To create a new branch, use the "checkout" command with the "-b" switch and a name to create a new branch. The "-b" switch will cause the creation of a new branch. If you use "checkout" without "-b," Git will attempt to checkout (i.e., make active) the specified branch.

Here is the command and the response:

```
% git checkout -b experimental
Switched to a new branch 'experimental'
%
```

You can verify that the new branch exists using the "branch" command:

```
% git branch
* experimental
  main
%
```

In the output above, notice the "*" in front of the "experimental" branch? This indicates that the starred branch is active, and any new commits will go into this branch.

The new branch has inherited all content from the previously active branch, "main." You can look at the content of the schematic editor, and you will see that it is identical to its content under the main branch.

I will introduce a change: a new capacitor, “C6”. The new schematic looks like this (Figure 13.25.6.14):

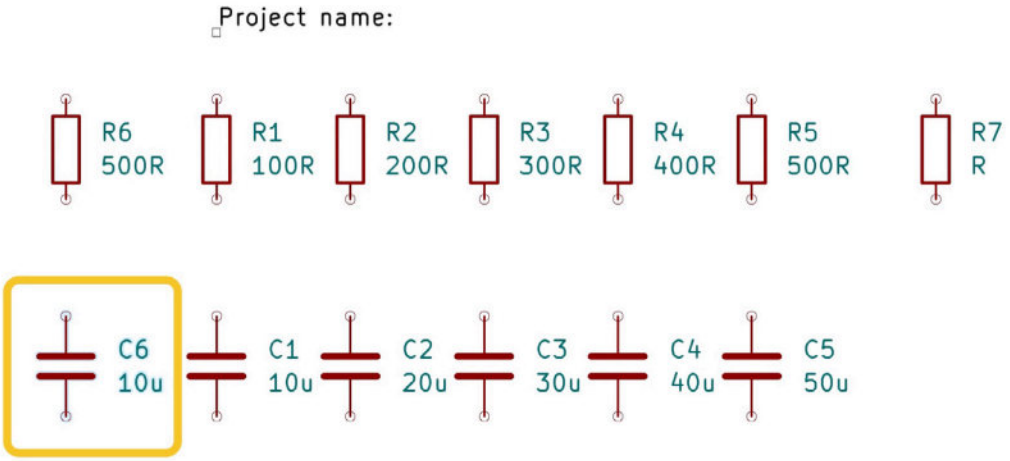


Figure 13.25.6.14: While working in the “experimental” branch, I changed the schematic.

Save the schematic editor, and return to the terminal. Get the new status of the project:

```
% git status
On branch experimental
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: Blank Project 2.kicad_sch

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

Git responds with the name of the current branch (“experimental”) and the file's name where the change is detected. Let’s commit this change with the “commit” command:

```
% git commit -am "Added capacitor."
[experimental d389l40] Added capacitor.

1 File changed, 22 insertions(+)
%
```

Git confirms that it has committed one new file to the "experimental" branch. The commit contained 22 new lines, and the first seven characters of the commit ID are d389l40.

Also, try a "git log" (Figure 13.25.6.15):

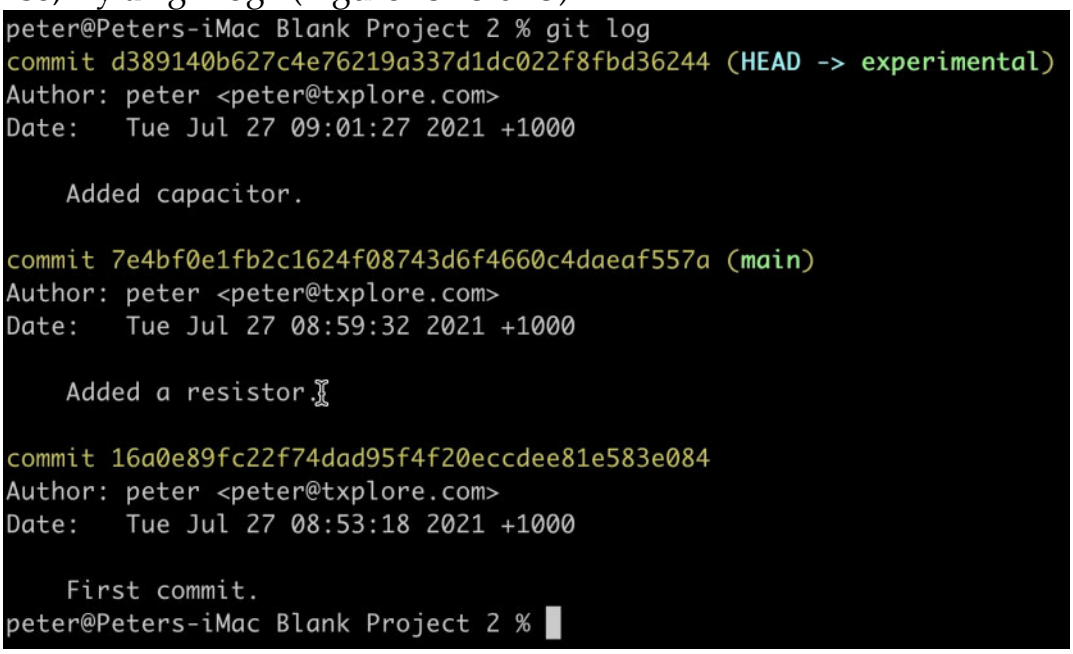


Figure 13.25.6.15: Git log showing the recent repository history.

The "git log" reports the full commit IDs. Notice the complete commit ID of the latest commit (the first in the list) and how the first seven characters of the ID match those reported by the commit command ("d389l40").

Say that you would like to go back to the main branch. You can checkout to the latest commit of the main branch without this affecting any work you have done in the experimental branch. To checkout to the HEAD (i.e., the newest commit) of a branch, use the "checkout" command:

```
% git checkout main
Switched to branch 'main'
```

Now, use "git log" to see the status of the repository in the main branch:

```
% git log
commit 7e4bf0e1bec1624f08743d6f4660c4daedfSS7d (HEAD -> main)
Author: peter <peter@txplore.com>
Date: Tue Jul 27 08:59:32 2021 +1000

    Added a resistor.

commit 16o0e89fc22F74dad95f4f20eccdee8le583e084
Author: peter <peter@txplore.com>
```

```
Date: Tue Jul 27 08:53:18 2021 +1000

    First commit.

%
```

As you can see, in the main branch, the new capacitor does not exist. I created the capacitor while working in the experimental branch, and I committed this change there.

Now, assume that you are happy with the change in the experimental branch, and you would like to "import" this change into the main working branch. To do this, you will use the "merge" command. I'll show you how in the next segment of this chapter.

25.7. Basic Git commands: merge

In the previous segment of this chapter, you created an experimental branch and made a change to it. Now, you want to import this change to the main working branch. The git term for this operation is "merge." Effectively, you want to merge one branch into another. To do this, you will use the "merge" command.

Start by confirming that you are currently working in the main branch (which is where you want to merge into):

```
$ git branch
  experimental
* main
```

The star next to "main" indicates that you are working in the main branch.

Do the merge:

```
% git merge experimental
Updating 7e4bf0e..d389140
Fast-forward
Blank Project Z.kicad_sch | 22 ++++++++++++++++++++++
1 file changed, 22 insertions(+)
%
```

The merge response provides information about what just happened. One file in the main branch was changed, and 22 new lines were inserted. The file that was changed is “Blank Project Z.kicad_sch.”

The “merge” command is very flexible. You can use it to merge not only a branch into another branch but also a specific commit into a branch, using the commit ID. In most cases, using branch names, as in the example above, is sufficient.

The line starting with “Updating” tells you that changes from commit ID “d389140” were merged into commit ID “7e4bf0e”.

Note that the merge operation does not overwrite contents. It merges contents. There is a lot of intelligence embedded into Git to ensure that work is not lost. If Git cannot figure out how to do a merge, it will ask you to clarify. Merge conflicts happen when, for example, the same lines in a file are changed in both the source and the destination conflict. In such cases, Git will not change anything and ask you to resolve a conflict manually.

With the merge completed, let's verify that the capacitor I added in the experimental branch now exists in the main branch. Open Eeschema, and look at the current state of the schematic (Figure 13.25.7.16):

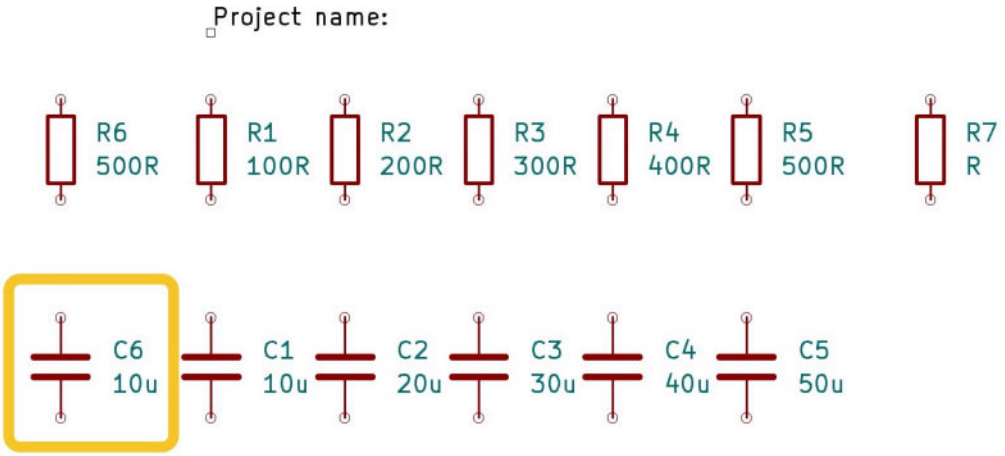


Figure 13.25.7.16: The new capacitor exists in the main branch.

The new capacitor exists in the main branch. You can also use the “log” command to get the recent repository history:

```
% git log
```

```
commit d389140b627c4e76219a337d1dc022f8fbd36244 (HEAD -> main,
experimental)
Author: peter <peter@txplore.com>
Date: Tue Jul 27 09:01:27 2021 +1000

    Added capacitor.

commit 7e4bf0e1fb2c1624f08743d6f4660c4daeaf557a
Author: peter <peter@txplore.com>
Date: Tue Jul 27 08:59:32 2021 +1000

    Added a resistor.

commit 16a0e89fc22F74dad95f4f20eccdee81e583e084
Author: peter <peter@txplore.com>
Date: Tue Jul 27 08:53:18 2021 +1000

First commit.
%
```

As you can see from the output above, commit d389140 exists in both the main and the experimental branches.

Now that the changes in the experimental branch are merged with "main," we no longer need the experimental branch. It is good practice to delete unneeded branches and keep the Git repository tidy. To delete a branch, use the "branch" command with the "-d" switch, like this:

```
% git branch –d experimental
Deleted branch experimental (was d389140).
%
```

Repeat the "git branch" command to confirm that the repository contains only the main branch.

If the branch that you want to delete contains unmarked changes, but you are sure that you do want to delete it, you can use the "-D" switch (instead of "-d"). This will force Git to delete a branch with unmarked changes. Without it, Git will ask you for confirmation. Remember: Git is designed to minimize the risk of losing work. If you try to do something that can result in lost work, Git will let you know and prevent you from losing work.

26. Sharing your KiCad project on 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 cloud service like Github. The repository on Github is a remote copy of your local repository. When 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.

In this chapter, you will learn how you can use Github to share your existing KiCad project with others. The project I will be using to demonstrate the process is the one I also used in the chapter "25. KiCad project management with Git".

The process has four steps:

1. Create a remote repository on Github.
2. Set the origin.
3. Sync the remote repository with your local Git repository.
4. Share Github repository with others.

Let's begin.

1. Create a remote repository on Github

Start by creating a free account on Github if you don't already have one. Then, login and create a new blank Github repository (Figure 13.26.1) :

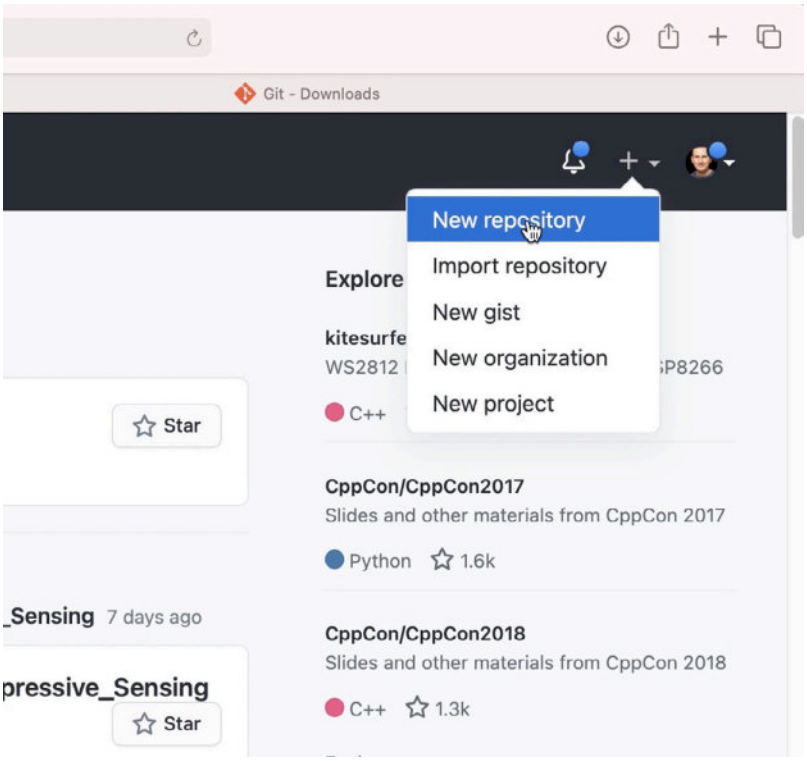


Figure 13.26.1: Create a new blank repository on Github.

Give it a name, a description, and an access type. I have set my demo repository to "Public." There are a few optional settings that you can enable, such as the addition of a README or ".gitignore" files (Figure 13.26.2):

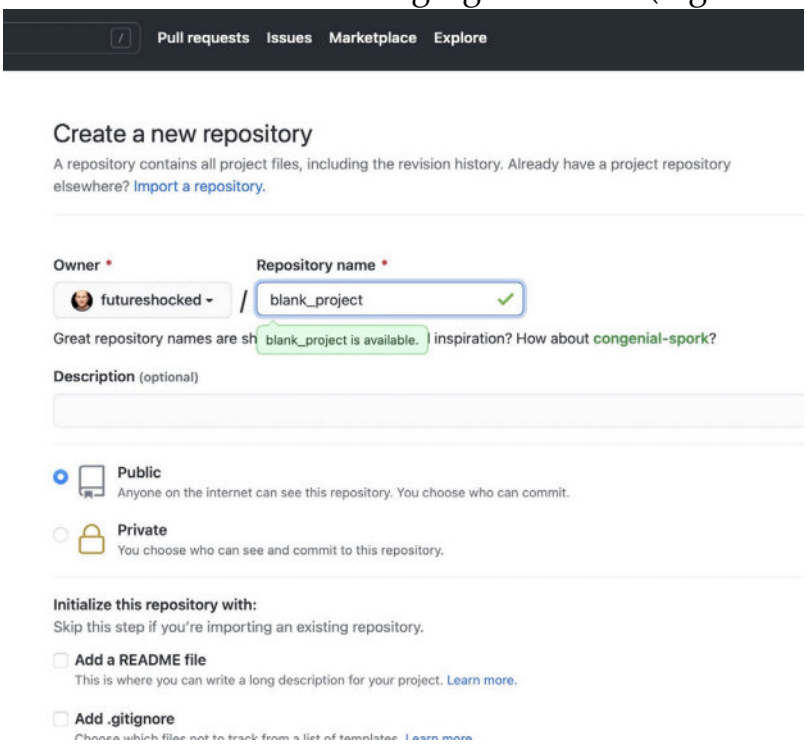


Figure 13.26.2: Create a new repository on Github.

Click on the green "Create repository" button to finish this step.

2. Set origin

When Github completes creating the new repository, it will show you a page containing information that you will need in the next few steps. Copy this information in a text editor so that you can use it later. Most important is the HTTPS and SSH address of your Github repository that appears at the top of the page ("1" in Figure 13.26.3):

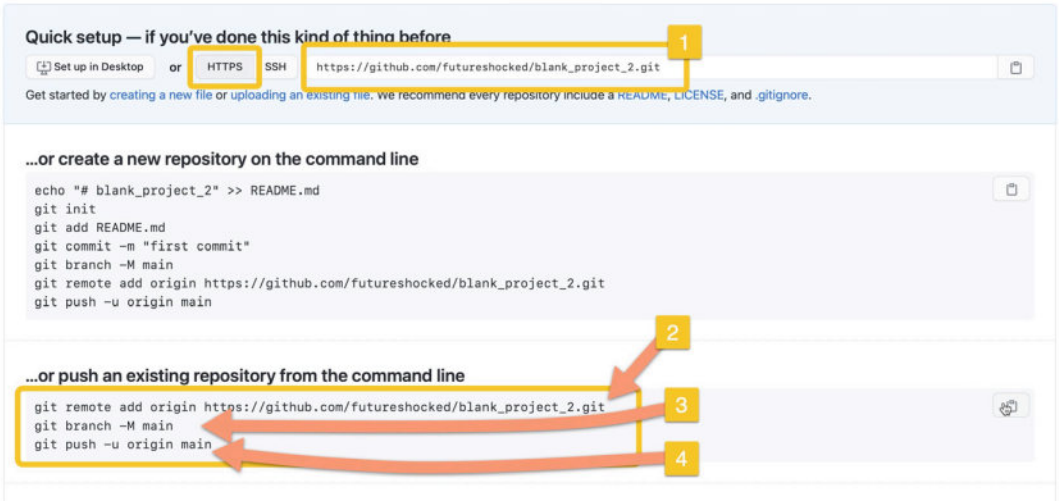


Figure 13.26.3: The URL of the new repository.

The same page contains instructions on setting the "origin" repository for your local Git repository. The name "origin" simply represents the remote repository of your local Git repository that (in our example) is hosted on Github. You may choose a different name if you wish, like "GitHub" or "cloud," however, remember that convention uses "origin" for the primary remote repository.

Also, notice at the bottom of the Github instructions in Figure 13.26.3 the commands Github suggests referring to the "main" branch. This means that we will be syncing the "main" branch first. If your local repository contains additional branches that you wish to "push" to Github, you can do this subsequently. You can choose to push specific branches only.

Next, go to your command line, and browse to the directory where you saved your project.

Copy the first line of the Github instructions ("2" in Figure 13.26.3):

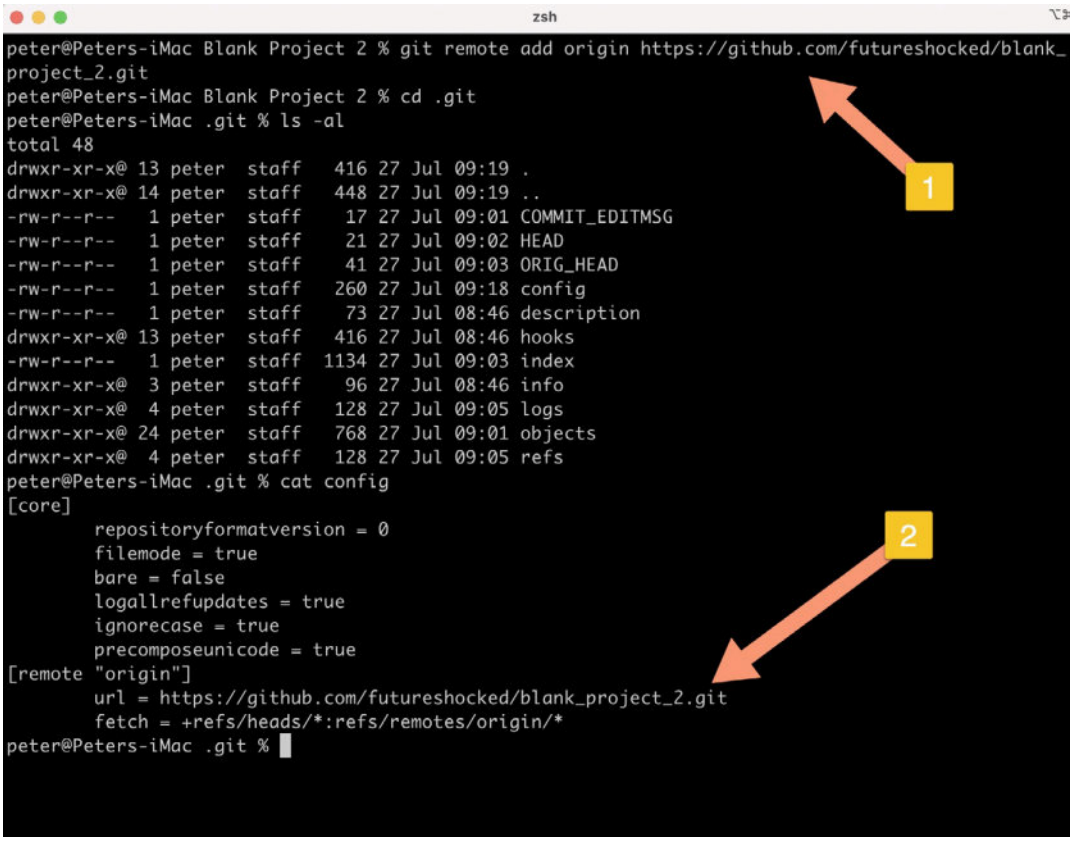


Figure 13.26.4: Setting the origin.

When you set the origin ("1" in Figure 13.26.4), Git will record the origin URL in the config file inside the ".git" folder. You can confirm this by using the "cat" command on the config file (or simply open config in a text editor). See "2" in Figure 13.26.4.

Github also recommends that you use "git branch -M main" to switch the name of the unborn branch to "main." This is unnecessary because I have already created the "main" branch in this repository. You can still issue this command, but it will not change anything.

Let's continue with step 3.

3. Sync the remote repository with your local Git repository.

I will now push the "main" branch of my local repository to Github. To do this, I will use the 3rd command from Figure 13.26.4. This requires setting up a new personal access token that you can use in place of your regular account password.

To create a personal access token (refer to Figure 13.26.5), go to Github, click on your account photo (top right of the Github page), then Settings (1), and Developer settings (2).

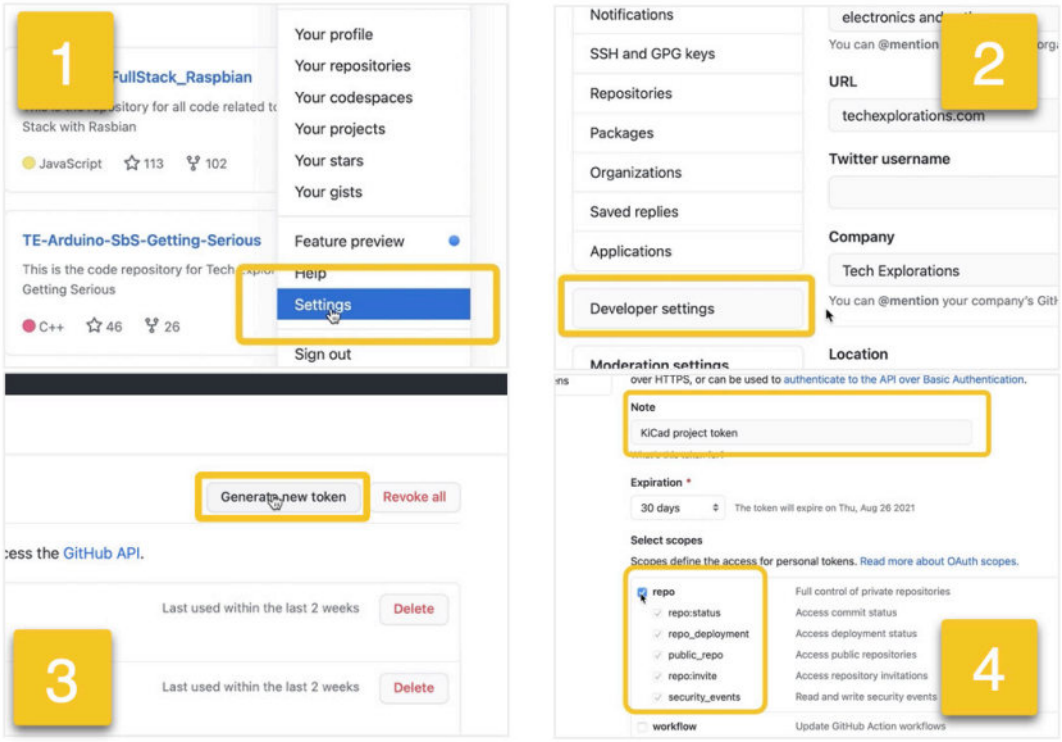


Figure 13.26.5: Generate a new personal access token.

Click on Personal access tokens in the Developer Settings page and then "Generate new token" (3). Give your new token a name, and select all radio buttons in the "repo" group (4). Click on the green "Generate token" button to finish the process.

You now have a new personal access token (Figure 13.26.6):

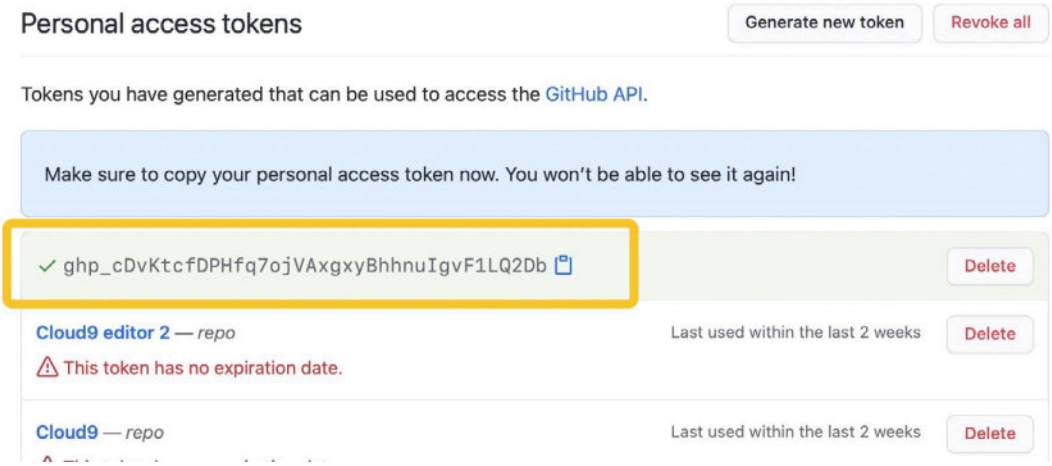


Figure 13.26.6: The new personal access token.

Continue at the command line. Copy command 4 from Figure 13.26.3:

```
% git push –u origin main
```

Github will ask for your username and password. Type or copy your username (or email).

For the password, use the personal access token that you created earlier (Figure 13.26.6)

Git will proceed to write "main" branch data to origin (i.e. to the remote repository):

```
peter@Peters-iMac Blank Project 2 % git push -u origin main
Username for 'https://github.com': futureshocked
Password for 'https://futureshocked@github.com':
Enumerating objects: 22, done.
Counting objects: 100% (22/22), done.
Delta compression using up to 16 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (22/22), 334.88 KiB | 7.13 MiB/s, done.
Total 22 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/futureshocked/blank_project_2.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
peter@Peters-iMac Blank Project 2 %
```

Figure 13.26.6: Synced "main" branch.

Your project ("main" branch of the repository) is now on Github. Refresh the project page to confirm:

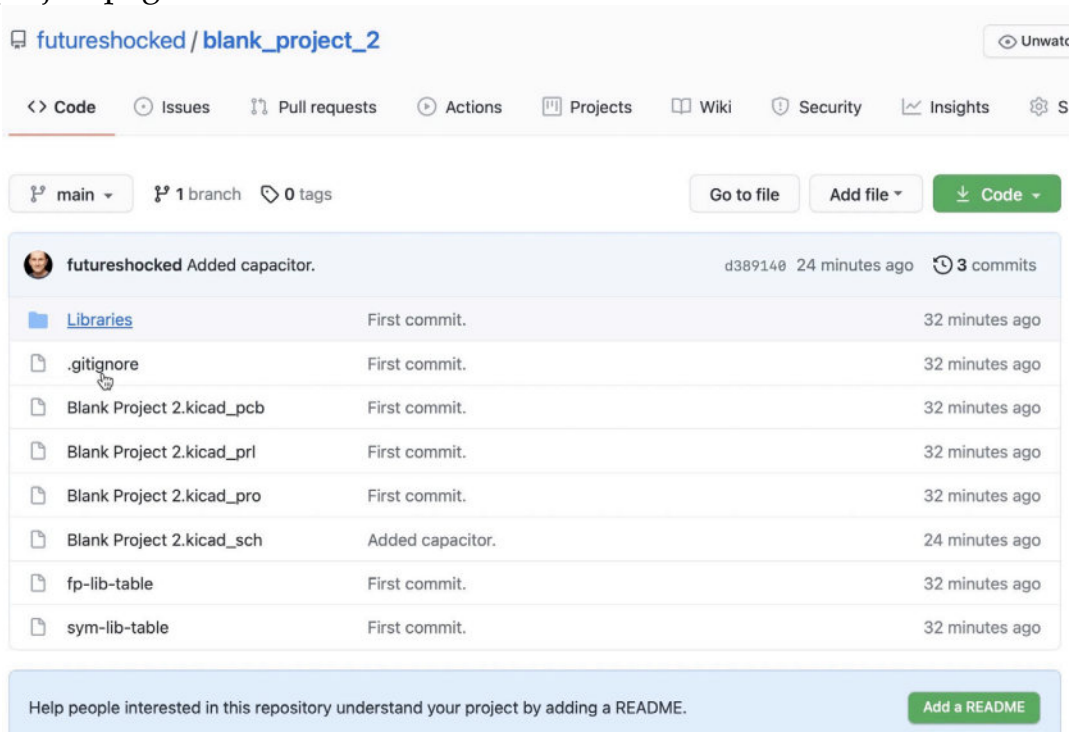


Figure 13.26.7: The KiCad project, "main" branch, on Github.

Take a moment to browse the files and look at their contents.

Now, say that you want to do some work on the local branch. For example, you'd like to create a new branch to do some experimental work. As you already know, you can create a new branch like this:

```
% git checkout -b  experimental2
```

Then, go to the schematic and make a small change, like add a new resistor. Save the schematic.

Back on the command line, commit the change, and look at the repository status:

```
% git checkout -b experimental2
Switched to a new branch 'experimental2'
peter@Peters-iMac Blank Project 2 % git branch
* experimental2
  main
% git status
On branch experimental2
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: Blank Project 2 kicad_sch

no changes added to commit (use "git add" and/or "git commit -a")
% git commit -am "Added R8."
[experimentalZ @bfaeSZ] Added R8.
 1 file changed, 22 insertions(+)
% git status
On branch experimentalZ
nothing to commit, working tree clean
%
```

The new branch ("experimental2") contains a change I'd like to push to Github. Branch "experimental2" does not exist on Github. To create the new branch and push its changes from local to "origin," I'll use this command:

```
% git push -u origin experimental2
Enumerating objects: 9, done. I
Counting objects: 100% (9/9), done.
Delta compression using up to 16 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 1.01 KiB | 1.01 MiB/s, done.
```

```
Total 7 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 2 local
objects.
remote:
remote: Create a pull request for 'experimental2' on GitHub by
visiting:
remote: https://github.com/futureshocked/blank_project_2/pull/new/
experimental2
remote:
To https://github.com/tutureshocked/blank_project_2.git
* [new branch] experimental2 -> experimental2
Branch 'experimental2' set up to track remote branch 'experimental2'
From 'origin'.
%
```

Git did not ask for my credentials again because I have configured it to cache them for 24 hours.

Refresh the repository webpage to confirm that the new branch was created on origin (1 in Figure 13.26.8):

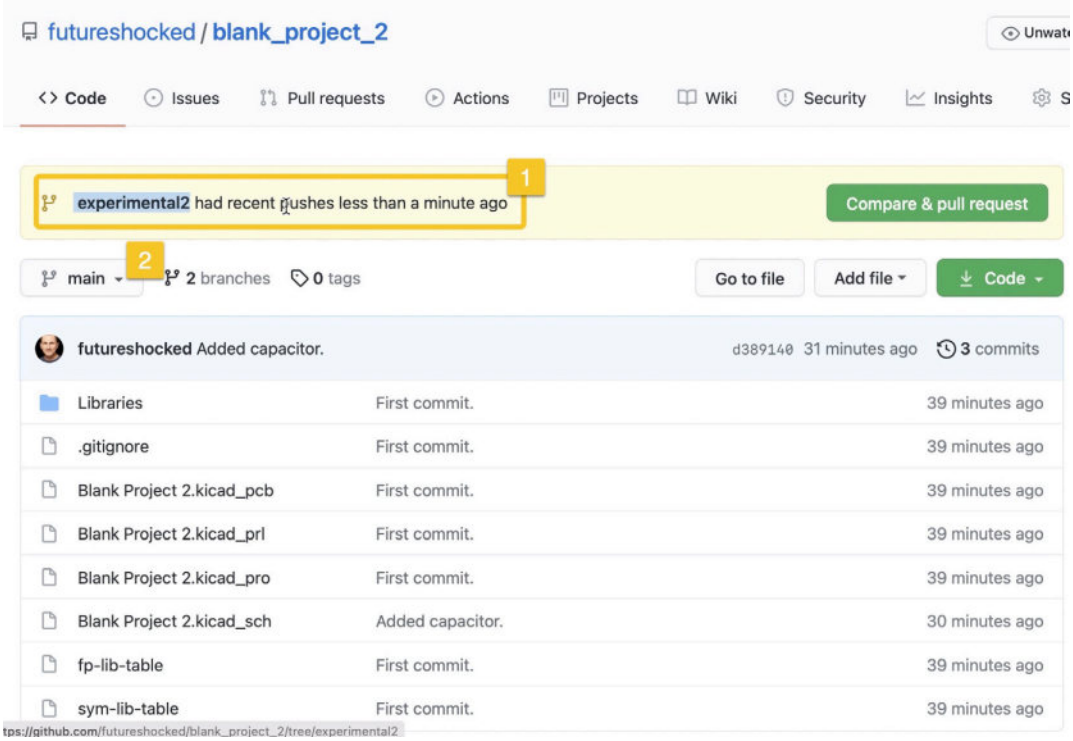


Figure 13.26.8: The new branch at origin.

You can use the branch dropdown on Github to switch between branches (2 in Figure 13.26.8).

4. Share Github repository with others

To share your project with others, just give them the repository URL on Github. Your collaborators will be able to clone the repository on their computer, make changes, and then push the changes to origin. You can pull the changes to your local repository using the "pull" command:

```
% git pull –u origin experimental2
```

The command above will pull changes made to the "experimental2" branch. Change the branch name to something else, like "main," to pull changes from any other branch.

Let's simulate this. You can use Github's edit function to make a change to a repository file. Below, I am making a small change to the schematic file and saving the change (all done on Github, not the local repository):

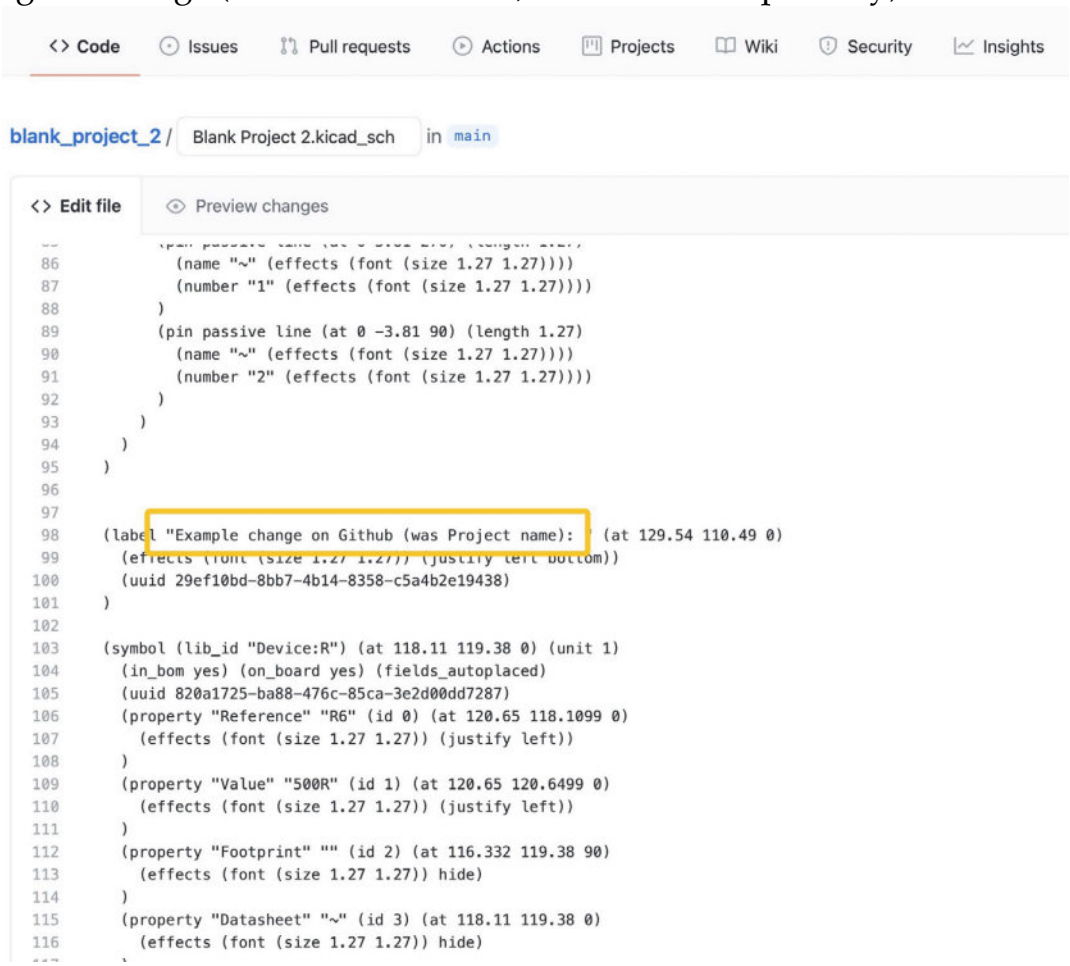


Figure 13.26.9: Made a change to the schematic file at origin.

Click on the green "Commit changes" button to commit and exit the editor.

Back on the command line, issue the “pull” command on the “main” branch (where I made the change):

```
% git pull origin main
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

From https://github.com/futureshocked/blank_project_2

* branch main -> FETCH_HEAD
  d389140..585ff40 main -> origin/main

Auto-merging Blank Project 2.kicad_sch
Merge made by the 'recursive' strategy.
   Blank Project 2.kicad_sch | 2 +-
   1 file changed, 1 insertion(+), 1 deletion(-)
%
```

Git confirms that it has pulled changes in the schematic file of the ” main” branch. If you switch your local working branch to “main”, and then open the schematic in Eeschema, you will see the change in the editor:

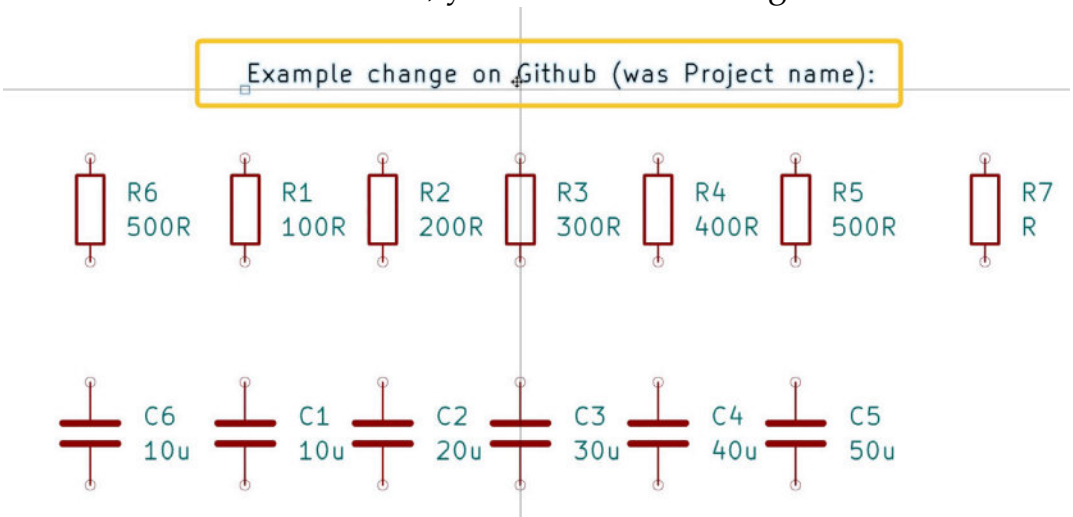


Figure 13.26.10: The change I made on Github appears in the schematic editor.

In Figure 13.26.10 (above) (above), you can see that the change that I made to the schematic file on Github appears in the Kicad schematic editor.

The process I have described above represents a simple use case or creating a new KiCad repository, sharing it with other people using Github, and syncing the repository with changes made by the project members.

Github is very versatile, and you can adjust it to your exact personal or team workflow requirements. To learn more about it, consider our dedicated Github course on the Tech Explorations website.

27. Customize the editor color scheme

From its early days, KiCad has been very configurable. In KiCad 6, it is now possible to configure the color theme used in the schematic and the layout editors.

You can create custom color schemes for Eeschema and Pcbnew in the Preferences window. Each app has its own theme settings. You can create multiple themes and quickly switch between them. See Figure 13.27.1 below for reference:

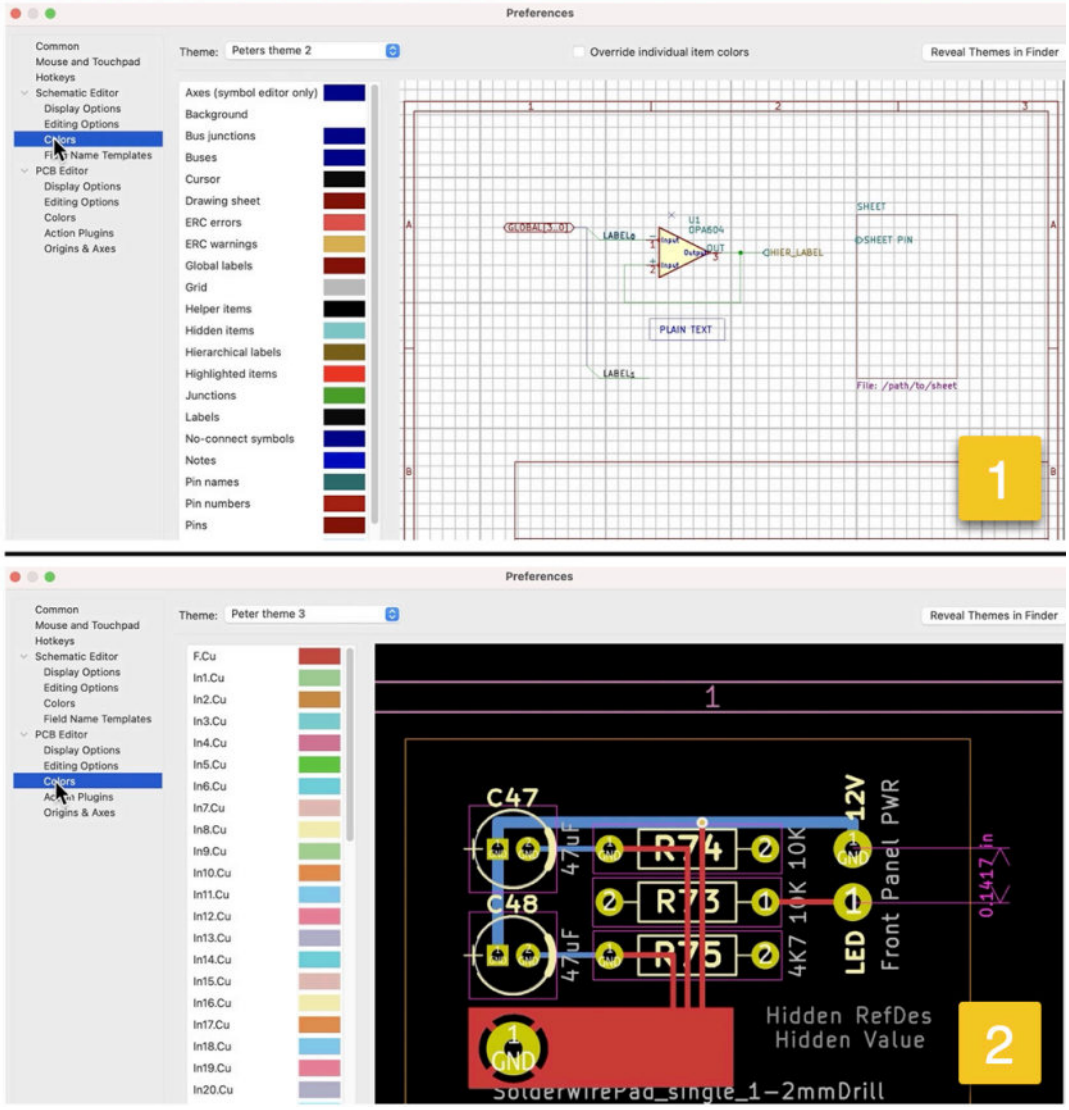


Figure 13.27.1: The Colors theme editors for Eeschema and Pcbnew.

Both editors work the same. You can select a theme via the drop-down menu at the top of the editor:

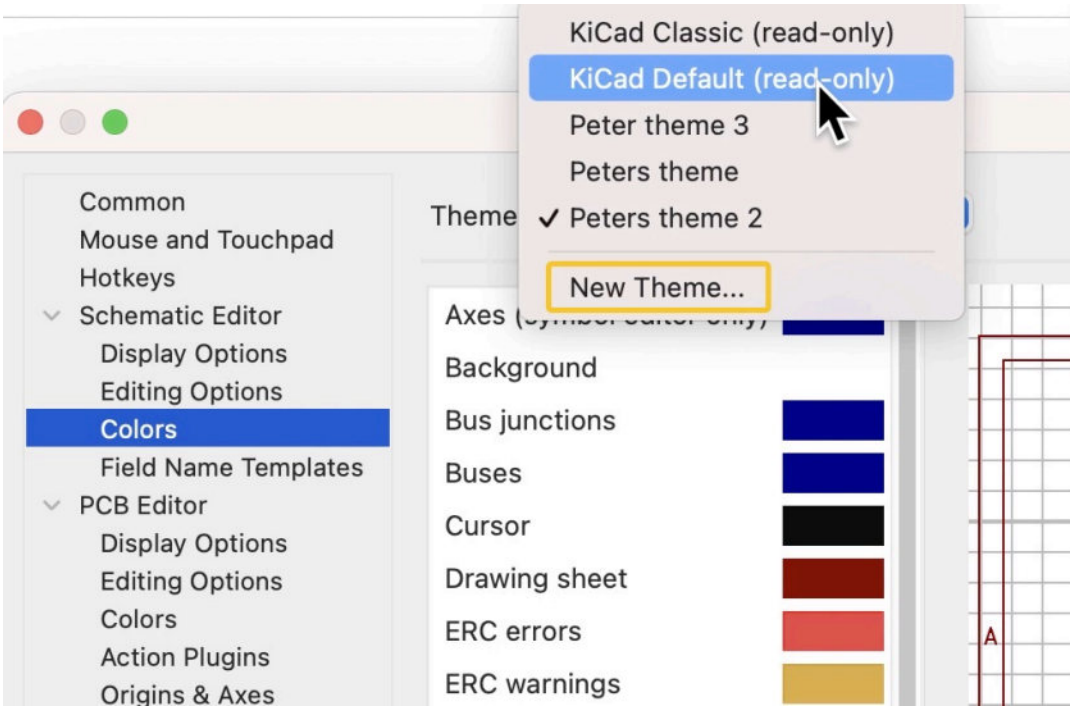


Figure 13.27.2: The theme drop-down.

You can click on “New Theme…” to create a new theme.

All themes except for the two build-in, “KiCad Classic” and “KiCad Default,” are editable. To change a color, double-click on the box that contains the color and use the color picker or defined colors tabs to select the new color. Then, click on OK to commit the new color.

For example, to change the color of the background in Eeschema, click on the box that contains the background color:

Figure 13.27.3: Assigned Gray 3 to Background.

In the example above, I have selected Grey 3 from the Defined Colors tab for the background. Click OK to dismiss the Color Picker window, and OK again to dismiss the Preferences window.

The schematic editor now looks like this:

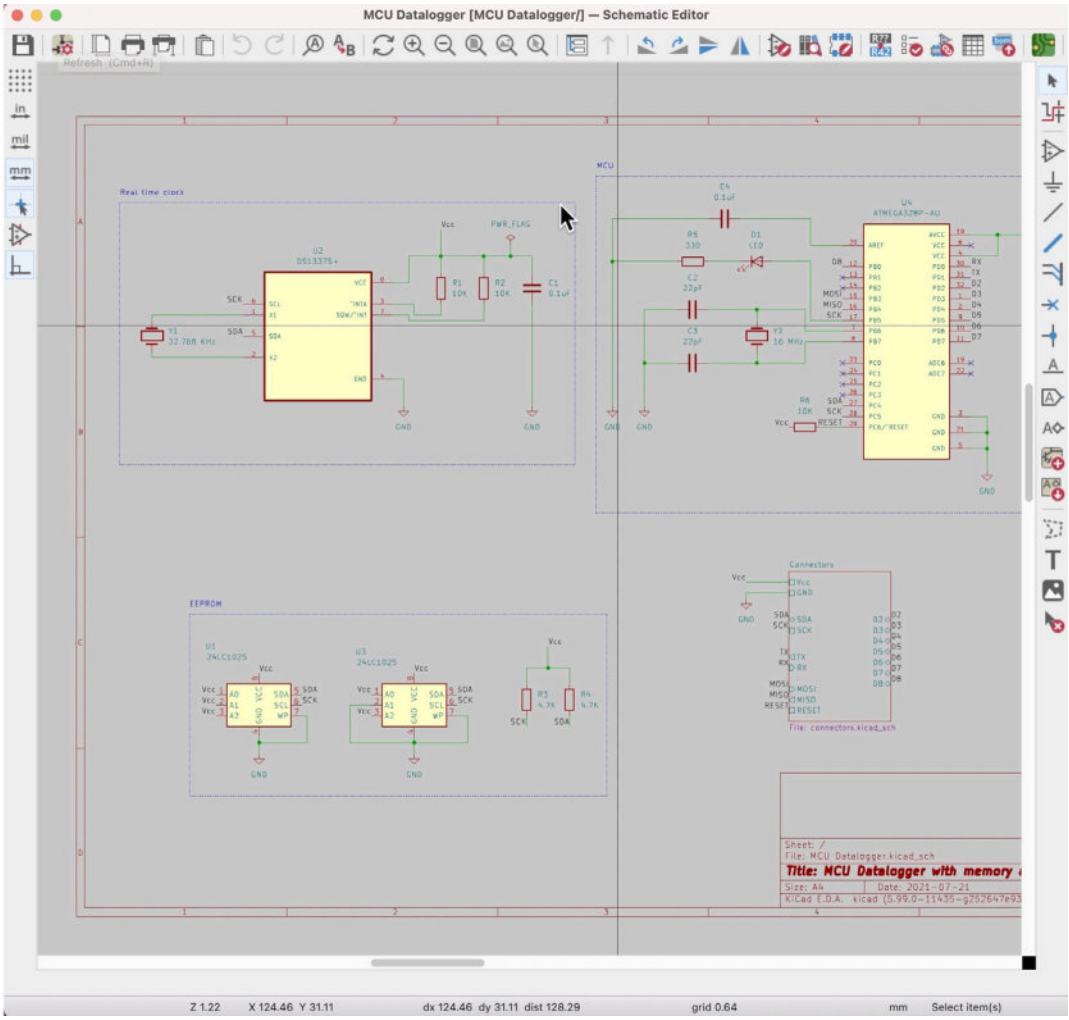


Figure 13.27.4: The schematic editor with Gray 3 for the background.

You can use the same method to set colors for all available items in Eeschema and Pcbnew.

I find it helpful to have a theme for regular editing work and another theme for printing. Pcbnew's default theme has a black background which is not suitable for printing on paper because of the amount of blank ink used. I have a printer-friendly theme with a white background so that my layout printouts look better on paper:

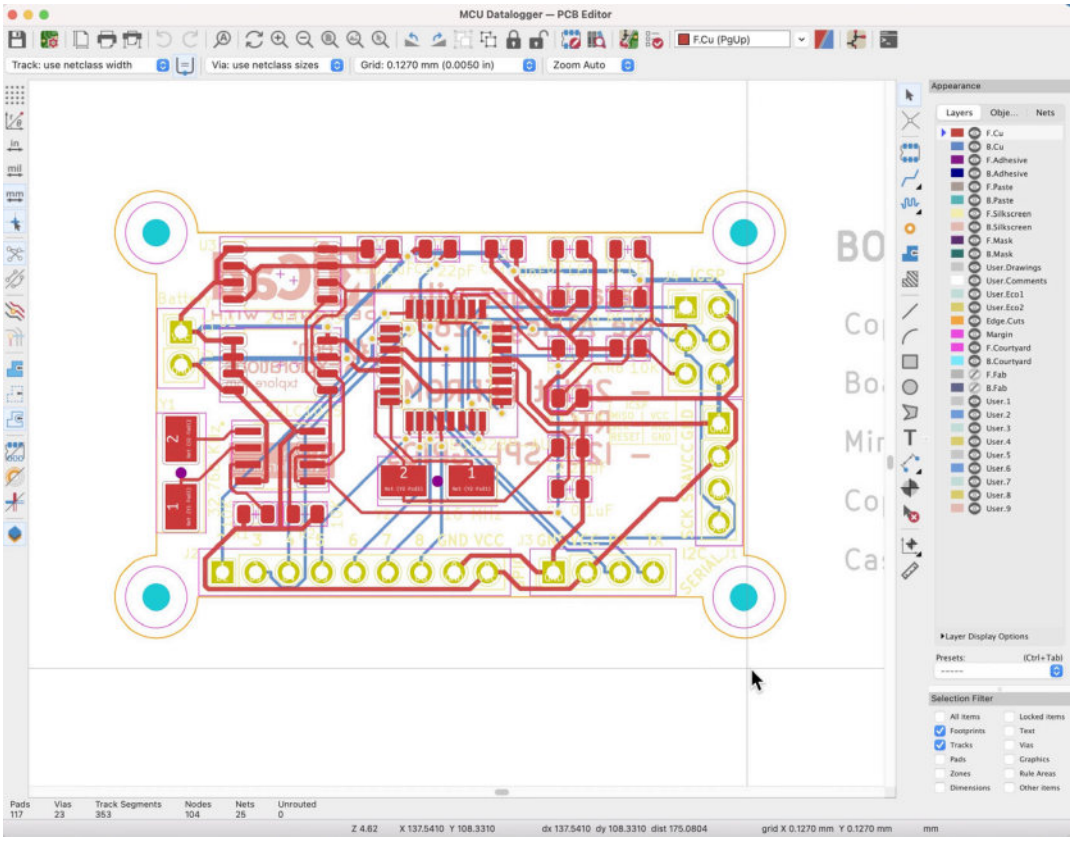


Figure 13.27.5: Pcbnew using a printer-friendly color theme.

28. Import an EAGLE, Altium, or Cadstar project

KiCad can work with PCB projects created in the CAD applications. KiCad 6, in particular, can import projects from Eagle, Altium, and Cadstar. In most cases, imported projects will need editing to correct issues that arose from incompatibilities between these CAD applications and KiCad.

I will show you how to use the import tool using an example. I will import an Eagle project for the Arduino Uno board into KiCad.

Start by going to the Arduino website from where you can download the Eagle files (see https://store-usa.arduino.cc/products/arduino-uno-rev3/):

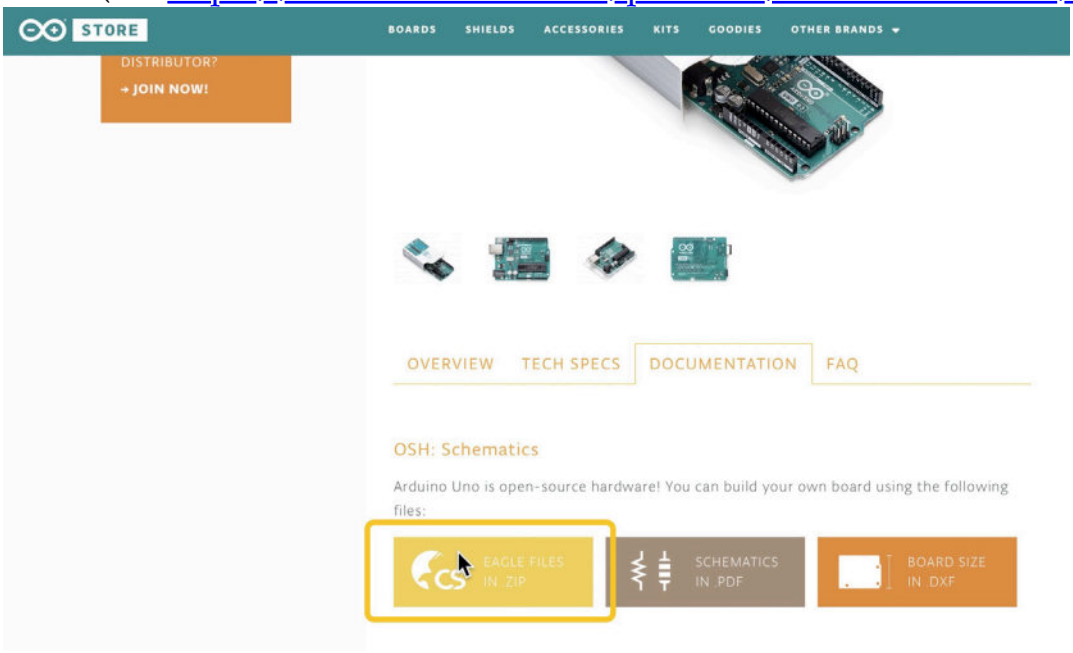


Figure 13.28.1: Get the Eagle project from the Arduino website.

The schematic I will import into KiCad looks like this:

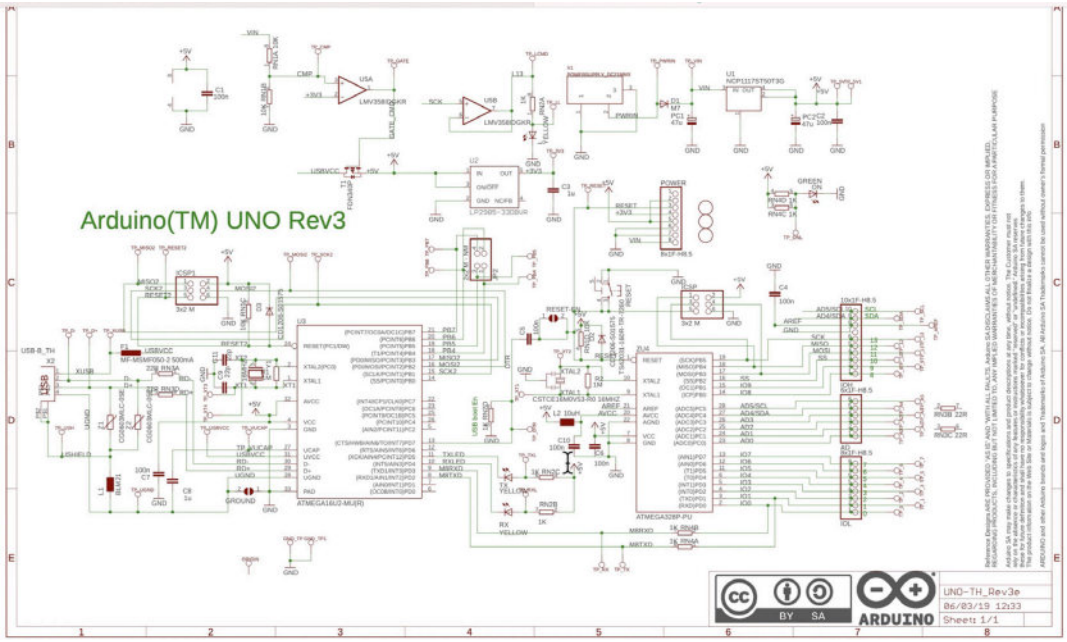


Figure 13.28.2: The Arduino Uno schematic in Eagle.

Expand the ZIP archive so you can see the two files it contains, the schematic (".sch") and the layout (".brd"):

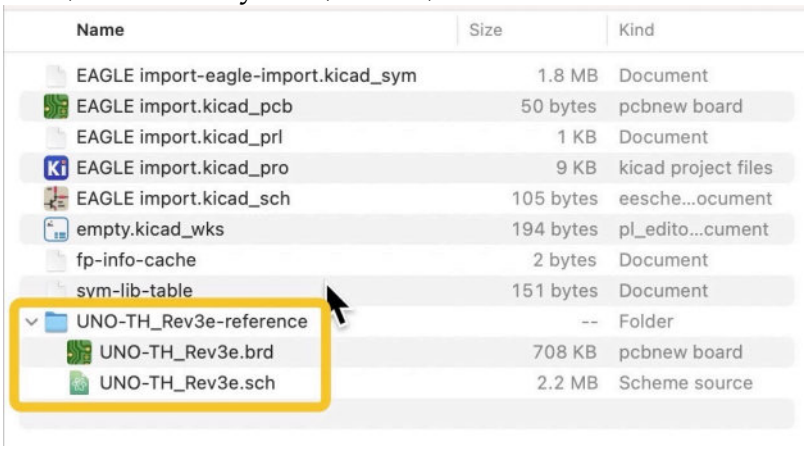


Figure 13.28.3: The two files of the ZIP file.

In KiCad, create a new project, and open Eeschema. Then click on File —> Import —> Non-KiCad Schematic.

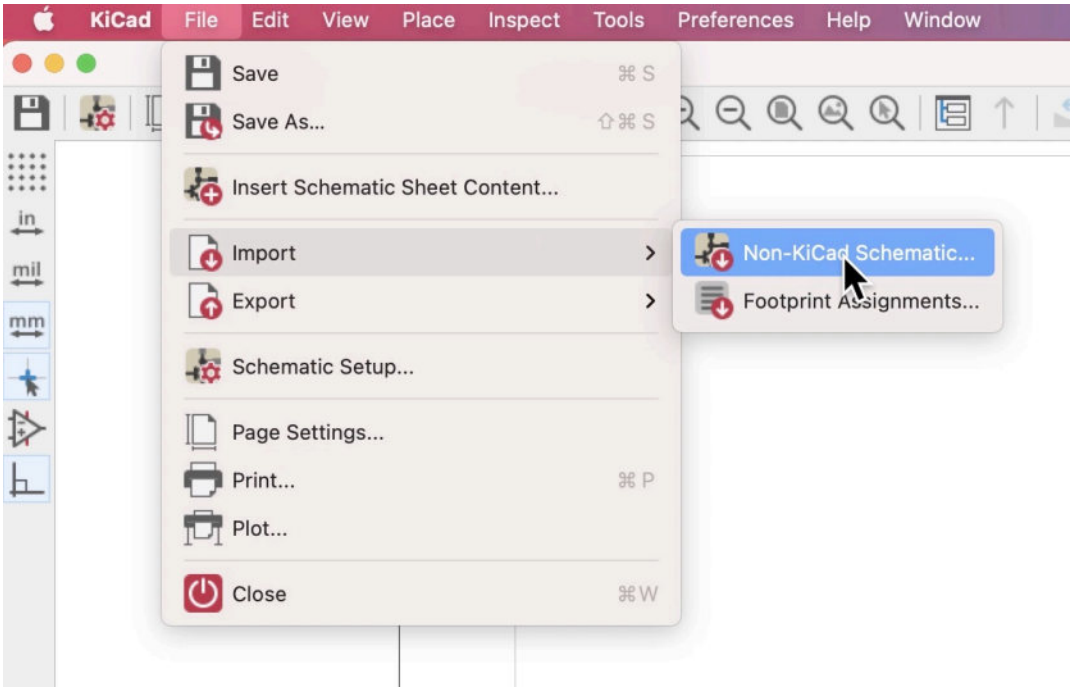


Figure 13.28.4: Importing a non-KiCad Schematic.

A file browser will appear. At the bottom of the browser, select the appropriate import file format ("1" in Figure 13.28.5), and then select the file with the ".sch" extension ("2" in Figure 13.28.5).

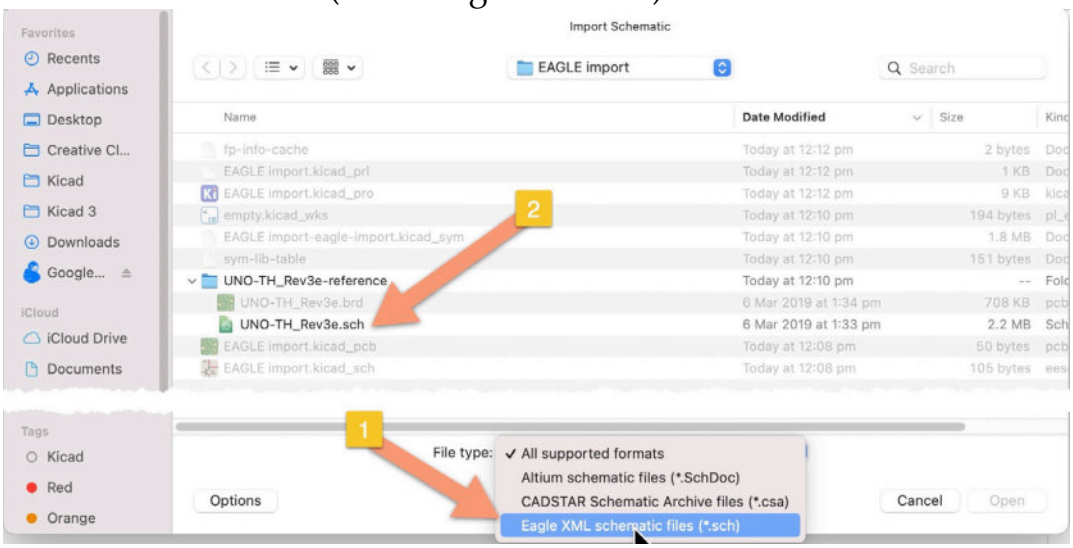


Figure 13.28.5: Select the file type and then the Eagle schematic file.

KiCad will try to import this file. KiCad will likely report one or more issues. For example, it may not be able to find a matching schematic component (Figure 13.28.6). KiCad will report this at the end of the import process. You can use this information to fix any issues that arise manually.

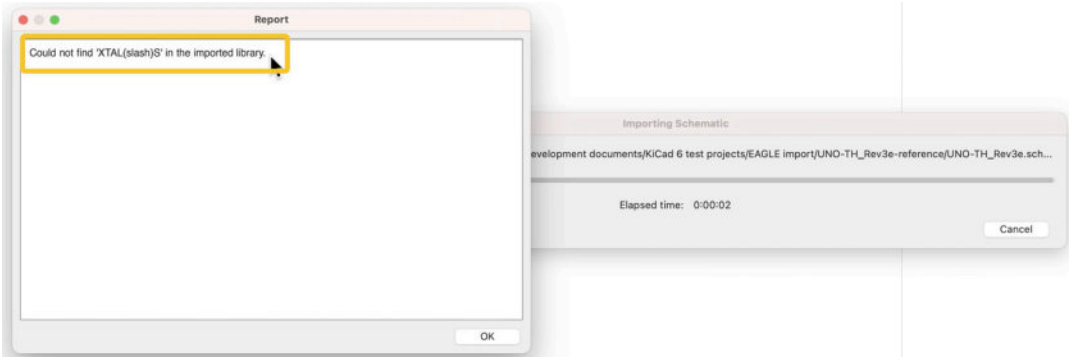


Figure 13.28.6: KiCad was unable to find a match for one component.

Click OK to dismiss the Report window and return to the schematic editor. At first glance, the import looks good:

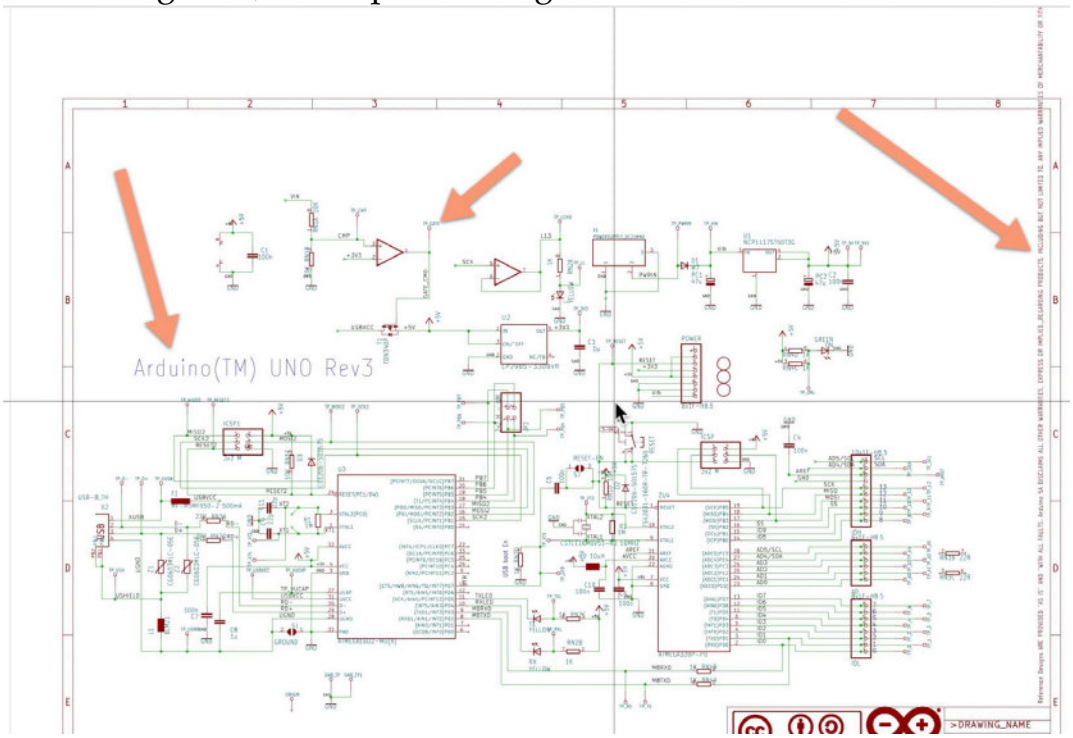


Figure 13.28.7: KiCad has imported the Arduino Uno Eagle schematic.

You can compare the screenshot in Figure 13.28.7 (above) with the one in Figure 13.28.2 and should notice a few differences:

- The editor sheet size is wrong.
- Text size and type are different.
- The black text on the right side of the schematic that appears in multiple lines in Eagle appears in a single (very long) line in KiCad.

You should take a bit of time to fix those issues, especially missing components. In this case, I will fix the problem with the single long line on the right side of the schematic simply by deleting it (click to select it, then hit the

delete key). This will remove the frame around the schematic, with its label imported from the Eagle schematic file. With this gone, I can move the schematic itself inside the KiCad schematic sheet and select an A3 size sheet from the Page Settings window:

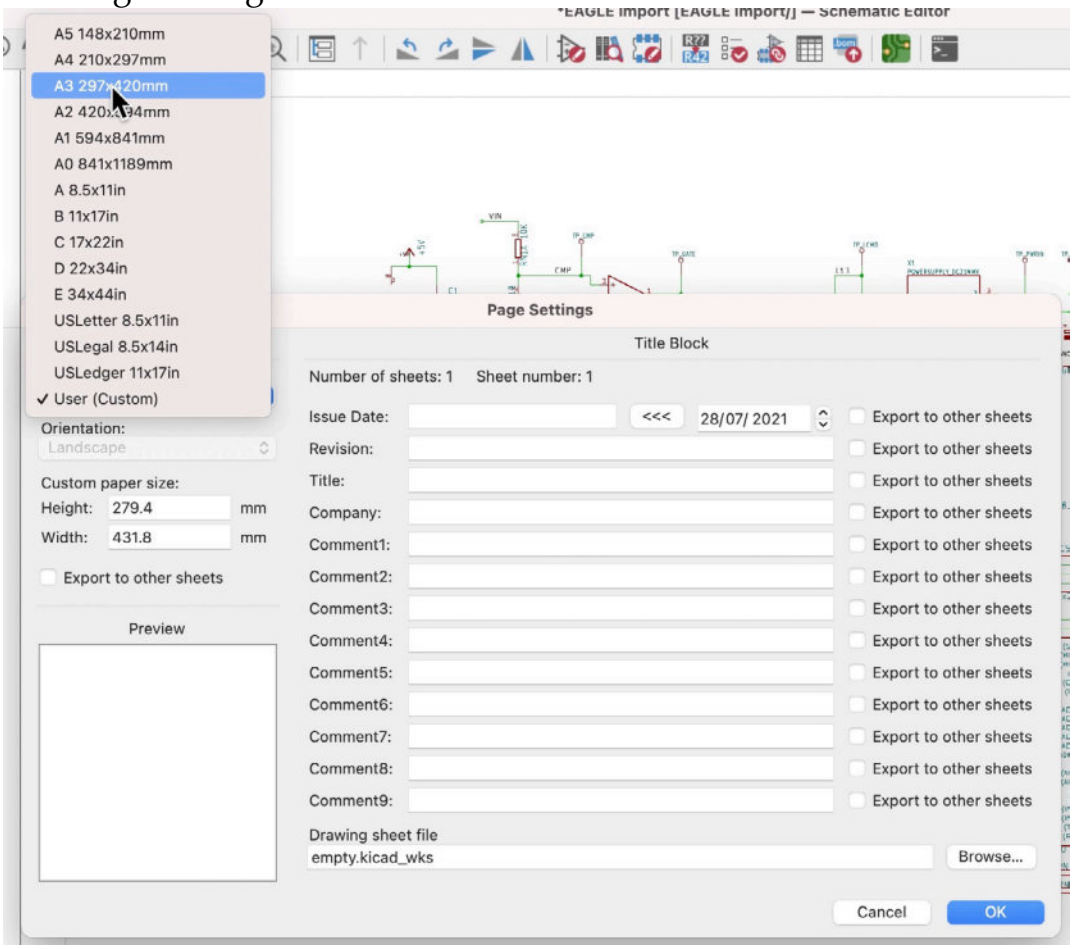


Figure 13.28.8: Set a new sheet size.

The imported schematic for the Arduino Uno now looks like this:

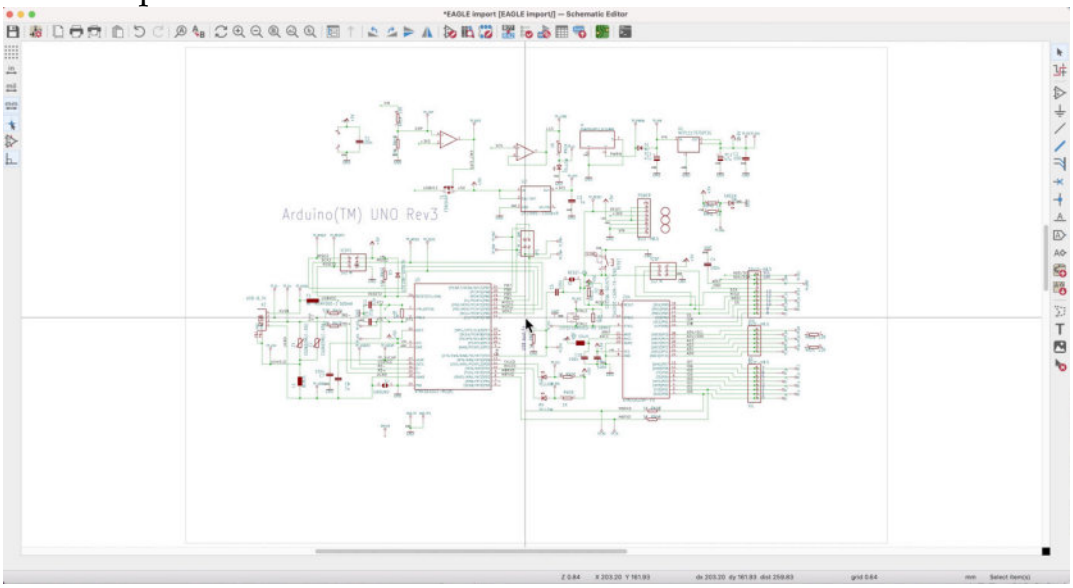


Figure 13.28.9: The Arduino Uno schematic imported in Eeschema.

Being mindful that different CAD applications have features and capabilities that don't "translate" directly to KiCad, you can import schematics from Eagle, Altium, or Cadstar into KiCad. In most cases, you will need to do a small amount of editing to correct outstanding issues.

29. The circuit simulator

One of the more frequent questions I receive from readers is about the circuit simulator that comes with KiCad. You can find the simulator in Eeschema under the Inspect menu (Figure 13.29.1).

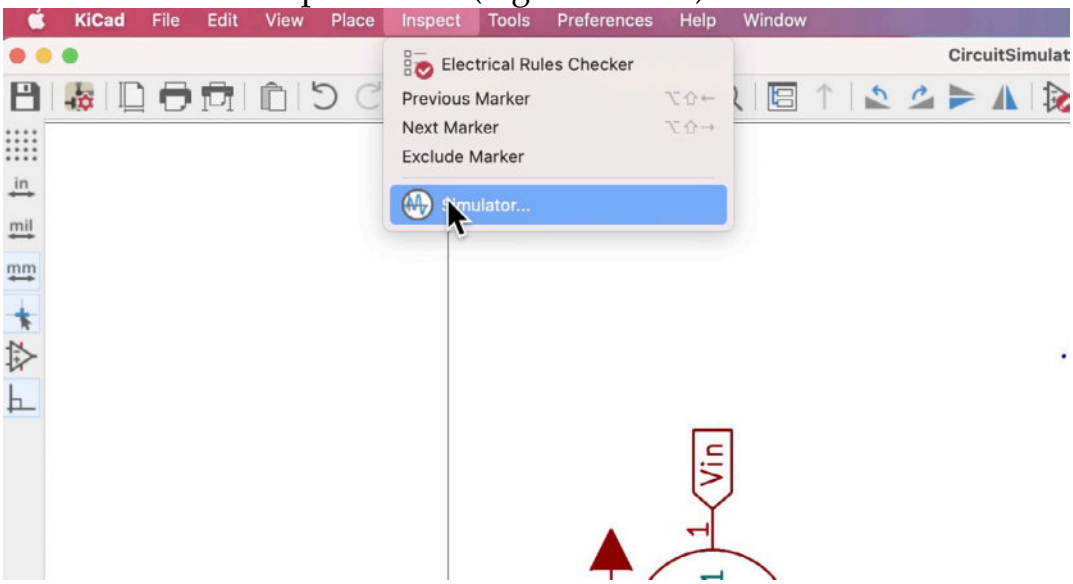


Figure 13.29.1: The circuit simulator in Eeschema.

The simulator does have a reasonably steep learning curve, but once you learn the basics, you will be able to do circuit analysis such as the one in Figure 13.29.2 (below):

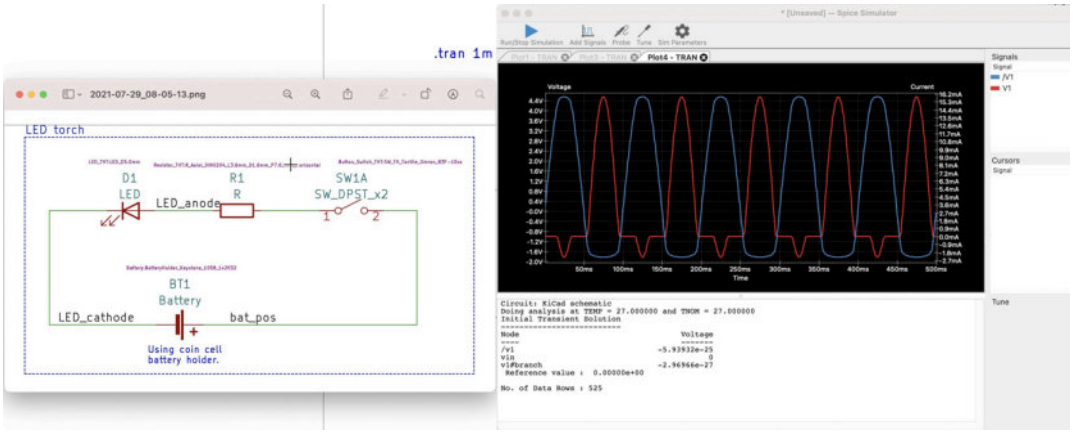


Figure 13.29.2: An example simulated circuit and the analysis outcome.

In the example above, I have simulated the schematic diagram of one of the projects in this course (the "LED torch" project). The simulation generated a graph of the voltage and current present in the circuit and represented 500 ms of operation.

Unfortunately, it is not possible to simulate a standard Eeschema schematic. First, you will need to make a few modifications to the schematic

so that the simulator can read its components. You will also need to assign appropriate models to each of the circuit components and then set the simulation parameters. In the example simulation above (Figure 13.29.2),the actual file that I simulated looks like the one below (Figure 13.29.3):

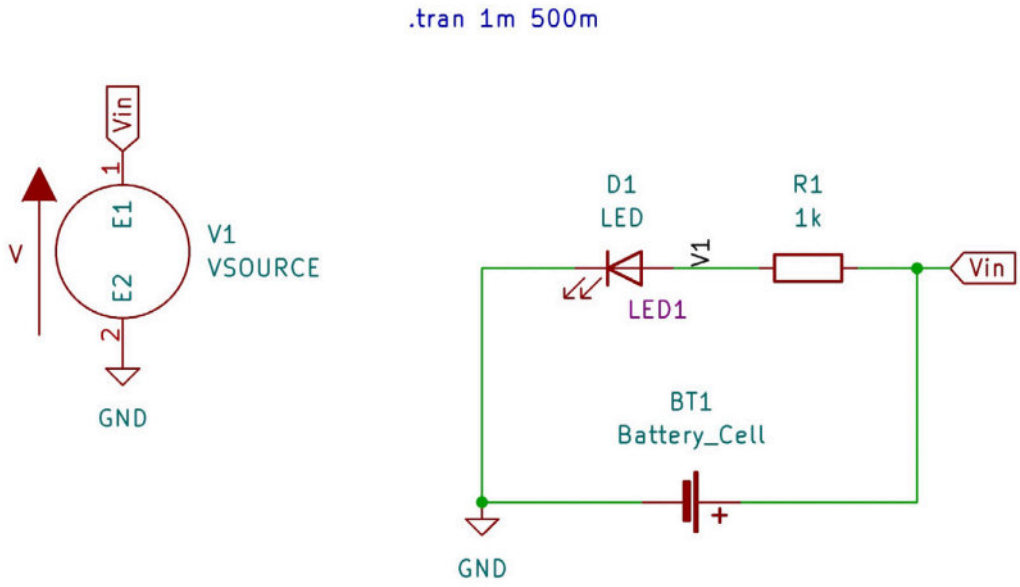


Figure 13.29.3: This circuit schematic is compatible with SPICE.

Notice that the new circuit contains a voltage source, and I have removed the switch so that the current can circulate. I have also added a text item above the schematic that includes the custom simulation directives.

I will show how to do all this later in this chapter and its segments.

Before we get started, I will discuss the historical underpinnings of the circuit simulator in KiCad.

The simulator integrated into KiCad is based on SPICE ("Simulation Program with Integrated Circuit Emphasis"). SPICE is an open-source analog electronic circuit simulator. Its development began almost 50 years ago at Berkley University and was originally written in Fortran. The last version of SPICE from Berkey University was published in 1993. Since that time, SPICE has seeded several open-source successors, like XSPICE and CIDER. Commercial versions have also been published, such as ISPICE, HSPICE, and LTSpice.

KiCad uses the ngspice variant, which is open-source. KiCad's spice implementation can use models created for the original Spice, as well as LTSpice, PSpice and HSpice. A model is a file that contains simulation instructions for a component. There are model files for any component in a circuit, such as resistors, transistors, and voltage sources. KiCad's simulator

compatibility with models from the various variants for Spice means that you can use models published by a large group of authors and companies.

Resources

In this chapter, I will give you a demonstration of the circuit simulator in Spice. As Spice is a complicated tool, I will certainly not cover more than some of the basics. To learn more, I suggest you use these resources:

1. KiCad Spice documentation.
2. Ngspice documentation.
3. The Spice page @ Berkeley.
4. The Spice circuit elements and models (also @ Berkeley).
5. The presentation slides from course ECE220, "Introduction to spice source files" (California State Polytechnic University, Pomona).

29.1. Prepare the circuit for simulation

To use KiCad's circuit simulator, the first task is to edit the schematic analyzed by the Spice simulation engine.

In my example (see Figure 13.29.1.4 below), I have made a couple of small modifications:

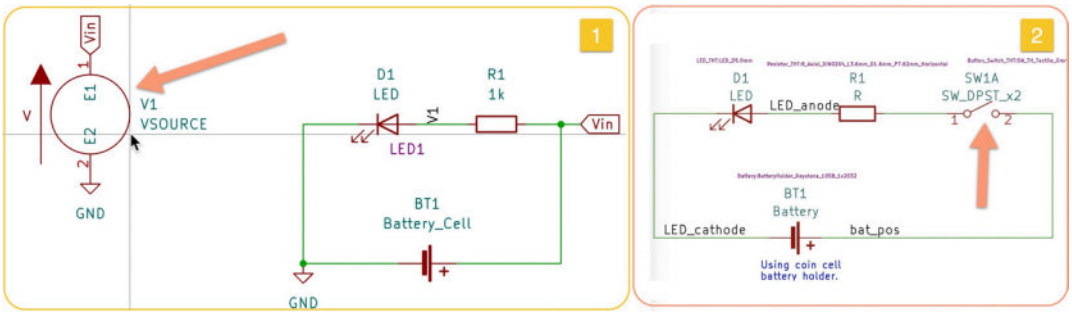


Figure 13.29.1.4: Original schematic (right), SPICE-compatible schematic (left).

The first modification is the voltage source. Many components (mainly passive ones, like resistors and capacitors) have symbols with SPICE models already defines. But others, such as the battery symbol in the example above, are not. It is possible to define a custom simulation model for symbols, but in the case of the battery, the simplest way to deal with it is to insert a voltage source symbol from the SPICE library. The voltage source symbol contains configuration parameters that the simulator can use in its analysis.

There are two steps to set the voltage source for this circuit:

1. Disable the battery cell for the simulation since I will not provide it with a SPICE model.

2. Add a voltage source and use either wires or labels to connect it to the circuit (thus replace the battery cell).

To disable the battery cell from the simulation, double-click on the battery symbol to bring up its properties window, click on the “Spice Model” button to bring up the spice model editor window, and check the “Disable symbol for simulation” box. Click OK and Ok to go back to the editor (Figure 13.29.1.5):

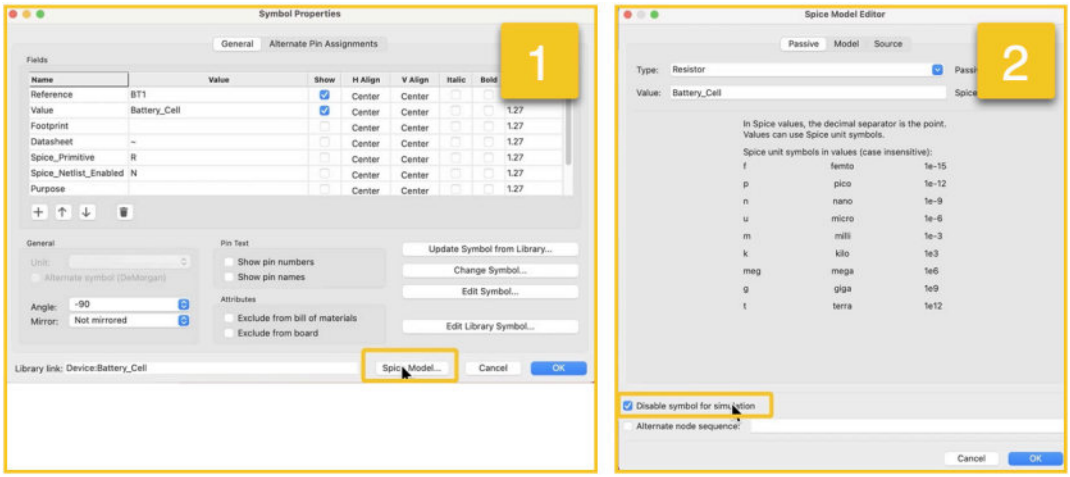
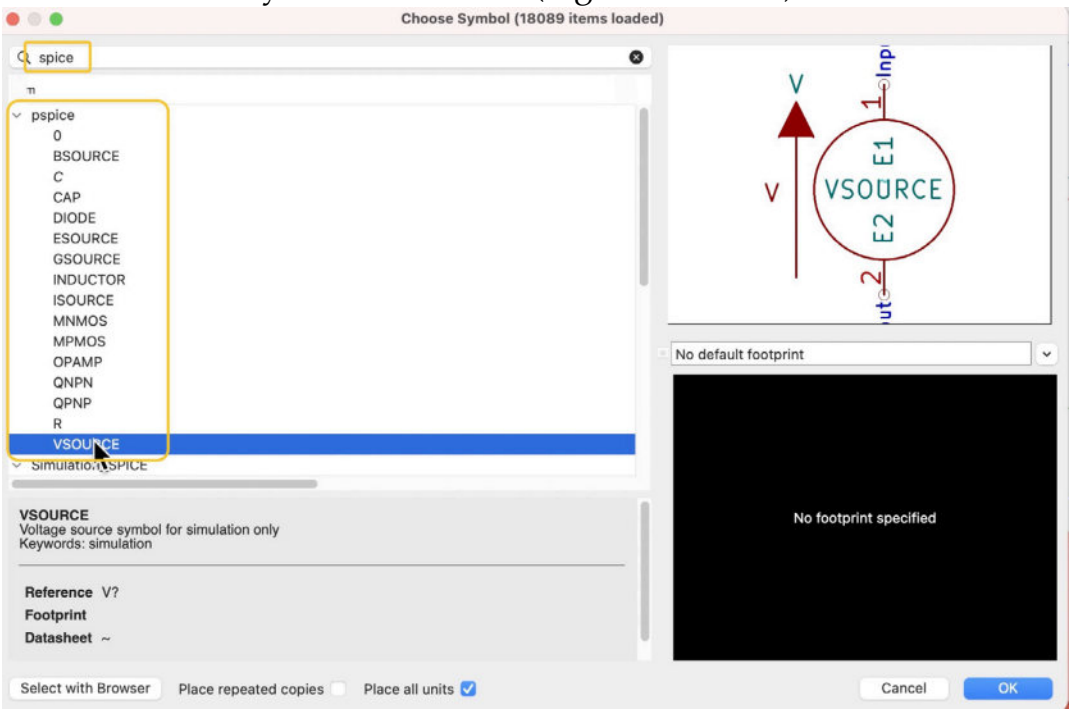


Figure 13.29.1.5: Disable the battery symbol for the simulator.

Next, you must provide a voltage source for the circuit symbol to replace the disabled battery symbol compatible with Spice. For this purpose, KiCad offers a library of Spice symbols. Choose the symbol tool from the right toolbar in Eeschema, and search for the spice (or “pspice” library). One of the symbols in that library is “VSOURCE” (Figure 13.29.1.6):



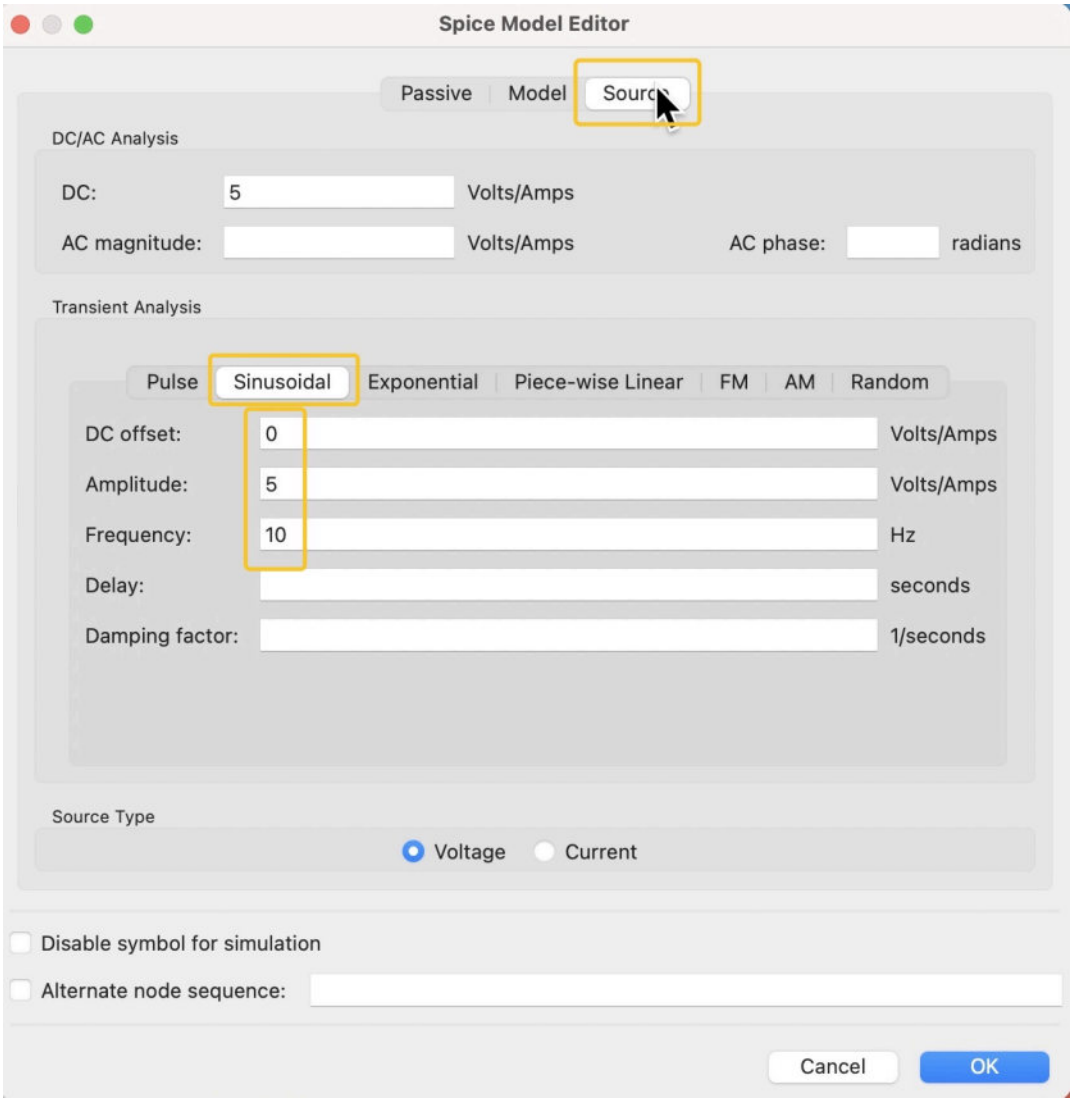
13.29.1.6: A voltage source symbol that is compatible with the simulator.

After you add the new voltage source in the editor, do the wiring. I prefer to use labels for a cleaner look. In Figure 13.29.1.4, notice how I have attached the "Vin" label and the GND symbol to the voltage source and the rest of the circuit schematic around the original battery cell symbol.

Let's drill into the voltage source symbol. Double-click on the voltage source symbol to bring up its properties window and click on the "Spice Model" button to see the Spice model editor window.

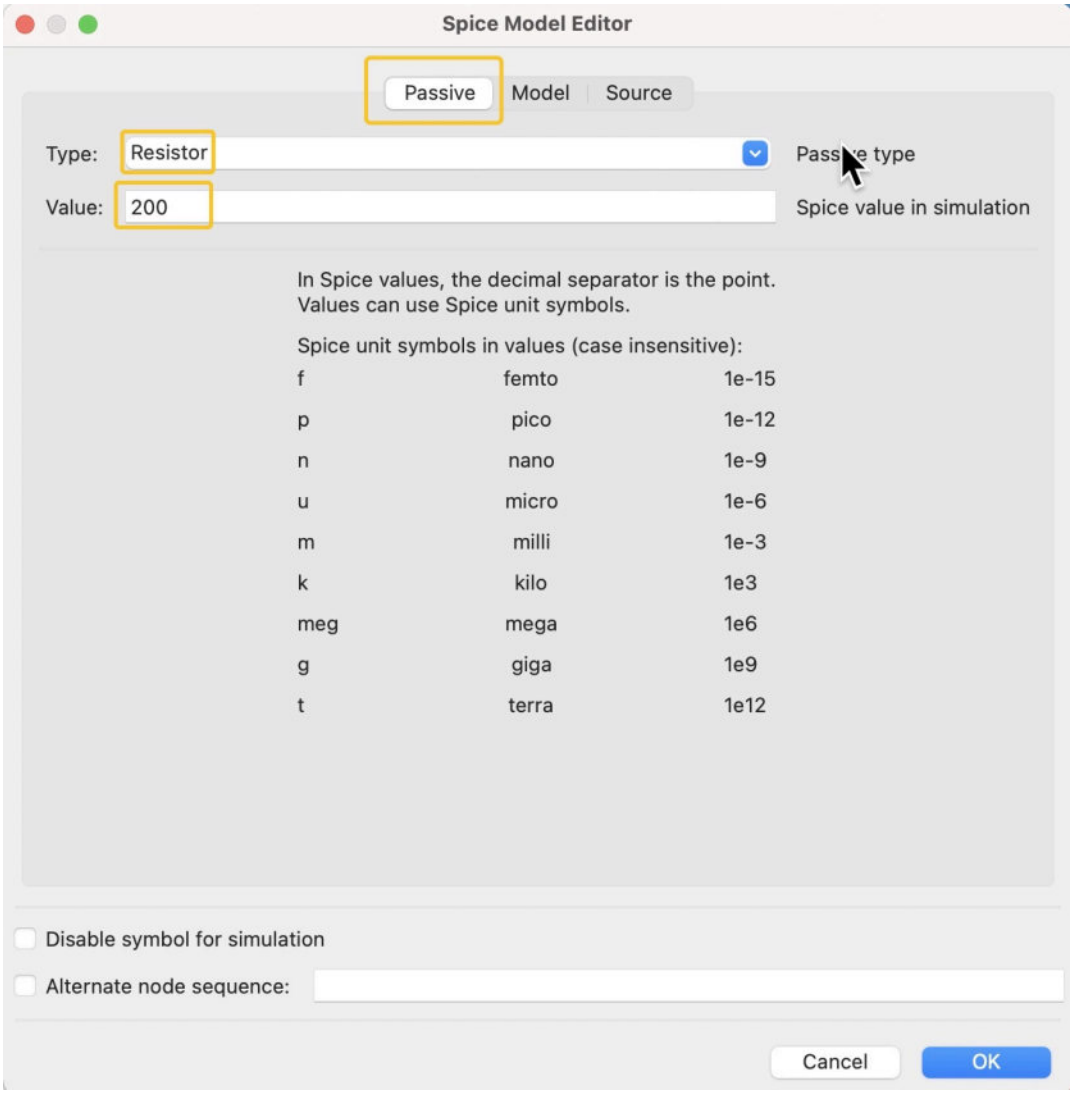
Every symbol in KiCad has a Spice model editor that you can use to set various parameters related to the simulation, including attaching code representing the simulation model of the real-life component.

The Spice model editor window contains three tabs: Passive, Model, and Source. Since I am working with the voltage source symbol, I click on "Source." This is where you can configure your power source. For example, you can set the source to operate in a pulse, sinusoidal or exponential pattern. Of course, you can set it as a simple DC or AC source. Below I have set the voltage source to produce a sinusoidal output with 5V amplitude at 10 Hz (Figure 13.29.1.7).



13.29.1.7: The simulation configuration for the voltage source.

Let's look at the other component in the circuit. First, the resistor, a passive component. Below you can see its simulation spice model (Figure 13.29.1.8):



13.29.1.8: The configuration for the resistor.

The resistor is a passive component, so I have clicked on the "Passive" tab to reveal the relevant options. From the "Type" drop down, I have selected "Resistor", and typed "200" in the "Value" field. If I want to set a 1kΩ resistor, I would type "1k" (notice the Spice unit symbols table below the fields). The Type drop down contains options for "resistor", "capacitor" and "inductor".

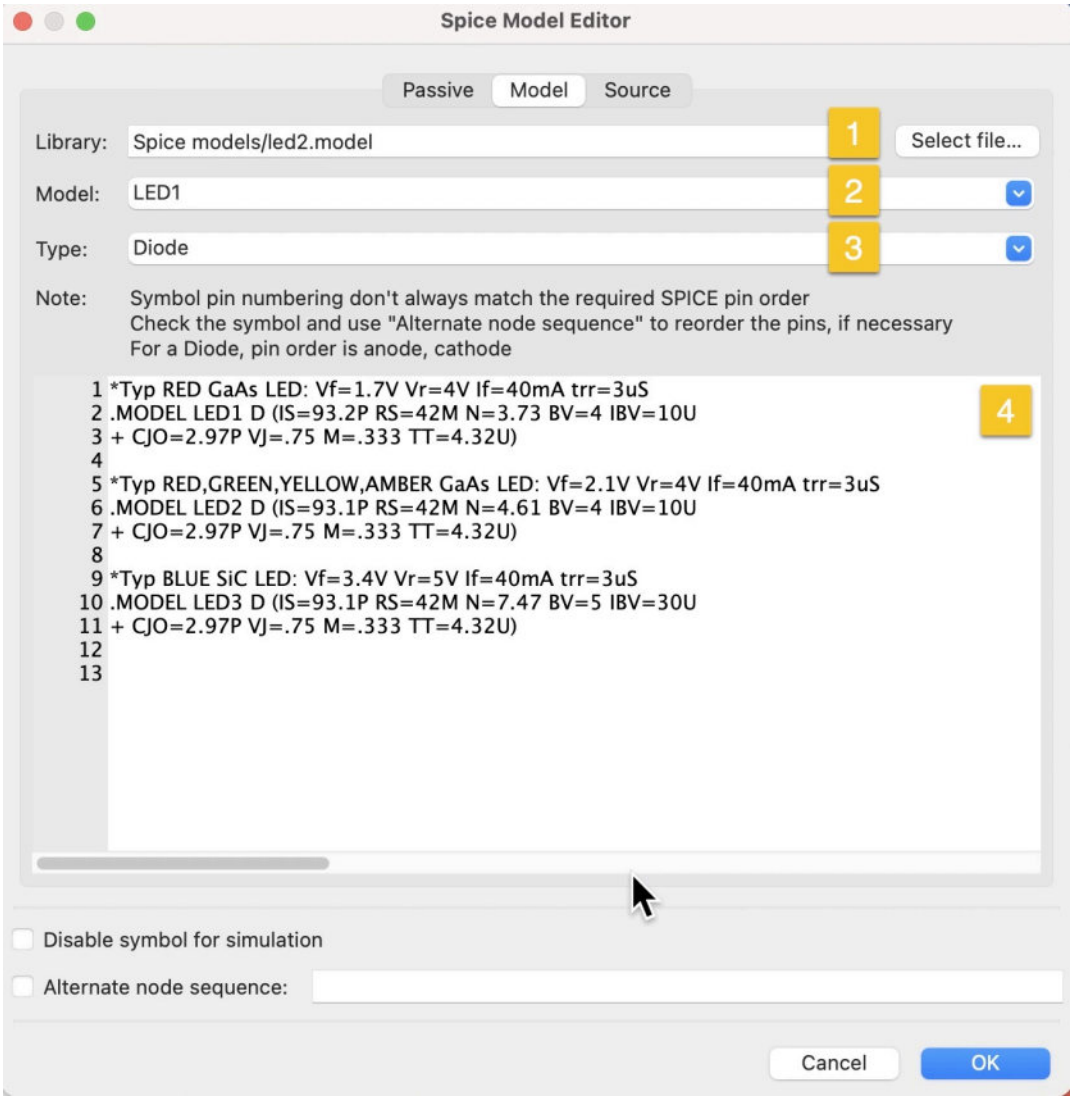
Click OK and OK to return to the editor.

Let's work on the LED next. The LED is a semiconductor component, not passive. For this reason I will need to specify a compatible Spice model. You can find Spice models for components by doing research on the Internet, or by writing them yourself. Component manufacturers often publish Spice models with their products (examples: Analog Devices, Microchip, National Instruments, PSpice, Littlefuse, Diodes). Universities or individuals also publish models on their websites (Examples: Berkeley University, All About Circuits).

Models are distributes as text files like this (here is my source):

```
*Typ RED GaAs LED: Vf=1.7V Vr=4V If=40mA trr=3uS
.MODEL LED1 D (IS=93.2P RS=42M N=3.73 BV=4 IBV=10U
+ CJO=2.97P VJ=.75 M=.333 TT=4.32U)

*Typ RED,GREEN,YELLOW,AMBER GaAs LED: Vf=2.1V Vr=4V If=40mA
trr=3uS
.MODEL LED2 D (IS=93.1P RS=42M N=4.61 BV=4 IBV=10U
+ CJO=2.97P VJ=.75 M=.333 TT=4.32U)

*Typ BLUE SiC LED: Vf=3.4V Vr=5V If=40mA trr=3uS
.MODEL LED3 D (IS=93.1P RS=42M N=7.47 BV=5 IBV=30U
+ CJO=2.97P VJ=.75 M=.333 TT=4.32U)
```

You can save this model in a text file and import it to the KiCad symbol or copy/paste using the spice model editor. For the LED symbol, double-click on it to bring up its Properties window, then click on "Spice Model." Once the Spice Model Editor window is up, click on the Model tab, and copy/paste the model from the sample above into the model text field (Figure 13.29.1.9):



13.29.1.9: Setting the Spice model for the LED symbol.

In the screenshot above, I have used the file method. I saved the model in a text file titled "led2.model" and used the file browser to find it and select ("1" in Figure 13.29.1.9).

The model file contains three individual models. Each has a unique name: "LED1", "LED2", and "LED3". I selected the model I wanted to use using the Model drop-down menu ("2" in Figure 13.29.1.9).

The various parameters that you see in the model definitions, such as "IBV" (Current at Breakdown Voltage), "BV" (Reverse Breakdown Voltage), and "TT" (Transit-time), are described in the Ngspice user manual. You can find these values in a component's datasheet; you can use these values in your models.

Click OK and OK to get back to the editor.

The circuit schematic is now ready for the simulation. In the next segment of this chapter, I will show you how to configure the simulator.

29.2. Configure the simulator

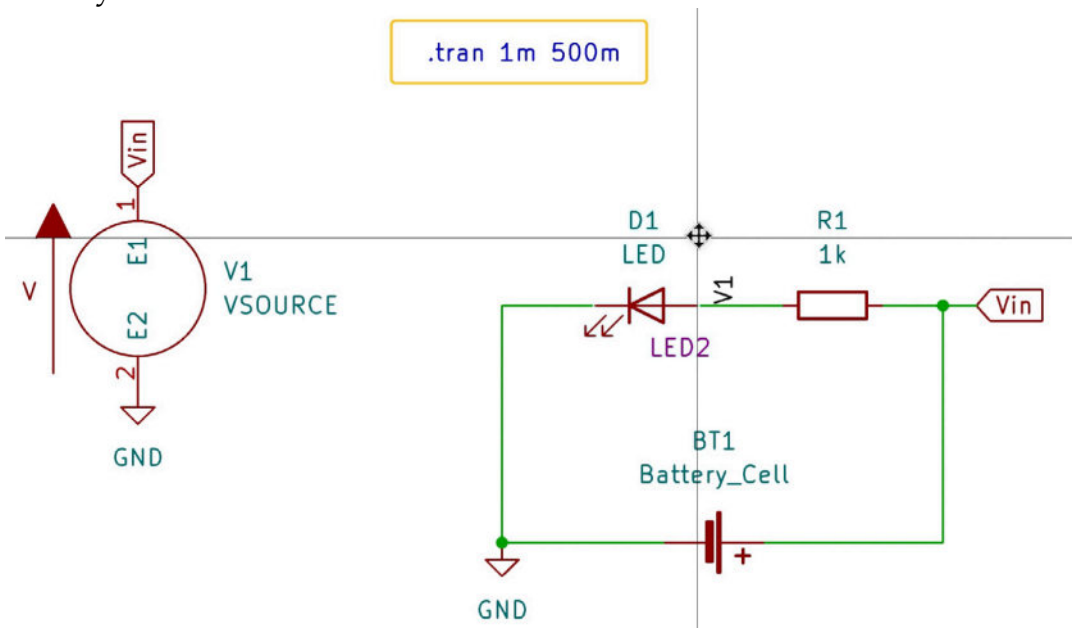
In the previous segment, I prepared my circuit schematic for analysis by the simulator. In this segment, I will configure the simulator. Compared to the work needed to prepare the circuit for the simulation, the simulator's configuration is much simpler.

To configure the simulator, there are generally two steps to complete:

First, set the spice simulation settings. These settings control things such as how many simulation points to calculate, the start and stop frequencies (for AC voltage sources), the behavior of the DC power source(s), the time step, and initial/final times.

Second, set the simulation probes to capture the simulation voltage and current data.

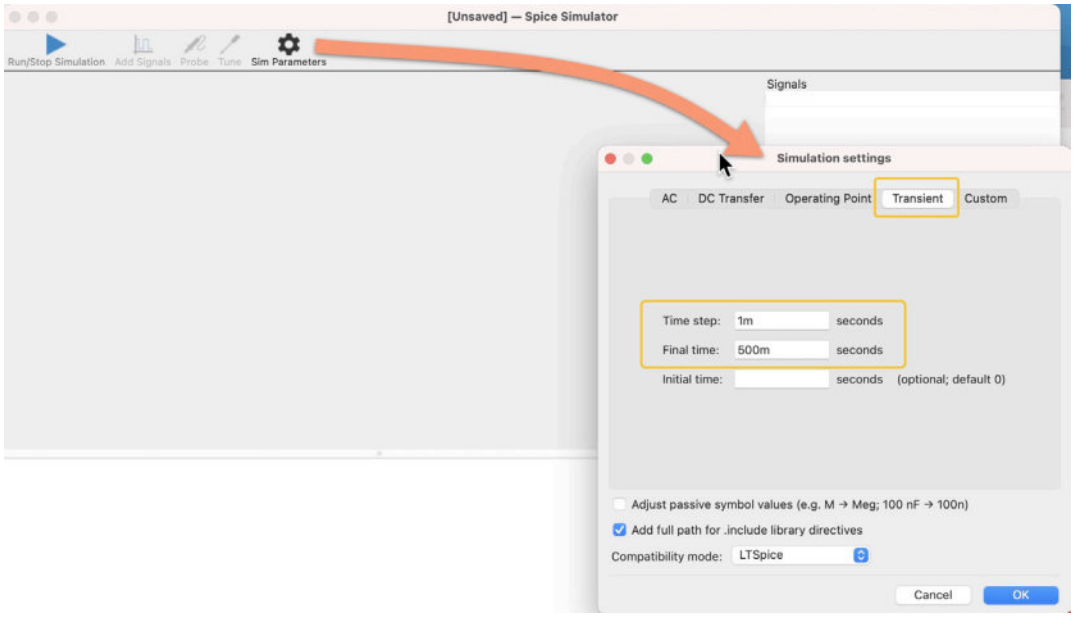
A convenient way to configure the simulator so that the settings persist is to create a text item above the circuit schematic and type the configuration string there. In Figure 13.29.2.10 (below) you can see the configuration string inside the yellow box:



13.29.2.10: The simulation configuration.

The simulator will detect the presence of the configuration string and read it, saving you the hassle of manual configuration every time you open Eeschema.

Let's see the impact that the configuration string has on the simulator. Bring up the simulator window by clicking on Inspect, Simulator. In the simulator window, click on the Sim Parameters button (Figure 13.29.2.11).



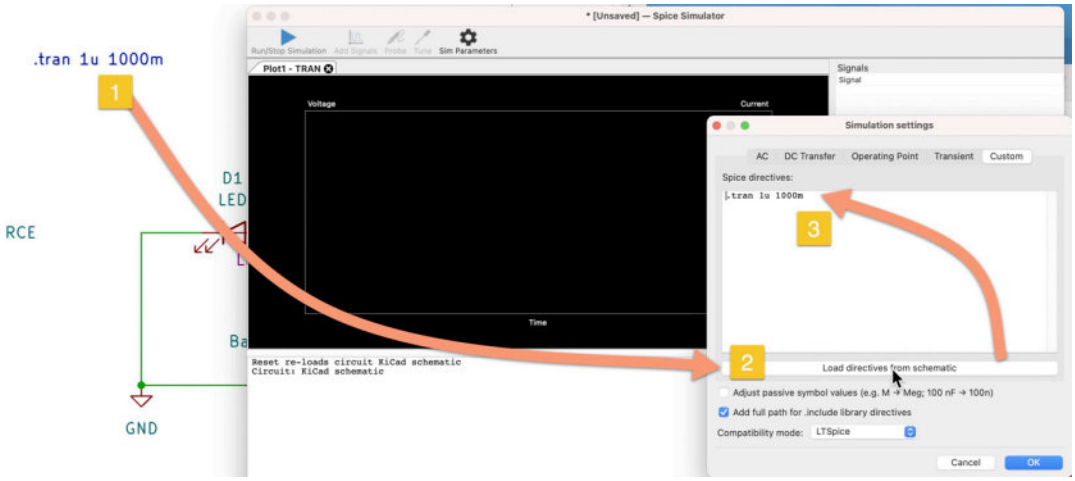
13.29.2.11: The simulation settings.

The simulation settings window will appear. The simulator has already read the information in the settings string from Figure 13.29.2.10 and has populated the “Time step“ and “Final time” text boxes in the Transient tab. Compare the values in the text boxes with those in the configuration string and notice that they match.

Before you close the simulation settings window, browse the contents of the other tabs to see some of the other available options. Click on the “Custom” tab to see the configuration string copied there from the schematic.

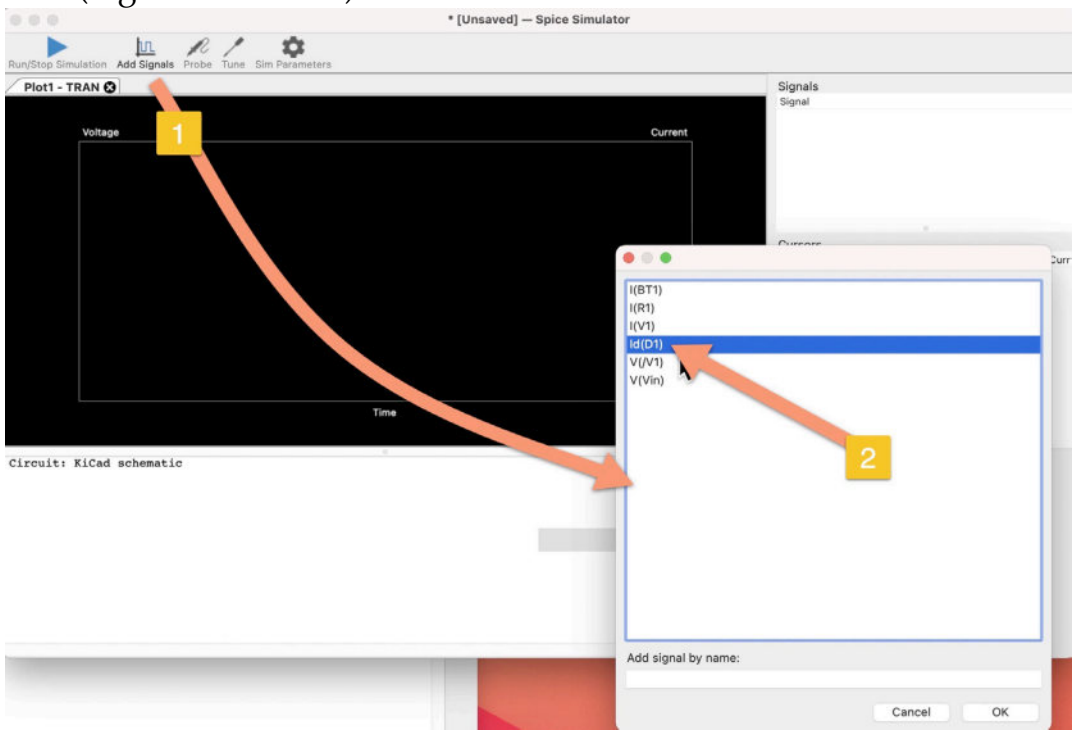
My simulation is now configured to start from time 0 ms to 500 ms, at a step of 1 ms. In total, the simulator will calculate 500 points of whichever values to set next. As the number of points increases, so does the amount of time the simulator will need to finish the work, so choose reasonable values for the simulation time.

If you make a change to the text string in the editor (“1”, below), you can update the simulator configuration by clicking on the button “Load directives from schematic” (“2”, below), under the Custom tab (Figure 13.29.2.12):



13.29.2.12: Updating simulation directives from the schematic.

I am almost ready to start the simulation. The last task to complete is to indicate the value(s) I'd like to capture to the simulator. Say I want to capture the current that is coming out of the LED. For this, I will add the signal (Id for D1) to the list of captured signals. Click on the "Add signals" button to bring up the signals window in the simulator window. Then select "Id(D1)" and click OK (Figure 13.29.2.13).



13.29.2.13: The simulator will capture the signal "Id(D1)".

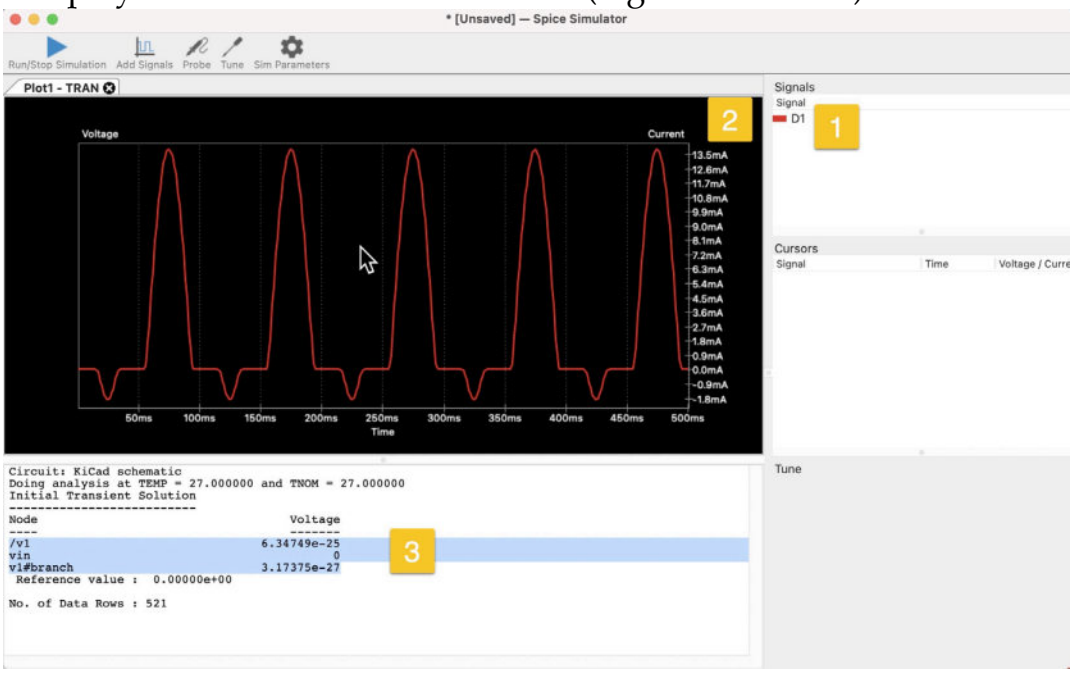
You may choose more than one signal if you wish.

The simulator is now ready to run. Let's do that in the next segment of this chapter.

29.3. Simulate

Time to simulate!

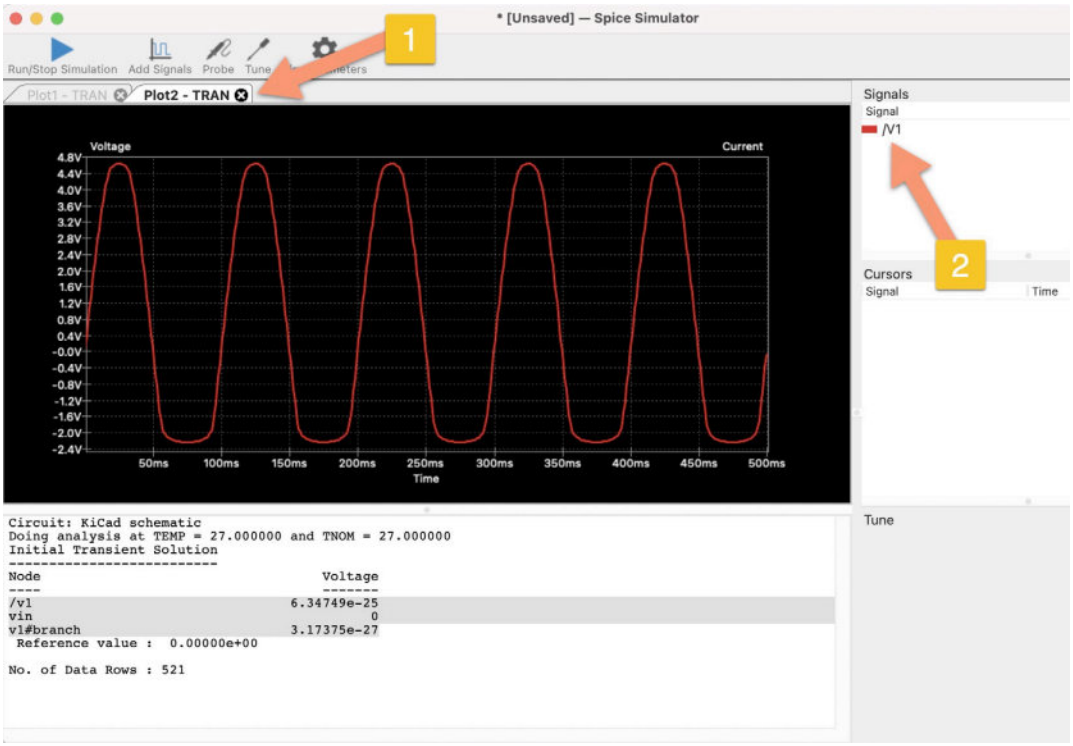
In Eeschema, open the simulator window (Inspect —> Simulator). Click the blue play button to run the simulation (Figure 13.29.3.14).



13.29.3.14: The simulation results.

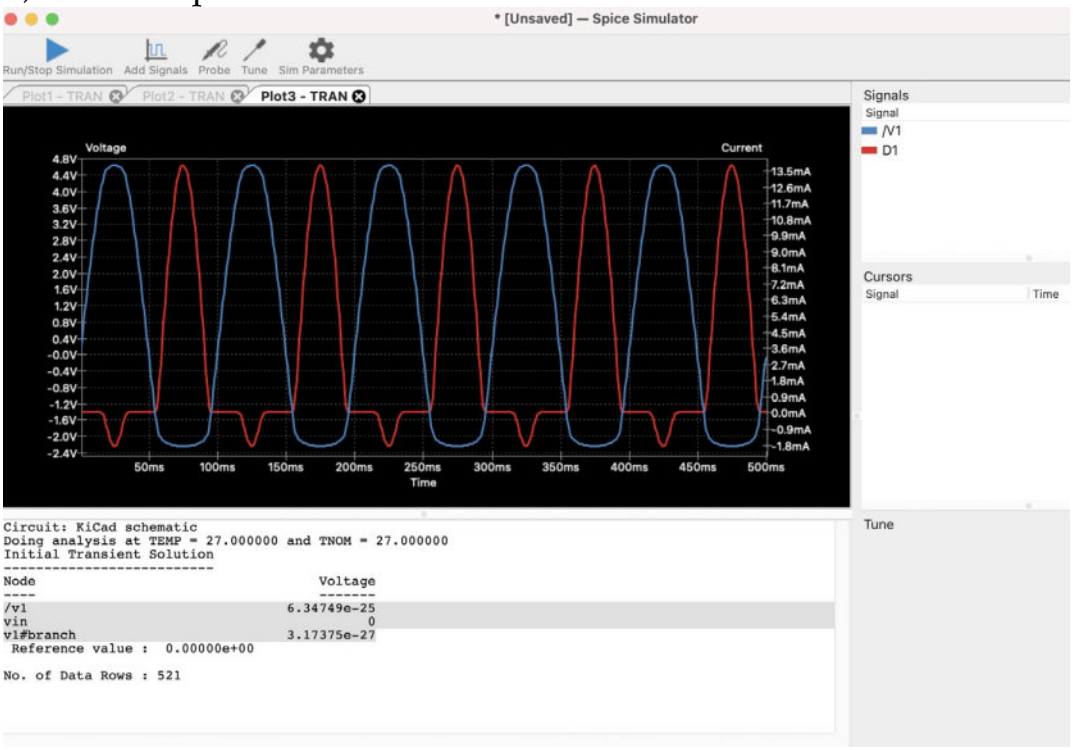
My simulation only contains 500 points, so it is very quick to finish. In the screenshot above, you can see the simulated signal (1), the plot (2), and the results (3).

You can create additional plots as needed. With the simulation window active, click ”File,” then “New Plot.” A new tab will appear in the simulator window that contains a blank plot. Let’s add a new signal. With the new plot tab selected, click on “Add Signals” and select “V(V1)” from the list. Click OK. The simulator will immediately draw the plot for the new signal (Figure 13.29.3.15):



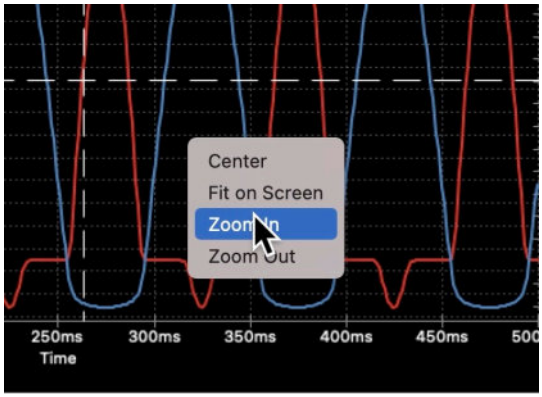
13.29.3.15: Plot for signal V1 in a new tab.

Of course, you can plot more than one signal on a single plot. Let's try one. Create a new plot, and assign it signals "Id(D1)" and "V(V1)" (you can multiple-select by holding down the Command or Control key as you click on a row). The new plot looks like this:



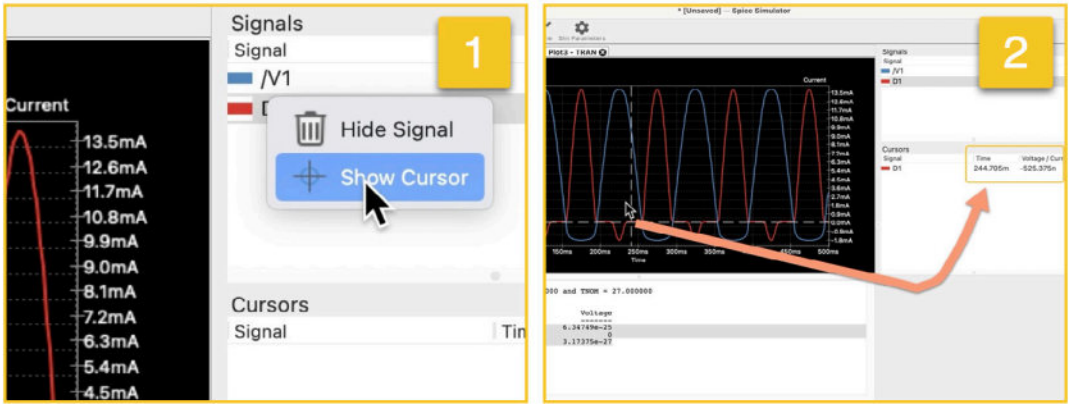
13.29.3.16: A plot with two signals.

The simulator plots are interactive. You can use your mouse and its scroll wheel to zoom in and out and pan.



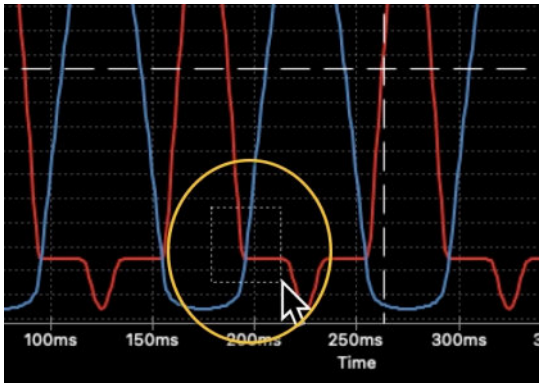
13.29.3.17: Context menu options for the plot.

You can also set a cursor on a signal so that you can look at values at specific points on a signal's plot:



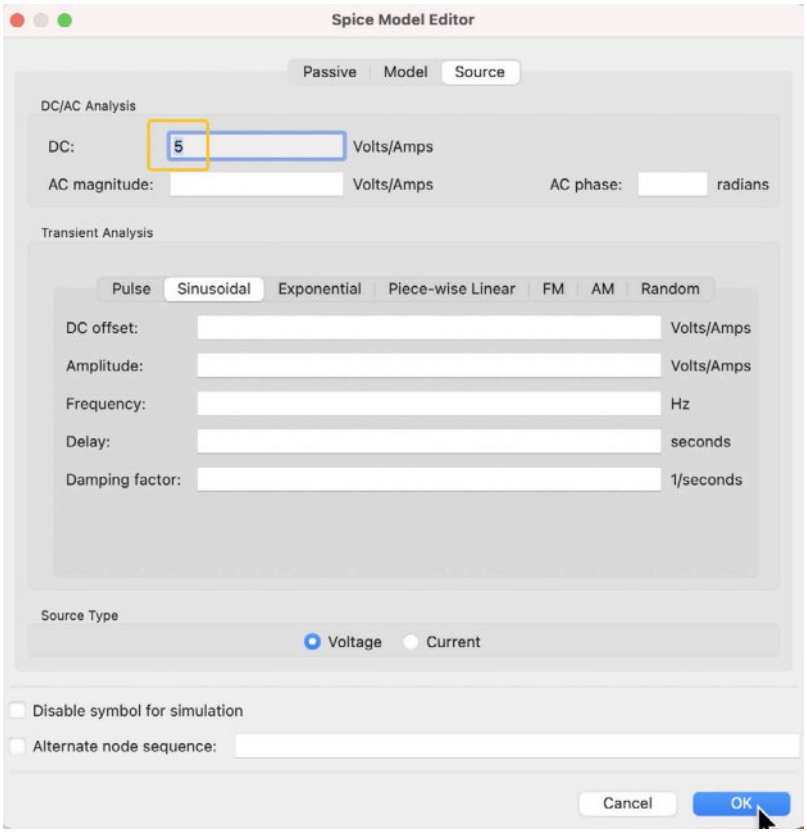
13.29.3.18: Using the cursor on signal D1.

You can also zoom into a specific region of the plot by drawing a rectangle with your mouse:



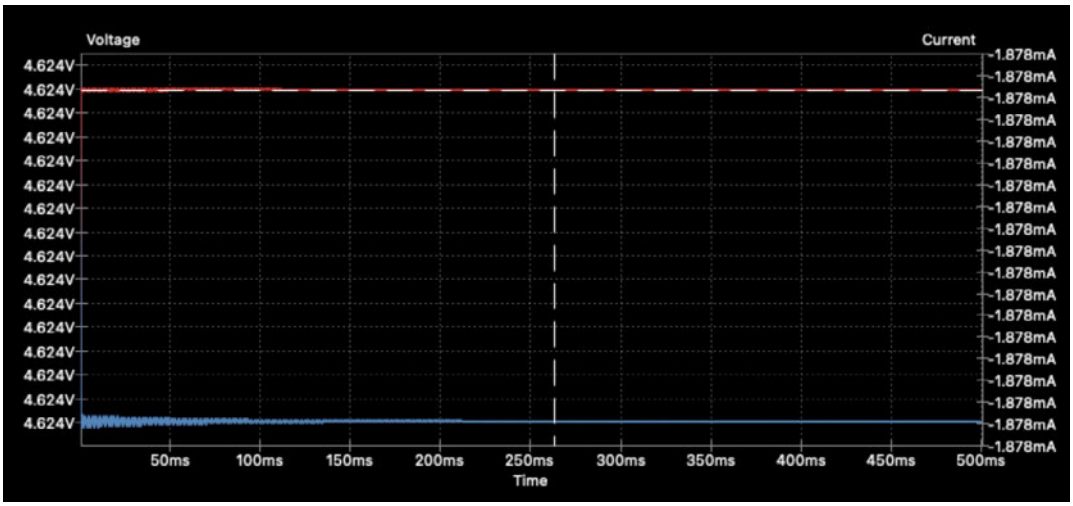
13.29.3.19: Zoom in to a region of the plot.

Let's make a change to our circuit and rerun the simulation. For the new experiment, I will change the voltage source configuration to be a 5 V stable source. Make the change in the voltage source symbol like this (see DC field in "Source"):



13.29.3.20: Changed the voltage source to 5 V stable.

Return to the simulator window and click on the blue play button to run the simulation. No need to change any of the simulator settings. The new Plot3 looks like this:



13.29.3.21: A new plot with the new voltage source settings.

With these basic simulator skills, you can try out a variety of scenarios. For example:

- What would happen to the voltage and current on the diode if the source voltage is 3V instead of 5V?
- What if it is a small AC voltage at 10Hz?

- What if you used a different type of LED?
- What if you double the value of the resistor?

These questions and many more are now easy to answer with the help of the simulator.

30. Import a KiCad 5 project

If you have KiCad 5 project that you would like to use with KiCad 6, you can import them. However, remember that KiCad 6 has introduced several significant changes in the project, schematic, layout, and library file formats. Because of these differences, importing a KiCad 5 project to KiCad 6 is possible but not straightforward.

This chapter will show you how you can import a KiCad 5 project into KiCad 6 using an example.

First, a review of the file format and project organization changes. In Figure 13.30.1 (below) you can see the project directories and file organisation of a KiCad 5 project (left, 1) and KiCad 6 project (right, 2).

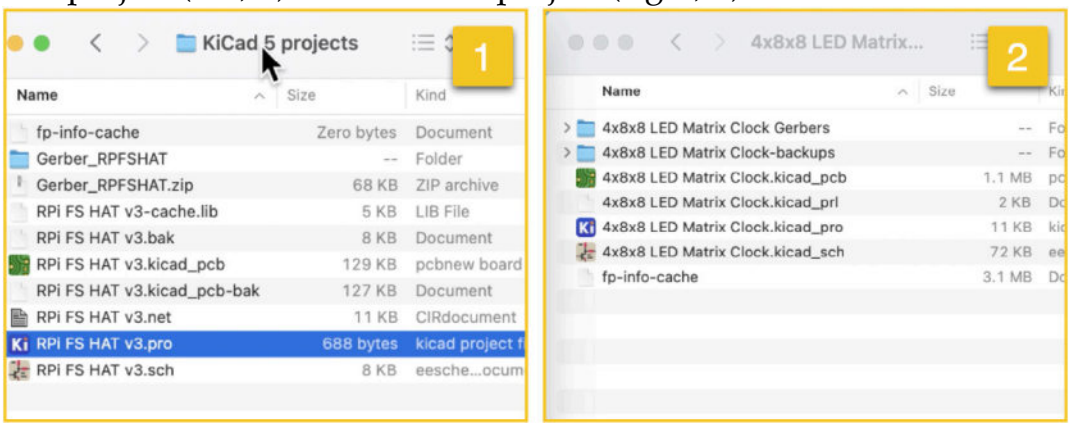


Figure 13.30.1: Project directories and organization. KiCad 5 (left) KiCad 6 (right).

In KiCad 5, the project (".pro") file and schematic (".sch") file do not use the KiCad 6 new S-Expressions format. Still, in KiCad 5, the layout (".kicad_pcb") file does use the S-Expressions format, making it easy to import into KiCad 6.

You can see examples of the content of those files below (Figure 13.30.2):

Figure 13.30.2: KiCad 5 file format examples.

In KiCad 6, all three files (project, schematic, and layout) use the S-Expressions format.

To facilitate the import of the KiCad 5 project to KiCad 6, KiCad 6 has an import utility that will handle the conversion of the legacy project and schematic files to use the new S-Expressions file format. The importer will also try to find appropriate matches for schematic symbols in the originating project. When the importer can't find suitable symbols, it will ask you to find a replacement manually.

Let's look at an example.

In Figure 13.30.1, you can see my old KiCad 5 project (left, 1). I will import this project into KiCad 6.

In Figure 13.30.1, you can see my old KiCad 5 project (left, 1). I will import this project into KiCad 6.

The importer "hides" in the regular Open option under File, in the KiCad 6 project window (Figure 13.30.3):

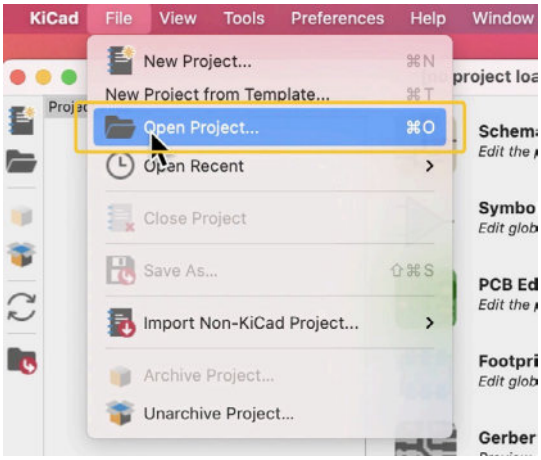


Figure 13.30.3: Open a KiCad 5 project as you would any KiCad project.

KiCad will ask for the location of the project. Navigate and select the file with the ".pro" extension in the KiCad 5 project directory:

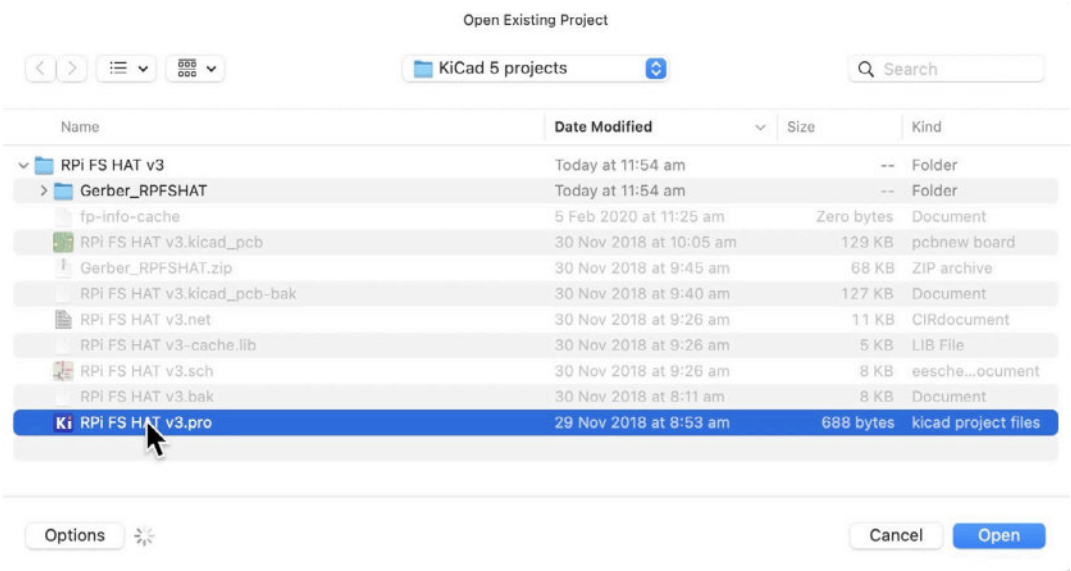


Figure 13.30.4: Open the ".pro" file.

Click "Open." The KiCad project window will show the project contents:

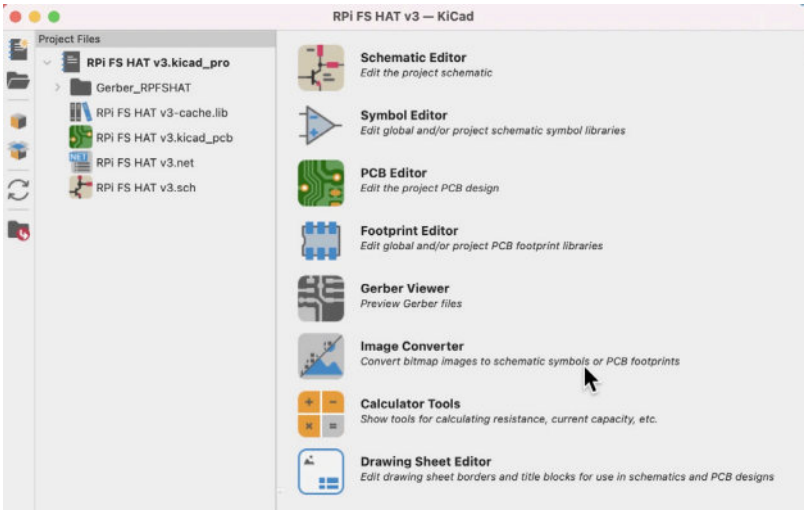


Figure 13.30.5: My old KiCad 5 project in the KiCad 6 project window.

Next, let's open Eeschema and check if there is any manual work left to be done. Click on the Schematic Editor button to open Eeschema. The project rescue helper window will appear. The window contains information about the symbols that the importer has updated and requests your approval (Figure 13.30.6).

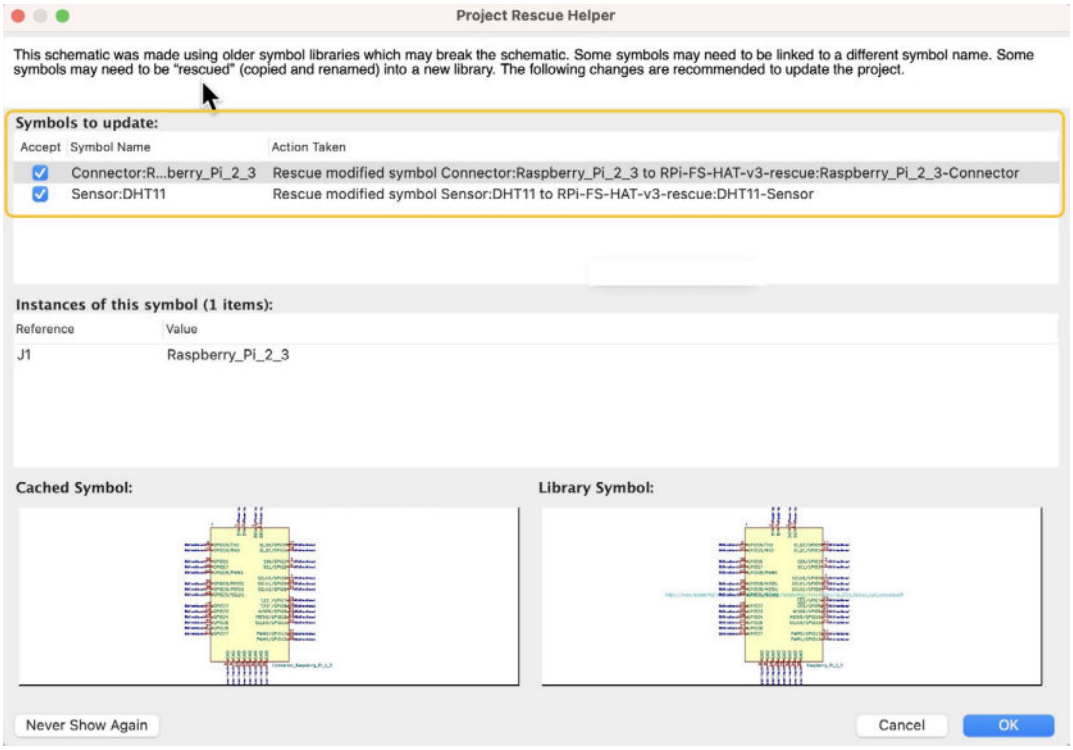


Figure 13.30.6: The importer has done all the work; it is prudent to double-check.

It is also possible that the importer has not been able to find a suitable symbol replacement, in which case it will ask you to find one. In the example above, I double-check that the symbol updates are appropriate. To do this, I use the Helper tool's information in the list "Symbols to update." Click on a row to see more details in the lower part of the window, including the symbol graphics, so that you can visually confirm the updates.

I accept the updates and click OK. The schematic editor appears and contains my updated project schematic:

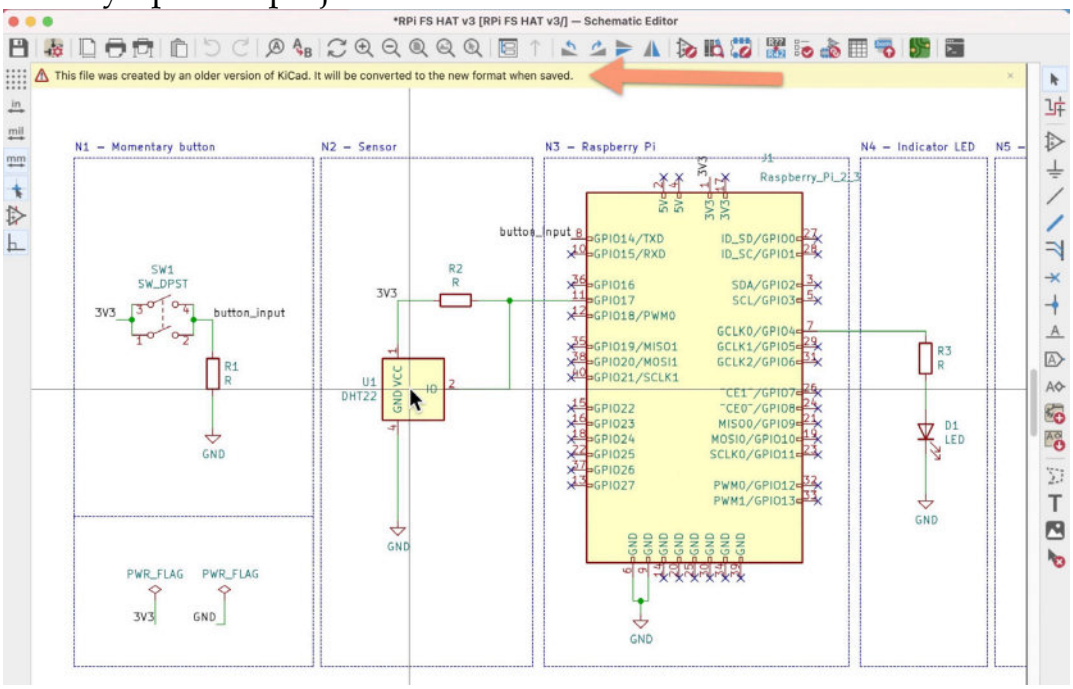


Figure 13.30.7: The updated project schematic.

Take the time to visually inspect the schematic in the editor and make any changes necessary as you would with any KiCad 6 schematic. Notice the warning band at the top of the editor. This indicates that the conversion is not yet complete. It will be done once you save the schematic. Type “Cmd-S” or “Ctr-S” to save the schematic to disk and complete the conversion. The warning will go away.

Return to the KiCad project window and also look at the project directory contents. Notice that there is a new file with the “.kicad_sch” extension. This file contains the schematic editor data using the new KiCad 6 file format (Figure 13.30.8).

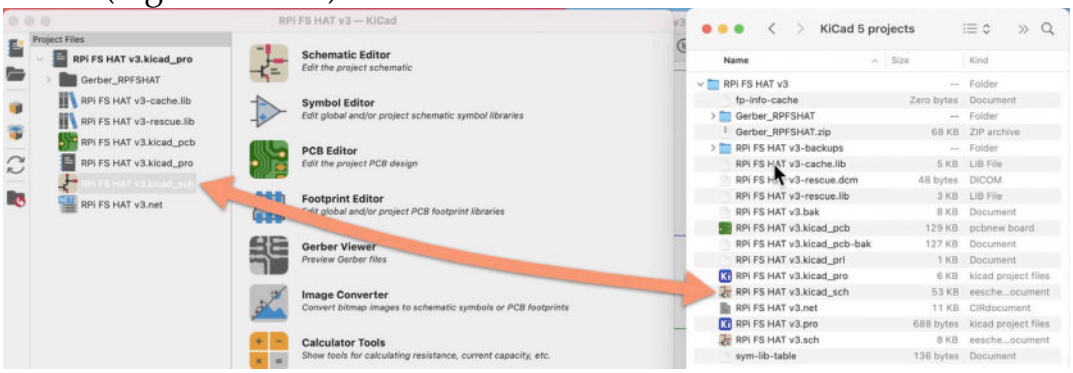


Figure 13.30.8: The new KiCad 6 schematic file.

The original “.sch” file is still in the project directory. You can delete it if you wish or keep it in case you need to use it later. The same applies to the old “.pro” file, replaced by the new “.kicad_pro” file.

Continue with the layout editor. Click on the Pcbnew button in the KiCad project window or Eeschema. The layout editor will appear (Figure 13.30.9):

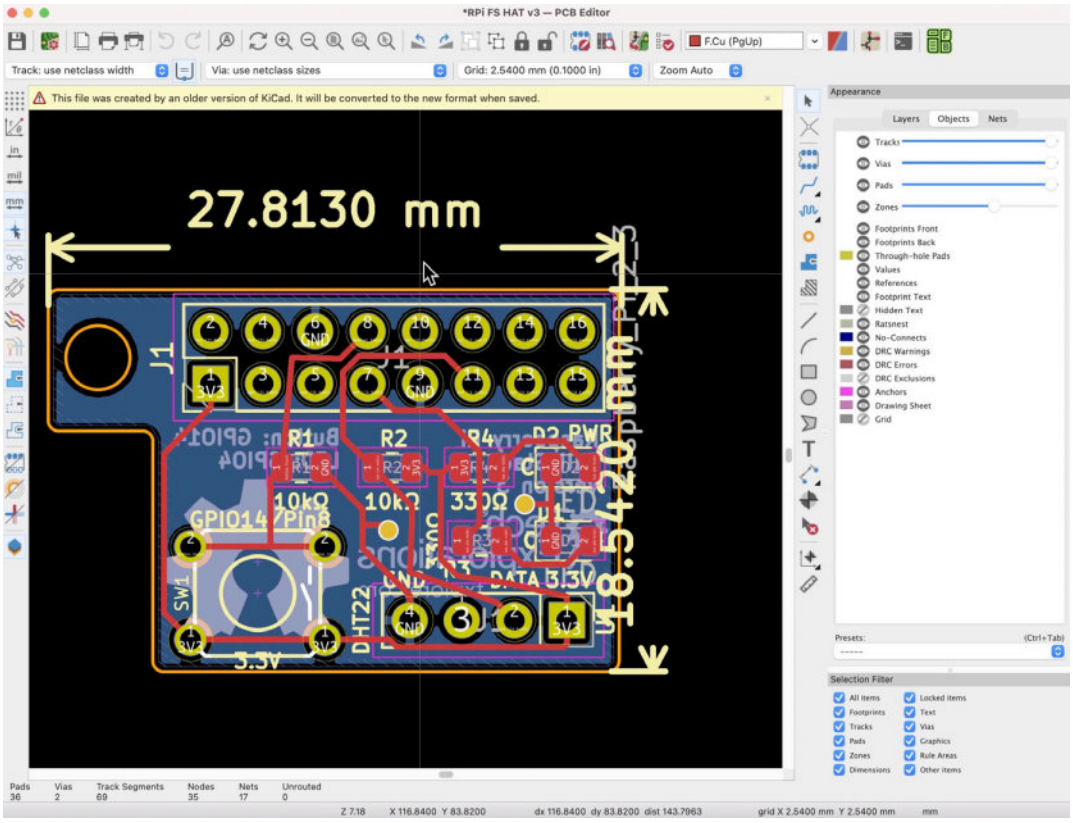


Figure 13.30.9: The KiCad 5 layout file in KiCad 6.

Pcbnew does not show a project rescue helper, only a warning that the conversion will be finished when the file is saved. Save the file to dismiss the notification and complete the conversion.

Because the layout editor depends on the schematic editor if you see problems (like incorrect layouts), the first place to look for clues is the schematic editor. Other issues that I have noticed in Pcbnew when importing Kind 5 projects are related to things like the grid size, text sizes, and colors. In most cases, you will be able to resolve all issues within a few minutes.

31. KiCad project templates

KiCad project templates can save you a lot of time when you start a new project. With a KiCad project template, your new project is pre-configured. It has its schematic layout editors already populated standard components that can form the basis you can build on.

You can choose one of several templates that KiCad ships with or create yours. You can create a user template from any of your existing KiCad 6 projects. In this chapter, I'll show you both options.

Template locations

KiCad system and user template are stored in specific directories. You can find and/or change those directories in your instance of KiCad in the "Configuration Paths" window (Figure 13.31.1). While in the KiCad project window, click on Preferences, Configure Paths.

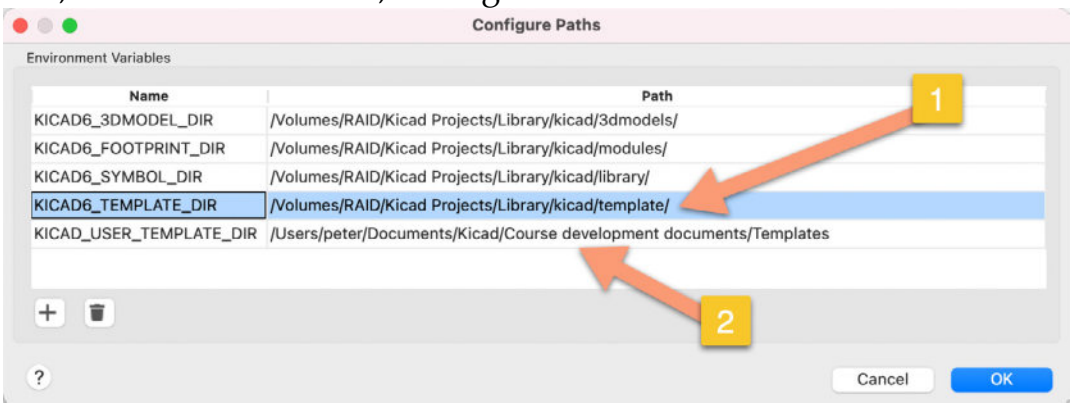


Figure 13.31.1: The Configure Paths window.

Notice that there are two template directories:

- KICAD6_TEMPLATE_DIR, where you can find the system templates.
- KICAD_USER_TEMPLATE_DIR, where you can store your custom templates.

The source of the system project templates is located on Gitlab. You can periodically look at Gitlab for new or updated templates and copy them to your KiCad instance system template directories.

Later in this chapter, I will show you how to create a custom project template. I will save mine in the specified KICAD_USER_TEMPLATE_DIR directory.

31.1. Using a system project template

A “System template” is a template that KiCad provides. For example, system templates contain Arduino Mega, Beaglebone Black, Raspberry Pi, or STM32 projects. Let’s look at an example.

I will create a new project that uses the Arduino Mini system project template.

Start KiCad and open the main project window. From the File menu, select “New Project from Template…” (Figure 13.31.1.2).

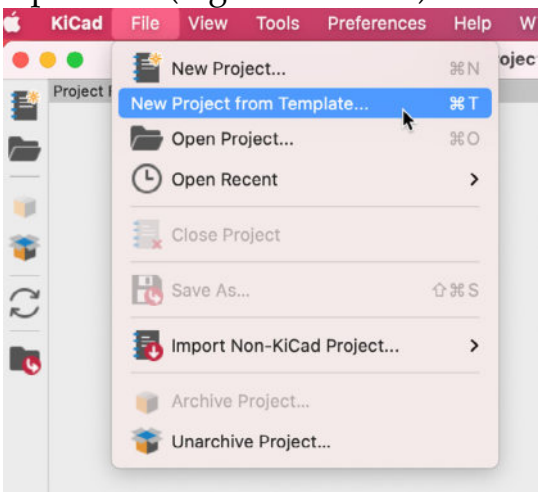


Figure 13.31.1.2: Create a new project from a template.

The Project Template Selector window will (Figure 13.31.1.3). Select the “System Templates” tab (1), use the scroller to pan the project templates left and right, and find the Arduino Mini template. Click to select it (2).

In the information pane (3), you will see a description of the template, including a view of the footprints and schematics that it contains. The information you see on the information page (3) is a simple HTML file. The creator of the template can provide as much or as little information as they want.

Notice the location of the selected template in the folder text box (4).

Click OK (5) to continue.

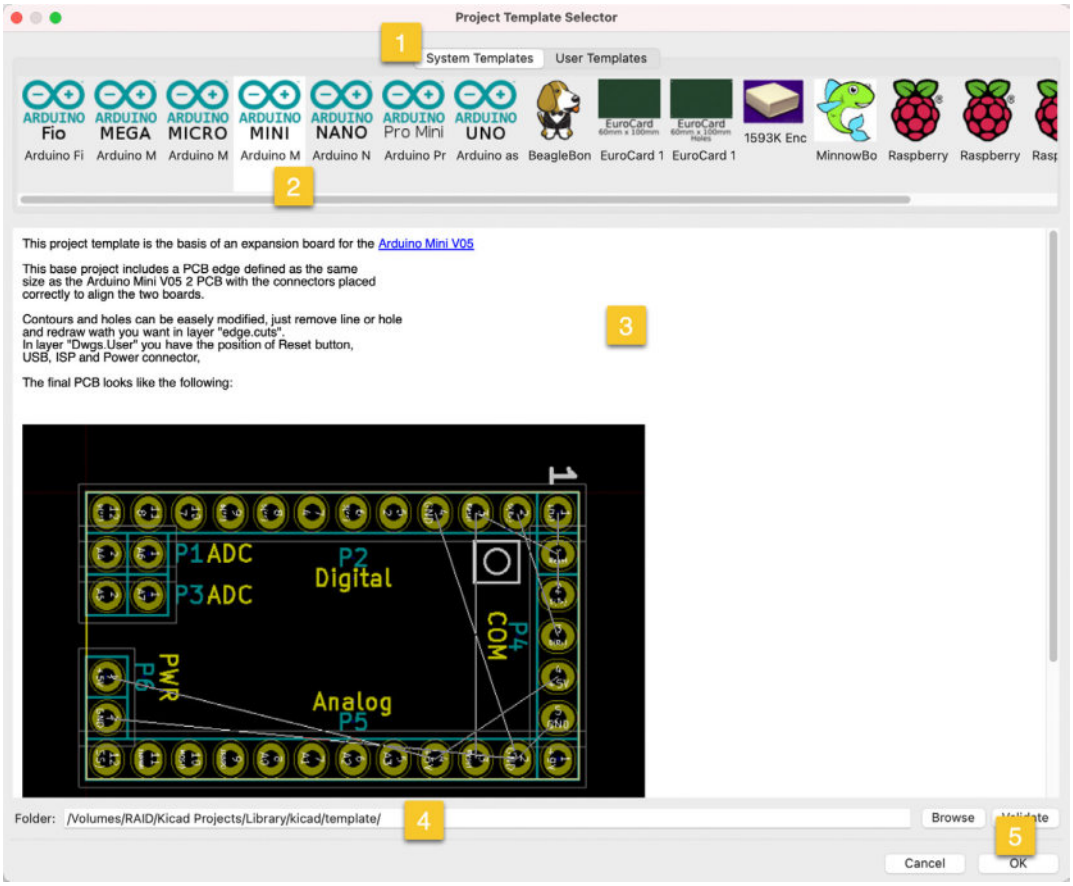


Figure 13.31.1.3: The project template selector.

KiCad will ask you to choose a location for the new (Figure 13.31.1.4). Give the project a name (1), check the new folder box (2), and click “save” (3).

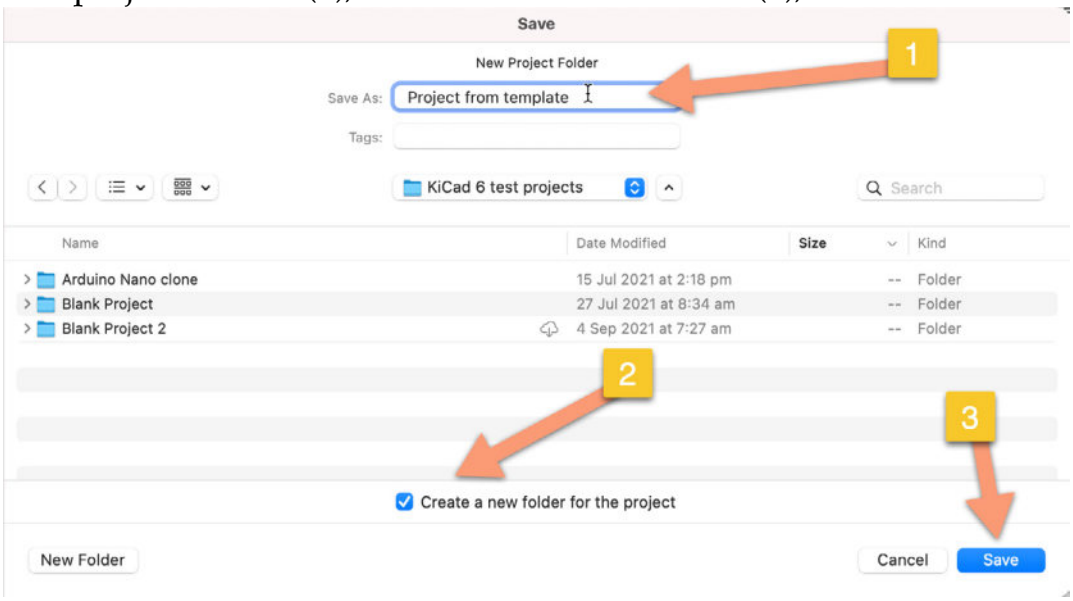
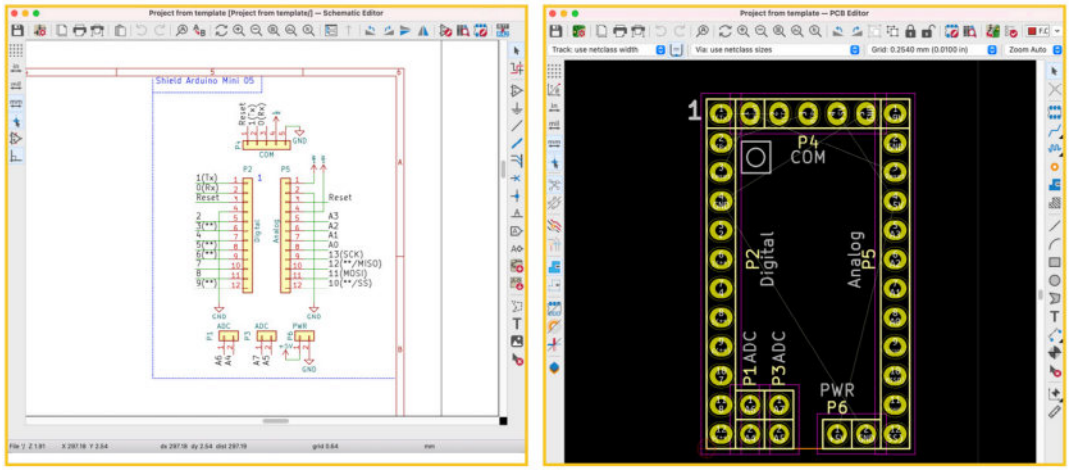


Figure 13.31.1.4: The location for the new project.

The new project, using the settings and starting schematic and layout from the project template, is ready.

Open the schematic and layout editors to see what the project looks at the moment (Figure 13.31.1.5):



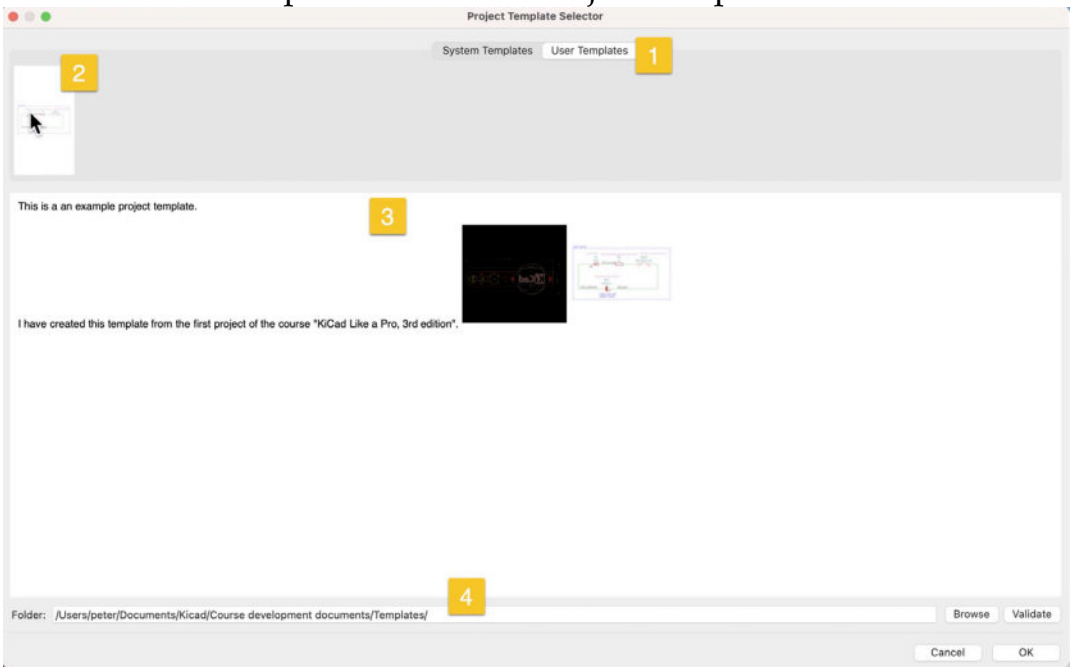
Eeschema Pcbnew

Figure 13.31.1.5: The new project, with the two editors populated.

As you can see, the new project is not blank. It contains the starting schematic and layout as inherited by the Arduino Mini template.

31.2. Create a user project template

It is possible to create custom user templates from your existing KiCad projects. When you have created a user project template, it will be available for use in the User templates tab of the Project Template Selector window:

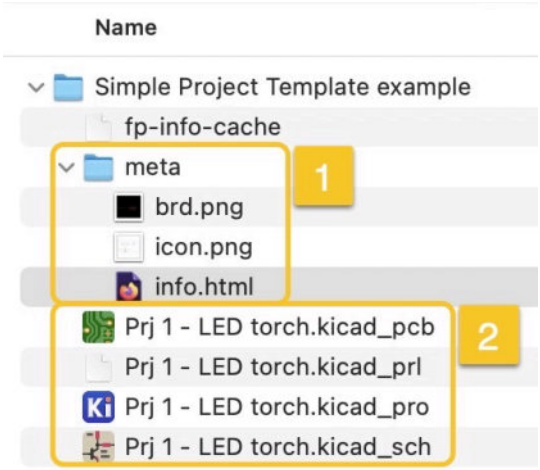


13.31.2.6: The User Templates tab.

In the example above, I have selected the User Templates tab (1) to see a single user project template (2). The information pane (3) contains a simple HTML document with rudimentary formatting.

Also, notice the location of the template (4).

Let's look inside the user template directory for this template. Doing so will give us clues about the necessary template files and their structure. See Figure 13.31.2.7 (below).



13.31.2.7: The contents of a user project template directory.

Two groups of files compose a user project template:

1. Project meta.
2. Project files.

The project metafiles are stored inside a folder with the name "meta." At a minimum, the meta folder must contain "icon.png" and "info.html." The PNG file is used to create an icon for the template (see item "2" in Figure 13.31.2.6). The PNG image must have a resolution of 64x64 pixels.

The HTML file composes the contents of the information pane (see item "3" in Figure 13.31.2.6). You can include additional files in the meta folder if you wish. For example, I have added a file named "brd.png" to display a small image of the layout in the information window. I use the file "brd.png" in my HTML code.

The project files are the three primary files of any KiCad project:

- The project file, ".kicad_pro".
- The schematic file, ".kicad_sch".
- The layout file, ".kicad_pcb".

You can get those files from your source KiCad project and copy them into the project template folder. To create this project template for this example, I simply copied the files I list above from my LED torch project. I did not make any changes to those files.

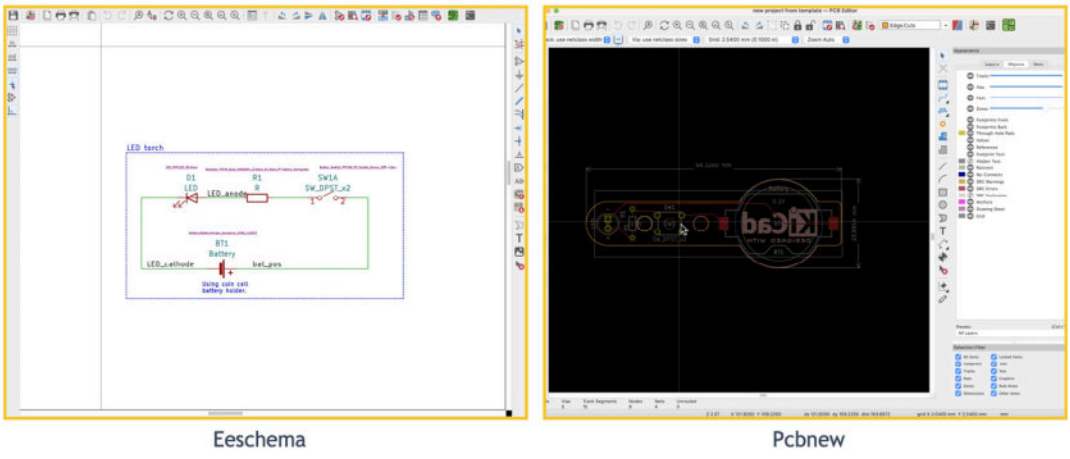
Below you can see the contents of the HTML file:

```
<html>
<head></head>
<body>
<p>This is a an example project template.</p>
<p>I have created this template from the
first project of the course ”KiCad Like a
Pro, 3rd edition”.</p>
<br>
<img src="brd.png">
<br>
<img src="icon.png”>
</body>
</html>
```

The paths are all relative to the location of the HTML file. You can also use regular HTTP links to external web pages.

Once you have created the new project template and stored it in the appropriate location, you can use it. In the KiCad project window, click File, and select “New Project from Template…“. The Project Template Selector will appear (see Figure 13.31.2.6 above). Select the “User Templates” tab, and click on the project template icon. Then click “OK” and choose a name and location for the new project.

Your new project is now ready to use, with its schematic and layout editor populated with the symbols and footprints inherited from the template (Figure 13.31.2.8):



13.31.2.8: The new project, with Pcbnew and Eeschema contents inherited from my user project template.

System and user project templates can save you a lot of time when starting new projects. I often capture my new projects when they are early in their development process to use them as templates in the future.

32. Archive/unarchive and share a project

KiCad has a project archiving feature that allows you to create a self-contained ZIP archive of your project. Once you have archived your KiCad project, you can store it for later use or share it with other people. The project archiver will keep all project dependencies (like symbols or footprints) in the archive so that your collaborators (or you) will not need to spend any time trying to fix library issues.

In this chapter, I will show you how to archive and un-archive a project using an example. I will archive one of my projects on my Mac OS computer and open it on my Windows 10 virtual machine.

First, open a KiCad project. With the KiCad project window showing, click on File, then Archive Project (Figure 13.32.1).

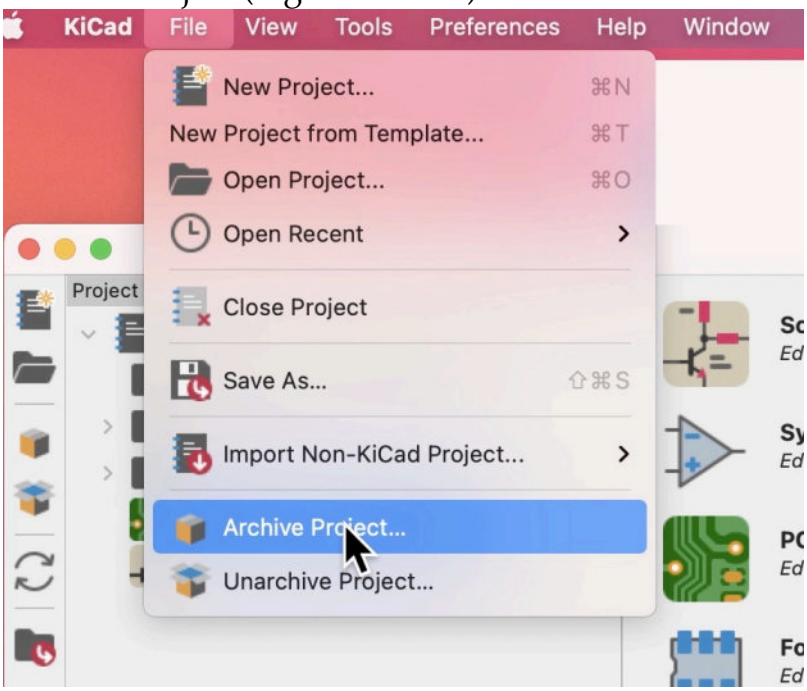


Figure 13.32.1: Archive a KiCad project.

KiCad will ask for a name and save location for the archive, and then create a new ZIP file (Figure 13.32.2):

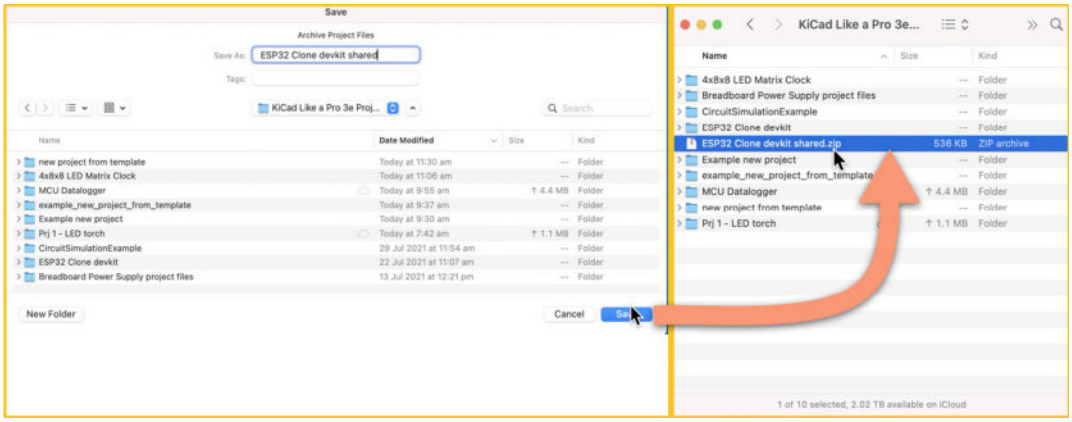


Figure 13.32.2: New archive is created.

With the new archive ZIP file available, I can copy it to my Windows 10 virtual machine (or a collaborator that used KiCad 6 on Windows). See “1” in Figure 13.32.3 below:

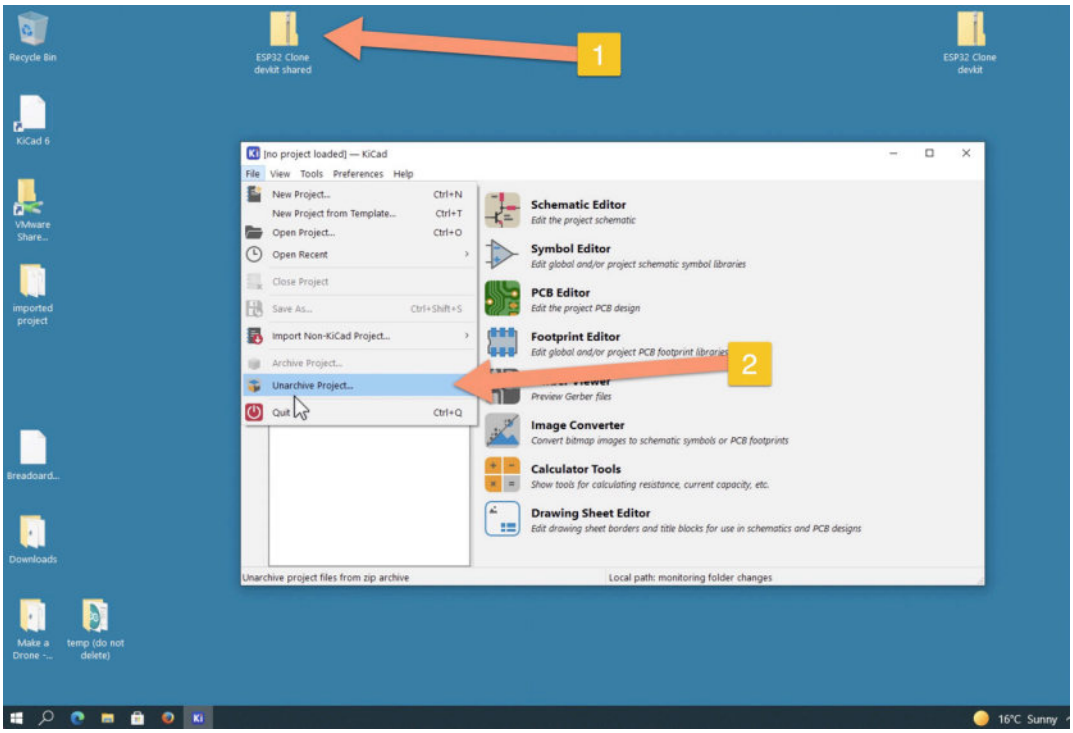


Figure 13.32.3: Unarchviging this archive in KiCad 6 running on Windows 10.

In the KiCad 6 project window, click on File and “Unarchive Project…”. KiCad will ask you to select the archive file, so use the file browser to locate the ZIP file and click Open (Figure 13.32.4).

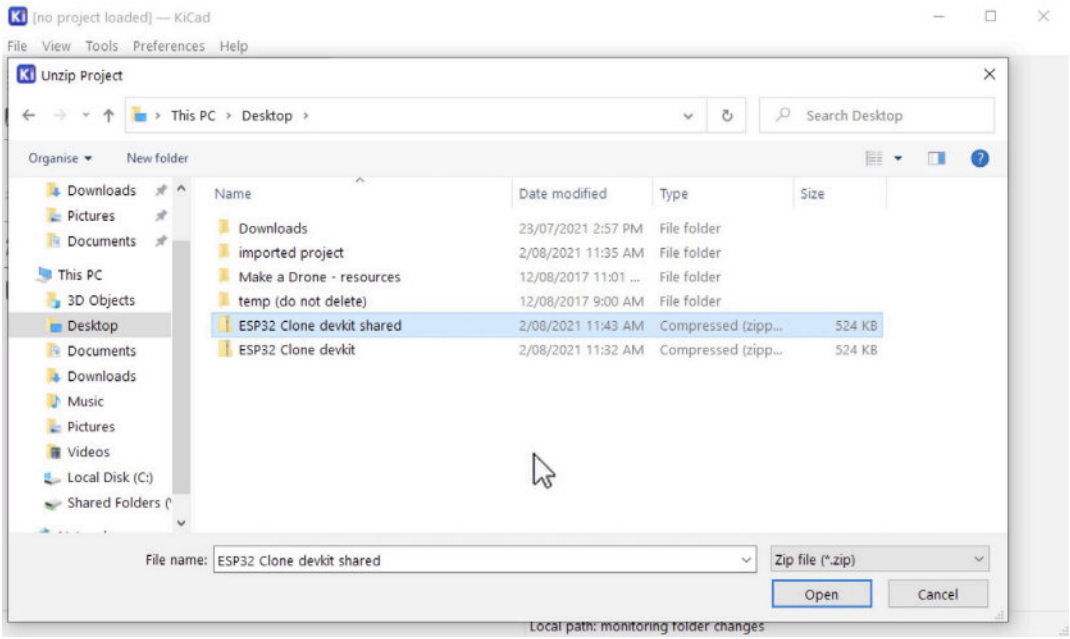


Figure 13.32.4: Navigating to the archive ZIP file.

Next, KiCad will ask you for a destination location for the unarchived project. I'll save mine to the desktop (Figure 13.32.5):

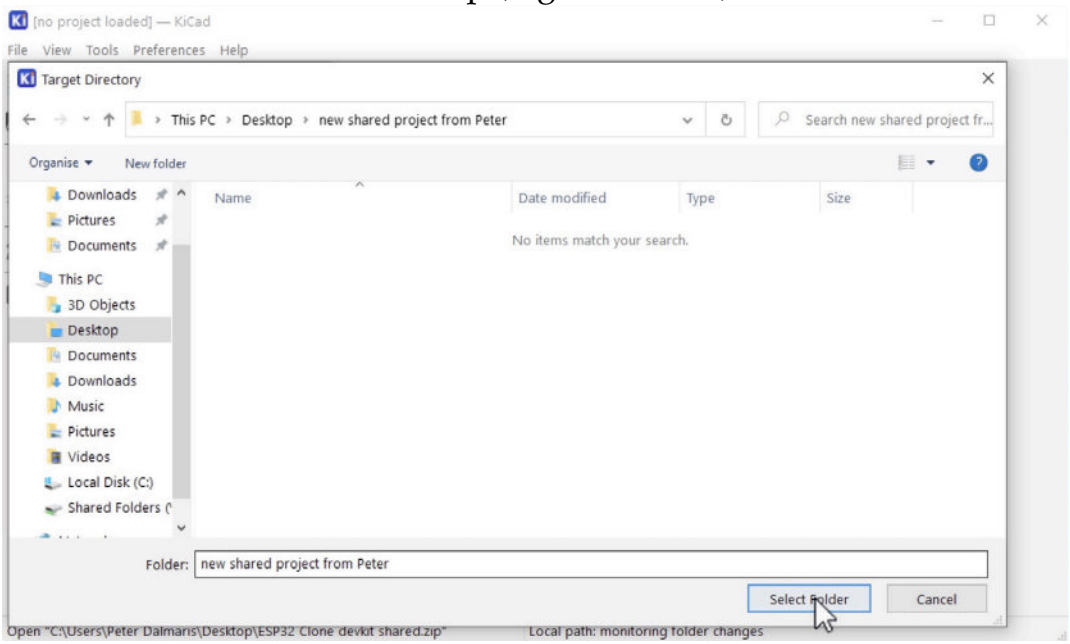


Figure 13.32.5: Select a destination for the unarchived project.

Navigate to the new folder, and look inside. You will see the contents of the project that I archived on my MacOS computer (Figure 13.32.6):

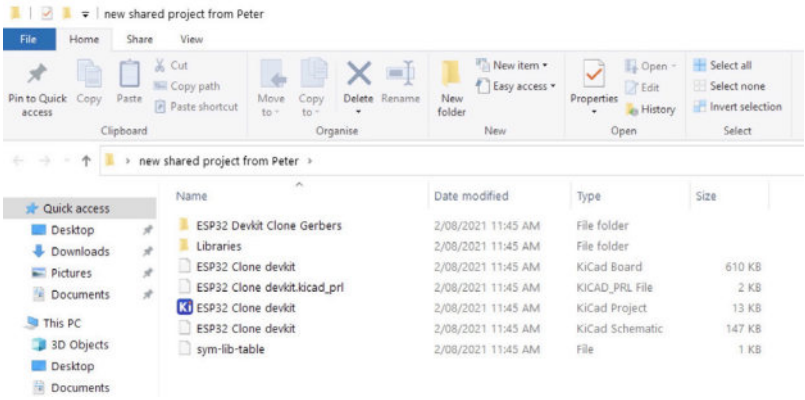


Figure 13.32.6: The contents of the unarchived project.

Still working in Windows 10, go to KiCad and open the project. You will see the project settings and the editor contents as they were when the archive was created (Figure 13.32.7).

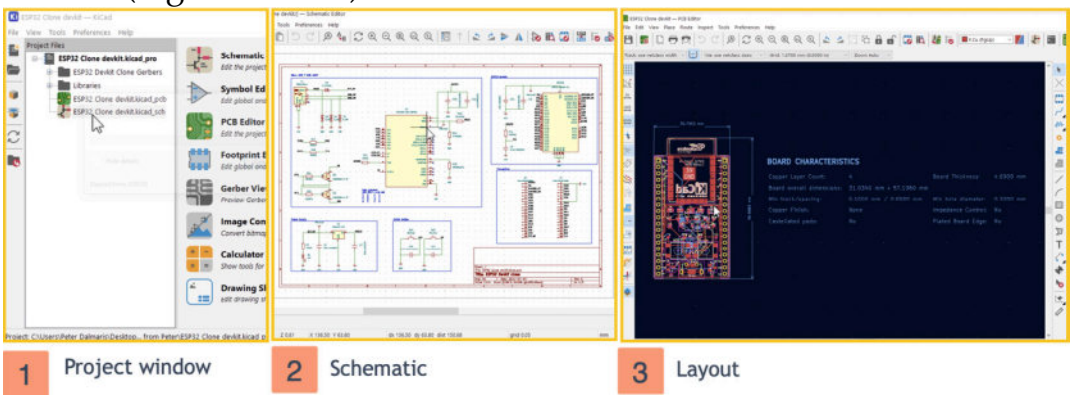


Figure 13.32.7: The project, unarchived in Windows 10.

KiCad's Archive Project tool provides a simple way to share projects with other people or safely store them.

33. Buses

In cases where you have several wires that belong to the same functional group, buses can help reduce clutter and risk of errors.

Let's look at an example. Say you want to connect a Z80 microprocessor to a memory chip. This connection requires a lot of wires for the address and data. In Eeschema, you would start with an arrangement like the one in Figure 13.33.1. Our objective is to connect pins A0 to A15 from the CPU to the pins with the same name on the RAM module and do the same for the data pins.

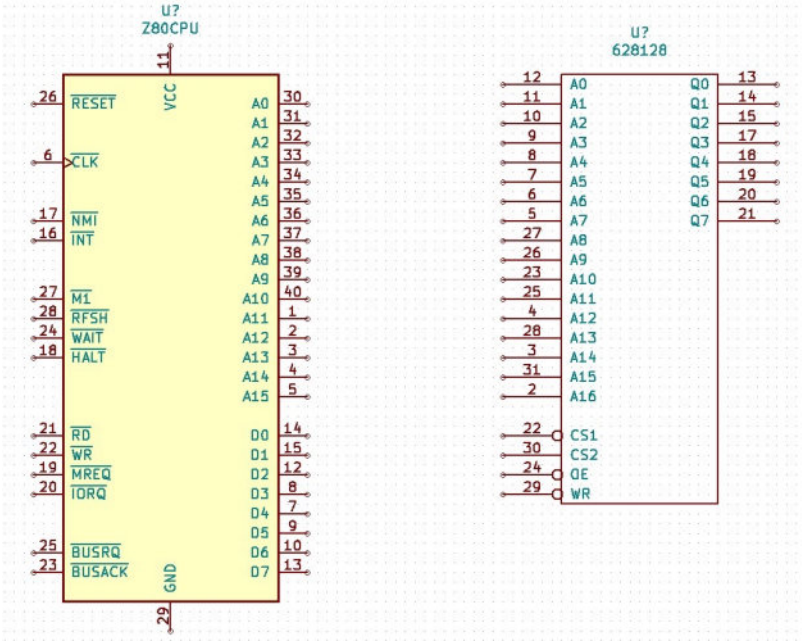


Figure 13.33.1: Let's connect the CPU to the RAM using buses.

You can use normal wires, and the schematic would be correct, although very (visually) busy. Instead, we will use the bus option, and create two busses, one for the address pins and one for the data pins. To do that, you will use two tools from the side menu, the Bus tool ("1", in the figure below), and the Bus Entry tool("2").

Figure 13.33.2: The Bus and Bus Entry tools.

Use the Bus tool first to draw a bus line in between the CPU and the RAM. In Figure 13.33.3 I have drawn a bus with this particular shape. It doesn't have to look like this; the exact shape is up to you.

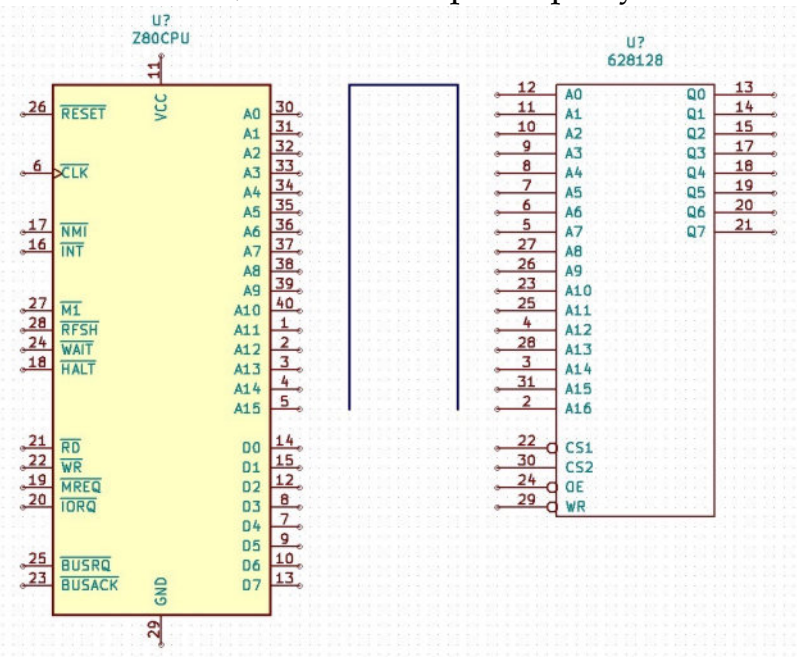


Figure 13.33.3: A bus is depicted by a thick blue line.

Next, use the Bus entry tool to create entry points from the pins to the bus. Attach the bus entries to the bus line (Figure 33.4), and then use a normal wire to connect the bus entry lines to the CPU pins if the bus entry lines themselves are not long enough (Figure 13.33.5). You can also move the bus entry line if needed to align them better.

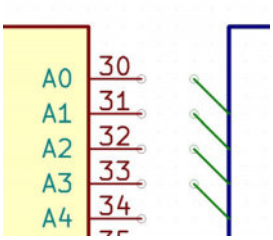


Figure 13.33.4: Bus entry lines for pins A0 to A3.

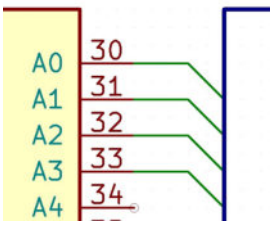


Figure 13.33.5: Using normal wires to connect the pins to the bus entries.

Do the same thing on the RAM module side. Your schematic will look like the example in Figure 13.33.6. In this example, I have grounded A16 of the

RAM module since the CPU address bus can only drive 16-bit addresses (not 17 bits, as the RAM module is capable of).

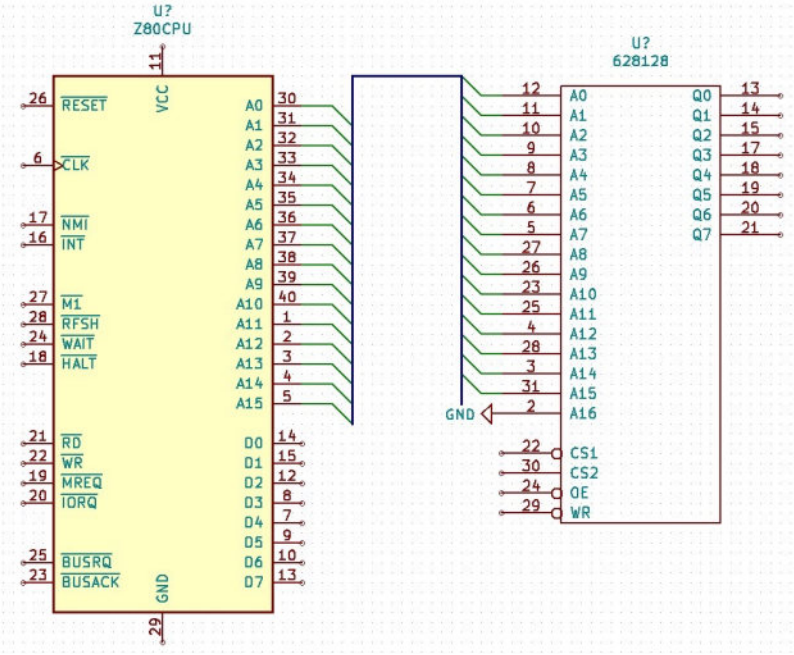


Figure 13.33.6: The Address bus, unlabelled.

To complete the bus wiring, you need to label each bus entry point. The labels allow Eeschema to know which pins within the bus are electrically connected. To quickly set create the labels, use the net label button, label the first net with 'A1', and then continue by using the Insert key to insert the rest of the labels automatically. Each time you press Insert, the next label, properly numbered, will appear. When you complete all labels on the CPU side, up to A15, manually create the next label as 'A0' and continue using the Insert key. The final result looks like the example in Figure 13.33.7.

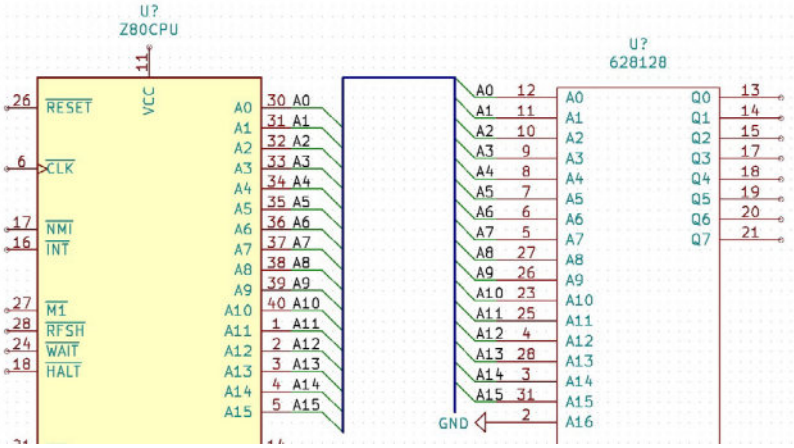


Figure 13.33.7: The address bus is now labeled.

Repeat the same process to wire the data bus. The result is in Figure 13.33.8.

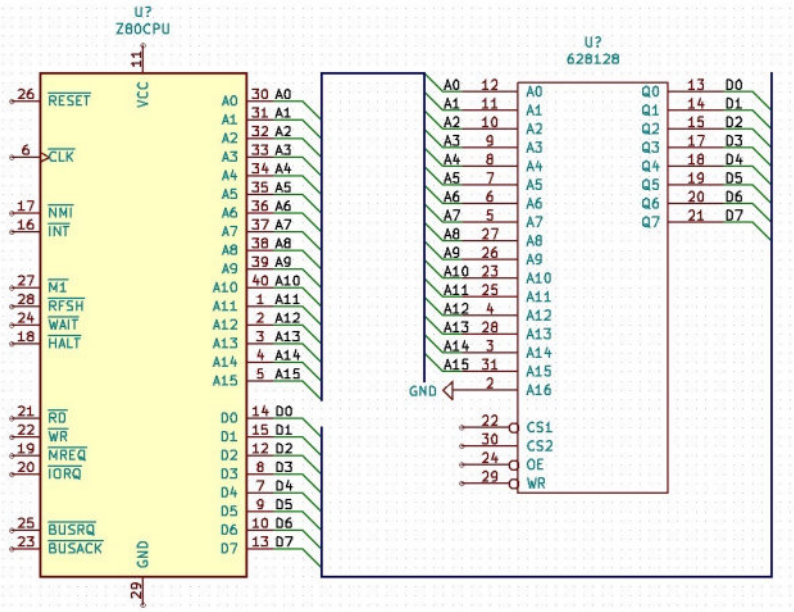


Figure 13.33.8: The data and address buses are fully wired.

As you can see, the resulting schematic is clear and easy to read. The two buses contain invisible but real electrical connections. When you import the netlist file into Pcbnew, these connections will be converted into individual tracks.

34. 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 according to various parameters, like the current you wish to convey through the trace, its total length, and the maximum temperature rise when that current is 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 much larger width than the standard width of signal traces.

To use the calculator, open the KiCad launcher window and click on the calculator icon(Figure 13.34.1).

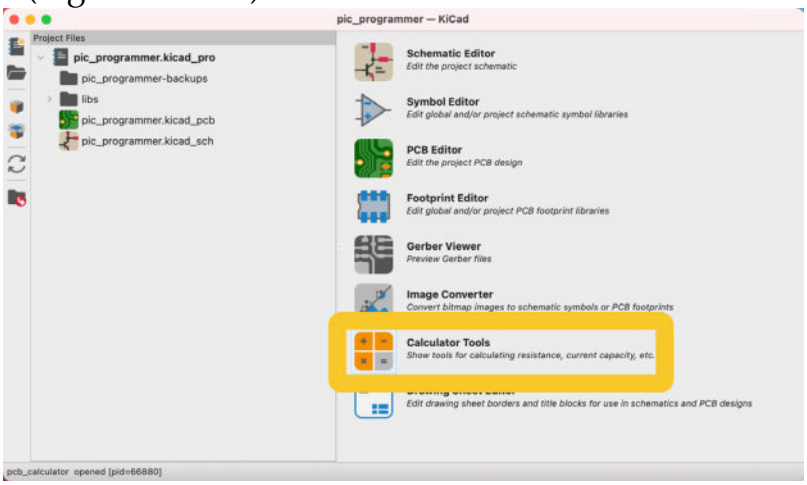


Figure 13.34.1: The calculator is available via the KiCad project window.

The calculator app 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 13.34.2 are reasonable. I have only altered the conductor length value to 20mm 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.

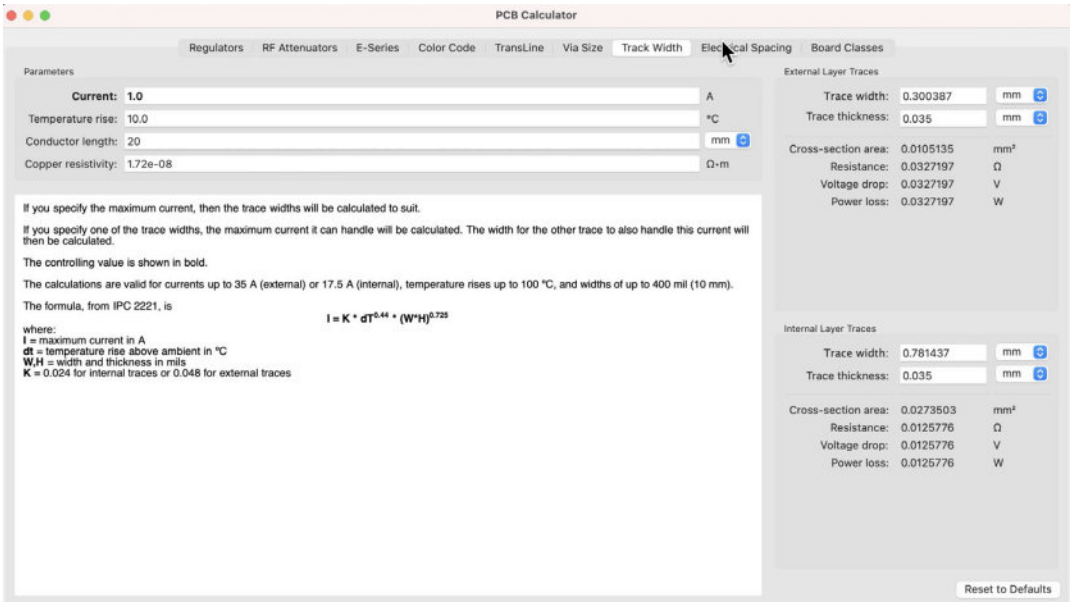


Figure 13.34.2:The Track Width calculator.

At the top right corner of the calculator, there is a field to 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 13.34.2, the suggested width is 0.30 mm.

35. Design a custom schematic sheet

You can create a custom version of the sheet layout. You can use your custom sheet to replace the default one in Eeschema. A popular reason for doing that is to include a logo on every project page.

In this recipe, you will learn how to create a simple custom sheet layout. The only difference between the default layout and the one we are about to create is that the custom one contains a logo graphic. Once you have your custom sheet layout file, you can use it in all your schematics.

To create and edit sheet layouts, KiCad provides a helper application called "Drawing Sheet Editor." To open this editor, go back to the KiCad project manager, and click on the Drawing Sheet Editor button at the bottom of the right pane (Figure 13.35.1).

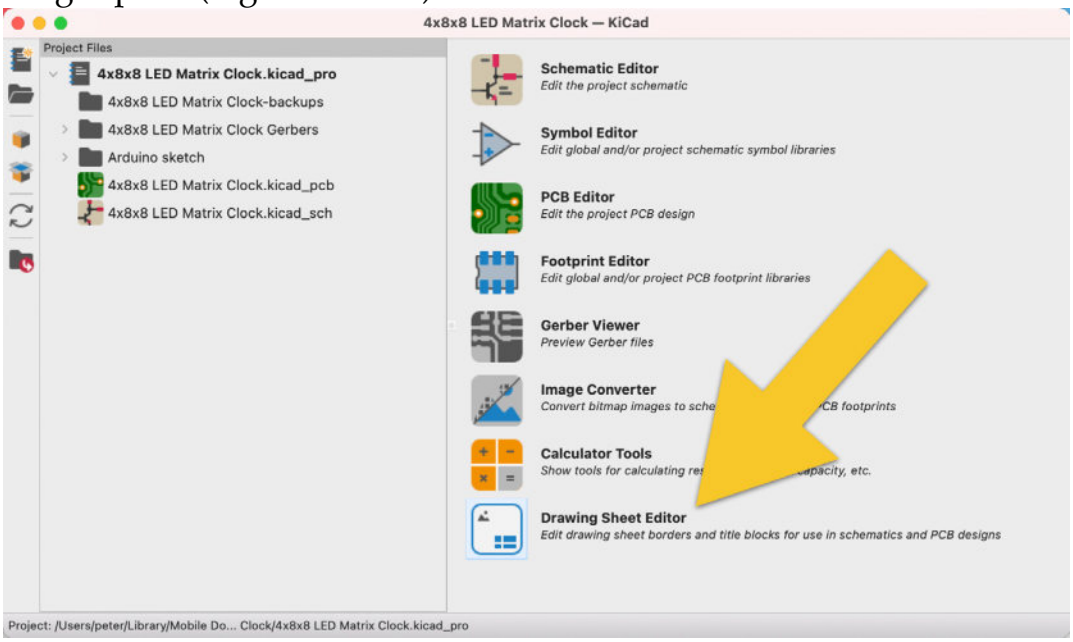


Figure 13.35.1: Starting the Drawing Sheet Editor.

The editor will start, showing the default layout. You can choose to work and edit the default layout, create a new one, or load and edit an existing layout.

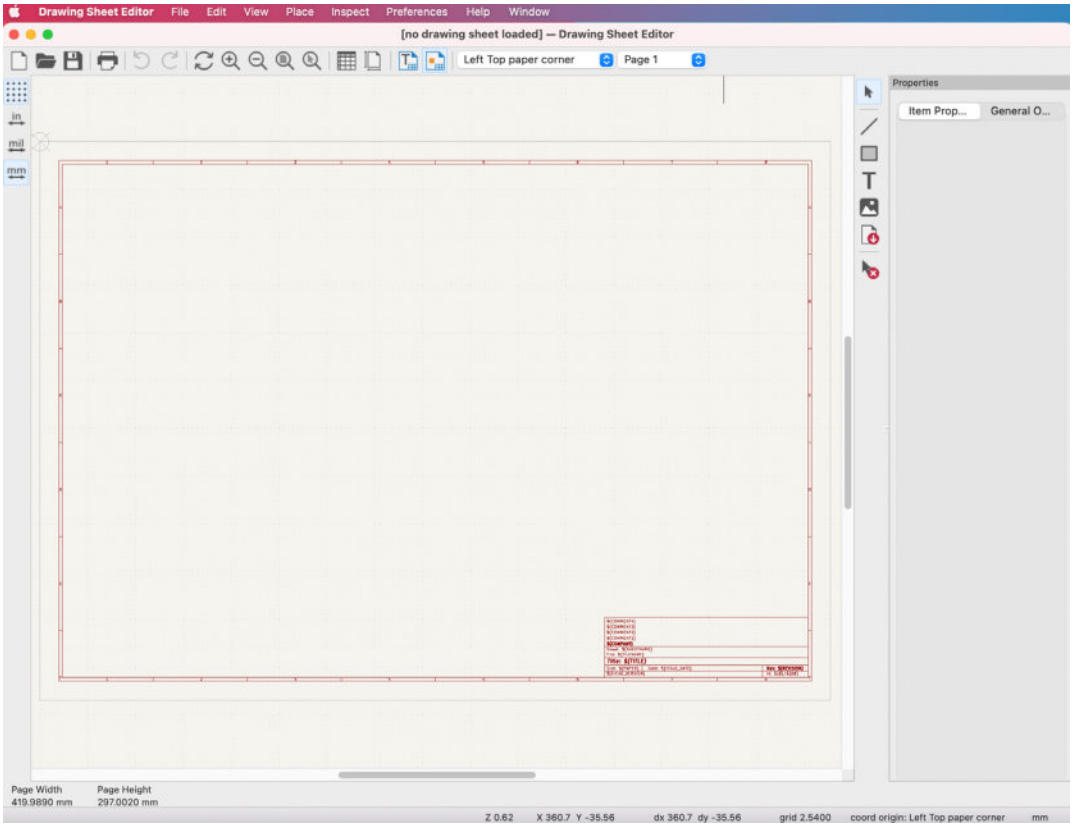


Figure 13.35.2: The Drawing Sheet Editor showing the default sheet layout.

A blank layout is empty of any borderlines and text placeholders, giving you maximum freedom to design. You can create a blank layout by choosing "New" from the File menu or clicking on the new document button in the top toolbar. You can also open an existing layout so that you can modify it to your requirements—KiCad ships with several sheets that you can use as-is in your schematics or modify them. To open a sheet layout, click on the Open button from the top toolbar and browse for the sheet you want to work on.

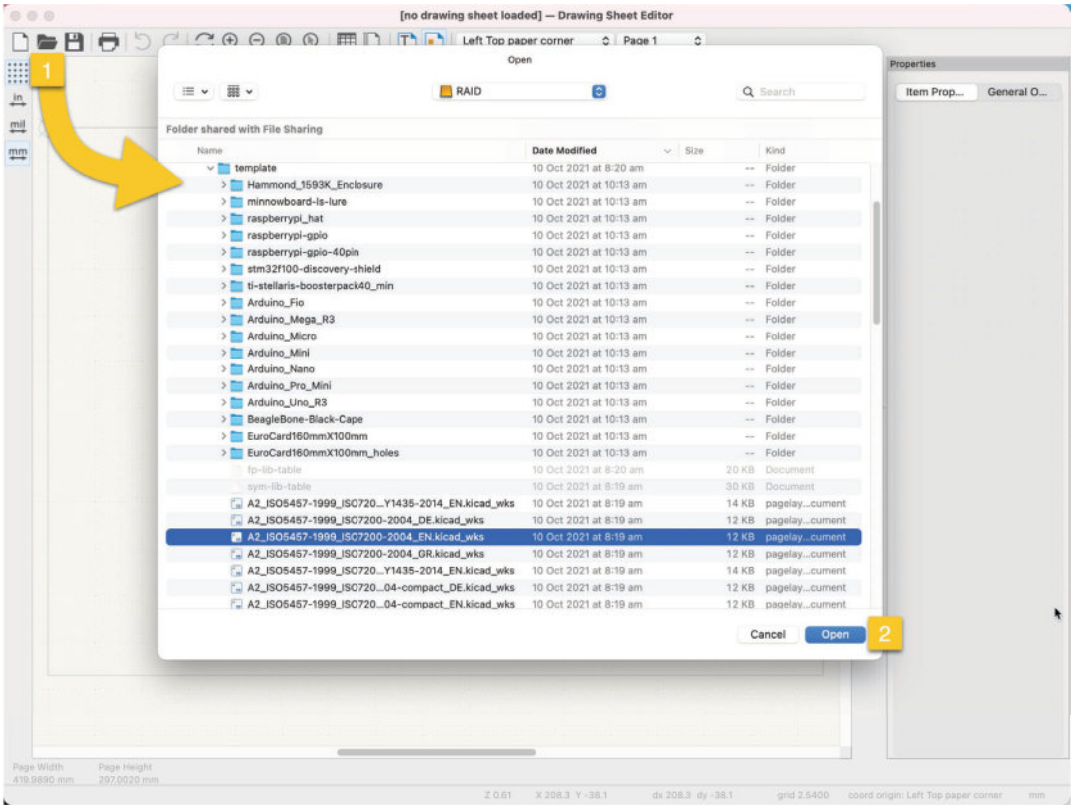


Figure 13.35.3: Open an existing sheet layout to edit.

If you don't remember where to find the schematic sheet layout files in your instance of KiCad, go back to the KiCad project window and select "Configure Paths" from the Preferences menu. In the Environment Variables table, look for the path stored in the KICAD6_TEMPLATE_DIR variable.

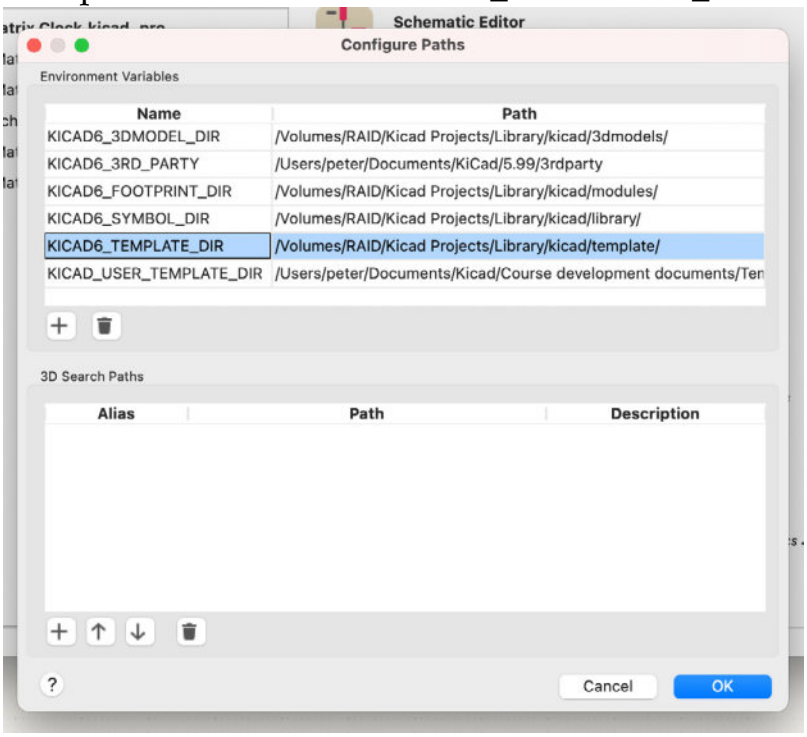


Figure 13.35.4: The path to my KiCad schematic editor sheet layout files.

For this example, I will be using the sheet layout with the filename " A2_ISO5457-1999_ISO7200-2004_EN.kicad_wks". You can see the template below:

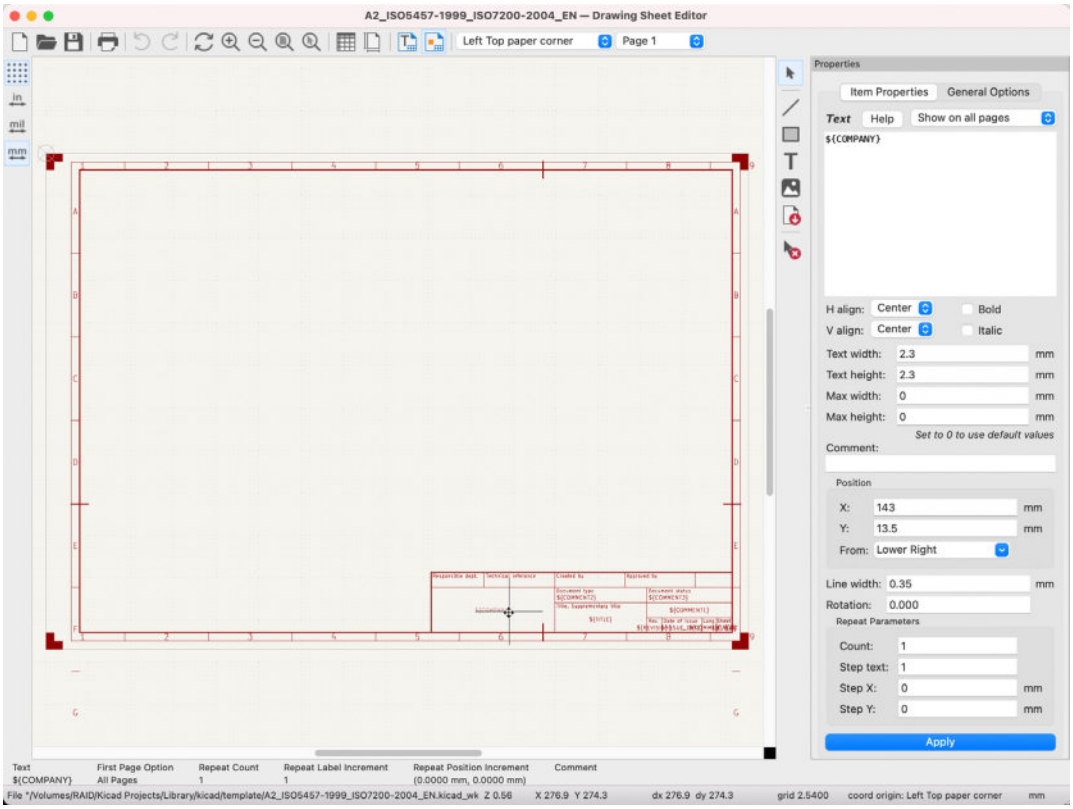


Figure 13.35.5: I will modify this sheet layout.

The editor provides drawing tools in the right toolbar and a properties pane on the right side. You can draw lines, boxes and add text. You can also insert images.

Each element has properties that you can edit in the properties pane. For example, below, I have used the line tool to draw a single line and then changed the line width to 5 mm in the properties pane:

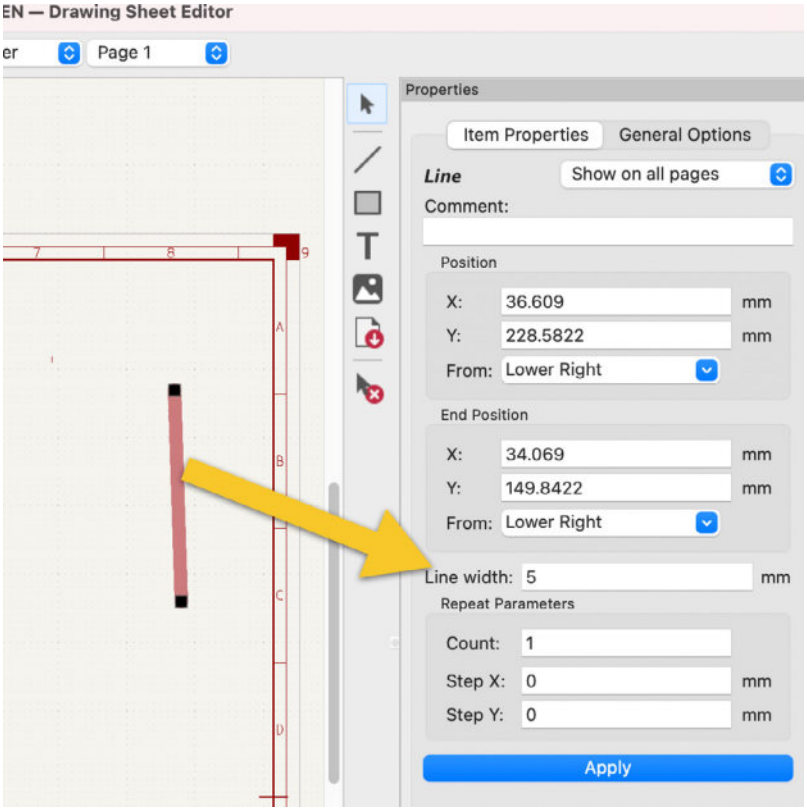


Figure 13.35.6: Every element has properties.

The most interesting and useful part of the sheet is the information box that contains the text placeholders. In the example below, I have selected the placeholder "${COMPANY}" to display its properties in the right pane (see below).

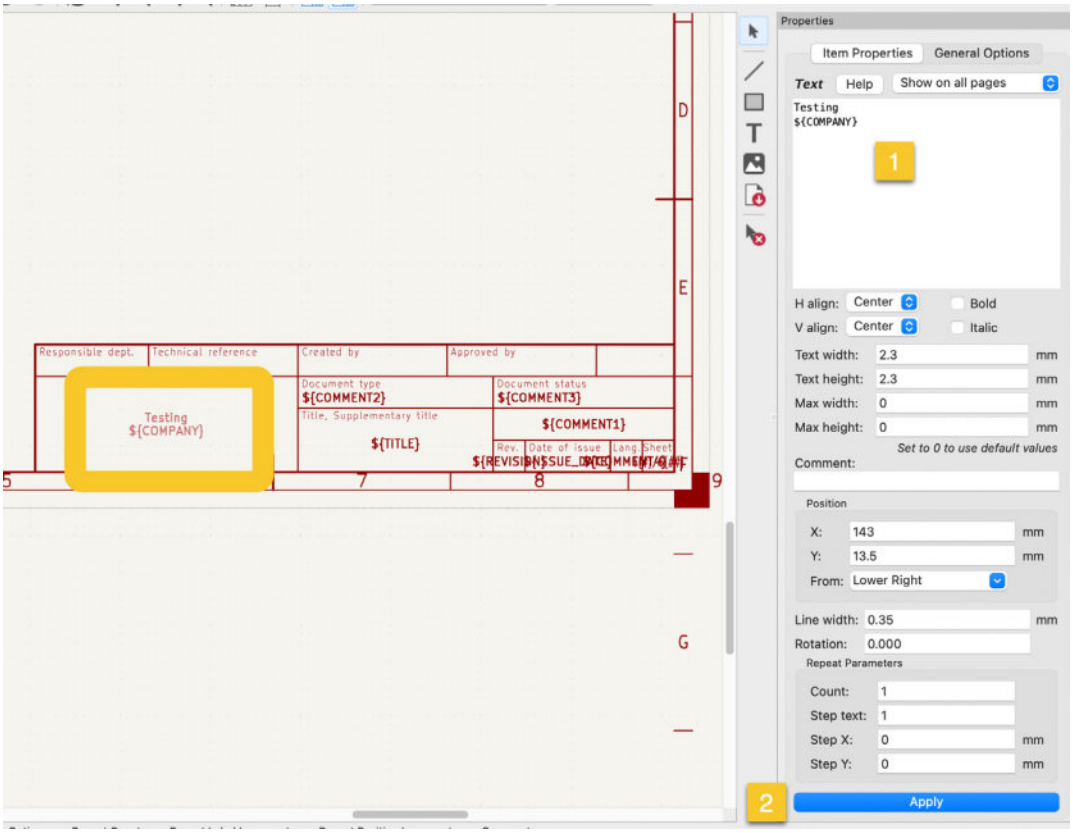


Figure 13.35.7: Editing the “COMPANY” placeholder.

In the Properties pane, I have added the word "Testing" in the text box, followed by the "${COMPANY}" variable in the following line ("1" in the figure above). Click "Apply" ("2") to finish editing this text placeholder, and then save the sheet editor.

Return to Eeschema, and bring up the Page Settings window (File —> Page Settings). Use the folder button to navigate and select the sheet layout you edited in the previous step in the File field. Notice that the "Company" placeholder has a field where you can type text. In other words, the placeholders that you define in the sheet editor become text fields in the Page Settings window of Eeschema. In the example below, I have typed some text before clicking "OK."

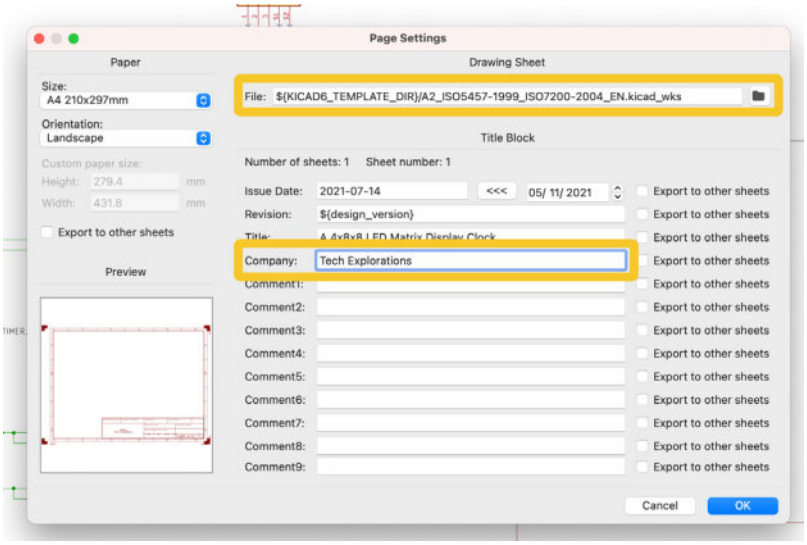


Figure 13.35.8: Setting a value for the Company text field.

In Eeschema, the information box shows the text you provided in the Page Settings text fields using the edited sheet layout. You can see the result of this work below:

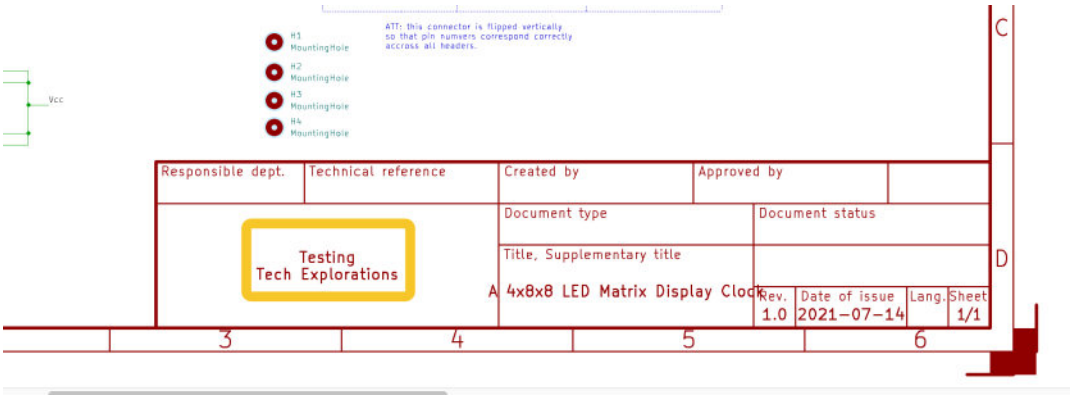


Figure 13.35.9: Using the customised sheet layout in this schematic.

You can also use formatting symbols to automatically display information such as the KiCad version used, current date, sheet number, etc. You can see the available format symbols in the table below.

Symbol	Description
%K	KiCad version
%Z	Paper format name
%Y	Company name
%D	Date
%R	Revision
%S	Sheet number
%N	Number of sheets
%Cx	Comment (x = 0 to 9 to identify the comment)

%F	Filename
%P	Sheet path
%T	Title

Table 13.35.1: A list of format symbols available in KiCad.

Another modification you can make is to add an image, such as a logo, to decorate the information box. Below, I have used the image tool from the right toolbar to add a PNG image in the Company text box:

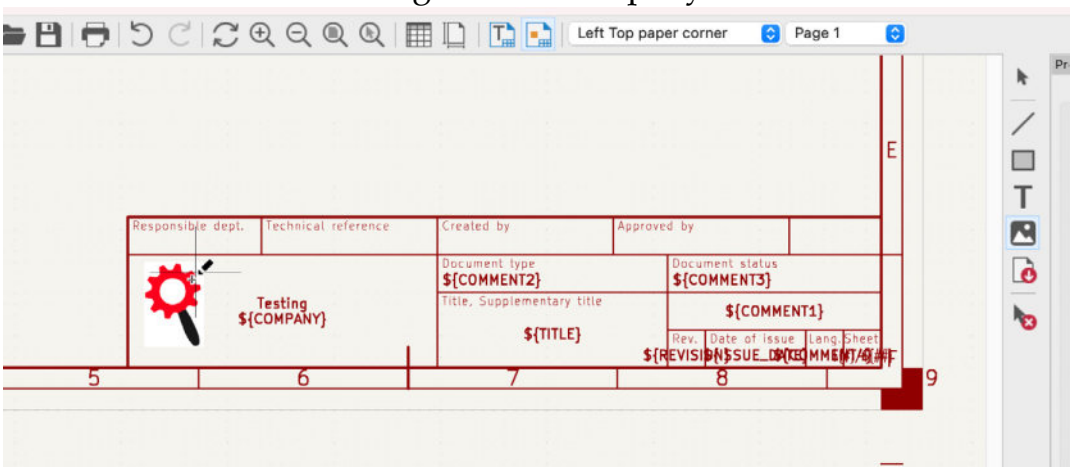


Figure 13.35.10: Added an image to the project information box.

Save the sheet layout and return to Eeschema. The schematic information box will now display the logo:

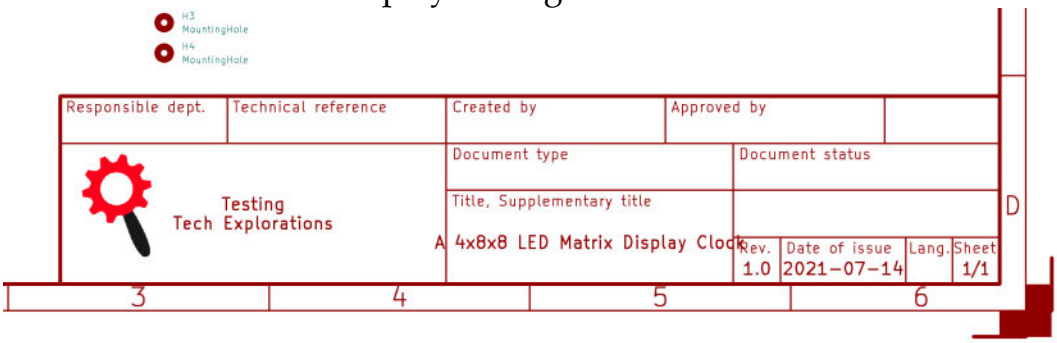


Figure 13.35.11: The image appears in the schematic information box.

You can use the same sheet layout in any of your projects simply by loading the sheet file to the project through the 'File' field in Page Settings.